Imperial College London

Imperial College London

Department of Mathematics

Financial Bubble Prediction with Neural Networks

Author: Yuchen Yan (CID: 02292404)

A thesis submitted for the degree of

MSc in Mathematics and Finance, 2022-2023

Declaration

The work contained in this thesis is my own work unless otherwise stated.

Signature: Yuchen Yan
Date: 2023.09.04

Acknowledgements

I would like to extend my sincere gratitude to my supervisor Lukas Gonon for his unwavering support, guidance, and mentorship. Special thanks to my colleagues and friends in the MSc Mathematics and Finance project, especially Flora Yu and Allen Qian, for the stimulating discussions, late-night brainstorming sessions, and for creating a collaborative and supportive environment. I would like to acknowledge Imperial College London for providing the necessary resources and a conducive environment for this whole year. Lastly, to all those who have been a part of this journey, directly or indirectly, I express my heartfelt gratitude.

Abstract

This thesis delves into employing deep learning techniques to predict the date of the financial bubble burst based on observed European Calls. We introduce the martingale approach [1] to define financial bubbles to provide a theoretical foundation mathematically. The work explores the bubble detection model from Biagini et al. [2] to judge if the asset is in a bubble state and the SVI volatility surface model from Gatheral and Jacquier [3] to extract information from European Calls. Upon detecting that an asset is in a bubble, we fit the asset's Calls to the SVI model and then use the parameters of SVI to predict the occurrence time of the bubble burst. We test our methodology's accuracy in numerical experiments for three assets. By predicting the date of a bubble burst, financial institutions, investors, and governments can assess risks associated with financial bubbles and take appropriate measures to mitigate potential losses. While there are limitations in that we can't get enough option data to train our model as students, this study offers a promising tool for all practitioners to anticipate potential bubble bursts and paves the way for future research.

Contents

List of Figures

List of Tables

Introduction

The financial market is complex and dynamic. A phenomenon of significant interest and study is the economic bubble. Typically, a financial or asset price bubble is characterized by a swift escalation in asset prices, significantly exceeding their intrinsic value, followed by a steep decline, commonly called the bubble burst. Such bubbles often precede financial crises, with historic instances like the Dot-com bubble serving as cautionary tales. The late 1990s witnessed this stock market bubble, fueled by a dramatic increase in internet penetration, an influx of venture capital, and soaring valuations for nascent web startups. However, the subsequent collapse was equally dramatic, leading to widespread economic downturns.

Numerous studies have delved into financial bubbles, offering unique insights and perspectives. Predicting the timing of the bubble burst remains an elusive challenge. A mathematical representation of bubbles is essential for their prediction. There are two famous mathematical definitions for bubbles: one is the charge approach introduced by Gilles and LeRoy [1, Section 3, page 328], and the other is the local martingale approach presented by Cox and Hobson [4, Definition 2.1, pages 480]. This article mainly focuses on the local martingale approach. The discounted market price of an asset represents a bubble when it is given by a strict local martingale under risk-neutral measure but not a martingale.

In this thesis, the underlying process follows local martingale dynamics, and true martingale dynamics are generated by the SABR model, introduced by Sin [5, Section 2, page 2]. By fine-tuning the parameters of the SABR model, we generate a balanced dataset: half of it is based on martingale training data, and the other half on strict local martingale training data. Then, we take them as the input of the bubble detection model from Biagini et al. [2, Section 3, page 17], a model-free deep learning approach to detect asset price bubbles. The model can detect if the input option prices come from a strict local martingale. In essence, it determines the presence of a financial bubble. We have reproduced part of the practical result from Biagini et al. and successfully detected some bubbles on Nvidia, Apple, and Tesla.

This thesis contributes works on bubble prediction. More specifically, after we use the model from Biagini et al. to detect bubbles from historic Call option prices, we define the day with the highest stock price as the bubble burst and label each day before the burst by the number of days between the day and the upcoming burst. Since we cannot generate samples with bubble detection for each day, we must use Calls from different days to jointly train the prediction model. However, the number of Calls contained in different days is different. To achieve consistent data dimensions and meanings within each sample, we propose a normalization strategy using the Surface Stochastic Volatility-inspired model introduced by Gatheral and Jacquier [3, Section 4, page 10]. By fitting an SVI model to the volatility surface for each trading day, we can transform the daily, unstructured option data into a volatility surface without significant loss of information. This methodology facilitates the generation of standardized input data. Then, we fed these data to a Fully Connected Neural Network and generated a single float number as the prediction of the number of days to the upcoming bubble burst.

The organization of the article is as follows. Chapter 1 outlines the theoretical backing for the bubble detection and prediction framework. Initially, from defining bubbles in a local martingale approach, we further link bubbles in the underlying assets into corresponding options. We also propose several volatility models, especially the SABR model in this Chapter, better capturing the market volatility and option prices. Chapter 2 delves into our feedforward neural networks (FNNs) in detail. After introducing every component in the FNN architecture, we construct the bubble detection model that successfully showcases the probability of being in a bubble. In Chapter 3, we explore the Stochastic Volatility Inspired surface. The estimated parameters in this volatility model serve as the inputs of our bubble prediction model, described in Chapter 4.

Chapter 1

Theoretical Background of Financial bubble Detection

Financial bubbles refer to situations where the price of an asset rises significantly beyond its actual worth, leading to an imbalance in supply and demand. This state creates an illusion of prosperity since the perceived value is much higher than the real one. The effects of this overvaluation ripple throughout the economy due to financial leverage, tying up numerous industries.

If a bubble bursts, the fallout can be catastrophic. Markets can plunge into chaos, collapsing the financial system. Many assets become worthless, erasing the wealth of countless investors. However, not all aspects of financial bubbles are negative. They can help concentrate capital, spurring economic progress and market stimulation. This is because anticipating ever-increasing asset values can motivate people to invest their savings, thereby boosting production. The adept use of financial instruments can aid in efficiently reallocating resources and amplifying asset turnover. Therefore, moderation is crucial.

Given their significance in contemporary finance, financial bubbles play a pivotal role in shaping market dynamics. Consequently, this thesis seeks to identify potential future manifestations of such bubbles and determine the anticipated temporal proximity of these occurrences.

This chapter delineates the mathematical framework associated with financial bubble concepts, laying a robust foundation for recognizing their presence within the options market. By harnessing the principles of martingale theory, we aim to elucidate the intricate dynamics of bubbles within asset markets. Our discourse then expands to encompass European vanilla options, noting an intriguing observation: the magnitude of bubbles within European Call options mirrors that of their underlying assets. Furthermore, the SABR volatility model is presented and is instrumental in creating the primary dataset for the inaugural training phase of our bubble detection neural network.

1.1 Bubbles and martingale theory

In the scholarly exploration of financial bubbles, two primary approaches have historically emerged: one is the charge approach as detailed in [1, Section 3, page 328], and the other is the local martingale approach [4, Definition 2.1, pages 480]. Notably, these approaches are proved to be analogous [6, Section 8, page 32].

We first introduce the charge approach, a more consistent definition with our intuition. Under this perspective, financial bubbles are construed as discrepancies between the intrinsic values of assets and their respective market assessments. More specifically, the charge approach aligns bubbles with two distinct asset price categories: fundamental and market prices.

The term 'fundamental price' denotes an asset's inherent worth or true valuation derived from a comprehensive analysis of intrinsic factors. When considering stocks, these factors encapsulate elements like dividends, future growth potential, the prevailing economic landscape, and the particular standing of the company in question. Mathematically, this is conceived as a conditional expectation of future cash flows [6, Definition 3.1, page 13]. In contrast, the second type of price, the 'market price,' refers to the actualized trading value of an asset, contingent upon the dynamics of market supply and demand.

The essence of the fundamental value is the risk-neural expectation of the asset's future integrated payoffs, which largely depends on the selected pricing model. However, in this thesis, our objective is deploying neural networks for bubble detection. Without specifying a presupposed pricing model, our inclination is towards harnessing the prowess of the local martingale approach.

Let us first recall the definition of martingales and local martingales.

Definition 1.1.1 (Martingales)**.** A stochastic process $X = (X_n)_{n=0}^{\infty}$ on a filtered probability space $(\Omega, \mathcal{F}, \{\mathcal{F}_n\}_{n=0}^{\infty}, P)$ is said to be a martingale if

1. X is $\{\mathcal{F}_n\}$-adapted.
2. $E[|X_n|] < \infty$, for every n.
3. $E[X_{n+1}|\mathcal{F}_n] = X_n$ a.s. for every n.

Definition 1.1.2 (Local martingales)**.** A stochastic process $X = (X_t)_{t\geq 0}$ is a continuous local martingale on $\left(\Omega, \mathcal{F}, \{\mathcal{F}_t\}_{t\geq 0}, P\right)$ if there exists a sequence of $\{\mathcal{F}_t\}$-stopping times $0 \leq T_1 \leq T_2 \leq ... \leq T_{n-1} \leq T_n \uparrow \infty$ a.s. as $n \to \infty$ such that $X^{T_n} = (X_{T_n \wedge t})_{t\geq 0}$ is a martingale for every n.

Strict local martingales refers to the local martingales that are not martingales. Now, we introduce the definition of financial bubbles under the martingale approach.

Definition 1.1.3 (Financial bubbles)**.** The price process S has a bubble if S is a strict local martingale under risk-neutral measure $\mathbb{Q}$.

In the classical mathematical finance theory, the pricing models only consider the finite horizon and the arbitrage-free condition. Thus, with this no-arbitrage condition, an equivalent martingale measure $\mathbb{Q}$ exists, under which all assets are priced. The asset dynamics display a typical martingale behavior under this risk-neutral measure $\mathbb{Q}$. Since there is no difference between the true fundamental price and the market price, all investors trade relatively fairly, and the financial bubbles may not exist. Many pricing models, including the classic Black-Scholes model, provide arbitrage-free asset pricing without considering the possibility of observing bubbles.

However, financial bubbles, like the Internet bubble, appeared in the real market. These observations are direct violations of the classical theory, which leads to a distinct gap between the fundamental price and its market price. Many factors, such as news, public sentiments, and speculations, can drive the market price from its fundamental price. The asset is overvalued when the market price is far above the fundamental price. After a long-period overvaluation, it is natural for the market to process new information and eliminate the arbitrage space. This appears as a sharp drop in price, which is usually captured after the occurrence of the financial bubble.

To explain the existence of bubbles, the modern mathematical finance theory considers the infinite horizon and replaces the arbitrage-free assumption for the market with the following assumption.

Assumption 1.1.4 (NFLVR)**.** The market satisfies No-Free-Lunch-with-Vanishing-Risk (NFLVR) condition.

Assumption 1.1.4, which has the mathematical formulation in [7, Definition 2.8, page 473], implies that there is no such a sequence of self-financing portfolios, which converges to an arbitrage strategy. Theoretically, when a market meets the NFLVR conditions, it is also Arbitrage-Free. NFLVR is a more strict condition. Suppose an arbitrage strategy exists in the market. In that case, it obviously does not meet the NFLVR conditions because there is a strategy that can consistently generate positive expected returns without taking risks. NFLVR excludes broader non-standard strategies or trading practices. When we assume the NFLVR condition, the asset dynamics display semi-martingale behavior under Q [7, Theorem 7.2, page 504]. Since the price of the asset is assumed to be non-negative, it is bounded below by zero, and any semi-martingale bounded below is a local martingale, which is proven by Ansel and Stricker [8, Proposition 3.3 and Corollary 3.5, pages 307-309]. Indeed, the price process of the asset could be just a martingale or a strict local martingale. In other words, this asset can have bubbles under the local martingale approach. We thus prove the existence of bubbles in the modern model.

To summarise, the classical theory studies the finite horizon under arbitrage-free conditions, while the modern theory studies the infinite horizon under NFLVR. Although NFLVR is a stricter assumption than arbitrage-free, bubbles can exist in the modern theory and cannot be allowed in the classic theory. This is because modern theory studies the infinite horizon, which is a broader area. So, even with stricter restrictions, bubbles are still allowed to exist.

1.2 Bubbles in options

This section discusses whether bubbles can occur in options with risky assets being the underlying. We start with the definition of the size of bubbles. We then develop step-by-step that European Call options may have bubbles, and if bubbles exist, the size of the bubbles equals to the size of the bubbles of their underlying assets. By contrast, European Put options cannot have bubbles.

Definition 1.2.1 (Size of bubbles)**.** Denote S as the market price and S^* as the fundamental price of the asset. We define the size of the bubble λ as:

$$\lambda = S - S^*$$

Let us then define equivalent local martingale measure and the First Fundamental Theorem [7, Corollary 1.2, page 479].

Definition 1.2.2 (Equivalent local martingale measure (ELMM))**.** Let $(\Omega, \mathcal{F}, \mathbb{F}, \mathbb{P})$ be a filtered complete probability space. An equivalent local martingale measure ($ELMM$) for a wealth process W is a probability measure $\mathbb{Q}$ equivalent to P on $\mathcal{F}_T$ such that W is a local $\mathbb{Q}$-martingale. We denote by $\mathcal{M}_{loc}(W)$ the set of all $ELMMs$ for W.

Theorem 1.2.3 (The Fundamental Theorem)**.** *A market satisfies NFLVR if and only if there exist an ELMM.*

Since modern theory assumes the market is NFLVR, the ELMM always exists in our studies.

We define the payoff of an option H on the underlying asset S at time t as Λ_t^H. To define the fundamental price of options, we need to use the so Called valuation measure $\mathbb{Q}^*$. The formula of valuation measure $\mathbb{Q}^*$ is introduced by Jarrow et al. [6, Definition 3.2, page 15]. This valuation measure is determined by $ELMMs$. $\left(\mathbb{Q}^i\right)_{i\geq 0} \in \mathcal{M}_{loc}(W)$ and the fundamental price of the option is defined as $E_{\mathbb{Q}^*}\left[H_T(S)|\,\mathcal{F}_t\right]$, which represents the conditional expectation of the derivative's time T payoff under the valuation measure $\mathbb{Q}_t^*$. Thus, the size of bubbles in option is:

$$\delta_t = \Lambda_t^H - E_{\mathbb{Q}^*}\left[H_T(S)\,|\mathcal{F}_t\right]$$

which is the difference between the market price and fundamental price of the option.

We are examining three common derivative securities: a forward contract, a European Put option, and a European Call option. They are all tied to the same risky asset S. Their distinct features are primarily characterized by their payoffs at maturity.

- (Forward Contract) With a strike price of K and maturing at T, its payoff is $[S_t - K]$. Its market price at time t is $V_t^f(K)$.
- (European Call Option) With a strike price K and maturity at T, its payoff is the positive part of $[S_t - K]$, represented as $[S_t - K]_+$. Its market price at time t is given by $C_t(K)$
- (European Put Option) With a strike price K and maturity at T, its payoff is the positive part of $[K - S_t]$, represented as $[K - S_t]_+$. Its market price at time t is given by $P_t(K)$

Also, let be the $V_t^f(K)^*$, $C_t(K)^*$ and $P_t(K)^*$ be the fundamental prices of the forward contract, Call option and a Put option, respectively. The following theorems directly imply these three derivatives introduced by Jarrow et al. [6, Section 6.1, page 12].

Theorem 1.2.4 (Put-Call parity for fundamental prices)**.**

$$C_t(K)^* - P_t(K)^* = V_t^f(K)^*.$$

Proof. At expiry T, for any $K > 0$,

$$(S_T - K)^+ - (K - S_T)^+ = S_T - K.$$

Since the fundamental price for an option with payoff H on the asset S at time t with valuation measure $\mathbb{Q}^*$ is $E_{Q^*}\left[H_T(S)\,|\mathcal{F}_t\right]$,

$$\begin{aligned} C_t^*(K) - P_t^*(K) &= E_{Q^*}\left[(S_T - K)^+\,|\mathcal{F}_t\right] - E_{Q^*}\left[(K - S_T)^+\,|\mathcal{F}_t\right] \\ &= E_{Q^*}\left[S_T - K|\mathcal{F}_t\right] \\ &= V_t^f(K)^* \end{aligned}$$

□

To generalise the Put-Call parity for market prices, we initiate our discussion by defining the concepts of 'dominance' and the 'no dominance' assumption. Notably, the 'no dominance' principle, coupled with the NFLVR previously introduced, forms the bedrock of modern market theory.

The intricate concept of dominance is meticulously delineated by Jarrow et al. in [6, Definition 2.5, page 11]. A comprehensive exposition of this idea demands an intricate framework of settings and formulae. For the sake of brevity, we summarise the essence of dominance in two primary facets to describe the circumstances under which one asset is considered 'dominated' by another.

Consider two financial assets, represented as ϕ_1 and ϕ_2. The first condition to be satisfied revolves around the identification of a 'stopping time σ such that σ is less than certain fixed time point T. For options, time T is conventionally regarded as the maturity. For all time $u > \sigma$, the relationship

$$G_{\sigma,u}(\phi_2) \geq G_{\sigma,u}(\phi_1), \tag{1.2.1}$$

is satisfied almost surely. Here, $G_{\sigma,u}$ is the net gain introduced by Jarrow et al. [6, Formula 18, page 11]. We can simply regard it as the profit of purchasing at σ and selling at u. The formula 1.2.1 implies that after σ, the profit of the ϕ_1 is always greater than or equal to the profit of ϕ_2. The secondary condition to be met is represented as:

$$E\left[\mathbb{1}_{\{G_{\sigma,u}(\phi_2)>G_{\sigma,u}(\phi_1)\}}|\mathcal{F}_\sigma\right] > 0, \tag{1.2.2}$$

which holds true almost surely. When both these conditions are concurrently satisfied, it can be conclusively stated that at time σ, ϕ_1 is dominated by ϕ_2. Simply put, this concept of dominance tells us that under certain circumstances, one asset may be better than another in terms of profit. Following this, we shall elucidate the 'no dominance' assumption.

Assumption 1.2.5 (No dominance)**.** Let the market be represented by a function $\Lambda_t : \Phi \to \mathbb{R}_+$ such that there are no dominated assets in the market.

This is Merton's [9, Assumption 1, page 143] no dominance assumption for modern theory. This assumption consists of two parts. One is the fact that every asset has a unique market price. The second is that every asset is neither dominant nor dominant security. Then, we can conclude the Put-Call parity for the market prices, which is the direct consequence of no dominance assumption. $C_t(K) - P_t(K) = V_f^t(K) = S_t - K$. Now, we can prove the vital conclusion of this section that European Put options have no bubbles while European Call options can have bubbles.

Theorem 1.2.6 (No bubble in European Put options)**.** *For all $K > 0$, the fundamental price of a European Put option always equals its market price, i.e. $P_t(K) = P_t(K)^*$.*

Proof. Since the payoff of Put option has an upper bound K, and we know the wealth process of the underlying asset S is a local martingale, S, therefore, results to be a uniformly integrable martingale. Hence, there is no bubble in European Put options. Switching this martingale approach results in the charge approach of bubbles; we can get that the fundamental price of a European Put option always equals its market price. □

Theorem 1.2.7 (Bubbles in European Call options)**.** *European Call options can have bubbles. If there exists a bubble in the underlying asset, its corresponding European Call option with any strike $K > 0$, also has a bubble with the same size, i.e.*

$$C_t(K) - C_t(K)^* = S_t - E_{Q^*}\left[H_T(S)|\mathcal{F}_t\right] = \delta_t.$$

Proof. We begin with the Put-Call parity for market prices and fundamental prices

$$C_t(K) - P_t(K) = S_t - K.$$

$$C_t(K) - P_t(K) = V_f^t(K)^* = E_{Q^*}[S_T|\mathcal{F}_t] - K \leq S_t - K.$$

Subtracting these two equations,

$$\begin{aligned} |C_t(K) - C_t^*(K)| - |P_t(K) - P_t^*(K)| &= (S_t - K) - V_f^t(K)^* \\ &= (S_t - K) - (E_{Q^*}[S_T|\mathcal{F}_t] - K) \\ &S_t - E_{Q^*}[S_T|\mathcal{F}_t] \end{aligned}$$

Since $P_t(K) = P_t^*(K)$, $C_t(K) - C_t^*(K) = S_t - E_{Q^*}[S_T|\mathcal{F}_t]$, which means Call options can have bubbles and their size are same as the bubbles of the underlying assets. □

Since the data of options is far more than that of the underlying asset itself and the Call options have the same size of bubbles as the underlying asset, we are inspired to utilize Call option data to study bubbles.

1.3 Bubbles in different volatility models

To obtain a model that uses European Call options to detect their underlying asset's financial bubbles, we need to use the prices of these options to determine whether their underlying assets satisfy the strict local martingale property. In the follow-up practice, we will use all Call option prices daily to estimate the probability of the underlying asset satisfying the strict local martingale property on that day. So, we need to generate the training dataset before training the model. It should be a collection of Call option prices, some of which have underlying assets that follow a strict martingale process, and others have underlying assets that follow a true martingale.

To construct our training dataset, we first need to select a model to make assumptions about the price dynamics of the underlying asset. In this section, we delve into the geometric Brownian motion and the correlated Black-Scholes, CEV, and SABR models. Among them, the BS model is the foundational model, while the CEV and SABR models offer a more refined depiction of implied volatility behavior. In Subsection 2.3.2, we employ the SABR model to generate our training dataset.

1.3.1 Disadvantages of the Black-Scholes model

The Black-Scholes model is the most famous option pricing model based on assuming that the underlying price follows a geometric Brownian motion. It has a relatively simple formulation. However, the ideal assumptions of this model lead to problems in its practical application. For the model itself, the Black-Scholes model fails to capture the volatility skew or the volatility smile in reality since it assumes a constant volatility. In addition, it also fails to generate strict local martingale processes, which acted as training dataset for this thesis.

Let us first review the assumptions of the BS model [10, Section 1, page 640]:

- The price change of the underlying asset follows the geometric Brownian motion

$$\mathrm{d}S_t = \mu S_t \mathrm{d}t + \sigma S_t \mathrm{d}W_t$$

- The risk-free rate is constant
- The underlying asset does not pay dividends
- No transaction fees and taxes
- The market is complete and arbitrage-free.

There are some limitations in the BS model. One of the most critical assumptions of the BS model is that the risk-neutral dynamic of the underlying asset is a geometric Brownian motion whose implied volatility is constant, meaning that all options with strike prices and expiration dates should have the same implied volatility. This is not the case in reality. Resolving the implied volatility through the option price of PHLX, we will find that the implied volatility is not a constant(see Figure 1.1). In real markets, when traders and analysts use the BS model to back-calculate implied volatility, they find that implied volatility is not constant for all strike prices. Conversely, implied volatility varies with the strike price, forming a smile-like curve, the so-called volatility smile. The volatility smile is an apparent deviation from the BS model in the actual market. Indeed, the volatility smile suggests that some basic assumptions of the BS model, such as constant implied volatility, may not apply to real markets.

In addition, in the BS model, the risk-neutral dynamics of the underlying asset is a global martingale, meaning it is martingale over all time intervals. This is because, under the risk-neutral measure, the dynamics of the asset are entirely random, without any deterministic growth or decay. Hence, the BS model can not generate a strict local martingale process. The simplicity of the BS model and its assumption of risk-neutral dynamics of the underlying asset lead it to produce a martingale process rather than a strict local martingale. But in more complex models, especially considering volatility smiles and stochastic volatility, the underlying asset dynamics may be closer to strict local martingale.

To better capture the observed market phenomena, especially the volatility smile, the local and stochastic volatility models are built as extensions to the Black-Scholes model. The CEV and SABR models are the most representative of the local volatility and stochastic volatility models, respectively.

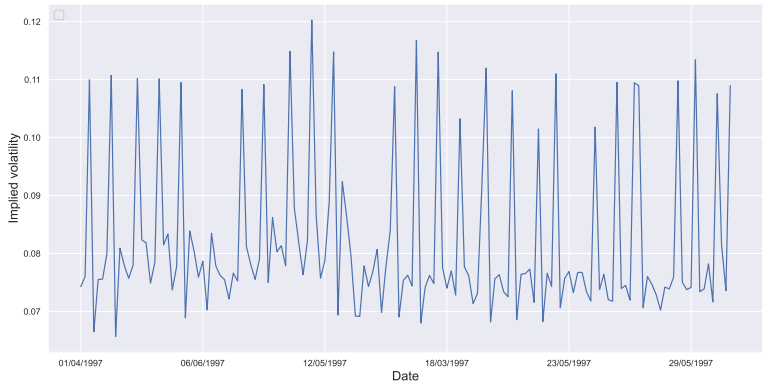


Figure 1.1: Implied volatility of PHLX.

1.3.2 Bubbles in the Local volatility model: CEV model

The local volatility model is one of the simplest revisions of the Black-Scholes model with two advantages. First, the local volatility models allow volatility to vary with underlying asset prices and time, better-capturing market dynamics and risk. Through Dupire's formula, the local volatility model provides us with a bridge connecting the market's option price and the underlying asset's dynamics. This allows us to understand better and explain price action in the market. The other advantage is that we can use the local volatility to generate a training dataset with bubbles by setting its parameter.

The local volatility assumes that the risk-neutral dynamics of stock prices satisfy:

$$\mathrm{d}S_t = (r-q)\, S_t \mathrm{d}t + \sigma\left(t, S_t\right) S_t \mathrm{d}W_t,$$

where $\sigma\left(t, S_t\right)$ is instantaneous volatility as a function of time and the stock price. The most important conclusion of the local volatility model is the Dupire formula, which relates the local volatility and implied volatility surfaces. Using the observed market prices of options (and their implied volatilities), the Dupire formula can back out the local volatility function.

Theorem 1.3.1 (Dupire formula)**.** *Let $C = C\left(K, T\right)$ be the price of a Call option. It is a function of time-to-maturity T and strike price K. The local volatility σ satisfies:*

$$\sigma^2\left(T, K\right) = \frac{\frac{\partial C}{\partial T} + (r-q)\, K \frac{\partial C}{\partial K} + qC}{\frac{K^2}{2} \frac{\partial^2 C}{\partial K^2}} \tag{1.3.1}$$

Proof. Let $p\left(s, t\right)$ be the probability distribution function(PDF) of the stock price at time t and the stock price S_t. It satisfies the Kolmogorov forward equation:

$$-p_t - (r-q)\left(s p_s\right) + \frac{1}{2}\left(\sigma\left(t, s\right) s^2 p\right)_{ss} = 0, t > 0,$$

the initial condition is $p\left(s, 0\right) = \delta_{S_0}\left(s\right)$. The price of the Call option can be written as:

$$C\left(K, T\right) = \mathrm{e}^{-rT} E_0\left[\left(S_T - K\right)_+\right] = \mathrm{e}^{-rT} \int_K^{+\infty} \left(s - K\right) p\left(s, T\right) \mathrm{d}s.$$

Taking the partial derivative of $C\left(T, K\right)$ with respect to K, we get

$$C_K\left(T, K\right) = \mathrm{e}^{-rT} \int_K^{+\infty} -p\left(s, T\right) \mathrm{d}s$$

Then taking the partial derivative with respect to K again, we get

$$C_{KK}\left(T, K\right) = \mathrm{e}^{-rT} p\left(s, T\right) \mathrm{d}s$$

Here we use Leibniz integral rule. Taking the partial derivative of $C(T,K)$ with respect to T, we get

$$C_T(K,T) = -rC(K,T) + \mathrm{e}^{-rT}\int_K^{+\infty}(s-K)\,p_T(s,T)\,\mathrm{d}s.$$

According to the Kolmogorov forward equation, we get

$$C_T = -rC + \mathrm{e}^{-rT}\int_K^{+\infty}(s-K)(r-q)(sp)_s\,\mathrm{d}s + \frac{1}{2}\mathrm{e}^{-rT}\int_K^{+\infty}(s-K)(r-q)\left(\sigma^2 s^2 p\right)_{ss}\mathrm{d}s.$$

Calculating integration by parts, we have

$$C_T = -rC + \mathrm{e}^{-rT}(r-q)\int_K^{+\infty} sp\mathrm{d}s + \frac{1}{2}\mathrm{e}^{-rT}\sigma^2 K^2 p.$$

According to all previous result and $\mathrm{e}^{-rT}\int_K^{+\infty} sp\mathrm{d}s = C - KC_K$ and $\mathrm{e}^{-rT}p = C_{KK}$, we get Dupire formula 1.3.1.

□

Given an existing volatility surface, the price of a Call option can be conceptualized as a function of both time to maturity T and strike price K, leading to the formation of a Call option price surface. Leveraging Dupire's formula, local volatility can be deduced by computing the partial derivatives of this Call option price. However, the process of determining local volatility is intricate. A smooth Black-Scholes implied volatility surface is imperative for accurately deriving local volatility using Dupire's methodology. While the local volatility model offers valuable insights, it has limitations. For instance, the model's portrayal of skew dynamics can be deemed unrealistic; it tends to underestimate the volatility of volatility, and the Greeks produced by the model might not always align with empirical observations. Despite these shortcomings, the local volatility model remains a popular choice in practice, especially for pricing barrier options.

Except for better capturing the volatility, the local volatility model also admits bubbles. We take the constant elasticity of variance (CEV) model as an example, which is the most famous local volatility model. The risk-neutral dynamics of it are

$$\mathrm{d}S_t = (r-q)S_t\mathrm{d}t + \sigma S_t^{\beta}\mathrm{d}W_t,$$

where β is the elasticity coefficient, Emanuel at el. [11, Section 2, page 534] has first proven that S_t is a true martingale for $\beta \leq 1$ and a strict local martingale for $\beta > 1$. Thus, we can generate the process with bubbles by setting the value of β and then calculating the option prices for our trading dataset.

1.3.3 Bubbles in the Stochastic volatility model: SABR model

The stochastic volatility model is another extension to the BS model, promoted by Hull and White [12, Section 1, page 282] in the late 1980s. Compared with the local volatility model, in which volatility changes deterministically based on the asset price and time, the stochastic model assumes the volatility is a stochastic process. Stochastic volatility has two sources of randomness: one for the asset and one for the volatility. The volatility σ_t of the underlying X_t is represented as a function $\sigma(Y_t)$, where Y_t is an auxiliary process. Typically, this auxiliary process Y_t is characterized as a diffusion:

$$\begin{aligned}\mathrm{d}X_t &= \phi X_t\mathrm{d}t + \sigma(Y_t)X_t\mathrm{d}W_t^1,\\ \mathrm{d}Y_t &= \mu Y_t\mathrm{d}t + \xi Y_t\mathrm{d}W_t^2,\\ \mathrm{d}W_t^1\mathrm{d}W_t^2 &= \rho\mathrm{d}t,\end{aligned}$$

where W_t^1, W_t^2 are two Wiener process and ρ is the correlation coefficient between them. Typically, correlation ρ is set to be negative to capture the reality that stock prices tend to go down when the volatility increases. This phenomenon is known as the leverage effect. Besides this, the stochastic volatility model can also generate data that satisfies local martingale. We take the SABR model as an example to display the process of generating strict local martingale data.

The SABR model is a stochastic volatility model proposed by Hagan [13, Section 2, page 5] in 2002. It specifies the dynamics of prices F_t and volatility σ_t as:

$$\begin{aligned} \mathrm{d}F_t &= \sigma_t F_t^{\beta} \mathrm{d}W_t^1, \\ \mathrm{d}\sigma_t &= \alpha \sigma_t \mathrm{d}W_t^2, \end{aligned} \tag{1.3.2}$$

where

$$\begin{aligned} \mathrm{d}W_t^1 \mathrm{d}W_t^2 &= \rho \mathrm{d}t, \\ W^1(0) &= W_0^1, \\ W^2(0) &= W_0^2. \end{aligned}$$

The SABR model abandons the assumption that the volatility is constant in the original BS model. It sets the forward price F_t of the underlying and the volatility σ_t, respectively, as random processes. The correlation between two random processes is ρ. The volatility σ_t follows lognormal distribution and incorporates the CEV model into the pricing equation. This makes the implied volatility estimated by the model consist of two parts. One part is the predictable part of the CEV model, and the other is the stochastic part in the stochastic volatility model. The fit and hedging effect of the model for the volatility smile is relatively good, but the solution is extremely complex and is an approximation solution. Due to the large number of parameters and their implicit correlation, it is difficult to effectively explain some parameters' estimation and economic meaning, such as beta.

In the SABR model, F_t and σ_t are random, whereas parameters β, α and ρ are constant. Among them, σ_t is a volatility-like parameter with a functional relationship with the implied volatility of at-the-money options. α represents the volatility of volatility, indicating the aggregation state of volatility. β determines the relationship between the underlying price and the average implied volatility. When $\beta \to 1$, the random model is close to lognormal, and when $\beta \to 0$, the random model is close to normal distribution. The SABR model is a strict local martingale if and only if $\rho \geq 0$. When $\rho < 0$, the SABR model is a true martingale. Thus, we can generate a training data set through the SABR model by adjusting the parameter ρ.

In general, the SABR model does not have a closed-form pricing formula. However, there are two special parameters settings under which the SABR model approximates the explicit Call price. The first setting is to set $\beta = 0$. Under this circumstance, the forward price F_t dynamics is just a combination of the volatility dynamics and a Wiener process, i.e. $\mathrm{d}F_T = \sigma_t \mathrm{d}W_t^1$. This refers to normal SABR, also Called the shifted lognormal model. This model has a simple representation of the approximated Call price but allows negative asset values, which cannot be observed when the underlying asset is an equity. Therefore, we abandon the case $\beta = 0$.

The second setting is to set $\beta = 1$. The system 1.3.2 then turns into the log-normal SABR model:

$$\begin{aligned} \mathrm{d}F_t &= \sigma_t F_t \mathrm{d}W_t^1 \\ \mathrm{d}\sigma_t &= \alpha \sigma_t \mathrm{d}W_t^2 \\ \mathrm{d}W_t^1 \mathrm{d}W_t^2 &= \rho \mathrm{d}t \end{aligned}$$

Then, from [13, Equation 94, page 21]

$$\xi = \frac{\alpha}{\sigma} \int_F^{F_0} \frac{\mathrm{d}u}{C(u)} = \frac{\alpha}{\sigma} \log\left(\frac{S_0}{F}\right)$$

which leads to an explicit implied volatility related to strike K. With this $\sigma(K)$, we can use Black's formula to compute the Call price.

Chapter 2

Artificial networks: FNN

Feedforward Neural Networks (FNNs) stand out for their unparalleled versatility in the realm of machine learning. Regardless of data intricacies or underlying semantics, when provided with appropriately processed input variables and target outputs, FNNs consistently yield robust and reliable models. This adaptability is further complemented by the extensive parameter space inherent to FNNs, granting practitioners a wide latitude for optimization and fine-tuning. While the internal dynamics of these networks might occasionally elude intuitive interpretation, and individual parameters may not always map directly to tangible market semantics, the empirical performance of FNNs remains indisputably commendable. Their ability to deliver impressive results across diverse datasets underscores their efficacy and reinforces their position as a cornerstone in modern machine-learning methodologies.

In this thesis, we use feedforward neural networks (FNN) twice. One is used to build the model of detecting bubbles, and once is used to create the model of predicting bubbles. The model for detecting bubbles needs each day's European Call option prices as input into the neural network and outputs a number between 0 and 1, representing the probability of being in a bubble that day. The prediction model uses the parameters of the SVI model fitted by the European Call option on the current day as the input value. The output result is a positive float, which means the number of days between the present day and the next bubble burst.

Section 2.1 introduces the history and development of Neural Networks. Section 2.2 delves into the foundational theoretical framework underpinning the FNN. A meticulous exploration of the parameters and configurations pertinent to the dual models we aim to construct will be undertaken. Section 2.3 presents a comprehensive implementation of our bubble detection model. The exposition on the bubble prediction model will be reserved for the next chapter, following our discourse on the SVI model.

2.1 Introduction of Neural Networks

Deep learning has been a research hotspot in the fields of computer science and artificial intelligence in recent years. It mainly focuses on modeling and representing complex data structures through multi-layer neural network models. The so-called "depth", as the name implies, refers to the frame with more layers in the model, which usually consists of multiple continuous and stacked processing layers.

The core idea of the neural network model is based on the working mechanism of the biological nervous system, especially the simplified simulation of the activity process of neurons. Mathematically, a basic neuron model can be viewed as a nonlinear map that summarizes its input signals, weights, accumulates, and nonlinearly transforms them to produce an output. Formally speaking, for a single neuron, its output y can be expressed as:

$$y = f\left(\sum_i \omega_i x_i + b\right),$$

where x_i represents the input variable. ω_i is the corresponding weight. b defines the bias term. f is a nonlinear activation function, like ReLU, Sigmoid, etc.

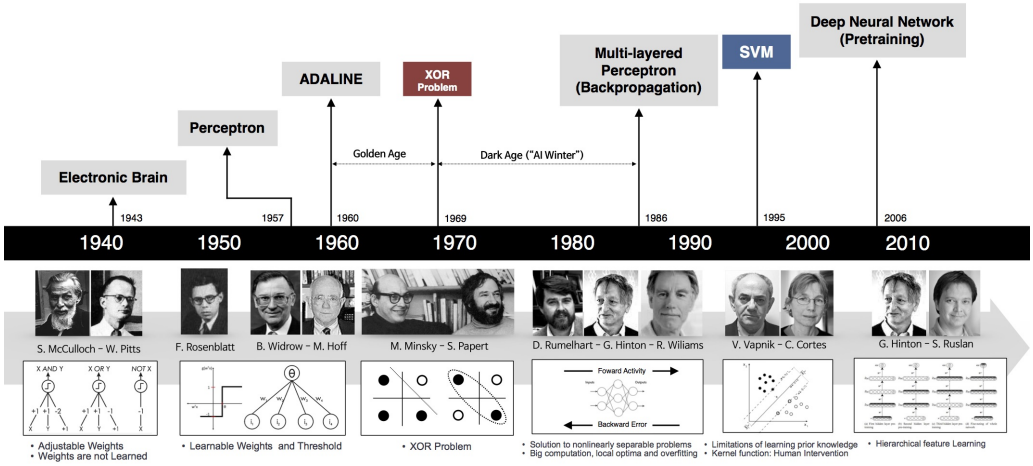


Figure 2.1: The history of Neural Networks over the last decades, source: `http://beamlab.org/deeplearning/2017/02/23/deep_learning_101_part1.html`

The history of neural networks(see Figure 2.1) can be traced back to the early 1950s [14], but it was not until the backpropagation algorithm proposed by Rumelhart et al. [15] in 1986 that the training of multi-layer neural networks achieved efficiency and feasibility. This algorithm efficiently computes the gradient of the loss function with respect to each weight based on the chain rule, thus providing a cornerstone for gradient descent optimization.

In the early 2000s, deep learning experienced a real turning point. In 2006, pre-training techniques for multi-layer neural networks enabled efficient training of deep architectures to solve a range of unprecedentedly complex tasks [16]. Subsequent research and application progress have sprung up, especially the remarkable achievements of convolutional neural network (CNN) in the field of image recognition and long and short-term memory network (LSTM) in the field of sequence processing, such as machine translation and speech recognition [17] [18].

At present, deep learning has gradually surpassed the boundaries of traditional fields and has been broadly used in various areas such as medical care, finance, and autonomous driving. The potential and prospects are still extensive. Coupled with the continuous optimization of algorithms, the constant enhancement of computing power, and the abundance of data resources, the future of deep learning is expected to achieve more technological innovations and application revolutions. With more researchers and engineers joining, the prospect of this field is still in vigorous development.

Looking forward to the future, the application and research of deep learning will continue to deepen, bringing more insights into artificial intelligence and many related fields.

2.2 Feedforward neural networks (FNN)

In this section, we will present the structure and essential properties of the FNN, which is inspired by deep learning lecture notes [19]. In addition, we discuss all the details of the FNN implementation in our thesis, including the choice of each structure component.

2.2.1 General architecture

Definition 2.2.1 (Feedforward neural network)**.** Let $I, O, r \in \mathbf{N}$. A function $f : \mathbb{R}^I \to \mathbb{R}^O$ is a feedforward neural network (FNN) with $r-1 \in \{0, 1, ...\}$ hidden layers. The i-th hidden layer contains $d_i \in \mathbf{N}$ units for $i \in \{1, 2, ...r-1\}$, and activation functions $\sigma_i : \mathbb{R}^{d_i} \to \mathbb{R}^{d_i}$, $i = 1, 2, ...r$, where $d_r = O$ and

$$f = \sigma_r \circ L_r \circ \cdots \circ \sigma_1 \circ L_1,$$

where $L_i : \mathbb{R}^{d_{i-1}} \to \mathbb{R}^{d_i}$, for any $i = 1, 1..., r$, is an affine function

$$L_i(x) := W^i x + b^i, \quad x \in \mathbb{R}^{d_{i-1}},$$

where $W^i = \left[W^i_{j,k}\right]_{j=1,...,d_i, k=1...d_{i-1}} \in \mathbb{R}^{d_i \times d_{i-1}}$ represents the weight matrix and $b^i = \left(b^i_i, ..., b^i_{d_i}\right) \in \mathbb{R}^{d_i}$ represents bias vector, with $d_0 := I$. We denote the class of such function f by

$$\mathcal{N}_r\left(I, d_1, ..., d_{r-1}, O; \sigma_1, ..., \sigma_r\right).$$

The integers $r, d_1, ..., d_i$ are hyperparameters. Weight matrices $W^1, ..., W^r$ and biases in $b^1, ...b^r$ are actual parameters. Activation functions $\sigma_1, ..., \sigma_r$ together with the variables mentioned above determine the network's architecture.

2.2.2 Activation functions

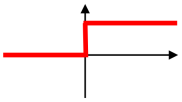
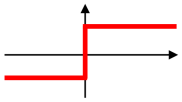
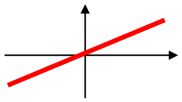
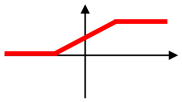
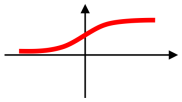
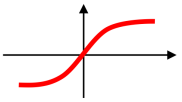
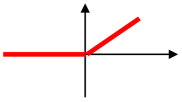
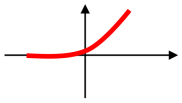
Activation function	Equation	Example	1D Graph
Unit step (Heaviside)	$\phi(z) = \begin{cases} 0, & z < 0, \\ 0.5, & z = 0, \\ 1, & z > 0, \end{cases}$	Perceptron variant	
Sign (Signum)	$\phi(z) = \begin{cases} -1, & z < 0, \\ 0, & z = 0, \\ 1, & z > 0, \end{cases}$	Perceptron variant	
Linear	$\phi(z) = z$	Adaline, linear regression	
Piece-wise linear	$\phi(z) = \begin{cases} 1, & z \geq \frac{1}{2}, \\ z + \frac{1}{2}, & -\frac{1}{2} < z < \frac{1}{2}, \\ 0, & z \leq -\frac{1}{2}, \end{cases}$	Support vector machine	
Logistic (sigmoid)	$\phi(z) = \frac{1}{1 + e^{-z}}$	Logistic regression, Multi-layer NN	
Hyperbolic tangent	$\phi(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$	Multi-layer Neural Networks	
Rectifier, ReLU (Rectified Linear Unit)	$\phi(z) = max(0, z)$	Multi-layer Neural Networks	
Rectifier, softplus	$\phi(z) = \ln(1 + e^z)$	Multi-layer Neural Networks	

Figure 2.2: Common one-dimensional activation functions and their properties, source: `https://sebastianraschka.com/faq/docs/activation-functions.html`

In deep learning and neural networks, activation functions are crucial components that introduce nonlinearity to the neural network, allowing the network to fit complex non-linear functions. No matter how many layers a neural network has, it can only represent linear transformations without an activation function.

Given input x, the purpose of the activation function f is to provide an output to the neuron $y = f(x)$. This output is then passed on to the next layer of the network. Introducing nonlinearity is a crucial factor for neural networks to be able to solve complex tasks. Nonlinearity enables

the network to approximate arbitrarily complex functions based on the universal approximation theorem of neural networks.

The logistic (sigmoid) function maps input values between 0 and 1. However, the function has the problem of vanishing gradients in its saturated regions (i.e., regions where the input values are extremely large or extremely small). The logistic function is often used to implement the activation function of the output layer of the binary classification task. The output value is taken as the probability of being classified into one of the classes. The hyperbolic tangent function maps input values between -1 and 1. Compared to the logistic function, its output is zero-centered. The ReLU is one of the modern neural networks' most commonly used activation functions. It's non-linear, even though it looks like a linear function. The ReLU brings a considerable speed boost to the training of deep learning models since its derivative is one at positive values and its computational cost is low. The activation functions mentioned above are one-dimensional. More one-dimensional activation functions are shown in 2.2. Also, there are some multi-dimensional activation functions. The softmax function is used in the output layer for multiclass classification. It converts an input vector into a probability distribution.

When choosing an activation function, factors include network depth, data distribution, required model complexity, and training speed. For example, in deep networks, it may be more beneficial to use the ReLU because it alleviates the problem of vanishing gradients. However, it can also lead to dead neurons, at which point the Leaky ReLU or other variants may be more suitable.

For the bubble detection model, it is imperative to recognize that we are addressing a binary classification problem. The objective is to ascertain the probability of the presence of a bubble. Given this framework, the output should naturally represent the likelihood of a bubble's existence. The sigmoid activation function emerges as the most fitting choice in this scenario, primarily because its output range, between 0 and 1, aligns seamlessly with the probabilistic interpretation we seek. By employing the sigmoid function, we ensure that the model's output can be directly and intuitively interpreted as a bubble's probability, lending clarity and credibility to our model's predictions.

In the design of our bubble prediction model, the primary objective is to estimate the number of days until the next bubble burst. Given this goal, the model's output should manifest as a non-negative float number, indicative of the duration. This requirement inherently classifies our task as a regression problem. Consequently, the ReLU activation function becomes optimal for the output layer. The non-negative output range of the ReLU function aligns perfectly with our model's intent, ensuring that the predictions are meaningful and contextually appropriate. By leveraging the ReLU activation, we guarantee a coherent representation of the anticipated days, enhancing the model's interpretability and reliability.

2.2.3 Universal approximation property

In the study of neural networks, the general approximation property is a core concept, which refers to the ability of neural networks with a specific structure to represent various complex functions. This universal approximation property builds solid support for the solvability of our problems. We employ this core property that the task of detecting and predicting financial bubbles can be expressed and solved by the FNNs.

To formalize this idea, clarifying two norms for functions is necessary. Let $K \subset \mathbb{R}^I$be compact, which means it is both closed and bounded. For any measurable $f : \mathbb{R}^I \to \mathbb{R}$, the supremum norm is defined as

$$||f||_{sup,K} := \sup_{x \in K} |f(x)|,$$

and for any $p \geq 1$, the L^p norm

$$||f||_{L^p(K)} := \left(\int_K |f(x)^p \, \mathrm{d}x \right)^{\frac{1}{p}},$$

Theorem 2.2.2 (Universal approximation property)**.** *Let $g : \mathbb{R} \to \mathbb{R}$ be a measurable function such that*

- *g is not a polynomial function,*
- *g is bounded on any finite interval,*

- *the closure of the set of all discontinuity points of g in* $\mathbb{R}$ *has zero Lebesgue measure.*

Moreover, let $K/subset\mathbb{R}^I$ *be compact and* $\epsilon > 0$*. Then:*

- *For any* $u \in C(K, R)$*, there exist* $d \in \mathbf{N}$ *and* $f \in \mathcal{N}_2(I, d, 1; g, Id)$ *such that*

$$||u - f||_{sup,K} < \epsilon.$$

- *Let* $p \geq 1$*. For any* $v \in L^p(K, \mathbb{R})$*, there exist* $d' \in \mathbf{N}$ *and* $h \in \mathcal{N}_2(I, d, 1; g, Id)$ *such that*

$$||v - h||_{L^p(K)} < \epsilon.$$

2.2.4 Loss function

The training of deep learning models is done by optimizing the loss function

$$l : \mathbb{R}^O \times \mathbb{R}^O \to \mathbb{R}.$$

A loss function is a scalar function that measures the difference between a model's prediction and the actual value. The goal of the model is to find a set of parameters that minimizes the value of this function.

If X is the model's prediction and Y is the true value, we seek an optimal f by minimizing the risk.

$$E\left[l\left(f\left(X\right), Y\right)\right]$$

In practice, we use empirical risk

$$\mathcal{L}\left(f\right) := \frac{1}{N}\sum_{i=1}^{N} l\left(f\left(x^i\right), y^i\right).$$

since we do not know the distribution of (X, Y).

Here are some common loss functions and their mathematical definitions:

- Mean Squared Error (MSE): For regression problems, mean squared error is the most commonly used loss function.

$$l\left(X, Y\right) = \frac{1}{N}\sum_{i=1}^{n}\left(x^i - y^i\right)^2$$

- Cross entropy loss: Cross entropy loss is most commonly used for classification problems.

$$l\left(Y, P\right) = -\sum_i y_i \log\left(p_i\right),$$

  where Y is the true label distribution (usually a one-hot encoded vector), while P is the probability distribution predicted by the model.

- Kullback-Leibler divergence: KL divergence Measures the difference between two probability distributions.

$$D_{KL}\left(P||Q\right) = \sum_i P\left(i\right) \log\left(\frac{P\left(i\right)}{Q\left(i\right)}\right)$$

  , where P is the true distribution and Q is the distribution predicted by the model.

For the bubble detection model, we choose binary cross-entropy loss. For the bubble prediction model, we select MSE as the loss.

2.2.5 Minibatch, epoch and earlystopping

Minibatch

When training neural networks or other deep learning models, using the entire dataset for weight updates can be very inefficient, especially in large data environments. To overcome this problem, we divide the dataset into smaller subsets Called mini-batches or minibatches.

Suppose the data set D has N samples $D = \{x_1, x_2, ..., x_N\}$, We can divide D into M mini-batches. Each mini-batch D_i has B samples $D_i = \left\{x_{(i-1)B+1}, ..., x_{iB}\right\}$where B is the size of the mini-batch, satisfying

$$M \times N = D$$

Epoch

One epoch means the model has done one forward and one backward pass through the entire training set. In other words, every time the model goes through all minibatches once, we say it has completed a cycle. Typically, models require multiple epochs of training to converge. Described in mathematical terms, if we have N training samples and choose the mini-batch size as B, then one cycle will contain $M = \frac{M}{B}$ mini-batch training.

Earlystopping

To avoid overfitting the model on the training set, we can use a regularization technique Called early stopping. At the end of each epoch, we evaluate the performance of the model using the validation set. If the validation performance of the model does not improve for P consecutive epochs, we stop training and use the best model.

Expressed in mathematical language, let $J(t)$ denote the validation loss at the end of the t-th epoch. We compute this loss at the end of each epoch. If there exists a threshold P such that for all $i \in [t - P, t]$, then we stop training.

To sum up, mini-batches, epochs, and early stopping are three critical concepts in deep learning training that work cooperatively to ensure models learn efficiently and avoid overfitting. In the design of our model's training regimen, we have judiciously set the number of epochs to 10,000, a notably substantial figure. And we've configured the patience parameter of the early stopping mechanism to 100. This strategic combination ensures a comprehensive exploration of the training landscape, allowing the model ample opportunity to converge. By adopting this approach, we effectively mitigate the risks associated with both underfitting and overfitting, striking an optimal balance that enhances the model's generalization capabilities and ensures robust performance on unseen data.

2.2.6 Optimiser

SGD

Stochastic Gradient Descent (SGD) is the first derivative-based optimization algorithm for finding the minimum of a loss function. SGD uses only one (or mini-batch) of training samples at each iteration to estimate gradients rather than using the entire training dataset.

The iterative update formula of SGD is:

$$\theta_{t+1} = \theta_t - \eta \nabla l(\theta_t),$$

where θ_t represents the model parameters at iteration t. $\nabla l(\theta_t)$ is the gradient of the loss function at θ_t. η is the learning rate, which determines the step size of the model parameter update.

Momentum

Momentum is an optimization technique based on the concept of Momentum in physics. In deep learning, using Momentum can help speed up the convergence of SGD in relevant directions and reduce oscillations. It simulates a rolling object that is affected by the current gradient and retains the previous velocity.

Momentum updates can be expressed as two formulas:

$$\begin{aligned} v_t &= \beta \times v_{t-1} + \eta \times \nabla l(\theta_t), \\ \theta_{t+1} &= \theta_t - v_t, \end{aligned}$$

where v_t is the velocity at time t and is also the moving average of the gradient. β is a decay factor close to 1, such as 0.9, representing the magnitude of Momentum.

The key idea of Momentum is that if a dimension keeps going up or down, then the cumulative velocity (moving average) will help to speed up the optimizer's progress in that dimension and speed up the training. On the other hand, if the gradient of one dimension keeps changing direction, the accumulation speed will be smaller, which helps to reduce the optimization shock.

This approach is constructive for saddle points on the optimization surface. Since near saddle points, the gradient can become very small, causing pure SGD to almost stall. Using Momentum can help the algorithm keep moving in this situation, finding a solution faster.

In short, the Momentum optimizer borrows from the concept that objects accumulate velocity when going downhill, which helps the optimizer traverse small flat areas and saddle points faster.

Adagrad

Adagrad is an adaptive learning rate optimization algorithm. Its core idea is to dynamically adjust the learning rate for each model parameter based on the historical gradient information of the parameter. Frequently updated parameters will have a smaller learning rate, while less regularly updated ones will have a larger one.

The parameter update rule can be described as follows:

- Cumulative squared gradients:

$$G_t = G_{t-1} + \nabla l\left(\theta_t\right)^2.$$

- Update parameters:

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{G_t + \epsilon}} \nabla l\left(\theta_t\right).$$

G_t represents the cumulative sum of squared gradients up to iteration t. ϵ is a minimal positive value to avoid the case where the denominator is zero.

The main advantage of Adagrad is that it does not require manual tuning of the learning rate, and it generally performs exceptionally well on sparse datasets and online learning tasks. However, due to the cumulative effect of squared gradients, the practical learning rate may drop rapidly to minimal values, allowing the algorithm to make very little progress or stop updating altogether late in training. To solve this problem, other algorithms, such as RMSProp and Adam, were later proposed, which improved the basic idea of Adagrad and added a moving average of the gradient to avoid the problem of premature decay of the learning rate.

RMSprop

RMSprop is an adaptive learning rate optimization algorithm, which solves the problem that Adagrad's learning rate may drop sharply during training. RMSprop adjusts the learning rate by using a moving average of gradients, thus ensuring that the learning rate remains appropriate throughout training.

The parameter update rule can be described as:

- compute the moving average of the squared gradient:

$$G_t = \beta G_{t-1} + (1-\beta) \nabla l\left(\theta_t\right)^2.$$

- Update parameters:

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{G_t + \epsilon}} \nabla l\left(\theta_t\right).$$

RMSprop dynamically adjusts the learning rate for each parameter, where the learning rate is adjusted based on the magnitude of the most recent gradient. This method allows the learning rate to maintain a relatively appropriate size during the training process, avoiding the problem of premature decay of the learning rate. RMSprop can usually provide a stable and efficient training process, especially for non-steady state and online learning tasks. RMSprop and its variants, such as Adam, are now more popular in the deep learning community because they solve the problem of premature decay of the Adagrad learning rate.

Adam

Adam is an adaptive learning rate optimization algorithm that combines the ideas of Momentum and RMSprop. Adam uses the first and second-moment estimates of the gradient to dynamically adjust the learning rate for each parameter. Due to its balance and efficiency, Adam is one of the most popular optimization algorithms in deep learning.

- compute estimates of the first moment (Momentum) and second moment (RMSprop) of the gradient:

$$\begin{aligned} v_t &= \beta_1 \times v_{t-1} + (1-\beta_1)\nabla l(\theta_t), \\ G_t &= \beta_2 G_{t-1} + (1-\beta)\nabla l(\theta_t)^2. \end{aligned}$$

- To correct for biased estimates in the initial phase, perform a bias correction:

$$\begin{aligned} \hat{v_t} &= \frac{v_t}{1-\beta_1}, \\ \hat{G_t} &= \frac{G_t}{1-\beta_2}. \end{aligned}$$

- Update parameters:

$$\theta_{t+1} = \theta_t - \frac{\eta \hat{v_t}}{\sqrt{\hat{G_t} + \epsilon}}.$$

Adam provides a dynamic learning rate adjustment for each parameter by synthesizing the ideas of Momentum and RMSprop. The first-order moment estimation provides the directional acceleration, and the second-order moment estimation ensures the adaptive learning rate. At the same time, Adam added a bias correction step to ensure the estimate is unbiased in the early stages of training. This comprehensiveness and balance make Adam perform well in many deep-learning tasks and become one of the default optimizer choices.

2.2.7 Backpropagation

The backpropagation algorithm is the core algorithm used in training neural networks, and it is an effective gradient calculation method based on the chain rule. It is divided into two stages: forward propagation and backpropagation. In forward propagation, the input is passed through the network layer by layer until the final output is produced. In backpropagation, the computation starts at the output layer and is propagated back layer by layer to compute the gradient of the loss function with respect to each parameter.

To show backpropagation theoretically, we first introduce the adjoint and chain rule. Assume that we are training a FNN $f_\theta \in \mathcal{N}_r(I, d_1, ..., d_{r-1}, O, \sigma_1, ..., \sigma_r)$ by SGD, σ_i is the component-wise application of a one-dimensional activation function $g_i : \mathbb{R} \to \mathbb{R}$, for $i = 1, ..., r$. Denote derivative of g_i as g_i'. For $x \in \mathbb{R}^I$,

$$\begin{aligned} \mathbf{z}^i = \left(z_1^i, ..., z_{d_i}^i\right) &:= W^i \mathbf{a}^{i-1} + \mathbf{b}^i, \quad i = 1, ..., r, \\ \mathbf{a}^i = \left(a_1^i, ..., a_{d_i}^i\right) &:= \mathbf{g}_i\left(\mathbf{z}^i\right), \quad i = 1, ..., r, \\ \mathbf{a}^0 &:= \mathbf{x}, \end{aligned}$$

so then $f_\theta(\mathbf{x}) = \mathbf{a}^r$ and $l(f_\theta(\mathbf{x}), \mathbf{y}) = l(\mathbf{a}^r, \mathbf{y})$. Adjoint $\delta^i = \left(\delta_1^i, ..., \delta_{d_i}^i\right) \in \mathbb{R}^{d_i}$ is defined as

$$\delta_j^i := \frac{\partial l}{\partial z_j^i}, j = 1, ..., d,$$

for any $i = 1, ..., r$.

Theorem 2.2.3 (Chain rule)**.** *For differentiable $G : \mathbb{R}^d \to \mathbb{R}$ and $\mathbf{F} = (F_1, ..., F_d) : \mathbb{R}^{d'} \to \mathbb{R}^d$, define $H(\mathbf{x}) = G(\mathbf{y})$ with $\mathbf{y} = (y_1, ..., y_d) = F(\mathbf{x})$, that is, $H = G \circ F : \mathbb{R}^{d'} \to \mathbb{R}$. Then*

$$\frac{\partial H}{\partial x_i}(\mathbf{x}) = \sum_{j=1}^{d} \frac{\partial G}{\partial y_j}(\mathbf{y}) \frac{\partial F_j}{\partial x_i}(\mathbf{x}).$$

Theorem 2.2.4 (Backpropagation)**.** *Using the chain rule and the adjoint, we can derive a backward recursive procedure for the components of $\nabla_\theta l(f_\theta(\mathbf{x}), \mathbf{y})$:*

$$\begin{aligned} \delta^r &= \mathbf{g}_r'(z^r) \odot \nabla_{\hat{y}} l(\mathbf{a}^r, \mathbf{y}), \\ \delta^i &= \mathbf{g}_r'\left(z^i\right) \odot \left(W^{i+1}\right)' \delta^{i+1}, && i = 1, ..., r-1, \\ \frac{\partial l}{\partial b_j^i} &= \delta_j^i, && i = 1, ..., r, j = 1, ..., d_i, \\ \frac{\partial l}{\partial W_{j,k}^i} &= \delta_j^i a_k^{i-1}, && i = 1, ..., r, j = 1, ..., d_i, k = 1, ..., d_{i-1}, \end{aligned}$$

where $\odot$ stands for the component-wise Hadamard product of vectors.

2.3 Application of neural network: bubble detection

We have introduced a set of fundamental neural network theoretical systems above. In this section, we provide an overview of a pivotal study that investigates the application of neural networks in detecting asset price bubbles. This discussion's foundational ideas and methodologies are rooted in the work of Biagini et al. [2, Section 2, page 5] . While we aim to offer a comprehensive summary and elucidation of their key insights and conclusions, we also integrate discussions from other related literature. Training by the data generated from the SABR model, our bubble detection neural network can tell the probability of a specific date being in a financial bubble from observed Call prices in the real market.

2.3.1 The feasibility of the model

Recall that Section 1.2 has mentioned that European Call options can have bubbles, and the bubble's size equals the bubble's size in their underlying assets. Therefore, we detect bubbles from European Call options since, on each date, many more Call prices are available in the market than a single spot price of the underlying.

Let us give some symbolic descriptions and variable definitions.

- Spaces M_{loc}:

$$M_{loc} = \left\{ X = (X_t)_{t\geq 0} \text{ continuous positive local martingale on } (\Omega, \mathcal{F}, \mathcal{Q}, \mathbb{F}) \text{ with } X_0 = x_0 \right\}$$

- C^X: the Call option price.
- Spaces M_L: $\{X \in M_{loc} : \text{X is a strict local martingale.}\}$.
- Set X: $\mathcal{X} := \left\{ C^X : X \in M_{loc} \right\} \cap C\left([0,\infty) \times (0,\infty), [0,\infty)\right)$.
- Bubble detection function F: a function that maps from $\mathcal{X}$ to $\{0,1\}$ for detecting asset price bubbles. It has the value one if and only if the underlying process is a strictly local martingale.
- Set A: $A := [0,\infty] \times (0,\epsilon)$, for some $\epsilon > 0$.
- $(T_n, K_n)_{n\in\mathbb{N}}$: a sequence of maturities and strikes which is dense in A.
- Probability measure μ and v: two probability measure on $\mathcal{X}$ satisfying

$$v\left(\{f \in \mathcal{X} \text{ such that } f(T_i, K_i) = g \text{ for all } i \in \mathbb{N}\,(T_i, K_i)\}\right) > 0 \text{ for any } g \in supp(\mu)$$

The following are some critical theorems from Biagini et al. [2, Section 2.1, pages 5-9] in the process of proving the feasibility of using neural networks to detect bubbles without proof.

Theorem 2.3.1 (The existence of F)**.** *There always exists a measurable function $F : \mathcal{X} \to \{0,1\}$ such that $F(X) = \mathbb{1}_{\{X\in M_L\}}$.*

Theorem 2.3.2 (Approximation sequence F^n)**.** *Fix $p \geq 1$. There always exists a sequence of functions $(F^n)_{n\in\mathbb{N}}$, $F^n : \mathbb{R}^n \to [0,1]$ such that*

$$\int_{\mathcal{X}} |F^n(g(T_1,K_1),...,g(T_n,K_n)) - F(g)|^p d\mu(g) \underset{n\to\infty}{\longrightarrow} 0$$

Theorem 2.3.3 (Neural network approximation)**.** *Fix $p > 1$. For any $\epsilon > 0$, there exists an $n \in \mathbb{N}$ and a neural network $\hat{F}^n : \mathbb{R}^n \to [0,1]$ such that*

$$\left(\int_{\mathcal{X}} |\hat{F}^n(g(T_1,K_1),...,g(T_n,K_n)) - F(g)|^p d\mu(g) \right)^{\frac{1}{p}} < \epsilon$$

Since $\hat{F}^n$ can approximate F^n, F^n can approximate F, so $\hat{F}^n$ can approximate F. These steps provide us with a structured approach to demonstrate that neural networks can approximate asset price bubble detection functions F.

2.3.2 Data preparation: SABR generation

Since the bubble is a strict local martingale, and we can distinguish it easily in the SABR model by setting the correlation parameter larger or equal to zero, the SABR model is a suitable alternative for data generation. In addition, together with setting $\beta = 1$, the SABR model can give explicit Call prices. Therefore, we employ the log-normal SABR model to generate a training dataset for our neural network.

In the scope of this research, our paramount objective is to devise a model proficient in detecting financial bubbles. For a single asset, the raw data under examination consists of an extensive array of Call option data, encompassing key metrics such as Call prices, strike values, maturities, and the underlying asset price. Our approach begins by segmenting this data chronologically, enabling a focused analysis of each day's dataset. For every distinct day, a specialized Fully Connected Neural Network is constructed.

So next, we only consider all Calls of a single stock within a single day. In this case, given the consistency in the underlying asset and the date, every option has the same underlying price. The variation in Call prices can be attributed to the diverse combinations of strikes K and maturities T. We generate many option prices for each unique pair of (K, T). These are meticulously derived from the SABR model, with each price being the outcome of a distinct SABR model configuration. While specific parameters within these SABR models are fixed, others are randomly sampled from uniform distributions. It's imperative to highlight that half of these SABR models are characterized by positive ρ values, signifying strict local martingales, while the remaining, with negative ρ values, represent true martingales. We compute corresponding Call prices by utilizing these SABR models' respective (K, T) pairs and the same underlying price. In essence, the SABR model facilitates the generation of a collection of Call prices, of which half indicate the presence of bubbles, while the others suggest their absence.

To enhance reader comprehension, we provide an illustrative example. For the case of Apple, our study focuses on analyzing the information from all Calls from July 7, 2021, to January 3, 2022, spanning a total of 126 trading days. In this context, a distinct neural network will be constructed for each trading day, resulting in a total of 126 neural networks. Each neural network is dedicated to predicting the likelihood of Apple being in a bubble state on its respective day.

To illustrate the process, let's consider the instance of Apple's Calls on July 7, 2021. On this day, there were a total of 182 Call options, representing 182 (K, T) combinations. Subsequently, we develop 200 SABR models with a mix of fixed and randomly generated parameters, as detailed in Section 2.3.3. Notably, among these models, 100 SABR models have $\rho > 0$, while the remaining 100 models have $\rho < 0$. This classification implies that 100 SABR models describe wealth processes as strict local martingales, while the other 100 represent true martingales.

Further progressing, based on the 182 (K, T) combinations, the price of Apple on July 7, 2021, and the interest rate, we calculate Call option prices. This process yields a matrix of Call prices with dimensions 182×200. Conceptually, these prices form 200 individual samples, each comprising a sequence of Calls corresponding to the 182 different (K, T) combinations. The variability between samples stems from the differing parameter choices in the SABR models.

Within this framework, 100 samples correspond to Call price sequences under the assumption of Apple being in a bubble state on July 7, 2021. They are labeled as "1". Conversely, the other 100 samples correspond to Call price sequences under the assumption of Apple not being in a bubble state on the same day, and are labeled as "0".

The subsequent step involves constructing a fully connected neural network with an input layer containing 182 nodes and an output layer with a single node. This network is trained using the generated samples. After training, we input the actual 182 Call prices from July 7, 2021, for Apple into the neural network, yielding a numerical output in the range of 0 to 1, indicating the probability of Apple being in a bubble state on that specific day.

It is essential to highlight that the (K, T) combinations for the 200 generated samples align precisely with the actual (K, T) combinations of Calls observed on that day. This alignment permits us to directly put the real Call prices into the model fitted by generated data. This approach is particularly apt for the bubble detection model, wherein the SABR models are adept at constructing the desired samples with both labels "0" and "1".

In contrast, for the bubble prediction model, the objective is to forecast the number of days until the next bubble burst, which is a regression task. Each day's Calls are associated with a single label, and there is no way to generate samples with varying labels. Consequently, we must leverage Calls from numerous distinct days to construct a model collectively. We still regard

each day's Calls as a single sample. The varying (K, T) combinations across different days lead to discrepancies in the number of Call prices. To address this, we propose using the SVI model discussed in Chapter 3 to fit a volatility surface for each day. Then select specific fixed (K, T) pairs and calculate their corresponding volatilities and Call prices. This process is a normalization of the shape and meaning of the sample, allowing the construction of a generalized prediction model.

In practice, the SVI model parameters derived from fitting the volatility surface for each day are employed as inputs for the neural network. This approach omits the pricing process, as these parameters can encapsulate the volatility surface, and consequently, the Call prices for any (K, T) pair can be determined. Further details about the bubble prediction model will be provided in Chapter 4.

2.3.3 Experimental result

The process of data preparation and training neural network has been introduced in Subsection 2.3.2. In this part, we focus on describing the data, introducing parameter settings and presenting results.

Data description

Since the purpose of this paper is to expand the the model from Biagini et al.[2] model into a complete bubble detection and prediction system, we first reproduce some conclusions of their model. From [2, Chapter 4, pages 24-29], we can almost be sure that by the end of 2021, there exsited financial bubbles for stocks of Nvidia, Apple and Tesla. We first find the dates when their stock prices reach their highest, and use them as the dates of bubbles burst. We study their respective Calls within the 180 days before the bubble burst (including non-trading days). Taking Nivida as an example, the date of bubbles burst is Nov 29, 2021. There were 126 trading days in the 180 days before the bubble burst, and there were approximately 400 calls on each trading day.

Parameter settings

200 SABR models are built to generate 200 samples. We set $\beta = 1$, in order to be able to compute the Call prices analytically. σ_0 and α are both uniformly distributed in $[0.1, 0.9]$. For 100 SABR models, ρ is uniformly distributed in $[0.1, 0.9]$ (the ones where the underlying is a strict local martingale). For the other 100 models, ρ is uniformly distributed in $[-0.9, 0.1]$ (the ones where the underlying is a true martingale).

The 200 generated samples are divided into training set, validation set and test set according to the ratio of 6:2:2. The neural network has two hidden layers, each with 32 nodes and activated by the ReLU function. The output layer comprises a single node with a sigmoid activation function. The sigmoid activation function outputs values in the range $(0, 1)$, which is suitable for binary classification tasks like the one being undertaken here.

The batchsize is 20 and the learning rate is 0.01. The epoch is set as 2000 while the patience of the earlystopping is 30. These settings ensures that the model is neither overfitting nor underfitting. The loss function chosen is Binary Cross-Entropy, which is appropriate for binary classification tasks involving probability estimates.

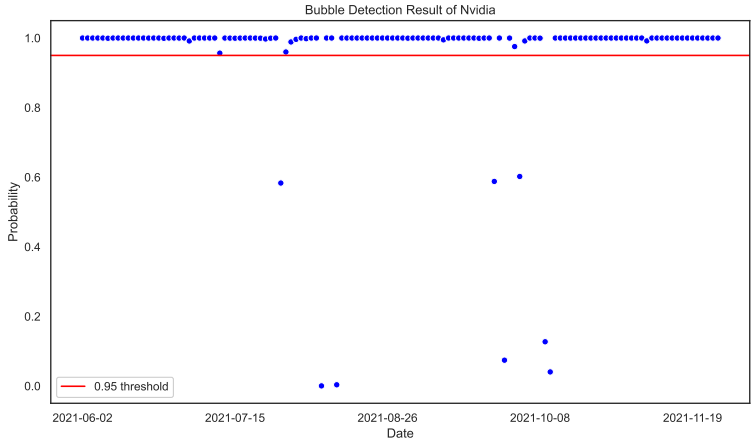
Display of results

We draw scatter plots that each point represents the probability of being in a bubble at the specific day.

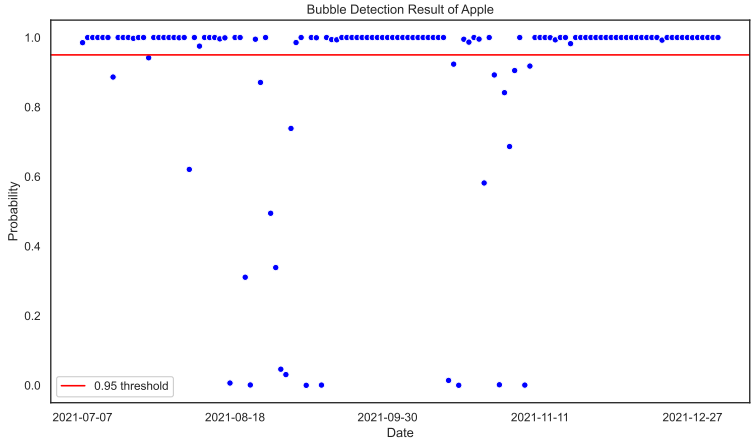
Figure 2.3(a) summarizes that for Nvidia, 118 of 126 trading days have probabilities more than 95% of being in a bubble, and the average value is 0.951. This result shows that we can almost be sure that Nvidia was in the bubble during this half year, consistent with Biagini et al. [2, Figure 1, page 26].

Figure 2.3(b) summarizes that for Apple, 101 out of 126 trading days have probabilities more than 95% of being in a bubble, and the average value is 0.888. This result shows that we can almost be sure that Apple was in the bubble during this half year, consistent with the result of Biagini et al. [2, Figure 2, page 27].

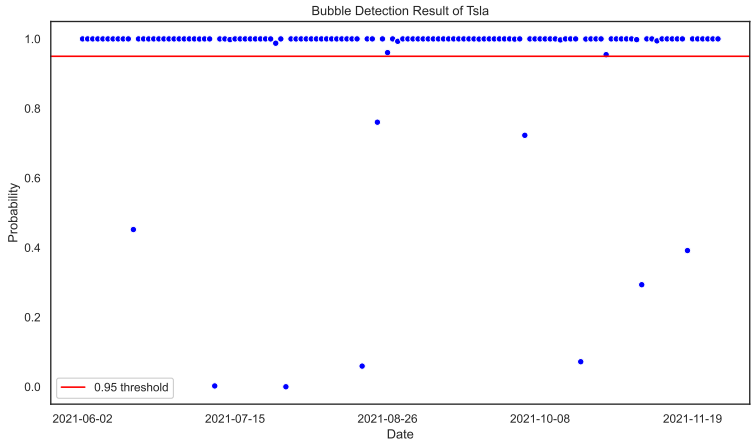
Figure 2.3(c) summarizes that for Tesla, 117 out of 126 trading days have probabilities more than 95% of being in a bubble, and the average value is 0.949. This result shows that we can



(a) Nvidia.



(b) Apple.



(c) Tesla.

Figure 2.3: Probability scatter plots of bubble detection on three stocks.

almost be sure that Tesla was in the bubble during this half year, consistent with the result of Biagini et al. [2, Figure 3, page 28].

Chapter 3

Arbirtrage-free SVI volatility surfaces

As discussed at the end of Subsection 2.3.2, the prediction model regards each day as a sample, and the label is the number of days between that day and the next bubble burst. Since we cannot generate samples with different labels like bubble detection for each day, we must use Calls from different days to jointly train the prediction model. However, the number of Calls contained in different days is different. More fundamental is the difference in the combination of (K, T), where K denotes the strike and T denotes the maturity. To achieve consistent data dimensions and meanings within each sample, we propose a normalization strategy using SVI models. By fitting an SVI model to the volatility surface for each trading day, we can compute Call prices for a set of fixed (K, T) combinations. This approach harmonizes the shape of each sample and aligns the meaning of the corresponding position in each sample.

Actually, we do not calculate Call prices in the implementation. The parameters derived from the SVI model fit for each day can also serve as the input. They uniquely define the volatility surfaces, which, in turn, determine the fixed (K, T) prices for each trading day. We can streamline the process by bypassing explicit pricing calculations by directly utilizing the parameters obtained from the SVI model as inputs to our neural network. This expedited approach enhances the efficiency of preparing training datasets.

In Section 3.2, we discuss the theoretical basis of the SVI model and the fitting process. In Section 3.3, we show some calibration examples.

3.1 Introduction

Uncertainty and volatility in financial markets have always been the core concerns of investors and practitioners. In this highly dynamic environment, effective volatility models are critical for option pricing, risk management, and trading decisions. Implied volatility is an important measure of expected market volatility, so developing accurate volatility models is critical to revealing market behavior and price trends.

The stochastic volatility inspired (SVI) was first introduced by scholars such as Gatheral [20], and the Arbitrage-free SVI Volatility surface (SSVI) was introduced by Gatheral and Jacquier [3]. It has attracted widespread attention and interest in the financial field. Unlike traditional volatility models, the Arbitrage-Free SVI model highlights its risk-free arbitrage characteristics, aiming to ensure that no risk-free arbitrage opportunities will appear in the model under market conditions. In recent years, this model has been used by more and more researchers and practitioners in option pricing, volatility forecasting, and risk management to provide more accurate market analysis tools and decision support.

In this chapter, we exhibit a class of SVI volatility surfaces with a simple closed-form representation, for which the absence of static arbitrage is guaranteed. Arbitrage-free SVI is fitted by each single day's option data. In fact, the option data information of each day is converted into a set of SVI parameter values. It is equivalent to a way of extracting and structuring information. We will use these parameters as the input data of the neural network in Chapter 4 to predict financial bubbles.

3.2 Theoretical basis

In this part, we introduce the theoretical basis of SSVI. Gatheral and Jacquier's article [3] has a complete theoretical explanation of SSVI. We only focus on the part that relates to the SSVI model fitting.

3.2.1 Setups and notations

Let us denote a stock price process $(S_t)_{t\geq 0}$ with natural filtration $(\mathcal{F}_t)_{t\geq 0}$. Observing under the risk-neutral measure $\mathbb{Q}$, the price dynamics in the real equity market are assumed to follow the Geometric Brownian Motion:

$$\mathrm{d}S_t = (r-q)\, S_t \mathrm{d}t + \sigma S_t \mathrm{d}W_t$$

where r is the risk-free rate, q is the dividend yield, and σ is the volatility of the underlying stock. In practice, we employ the returns of the U.S. 3-month Treasury Bill to represent the risk-free rate. The process $(W_t)_{t\geq 0}$ is a Brownian Motion with independent normal increments.

Under this price setting, we will find the Black-Scholes implied volatility by solving the Black-Scholes formula inversely. Let us denote K as the strike price. Always standing at time 0, with the time to expiration t, the vanilla Call has a payoff function equal to $(S_t - K)_+$. Therefore, by the classic Black-Scholes model, the time t price of the vanilla Call option with strike K is computed by the below expression

$$\begin{aligned} C(S_0, K) &= S_0 \mathrm{e}^{-qt}\mathcal{N}(d_1) - K\mathrm{e}^{-rt}\mathcal{N}(d_2), \\ d_1 &= \frac{\ln\left(\frac{S}{K}\right) + (r - q + \frac{\sigma^2}{2})T}{\sigma\sqrt{t}}, \\ d_2 &= d_1 - \sigma\sqrt{t} \end{aligned}$$

We define a new parameter k to represent the log-moneyness of a European Call as

$$k = \log\left(\frac{K}{S_0 \mathrm{e}^{(r-q)t}}\right). \tag{3.2.1}$$

It is obvious that if $k = 0$, the Call option is at the money under the measure-$\mathbb{Q}$ expectation, where $S_0\mathrm{e}^{(r-q)t}$ is just the forward price of the stock at time t. Therefore we can transform the Call price $C(K,t)$ dependent to K and t into $C(k,t)$.

The Black-Scholes implied volatility is the solution to the function $C_{market} = C(k,t)$. Denote this solution as $\sigma_{B.S.}$. We further combine the information of implied volatility with time to maturity. This is how we define the market total implied variance:

$$w_{market}(k,t) = \sigma_{BS}^2(k,t)\, t. \tag{3.2.2}$$

Indeed, the implied volatility, implied variance, and total implied variance are equivalent. However, $w(k,t)$ is essential for describing the volatility surface.

The volatility surface refers to a two-dimensional mapping (k,t) to $w(k,t)$. For any fixed expiration $t > 0$, the function k to $w(k,t)$ will represent a slice. We present two distinct but equivalent parameterizations of the total implied variance for slices, providing an exact correspondence between them. When referring to a maturity slice, we will use the notation $w(k;\mathcal{X})$, where $\mathcal{X}$ represents a set of parameters while omitting the dependence on t.

3.2.2 SVI formulations and the relationship between SVI and SSVI

In this section, we discuss the raw SVI parameterization introduced by Gatheral [20, Page 6], which is very tractable and has become popular with practitioners. However, it is difficult for the raw model to find precise conditions on parameters to prevent arbitrage. So we further discuss the SVI Jump-Wings(SVI-JW) and SSVI parameterization introduced by Gatheral and Jacquier's article [3]. In the process of introducing them one by one, we will also clarify the relationship between them.

Definition 3.2.1 (The raw SVI parameterization)**.** For a given parameter set $\mathcal{X}_R = \{a, b, \rho, m, \sigma\}$, the raw SVI parameterization of total implied variance is defined as

$$w_{raw}(k;\mathcal{X}_R) = a + b\left\{\rho(k-m) + \sqrt{(k-m)^2 + \sigma^2}\right\},$$

where $a \in \mathbb{R}$, $b \geq 0$,$|\rho| < 1$, $m \in \mathbb{R}$, $\sigma > 0$, and $a = b\sigma\sqrt{1-\rho^2} \geq 0$, which ensures that $w_{raw}(k; \mathcal{X}_R) \geq 0$ for all $k \in \mathbb{R}$. This condition ensures the non-negativity of the minimum value of the function $w_{raw}(\cdot; \mathcal{X}_R)$. Additionally, it is important to note that the function k to $w_{raw}(k; \mathcal{X}_R)$ exhibits (strict) convexity across the entire real line. Consequently, modifications in the parameters yield the following outcomes:

- Increasing a raises the overall variance level, a vertical translation of the smile.
- Increasing b raises the slopes of both the Put and Call wings, leading to a narrower smile.
- Increasing ρ decreases the slope of the left wing (right wing), resulting in a counter-clockwise rotation of the smile.
- Increasing m shifts the smile to the right.
- Increasing σ diminishes the at-the-money (ATM) curvature of the smile.

Notably, we exclude the trivial cases where ρ equals 1 or -1, resulting in strictly increasing and decreasing volatility smiles, respectively. Similarly, the case of $\sigma = 0$ corresponds to a linear smile and is also excluded.

The raw SVI parameterization is capable of outputting the total implied variance of the time to expiration t. However, it isn't easy to interpret the realized meaning of the raw SVI parameters. In addition, without involving t, these raw parameters may not be as stable as expected. Therefore, we incorporate time information into the raw parameterization to obtain SVI Jump-Wings.

Definition 3.2.2. [The SVI-JW parameterization] For a given time to expiration $t > 0$ and a parameter set $\mathcal{X}_J = \{v_t, \psi_t, p_t, c_t, \widetilde{v}_t\}$ the SVI-JW parameters are defined from the raw SVI parameters as

$$\begin{aligned} v_t &= \frac{a+b\left\{-rhom+\sqrt{m^2+\sigma^2}\right\}}{t}, \\ \psi_t &= \frac{b}{2\sqrt{w_t}}\left(-\frac{m}{\sqrt{m^2+\sigma^2}}+\rho\right), \\ p_t &= \frac{b}{\sqrt{w_t}}\left(1-\rho\right), \\ c_t &= \frac{b}{\sqrt{w_t}}\left(1+\rho\right), \\ \widetilde{v_t} &= \frac{1}{t}\left(a+b\sigma\sqrt{1-\rho^2}\right). \end{aligned}$$

These parameters are dependent on the time to expiration and have the following financial implications:

- v_t represents the at-the-money variance.
- ψ_t represents the at-the-money skew.
- p_t represents the left wing's slope.
- c_t represents the right wing's slope.
- $\widetilde{v_t}$ is the minimum implied variance.

SVI-JW is a well-interpreted formulation that allows us to directly involve the arbitrage-free condition in the relationships between its parameters.

Proposition 3.2.3. *[Arbitrage-free condition in SVI-JW] Assume that the first three parameters of SVI-JW parameterization v_t, ψ_t, p_t are fixed. A volatility smile is guaranteed without arbitrage if we choose the other two parameters by*

$$\hat{c_t} = p_t + 2\psi_t, \quad \hat{\widetilde{v}_t} = \frac{4v_t p_t \hat{c_t}}{(p_t + \hat{c_t})^2}$$

This proposition further emphasizes the significance of SVI-JW in a direct guarantee of free butterfly arbitrage from the perspective of parameter redefinition. We utilize this proposition in the SSVI fitting to constrain the volatility slices always free of arbitrage.

Definition 3.2.2 showcases that the SVI-JW parameterization comes from the raw formulation of SVI. Indeed, we also need a way to transform SVI-JW back to its corresponding raw form since the raw parameterization captures the total implied variance without the time to maturity t being a variable.

Theorem 3.2.4 (Convert parameters from SVI-JW to raw)**.** *Define β and α as*

$$\beta := \rho - \frac{2\psi\sqrt{w_t}}{b} \quad \textit{and} \quad \alpha := sign\,(\beta)\sqrt{\frac{1}{\beta^2} - 1}.$$

where $\beta \in [-1, 1]$. Then, SVI-JW parameters can be converted into raw parameters by:

$$\begin{aligned} b &= \tfrac{\sqrt{w}}{2}\left(c_t + p_t\right), \\ \rho &= 1 - \tfrac{p_t\sqrt{w_t}}{b}, \\ a &= \widetilde{v_t}t - b\sigma\sqrt{1-\rho^2}, \\ m &= \tfrac{(v_t - \widetilde{v_t})t}{b\left\{-\rho + sign(\alpha)\sqrt{1+\alpha^2} - \alpha\sqrt{1-\rho^2}\right\}}, \\ \sigma &= \alpha m. \end{aligned}$$

After, we give two important formulations of the SVI, being slices of the volatility surface. Before modeling the volatility surface, there is a crucial concept, the at-the-money (ATM) implied total variance, denoted as $\theta_t := \sigma_{BS}^2\left(0, t\right)t$. The ATM point is the center of the volatility smile curve. Therefore, the ATM implied total variance is used as a calibration target. By calibrating the model to match the ATM implied total variance, the SSVI model can better capture the dynamics of the volatility skew. Now, we are ready to give a definition of the volatility surface with a Heston-like function smoothing θ_t.

Definition 3.2.5 (Surface SVI with a Heston-like smoothing function)**.** Let ϕ be a smooth map from $\mathbb{R}_+^* \to \mathbb{R}_+^*$ following a Heston-like parameterization

$$\phi\left(\theta\right) = \frac{1}{\lambda\theta}\left(1 - \frac{1 - \mathrm{e}^{-\lambda\theta}}{\lambda\theta}\right) \tag{3.2.3}$$

where $\lambda > 0$. SSVI is thus referred to as a volatility surface with an expression

$$w_{SSVI}(k, \theta_t) = \frac{\theta_t}{2}\left\{1 + \rho\phi\left(\theta_t\right)k + \sqrt{\left(\phi(\theta_t)k + \rho\right)^2 + \left(1 - \rho^2\right)}\right\} \tag{3.2.4}$$

Note that there are, in total, two parameters ρ and λ in this Heston-like-SSVI. The ATM implied total variance θ_t is not actually a parameter since we observe it from the market. If there is a direct at-the-money option, we can choose its value without extra effort. Otherwise, we need to interpolate the closest options to find the value of θ_t.

We now give the relationship between our SSVI and the previous SVI slices.

Proposition 3.2.6. *The SVI-JW are equivalent with the SSVI in the sense of parameters*

$$v_t = \frac{\theta_t}{t}, \quad \psi_t = \frac{1}{2}\rho\sqrt{\theta_t}\phi(\theta_t), \quad p_t = \frac{1}{2}\sqrt{\theta_t}\phi(\theta_t)(1-\rho)$$

$$c_t = \frac{1}{2}\sqrt{\theta_t}\phi(\theta_t)(1+\rho), \quad \widetilde{v_t} = \frac{\theta_t}{t}(1-\rho^2)$$

This proposition is important since we are capable of transforming the surface information into slices. Recall that the SVI-JW is advanced in guaranteeing the no-arbitrage condition in its parameters (see Proposition 3.2.3). Hence, it is reasonable to keep the first three parameters transformed from the SSVI and calculate the rest in the SVI-JW by no-arbitrage conditions.

Since we identify the smoothing function for θ_t to be a specific Heston-like form, we can give a constraint for the volatility surface to be free of arbitrage to some extent.

Proposition 3.2.7. *If $\lambda \geq \left(1 + |\rho|\right)/4$, the SSVI can be free of static arbitrage up to some maximum expiry.*

This single condition is not sufficient to make the whole SSVI free of arbitrage. The four conditions refer to [3, Corollary 4.1, page 16]. Nevertheless, this condition is enough for us to constrain the SSVI fitting to obtain a first-step set of parameters. Fine-tuning is necessary in the following fitting process.

3.2.3 SSVI and SVI fitting process and algorithm

Currently, the more popular codes on the SSVI model on the Internet are codes implemented by Jim Gatheral using Matlab. This section shows the process and algorithm of using Call option data to fit the SSVI model in Python.

Step 1: Prepare data

Fitting the SSVI model begins with preparing the raw dataset.

- Use dividend and close price S to calculate divident yield p by $p = \frac{\text{Annual dividend}}{\text{S}} 100\%$.
- The forward price F of the underlying asset is calculated using the close price S, interest rate r, dividend yield p and maturity t.

  $$F = S\mathrm{e}^{(r-q)t}$$

  This forward price F represents the expected future price of the asset, considering market conditions.
- Non-usable data is removed from the dataset to ensure the quality of the analysis. This includes options with non-positive bid-ask spreads, bids below a certain threshold, and options that are "in-the-money" (ITM) based on their strike prices and the forward price. The data is filtered to exclude these cases.
- The Mid price m, the average of bid and ask prices, is calculated for each option. This is often used as a representative price for further analysis.
- Log-moneyness k is computed for each option by equation 3.2.1, which represents the relative position of the option's strike price with respect to the forward price.
- Calculate implied volatility σ_{BS} under Black-Scholes model. It is a measure of the market's expectation of future volatility. Then calculate market total implied variance w_{market} by equation 3.2.2.

Step 2: Use linear interpolation for ATM total implied variance

In this step, the goal is to estimate the total implied variance at the at-the-money (ATM) point θ for various maturities. The total implied variance captures the market's expectation of the future variance in the asset's price over the option's maturity period. Linear interpolation is employed to estimate the total implied variance at the exact ATM point, as it may not be available in the original data set.

The process starts by identifying unique maturity values from the options data. These maturities represent different expiration dates for the options. The amount of unique maturity values is denoted by T. Iterate over each unique maturity value t_i for $i \in \{1, ..., T\}$ to calculate ATM total implied variance $\theta(t_i)$.

if the log-moneyness k for the current maturity t_i contain the ATM point ($k = 0$). If the ATM point exists in the data, the corresponding total implied variance $\theta(t_i)$ is directly assigned to w. If the ATM point is not available in the data, linear interpolation is performed. Linear interpolation estimates the total implied variance at the ATM point $\theta(t_i)$ using nearby log-moneyness values and their corresponding total implied variance values. This approximates the total implied variance at the exact ATM point.

This step ensures that the total implied variance is estimated at the ATM point for all maturities, even when the exact ATM point is not present in the original data. Using linear interpolation allows a smooth estimation of total implied variance at the ATM point, enhancing the accuracy of the volatility surface modeling for different maturity periods.

Step 3: Fit SVI surface by estimating parameters ρ and λ

In this step, the goal is to fit the SVI surface to the given data by estimating the parameters ρ in equation 3.2.4 and λ in equation 3.2.3. The SVI surface is used to model the total implied variance as a function of log-moneyness k and ATM total implied variance $theta(t)$. The parameters are subject to specific bounds and constraints based on the nature of the SVI model.

The parameters ρ and λ are estimated while adhering to specific bounds and constraints. The parameter ρ is constrained to the range $(-1, 1)$, indicating its correlation with the volatility process. The parameter λ is constrained to be greater than 0, as it represents a scale factor. For Heston-like-SSVI, the constraint enforces $(1 + |\rho|) \leq 4\lambda$ to prevent arbitrage. Random start values are generated within the parameter bounds for optimization convergence.

The optimization problem is set up to minimize the difference between the observed market total implied variance w_{market} and the model's estimated total implied variance w_{SSVI}.

Step 4: Transforming to SVI-JW parameters and fitting

In this step, the SVI parameters are transformed to SVI-JW parameters and then iteratively fit the SVI-JW data for each maturity.

We take the fitted SSVI model in step 3 as the initial guess and then fit SVI slice-by-slice with a heavy penalty for certain arbitrage conditions. A more detailed process is as follows.

- Transformation and Initialization: Based on the selected SVI model(Heston-like), the SSVI parameters are transformed to SVI-JW parameters using Proposition 3.2.6. Thus, the initial values of SVI-JW parameters are set.
- Iterative Fitting for Each Maturity: A loop is initiated to iterate through each unique maturity value (denoted by t) in reverse order. This is the so-called fitting SVI slice by slice.
- Depending on the current iteration t, SVI-JW slices are constructed. We construct the before slice, the after slice, and the current slice in each iteration.
- Change SVI parameters slice-by-slice so as to minimize the sum of squared distances between the fitted total implied variance w_{SVI_JW} and market total variance w_{market} with a big penalty for crossing either the previous slice or the next slice to prevent calendar arbitrage. In particular, it should be noted that in this optimization process, our optimization goal has three parameters v, ψ, and p.
- To prevent butterfly arbitrage, use Proposition 3.2.3 to calculate c and $\widetilde{v_t}$.

Fitting slice by slice is a more abstract process. To demonstrate the fitting process clearly, we present some related code in Appendix A.1.

3.3 Calibration examples

We take Nvidia, Apple, and Tesla option quotes on Sep 2, 2021. The result of fitting SVI following the Section 3.2.3 is shown in Figure 3.1, Figure 3.2 and Figure 3.3. Except for a few expiration dates, the fit quality is almost perfect.

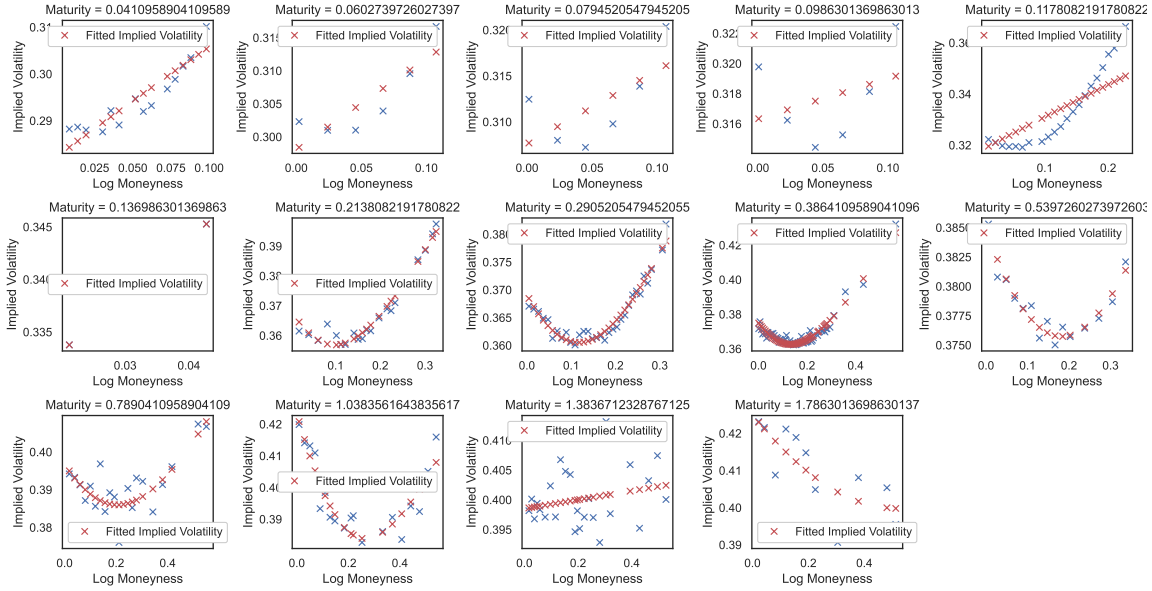


Figure 3.1: SSVI fitting results of Nvidia on 2021-10-06: blue dots are mid implied volatilities, and red dots are the SVI fits following the Section 3.2.3.

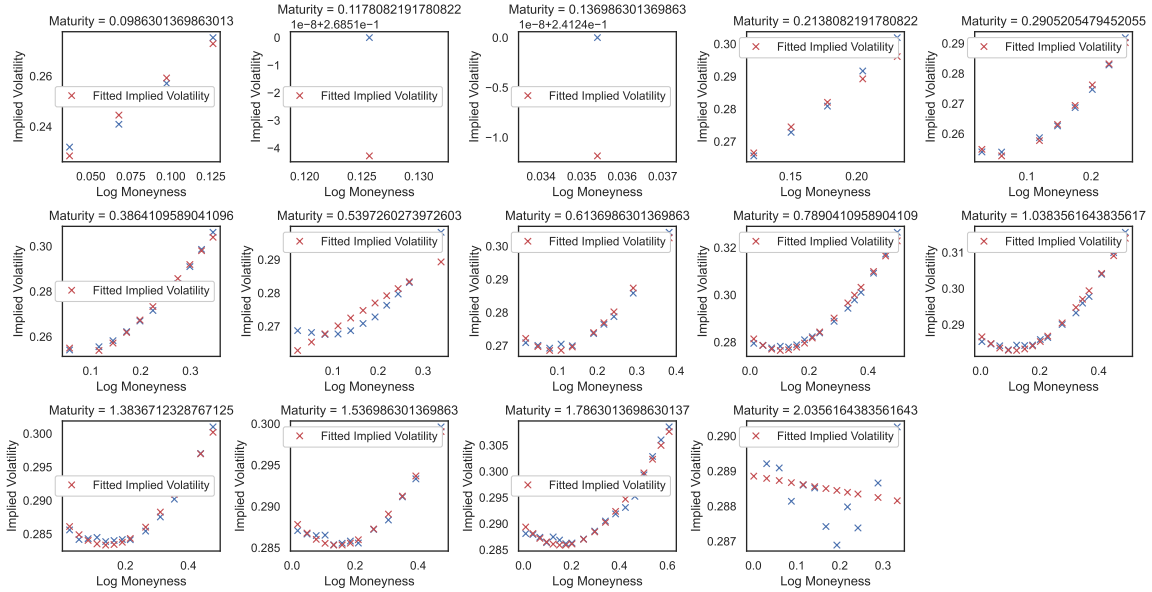


Figure 3.2: SSVI fitting results of Apple on 2021-10-06: blue dots are mid implied volatilities, and red dots are the SVI fits following the Section 3.2.3.

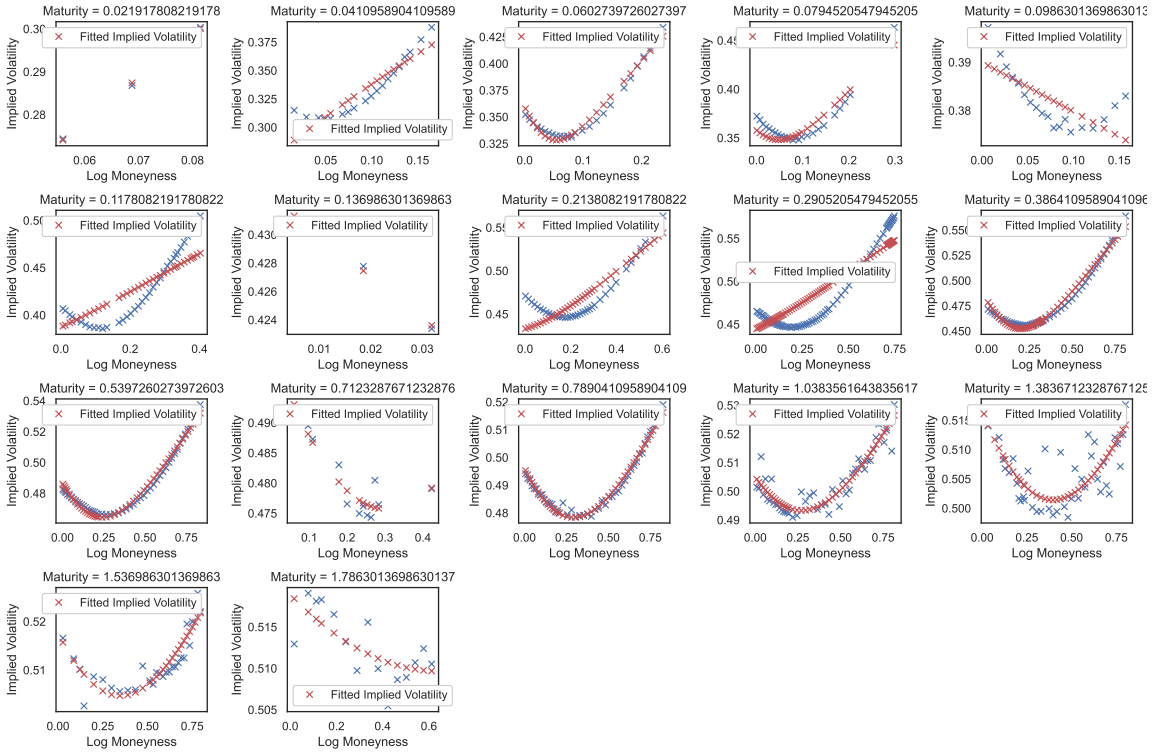


Figure 3.3: SSVI fitting results of Tesla on 2021-10-06: blue dots are mid implied volatilities, and red dots are the SVI fits following the Section 3.2.3.

Chapter 4

Bubble prediction

The bubble prediction model aims to use European Call data to predict the next date a bubble burst appears. We construct a neural network to accomplish prediction. For each day, the parameters of the fitted SVI for each day are used to represent the information of Calls and are regarded as the input, and the output is supposed to be a float number, meaning the number of days after which the bubble will burst.

The prediction model does not involve complex mathematical theories but focuses on the processing of data sets and the setting of neural networks. In this chapter, we introduce our data preparation and neural network settings and show calibration results.

4.1 Data preparation

After we have detect a financial bubble by following Section 2.3, we will search for the burst date of this bubble, which is simply defined as the highest price of the asset. This is because we have observed that after the highest price of an asset, there is often a significant decline. Then, we label each date before the burst date using the number of days between the day and the burst.

Then, we set up the input dataset. Since the parameters of the fitted SVI model consist of most of the information of Calls, as we introduced in Chapter 3, we build the input based on these parameters. Notably, for each day, the parameters calculated by following the Section 3.2 is a $\mathrm{T}\times 5$ matrix, where T is the number of unique maturity values of that day, and 5 represents the five parameters of the SVI-JW model. This corresponds to Definition 3.2.2, in which the parameters of the SVI-JV model depend on the time to expiration. Since T has different values for different days, We need to use the interpolation to calculate the parameters of the SVI-JW model under some fixed time to expiration. We set nine points using the interpolation method $\mathrm{T} = \{0.02, 0.05, 0.1, 0.2, 0.4, 0.8, 1.2, 1.5, 1.8\}$. After interpolation, each trading day of each asset will correspond to a 9×5 parameter matrix.

We focus on the same dataset as in Subsection 2.3.3. There are 372 samples in total (126 for each asset), divided into training set, validation set and test set according to the ratio of 6:2:2.

4.2 Neural network settings

The 200 generated samples are divided into training set, validation set and test set according to the ratio of 6:2:2. The neural network has two hidden layers, each with 32 nodes and activated by the ReLU function. The output layer comprises a single node with a sigmoid activation function. The sigmoid activation function outputs values in the range (0, 1), which is suitable for binary classification tasks like the one being undertaken here.

The neural network has three hidden layers, with 64, 32 and 16 nodes, respectively, activated by the ReLU function. The output layer comprises a single node with a ReLU activation function, which is suitable for generating a positive float number.

The batch size is 25, and the learning rate is 0.01. The epoch is set as 2000 while the patience of the earlystopping is 50. The loss function is MSE, which is appropriate for regression tasks.

4.3 Calibration example

Train loss	Validation loss	Test loss
594.59	973.68	1310.1

Table 4.1: Evaluation of the bubble prediction model.

Table 4.3 highlights the occurrence of overfitting during the model fitting process. One possible explanation for this phenomenon is the limited sample size. This limitation arises from the constraints on accessing options data in the market. While the test set error appears notably high, it is crucial to recognize that this outcome primarily stems from a few predictions exhibiting significant discrepancies. In reality, the model demonstrates a reasonable predictive capacity for the majority of samples. Subsequent analyses will shed light on the model's predictive competence by presenting additional results.

There are a total of 76 samples in the test set. We generated 76 random numbers between 0 and 180 from the uniform distribution as a comparison with our prediction results to reflect the prediction ability of our model.

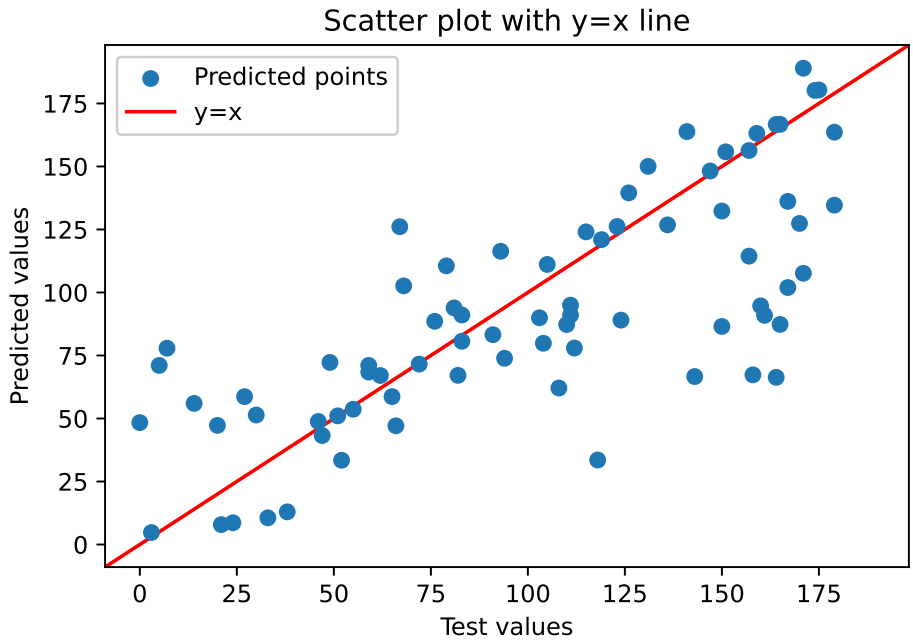


Figure 4.1: Predicted values from the bubble prediction model against real test values.

Figure 4.1 and Figure 4.2 show the predicted value and randomly generated value for the test dataset. It is evident that predicted points are closer to the straight line of $y = x$, indicating that the prediction results of the model are closer to the actual value than the randomly generated results.

The day difference is defined as

$$|\text{predicted or random values} - \text{true values of test}|$$

day difference	below 10	below 20	below 30	below 40
predicted values	25	52	51	57
random values	8	18	23	29

Table 4.2: Day difference between predicted/random values and true values.

Tabel 4.3 presents the count of days by which randomly generated results and the model's prediction deviate from the true values across varying thresholds. Under each threshold, the day

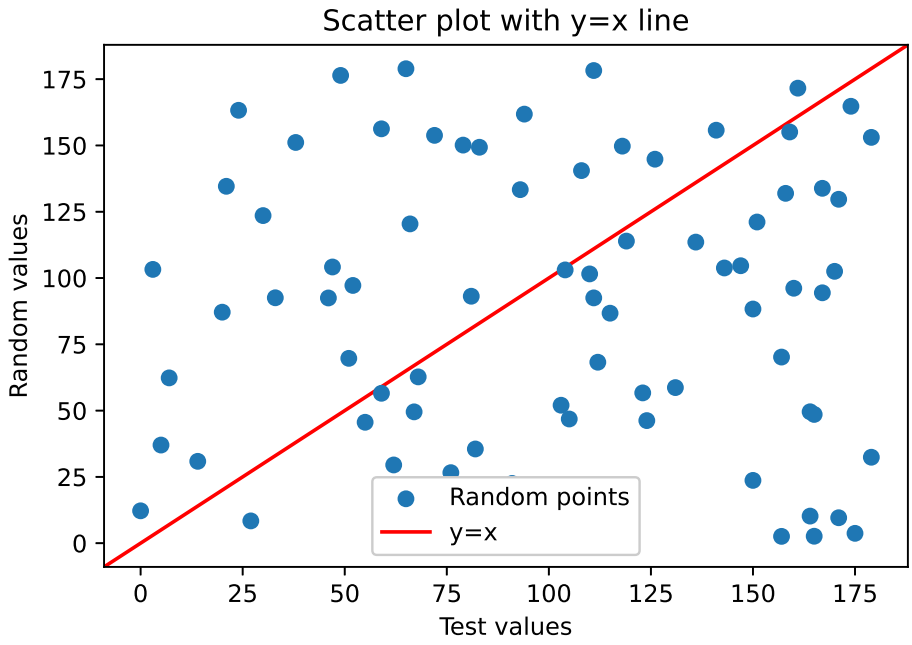


Figure 4.2: Random values against real test values.

differences of the prediction model are at least twice that of the generated values, demonstrating the model's ability to predict the date of the bubble burst.

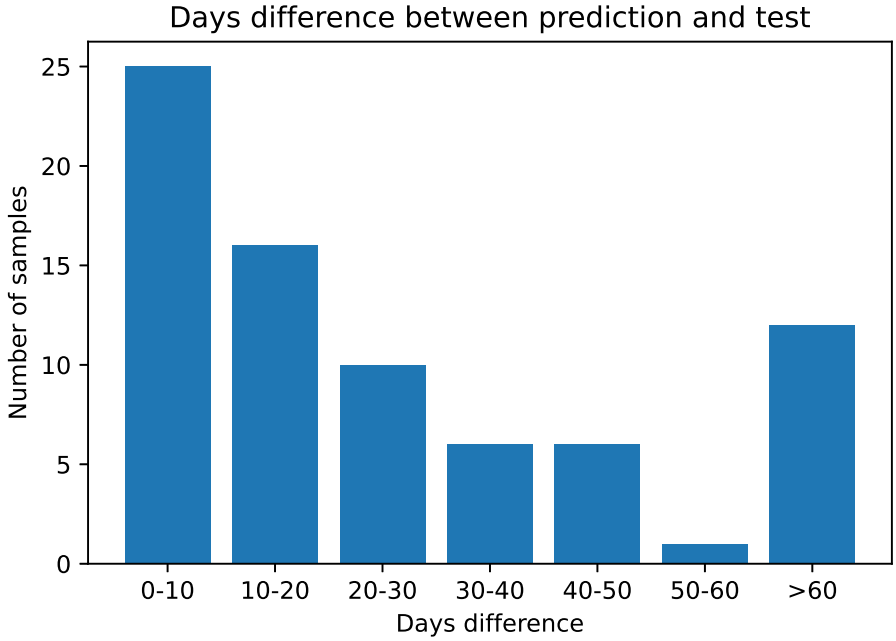


Figure 4.3: The day difference between prediction and test.

More specifically, Figure 4.3 and Figure 4.4 show the number of days differences within different ranges. Most of the day differences between the model's prediction and true values are below 30 days, and the number of samples in the range $(0, 10)$ is the largest. In contrast, most generated values have more than 60 days of differences from actual values. Hence, our bubble prediction model has an excellent effect on predicting the appearance date of bubbles.

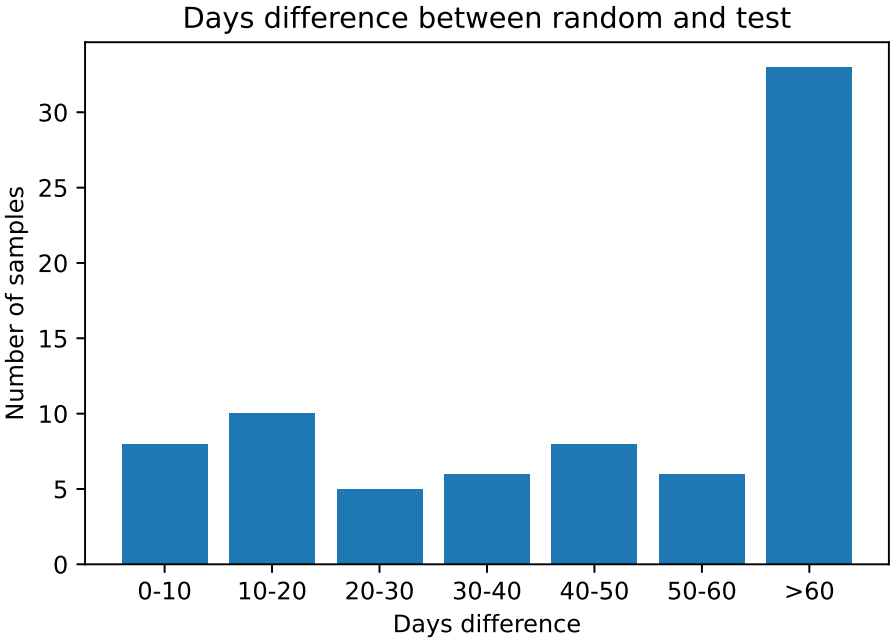


Figure 4.4: The day difference between random and test.

Conclusion

This article can be seen as an extension to the detection model from Biagini et al. [2]. In this article, we propose a bubble prediction model based on deep learning methodology. Once the detection model detects an asset being in a bubble, this trained neural network is then used to predict the date of the upcoming bubble burst. The deep neural network takes as input parameters of the SVI model fitted by daily European Calls for a given underlying asset and returns as output a positive float number, which represents the number of days after which the bubble will burst.

We first reproduce the bubble detection model from Biagini et al. and get the same calibration results. After finding the bubble using the bubble detection model, the highest point of the stock price during that period is used as the bubble burst time. Then, the European Calls before the bubble burst were used to fit the SVI model slice by slice, and a good volatility surface was obtained. Then, the parameters of SVI are used as input to train the neural network to predict the time when a bubble burst occurs.

We evaluated the efficacy of our prediction model by contrasting its results with those generated randomly. The outcomes from our model consistently outperformed the random results in various aspects, underscoring its robust capability in forecasting the onset of a bubble burst.

A limitation of this study is our restricted access to extensive option data, which led to certain overfitting tendencies in our model. For future enhancements, acquiring a larger dataset of options or employing techniques like Variational Autoencoders (VAE) and Generative Adversarial Networks (GAN) to generate a more diverse sample pool would be beneficial.

Appendix A

Complementary Python code

A.1 Fit SVI by slices

```
def fit_svi(x0, k, total_implied_variance, slice_before, slice_after, tau):
    # Define the bounds for variables
    large = 1e5
    small = 1e-6
    lb = [small, -large, small, small, small]
    ub = [large, large, large, large, large]

    def targetfun(x):
        return fitfunction(x, k, total_implied_variance, slice_before, slice_after,
     tau)

    # Only optimize first three variables, final two are set by no-arbitrage
    condition
    x0 = x0[:3]
    lb = lb[:3]
    ub = ub[:3]

    # Constraints
    constraints = {'type': 'ineq', 'fun': lambda x: np.array([2*x[1] + x[2]])}

    result = minimize(targetfun, x0, method='SLSQP', bounds=list(zip(lb, ub)),
    constraints=constraints)

    return result.x

def fitfunction(x, k, total_implied_variance, slice_before, slice_after, tau):
    x_new = np.zeros(5)
    x_new[:3] = x

    # Use Proposition 3.2.3 to prevent butterfly arbitrage
    x_new[3] = x[2] + 2 * x[1]
    x_new[4] = x[0] * 4 * x[2] * x_new[3] / (x[2] + x_new[3])**2
    model_total_implied_variance,_ = svi_jumpwing(k, x_new, tau)

    # Target of minimization
    value = np.linalg.norm(total_implied_variance - model_total_implied_variance)

    # Penalty for crossing
    if slice_before is not None and np.any(model_total_implied_variance <
    slice_before):
        value = 1e6
    if slice_after is not None and np.any(model_total_implied_variance >
    slice_after):
        value = 1e6

    if np.isnan(value):
        value = 1e6

    return value

def fit_by_slice(data):
```

```
    total_implied_variance = cal_total_vol(data)
    log_moneyness = data['log_moneyness']
    maturity = data['maturity']
    maturities = np.sort(np.unique(maturity))
    T = len(maturities)
    parameters = np.zeros((5, T))

    # Initial parameters for SVIJV model
    v,psi,p,c,vt = SSVI_to_SVIJW(data)

    # Fit SVIJW slice by slice
    for t in range(T-1, -1, -1):
        pos = maturity == maturities[t]
        log_moneyness_t = log_moneyness[pos]
        total_implied_variance_t = total_implied_variance[pos]

        if t == T - 1:
            param_before = [v[t-1], psi[t-1], p[t-1], c[t-1], vt[t-1]]
            slice_before,_ = svi_jumpwing(log_moneyness_t, param_before, maturities
    [t-1])
            slice_after = None
        elif t == 0:
            slice_before = None
            param_after = [v[t+1], psi[t+1], p[t+1], c[t+1], vt[t+1]]
            slice_after,_ = svi_jumpwing(log_moneyness_t, param_after, maturities[t
    +1])
        else:
            param_before = [v[t-1], psi[t-1], p[t-1], c[t-1], vt[t-1]]
            slice_before,_ = svi_jumpwing(log_moneyness_t, param_before, maturities
    [t-1])
            param_after = [v[t+1], psi[t+1], p[t+1], c[t+1], vt[t+1]]
            slice_after,_ = svi_jumpwing(log_moneyness_t, param_after, maturities[t
    +1])

        param0 = [v[t], psi[t], p[t], c[t], vt[t]]
        parameters[0:3, t] = fit_svi(param0, log_moneyness_t,
    total_implied_variance_t, slice_before, slice_after, maturities[t])

        # Use Proposition 3.2.3 to prevent butterfly arbitrage
        parameters[3, t] = parameters[2, t] + 2 * parameters[1, t]
        parameters[4, t] = parameters[0, t] * 4 * parameters[2, t] * parameters[3,
    t] / (parameters[2, t] + parameters[3, t]) ** 2

    return parameters
```

Bibliography

[1] Christian Gilles and Stephen F LeRoy. Bubbles and charges. *International Economic Review*, pages 323–339, 1992.

[2] Francesca Biagini, Lukas Gonon, Andrea Mazzon, and Thilo Meyer-Brandis. Detecting asset price bubbles using deep learning, 2022. arXiv:2210.01726.

[3] Jim Gatheral and Antoine Jacquier. Arbitrage-free svi volatility surfaces. *Quantitative Finance*, 14(1):59–71, 2014.

[4] Alexander MG Cox and David G Hobson. Local martingales, bubbles and option prices. *Finance and Stochastics*, 9:477–492, 2005.

[5] Carlos A Sin. Complications with stochastic volatility models. *Advances in Applied Probability*, 30(1):256–268, 1998.

[6] Robert A Jarrow, Philip Protter, and Kazuhiro Shimbo. Asset price bubbles in incomplete markets. *Mathematical Finance: An International Journal of Mathematics, Statistics and Financial Economics*, 20(2):145–185, 2010.

[7] Freddy Delbaen and Walter Schachermayer. A general version of the fundamental theorem of asset pricing. *Mathematische annalen*, 300(1):463–520, 1994.

[8] Jean-Pascal Ansel and Christophe Stricker. Couverture des actifs contingents et prix maximum. In *Annales de l'IHP Probabilités et statistiques*, volume 30, pages 303–315, 1994.

[9] Robert C Merton. Theory of rational option pricing. *The Bell Journal of economics and management science*, pages 141–183, 1973.

[10] Fischer Black and Myron Scholes. The pricing of options and corporate liabilities. *Journal of political economy*, 81(3):637–654, 1973.

[11] David C Emanuel and James D MacBeth. Further results on the constant elasticity of variance call option pricing model. *Journal of Financial and Quantitative Analysis*, 17(4):533–554, 1982.

[12] John Hull and Alan White. The pricing of options on assets with stochastic volatilities. *The journal of finance*, 42(2):281–300, 1987.

[13] Patrick Hagan, Andrew Lesniewski, and Diana Woodward. Probability distribution in the sabr model of stochastic volatility. In *Large deviations and asymptotic methods in finance*, pages 1–35. Springer, 2015.

[14] Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

[15] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.

[16] Geoffrey E Hinton, Simon Osindero, and Yee-Whye Teh. A fast learning algorithm for deep belief nets. *Neural computation*, 18(7):1527–1554, 2006.

[17] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436–444, 2015.

[18] Alex Graves and Alex Graves. Long short-term memory. *Supervised sequence labelling with recurrent neural networks*, pages 37–45, 2012.

[19] Lukas Gonon and Mikko Pakkanen. Deep learning lecture notes. 2022.

[20] Jim Gatheral. A parsimonious arbitrage-free implied volatility parameterization with application to the valuation of volatility derivatives. *Presentation at Global Derivatives & Risk Management, Madrid*, 2004.