DISS. ETH NO. 28442

TWO PERSPECTIVES ON CONSISTENT MODELLING OF OPTION MARKETS

A thesis submitted to attain the degree of
DOCTOR OF SCIENCES
(Dr. sc. ETH Zurich)

presented by
MATTEO GAMBARA

MSc. Math. Swiss Federal Institute of Technology Lausanne

born on November 8, 1991

accepted on the recommendation of
Prof. Dr. Josef Teichmann examiner
Prof. Dr. Sergey Nadtochiy co-examiner

2022

To my family, who has always supported me.
I know I am lucky to have you.

And to Prof. Giampiero Spiga, who unexpectedly passed away this year,
my first advisor during the Bachelor. I hope I made him proud.

'Ich sage euch: man muss noch Chaos in sich haben,
um einen tanzenden Stern gebären zu können. Ich
sage euch: ihr habt noch Chaos in euch.'
– Friedrich Wilhelm Nietzsche, Also Sprach Zarathustra, 1883-1885

Abstract

In this thesis, we deal with the problem of time-consistent, arbitrage-free modelling of derivatives' prices together with their respective underlyings. This is a well-known and intriguing problem in mathematical finance due to several non-linear static and dynamic constraints, which has evident consequences in many areas of interest, from portfolio optimization to risk-management, from pricing to hedging.

The main topic of investigation is extremely general, since it concerns how to exploit incoming information available in financial markets to model, at the same time, one or more underlying assets together with the associated derivatives written on them, while respecting and maintaining through time all natural constraints. For this reason, calibration and, even before, choice of the models' pool are of central relevance and, in fact, they establish a *liaison* across the entire proposal.

In this thesis, two novel perspectives on such time-consistent modelling are examined with the assistance of modern machine learning algorithms, in particular, deep learning, which is nowadays the state-of-the-art for new theoretical and practical breakthroughs in many fields of science and engineering, finance included. For consistent recalibration models as well as for Bayesian averaging models, we provide theoretical foundations and well-working high-level implementations in Python.

The first issue that we deal with involves model re-calibration: this is a rather common problem in the financial industry that is often ignored or overlooked because of the lack of meaningful and efficient solutions and because of robustness of non-consistent solutions. In brief, every day a model is chosen by calibration to the prevailing market conditions; thereupon, pricing, hedging, and other risk-management decisions are taken based on the selected model. Inconsistencies start to arise when new information becomes available in the market and the calibrated model is not able to include this information any longer. The problem is even more compelling when decisions that were taken under the calibrated model are still in force. Historically, there have been two different approaches to solve this issue: on one hand, the so-called factor models aims at expressing the dynamics of the underlying with complex models that should be able to reflect its evolution together with its derivatives in time; on the other hand, market models try to incorporate derivatives' prices as state variable. We will investigate another approach, namely consistent recalibration models, that tries to tackle the problem by modelling of a codebook, a map between certain parameters, the underlyings and their derivatives, exploiting inherent redundancies of the codebook to compensate for parameters evolution. In such a way, we will manage to obtain an infinite dimensional model that is able to consistently satisfy no-arbitrage constraints and that is locally appearing as a factor model, thus keeping some degrees of analytical tractability. Since this approach is empowered by a HJM framework, well-known spot and drift conditions need to remain valid over time. One of our achievements is to take advantage of deep learning technology to integrate such conditions while being able to (partially) invert the map between model parameters and model prices, which is known to be an ill-conditioned problem.

The second challenge we face is rather related to pricing and hedging in the financial market simulating a real-world scenario, thus in incomplete settings, by allowing transaction fees and other market constraints (e.g. liquidity restrictions, short selling, etc. . .). The starting point is the algorithm known as Deep Hedging, which allows to efficiently find an optimal hedging strategy in very general environments, leveraging on the trajectories of a specific model. The question is again: how can we include incoming market information efficiently and base risk management decisions on it. Our proposal is to overcome this main restriction of deep hedging, namely that a model is fixed a priori, by employing Bayesian techniques in the sense of 'Estimate Nothing' by Dümbgen and Rogers (2014), to reach a market-consistent artificial hedgers along changing environments. In order to consider such a general setting, we resort to a setting of robust finance and we find in Bayesian updating the ideal procedure to assimilate incoming market information. The outcome of such update will be a posterior distribution over potential models that is *not* used to select one particular model, as it would be usual, but to average strategies over all present models of the pool. While this is a very effective technique, it normally suffers from numerical complexity, leading to over-simplified pools of models.

In this sense, it is not only Deep Hedging profiting from Estimate Nothing, but also the latter gaining from the former, since neural networks can learn exceptionally complex maps whose subsequent evaluation is basically instantaneous. In this second case, hence, time-consistency is granted in the form of a trading strategy which is able to include a remarkably general setting by Bayesian updating and averaging, and which can finally be trained on available real world data.

Sommario

In questa tesi, trattiamo il problema riguardante la modellizzazione coerente nel tempo (*time-consistent*) dei prezzi dei derivati e dei loro sottostanti. Questo è un problema noto e non banale di matematica finanziaria a seguito di numerosi vincoli non lineari, sia statici che dinamici, che ha risvolti evidenti in molte aree d'interesse: dall'ottimizzazione di portafoglio al risk-menagement, dal pricing (prezzaggio) all'hedging (copertura).

Il principale tema d'analisi è estremamente generale dato che riguarda come utilizzare informazioni in entrata disponibili nei mercati finanziari per modellizzare, allo stesso tempo, uno o più sottostanti e i relativi derivati scritti su di essi, rispettando e conservando nel tempo i diversi vincoli che si richiedono necessari. Per questa ragione, la calibrazione e, ancora prima, la scelta del modello sono di fondamentale importanza e, di fatto, stabiliscono una *liaison* atrraverso tutta l'intero progetto.

In questa tesi, due nuove prospettive su tale modellizzazione *time-consistent* sono esaminate con l'ausilio di algoritmi di apprendimento automatico (machine learining), in particolare, di (apprendimento profondo) deep learning, che sono attualmente tecniche d'avanguardia (lo stato dell'arte) per la risoluzione o il progresso in problemi pratici e teorici concernenti diversi campi della scienza e dell'ingengneria, incluso quello finanziario. Forniamo, nel corso della tesi, modelli per la ricalibrazione coerente e per una media Bayesiana che sono teoricamente ben fondati, dimostrandone la validità attraverso un'implementazione nel linguaggio di programmazione Python.

Il primo problema affrontato in questa tesi rigurda la 'ri-calibrazionne del modello'. Questo è un problema piuttosto comune nel settore finanziario ed è spesso ignorato a causa della mancanza di soluzioni matematicamente fondate ed efficienti e a causa della robustezza di soluzioni non coerenti. In breve, ogni giorno un modello viene scelto attraverso la sua calibrazione alle predominanti condizioni di mercato ed il pricing, l'hedging ed altre decisioni legate alla gestione del rischio vengono quindi prese sulla base del modello selzionato. Delle inconsistenze cominciano a sorgere quando nuove informazioni diventano disponibili nel mercato e il modello calibrato non risulta più in grado di rappresentare il contesto di lavoro. Inoltre, il fatto che alcune decisioni, prese durante il periodo di validità del modello calibrato, risultino comunque attive anche dopo l'arrivo di nuove informazioni rende ancora più impellente la risoluzione del problema. Storicamente, due diversi approcci sono stati utilizzati per risolvere tale complicazione: da un lato, i cosiddetti *factor models* che descrivono la dinamica del sottostante con modelli complessi che dovrebbereo essere in grado di rifllettere la sua evoluzione nel tempo; dall'altro lato, invece, i market models che provano ad incorporare i prezzi dei derivati come variabili di stato. Noi investigeremo un altro approccio, basato sui consistent recalibration models, che prova ad affrontare il problema attraverso la modellizzazione di un codebook, una biiezione tra i sottostanti ed i corrispondenti derivati, sfruttado le ridondanze nel modello per compensare l'evoluzione dei parametri. In questo modo, riusciamo ad ottenere un modello di dimensione infinita in grado di rispettare

consistentemente i vincoli di non arbitraggio e che sia localmente rappresentabile come un factor model, mantenendo quindi alcuni aspetti di trattabilità analitica. Dato che questo approccio è consentito dal struttura HJM, le note condizioni di drift e spot devono rimanere valide nel tempo. Uno dei nostri risultati è quello di sfruttare la tecnologia deep learning per integrare queste condizioni ed invertire la mappa tra i parametri del modello e i prezzi, che è risaputo essere un problema matematicamente mal condizionato (e perciò di difficile risoluzione).

La seconda sfida che ci poniamo di risolvere riguarda il prezzaggio e l'hedging in mercati finanziari che simulino la realtà il più fedelmente possibile, quindi in un ambiente 'incompleto', in cui siano ammessi costi di transizione ed altri vincoli (limiti sulla liquidità o sulla vendita allo scoperto, e così via). Il punto di partenza è l'algoritmo denominato Deep Hedging, che consente di trovare una strategia di copertura ottimale in un ambiente del tutto generale, sfruttando le traiettorie di un particolare modello. La domanda è ancora come includere informazione in entrata in maniera efficiente e come basare le decisioni sulla gestione del rischio su queste. La nostra proposta è superare il più grande difetto del Deep Hedging, ovvero il dover fissare un modello a priori, utilizzando delle tecniche Bayesiane, come sviluppate da Dümbgen e Rogers 'Estiname nothing' (2014), per costruire un agente artificiale che sia in grado di trovare strategie di copertura in un ambiente in continua evoluzione. In effetti, l'aggiornamento Bayesiano è la procedura ideale per assimilare nuove informazioni in maniera efficiente, tra i vari metodi offerti in robust finance. Il risultato di tali aggiornamenti ripetuti sarà una distribuzione a posteriori sul gruppo dei modelli disponibili, ma non sarà usata per selezionarne uno, quanto piuttosto per mediare tra tutti. Questa tecnica, anche se molto efficace, è carente dal punto di vista numerico, in quanto richiede che i modelli siano facilmente calcolabili e di conseguenza, sovente, conduce ad una selezione di potenziali modelli piuttosto semplificata.

Fatte queste premesse, non solo Deep Hedging è in grado di approfittare da Estimate Nothing, ma sarà anche il secondo a beneficiare del primo siccome le reti neurali possono apprendere funzioni eccezionalmente complesse e poi calcolarle in maniera praticamente istantanea. In questo secondo caso, quindi, la coerenza nel tempo è garantita sotto forma di una strategia di trading che è in grado di operare in ambiente notevolmente complesso utilizzando l'aggiornamento e la mediazione Bayesiane e che pertanto può essere utilizzato su dati veri e propri, nella realtà.

Contents

Notation

The reader is supposed to be familiar with basic concepts from stochastic analysis and mathematical finance.
The following notations/definitions we will be using throughout the text.

- **Big $\mathcal{O}$ and big Ω notations:**
  The big $\mathcal{O}$ notation is used to describe asymptotic upper bound, while the big Ω notation provides asymptotic lower bound. Let f, g be two functions onto real numbers $\mathbb{R}$, defined on the same domain. If they are defined on an unbounded set, writing $f(x) = \mathcal{O}(g(x))$ for $x \to +\infty$ means that there exists x_0 and $M > 0$ constant such that $|f(x)| \leq Mg(x)$ for all $x \geq x_0$. The same notation can be used to describe the behaviour in the neighbourhood of a real number $z \in \mathbb{R}$. In this case, $f(x) = \mathcal{O}(g(x))$ for $x \to z$ means there exists δ and M positive constants such that $|f(x)| \leq Mg(x)$ for all x for which $0 < |x - z| \leq \delta$.
  Similarly, we have $f(x) = \Omega(g(x)) \iff g(x) = \mathcal{O}(f(x))$. The notation $f(x) \in \Omega(g(x))$ or $g(x) \in \mathcal{O}(f(x))$ could be used alternatively.

- **Big Θ notation:**
  The Θ notation used as $f(x) = \Theta(g(x))$ refers to two functions f, g with the same domain. It means that f is bounded both above and below by g asymptotically. In other words, $f(x) = \mathcal{O}(g(x))$ and $f(x) = \Omega(g(x))$. The notation $f(x) \in \Theta(g(x))$ could be used alternatively.

- **Convex set, convex function, strong convexity and subgradient:**
  Let S be a vector space over the field of real numbers. A set $C \subset S$ is called a *convex set* if the line segment between any two points of C entirely lies in C, i.e. for any $0 \leq t \leq 1$ we have $tx_1 + (1-t)x_2 \in C$ for all $x_1, x_2 \in C$. If a metric ρ is present on S, then we define the *diameter* δ of C as the $\delta_C = \sup_{x,y \in C} \rho(x, y)$ (which can of course be ∞).

  A function $f : S \to \mathbb{R}$, with $S \subseteq \mathbb{R}^d$ convex set, is said to be *convex* if for any $0 \leq t \leq 1$ and any $x_1, x_2 \in X$ we have

  $$f(tx_1 + (1-t)x_2) \leq tf(x_1) + (1-t)f(x_2).$$

  If the inequality is strict for $x_1 \neq x_2$ and $0 < t < 1$, then the function is called *strictly convex*.
  Let $\mu > 0$. A differentiable function $f : \mathbb{R}^d \to \mathbb{R}$ is said to be *μ-strongly convex* (or strongly convex with parameter μ) if for any $x, y \in \mathbb{R}^d$ we have

  $$f(y) \geq f(x) + \langle \nabla f(x), y - x \rangle + \frac{\mu}{2}\|y - x\|^2.$$

  Intuitively speaking, strong convexity means that there exists a quadratic lower bound on the growth of the function.

An equivalent condition for a differentiable function f to be strongly convex with constant $\mu > 0$ is that the function

$$g(x) := f(x) - \frac{\mu}{2}\|x\|^2$$

is convex for any x. Note that strong-convexity implies strictly convexity, which implies convexity.

We say that a vector $v \in \mathbb{R}^d$ is a *subgradient* of $f : \mathbb{R}^d \to \mathbb{R}$ at $x \in \mathbb{R}^d$ if for any $y \in \mathbb{R}^d$ we have

$$f(y) \geq f(x) + v^\top (y - x).$$

If f is convex and differentiable, then its gradient at x is a subgradient.

Chapter 1

Introduction

1.1 The modelling paradigm

In 2006 René Carmona and Michael Tehranchi wrote in the introduction of their book [CT06a] on fixed income markets (also known as interest rate derivatives' markets) that *"the level of complexity of the bond market is higher than for the equity markets: one simple reason is contained in the fact that the underlying instruments on which the derivatives are written are more sophisticated than mere shares of stock. [...] Indeed on each given day t, instead of being given by a single number S_t as the price of one share of a common stock, the term structure of interest rates is given by a curve determined by a finite discrete set of values."* A similar preamble was already given by Damir Filipović in 2001 ([Fil01]): *"Bond markets differ in one fundamental aspect from standard stock markets. While the latter are built up by a finite number of traded assets, the underlying basis of a bond market is the entire term structure of interest rates: an infinite dimensional variable which is not directly observable. On the empirical side this necessitates curve fitting methods for the daily estimation of the term structure. Pricing models on the other hand, are usually built upon stochastic factors representing the term structure in a finite dimensional state space, making them computationally tractable."*

Mathematical finance has been evolving a lot since early 2000's but the problem of time-consistent joint modelling of term structures of derivatives' prices together with their underlyings still remains intriguing. This has to do with the fact that non-linear static and dynamic constraints have to be satisfied for such a type of modelling.

The modeling paradigm predominant in the 20th century, which is built upon model choice and model evaluation with appropriate uncertainty quantification, is now challenged by machine learning approaches, which just learn desired input-output maps on data, i.e. end to end learning. It is the goal of this proposal to combine generic machine learning methods with insights from modeling to improve quality, robustness and sustainability of the machine learning approaches.

In the last fifty years fine and sophisticated technology has been developed to select and evaluate models for the purposes of risk management or portfolio choice in financial industry. This is in line with classical modeling paradigms from science and technology: choose a class of stochastic models for the dynamics of the underlyings (e.g. Black-Scholes model, Heston model, local volatility model, local stochastic volatility model, rough volatility models, etc), calibrate it to data, and calculate prices, strategies, and predictions given your preferences, see, e.g. [GH14].

Very quickly, in view of recalibration problems, a neo-classical approach has been introduced

and ameliorated: choose a class of stochastic (market) models for the underlying together with frequently observed derivatives (variance swap models, implied volatility surface models, etc) as state variables, calibrate it to data, and calculate prices, strategies, and predictions given your preferences. We name here the HJM approach commonly applied to model to interest rates and more complicated markets, see, e.g. [CN11; CN12], [KK15].
Despite its theoretical attractiveness, which converts all traded quantities, in particular the derivatives, to state variables, the analytic, geometric and numerical problems of this approach are substantial and only the arrival of machine learning technology gives new hope for efficient, industry-ready algorithms for robust finance. This will be one application of the current research proposal, where well working modeling approaches are transferred into trainable structures to obtain efficient market models.

Another strand of literature in mathematical finance, which is in line with uncertainty quantification in statistics or machine learning, is given through robust finance: choose a class of stochastic models and narrow it down through calibration to vanilla options (or other liquid instruments), but take the statistical and inverse problem significance levels seriously, then consider pricing, hedging, investing as a zero-sum game against an adversary, who chooses the worst model given some preferences, see, e.g. [Eck+21] and the references therein and, in particular, [Gie+20]. Again only recently, through the arrival of machine learning technology, efficient algorithms could be suggested to apply robust finance in concrete situations, see, e.g. [LSS21]. Bayesian techniques in the spirit of Bayesian Model Averaging, have also been suggested, for instance the innovative approach proposed in "Estimate Nothing", see [DR14], which is taken up in this proposal too: choose a class of stochastic models and calculate a posterior density according to market data (with respect to some chosen prior). Weight prices or strategies (given your preferences in each single model) according to the posterior density and *do not* choose a model.

In all cases of this list of classical methods we face the following two problems:

- Calibration: the richer market data and the more complicated the model, the more challenging the issue of calibration is.
- Model evaluation: the more sophisticated the model or problem, the more time consuming the evaluation. Often these models and problems must be dramatically simplified to be applied or to be solved in practice.

Machine Learning is shedding new light on both problems, but it is also clear that one cannot apply machine learning technology out of the box to answer all questions from Finance and Economics, since, in contrast to many standard machine learning problems, we deal with two non-standard features:

- data are limited, of non-stationary nature, noisy and often too small to make model selection significant;
- problems are potentially infinite dimensional, geometrically constrained, and formulated as a game in contrast to having a fixed loss function.

When it comes to modelling a segment of the market containing underlyings and derivatives written on them, then – mathematically speaking in view of the above insights of the classical or neo-classical point of view – we are dealing with the following problem: consider an often non-linear submanifold C of some (Banach) space $\mathcal{B}$

$$C \subset \mathcal{B}$$

which is considered the set of eligible codes. The decoder $\mathcal{D}$, which is often a non-linear function, maps codes to prices in some $\mathbb{R}^I$, where I can be an arbitrary index set. The goal is to describe stochastic processes X taking values in $\mathcal{B}$ such that

- $X_t \in C$ almost surely for all times t, i.e. C is left invariant by the evolution of X in $\mathcal{B}$ (static conditions).
- The decoder $\mathcal{D}$ maps X to a set of semimartingales allowing for an equivalent martingale measure (more generally separating measure in the sense of [CKT16] - dynamics conditions).

This mathematical problem is apparently quite delicate, in particular since usually the inverse of the decoder map $\mathcal{D}$ is – if it exists at all – difficult to calculate.

Let us introduce two additional notions, which will be pivotal in the sequel.

Definition 1.1.1 (Tangent model)**.** *Assume that $C = F \times \Theta$ has a non-trivial product structure (globally or locally) into factors taking values in some manifold F and parameters taking values in another manifold Θ.*
We call X a tangent model *if the absence of static and dynamic arbitrage is guaranteed* and *the Θ-component of X is subject to a deterministic dynamics.*

Factor models, as introduced below, will fall into this category. Usually the decoder map $\mathcal{D}$ is easy to calculate in this case (calculating its inverse is a calibration task). However, those models often are not flexible enough, i.e. re-calibration done on different days leads to different constants.

Definition 1.1.2 (Consistent recalibration model)**.** *Assume that $C = F \times \Theta$ has a product structure (globally or locally) into factors taking values in F and parameters taking values in Θ. Assume that we are given tangent models for every $\theta \in \Theta$ covering all possible instantaneous market configurations.*
Assume additionally that $\Theta = \Theta_1 \times \Theta_2$ has a non-trivial product structure, and that the decoder map is injective only if $\theta_2 \in \Theta_2$ is fixed. We call X a consistent re-calibration model *if the absence of static and dynamic arbitrage is guaranteed and the Θ_2-projection of X is Markov process.*

Consistent re-calibration models work in the following way:

- choose a dynamics of for the Θ_2 component of X,
- given this dynamics choose a dynamics of the Θ_1 component such that absence of dynamic arbitrage is guaranteed. This can be done by concatenating the given tangent model in an appropriate way.

In Chapter 3 consistent re-calibration models are introduced and put forward by means of machine learning, leveraging on the previous work made by Josef Teichmann and coauthors in [RT17], [Har+18].

In contrast to consistent re-calibration models, a second perspective, also enabled thanks to new machine learning technology, shall be introduced in Chapter 4. We have decided to name it “model free deep hedging” and it is based on a well known Bayesian approach to model selection that has been introduced by Chris Rogers and Moritz Dümbgen in Finance, see [DR14]. Apparently time series information on options as well as underlyings is included in the setup, *but* in contrast to consistent re-calibration models actually *no* model is chosen. Instead one heads

directly for risk management and takes a posterior superposition of model-based risk management decision. What we additionally add is training of this superposition and the possibility to get it going through deep hedging technology. Theoretically speaking this is an instance of [Hoe+99] or, more recently, [Ste17]. The major steps are:

a) Choose a pool of models Θ with pricing, hedging and prediction operators mapping current states and contract specifications to the relevant quantities or operations.

b) Choose a prior on Θ.

c) Choose a likelihood which compares incoming data to model quantities and update the prior according to Bayes formula.

d) After a burn-in phase the posterior is *not* used to select a model, but the posterior is rather applied as defining a model mixture. All sorts of operations are weighted with respect to this mixture. In this sense no specific model is selected.

The significant advantage of artificial traders is their robustness with respect to market frictions and with respect to input information: in contrast to classical Deep Hedging ([Bue+19]), where the scenarios are exogenously given and constitute the training data, we can also train the artificial trader for any model $\theta \in \Theta$ with an efficient training procedure. We then use the Dümbgen-Rogers approach to hedge via mixing the strategies for each θ with the posterior density. An additional training of the mixing weights is possible along time series data. We can represent it as a superposition of another distribution π, conveniently weighted by a constant ε, to the posterior $\pi_{\text{posterior}}$. The danger of over-fitting, which is inherent in case of small data sets, is reduced due to the fact that we do a regression on well trained (on a specific model) strategies.

This is an instance of transfer learning: data consist of time series data $\mathcal{T}$ and derivatives price data $\mathcal{D}$ along some past period: this leads to a posterior law $\pi_{\text{posterior}}$ on the set of models.

Then we can consider the objective function solved by model free deep hedging as follows. We generate artificial time series data for model $\theta \in \Theta$ on which we train H^θ for specific hedging tasks to obtain model specific strategies H^θ *and* we train the superposition of those model specific strategies $\int_\Theta H^\theta \pi(d\theta)$ on $\mathcal{T}$ to solve the hedging task. This can be encoded in an objective function via

$$L(\pi_{\text{posterior}} + \varepsilon\pi, H^\cdot, \mathcal{T}) + \lambda \int_\Theta P^\theta(H^\theta, \mathcal{T}^\theta_{\text{artificial}})\, \pi_{\text{posterior}}(d\theta)\,,$$

where L corresponds to the training loss on the actual data (e.g. the historically observed trajectory), and P^θ corresponds to the (possibly similar kind of) training loss on artificial data. In this sense "Estimate nothing", [DR14], appears as regularization term of an otherwise badly defined learning problem due to lack of training data.

This proposed thesis is organized as follows:

- In the remainder of the introduction, by now classical approaches to market models are presented showing their analytic beauty and complexity.
- In Chapter 2 basics of machine learning technology by means of neural networks are introduced for later use. Note that neural networks are now enjoying widespread popularity in all engineering and scientific fields, and finance is no exception (see, for example, [RW20]).
- In Chapter 3 results on consistent recalibration models are presented, in particular a high-level implementation of a full-fledged model taking underlying prices and a volatility surface

as state space input. This yields the first perspective on modelling option markets in a consistent way.

- In Chapter 4 model free deep hedging is conceptually introduced together with a high-level implementation of it in case of a pool of Merton models (Poisson jump extended Black-Merton-Scholes model). This yields a second, novel perspective on modelling option markets in a way that takes time series inputs of underlying prices and their options.

1.2 Interest rate market

In the sequel we present the by now classical theory of interest rates (which considers bond prices as derivatives in a market with stochastic discounting) and take it as a role model for more general markets of derivatives on equities, fixed income or foreign exchange underlyings.

The time t value of a unit of wealth, e.g. dollar, at time $0 \leq t \leq T < +\infty$ is expressed by the zero-coupon bond (briefly, ZCB) with maturity T, denoted $P(t,T)$. This is a contract which guarantees the holder to get one unit at T settling the price for this future gain at time t in the default free case, which we shall assume in the sequel. We assume that for any $t > 0$ the quantity $P(t,T)$ is a random variable on the filtered probability space $(\Omega, \mathbb{F}, \mathbb{P})$ and that it is also almost surely differentiable in the second variable T. Of course this yields a family, indexed by T, of adapted stochastic processes $(P(t,T))_{0\leq t\leq T}$. In the sequel, since we are mainly concerned with a general introduction of notions, we assume that all stated quantities exist (and are well-defined).

As a result, the definition of the *instantaneous forward rate* $f(t,T)$, which is forward view on the rate at time t, on a riskless loan that starts at T and is returned an instant later, is given by

$$f(t,T) := -\frac{\partial}{\partial T} P(t,T).$$

Consequently, the *short interest rate* is $r_t = f(t,t)$ (mind the limit). Notice that from the forward rate definition and the fact that for any $T \geq 0$ $P(T,T) = 1$, we have

$$P(t,T) = \exp\left(-\int_t^T f(t,u)\, du\right).$$

Analogously for r, we can define the *money-market account* B_t, also called bank account, as the asset which instantaneously grows at the short rate r_t through the equation $dB_t = r_t B_t\, dt$, with the convention $B(0) = 1$, which gives

$$B_t = \exp\left(\int_0^t r_s\, ds\right).$$

One of the simples ways to model a fixed income market is to represent ZCBs through a *factor model*. This approach consists in creating a deterministic map between the factor X and $P(t,T)$, i.e. $(t,T,X_t) \mapsto P^X(t,T;X_t)$. In general, we can consider X as taking values in $E \subseteq \mathbb{R}^k$ and we call all elements of the vector X factors of the model. Often it is possible to give an economic or econometric interpretation to all these factors, and this is a reason why, beyond simplicity, these kinds of model are still widely used. Consequently, another deterministic map can be identified between the factor and $f(t,T)$, in this case denoted $f^X(t,T;X_t)$. In practice, the factor X is usually modelled as an Itô process, satisfying an SDE of the form

$$dX_t = \mu^X(t,X_t)\, dt + \sigma^X(t,X_t)\, dW_t,$$

where the maps μ^X and σ^X defined as $[0, +\infty) \times E \to \mathbb{R}^k$ are deterministic. For factor models we assumed that X is solely responsible for the evolution of interest rates, which implies that the filtration is generated by the factor itself: $\mathcal{F}_t = \mathcal{F}_t^X$.

Significant examples of factor models are short rate models, where the economic factor X is taken to be one dimensional and equal to the short rate r. In this case, it is custom to postulate the dynamics of r under an equivalent martingale measure $\mathbb{Q}$, implying the absence of arbitrage (that is, loosely speaking, the possibility of making money with positive probability, but being sure of no downside risk, and at zero initial investment nothing[1]). Prices of ZCB then just appear as derivatives prices with payoff 1 at time T under a stochastic discount factor. Notice that the short rate, which is not a traded quantity, does not have to satisfy martingality constraints.

In this way, we do not have to specify the market price of risk and we can just count on any Markovian process. All the dynamic equation for r will be of the form:

$$dr_t = \mu(t, r_t)\, dt + \sigma(t, r_t)\, dW_t.$$

Most popular examples are named after their authors: the Vašíček model ([Vaš77]), the Cox-Ingersoll-Ross (CIR) model ([CIR85]), the Dothan model ([Dot78]), the Ho-Lee model ([HL86]), the Hull-White extended Vašíček model, the Hull-White extended CIR model (both in [HW90]), and so on. Despite the fact that these models are still in use nowadays, there are many shortcomings, for example:

1. The dynamics of *all* ZCBs is driven by one Markovian factor.

2. Often, the complexity of the model is not rich enough to represent complete term structures, i.e. the available parameters in μ and σ are not sufficient to explain the market prices.

3. Short-term interest rate models are all "rigid", in the sense forward rates for different maturities move in parallel. As underlined in the seminal paper by Heath, Jarrow and Morton [HJM92] and recalled by Carmona in [Car07], specifying a short rate model is equivalent to specifying the left hand-point of the forward curve (recall that for the short rate r we have $r_t = f(t, t)$), which is then shaping the entire forward curve.

1.3 Heath Jarrow Morton (HJM) models for interest rates

In the late eighties, in order to overcome some theoretical issues in the fixed income markets, Heath, Jarrow and Morton proposed a new framework to model the entire forward curve directly [HJM92]. The main problems they addressed were

- *calibration of the current term structure available at time* $t = 0$, that is the possibility of reproducing the (default-free) bond prices found in the market at initial time with the adopted model;

- *calibration of derivatives' contracts at time* $t = 0$, that is the possibility of reproducing derivatives' prices on those (liquid) underlyings identifying the term structure; and

- *calibration of the same zero coupon bonds and derivatives at time* $t \in (0, T_{\max}]$ where $T_{\max} := \max_i T_i$ is the largest maturities available in the market.

[1] An introduction to no arbitrage theory is provided in Section 4.3.

While the first two requirements can be achieved through rich enough one period models, the third is closely related with dynamic features of the model and, as it can be easily imagined, it is more difficult to obtain. This is for example the case when considering *short rate models* to price fixed income instruments. If we take for instance the Vašíček model [Vaš77] and we try to calibrate it to a forward curve, we can possibly reach a perfect matching of the curve at time $t = 0$ using the so-called Hull-White extension, but we have to abandon the idea of having the same successful calibration for future times. A new question then arises: when should we *re-calibrate* the model? Indeed the future observed forward curve will most likely not coincide with the one prescribed by our (already calibrated) model. In other words, a short rate model is *not* a dynamic model, since it is just creating an artificial machinery to capture data at only one instant of time.

For this purpose, HJM models, after the names of the three authors, are specifying the dynamics of the whole *forward rate* $f(t,T)$ for $0 \leq t \leq T$. In fact, calibration at initial time coincides with fitting the curve to the dynamic equation for f, while the evolution in time of the same equation will be used to specify the conditional probabilities for future prices.

In mathematical terms, Heath, Jarrow and Morton proposed to model f as an Itô process for each T given by the dynamics

$$df(t,T) = \alpha(t,T)\,dt + \sum_{i=1}^{d} \sigma^{(i)}(t,T)\,dW_t^{(i)}, \tag{1.1}$$

where W is a d-dimensional Wiener process and for any $j \in \{1,2,\ldots,d\}$ and any time T the processes $\sigma^{(j)}(t,T)$ and $\alpha(t,T)$ are predictable with respect to the filtration generated by W.

Given the new design, we have to specify under which conditions such models are free of arbitrage: indeed they have to satisfy two important conditions:

1. **Spot consistency condition**
   For any t the limit $\lim_{T\to t} f(t,T) = f(t,t) = r_t$ (which is assumed to exist) has to be identified with the short rate at time t. In particular, we have

   $$r_t = f(t,t) = f(0,t) + \int_0^t \alpha(s,t)\,ds + \sum_{i=1}^{d} \sigma^{(i)}(s,t)\,dW_s^{(i)}$$

   and if we add reasonable conditions[2] on the functions α and σ, assuming their differentiability in the second variable, an application of Fubini's theorem shows that

   $$r_t = r(0) + \int_0^t \zeta(u)\,du + \sum_{i=1}^{d} \int_0^t \sigma^{(i)}(u,u)\,dW_u^{(i)},$$

   with $\zeta(u) := \alpha(u,u) + \partial_u f(0,u) + \int_0^u \partial_u \alpha(s,u)ds + \sum_i^d \int_0^u \partial_u \sigma^{(i)}(s,u)dW_s^{(i)}$. This gives some restrictions on the possible models for the short rate and that is why an equation $r_t = f(t,t)$ is called a spot consistency condition.

2. **HJM drift condition**
   To guarantee an arbitrage[free market, we need to choose a class of traded contracts and

[2]Check Proposition 6.1 in [Fil09] for the exact statement.

impose (local) martingality on their discounted prices. Heath, Jarrow and Morton chose zero coupon bonds, which can be re-written in function of the forward rate as

$$P(t,T) = e^{-\int_t^T f(t,s)\,ds},$$

and obtained as immediate byproduct a condition on the drift term of f:

$$\alpha(t,T) = \sum_{i=1}^{d} \sigma^{(i)}(t,T) \int_t^T \sigma^{(i)}(t,s)\,ds.$$

It is now visible that if before we only had to specify the factors' vector X_t through X_0 and the two deterministic functions μ^X and σ^X, now we have to determine an entire curve through the initial forward curve $f(0,T)$ for any $T \geq 0$ and the volatility stochastic process $(\sigma_t)_{t\geq 0}$. It is clear that we have much more freedom in the second case, in particular we do not fix the dynamics by calibration.

The problem posed by calibration of Equation (1.1) is not trivial. For any fixed $T > 0$, it can indeed be seen as composed by infinitely many equations for all $t \in [0,T]$. Still, this is referenced as a *finite rank HJM model* because the Brownian motion W is finite dimensional. We will not enter in our formulation in the infinite dimensional case for simplicity, although it is possible to work with it as shown in [CT06a] in the spirit of Da Prato and Zabczyk [DZ14].

1.4 HJM models for equity markets

A similar approach has been developed in credit markets, see [Car07] and references therein for an introduction and, but also for equity markets. Before diving in the problem's description, we should remark the different environment: we are not dealing with interest rates any longer, but rather with a market consisting of an underlying asset together with derivatives, such as futures, options, and so on. European call and put options are the benchmark instruments in this land, and we will stick to them.

Of course, this will lead to a more complex setting because of increased dimension, since the term structure is now *cubic*, being a function of three variables: two accounting again for time, running time and maturity time, as for fixed income instruments, and one for strikes. It is possible to distinguish many different approaches, but we are going to talk mainly about those, developed in the same years, by Kallsen and Krühner [KK15] and by Carmona and Nadtochiy [CN12; CN11; CMN17], using different codebooks[3] in their respective works.

Still, the purpose in the two streams of work is always the same: specifying a set of liquidly traded instruments including underlyings and derivatives; and describe their dynamics with a time-consistent[3] model, which is able to produce sufficiently rich price dynamics at any instant of time in an arbitrage-free way.

As usual in this context, we start considering at time t a generic asset price S_t and the related derivatives' prices, e.g. call option prices, denoted by $C_t(T,K)$, for a maturity $T \geq t$ and strike $K > 0$. We consider all these quantities as observable in the market and generated by an unknown *true* model, even though this is usually a strong assumption. In reality underlyings' prices are available on finer time grids as derivatives. At this point, following the classical factor model

[3] It will become clearer later what we mean with this expression.

approach, it is natural to think of a stochastic differential equation (SDE) to model the dynamics of the underlying asset and a map to compute the relative call prices used to calibrate model parameters (i.e. SDE parameters) to match such prices. While this procedure could work well if we wanted to calibrate the model at time t, its outcome result might be inadequate already for time $t + \varepsilon$ ($\varepsilon > 0$), because of newly incoming information (e.g. new observed market data). Therefore, the model needs to be re-calibrated.

But frequent recalibration is not satisfactory from a theoretical point of view because parameters are meant to be deterministic and constant. This problem is overcome by giving options' prices the same dignity as underlying asset prices, thus thinking of derivatives as state variables of the model.

Models with such peculiarity are usually called *market models*. This entails a new working framework which is now popular with the name of HJM, since it was inspired by the seminal paper of Heath, Jarrow and Morton [HJM92] preserving the same brilliant ideas.
The price to pay for this neo-classical approach is modelling complexity. In fact, on one hand, we have the dynamics chosen to model the underlying asset S, which create a map between the parameters and S itself; on the other, we have to define a map between the same parameters and the term structure.

To account for all possible term structures we should consider sets of manifolds in the infinite 2-dimensional space of variables (times to maturity and strikes) which we would like to model with SDEs. For that, we need some kind of calculus on these manifolds and, taking inspiration from geometry, we bring differentiation into a more familiar linear space. This space, which is usually called a *chart* in the language of geometry, is called *codebook* in mathematical finance as introduced by Carmona in [Car07]. Consequently, the map between the two spaces is called *encoder* or *decoder*. Given its crucial role, it is clear that in the HJM original framework the forward rate $f(t, T)$ was the code, for which we assumed specific dynamics. We will follow the same principle here giving birth to a dynamic model which is able to evolve in time. Often the code itself corresponds to a model, like in interest rate theory where each forward rate curve corresponds to a deterministic interest rate evolution.

Whenever we stop this evolution at one code value, we should be able to produce the same observable prices generated by the unknown true model and that is why, borrowing again the words of Carmona and Nadtochiy in [CN12], we call this local and static model a *tangent* model. Since this equality must hold for all times, we can speak about time-consistency (which was previously mentioned).

The two different paths followed by the above mentioned authors are due to the different codebooks' choice they made. Indeed, although many possible choices are legitimate, they all have to satisfy specific conditions for both the underlying asset prices and the derivatives' prices, in order to generate free-arbitrage models. In what follows, we will distinguish between two different kinds of possible arbitrage opportunities. **Static arbitrage** entails the chance of setting up a static portfolio today in the existing price grid which might generate a profit, without any risk of a loss. An example could be the violation of the put-call parity for European options.

In multi-period models, in addition to static arbitrage opportunities, there may exist **dynamic arbitrage** opportunities, which arise from the possibility of generating arbitrage from the current portfolio, but in a future instant of time. More precisely, *dynamic* no arbitrage constraints are expressed as restrictions on the joint dynamics of the underlying asset and call prices to have the martingale property, while *static* no arbitrage constraints to be satisfied for all $0 \leq t \leq T$ and $K \geq 0$ need to translate in the the following conditions for call prices (see [Rop10]):

(i) $C_t(T,K)$ is a convex function in K for any $0 \leq t \leq T$,

(ii) $C_t(T,K)$ is non-decreasing in T for all $K \geq 0$,

(iii) $\lim_{K\to+\infty} C_t(T,K) = 0$ for all $0 \leq t \leq T$,

(iv) $(S_t - K)_+ \leq C_t(T,K) \leq S_t$ for all $K > 0$ and $0 \leq t \leq T$,

(v) $C_T(T,K) = (S_T - K)_+$ for all $K \geq 0$.

We can thus summarize all these conditions on the given stochastic basis by requiring the existence of an equivalent measure $\mathbb{Q}$ such that

$$\mathbb{E}_{\mathbb{Q}}[(S_T - K)_+|\mathcal{F}_t] = C_t(T,K) \text{ and } C_t(T,0) = \mathbb{E}_{\mathbb{Q}}[S_T|\mathcal{F}_t] = S_t,$$

for $0 \leq t \leq T$ and $K \geq 0$ (the case $K = 0$ is treated by considering equality with the underlying). As shown by Roper in [Rop10], these conditions are both sufficient and necessary for having no-arbitrage given that S is a non-negative martingale.

If we define $\mathcal{K}$ to be the set of available strikes and $\mathcal{T}$ the set of available maturities of the liquid derivatives at hand[4], we can think of the codebook as a stochastic process X taking values in some code space

$$dX(t) = u(t)\,dt + v(t)\,dW(t), \tag{1.2}$$

that creates a one-to-one map, a bijection, with the liquidly traded derivative assets, in our case call options $(C(T,K))_{T\in\mathcal{T},K\in\mathcal{K}}$, such that

$$X_t \mapsto \mathcal{D}(X_t)(T,K) = C_t(T,K) \quad \text{for all } T \in \mathcal{T},\ K \in \mathcal{K}. \tag{1.3}$$

One obvious instance of the decoder map $\mathcal{D}$ is the Black-Scholes formula ([BS73]), which prescribes X to be the implied volatility and the current price of the underlying.

[4]While in real markets these sets are discrete, in the literature also the continuous case is taken into consideration.

Box 1.1 – Codebook calibration

Independently of the codebook choice that can be operated, whose consequences will become soon clearer, several issues are to be considered. Namely,

1. For every instant of time $t \in [0, \max_T\{T \in \mathcal{T}\}]$, call option prices should satisfy the static arbitrage constraints listed above for all strikes and maturities.
2. To impose absence of dynamic arbitrage we need to guarantee the existence of a local martingale measure $\mathbb{Q}$ equivalent to $\mathbb{P}$. This will trigger other restrictions on the model choice, in particular affecting the coefficients in (1.2). Typically, the restrictions are non-linear and are imposed on the drift coefficient μ implying that in many cases v, the volatility coefficient, is our only modeling choice.
3. For the model to be meaningful, we also have to verify under which conditions we have existence and uniqueness of a solution for the SDE system represented in Equation (1.2).
4. Last but not least, it should also possible to link find good parametrization for the coefficient v to allow for concrete calibration.

The fact that all constraints here summarized must hold simultaneously makes the task rather difficult and challenging.

Given the complexity of the problem, it is clear that the choice of a codebook X is of central relevance.

In particular, different codebooks are responsible for imposing different no-arbitrage constraints, e.g. by requiring martingality, and so the question has been raised which codebook should be used to facilitate the most in dealing with such constraints. Roughly speaking, three suggestions have been made, which we shall shortly summarize here:

1. **Implied volatility/variance codebook**
   This is maybe the most obvious choice when thinking of a term-structure for equity markets and it is suggested by the well-known bijective relation between European option prices and implied volatilities (obtained by the same prices through the inversion of Black-Scholes formula) given the current price of the underlying. Indeed, given the current asset price S_t there is a unique (implied) volatility $\sigma_t(T, K)$ such that the Black-Scholes formula BS produces the correct market price

   $$\mathrm{BS}(T, K, S_t, \sigma_t(T, K)) = C_t(T, K)$$

   for $0 \leq t \leq T$ and $K \geq 0$. This approach was firstly taken by Schönbucher in [Sch99a], followed by Schweizer and Wissel in [SW08b] some years later in 2007. With modern terminology, the codebook used in these references is the *forward implied variance*, for which dynamics are introduced as Ito process, defined through

   $$X(t, T) := \frac{\partial}{\partial T}\Big((T - t)\,\sigma_t^2(T)\Big),$$

   where we dropped the dependence on the strike. This, however, yields two problematic aspects: first, how to deal with the dynamic absence of arbitrage, and, second, how to express the static absence of arbitrage conditions for implied volatilities. In principle, it

is possible to list all necessary conditions to avoid static arbitrage, the five mentioned above, but due to the inversion of BS these become quite complex. This explains why the authors of [SW08b] restrict themselves to the case of a single strike (just a line in the entire surface!). In this simpler case, under some regularity assumptions, it is possible to rule out dynamic arbitrage obtaining a spot consistency condition, i.e. $X(t,t) = \sigma_t^2(t)$ and also a drift condition (we reference to [SW08b] or [Car07] for the formulation).

2. **Local volatility codebook**
Another attempt has been formulated by Carmona and Nadtochiy in [Car07] and [CN09] and, in the same years, by Schweizer and Wissel in [SW08a] and [Wis07]. The goal is again to model a financial market with the following liquid assets: a bank-account, a stock (the underlying) and a (complete) set of European call options. The idea was inspired by Dupire's formula for *local volatility* ([Dup94]), that is

$$a^2(T,K) = 2\frac{\frac{\partial C}{\partial T} + rK\frac{\partial C}{\partial K}}{K^2\frac{\partial^2 C}{\partial K^2}},$$

given r constant (instantaneous) interest rate and hence modelling the underlying (under the equivalent martingale measure) as

$$dS_t = rS_t\,dt + a(t,S_t)S_t\,dW_t\,, \qquad S_0 > 0.$$

Note that in [CN09] the problem is tackled for $\mathcal{K} = (0,+\infty)$ and $\mathcal{T} = (0,+\infty)$, thus in high generality. Initially thought for calibration purposes at one instant of time only, it turned out that this approach can be very useful for dynamic modelling as well. First, static arbitrage is now assessed by just imposing non-negativity of the local volatility a, which makes it preferable compared to implied volatility. Second, and most important, it is possible to have a one-to-one correspondence between local volatility and call prices, which is a necessary condition for being a valid codebook. This is due to Breeden-Litzenberger trick and Kellerer's existence theorem. One way of building this correspondence is using the Breeden-Litzenberger formula (from [BL78]), observing that the marginal densities have a convex-order (by Jensen inequality) and then apply Kellerer's existence theorem (see [Kel72]) - check Section 6.3 in [Car07] for a thorough overview.

As these links only set for existence, we address the motivated reader to [BPS22] (and references therein) for a discussion on uniqueness.

Nevertheless, this method has critical shortcomings. Because we are lacking a continuous surface in reality, the calibration of the local volatility is often complex and non-stable, depending on derivatives that have to be calculated on a discrete set. It can be seen that the problem is also ill-posed in the sense of Hadamard ([Had02]). For more detail, see, for example, [Cré03].

Definition 1.4.1 (Ill-posed problem)**.** *The three conditions Hadamard established at the beginning of the 20^{th} century to define well-posedness of a problem $X \supseteq U \ni x \mapsto F(x) = y \in Y$ are existence of a solution, i.e. $\forall\, x \in U, \exists\, y \in Y$; uniqueness of the solution; and continuity with respect to the input, that is, if we are in metric spaces with distances d_X and d_Y, $\forall\, \varepsilon > 0, \exists\, \delta_\varepsilon > 0$ such that $\forall\, x_1, x_2 \in U$ and $y_1, y_2 \in Y$ for which we have $y_1 = F(x_1)$ and $y_2 = F(x_2)$ we must have $d_X(x_1,x_2) < \delta_\varepsilon \Rightarrow d_Y(y_1,y_2) < \varepsilon$. If one (or more) of these conditions is not fulfilled, a problem is said to be* ill-posed*.*

Moreover, the restrictions to be imposed to avoid dynamic arbitrage are quite involved: the conditions are not given explicitly, but as solution to a PDE. Simpler conditions for absence of dynamic arbitrage are found in [SW08a], where the case $\mathcal{K} = (0, +\infty)$ and $\mathcal{T} = T \in \mathbb{R}$ is considered (just one maturity). The *local implied volatility* is defined as

$$a_t(K) := \frac{\varphi(\Phi^{-1}(-\frac{\partial C_t(T,K)}{\partial K}))}{K \frac{\partial^2 C_t(T,K)}{\partial K^2} \sqrt{T-t}},$$

where the derivatives of the call prices are taken with respect to strikes and where φ and Φ are the probability density function (pdf) and cumulative distribution function (cdf) of a standard normal random variable. The formula can also be derived from the formulas for the Greeks in the Black-Scholes setting, from which we have

$$d_2 := \frac{\log\left(\frac{S_t}{K}\right) + \left(r - \frac{\sigma^2}{2}\right)(T-t)}{\sigma\sqrt{T-t}},$$

then $C_K = e^{-r(T-t)}\Phi(d_2)$ (from which we isolate d_2) and $C_{KK} = e^{-r(T-t)}\varphi(d_2)\frac{1}{K\sigma\sqrt{T-t}}$ (from which we isolate σ).
Finally, the completely discrete case is debated in [Wis07]: the same codebook of [SW08a] is defined on a lattice for a discrete set of strikes and maturities and through finite difference second order approximations to derivatives. In this case, conditions for absence of static and dynamic arbitrage are found, together with compatible requirements for the existence and uniqueness of a solution to Equations (1.2).

3. **Lévy-related codebook**
The last attempt we would like to describe seizes with both hands the flexibility offered by Lévy/jump processes. In this direction moved Carmona and Nadtochiy ([CN12; CN11] and [CMN17]), as well as Kallsen and Krühner ([KK15]) in more recent years. For this reason, in the proceedings we will refer to this approach as CNKK, although the authors chose slightly different codebooks in their respective works. All authors focused on the general continuous case for both strikes and expiries. Pricing and call-martingality conditions are simplified through (standard) Fourier methodologies.

Starting from [CN12], it is argued that not all call price surfaces can be reproduced by local volatility (but only those where the underlying S follows a regular Itô process - this result is due to Gyöngy [Gyö86][5]), since this will "explode" (unbounded behaviour) for short maturities (for $T \to t$, $\frac{\partial^2 C_t(T,K)}{\partial K^2} \to +\infty$). Consequently, they try to enlarge the class remaining in the semimartingales' realm by adding jumps and, in fact a pure-jump process is considered: more precisely, we have $S_t = \exp(X_t)$ with X pure-jump *additive* (or *time-inhomogeneous*[6]) process with deterministic compensator κ:

$$X_T = X_t + \int_t^T \int_{\mathbb{R}} (e^x - 1 - x)\kappa(s,x)\,dx\,ds + \int_t^T \int_{\mathbb{R}} x(N(ds,dx) - \kappa(s,x))\,dx\,ds,$$

where N is the Poisson random measure associated to X. Since we are possibly dealing

[5]The extension of Gyöngy's mimicking theorem to a semimartingale setting with jumps has been formalized by Bentata and Cont [BC12].

[6]Time homogeneous processes, such as Lévy processes, are not well-suited to capture an entire price surface, but only "slices" (smiles) of the same.

with infinite activity[7] processes, we have to require that

$$\int_0^t \int_{\mathbb{R}} (|x| \wedge 1)|x|(1 + e^x)\kappa(ds, dx)\, dx\, ds < +\infty.$$

It is clear that here the processes X are infinitely divisible. From an application of the Itô formula we can see that S is a non-negative martingale, while from the Lévy-Khintchine version for semimartingales it follows that κ is in unique correspondence with such processes. Finally, the bijective relation between call prices and κ itself is guaranteed by the fact the price $C_t(T, K)$ is determined by the distribution of $(S_s)_{s\in[t,T]}$ and the initial price S_t, which in turn only depend on κ and the observable S_t. The codebook is thus given by the densities $\{\kappa_t(T, x) \,|\, T > t,\, x \in \mathbb{R}\}$ and the dynamics they specify are of Itô type:

$$\kappa_t = \kappa_0 + \int_0^t \alpha_u\, du + \sum_{k=1}^{m} \int_0^t \beta_u^{(k)}\, dW_u^{(k)}, \quad \forall\, t \in [0, T),$$

under appropriate conditions for $(\alpha_u)_{u\in[0,T)}$ and $\beta^{(k)}_{u\in[0,T)}$ for $k = 1, \ldots, m$ and satisfying $\operatorname{ess\,inf}_{x\in\mathbb{R}} \kappa_t(u, x) \geq 0$ for all $u \in [t, T)$. Drift restrictions on α will then result from no-arbitrage considerations (Theorem 4.7 in [CN12]).
This approach is widen by the same authors in [CN11], because it does not provide the possibility of a continuous martingale component in the evolution of the underlying asset. For this reason, a "volatility" component is added and given as a Itô process (analogously to what is done for κ), even if it turns out that this process has to be deterministic to avoid arbitrage.

The approach followed by Kallsen and Krühner in [KK15] is still exploiting time-inhomogeneous Lévy process, but differs in the choice of the codebook. In this case, the codebook is the forward characteristic process and leads to restrictions that are easier to handle or verify than the previous models, allowing at the same time for more flexibility. Since this will be our choice in the following, we will give mathematical details in the forthcoming sections.

[7] With infinite activity process we mean a process whose integrated Lévy measure on the real line is infinite, i.e. $\int_{\mathbb{R}} \kappa(ds, x)dx = +\infty$, which is equivalent to $\int_{\|x\|\leq 1} \kappa(ds, x)dx = +\infty$.

Chapter 2

Neural Networks

2.1 Supervised learning

This introductory section is based on [SB14] and [Bac22]. Supervised learning is the branch of Machine Learning that is taking care of learning a map between some input and output data based on a given sample. In other words, supervised learning is the mathematical/computational equivalent of "learning through examples". More precisely, given a finite set of observations of the type $(x_i, y_i) \in \mathcal{X} \times \mathcal{Y}$, $i = 1, \dots, N$, the main goal is to predict the value of $y \in \mathcal{Y}$ for an unseen value $x \in \mathcal{X}$. The dataset we have at our disposal, the "examples", forms the so-called *training set*, while all other data for which there is no known prediction a priori forms the *test set*[1]. Elements belonging to $\mathcal{X}$ are usually called input, features or covariates, while elements of $\mathcal{Y}$ output, labels or responses. Supervised learning problems are usually distinguished in two different classes: if labels y_i are discrete values, we talk of *classification*, while if y_i are taking values in intervals, such as $[a, b]$ for $a \leq b$ real numbers, or $\mathbb{R}$, then we talk of *regression*.

The task might be difficult for a series of reasons. The following list is not intended to be exhaustive, but to give a general idea of the difficulties most frequently encountered.

1. The relation that links x_i to y_i might be quite complex, e.g. non-linear.
2. The relation that links x_i to y_i might be (partially) stochastic, e.g. there could be additive noise in the labels y_i due to measurement errors.
3. The number of observations we could exploit for learning is too low, increasing to a certain extent the uncertainty.
4. The space $\mathcal{X}$ might be very large, in mathematical terms, it could be high dimensional.
5. It might even be possible that unseen data are originated in a different way than training data. Thus what learned on the training set does not generalize well to the test set.

All these points present real challenges when dealing with supervised learning and makes the task interesting both from a mathematical and computational perspective.
The mathematical formulation of machine learning in general, and supervised learning in particular, is based on a probabilistic formulation. Thus, the training set $(x_i, y_i)_{i=1}^N$ we are given is seen

[1] In practice, it is common to put aside a part of the training set to measure the performance of the learned map on data for which the output is known.

as made by realizations of two random variables $(X, Y) \in \mathcal{X} \times \mathcal{Y}$ sampled in a i.i.d. way, that is every tuple (x_i, y_i) is **i**ndependent and **i**dentically **d**istributed with respect to a joint probability distribution $\mathcal{P}$ which is of course unknown. Moreover, throughout the script, we assume the distribution $\mathcal{P}$ that generated the training set is assumed to be the same as the distribution that generates all unseen data belonging to the test set. Eventually, the goal is to find a function $f : \mathcal{X} \to \mathcal{Y}$ that can produce "good" predictions on unseen data.

In order to measure the performance on test data, we need a criterion. Generally, the standard choice is trying to minimize the expected value of the discrepancy recorded on the test data. For this, it is useful to introduce the concept of *loss function*, i.e. a function $L : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}$ (often to $\mathbb{R}_{\geq 0}$) such that $(y, y') \mapsto L(y, y')$, which computes the error we are making while approximating the value y with y'[2]. Different loss functions are possible according to the problem at hand:

- Binary classification: $\mathcal{Y} = \{0, 1\}$, the 0-1 loss is $L(y, y') = \mathbb{1}_{\{y \neq y'\}}$, that is the loss is null of the $y = y'$ or 1 if the two are different.
- Binary classification: $\mathcal{Y} = \{0, 1\}$, the cross entropy is $L(y, y') = -y \log(p) - (1-y) \log(1-p)$, where p is the probability that y belongs to class '1'.
- Multiclass classification with C distinct groups, $\mathcal{Y} = \{1, 2, \ldots, C\}$, for $k \in \mathbb{N}$, we may choose again cross entropy: $-\sum_{j=1}^{C} y_j \log(p_j)$, where y_j is equal to 1 only if the element belongs to class j, p_j is the probability that the element is in class j. This loss function is defined in a different way, since the space of probabilities and possible values of y do not coincide.
- Regression: $\mathcal{Y} \subset \mathbb{R}$, the most common loss function is the squared error, that is $L(y, y') = (y - y')^2$.
- Regression: $\mathcal{Y} \subset \mathbb{R}$, another possibility is the absolute error, that is $L(y, y') = |y - y'|$.

2.1.1 Risk

As already said, the goal of supervised learning is to minimize the expected value of the loss function on the test dataset. For this, we need the definition of (expected) risk.

Definition 2.1.1 ((Expected) Risk)**.** *Given a function $f : \mathcal{X} \to \mathcal{Y}$, a loss function $L : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}$ and a joint distribution $p_{X,Y}$, we define the* (expected) risk *as*

$$\mathcal{R}(f) := \mathbb{E}_{X,Y}\left[L(Y, f(X))\right] = \int_{\mathcal{X} \times \mathcal{Y}} L(y, f(x))\, dp_{X,Y}(x, y). \tag{2.1}$$

Since the distribution $p_{X,Y}$ is not known, but we are provided with a sample set of observations $\mathcal{D}_N := \{(x_i, y_i) : i = 1, \ldots, N\}$, we can only approximate the risk with its empirical counterpart:

Definition 2.1.2 (Empirical risk)**.** *Given a function $f : \mathcal{X} \to \mathcal{Y}$, a loss function $L : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}$ and a set of observations $\mathcal{D}_N = \{(x_i, y_i) : x_i \in \mathcal{X}, y_i \in \mathcal{Y}, i = 1, \ldots, N\}$, the* empirical risk *of f is defined as*

$$\widehat{\mathcal{R}}(f) := \frac{1}{N} \sum_{i=1}^{N} L(y_i, f(x_i)). \tag{2.2}$$

[2]Unless explicitly specified, the loss functions we will use in the following are symmetric. For this reason we will not make a distinction, in general, between the role of y and y'.

Remark 2.1.3. Note that it might be useful to consider $\mathcal{D}_N$ as a possible realization of a random process $\mathbb{D}$. In this sense, also the empirical risk $\widehat{\mathcal{R}}$ becomes a random quantity.

Example of risks, taking into considerations the previously introduced loss functions, are:

- Binary classification: $\mathcal{Y} = \{0, 1\}$, for the 0-1 loss we have $L(y, y') = \mathbb{1}_{\{y \neq y'\}}$ and so $\mathcal{R}(f) = \mathbb{E}[\mathbb{1}_{\{Y \neq f(X)\}}] = \mathbb{P}(f(X) \neq Y)$, that is the probability of making a mistake.
- Regression: $\mathcal{Y} \subset \mathbb{R}$, the for the squared error loss function the risk is the mean squared error (MSE): $\mathcal{R}(f) = \mathbb{E}[(Y - f(X))^2]$; while for the absolute error loss function the risk is the mean absolute error (MAE): $\mathcal{R}(f) = \mathbb{E}[|Y - f(X)|]$.

What would happen if we knew the unknown distribution $p_{X,Y}$ that generated the sample set $\mathcal{D}_N$? In this case, we could actually minimize the expected risk.

Definition 2.1.4 (Bayes risk and Bayes predictor)**.** *Given a loss function $L : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}$ and a joint probability distribution $p_{X,Y}$ for the random variables $(X, Y) \in \mathcal{X} \times \mathcal{Y}$, we define the* Bayes risk *as*

$$\mathcal{R}^\star := \inf_{f \in \mathcal{Y}^\mathcal{X}} \mathbb{E}[L(Y, f(X))],$$

under the condition that f are measurable. Accordingly, we can define the Bayes predictor *as*

$$f^\star := \operatorname*{argmin}_{f \in \mathcal{Y}^\mathcal{X}} \mathbb{E}[L(Y, f(X))].$$

Remark 2.1.5. Let us denote with $p_X(\cdot)$ and $p_{Y|X}(\cdot|X = x)$ the probability density functions of X and $Y|X = x$, respectively. Using the law of total expectation we can rewrite the expected risk as

$$\begin{aligned}\mathcal{R}(f) &= \mathbb{E}_{X,Y}\left[L(Y, f(X))\right] = \mathbb{E}\left[\mathbb{E}\left[L(Y, f(X))|X\right]\right] \\ &= \int_\mathcal{X} \mathbb{E}\left[L(Y, f(X))|X = x'\right] \, dp_X(x')\end{aligned}$$

Since $dp(x')$ is non-negative, we can actually only look at the inner expectation for the Bayes predictor and pointwise minimize it:

$$\begin{aligned}f^\star(x) &= \operatorname*{argmin}_{f \in \mathcal{Y}^\mathcal{X}} \mathbb{E}[L(Y, f(X))|X = x] = \operatorname*{argmin}_{y' \in \mathcal{Y}} \mathbb{E}[L(Y, y')|X = x] \\ &= \operatorname*{argmin}_{y' \in \mathcal{Y}} \int_\mathcal{Y} L(y, y') \, dp_{Y|X}(y'|x).\end{aligned}$$

Moreover, note the Bayes predictor is not unique, but all Bayes predictors leads to the same Bayes risk value.

Remark 2.1.6. It is clear that Bayes predictors depend on the choice of the loss function that is operated when setting up the supervised learning task. In the case of regression with $\mathcal{Y} = \mathbb{R}$, if we choose the square loss function, we get

$$f^\star(x') \in \operatorname*{argmin}_{y' \in \mathbb{R}} \mathbb{E}[(y - y')^2 \,|\, x = x'] = \mathbb{E}[y \,|\, x = x'].$$

While for the case of absolute error, we recover $f^\star(x') = \text{median}(y \,|\, x = x')$.

2.1.2 Empirical risk minimization

To concretely find a map f between $\mathcal{X}$ and $\mathcal{Y}$, we can pursue different strategies involving different machine learning algorithms:

Definition 2.1.7 ((Learning) Algorithm)**.** *A* (learning) algorithm *is a map* $\mathcal{A} : \bigcup_{n\in\mathbb{N}}(\mathcal{X}\times\mathcal{Y})^n \to \mathcal{Y}^{\mathcal{X}}$ *that maps data into the space of functions from* $\mathcal{X}$ *to* $\mathcal{Y}$*. Thus, for any* $\mathcal{D}_N$ *(*$N \in \mathbb{N}$*), we have* $\mathcal{A}(\mathcal{D}_N) = f$ *such that* $f : \mathcal{X} \to \mathcal{Y}$*.*

One conceivable strategy, which is sometimes referred to as *local averaging*, consists in approximating the Bayes predictor in a neighbourhood of the input x. This approach originated popular algorithms, such as the K-nearest neighbours algorithm, usually adopted in classification tasks, in which the belonging of x to a group is given by the belongings of its K nearest observation points.
We will here adopt another strategy, called *empirical risk minimization* which consists in selecting a candidate model f from a class $\mathcal{F}$, usually called *hypothesis class*, and then selecting inside $\mathcal{F}$ the model that best suits our purposes. The class $\mathcal{F}$ is normally indexed by parameters $\theta \in \Theta \subseteq \mathbb{R}^p$, with $p \in \mathbb{N}$ being the dimension of the parameter space Θ. Hence, in this approach, we are not trying to mimic the Bayes predictor directly, but rather to find the best predictor through (careful) minimization of the empirical risk:

$$\hat{f} \in \operatorname*{argmin}_{f\in\mathcal{F}} \widehat{\mathcal{R}}(f) = \frac{1}{N}\sum_{i=1}^{N} L(y_i, f(x_i)),$$

which then translates to

$$f_{\theta^\star} \in \operatorname*{argmin}_{\theta\in\Theta} \widehat{\mathcal{R}}(f_\theta) = \frac{1}{N}\sum_{i=1}^{N} L(y_i, f_\theta(x_i)),$$

since the class $\mathcal{F}$ is parametrized by θ. In the context of machine learning, the minimization procedure is mainly done in an iterative way and is called *training*. Examples of algorithms that exploit this minimization problem are *linear models*, for example of the type $f_\theta(x) = \theta^\top\phi(x)$, for some $\phi(x) \in \mathbb{R}^p$ generally called *feature vector* which is assumed to be known, or *neural networks*, as we will see.

After training, we should be able to assess the "abilities" of the model on test data. According to the performance, we can distinguish models that are

- *underfitting*: when a model produces poor performance on both training data, since it is not able to interpolate or to classify correctly the observations, and on the test data. This occurs when the model cannot adequately capture the structure or patterns inherent in data. Possible motivations are the choice of the hypothesis class or the need for more parameters/features.

- *overfitting*: when a model delivers good performance on training data, but is not able to generalize well enough on test data. This occurs when the model is too specialized on the observations at the point of having extracted residual variation that might be due, originally, to noise in the measurements. One possible motivation is having too many parameters at disposal for the selected hypothesis class.

Often, the best remedy to avoid underfitting is choosing another class $\mathcal{F}$ (or at least increasing the number of parameters of the model, if this is allowed). On the other hand, avoiding overfitting might require some other techniques. The most common is probably "capacity control", which consists in shrinking the norm of some parameters by adding a penalization term to the loss function (or risk), thus resulting in

$$F(\theta) := \widehat{\mathcal{R}}(f_\theta) + \lambda\Omega(\theta).$$

The additional term is usually called *regularization term* and its importance on the overall new function is influenced by $\lambda > 0$. If we think of θ as a vector, classical examples for $\Omega(\theta)$ are the L^2-norm, i.e. $\|\theta\|_2^2$, or the L^1-norm, i.e. $\|\theta\|_1$. The intuition behind this kind of regularization is that by reducing the norm of parameters that are not essential in minimizing the risk will "reduce" the dimension of the space Θ. The new function F is normally called *target function* or *objective function*.
One could finally sum up all what said so far, by considering that the minimization of the risk as defined in (2.1), using the empirical risk minimization approach, becomes

$$\min_{\theta\in\Theta} F(\theta) = \min_{\theta\in\Theta} \frac{1}{N}\sum_{i=1}^{N} L(y_i, f_\theta(x_i)) + \lambda\Omega(\theta), \tag{2.3}$$

with $\lambda \geq 0$ (for $\lambda = 0$ there is no regularization effect). In this context, it is also natural to adapt Definition 2.1.7 of learning algorithm as a map onto Θ, since every particular $\theta \in \Theta$ is characterizing a particular function f_θ.

Remark 2.1.8 (**Bias–Variance Tradeoff**). It is customary in machine learning to decompose the expected risk, as defined in Definition 2.1.1, on unseen test data in different terms to account for a well-known phenomenon: the bias-variance tradeoff, that is the property of a model that the variance given by the calibrated model parameters estimated across samples can be reduced by increasing the bias of the same. This was already observed in 1952 by Grenander ([Gre52]) with the name of "uncertainty principle", but became popular to the machine learning community with the work of Geman and coauthors 40 years later ([GBD92]). The variance captures how much the estimator changes if a new training dataset is considered (from the same data distribution). Ideally, the variation should be minimal. On the other hand, the bias should retain the error we are making by selecting a particular hypothesis class $\mathcal{F}$ and, in particular, it is not decreased by increasing the number of training data-points. Often, if we suppose a relation of the kind $y = f(x) + \varepsilon$ for our supervised learning task, there is another term stemming from the noise ε, which is usually considered as a zero-mean independent random variable with variance equal to σ^2. This is called *irreducible error* since it concerns noise that is inherent in data.
By increasing the capacity of the model, we decrease the (squared) bias of the estimated parameters and at the same time increase their variance according to the following relation for the expected risk with squared loss function (see also Definition 2.1.1):

$$\mathbb{E}_{\substack{(x,y)\sim P,\\ \mathcal{D}_N\sim P^N}}\left[\left(y - \hat{f}(x;\mathcal{D}_N)\right)^2\right] = \left(\mathrm{Bias}_{\mathcal{D}_N}\left[\hat{f}(x;\mathcal{D}_N)\right]\right)^2 + \mathrm{Var}_{x,\mathcal{D}_N}\left[\hat{f}(x;\mathcal{D}_N)\right] + \sigma^2 \tag{2.4}$$

where $\hat{f}(\cdot\,;\mathcal{D}_N)$ has been chosen inside $\mathcal{F}$ as a result of a learning algorithm $\mathcal{A}$ (see Definition 2.1.7) and has been trained on the training set $\mathcal{D}_N = \{(x_i, y_i) : x_i \in \mathcal{X}, y_i \in \mathcal{Y}, i = 1, \ldots, N\}$ sampled by P. The bias term is defined as $\mathrm{Bias}_{\mathcal{D}} := \mathbb{E}_x[\hat{f}(x;\mathcal{D})] - f(x)$, the variance is $\mathrm{Var}_{x,\mathcal{D}} := \mathbb{E}_{x,\mathcal{D}}[(\hat{f}(x;\mathcal{D}) - \mathbb{E}_{\mathcal{D}}[\hat{f}(x;\mathcal{D})])^2]$ and ε is the error. When we add more and more parameters to the model, by increasing its complexity, variance becomes our first concern, while bias decreases. For

instance, the more polynomial terms are added to a linear regression, the greater the resulting model complexity. As a matter of fact, it might become even possible to fit the errors $(\varepsilon_i)_{i=1}^N$ inherent in measurements (overfitting).

2.2 Minimization algorithms

Problem (2.3) is now set up to be solved. Ideally, we should be able to find an analytical solution for f_θ, but this is not often the case. In addition to this, computing the analytical solution given the relevant amount of data we might be given, is usually impractical when not unfeasible. For example, as in the case for linear regression with least square, this approach might require inversion of enormous matrices that is effectively computationally very expensive and could even pose problems from a memory perspective.
For these reasons, many times iterative algorithms that are able to handle cheaper operations are used instead. In the following we are going to describe the two most common algorithms used in practice for training of different hypothesis classes, including neural networks. Both of them rely on a random (or, if possible, smart) initialization for the starting point $x_0 \in \mathcal{X} \subseteq \mathbb{R}^d$ and then they move step-by-step along a path that is selected by a form of the type $x_{t+1} \leftarrow x_t + \Delta_t$, where Δ_t depends on the local properties of the function at x_t (and may include additional information, such as the history of the path up to step t). Hopefully, the path designed in this way will lead us to a minimum[3] of the target function. For the algorithms under examination here, we will take Δ_t to be proportional to the gradient of the objective function.
The interested reader will find more information on these topics in [BV04] and [KNS16].

2.2.1 Gradient Descent (GD)

The method is a first-order iterative optimization procedure for finding a local minimum of a differentiable function and its origin is commonly attributed to Cauchy [Cau47]. This algorithm is also known as *steepest descent* since the goal is computing the direction where the target function F decreases most rapidly in a neighbourhood of the current point, and then follow this direction. It turns out that this direction is in fact provided by the gradient $\nabla_\theta F(\theta)$.

Algorithm 1: Gradient Descent

Input: $\mathcal{D}_N = \{(x_i, y_i) : i = 1, \dots, N\}$, F
Output: stationary point for F

Pick θ randomly inside Θ;
$\theta_0 \leftarrow \theta$;
for $t \geq 0$ **do**
    $\theta_{t+1} \leftarrow \theta_t - \gamma_t \nabla_\theta F(\theta_t)$;
    if stopping rule satisfied **then**
        End;

The positive quantity γ is called *learning rate* or *step size* and can be chosen adaptively, that is its value can change (usually, decrease) according to a predetermined scheme or if some conditions are met, for example if the function F has not decreased in the last iterates.
Since it is not clear when to stop a priori, different *stopping rules* are employed. The most common are:

[3] Analogous algorithms can be used for maximization.

- A maximum number of iterations has been reached;
- The value of $\|\nabla F\|$ (for some norm) is below a certain threshold;
- The relative decrease of $F(x_t)$ is below a certain threshold.

Proposition 2.2.1. *Let $x, y \in \mathbb{R}^d$ and $g \in C^1(\mathbb{R}^d)$ be such that ∇g is L-Lipschitz. Then the two following claims hold true with the norm induced by the inner product:*

a) $|g(x) - g(y) - \langle \nabla g(x), x - y\rangle| \leq \frac{L}{2}\|x - y\|^2$

b) *If g is convex, then $g(x) - g(y) - \langle \nabla g(x), x - y\rangle \leq -\frac{1}{2L}\|g(x) - g(y)\|^2$.*

Remark 2.2.2. Statement a) of Proposition 2.2.1 provides a better bound in the sense of a Taylor approximation with a quadratic function. Statement b), on the other hand, is a strengthening of the convexity condition (for $L \to +\infty$, we recover the classical convexity condition).

Remark 2.2.3. Statement a) of Proposition 2.2.1 is also relevant because it assures that using gradient descent we can recover a non-increasing path for any function g satisfying the above-mentioned conditions. Let us take $\gamma_t \equiv \gamma > 0$. If we choose $x = x_t$ and $y = x_{t+1}$, then we have $x - y = \gamma \nabla g(x_t)$ and

$$|g(x_t) - g(x_{t+1}) - \langle \nabla g(x_t), \gamma \nabla g(x_t)\rangle| \leq \frac{L}{2}\|\gamma \nabla g(x_t)\|^2,$$

which implies

$$g(x_t) - g(x_{t+1}) \geq \gamma \left(1 - \gamma \frac{L}{2}\right) \|\nabla g(x_t)\|^2 \tag{2.5}$$

which is non-negative only if $1 - \gamma\frac{L}{2} \geq 0 \iff \gamma \leq \frac{2}{L}$, which translates into $\gamma \in (0, 2/L)$.

Eventually, the application of gradient descent will lead the sequence $(x_t)_{t\geq 0}$ to a stationary point:

Theorem 2.2.4. *Let $g \in C^1(\mathbb{R}^d)$ be such that ∇g is L-Lipschitz and $x_{t+1} \leftarrow x_t - \gamma \nabla g(x_t)$ with $\gamma \in (0, 2/L)$ and $x_0 \in \mathbb{R}^d$. Then we have*

a) *$g(x_t) > g(x_{t+1})$ unless $\nabla g(x_t) = 0$.*

b) *If g is bounded from below, $\nabla g(x_t) \to 0$ for $t \to +\infty$.*

c) *If g attains a minimum $x^\star$, by taking $\gamma = \frac{1}{L}$, for all $T \in \mathbb{N}$ we have*

$$\min_{0\leq t<T} \|\nabla g(x_t)\|^2 \leq \frac{2L\,(g(x_0) - g(x^\star))}{T} = \mathcal{O}\left(\frac{1}{T}\right). \tag{2.6}$$

Proof. Claim a) is a consequence of Remark 2.2.3.
Claim b) can be seen by summing over the different iterations between $t = 0$ and $t = T$. From Inequality 2.5, we get

$$\begin{aligned} g(x_T) - g(x_0) &\geq \sum_{t=0}^{T-1} \gamma \left(1 - \gamma \frac{L}{2}\right) \|\nabla g(x_t)\|^2 \\ &\geq \gamma T \left(1 - \gamma \frac{L}{2}\right) \min_{0\leq t\leq T-1} \|\nabla g(x_t)\|^2. \end{aligned}$$

Since the left hand side is bounded from below, if we let $T \to +\infty$, then to have a finite right hand side, it must be that $\|\nabla g(x_t)\| \to 0$.
Claim c) follows from inequalities of b) since we have $g(x_0) - g(x^\star) \geq g(x_0) - g(x_T)$ and by choosing $\gamma = 1/L$. □

Remark 2.2.5. As desired, gradient descent is able to reach for a fairly general class of functions a stationary point. Unfortunately, the guarantee we have from (2.6) is not entirely satisfactory, because if we want to have $\min \|\nabla g\|^2 \leq \varepsilon$ then we need at least $t = \mathcal{O}(\frac{1}{\varepsilon})$ iterations. This rate of convergence is called *sub-linear*. The progress we make decreases the more we run the algorithm. We need generally 10 iterations to get one correct digit, 100 to get two correct digits, 1'000 for three and so on. In other words, the algorithm is said to converge in *exponential time*.

Remark 2.2.6. If the function g of Theorem 2.2.4 is convex, then gradient descent will lead the sequence $(x_t)_{t\geq 0}$ to a global minimum.
Despite having this guarantee, convex functions do not achieve a better rate for convergence.

To have a faster convergence, that is, to get a better convergence rate, we need to add other assumptions. That is why standard results for the convergence of gradient descent often consider strong-convexity.

Theorem 2.2.7. *Let $g \in C^1(\mathbb{R}^d)$ have L-Lipschitz gradient and be μ-strongly convex function for $\mu > 0$. Then for $x_0 \in \mathbb{R}^d$, let $x_{t+1} \leftarrow x_t - \gamma \nabla g(x_t)$ for all $t \geq 0$. Then,*

$$g(x_t) - g(x^\star) \leq \left(1 - \frac{\mu}{L}\right)^t (g(x_0) - g(x^\star)).$$

Remark 2.2.8. If g is a convex function, then adding an L^2-regularizer term of the type $x \mapsto \lambda/2 \|x\|^2$ makes it strongly convex with μ at least equal to $\lambda > 0$.
Thus, this addition can even accelerate convergence.

The result of Theorem 2.2.7 is now quite promising, but requiring strong-convexity might be excessive for practical problems. For example, note that every strongly-convex function is also strictly convex and, thus, admits a unique global minimum. Luckily, it is indeed possible to weaken this assumption. In order to state more important results, we first need to introduce the following inequality (from the translated work of Polyak [Pol63]):

Definition 2.2.9 (Polyak-Łojasiewicz inequality)**.** *Let $h : \mathbb{R}^d \to \mathbb{R}$ a differentiable function which attains a global minimum at $x^\star$. The function h is said to satisfy the* Polyak-Łojasiewicz (PL) inequality *if for any $x \in \mathbb{R}^d$ there exists a $\mu > 0$ such that*

$$\|\nabla h\|^2 \geq 2\,\mu \left(h(x) - h(x^\star)\right).$$

Lemma 2.2.10. *Strong-convexity implies Polyak-Łojasiewicz inequality. More precisely, if it exists $\mu > 0$ such that $g : \mathbb{R}^d \to \mathbb{R}$ is μ-strongly-convex, then Polyak-Łojasiewicz inequality holds for all subgradients.*

Proof. If the function g is μ-strongly convex, then we have by definition that

$$g(y) \geq g(x) + \langle \nabla g(x), y - x\rangle + \frac{\mu}{2}\|y - x\|^2.$$

If we minimize the inequality with respect to the variable y, the relation is kept. Hence, we obtain

$$\frac{\partial\,\mathrm{RHS}}{\partial y} = \nabla g(x) + \mu(y - x) = 0 \iff y = x - \frac{1}{\mu}\nabla g(x).$$

Now, we can denote with z the value for which g has its minimum and replace the value of y at the right hand side:

$$g(z) \geq g(x) - \frac{1}{2\mu}\|\nabla g(x)\|^2,$$

which is Polyak-Łojasiewicz inequality. □

Remark 2.2.11. If a C^1-function f satisfies the Polyak-Łojasiewicz inequality, then every stationary point is also a global minimum.

Remark 2.2.12. Satisfying the Polyak-Łojasiewicz inequality is independent of being a convex function. For example, the function $x \mapsto x^2 + 3\sin^2 x$ satisfies the PL inequality with $\mu = 1/32$, but is not convex. On the other hand, $x \mapsto |x|$ is convex, but does not satisfy the PL inequality.

Theorem 2.2.13. *Let $g \in C^1(\mathbb{R}^d)$ satisfy the Polyak-Łojasiewicz inequality for some $\mu > 0$, have L-Lipschitz gradient and a global minimum attained at $x^\star$. For $\gamma \in [0, 2/L]$, for all $T \in \mathbb{N}$ and $x_0 \in \mathbb{R}^d$ the sequence $x_{t+1} \leftarrow x_t - \gamma \nabla g(x_t)$ satisfies*

$$\begin{aligned} g(x_T) - g(x^\star) &\leq (1 + \gamma\mu(\gamma L - 2))^T \left(g(x_0) - g(x^\star)\right) \\ &= \left(1 - \frac{\mu}{L}\right)^T \left(g(x_0) - g(x^\star)\right) \quad \text{for } \gamma = 1/L. \end{aligned}$$

In this way, we are able to recover the same convergence we had for strongly-convex functions.

Remark 2.2.14. Since the quantity $1 - \mu/L \in (0, 1)$, we have *linear convergence*. Moreover, since we have $1 - z \leq \exp(-z)$ for $z \in \mathbb{R}$, we can write

$$g(x_T) - g(x^\star) \leq \exp\left(-T\frac{\mu}{L}\right)\left(g(x_0) - g(x^\star)\right).$$

If we impose the error being less than ε, we get

$$T \geq \frac{L}{\mu}\log\left(\frac{g(x_0) - g(x^\star)}{\varepsilon}\right) = \mathcal{O}(\log(1/\varepsilon)).$$

Remark 2.2.15. We conclude this excursus on gradient descent by showing that for L-Lipschitz gradient functions which satisfy the PL inequality, we always have $\mu/L \leq 1$.
First of all, we recall claim a) of Proposition 2.2.1 which establishes that

$$g(x) \leq g(y) + \langle \nabla g(x), x - y\rangle| + \frac{L}{2}\|x - y\|^2.$$

By taking the gradient on both sides and minimizing, we get

$$g(x^\star) \leq g(y) - \frac{1}{2L}\|\nabla g(y)\|^2.$$

If we now apply the PL inequality to $g(y) - g(x^\star)$ we get

$$\frac{1}{2L}\|\nabla g(y)\|^2 \leq g(y) - g(x^\star) \leq \frac{1}{2\mu}\|\nabla g(y)\|^2,$$

from which the thesis.

Remark 2.2.16 (**Gradient Flow**). The continuous version of gradient descent is called *gradient flow*, it is useful to prove theoretical results and it is basically obtained when the learning rate (or step-size) tends to zero. To derive it, let us consider, for constant $\gamma_t = \gamma > 0$, a linear interpolation between two subsequent points on the learning path $(x_t)_{t\geq 0}$ (for $t \in \mathbb{N}_0$ discrete) and suppose there exists a $n \in \mathbb{N}$ such that $x_t = X(n\gamma)$ for a function $X : \mathbb{R}_{\geqslant 0} \to \mathbb{R}^d$. Then we can write

$$X(t+\gamma) = x_{t+1} = x_t - \gamma \nabla f(x_t) = X(t) - \gamma \nabla f(X(t)),$$

from which we obtain

$$\frac{X(t+\gamma) - X(t)}{\gamma} = -\nabla f(X(t)).$$

At this point, by letting γ going to zero, the result is

$$\frac{d}{dt} X(t) = -\nabla f(X(t)), \tag{2.7}$$

which is an ODE (ordinary differential equation) that is well-defined if, for example, ∇f is Lipschitz continuous.

2.2.2 Stochastic Gradient Descent (SGD)

Since the evaluation at each iteration of the function ∇F might be computationally costly since it requires access to the entire training set, researchers have come up with an alternative solution which settles for *unbiased estimates* G of the gradient, such that for all $t \geq 0$

$$\mathbb{E}\left[G(\theta_t) | \, \theta_t \right] = \nabla F(\theta_t).$$

The first attempt for a stochastic approximation was made in 1951 by Robbins and Monro [RM51].

Algorithm 2: Stochastic Gradient Descent

Input: $\mathcal{D}_N = \{(x_i, y_i) : i = 1, \dots, N\}$, F
Output: stationary point for F

Pick θ randomly inside Θ;
$\theta_0 \leftarrow \theta$;
for $t \geq 0$ **do**
  Sample $k(t) \in \{1, \dots, N\}$ and build the gradient G_t of $\theta \mapsto L(y_{k(t)}, f_\theta(x_{k(t)}))$;
  $\theta_{t+1} \leftarrow \theta_t - \gamma_t G_t(\theta_t)$;
  if stopping rule satisfied **then**
    End;

In order to decrease the variance linked with the unbiased estimate G, *mini-batches* are also possible. This approach consists in sampling multiple indices I_t inside $\{1, \dots, N\}$ and to compute the approximate gradient as $1/|I_t| \sum_{i \in I_t} G_t\big(\theta_t; (x_i, y_i)\big)$, where $G_t\big(\theta_t; (x_i, y_i)\big)$ denotes the gradient for the i^{th}-sample (x_i, y_i) at iteration t. For all $t \geq 0$, all functions $G_t\big(\theta_t; (x_i, y_i)\big)$ are i.i.d.

Theorem 2.2.17. *Let $f \in C^1(\mathbb{R}^d)$ satisfy the Polyak-Lojasiewicz inequality for some $\mu > 0$, have L-Lipschitz gradient and a global minimum attained at $x^\star$. For any $T \geq 0$, $x \in \mathbb{R}^d$, let $(G_t(x))_{t=0}^T$ be i.i.d. random variables in $\mathbb{R}^d$ such that $\mathbb{E}[G_t(x)] = \nabla f$. Let us consider the sequence obtained by following Algorithm 2, with $x_0 \in \mathbb{R}^d$.*

a) *If* $\forall x, t$, $\mathbb{E}[\|G_t(x)\|^2] \leq \alpha$ *and* $\gamma \in (0, \frac{1}{2\mu})$, *then*

$$\mathbb{E}[f(x_T)] - f(x^\star) \leq (1 - 2\mu\gamma)^T \big(f(x_0) - f(x^\star)\big) + L\frac{\gamma\alpha^2}{4\mu}.$$

b) *If* $\forall x, t$, $\mathbb{E}[\|G_t(x)\|^2] \leq \beta^2\|\nabla f(x)\|^2$ *and* $\gamma = \frac{1}{\beta^2 L}$, *then*

$$\mathbb{E}[f(x_T)] - f(x^\star) \leq \left(1 - \frac{\mu}{\beta^2 L}\right)^T \big(f(x_0) - f(x^\star)\big).$$

Remark 2.2.18. Claim a) of Theorem 2.2.17 shows that using a constant learning rate γ is not sufficient to establish convergence. At a certain point, we will remain stuck in a region where stochastic noise is predominant and we can only get closer to the solution if γ is also approaching zero.
To tackle this problem, we need to add the condition expressed in b), assuming that stochastic noise inherent in G becomes smaller as we are approaching the stationary point.
Because of statement a), it becomes clear the strategy that is normally adopted when using stochastic gradient descent: once the loss function F is not improving, then the learning rate γ is halved (or anyway reduced) to approach to the solution.

Under the convexity assumption for the target function, the next result shows that appropriately decreasing the step size can actually guarantee convergence. For this, neither differentiability nor the Polyak-Łojasiewicz condition are necessary, but we additionally require restriction to a convex compact set.

Theorem 2.2.19. *Let* $P_C : \mathbb{R}^d \to C$ *be the projection onto a compact convex set* $C \subset \mathbb{R}^d$ *with diameter* δ *(implying that* $\forall x, y \in C$, $\|x - y\|_2 \leq \delta$*) and let* $f : C \to \mathbb{R}$ *be convex with global minimum at* $x^\star \in C$. *For any* $T \geq 0$, $x \in \mathbb{R}^d$, *let* $(G_t(x))_{t=0}^T$ *be i.i.d. random variables in* $\mathbb{R}^d$ *such that* $\mathbb{E}[G_t(x)]$ *is any subgradient* ∇f *of* f *at* x. *Moreover, suppose* $\mathbb{E}[\|G_t(x)\|^2] \leq \alpha^2$ *and let* $(\gamma_t)_{t=0}^{T-1}$ *be a finite sequence such that* $0 \leq \gamma_t \leq \gamma_{t-1}$ *for* $t = 0, \dots, T-1$. *Consider a sequence staring at* $x_0 \in \mathbb{R}^d$ *and following* $x_{t+1} = P_C(x_t - \gamma_t G_t(x_t))$. *Then* $\bar{x}_t := \frac{1}{T}\sum_{t=0}^{T-1} x_t$ *satisfies*

$$\mathbb{E}[f(x_T)] - f(x^\star) \leq \frac{1}{2T}\left[\frac{\delta^2}{\gamma_T} + \alpha^2 \sum_{t=1}^{T} \gamma_t\right],$$

and if we define $\gamma_t := \frac{\delta}{\alpha\sqrt{2t}}$, *then we have* $\mathbb{E}[f(x_T)] - f(x^\star) \leq \sqrt{\frac{2}{T}}\delta\alpha$.

2.2.3 Momentum and recent developments

Nowadays, the state of the art for minimization algorithm of gradient type includes also another term, called *momentum*. The name derives "from a physics analogy, in which the negative gradient is a force moving a particle through parameter space, according to Newton's laws of motion." (from [GBC16]). Gradient methods are known to have troubles navigating ravines, i.e. areas where the curvature of the surface on which we are moving is more pronounced for some directions than others. This particular behavior, which is typical close to local optima, is slowing down the descent process, since the directions with the largest absolute gradient are forcing the learning path to bump among each other, while making imperceptible progress in the other directions. The physical intuition is the following: a particle (a sphere) starts off by following the gradient, but once it has reached a certain velocity, it does not longer move according to

the steepest descent, since its the trajectory will be influenced by its velocity (its *momentum*) as well. In this sense, momentum damps the oscillations resulting from gradient descent. For this interpretation, a simpler version of this method is also called "heavy-ball" method and it is attributed to Boris Polyak ([Pol64]). Mathematically, we can modify both Algorithms 1 and 2 so that every step of the loop becomes:

$$\begin{aligned} z_{t+1} &\leftarrow \beta z_t + \nabla G_t(\theta_t) \\ \theta_{t+1} &\leftarrow \theta_t - \gamma_t z_{t+1} \end{aligned}, \tag{2.8}$$

where the new parameter $\beta \in [0, 1]$ is weighting the memory of the trajectory (common values are around $\beta = 0.95$ or 0.99). Note that although this method will in general accelerate convergence to a local minimizer, it is possible to build counter-examples where the method does not converge even if the target function is convex (see [LRP16]).
The method was subsequently refined in 1983, by Yurii Nesterov, Polyak's student, and is today known with the name of "Nesterov accelerated gradient" (NAG) ([Nes83]). The updating rule is very similar to (2.8):

$$\begin{aligned} z_{t+1} &\leftarrow \beta z_t + \gamma_t \nabla G_t(\theta_t + \beta z_t) \\ \theta_{t+1} &\leftarrow \theta_t + z_{t+1} \end{aligned}. \tag{2.9}$$

NAG first performs a partial update of θ_t, computing $\theta_t + \beta z_t$, which is similar to θ_{t+1}, but missing the as yet unknown correction. This benign-looking difference seems to allow NAG to change z in a quicker and more responsive way, letting it behave more stably than the standard momentum approach in many situations, especially for higher values of β. Nesterov showed that after t iterations of NAG, the traditional convergence rate of $\mathcal{O}(1/t)$ of gradient descent (Theorem 2.2.4), becomes $\mathcal{O}(1/t^2)$ and is optimal among first-order techniques that can access only to gradient evaluations ([Nes14]). We refer the reader interested in optimization theory to [Sut+13] for a comparison between momentum and NAG approaches, with a particular view on the implications these methods have for neural networks.

These approaches, combined with other techniques, such as adaptive choice of the learning rate, are now standard in machine learning and, in particular, in deep learning, where they are used for the training of neural networks. The most famous algorithms are Adagrad ([DHS11]), Adadelta ([Zei12]), RMSProp[4], Adam ([KB15]), Nadam ([Doz16]) and so on. We also suggest Chapter 8 of [GBC16] for a broader discussion of the topic.

In the rest of the manuscript, we are going to specialize on regression tasks for a particular hypothesis class: (feedforward) neural networks.

2.3 Definitions for Neural Networks

Neural network theory has attracted the interest of many researchers in recent years and it is not the goal of this manuscript to provide a comprehensive guide to the topic. This would be however a hard undertaking since the number of publication is incredibly vast and new results are published daily. Henceforth, we do not pretend that what is written here is the most updated version on any particular subject, although an attempt is made in this direction.
Conventionally, the history of neural networks starts in 1943 with Warren McCulloch, a neurophysiologist, and Walter Pitts, a young mathematician, who proposed a computational model based on electric circuits for neural networks ([MP43]). While the concept of artificial

[4]First published on slides for lecture notes at `http://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf`

neural network to which we implicitly refer in the continuation of the work dates back to the late fifties and is due to Frank Rosenblatt ([Ros57]), when the concept of *perceptron* - a machine that could receive different input and return output as a weighted sum of the former - was introduced. Many years later, (artificial) neural networks represent the state-of-the-art techniques to solve many computational challenging tasks and are one of the most promising research area for the years to come. Two interesting references we suggest for a generic introduction on neural network's theoretical aspects are [GBC16], which is rather a basic and conventional text, and [Ber+21] for a more mathematical and up-to-date treatment.

In the following we just consider feedforward neural networks, unless otherwise stated, for which we give the definition. Note that feedforward neural networks were the first and simplest type of artificial neural network devised. The information moves in only one direction - forward - from the input nodes, through the hidden nodes (if any) until the output nodes. Henceforth, there are neither cycles nor loops[5].

Definition 2.3.1 (Neural Network, [GRK20])**.** *Let $d, s, L \in \mathbb{N}$. A* neural network Φ *with input dimension d, output dimension s and L layers is a sequence of matrix-vector tuples of the type*

$$\Phi = \big((W^{(1)}, b^{(1)}), \dots, (W^{(L)}, b^{(L)})\big),$$

where $n_0 = d$, $n_L = s$ and $n_1, \dots, n_{L-1} \in \mathbb{N}$, and where each $W^{(i)}$ belongs to $\mathbb{R}^{n_i \times n_{i-1}}$ and $b^{(i)} \in \mathbb{R}^{n_i}$.
Given such a Φ, we can define for $K \subset \mathbb{R}^d$ and $\sigma : \mathbb{R} \to \mathbb{R}$ the associated realization of Φ *with activation σ over K as the map $R_\sigma(\Phi) : K \to \mathbb{R}^s$ such that*

$$R_\sigma(\Phi)(x) = x^{(L)},$$

where $x^{(L)}$ is defined as following:

$$\begin{aligned} x^{(0)} &:= x \\ x^{(i)} &:= \sigma(W^{(i)} x^{(i-1)} + b^{(i)}) \qquad \text{for } i = 1, \dots, L-1, \\ x^{(L)} &:= W^{(L)} x^{(L-1)} + b^{(L)}, \end{aligned} \tag{2.10}$$

*where the application of the function σ is componentwise. We call $N(\Phi) := d + \sum_{j=1}^{L-1} n_j + s$ the number of neurons of the neural network Φ, $L(\Phi)$ the number of layers[6] (*depth*) and $M(\Phi) := \sum_{j=1}^{L} \|W^{(j)}\|_0 + \|b^{(j)}\|_0$ the number of non-zero weights of Φ (*connectivity*). Finally, the maximum norm of the network is denoted by $B(\Phi) := \max_{j=1,\dots,L} \max\{\|W^{(j)}\|_\infty, \|b^{(j)}\|_\infty\}$ and the* width *of Φ is given by $W(\Phi) = \max_{j=1,\dots,L-1} n_j$. For the realization of such neural network we adopt the notation $\Phi \in \mathcal{N}_{d,s}$ (or $\mathcal{N}_{d,s}^{\sigma}$ if the activation function σ is used in all layers but the output).*

In the rest of the manuscript, it should be clear from the context whether we mean just the collection of weights or the realization of the neural network and for this reason, we will just use the shorter name neural network.
Notice in (2.10) that the last layer has been defined with no activation function. This is not always the case, for example in classification-type problems we might have a logistic activation

[5]Loops are edges of a network with both ends at the same node. All loops are cycles, but not all cycles are loops.

[6]The input layer is usually not considered as a layer itself and it is usually referred to as 0^{th} layer.

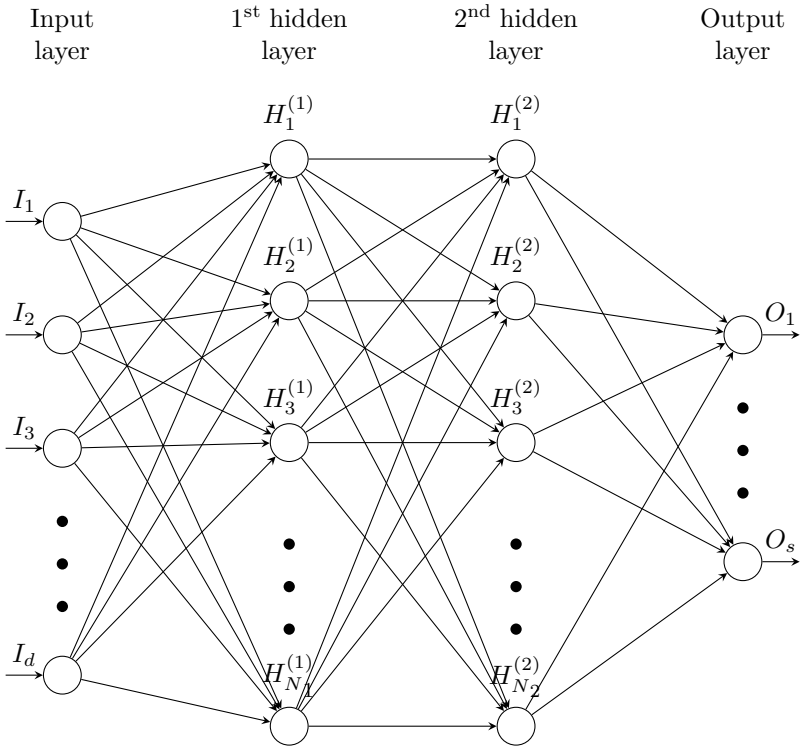


Figure 2.1: Graphical representation of a neural network with 3 layers.

function, but it is a standard choice for regression-type problems.
Layers between the input and the output layers, i.e. those in position $1, \dots, L-1$, are called *hidden layers*. Neural networks with just 1 hidden layer, having thus depth $L = 2$ in our terminology, are said *shallow neural networks*. In graphical terms, neural networks are usually represented as directional graphs, where every tuple of the type $(W^{(i)}, b^{(i)})$ is summarized by a layer (usually, a list of circles) to which the activation function is applied; edges between neurons are denoting the flow of sequential operations (see Figure 2.1).

Moreover, we will not take into consideration degenerate cases: input and hidden layers are supposed to always have at least one outgoing connection (edges in the plot), while the output layer at least an incoming connection.
For the connectivity we have the following estimate

$$M(\Phi) \leq L(\Phi)\, W(\Phi)\big(W(\Phi) + 1\big). \tag{2.11}$$

2.4 Universality properties for shallow networks

The purpose of this section is to prove that shallow neural networks can approximate well continuous function, as obtained by Cybenko in 1989 ([Cyb89]). We will also state other results similar for other sets.

Due to their simplicity, we can explicitly write down how shallow neural networks act on

input vectors $x \in \mathbb{R}^d$:

$$R_\sigma(\Phi)(x)_j = \sum_{k=1}^{N} w_{j,k}^{(2)}\, \sigma\left(\sum_{i=1}^{d} w_{k,i}^{(1)} x_i + b_k^{(1)}\right) + b_j^{(2)}, \qquad j = 1, \dots, s, \tag{2.12}$$

where $b^{(1)} \in \mathbb{R}^N, b^{(2)} \in \mathbb{R}^s$ are called *bias*, $w_k^{(1)} \in \mathbb{R}^d$ is the k^{th}-row of $W^{(1)}$, $w_j^{(2)} \in \mathbb{R}^N$ is the j^{th}-row of $W^{(2)}$ and are said *weights*, and σ is the activation function (more about those later), with an output vector of dimension s.

Activation functions are applied componentwise to every entry of a vector and are called in this way because are supposed to replicate the behaviour of real (physical!) neurons whose voltage needs to exceed a certain threshold to be able to transmit signals to neighbouring neurons. In mathematical terms, we define an activation function as a non-linear function, which is able to "fire" a signal (non-zero output) only when the input is above the threshold. A typical example of activation functions is given by sigmoidal functions:

Definition 2.4.1 (Sigmoidal function)**.** *A measurable function $\sigma : \mathbb{R} \to \mathbb{R}$ is called* sigmoidal *if it satisfies*

$$\lim_{x \to -\infty} \sigma(x) = a \quad \text{and} \quad \lim_{x \to +\infty} \sigma(x) = b,$$

with $a \neq b$ in $\mathbb{R}$.

Examples of such functions are (among the others)

- Sigmoid (or logistic) function: $\sigma_\theta(x) = \frac{1}{1+e^{-\theta\, x}}$, for $\theta \in \mathbb{R}$;
- Hyperbolic tangent: $\sigma(x) = \tanh(x) = \frac{\sinh(x)}{\cosh(x)} = \frac{e^x - e^{-x}}{e^x + e^{-x}}$;
- Heaviside function: $\sigma(x) = \mathbb{1}_{\{x \geq 0\}}$.

Actually, Cybenko could prove his famous result for a broader class of functions:

Definition 2.4.2 (Discriminatory function)**.** *Let $M(I_n)$ denote the space of finite signed Borel measures*[7] *μ on the n-dimensional unit-cube $I_n := [0,1]^n \subset \mathbb{R}^n$. A function $\sigma : \mathbb{R} \to \mathbb{R}$ is called* discriminatory *if, for every $\mu \in M(I_n)$, we have*

$$\forall\, w \in \mathbb{R}^n \text{ and } b \in \mathbb{R}, \ \int_{I_n} \sigma(w^T x + b)\, d\mu(x) = 0 \quad \Longrightarrow \quad \mu \equiv 0.$$

Unfortunately, his proof is non-constructive as it makes use of Hahn-Banach and Riesz theorems.

Theorem 2.4.3 (Hahn-Banach)**.** *Let M be a subspace of $(X, \|\cdot\|)$ and let $F_0 : M \to \mathbb{R}$ be a bounded linear functional on $(M, \|\cdot\|)$. Then, there exists a bounded linear functional $F : X \to \mathbb{R}$ on $(X, \|\cdot\|)$ that is an extension of F_0 and satisfies*

$$|||F_0||| := \sup_{x \in M, \|x\| \leq 1} F_0(x) = \sup_{x \in X, \|x\| \leq 1} F(x) =: |||F|||.$$

Corollary 2.4.4 (Consequence of Hahn-Banach)**.** *Let $(M, \|\cdot\|)$ be a subspace of $(X, \|\cdot\|)$. Suppose that every bounded linear functional $F : X \to \mathbb{R}$ on $(X, \|\cdot\|)$ with $F(x) = 0$, for all $x \in M$, is identically zero on the entire space X. Then, M is dense in X.*

[7] That is, given a measurable space (X, Σ), $\mu : \Sigma \to \mathbb{R}$ is said to be a (finite) signed Borel measure if $\mu(A) \in \mathbb{R}$ for all Borel sets $A \in \Sigma$, $\mu(\varnothing) = 0$ and for any sequence of disjoint Borel sets we have $\mu(\bigcup_{n\in\mathbb{N}} A_n) = \sum_{n\in\mathbb{N}} \mu(A_n)$.

Let $C(I_d)$ be the set of real-valued continuous functions on I_d and let us equip the space $C(I_d)$ with the *sup-norm*: $\|f\|_\infty := \sup_{x\in I_d} |f(x)|$.

Theorem 2.4.5 (Riesz representation)**.** *For every bounded linear functional L on $C(I_d)$, there exists a unique $\mu \in M(I_d)$ such that*

$$\forall\, f \in C(I_d),\;\; L(f) = \int_{I_d} f(x)\, d\mu(x).$$

Let us now restrict ourselves to the simpler case where the neural network maps the input to $\mathbb{R}$, ignoring the bias terms[8] which have been denoted as $b^{(2)}$ in (2.12). We can rewrite the shallow neural network as:

$$R_\sigma(\Phi)(x) = \sum_{k=1}^{N} w_k^{(2)}\, \sigma\left(\sum_{i=1}^{d} w_{k,i}^{(1)} x_i + b_k^{(1)}\right) = \sum_{k=1}^{N} w_k^{(2)} \sigma\left(\left\langle w_k^{(1)}, x\right\rangle + b_k^{(1)}\right), \tag{2.13}$$

with x, $w_k^{(1)} \in \mathbb{R}^d$ and $w_k^{(2)} \in \mathbb{R}$.

Theorem 2.4.6. *Let $\sigma : \mathbb{R} \to \mathbb{R}$ be a continuous discriminatory function and define*

$$S(\sigma) := \operatorname{span}\left(\left\{G_{w,b} \,:\, w \in \mathbb{R}^d,\, b \in \mathbb{R}\right\}\right), \tag{2.14}$$

for any $w \in \mathbb{R}^d$, $b \in \mathbb{R}$ and $G_{w,b} : \mathbb{R}^d \to \mathbb{R}$ defined as $G_{w,b}(x) := \sigma(\langle w, x\rangle + b)$. Then, $S(\sigma)$ is dense in $C(I_n)$.

Proof. $S(\sigma)$ is a linear subspace of $C(I_d)$ by construction. Let L be a bounded linear (i.e. continuous) functional on $C(I_d)$ with $L(F) = 0$ for all $F \in S(\sigma)$. By Riesz representation theorem, there is a unique $\mu \in M(I_d)$ such that

$$\forall\, f \in C(I_d), \quad L(f) := \int_{I_d} f(x)\, d\mu(x). \tag{2.15}$$

Since L is identically zero on $S(\sigma)$ we have

$$L(G_{w,b}) = \int_{I_d} G_{w,b}(x)\, d\mu(x) = \int_{I_d} \sigma(\langle w, x\rangle + b)\, d\mu(x) = 0, \tag{2.16}$$

for any $w \in \mathbb{R}^d$ and $b \in \mathbb{R}$. Since σ is discriminatory (2.16) implies $\mu = 0$ and, thus, $L = 0$. By Corollary 2.4.4 it follows that $S(\sigma)$ is dense in $C(I_d)$. □

To finally prove that shallow neural networks can well approximate functions in $C(I_d)$ under the norm $\|\cdot\|_\infty$, it is enough to state the following:

Proposition 2.4.7. *Every sigmoidal function $\sigma : \mathbb{R} \to \mathbb{R}$ is discriminatory.*

In practice, many activation functions other than sigmoidal are used to build a neural network. One of the most common is called **re**ctified **l**inear **u**nit (ReLU), defined as

$$f(x) = \max\{x, 0\} = x^+, \tag{2.17}$$

and it can be proven that it is also discriminatory.

[8] This does not affect the validity of the result as it is always possible apply the linear transformation to the output in a second moment.

Proposition 2.4.8. ReLU *is a discriminatory function.*

Proof. The purpose of the proof is to show that a combination of ReLU's is a sigmoidal function, which we know being discriminatory, and then that every single piece of the combination is also discriminatory (by construction). Let us define the function

$$g(x) := f(wx+b) = \mathrm{ReLU}(wx+b) - \mathrm{ReLU}(wx+b-1)$$

and we note that g behaves like a sigmoidal function with limits 1 for $x \to +\infty$ and 0 for $x \to -\infty$. Thanks to Proposition 2.4.7, we have

$$\forall\, w \in \mathbb{R} \text{ and } b \in \mathbb{R}, \ \int_{[0,1]} f(wx+b)\, d\mu(x) = 0 \implies \mu \equiv 0.$$

Thus, it must be that for any $w, b \in \mathbb{R}$, $\int_{[0,1]} \mathrm{ReLU}(wx+b)\, d\mu(x) = 0$ also implies $\mu \equiv 0$. □

Analogous results can be obtained for other similar activation functions like

- Leaky ReLU: $\sigma_\theta(x) = \begin{cases} x & \text{if } x \geq 0, \\ \theta x & \text{otherwise,} \end{cases} \quad \theta \in (0,1)$;
- **E**xponential **l**inear **u**nit (ELU): $\sigma_\theta(x) = \begin{cases} x & \text{if } x \geq 0, \\ \theta(e^x - 1) & \text{otherwise,} \end{cases} \quad \theta \geq 0.$

Remark 2.4.9. Using non-linear activation functions is a necessary condition for Theorem 2.4.6. If we took only linear functions for the hidden layer, we could recast the entire formula (2.12) as combination of linear functions, which is still a linear function. Ergo, we could only approximate with the desired accuracy linear functions.

Remark 2.4.10. Note that by employing activation functions such as ReLU or Leaky ReLU we automatically lose the differentiability property which was used in Section 2.2. However, when numerically approximating the derivatives, things go more smoothly that one could have expected. This is an active research area and we will also report some important results, as Lemma 2.6.4.

Due to Theorem 2.4.6, shallow neural networks are called *universal approximators*. In fact, this result allows us to approximate any continuous function: for any $f \in C(I_d)$, $\varepsilon > 0$ and any appropriate activation function, there exists $N \in \mathbb{N}$ and a neural network $\Phi_{f,\varepsilon}$, whose hidden layer dimension is N, for which

$$\|f - R(\Phi_{f,\varepsilon})\|_\infty < \varepsilon.$$

In particular, the width N of Φ does not remain uniformly bounded over $C(I_d)$ and ε, but grows if we require more and more accuracy.

Not only can shallow neural networks approximate unknown mappings arbitrary well, but they can do the same simultaneously with their derivatives. This was proven by Hornik, Stinchcombe and White in 1990 ([HSW90]) by using density arguments in L^p and Sobolev spaces. They could show that shallow neural networks are *m-uniformly dense on compacta*:

Definition 2.4.11 (m-uniformity on compact sets)**.** *Let $m, l \in \mathbb{N}_0$, $0 \leq m \leq l$, and $U \subset \mathbb{R}^d$ be given, and let $S \subset C^l(U)$. Suppose that for any $f \in S$, compact $K \subset U$ and $\varepsilon > 0$ there exists $\Phi \in \mathcal{N}_{d,1}$ such that $\max_{|\alpha| \leq m} \sup_{x \in K} |D^\alpha f(x) - D^\alpha \Phi(x)| < \varepsilon$. Then we say that Φ is* m-uniformly dense on compacta in S.

Let us denote with λ the Lebesgue measure on $\mathbb{R}$ or $\mathbb{R}^d$. Their proofs rely on the concept of l-finite hidden units:

Definition 2.4.12 (l-finiteness)**.** *Let $l \in \mathbb{N}_0$ be given. σ is l-finite if $\sigma \in C^l(\mathbb{R})$ and $0 < \int |D^l\sigma(x)|\, d\lambda(x) < +\infty$.*

This definition finds its *raison d'être* with the following lemma and theorem:

Lemma 2.4.13. *If σ is l-finite then for any $0 \leq m \leq l$ there exists $h \in S_1^m(\mathbb{R}, \lambda) := \{f \in C^m(\mathbb{R}) : \max_{|\alpha|\leq m} \|D^\alpha f\|_{L^1(\mathbb{R},\lambda)} < +\infty\}$ such that $h \neq 0$ and $\mathcal{N}_{d,1}^h \subset \mathcal{N}_{d,1}^\sigma$.*

Theorem 2.4.14. *Let $0 \neq \sigma \in S_1^m(\mathbb{R}, \lambda)$ for some $m \in \mathbb{N}$. Then $\mathcal{N}_{d,1}^\sigma$ is m-uniformly dense on compacta in $C_\downarrow^\infty(\mathbb{R}^d) := \{f \in C^\infty(\mathbb{R}^d) : \forall\, \alpha, \beta \in \mathbb{N}^d, \lim_{|x|\to\infty} x^\beta D^\alpha f(x) = 0\}$, where for the multi-index β we have $x^\beta := x_1^{\beta_1} \cdots x_d^{\beta_d}$ and $|x| := \max_{1\leq r\leq d} |x_r|$.*

The logistic and hyperbolic tangent activation functions are actually l-finite for all $l \in \mathbb{N}$, while this does not hold for trigonometric or polynomial functions ([HSW90]). Eventually, arbitrary approximation in the spaces

- $S_p^m(\mathbb{R}, \lambda) := \{f \in C^m(\mathbb{R}) : \max_{|\alpha|\leq m} \|D^\alpha f\|_{L^p(\mathbb{R},\lambda)} < +\infty\}$,
- $S_p^m(\text{loc}) := \{f \in C(\mathbb{R}^d) : \forall\, U \text{ open bounded} \subset \mathbb{R}^d, f \in S_p^m(U, \lambda)\}$.
- $W_p^m(U) := \{f \in L_{\text{loc}}^1(U) : \partial^\alpha f \in L^p(\mathbb{R}, \lambda),\ 0 \leq |\alpha| \leq m\}$ where $\partial^\alpha \cdot$ denotes the α^{th} weak derivative and U open subset of $\mathbb{R}^d$[9],

is a consequence of density of $C_\downarrow^\infty(\mathbb{R}^d)$ in those spaces.

2.5 The importance of depth

It is now clear that neural networks can approximate functions in different topologies. But it still remains questionable how efficiently this can be done.

Before dwelling on estimates of how deep networks are better than their shallow versions, it is interesting to report another work from Kurt Hornik in 1991 ([Hor91]) where it is shown that it is not the particular type of activation function which gives the universal approximation property to neural networks, but rather the possibility of stacking more layers (at least one hidden layer) to form more complex structures. The result could be proven for several convergence statements in Sobolev-type norms, as those of [HSW90], for continuous, bounded and non-constant activation functions. The dispute on activation functions' type (for universal approximation) was finally solved by Lesho and coauthors in 1993 for locally bounded piecewise continuous activation functions[10] ([Les+93]) and in 1999 by Pinkus ([Pin99]) for continuous activation functions. In particular, in the second article it is highlighted that uniform convergence on compacta in $C(\mathbb{R}^d)$ is achievable if and only if activation functions are non-polynomial. This conditions turned out to be necessary also in the first article.

After this short *excursus*, let us now turn out attention to *deeper* neural networks and how these can effectively obtain better results then shallow networks.

[9] The space of locally integrable functions is defined as $L_{\text{loc}}^1(U) := \{f : U \to \mathbb{R} : f|_K \in L^1(K, \lambda) \text{ for any compact } K \subset U\}$.

[10] This work also answered - negatively - a doubt raised by Hornik whether continuity of activation functions was necessary.

To start taking on this point, we will initially focus on smooth functions through which it will be later possible to approximate many other classes of functions. Since it is clear that linear functions can be approximated by minimizing the difference at will by definition, the first tile is represented by the quadratic function $f(x) = x^2$, and the main idea is due to Dmitry Yarotsky [Yar17] leveraging an observation of Telgarsky [Tel16]. The idea borrowed from Telgarsky consists in noting that the "tooth" (or Δ, for his shape) function $\Delta : [0,1] \to [0,1]$

$$\Delta(x) := \begin{cases} 2x & x < 1/2 \\ 2(1-x) & x \geq 1/2 \\ 0 & \text{elsewhere} \end{cases}$$

can be composed iteratively with itself to become a "sawtooth" function

$$\begin{aligned} \Delta^s(x) &= \underbrace{\Delta(\Delta(\dots\Delta}_{s \text{ times}}(x)\dots)) \\ &= \begin{cases} 2^s\left(x - \frac{2k}{2^s}\right) & x \in \left[\frac{2k}{2^s}, \frac{2k+1}{2^s}\right], k = 0, 1, \dots, 2^{s-1} - 1 \\ 2^s\left(\frac{2k}{2^s} - x\right) & x \in \left[\frac{2k-1}{2^s}, \frac{2k}{2^s}\right], k = 1, \dots, 2^{s-1} \end{cases} \end{aligned}$$

which has 2^{s-1} uniformly distributed "teeth" inside the interval $[0,1]$ (for $s = 0$ we consider the identity function $\Delta^0(x) = x$). An example is shown in Figure 2.2. Since the function Δ can be exactly represented as a (ReLU) neural network and the sawtooth function can be used to approximate the function $x - x^2$, we are able to achieve the conclusion.

Proposition 2.5.1. *There is a constant $C > 0$ such that for any $0 < \varepsilon < \frac{1}{2}$ there exists a neural network $\Phi \in \mathcal{N}_{1,1}$ defined on $[0,1]$ with $L(\Phi) \leq C\log(\varepsilon^{-1})$, $W(\Phi) = 3$, satisfying $\Phi(0) = 0$ and $\|\Phi(x) - x^2\|_{L^\infty([0,1])} \leq \varepsilon$.*

Proposition 2.5.1 is important for two reasons at least. First, it shows that varying the depth of the network allows to reach the desired approximation. We will see later that it is not always possible to just trade depth and width as if they were equivalent features of a network. In fact, there exists an optimal number of layers and, if the architecture has fewer layers than optimal, then the network needs to have significantly more parameters, to achieve the same approximation fidelity (see Theorem 2.5.14 below). Second, this is an essential result because it enables approximation of the multiplication operator, i.e. $f(x,y) = xy$, again through a (ReLU) network with finitely many units. The proof is substantially based on the polar identity, that is $xy = \frac{1}{2}\left[(x+y)^2 - x^2 - y^2\right]$.

Proposition 2.5.2. *There is a constant $C > 0$ such that for any $0 < \varepsilon < \frac{1}{2}$ there exists a neural network $\Phi \in \mathcal{N}_{2,1}$ defined on the interval $[-D, D]$ with $L(\Phi) \leq C(\log(\lceil D \rceil) + \log(\varepsilon^{-1}))$, $W(\Phi) \leq 5$, $B(\Phi) \leq 1$, satisfying $\Phi(x,0) = \Phi(0,y) = 0$ and $\|\Phi(x,y) - xy\|_{L^\infty([-D,D]^2)} \leq \varepsilon$.*

It is now natural to progress with monomials and, hence, polynomials.

Proposition 2.5.3. *There is a constant $C > 0$ such that for all $m \in \mathbb{N}$, $a = (a_i)_{i=0}^m \in \mathbb{R}^{m+1}$, $D \in \mathbb{R}_+$ and $0 < \varepsilon < \frac{1}{2}$, there exists a neural network $\Phi \in \mathcal{N}_{1,1}$ with $L(\Phi) \leq Cm[\log(\varepsilon^{-1}) + m\log(\lceil D \rceil) + \log(m) + \log(\lceil \max_{i=0,\dots,m} a_i \rceil)]$, $W(\Phi) \leq 9$ and norm bounded weights, i.e. $B(\Phi) \leq 1$, which satisfies $\|\Phi(x) - \sum_{i=0}^m a_i x^i\|_{L^\infty([-D,D])} \leq \varepsilon$.*

This result is actually more powerful than what it reads because from Stone-Weierstrass theorem ([Sto48]) we know that polynomials can ε-approximate any continuous function (meaning that the L^∞-norm of the difference is less than ε) on a compact set:

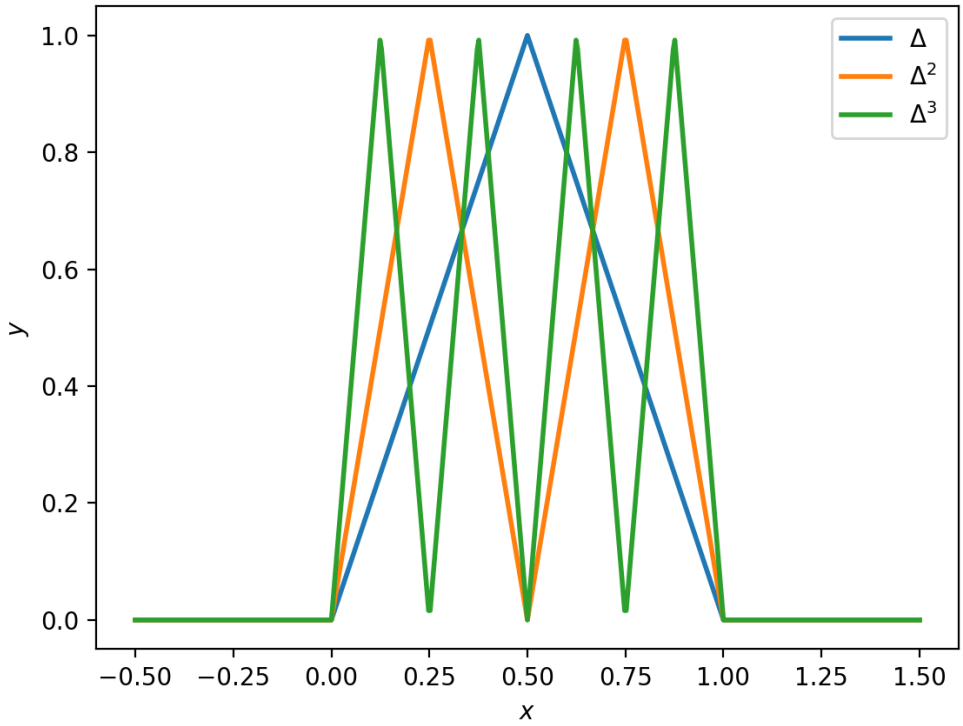


Figure 2.2: The Δ function is composed with itself on the interval $[0, 1]$.

Theorem 2.5.4 (Stone-Weierstrass, [Sto48]). *Let $[a, b] \subseteq \mathbb{R}$ and $f \in C([a, b])$. Then, for every $\varepsilon > 0$, there exists a polynomial π such that $\|f - \pi\|_{L^\infty([a,b])} \leq \varepsilon$.*

As a result, Proposition 2.5.3 implies a universal approximation theorem for ReLU networks with finite width (not more than 9) and variable depth. By recalling Relation (2.11), we can impose an upper bound on the connectivity of neural networks with finite width and whose depth scales polylogarithmically in ε^{-1} (i.e. polynomially in ε^{-1}).

Remark 2.5.5. For any continuous function on a compact set $K \subset \mathbb{R}$, the approximation error $\|\Phi - f\|_{L^\infty(K)}$ decays at least exponentially fast in the connectivity, that is with the number of weights (or parameters) of the neural network in charge for approximation.

Analogous results are valid for sinusoidal functions, like sine and cosine, without the need to resort to polynomial approximation, see for example [Elb+21].

Similar general conclusions were also previosuly obtained by Yarotsky (Theorem 1 in [Yar17]), for any function f in Sobolev spaces, exploiting partition of unity and Taylor expansions. In this case, the goal is showing that the connectivity for d-variate, n-times (weakly) differentiable functions scales with an order of $\varepsilon^{-d/n} \log(\varepsilon^{-1})$ if we require ε-approximation:

Theorem 2.5.6. *Let $W^{n,\infty}([0, 1]^d)$ be the space of functions on $[0, 1]^d$ whose weak derivatives up to order n belong to $L^\infty([0, 1]^d)$ with norm defined as $\|f\|_{\mathcal{W}^{n,\infty}([0,1]^d)} :=$*

$\max_{\boldsymbol{\alpha}:|\alpha|\leq n} \operatorname{ess\,sup}_{x\in[0,1]^d} |D^n f(x)|$, *with* $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_d) \in \{0, 1, \ldots, \}^d$, $|\boldsymbol{\alpha}| = \sum_{i=1}^d \alpha_i$. *Let* $B_{n,d}$ *be the unit ball of the space* $W^{n,\infty}([0,1]^d)$. *For any* $n, d \in \mathbb{N}$ *and* $\varepsilon \in (0, \frac{1}{2})$, *there exists a ReLU neural network* Φ *such that* Φ *is able to* ε*-approximate* f *on* $B_{n,d}$ *having* $L(\Phi) \leq C(\log(\varepsilon^{-1}) + 1)$ *and* $M(\Phi) \leq C\varepsilon^{-d/n}(\log \varepsilon^{-1} + 1)$, *for some constant* $C = C(n, d)$.

But is large depth - more than one hidden layer - really needed? Or can we just rely on increasing width for function approximations? Only recently have researchers understood that deep neural networks perform actually "better" than shallow networks, although this was already clear from empirical evidence. The first to show that deep networks cannot be approximated by adapted (more) shallow neural networks was Telgarsky ([Tel16]). As already mentioned above, his proofs rely on the so called Δ-function and on the concept of complexity based on affine regions:

Definition 2.5.7 (Number of affine pieces of f, $N_A(f)$)**.** *For any univariate function* $f : \mathbb{R} \to \mathbb{R}$, *let* $N_A(f)$ *denote the number of affine regions of* f, *that is the minimum cardinality (or* ∞*) of a partition of* $\mathbb{R}$ *so that* f *is affine when restricted to each region.*
In the literature, affine pieces are sometimes also called linear regions.

Before stating the mathematical statement, we need some intermediary results on the number of affine pieces and on properties of ReLU-networks.

Lemma 2.5.8. *Let the univariate scalar functions* $f, g, g_1, \ldots, g_k$ *and the scalars* $a_1, \ldots, a_k, b$ *be given. Then*

1. $N_A(f + g) \leq N_A(f) + N_A(g)$
2. $N_A\left(\sum_{j=1}^k a_j g_j + b\right) \leq \sum_{j=1}^k N_A(g_j)$
3. $N_A(f \circ g) \leq N_A(f) \cdot N_A(g)$
4. $N_A\left(f\left(\sum_{j=1}^k a_j g_j + b\right)\right) \leq N_A(f) \cdot \sum_{j=1}^k N_A(g_j)$

Lemma 2.5.9. *Let* $f : \mathbb{R} \to \mathbb{R}$ *be a ReLU-network with* L *layers of widths* $(m_1, \ldots, m_L)$ *with* $m = \sum_i m_i$. *Let* $g : \mathbb{R} \to \mathbb{R}$ *denote the output of some node in layer* i *as a function of the input. Then the number of affine pieces* $N_A(g)$ *satisfies*

$$N_A(g) \leq 2^i \prod_{j<i} m_j.$$

Moreover, $N_A(f) \leq (2m/L)^L$.

Remark 2.5.10. From Lemma 2.5.8 stems one of the intuitions why deep networks are more powerful: the composition of functions, which takes place by stacking layers in a network, is able to increase the number of affine regions multiplicatively (and not additively).

We can now state the following proposition, due to Telgarsky (similar results can be found with different levels of generalizations):

Proposition 2.5.11. *For any* $L \geq 2$, $f = \Delta^{L^2+2}$ *is a ReLU-network with* $3L^2 + 6$ *nodes and* $2L^2 + 4$ *layers, but any ReLU-network* g *with less than* 2^L *nodes and less than* L *layers cannot approximate it:*

$$\|f - g\|_{L^1([0,1])} = \int_{[0,1]} |f(x) - g(x)|\, dx \geq \frac{1}{32}.$$

Remark 2.5.12 (Multivariate functions). If we replaced $\mathbb{R}$ with $\mathbb{R}^d$ for $d > 1$, we can resort results from Eldan and Shamir [ES16]. They are able to show that there exists a radial function on $\mathbb{R}^d$ with bounded support that can be well approximated by a network with 3 layers, but cannot be approximated with the same accuracy by another network with 2 layers unless the width grows exponentially with the dimension d. The result is valid also for ReLU-networks.

One of the first results in this sense is again due to Yarotsky in [Yar17] (Theorems 4 and 6):

Theorem 2.5.13. *Let $B_{n,d}$ be the unit ball of the space $W^{n,\infty}([0,1]^d)$. For any $n, d \in \mathbb{N}$, $\varepsilon \in (0, \frac{1}{2})$ and $f \in B_{n,d}$, there exists a ReLU network Φ that is able to ε-approximate the function f in the $W^{n,\infty}([0,1]^d)$-norm that has $M(\Phi) \geq C_1 \varepsilon^{-d/(2n)}$, for some constant $C_1 = C_1(d,n) > 0$. Moreover, if the number of layers $L(\Phi) \leq C_2(\log(\varepsilon^{-1}))^p$ for any $p \geq 0$ and some $C_2 > 0$, then it must be $M(\Phi) \geq C_3 \varepsilon^{-d/n}(\log(\varepsilon^{-1}))^{-2p-1}$, for some $C_3 = C_3(n, d, p, C_2) > 0$.*

Theorem 2.5.13 points out a necessary condition on the number of layers. Note that for $p = 1$ the condition is compatible with the sufficient condition found in Theorem 2.5.6.

Theorem 2.5.14. *Let $f \in C^2([0,1]^d)$ be a non-linear function. Then for $L \in \mathbb{N}$ any ReLU neural network $\Phi \in \mathcal{N}_{d,1}$ approximating f with L^∞-error at most $0 < \varepsilon < \frac{1}{2}$ must have at least $C\varepsilon^{-1/2(L-2)}$ weights for some constant $C = C(f, L) > 0$.*

It follows from this theorem that a minimum number of hidden layers is required to approximate C^2 non-linear functions. Only for very smooth functions with bounded derivatives, we can hope for smaller-sized networks (Lemma 3.7 in [Elb+21]):

Proposition 2.5.15. *Let $\mathcal{S}_{[-1,1]} := \{f \in C^\infty([-1,1]) : \|f^{(n)}(x)\|_{L^\infty([-1,1])} \leq n! \; \forall n \in \mathbb{N}_0\}$. There exists a constant $C > 0$ such that for any $f \in \mathcal{S}_{[-1,1]}$ and $\varepsilon \in (0, \frac{1}{2})$ there is a network Ψ that ε-approximate f for which $L(\Psi) \leq C(\log(\varepsilon^{-1}))^2$, $W(\Psi) \leq 9$ and norm bounded weights ($B(\Phi) \leq 1$).*

Similar statements, inspired by Theorem 2.5.14, are also obtained by Petersen and Voigtlaender, but on the L^p-norm and for a non-linear function f that is slightly smoother, i.e. $f \in C^{2+\delta}$ for any $\delta > 0$ (Appendix C in [PV18]). The relation between depth and width/number of neurons is made even more precise in the same paper:

Proposition 2.5.16. *Let $f \in C^3([0,1]^d)$ be non-linear and $p \in (0, \infty)$. If there are constants $C > 0$ and $\theta > 0$, $(\varepsilon_k)_{k\in\mathbb{N}}$ a sequence of positive elements converging to 0, and $(\Phi_k)_{k\in\mathbb{N}}$ a sequence of ReLU neural networks satisfying for all $k \in \mathbb{N}$*

- $\|\Phi_k - f\|_{L^p([0,1]^d)} \leq C\varepsilon_k$,
- $M(\Phi_k) \leq C\varepsilon_k^{-\theta}$ *or* $N(\Phi_k) \leq C\varepsilon_k^{-\theta}$,

then it must be $\liminf_{k\to\infty} L(\Phi_k) \geq \frac{1}{2\theta}$.

Remark 2.5.17. We have seen that the number $N_A(f)$ of affine pieces for f feedforward neural network is a key indicator for developing theory on networks. There exists trivial bounds on such quantity. For example, if we take ReLU as activation function for the network $f : \mathbb{R} \to \mathbb{R}$, we note that the input is split at most in 2 every time the activation function is applied (exactly in 2 parts if 0 is crossed). In this case, we can derive

$$N_A(f) \leq 2^{N(f)},$$

where $N(f)$ is the total number of neurons. On the other hand, we have that the minimum number of linear regions for any neural network is 1, obtained by setting all weights and biases to 0.
For simplicity, people have focused on lower bounds for networks with piecewise linear activation functions, e.g. [Mon+14] and [Mon17], to understand the expressive power of deep architectures. Theoretically, it is possible to show that the number of regions created by a network with L hidden layers, d-dimensional input and $n_1, \dots, n_L$ neurons on the other layers is at least

$$N_A(f) \geq \prod_{i=1}^{L-1} \left\lfloor \frac{n_i}{d} \right\rfloor^d \cdot \sum_{j=0}^{d} \binom{n_L}{j}$$

and thus grows exponentially with L and polynomially in d. For ReLU neural networks, they could derive the following statement:

Proposition 2.5.18. *A ReLU neural network Φ with d input units and L hidden layers of width $m \geq d$ can compute functions that have*

$$N_A(\Phi) = \Omega\left(\left(\frac{m}{d}\right)^{(L-1)} m^d\right)$$

linear regions.

This already shows that deep ReLU networks have much higher expressive power, in this sense, with respect to shallow ReLU networks, for which the maximal number of linear regions is $\sum_{j=0}^{d} \binom{m}{j}$, with m being the number of neurons in the hidden layer.
Despite this result, it has been observed that this does not quite correspond to the behaviour to be expected. The first results on the expected number of regions were obtained by Hanin and Rolnick ([HR19]) for the case of ReLU networks or single-argument piecewise linear activation mappings onto $\mathbb{R}$. They show that if the distribution of parameters is such that the weights and biases values are random, biases always have a conditional distribution (given to all other weights and biases) with respect to the Lebesgue measure and the expected gradients of activation values are bounded, then the expected number of linear regions $\mathbb{E}[N_A(f)]$ can be much smaller than the (trivial) upper bound and, in particular, scales with the volume of the region and the total number of neurons.

Remark 2.5.19. The use of affine regions to express the complexity of a neural network has emerged quite naturally in recent years, but other proposals were made in the past, such as *Betti numbers* (borrowed from algebraic topology), which count the numbers of connected components and the numbers of holes of increasing dimensions on a specific subset $S \subset \mathbb{R}^d$ ([BS14]). A subset of $\mathbb{R}^d$ has d Betti numbers and its topological complexity is measured by summing all Betti numbers $b_i(S)$ for $i = 0, \dots, d-1$. In this case, the authors could show that deep architectures can realize maps of higher complexity than shallow ones, for some standard sigmoidal activation functions used in the hidden layers of a network Φ, by studying the set $\mathcal{S} := \{x \in \mathbb{R}^d : \Phi(x) \geq 0\}$ which is a standard choice for binary classification.

Remark 2.5.20 (**VC Dimension**). Another classical way of measuring the capacity, in the sense of expressive power, in machine learning is *VC dimension*, after the names of the mathematicians Vladimir Vapnik and Alexey Chervonenkis ([VC71]).
Let $\mathcal{H}$ denote a hypothesis class of functions from $\mathcal{X}$ to $\mathcal{Y} = \{0, 1\}$ (for a binary classification task). For any $m \in \mathbb{N}$, we define the *growth function* associated to $\mathcal{H}$ as

$$\Pi_{\mathcal{H}}(m) := \max_{x_1, \dots, x_m \in \mathcal{X}} |\{(h(x_1), \dots, h(x_m)) : h \in \mathcal{H}\}|.$$

If $|\{(h(x_1),\dots,h(x_m)) : h \in \mathcal{H}\}| = 2^m$, we say that $\mathcal{H}$ *shatters* the set $\{x_1,\dots,x_m\}$. The Vapnik-Chervonenkis dimension of $\mathcal{H}$ is the size of the largest shattered set, i.e. the largest m such that $\Pi_{\mathcal{H}}(m) = 2^m$ and is denoted by VC-dim($\mathcal{H}$). If there is no maximum, then we set VC-dim($\mathcal{H}$) $= +\infty$. When VC-dim($\mathcal{H}$) is finite, $\mathcal{H}$ is called a *VC class*. The meaning is that the classifier belonging to any $\mathcal{H}$ with VC dimension m is that the algorithm can always learn a perfect classifier for any labeling of at least one configuration of those m data points. A classical example is represented by binary classification of three points on the plane $\mathbb{R}^2$ through linear models. Since this is possible for three points, but not for four, the VC dimension is 3.
In general, models that have a high value for the VC dimension also present a certain degree of model complexity which should allow for good approximations. But a higher VC dimension also provides loose bounds when comparing expected risk and empirical risk (Chapter 3.3 from [MRT18]): for all $\delta \in (0,1)$ and h in the VC class $\mathcal{H}$, if the training set size N is larger than VC-dim($\mathcal{H}$), we have

$$\mathbb{P}\left(\mathcal{R}(h) - \widehat{\mathcal{R}}(h) \leq \sqrt{\frac{8\,\text{VC-dim}(\mathcal{H})\log\left(\frac{2eN}{\text{VC-dim}(\mathcal{H})}\right) + 8\log\left(\frac{4}{\delta}\right)}{N}}\right) \geq 1-\delta. \tag{2.18}$$

Inequality (2.18) is sometimes called *VC generalization bound*. So far, we have only considered VC dimension in terms of classification tasks, but it is also possible to extend the results for real-valued functions needed for regressions. By following [Bar+19] and the references therein, we can also consider the concept of VC-dimension for a hypothesis class $\mathcal{F}$ of functions from $\mathcal{X}$ to $\mathbb{R}$, namely VC-dim($\mathcal{F}$) := VC-dim(sign($\mathcal{F}$)), where sign($\mathcal{F}$) := $\{\text{sign}(f) : f \in \mathcal{F}\}$ and the sign function is defined as sign(x) := $\mathbb{1}_{\{x>0\}}$[11]. This article describes (nearly-)tight bounds for the VC dimension of neural networks with piecewise linear activation functions, such as ReLU. The largest VC dimension VC-dim(M, L) of such types of networks with M parameters (weights and biases) and L layers is such that there exist two constant $c, C > 0$ for which

$$c\,ML\log(M/L) \leq \text{VC-dim}(M,L) \leq CML\log(M),$$

from which we can write VC-dim(M, L) $\in \Theta(ML)$ (up to a logarithmic factor) using the big Θ notation. In view of (2.18), we have that this bound is interesting only if we consider a training set of size N which scales with the number of parameters of the neural network, but this is not the case for the successful applications that neural networks found in recent years, where the number of parameters is much larger than N ("overparametrization"). We will say more on this topic in Sections 2.6 and 2.7.

2.5.1 Kolmogorov-Donoho rate theory

In [Böl+19] Bölcskei, Grohs, Kutyniok and Petersen proposed another measure for complexity of a function class $\mathcal{C}$, namely the number of bits needed to describe any element of $\mathcal{C}$ to within prescribed accuracy. The theory they developed takes the name of "Kolmogorov-Donoho theory" and it relates the complexity of $\mathcal{C}$ with neural network complexity expressed in terms of connectivity M and memory requirements for storing both the topology (architecture) of the network and its quantized weights. Quantization refers to techniques for performing computations and storing tensors at lower bit-widths than floating point precision. In practice, one may want to convert `float32` variables, which is a floating point format occupying 32 bits in computer memory, to

[11] This is theoretically grounded on the concept of *pseudodimension* from Pollard [Pol84].

`int16`, integer numbers occupying 16 bits, and perform some or all of the operations on tensors with integers rather than floating point values. This approach was firstly suggested by Han, Mao and Delly in 2016 [HMD16] together with other procedures (such as pruning) to decrease the storage space and computational burdens of deep neural networks, at the cost of modifying the accuracy of the trained network.

The theory developed in [Böl+19] takes into consideration compact set of functions $\mathcal{C}$ in $L^2(\Omega)$, named *function classes*, with $\Omega \subset \mathbb{R}^d$ and is based on functions, named encoders and decoders, that can map elements of the function class to a string of bits and reconstruct the element of the class from that string, respectively. For a given $\ell \in \mathbb{N}$, we denote

$$\mathfrak{E}^\ell := \left\{ E : \mathcal{C} \to \{0,1\}^\ell \right\}$$

the set of *binary encoders of* $\mathcal{C}$ *of length* ℓ and

$$\mathfrak{D}^\ell := \left\{ D : \{0,1\}^\ell \to L^2(\Omega) \right\}$$

the set of *binary decoders of length* ℓ. A similar *data-compression* problem was already faced by David Donoho in 1992 ([Don93]) when he proposed using a min-max approach that we now closely replicate: the idea is that of reconstructing the initial signal, in our case a function $f \in \mathcal{C}$, after having processed it through an encoder-decoder pair tolerating a maximum error equal to ε.[12] In this procedure, we would like to take the minimal possible length ℓ that is valid for every function inside $\mathcal{C}$.

Definition 2.5.21 (Minmax code length $L(\varepsilon, \mathcal{C})$, optimal exponent $\gamma^\star(\mathcal{C})$)**.** *Let $d \in \mathbb{N}$, $\Omega \subset \mathbb{R}^d$ and $\mathcal{C} \subset L^2(\Omega)$ a compact function class. For any $\varepsilon > 0$, the quantity*

$$L(\varepsilon, \mathcal{C}) := \min \left\{ \ell \in \mathbb{N} \,:\, \exists (E, D) \in \mathfrak{E}^\ell \times \mathfrak{D}^\ell, \sup_{f \in \mathcal{C}} \|f - D(E(f))\|_{L^2(\Omega)} \leq \varepsilon \right\}$$

is called min-max code of length. *The* optimal exponent $\gamma^\star(\mathcal{C})$ *is defined as*

$$\gamma^\star(\mathcal{C}) := \sup \left\{ \gamma \in \mathbb{R} \,:\, L(\varepsilon, \mathcal{C}) \in \mathcal{O}(\varepsilon^{-1/\gamma}),\, \varepsilon \to 0 \right\}.$$

The optimal exponent $\gamma^\star(\mathcal{C})$ determines the minimum growth rate of $L(\varepsilon, \mathcal{C})$ as the error ε goes to zero and can hence be seen as quantifying the "description complexity" of the function class $\mathcal{C}$. The larger $\gamma^\star(\mathcal{C})$, the smaller growth rate of $L(\varepsilon, \mathcal{C})$ and, thus, the smaller the memory requirements are.

For function approximation in Hilbert spaces, we resort to the important notion of dictionary:

Definition 2.5.22 (Dictionary, best M-term approximation error/rate)**.** *Let $d \in \mathbb{N}$, $\Omega \subset \mathbb{R}^d$ and $\mathcal{C} \subset L^2(\Omega)$ a compact function class. A* dictionary $\mathcal{D}$ *is a set of functions $(\varphi_i)_{i \in \mathbb{N}} \subset L^2(\Omega)$. Let $f \in \mathcal{C}$ and $M \in \mathbb{N}$,*

$$\Gamma_M^{\mathcal{D}}(f) := \inf_{\substack{I_{f,M} \subseteq \mathbb{N},\, |I_{f,M}| = M, \\ (c_i)_{i \in I_{f,M}}}} \left\| f - \sum_{i \in I_{f,M}} c_i \varphi_i \right\|_{L^2(\Omega)}.$$

[12] The theory brings the name of Andrej Kolmogorov since the concept of optimal encoding is intimately connected to Kolmogorov entropy, also known as metric entropy. We refer the interested reader to [Gro15] for more information.

We call $\Gamma_M^{\mathcal{D}}(f)$ *the* best M-term approximation error of f in $\mathcal{D}$. *The supremal* $\gamma > 0$ *such that*

$$\sup_{f \in \mathcal{C}} \Gamma_M^{\mathcal{D}}(f) \in \mathcal{O}(M^{-\gamma}),\ M \to +\infty,$$

is denoted by $\gamma^\star(\mathcal{C}, \mathcal{D})$ *and is called the* best M-term approximation rate of $\mathcal{C}$ in the dictionary $\mathcal{D}$.

The intuition behind the approximation rate $\gamma^\star(\mathcal{C}, \mathcal{D})$ is that it should quantify how difficult it is to approximate the a given function class $\mathcal{C}$ using a fixed dictionary $\mathcal{D}$. Note that if the dictionary $\mathcal{D}$ is dense (and countable[13]) in the class $\mathcal{C}$, then any $f \in \mathcal{C}$ is approximated with arbitrary accuracy by one term of $\mathcal{D}$. In this case, we set $\gamma^\star(\mathcal{C}, \mathcal{D}) = +\infty$. Since looking inside $\mathcal{D}$ for the "best" M elements participating in the best approximation is, in general, unfeasible due to the infinite dimension of $\mathcal{D}$ (in fact, we would even need to infinite bits to store the information for the correct indices), the concept of *polynomial-depth* search was introduced by Donoho ([Don93]). The idea is to pick a polynomial π and to look only inside the first $\pi(M)$ elements of the dictionary $\mathcal{D}$. In this way, we get the best *effective* M-term approximation:

Definition 2.5.23 (Effective best M-term approximation rate of $\mathcal{C}$ in the dictionary $\mathcal{D}$)**.** *Let* $d \in \mathbb{N}$, $\Omega \subset \mathbb{R}^d$, $\mathcal{C} \subset L^2(\Omega)$ *a compact function class and* $\mathcal{D} = (\varphi_i)_{i \in \mathbb{N}} \subset L^2(\Omega)$ *a dictionary. For* $M \in \mathbb{N}$ *and* π *polynomial, let*

$$\varepsilon^\pi_{\mathcal{C},\mathcal{D}}(M) := \sup_{f \in \mathcal{C}} \inf_{\substack{I_{f,M} \subset \{1,2,\dots,\pi(M)\} \\ |I_{f,M}| = M,\ |c_i| \leq \pi(M)}} \left\| f - \sum_{i \in I_{f,M}} c_i \varphi_i \right\|_{L^2(\Omega)}$$

and

$$\gamma^{\star,\mathit{eff}}(\mathcal{C}, \mathcal{D}) := \sup \left\{ \gamma \geq 0 : \exists \pi \text{ polynomial s.t. } \varepsilon^\pi_{\mathcal{C},\mathcal{D}}(M) \in \mathcal{O}(M^{-\gamma}), M \to +\infty \right\}.$$

We refer to $\gamma^{\star,\mathit{eff}}(\mathcal{C}, \mathcal{D})$ *as the* effective best M-term approximation rate of $\mathcal{C}$ in the dictionary $\mathcal{D}$.

The boundedness condition imposed on the coefficients c_i will be clarified later, when employing neural networks.

Definition 2.5.24 (Optimal representability by dictionaries)**.** *Let* $d \in \mathbb{N}$ *and* $\Omega \subset \mathbb{R}^d$. *If the effective best* M*-term approximation rate of the function class* $\mathcal{C} \subset L^2(\Omega)$ *in the dictionary* $\mathcal{D} = (\varphi_i)_{i \in \mathbb{N}} \subset L^2(\Omega)$ *satisfies* $\gamma^{\star,\mathit{eff}}(\mathcal{C}, \mathcal{D}) = \gamma^\star(\mathcal{C})$ *then we say that the function class* $\mathcal{C}$ *is* optimally representable by $\mathcal{D}$.

Examples of optimal representable function classes are

- L^2-Sobolev spaces: $\mathcal{C} = \{f : \|f\|_{W_2^m} \leq C\}$ with a dictionary made by Fourier or Wavelet basis. In this case we have $\gamma^\star(\mathcal{C}) = m$.
- L^p-Sobolev spaces: $\mathcal{C} = \{f : \|f\|_{W_p^m} \leq C\}$ with a dictionary made by Wavelet basis, with $\gamma^\star(\mathcal{C}) = m$.
- Hölder spaces: let $\Omega \subset \mathbb{R}$ bounded, $\alpha \in (0, 1]$ and $\mathcal{C} = C^\alpha(\Omega) := \{f \in C^0(\Omega) : \sup_{\substack{x,y \in \Omega \\ x \neq y}} \frac{|f(x) - f(y)|}{|x - y|^\alpha}\}$ with dictionary made by wavelet basis and $\gamma^\star(\mathcal{C}) = \alpha$.

We can now formulate analogous definitions for neural networks.

[13] As required by Definition 2.5.22

Definition 2.5.25 (Best M-weight approximation error/rate)**.** *Let $d \in \mathbb{N}$, $\Omega \subset \mathbb{R}^d$, $\mathcal{C} \subset L^2(\Omega)$ a compact function class, for $f \in \mathcal{C}$ and $M \in \mathbb{N}$*

$$\Gamma_M^{\mathcal{N}}(f) := \inf_{\substack{\Phi \in \mathcal{N}_{d,1} \\ M(\Phi) \leq M}} \|f - \Phi\|_{L^2(\Omega)}.$$

We call $\Gamma_M^{\mathcal{N}}(f)$ the best M-weight approximation error of f*. The supremal $\gamma > 0$ such that*

$$\sup_{f \in \mathcal{C}} \Gamma_M^{\mathcal{N}}(f) \in \mathcal{O}(M^{-\gamma}),\ M \to +\infty,$$

is denoted by $\gamma^\star_{\mathcal{N}}(\mathcal{C})$ and is called the best M-weight approximation rate of $\mathcal{C}$ by the neural networks*.*

Note that the infimum in Definition 2.5.25 is taken over all possible networks, independently of architectures, that have input dimension d, output dimension 1, at most M non-zero weights, but variable depth L. The restrictions we imposed above in Definition 2.5.23 to reach the notion of *effective* approximation are now translated in neural networks whose depth and whose coefficients are polynomially bounded. From Proposition 2.5.3, we notice that depth scales polynomially in $\log(\varepsilon^{-1})$. Since we are interested in approximation error decay according to $\varepsilon \sim M(\Phi)^{-\gamma}$ (from Definition 2.5.25), we can try limiting $L(\Phi)$ with a polynomial in $\log(M(\Phi))$. Imposing such restrictions makes it convenient the following definition:

Definition 2.5.26 ($\mathcal{N}^\pi_{M,d,d'}$)**.** *For M, d, $d' \in \mathbb{N}$, and π a polynomial, we define*

$$\mathcal{N}^\pi_{M,d,d'} := \{\Phi \in \mathcal{N}_{d,d'} \,:\, M(\Phi) \leq M,\, L(\Phi) \leq \pi(\log M),\, B(\Phi) \leq \pi(M)\}.$$

Next, we formalize the concept of effective best M-weight approximation rate subject to polylogarithmic depth and polynomial weight growth:

Definition 2.5.27 (Effective best M-weight approximation rate of $\mathcal{C}$ with neural networks)**.** *Let $d \in \mathbb{N}$, $\Omega \subset \mathbb{R}^d$, $\mathcal{C} \subset L^2(\Omega)$ a compact function class. For $M \in \mathbb{N}$ and π polynomial, let*

$$\varepsilon^\pi_{\mathcal{N}}(M) := \sup_{f \in \mathcal{C}} \inf_{\Phi \in \mathcal{N}^\pi_{M,d,1}} \|f - \Phi\|_{L^2(\Omega)}$$

and

$$\gamma^{\star,\mathit{eff}}_{\mathcal{N}}(\mathcal{C}) := \sup\left\{\gamma \geq 0 : \exists \pi \text{ polynomial s.t. } \varepsilon^\pi_{\mathcal{N}}(M) \in \mathcal{O}(M^{-\gamma}), M \to +\infty\right\}.$$

We refer to $\gamma^{\star,\mathit{eff}}_{\mathcal{N}}(\mathcal{C})$ as the effective best M-weight approximation rate of $\mathcal{C}$*.*

We have now our first important result:

Theorem 2.5.28. *Let $d \in \mathbb{N}$, $\Omega \subset \mathbb{R}^d$ bounded, $\mathcal{C} \subset L^2(\Omega)$. Then we have*

$$\gamma^{\star,\mathit{eff}}_{\mathcal{N}}(\mathcal{C}) \leq \gamma^\star(\mathcal{C}),$$

that is the optimal exponent $\gamma^\star(\mathcal{C})$ is an upper bound for the effective best M-weight approximation rate of $\mathcal{C}$.

Consequently, we will say that a function class $\mathcal{C}$ is *optimally representable by neural networks* if $\gamma^{\star,\mathrm{eff}}_{\mathcal{N}}(\mathcal{C}) = \gamma^\star(\mathcal{C})$.

It is now natural to ask whether neural networks can actually optimally represent dictionaries for which we already know the approximation properties.

Definition 2.5.29 (Effectively representable dictionaries)**.** *Let $d \in \mathbb{N}$, $\Omega \subset \mathbb{R}^d$, $\mathcal{D} = (\varphi_i)_{i\in\mathbb{N}} \subset L^2(\Omega)$ be a dictionary. We call $\mathcal{D}$* effectively representable by neural networks*, if there exists a bivariate polynomial π such that for all $i \in \mathbb{N}$, $\varepsilon \in (0, \frac{1}{2})$, there is a neural network $\Phi_{i,\varepsilon} \in \mathcal{N}_{d,1}$ for which $M(\Phi_{i,\varepsilon}) \leq \pi(\log \varepsilon^{-1}, \log i)$, $B(\Phi_{i,\varepsilon}) \leq \pi(\varepsilon^{-1}, i)$ and $\|\varphi_i - \Phi_{i,\varepsilon}\|_{L^2(\Omega)} \leq \varepsilon$.*

The conditions listed in Definition 2.5.29 are needed to control the connectivity and the values of the weights are also similar to those we imposed on the elements of dictionaries and are essential to get polylogarithmic boundedness. The upper bound on connectivity $M(\Phi)$ given by $\log(i)$ is responsible for the polynomial depth search constraint of the best M-term approximation in $\mathcal{D}$; while being interested in an error $\varepsilon = M^{-\gamma}$ translates in the condition with $\log(\varepsilon^{-1})$. Similarly, weights of $\Phi_{i,\varepsilon}$ are guaranteed to be polynomial in M by the other restriction $B(\Phi_{i,\varepsilon}) \leq \pi(\varepsilon^{-1}, i)$. Eventually, we can conclude with the following theorem:

Theorem 2.5.30. *Let $d \in \mathbb{N}$, $\Omega \subset \mathbb{R}^d$ bounded, $\mathcal{C} \subset L^2(\Omega)$ be a function class and $\mathcal{D} = (\varphi_i)_{i\in\mathbb{N}} \subset L^2(\Omega)$ be a dictionary that is effectively representable by neural networks. Then, for every $0 < \gamma < \gamma^{\star,\mathit{eff}}(\mathcal{C}, \mathcal{D})$, there is a polynomial π and a map $\Psi : (0, \frac{1}{2}) \times \mathcal{C} \to \mathcal{N}_{d,1}$ such that for all $f \in \mathcal{C}$ and $\varepsilon \in (0, 1/2)$ the network $\Psi(\varepsilon, f)$ satisfies the following conditions:*

1. *$\Psi(\varepsilon, f)$ has quantized weights,*
2. *$\|f - \Psi(\varepsilon, f)\|_{L^2(\Omega)} \leq \varepsilon$,*
3. *$L(\Psi(\varepsilon, f)) \leq \pi(\log \varepsilon^{-1})$, $B(\Psi(\varepsilon, f)) \leq \pi(\varepsilon^{-1})$ and $M(\Psi(\varepsilon, f)) \in \mathcal{O}(\varepsilon^{-1/\gamma})$ for $\varepsilon \to 0$.*

Moreover, it holds

$$\gamma^{\star,\mathit{eff}}_{\mathcal{N}}(\mathcal{C}) \geq \gamma^{\star,\mathit{eff}}(\mathcal{C}, \mathcal{D}).$$

Theorem 2.5.30 allows to conclude. If $\mathcal{D}$ optimally represents the function class $\mathcal{C}$ in the sense of Definition 2.5.24, i.e. $\gamma^{\star,\mathrm{eff}}(\mathcal{C}, \mathcal{D}) = \gamma^\star(\mathcal{C})$, and if it is, in addition, effectively representable by neural networks as described in Definition 2.5.29, then, thanks to Theorem 2.5.28, which says that $\gamma^{\star,\mathrm{eff}}_{\mathcal{N}}(\mathcal{C}) \leq \gamma^\star(\mathcal{C})$ we have

$$\gamma^{\star,\mathrm{eff}}(\mathcal{C}, \mathcal{D}) \leq \gamma^{\star,\mathrm{eff}}_{\mathcal{N}}(\mathcal{C}) \leq \gamma^\star(\mathcal{C}) = \gamma^{\star,\mathrm{eff}}(\mathcal{C}, \mathcal{D})$$

and hence $\mathcal{C}$ is optimally representable by neural networks ($\gamma^{\star,\mathrm{eff}}_{\mathcal{N}}(\mathcal{C}) = \gamma^\star(\mathcal{C})$).
As proved in [Böl+19], all affine dictionaries, which consists of dilations and translations applied to generator functions, are effectively representable by neural networks and, thus, any function class that is optimally represented by an arbitrary affine system is optimally represented by neural networks. Among the others, wavelets are a special case of affine dictionaries. The same holds true for Gabor dictionaries, generated by translations in time and frequency (through a modulation operator $f(t) \mapsto e^{2\pi \mathrm{i} \langle \cdot, t\rangle} f(t)$) of a generator function.

2.5.2 A particular architecture: Residual Networks

Neural networks are an extremely flexible Machine Learning methodology, in the sense that they can adapt to various tasks, even completely different among them. Part of their success is certainly due to the different structure and the possibility of modifying layers quite freely. We will not study in this work recurrent neural networks or convolutional neural networks, but it might be worth spending time on a particular architecture of feedforward networks which turned out to be really successful, namely *residual networks* or *ResNet* for short. This architecture has been proposed initially in [He+16] in order to explore performance of very deep models (up to

hundreds layers), without surrendering to training degradation accuracy when stacking more and more layers. For this reason, it has been applied successfully in a number of applications, for example in [He+16; Hua+16].
The main idea is the introduction of *residual blocks*, which are heaps of fully connected layers plus an "external" layer which is skipping all of them and joining, through summation, the output of the last layer belonging to the heap. The goal of this external layer is move forward the input of the residual block without any transformation. From a mathematical point of view, it corresponds to an identity layer. For this reason, the remaining part, denoted with $\mathcal{F}$ in the original reference, which is then actively trained, is called residual (and $\mathcal{F}$ residual map). The output of the entire block is then summarized by the following:

$$\mathcal{H}(x) = \mathcal{F}(x) + x.$$

To the extreme, if an identity mapping were optimal, it would be easier to push the residual to zero than to fit an identity mapping by a stack of nonlinear layers[14]. One example of residual map is $\mathcal{F}(x) = W_2\sigma(W_1x)$, where the residual is made of two layers, denoted by the weights W_1 and W_2 and where σ is the non-linear activation function. But in principle, one can make the structure of $\mathcal{F}$ as complex as desired. Note that adding a skip connection (the identity) does not impact the computational aspect: we are not introducing new weights that should be trained, but simply new matrix-vector operations that are supposed to be rather cheap for a computer.

Most importantly, ResNets have shown their utility in solving the *vanishing gradient problem*, a phenomenon that is quite common when training very deep neural networks using gradient-based methods, such as stochastic gradient descent. This problem is particularly evident with activation functions such as sigmoid or hyperbolic tangent, that are flattening away from the origin and, hence, having a gradient that is basically null in these regions. Since the update of layer's weights is proportional to the partial derivative of the error committed by comparing the prediction and the label, having a vanishing gradient has the consequence that weight's update becomes more and more difficult since the weights effectively do not evolve with additional iterations. Skip connections of ResNets allow gradient information to pass through the layers, helping maintain signal propagation even in deeper networks. This interpretation is also shared and studied by Veit and coauthors in [VWB16], where they propose to see ResNets as a collection of several paths with different length. In particular, they bring arguments to explain that the success of such networks in avoiding the vanishing gradient problem is due to the creation of short paths that can efficiently spread the information.

Finally, ResNets are interesting also for their link to neural ODEs as introduced in [Che+18]. Residual networks can be summarized by the following transformation (for hidden layers):

$$h_{t+1} = h_t + \mathcal{F}(h_t, w_t), \qquad \text{for } t = 0, \dots, T-1,$$

where $h_0 = h(0)$ can be seen as the input layer and h_T the output layer, where we made explicit the dependence of the residual map $\mathcal{F}$ on the weights w_t of the residual block. This iterative update can be seen as an Euler discretization scheme of a continuous counterpart. If we stack more and more layers and take the limit, then we can consider ResNets "converging" to ODEs of the type

$$\frac{dh(t)}{dt} = \mathcal{G}(h(t), w, t),$$

[14] If the dimension of the input x and output $\mathcal{H}(x)$ do not match, it is possible to add a matrix of weights responsible for projection.

for an appropriate function $\mathcal{G}$ derived from $\mathcal{F}$. This has triggered new ways of studying theoretical properties and characteristics of neural networks. See, for example, the already cited [Che+18], but also [CLT15] and [Coh+21] and references therein.

2.6 Attainment of global minima

One of the most striking characteristics of neural networks, which is still not fully understood, but opened doors for their success, is their ability to generalize very well on unseen data despite being trained only on a finite sample set. This phenomenon can be verified very easily, even with shallow networks, if we try to fit noisy observations in a typical regression task, which usually implies choosing mean-squared error as loss function and employing (stochastic) gradient descent in combination with backpropagation to optimize the weights of the network. Since we know that shallow networks are universal approximators (cf. Theorem 2.4.6), we might expect that they overfit, as described in Section 2.1.2, training data quite easily, in particular if the number of weights is larger than the number of training points (which is often the case), or at least, we might expect them to generalize arbitrarily "bad" on points in unexplored regions. An essential step forward in this direction was made by C. Zhang and coauthors ([Zha+16]) by showing that neural networks can easily fit random labelled data, reaching zero training error. On the other hand, test error is then simply by random guessing since there is no correlation between training and test labels. Consequently, neural networks (with enough parameters) have the ability to fully memorize the entire training dataset. Despite these surprising facts, when we train a network for a finite amount of time (i.e. for finite epochs), we see from empirical evidence that no overfit occurs. Quite stunningly, the solution that is reached in this way is more desirable for standard applications than the solution obtained by a perfect minimization of the loss function (more on this in Section 2.7.1).

It has been shown in recent works that Neural Networks trained with stochastic gradient descent (SGD) can achieve global minima for a given loss function in polynomial time and, hence, minimize the training error, as already reported in ([Zha+16]). This was shown, for example, by Allen-Zhu and coauthors in [ALS19a], and, even if similar results were reached at the same time by other researchers ([Du+19]), this is still a very active research area. To show the results the only non-technical assumptions are made consist in:

- non-degeneracy of the input data, that is they assume that there exists $\delta > 0$ such that the Euclidean norm $\|x_i - x_j\|_2 \geq \delta$ for all $i \neq j$ indexing the elements of the dataset;
- overparametrization, which is equivalent to say that the number of parameters composing the network is (much) larger than the amount of data available for training;
- He weight initialization [He+15], that is a Gaussian random initialization of the weights of the network at initial time, hence, if we denote with m the width of hidden layers and s the dimension of the output layer, the entries of the matrices will be sampled as $\mathcal{N}(0, 2/m)$ for input and hidden layers and as $\mathcal{N}(0, 1/s)$ for the output layer.

Mathematically, they derived that the dependence for the width W for a network with L layers is polynomial in L, δ^{-1} and n, where N is the size of the dataset: to have an ε-error on the mean squared error (the mean of the ℓ_2-norm, usual choice for regression problems) a sufficient condition is $W \geq \text{POLY}(L, \delta^{-1}, N)$ in at most $T = \text{POLY}(L, \delta^{-1}, N) \log \varepsilon^{-1}$ iterations. Let us be more precise here. We consider a feedforward neural network $\Phi : \mathbb{R}^d \to \mathbb{R}^s$ with L^2-loss function

$L(y, \Phi(x; w)) = \frac{1}{2}\|y - \Phi(x; w)\|_2^2$, where $y \in \mathbb{R}^s$, $x \in \mathbb{R}^d$ and w denotes the parameters of Φ. Since we are optimizing over w, we want to solve the problem

$$\inf_w F(w) := \sum_{i=1}^{N} L(y_i, \Phi(x_i; w)) \tag{2.19}$$

for (x_i, y_i) in the training dataset $\mathcal{D}_n := \{(x_i, y_i) : i = 1, \dots, N\}$.

Theorem 2.6.1 (Global convergence for GD)**.** *Suppose that the assumptions previously stated hold true and that* $m \geq \Omega(\textsc{poly}(N, L, \delta^{-1}) \cdot s)$*. Starting from He random initialization, with probability at least* $1 - e^{-\Omega(\log^2 m)}$*, gradient descent with learning rate* $\gamma = \Theta(\frac{s\delta}{\textsc{poly}(N,L)\cdot m})$ *finds a point* $F(w) \leq \varepsilon$ *in* $T = \Theta(\frac{\textsc{poly}(N,L)}{\delta^2} \cdot \log \varepsilon^{-1})$ *iterations.*

Theorem 2.6.2 (Global convergence for SGD)**.** *Suppose that the assumptions previously stated hold true, that mini-batches have size* $b \in \{1, \dots, N\}$ *and that* $m \geq \Omega(\frac{\textsc{poly}(N,L,\delta^{-1})\cdot s}{b})$*. Starting from He random initialization, with probability at least* $1 - e^{-\Omega(\log^2 m)}$*, stochastic gradient descent with learning rate* $\gamma = \Theta(\frac{bs\delta}{\textsc{poly}(N,L)\cdot m \log^2 m})$ *finds a point* $F(w) \leq \varepsilon$ *in* $T = \Theta(\frac{\textsc{poly}(N,L)\cdot \log^2 m}{\delta^2 \cdot b} \cdot \log \varepsilon^{-1})$ *iterations.*

Remark 2.6.3. In both theorems, the result is independent of the input dimension d. Moreover, the convergence rate is linear, since the error ε drops exponentially fast with T.

The result holds with high probability for training with gradient descent (GD) or stochastic gradient descent (SGD) for different architectures, such as deep feedforward networks (DNNs), Convolutional Neural Networks (CNNs), Residual Neural Networks (ReNtes) and Recurrent Neural Networks (RNNs) [ALS19b] and for different (possibly non-convex) loss functions, among which cross-entropy. Moreover, all results are obtained for ReLU networks, thus applying ReLU activation to each neuron, which is well-known for having discontinuous first derivative. This last issue is tackled with what is called in the paper "semi-smoothness":

Lemma 2.6.4. *Let us denote with* w *all the weights of the network, with* W *those associated to hidden layers and with* $W^{(0)}$ *the initial weights (randomly chosen). With probability at least* $1 - e^{-\Omega(m/\textsc{poly}(L, \log m))}$ *over the randomness of the weights* w*, we have that for every* $W, W' \in \mathbb{R}^{(m\times m)^L}$ *with*

$$\|W - W^{(0)}\|_2 \leq \frac{1}{\textsc{poly}(L, \log m)} \quad \text{and} \quad \|W'\|_2 \leq \frac{1}{\textsc{poly}(L, \log m)}$$

we have that

$$\begin{aligned} F(W + W') \leq & F(W) + \langle \nabla F(W), W' \rangle + \frac{\textsc{poly}(L)\sqrt{N\, m \log m}}{\sqrt{s}} (F(W))^{\frac{1}{2}} \|W'\|_2 \\ & + \mathcal{O}\left(\frac{NL^2 m}{s}\right) \|W'\|_2, \end{aligned}$$

where $\nabla F(W) = (\nabla_{W_1} F(W), \dots, \nabla_{W_L} F(W))$.

With Lemma 2.6.4 it is possible to circumvent the usual assumption of Lipschitz smoothness, which is usually required in classical optimization theory (see also Section 2.2), that requires the the target function being at least twice differentiable.
Another interesting result which is worth mentioning is the following:

Theorem 2.6.5. *Let us denote with w all the weights of the network, with W those associated to hidden layers and with $W^{(0)}$ the initial weights (randomly chosen). With probability at least $1 - e^{-\Omega(m/\text{POLY}(N,L,\delta^{-1}))}$ over the randomness of the weights w, we have that for every $\ell \in \{1,\dots,L\}$, every $i \in \{1,\dots,N\}$ and every W such that $\|W - W^{(0)}\|_2 \leq \frac{1}{\text{POLY}(N,L,\delta^{-1})}$,*

- $\|\nabla_{W_\ell} F_i(W)\|_F^2 \leq \mathcal{O}\left(\frac{F_i(W)}{s} \times m\right)$ *and* $\|\nabla_{W_\ell} F(W)\|_F^2 \leq \mathcal{O}\left(\frac{F(W)}{s} \times m\,N\right)$,
- $\|\nabla_{W_\ell} F(W)\|_F^2 \geq \Omega\left(\frac{\max_{1\leq i\leq N} F_i(W)}{sN/\delta} \times m\right)$,

from which

$$\|\nabla F(W)\|_F^2 \leq \mathcal{O}\left(F(W) \times \frac{LNm}{s}\right) \quad \text{and} \quad \|\nabla F(W)\|_F^2 \geq \Omega\left(F(W) \times \frac{\delta m}{sN^2}\right).$$

The two statements of Theorem 2.6.5 say that when the objective function is small, then also the gradient will be, and that when it is large, also the gradient will be, provided that we are quite close to the random initialization $W^{(0)}$. This implies that we do not have saddle points and we may hope to find an actual global minimum.

2.7 Implicit Bias or Regularization

We just saw in Section 2.6 that neural networks can actually learn to interpolate the entire training dataset and indeed are also able to attain global minima, under some mild assumptions, from a theoretical point of view. Despite "learning" the entire training set, researchers found out that neural networks are not overfitting, with the consequence of a poor generalization or poor accuracy on unseen data. This observation has been made clear by Belkin and coauthors in 2019 [Bel+19] showing empirical evidence of a phenomenon they call *double-descent* that is common to many learning algorithms, such as neural networks and also linear models. In the latter case, this phenomenon was actually proven in [Has+22] by Hastie and coauthors. This is clearly visible as an extension of the standard U-shaped curve that is normally used to represent the performance of a fixed model as a function of its expressivity/capacity. This was actually the traditional landscape of the classical machine learning theory, depicted in plot A of Figure 2.3). The x-axis represents the capacity of a hypothesis class $\mathcal{H}$, while the y-axis is the risk: the dashed line denotes training risk and the solid line denotes the test or generalization risk. The goal was to find a model whose capacity could minimize the test error, reaching equilibrium between underfitting and overfitting. This is exactly the same setting of the *bias-variance tradeoff* described in Remark 2.1.8. The optimal point, in this framework, is usually realized with low training and test error, the former being generally lower then the latter (but still strictly positive). On the other hand, plot B in Figure 2.3 represents the new paradigm for models with increased capacity (e.g. by increasing the number of neurons in a network). As we can see, the test error is decreasing in two different regions, from which the name *double descent*. The novelty is represented by the so-called *interpolation threshold*, a point after which the model has completely memorized the training set and the training error is null; for this reason, this new region is only accessible with overparametrized models. Quite surprisingly, the authors of [Bel+19] found out that the test error decreases again while further increasing the capacity, reaching lower risk levels than in the standard regime. Moreover, at the same time, the training error is kept constant at zero: overparametrized model can perfectly fit the training dataset and still generalize well.

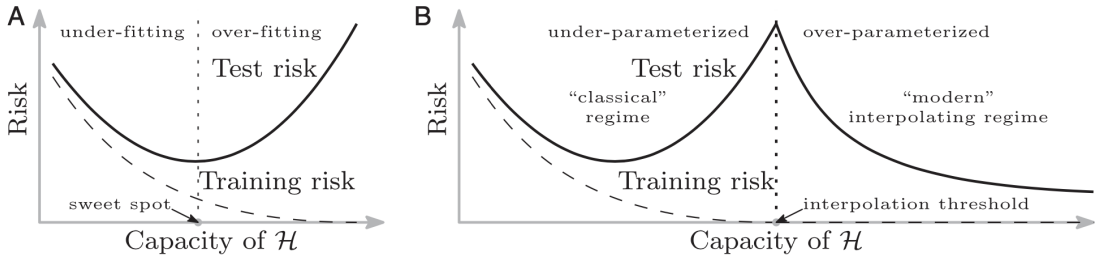


Figure 2.3: Double descent, figure from [Bel+19]. On the left, plot A, represented the situation as was considered classically. The goal is to find a model inside the hypothesis class $\mathcal{H}$ with good balance between training and test error ("sweet spot"), to avoid underfitting and overfitting. On the right, plot B, is showing double descent: overparametrized models can reach lower test error while keeping training error to zero.

Remark 2.7.1. At this point, given the phenomenon of double descent, it might become questionable whether explicit regularization is useful or needed when employing neural networks. After all, it seems that having overparametrization with implicit regularization can provide good performance. At the moment, the issue is still open. Nevertheless, it could be useful to add an explicit regularization, such as weight decay, if we have prior knowledge on the final solution or if we want to impose personal beliefs on the parameters. Such connections between regularization and priors for neural networks have been studied in many works, starting from [Gra11]. For more information and an overview of the connection between regularization and priors, we remind the reader to [Vla+19].

The crucial assumption for this newly observed phenomenon is *over-parametrization*. This has an important consequence that we have not analyzed so far: when the number of parameters of the network is larger then the number of points of the dataset, then the minimizer is *not* unique. Clearly, the value of the loss function is the same for all global minima, but one should also think about which solution is chosen out of all the solutions that minimize the loss (e.g. a solution that minimizes a specific functional among all the global minimizers of the original loss). In particular, gradient descent and stochastic gradient descent have been empirically shown to lead the training of neural networks towards "good" solution, that is with low generalization error, as it is the case for overparametrized models. Hence, it has become common in the literature to read that (stochastic) gradient descent is *biased* towards good global minima, and that is why this phenomenon took the name of *implicit bias* or *implicit regularization*.

Definition 2.7.2 (Implicit regularization (for deterministic algorithms))**.** *For the minimization Problem* (2.2)*, consider a deterministic algorithm $\mathcal{A}$ with output in Θ. We say that algorithm $\mathcal{A}$ solves Problem* (2.2) *and has* implicit regularization $G : \mathbb{D} \times \Theta \to \mathbb{R}$ *if*

$$\mathcal{A}(\mathcal{D}_n) \in \underset{\theta' \,:\, F(\theta') = \inf_\theta F(\theta)}{\operatorname{argmin}} G(\mathcal{D}_n, \theta').$$

While valid and complete explanations are still missing and are object of ongoing research for deep neural networks, it is possible to give an idea of this kind of regularization for simpler cases. Among the others, the problem was studied by Gunasekar and coauthors in 2018 [Gun+18] for the linear case, with different algorithms and for different loss functions, depending on the task, whether classification or regression under the assumptions that the number of features of the input vectors is larger than the number of samples (overparametrization regime) and the fact that the infimum for Problem (2.2) is actually a minimum ("observations are realizable" in machine learning jargon).

Theorem 2.7.3. *For the underdetermined, realizable linear system*

$$\min_{X\in\mathcal{X}} \frac{1}{2}\|AX - Y\|_2^2$$

the gradient descent algorithm (GD) for $x_0 \in \mathcal{X} = \mathbb{R}^d$, appropriate learning rate η and infinite iterations has implicit regularization $G(\mathcal{D}_n, x) = \|x - x_0\|_2$.

Corollary 2.7.4. *Let us denote with $\mathcal{G} := \{X \in \mathcal{X} : \mathbb{E}[L(f(X; \mathcal{A}(\mathcal{D}_n)), Y)] = 0\}$. If the loss L is convex and has a unique minimum, then the iterates x_t of GD converge to the global minimum that is closest to initialization x_0 in the ℓ^2-distance:*

$$x_t \underset{t\to+\infty}{\longrightarrow} \operatorname*{argmin}_{x\,\in\,\mathcal{G}} \|x - x_0\|_2.$$

Remark 2.7.5. The same results hold true for stochastic gradient descent. For this and other conclusions, we refer the interested reader to [RH19].

In order to approach the same problem for deep neural networks, people started studying another similar issue, that is *matrix completion* for neural networks, also known as matrix factorization. Matrix completion is the task of filling in the missing entries of a partially observed matrix. Given a matrix $W^\star$, the unseen entries are normally randomly selected among its elements. Without any restrictions on the number of degrees of freedom in the matrix $W^\star$, this problem is underdetermined since the hidden entries could be assigned arbitrary values. Thus we require some assumption on the matrix to create a well-posed problem, such as assuming it has maximal determinant, is positive definite, or is low-rank. In this sense, matrix completion can be viewed as a prediction problem. The natural approach to solve this problem with shallow neural networks consists of using the identity function as activation (which is equivalent to having no activation function), thus creating a *linear* network, and seeing the task as a matrix factorization: $W^\star = W_2W_1$, where W_i are the weight-matrices associated to the layers.

Definition 2.7.6 (Deep matrix factorization)**.** *Deep matrix factorization consists of parametrizing a matrix $W^\star$ as a product of L matrices:*

$$W^\star = W_L W_{L-1} \cdots W_1$$

in such a way that $W^\star$ can be seen as the product of all weight matrices of a depth-L linear neural network (i.e. a neural network with no activation functions).

If we assume dimensions $d_i \in \mathbb{N}$ for $i = 1, \dots, L$ for deep matrix factorization and define the set of missing observations as $\Xi \subset \{1, 2, \dots, d\} \times \{1, 2, \dots, d'\}$, we can write the loss function as

$$L : \mathbb{R}^{d\times d'} \to \mathbb{R}_{\geq 0}, \quad W_{L:1} \mapsto \sum_{i,j\in\Xi} \left((W_{L:1})_{i,j} - W^\star_{i,j}\right)^2,$$

where $W_{L:1} = W_L W_{L-1} \cdots W_1$, $W_i \in \mathbb{R}^{d_i \times d_{i-1}}$ with $d_L = d$ and $d' = d_0$. Note that using many matrices - many factors - amounts to minimizing an overparametrized target function. The approach entailing deep networks has been tried by Arora and coauthors in 2019 [Aro+19] where it was shown that deeper matrix factorizations (having larger L) yield more accurate recovery when $W^\star$ is low-rank and conjectured that gradient descent solutions provide an implicit regularization that cannot be described as minimization of mathematical norms (or quasi-norms). This has been finally proved in 2020 by Raznin and Cohen [RC20]. They also suggest that minimization of ranks, rather than norms, might offer more insight in the problem.

Similar negative results were found by Vardi and Shamir in 2021 [VS21], although they focused their activity on ReLU shallow networks again with mean squared loss function minimized with GD with infinitesimal step size (i.e. gradient flow, see Remark 2.2.16). We will denote with w_∞ the network weights solution to the minimization problem after convergence of GD algorithm. Perhaps surprisingly, in their quest for the function G in the sense of Definition 2.7.2, they show that already for extremely simple neural networks involving a single ReLU neuron or one thin hidden layer, implicit regularization cannot be explicitly expressed as a function of the model parameters.

Theorem 2.7.7 (Single neuron)**.** *Consider gradient flow (see Remark 2.2.16) starting from* $w_0 = 0 \in \mathbb{R}^3$ *with mean squared error* $F(w) := \sum_{i=1}^{3} (\mathrm{ReLU}(\langle x_i, w \rangle) - y)^2 = (\mathrm{ReLU}(X\, w) - y)^2$ *with* $x_i, y \in \mathbb{R}^3$ *and* $X \in \mathbb{R}^{3\times 3}$*. Let* G *be the implicit regularizer in the sense of Definition 2.7.2, that is such that for every input* (X, y) *where GD converges to* w_∞ *with* $F(w_\infty) = 0$*, we have* $w_\infty \in \operatorname{argmin}_w G(w)$ *such that* $F(w_\infty) = 0$*. Then* G *is constant on* $\mathbb{R}^3 \setminus \{0\}$*.*

Remark 2.7.8. Theorem 2.7.7 implies that implicit regularization is trivial or, however, not feasible, for single neuron networks. The result can be generalized to dimensions d larger than 3.

Despite this negative result, in the same setting, we have that G can be approximated, within a multiplicative factor of 2, by a ℓ^2-norm:

Theorem 2.7.9. *Consider gradient descent applied to a ReLU single neuron on the objective given by mean square error, where* $\sigma : \mathbb{R} \to \mathbb{R}$ *is a monotonically non-decreasing activation function (e.g. ReLU). Assume that* w_∞ *exists and* $F(w_\infty) = 0$*. Let* $w^\star \in \operatorname{argmin}_w \|w - w_0\|_2$ *such that* $F(w) = 0$*. Then,* $\|w_\infty - w_0\|_2 \leq 2\|w^\star - w_0\|_2$*.*

Similar results are valid, under mild assumptions, also for shallow neural networks. For such networks an important implicit bias property is that gradient descent (with infinitesimal step size) enforces the differences between square norms across different layers to remain invariant. But if one analyzes implicit bias inside the region of weights having such property, then the authors in [VS21] show that the implicit regularizer G is again constant on it.

2.7.1 Early Stopping

Even if we have theoretical guarantee that global minima can be attained, this is not what is done in practice, when training is artificially stopped because of time constraints. Standard criteria for stopping are the achievement of a fixed number of epochs or simply because the loss computed on a validation/test set does not improve any longer. The last criterion is known as *early stopping* and its main purpose is that of avoiding overfitting by interrupting the training before the maximal amount of (stochastic) gradient descent iterations has been reached. While mainly used to limit the time of training, early stopping can be useful since it is not easy to distinguish the double-descent phenomenon for neural networks. This observation, already made in [Bel+19], has its main causes in the fact that networks heavily rely on stochasticity, in particular, stochastic gradient descent is quite sensible to random initialization, and on a loss function which is far from being convex. For this reason, to an increase in the number of parameters does not always immediately correspond a decrease in the training error.
Another standard regularization technique that has been widely used in machine learning is *weight decay*, which consists in adding a penalization in terms of a norm of the weights (e.g. ℓ^2-norm) to the loss function.

The effect of early stopping as a regularizer in connection with other forms of regularization was studied by Heiss and coauthors [HTW19] for a special kind of neural networks. In the paper, the authors investigate which effects weight decay and early stopping have on the approximation procedure, showing that we are actually solving a specific problem, which also includes regularization, in the associated function space. The central object of study is "wide ReLU randomized shallow NN" (wR-RSN):

Definition 2.7.10 (Randomized shallow neural networks)**.** *Let $(\Omega, \Sigma, \mathbb{P})$ be a fixed probability space and let the activation function $\sigma : \mathbb{R} \to \mathbb{R}$ be Lipschitz continuous and non-constant. Then a* randomized shallow neural network (RSN) *is defined as $\mathcal{RN} : \mathbb{R}^d \to \mathbb{R}$ such that*

$$\mathcal{RN}_{w,\omega}(x) := \sum_{k=1}^{n} w_k \sigma \left(\sum_{j=1}^{d} v_{k,j}(\omega) x_j + b_k(\omega) \right) \quad \forall\, \omega \in \Omega, \forall\, x \in \mathbb{R}^d,$$

where $n \in \mathbb{N}$ is the number of neurons, $d \in \mathbb{N}$ the input dimension, $(v_k)_{k=1}^n : (\Omega, \Sigma) \to \mathbb{R}^d$ and $(b_k)_{k=1}^n : (\Omega, \Sigma) \to \mathbb{R}$ form the weight matrix and the bias vector, sampled as i.i.d. random variables. The only trainable weights are $(w_k)_{k=1}^n$.

The theory developed will focus on neural networks that have only ReLU activation functions, for which only the output layer (without activation functions) is trained, while the input layer's weights ($v_{k,j}$ and b_k) are kept constant during training. The adjective *wide* stands for the fact that the results hold in the limit for the number of neurons n in the hidden layer going to infinity. The training loss function L is the mean squared error applied to a dataset $(x_i, y_i) \in \mathbb{R}^d \times \mathbb{R}$ of N tuples.

Remark 2.7.11. Since the optimization problem is convex in the last-layer weighs w, the gradient descent actually converges to a global minimum. This actually translates in the fact that all observations are realizable:

$$\mathbb{P}\left(\lim_{t \to \infty} \mathcal{RN}_{w_t,\omega}(x_i) = y_i \ \ \forall i = 1, \dots, N \right) = 1.$$

For the first result we would like to recall here, we need the following definitions.

Definition 2.7.12 (Ridge regularized RSN)**.** *Let $\mathcal{RN}_{w,\omega}$ be a randomized shallow network as introduced in Definition 2.7.10, L a given loss functional and $\lambda > 0$. The* Ridge regularized RSN *is defined as*

$$\mathcal{RN}_{\omega}^{*,n,\lambda} = \mathcal{RN}_{w^{*,n,\lambda}(\omega),\omega}$$

where $w^{*,n,\lambda}(\omega) \in \operatorname{argmin}_{w \in \mathbb{R}^n} L_{\mathcal{RN}_{w,\omega}} + \lambda \|w\|_2^2$ for all $\omega \in \Omega$.

Definition 2.7.13 (Weighted regression spline & spline interpolation)**.** *Let $\lambda > 0$ and $x_i \in \mathbb{R}$ for all $i = 1, \dots, N$. The smoothing* regression spline *is defined as $f^{*,\lambda} : \mathbb{R} \to \mathbb{R}$ such that*

$$f_g^{*,\lambda} \in \operatorname*{argmin}_{f \in C^2(\mathbb{R})} \left(\sum_{i=1}^{N} (f(x_i) - y_i)^2 + \lambda\, g(0) \int_{\operatorname{supp} g} \frac{(f''(x))^2}{g(x)}\, dx \right),$$

for $g : \mathbb{R} \to \mathbb{R}_{\geq 0}$. For constant function g (and other moderate conditions), we recover the "usual" natural cubic splines that are common in the literature. With spline interpolation $f^{*,0+}$ we mean the function $f^{*,0+} : \mathbb{R} \to \mathbb{R}$ obtained as

$$f^{*,0+} := \lim_{\lambda \to 0_+} f^{*,\lambda} \in \operatorname*{argmin}_{\substack{f \in C^2(\mathbb{R}), \\ f(x_i) = y_i\ \forall i = 1, \dots, N}} g(0) \int_{\operatorname{supp} g} \frac{(f''(x))^2}{g(x)}\, dx.$$

In [HTW19], it is shown that over compact sets $K \subset \mathbb{R}$ the limit in probability between Ridge regularized RSN and weighted adapted splines, penalizing second order derivative, in the Sobolev metric $d_{W^{1,\infty}(K)}$, whose associated norm is $\|f\|_{W^{1,\infty}(K)} := \max\{\sup_{x\in K}|f(x)|, \sup_{x\in K}|f'(x)|\}$, is equal to 0. Quite interestingly, the weight-function g is associated to the distribution of the initial random initialization of the weights in the RSN. For this (specific) case, this statement highlights that if a Ridge-type regression is taking place during training, then this has evident consequences on the function spaces approximated by neural networks.

In the same context, stopping the training before reaching global minima has also analogous effects on the function space. First of all, let us define the *minimum norm* RSN:

Definition 2.7.14 (Minimum norm RSN)**.** *Using the notation from Definition 2.7.10, the* minimum norm RSN *is defined as* $\mathcal{RN}^{*,n,0_+} := \mathcal{RN}_{w^{*,n,0_+}}$ *with weights*

$$w^{*,n,0_+}(\omega) := \lim_{\lambda\to 0_+} w^{*,n,\lambda}(\omega) \quad \forall \omega \in \Omega.$$

For such RSN the input provided to the output layer is given by

$$X_{i,k}(\omega) = \sigma\left(\sum_{j=1}^{d} v_{k,j}(\omega)x_{i,j} + b_k(\omega)\right) \quad \forall\, \omega \in \Omega,$$

where $v_{k,}(\omega)$ and $b_k(\omega)$ were introduced in Definition 2.7.10, $X(\omega) \in \mathbb{R}^{N\times n}$ for any $\omega \in \Omega$ and $x_{i,j}$ denotes the j^{th} component of the i^{th} (training) sample x_i. Then, it can be shown (Lemma 3.18 in [HTW19]) that time-T weights[15] $w_T^n(\omega)$ are equal to

$$w_T^n(\omega) = w^{*,n,0_+}(\omega)\left(1 - \exp\{-2TX^\top(\omega)X(\omega)\}\right), \quad \forall\, \omega \in \Omega,$$

if the gradient flow is initialized at zero. So when we stop training before convergence, we are actually introducing a Ridge-type regularization for our problem. As shown in the paper, theoretical thoughts and empirical experiments suggest that the solution obtained from running gradient flow up to time T is extremely close to the the solution of the ridge regularized problem with $\lambda = 1/(2T(e-1)$. It is also clear that for $T \to +\infty$ we have $\lim_{T\to+\infty} w_T^n(\omega) = w^{*,n,0_+}(\omega)$ for all $\omega \in \Omega$, that is the time-T solution weights converge to the minimum norm solution weights. The regularizer effect introduced by early stopping also enters the phenomenon of what is called *implicit regularization.*

[15] Time-T weights are associated with $\tau = T/\gamma$ steps of the Euler discretization scheme which is used in gradient descent approximating the gradient flow (Remark 2.2.16), where $\gamma > 0$ is the learning step size. Results in the paper are generally shown for gradient flow, but gradient descent with sufficiently small learning rate can be made arbitrarily close to it.

Chapter 3

Consistent Recalibration Models for Equities

The idea that we are going to pursue in the following takes a cue from the same codebook exploited in [KK15], but we develop it around the framework of affine models, which provides great flexibility and closed form formulas for derivatives' pricing. The same approach has been clearly outlined for the first time by Anja Richter and Josef Teichmann in 2017 [RT17] for discrete time, while we will consider the more general continuous time settings. This chapter is based on [GT21].

The key idea which allows to build a bridge between the affine world and the HJM philosophy treated so far in Chapter 1 is represented by the fact that the solution to Riccati ODEs stemming from the affine model are equal, as we will see, to the (time-)integral of the codebook. From this very simple intuition, which comes from affine process' theory, we can then put into motion the codebook as already done by Kallsen and Krühner. Then, at every fixed point in time, we will be able to originate a *tangent* affine model (introduced in Definition 1.1.1) which is free of arbitrage, ruling out static arbitrage. At the same time, by requiring appropriate restrictions on the drift, analogously to the HJM drift condition, and on the short end of the codebook, as the spot condition, we will get rid of dynamic arbitrage.
Once these prerequisites are dealt with, we can shed light on the advantages of this approach.

- In first place, pricing is at hand thanks to Fourier transform techniques, which are applicable in the context of affine processes.
- In second place, calibration is also quite easily provided: in order to keep constant parameters, many market models are by their very definition high dimensional, leading to very delicate (or unstable) calibration procedures. We will relax this hypothesis by considering some of the parameters of the model as state variables, thus authorizing their evolution in time. Nonetheless, this will not break the dynamic constraints we are bound to respect: such parameters' changes will be balanced by a suitable Hull-White extension, without affecting the marginal distributions of the other (original) state variables. For this reason, this kind of model has been called *consistent recalibration model* (see also Definition 1.1.2). In a few words, the recalibration procedure will only contribute to changing model states (which are indeed expected to evolve), without giving the possibility for any arbitrage and without leaving the selected model.

- In third place and finally, pricing and calibration are made easy because in this way we are boiling down the handling of an unwieldy infinite dimensional object, that is the equation governing the dynamics, to a local finite dimensional one.

That is why we think this approach should deliver a considerable improvement in equity market models.

As just stated, *calibration* is one of the most appealing feature of this approach. But what are we actually meaning with it? In general, calibration consists in finding the "best" model among a larger set, a pool, in the sense to be able to reproduce the variables observed in reality. If we had to deal with data in the form of historical series, we would see this problem as a statistical inference problem, trying to find the model, defined by its parameters, that matches the observed data, such as different price time series, under the physical measure (usually denoted by $\mathbb{P}$); while if we deal with current market data, we are interested in replicating the current observed data, such as price or implied volatility surfaces, under an equivalent (local) martingale measure (usually denoted by $\mathbb{Q}$). From what we have already said in the previous sections, it is clear that we are considering the second formulation, which is mathematically translated in finding the solution to an inverse problem.

It is the goal of this chapter to show how calibration can take place benefiting from the CNKK approach in the setting of Consistent Recalibration Models, i.e. by considering tangent affine models. Basically speaking, this amounts to storing the information of a non-linear drift operator in a neural network in an optimal way, when the time evolution is locally mimicking a dynamically changing affine model. An important article that is dealing with similar issues is [CMN17]. In this work, the authors propose an algorithm based on Monte Carlo simulations to generate implied volatility samples that are consistent with present and past observations and then compare this method with others to tackle the problem of minimal-variance portfolio choice. The steps required for the estimation procedure, both *static*, to avoid static arbitrage, and *dynamic*, to avoid dynamic arbitrage, are quite involved and demand some tricks or simplifications. In our view, the method we present here is a simpler way to consistently construct term structure dynamics, which do *not* come from finite dimensional realizations.

Actually this information, which is stored in the drift, corresponds to solving an inverse problem or a calibration problem, see [CKT20] and the references therein for a general background of this problem: more precisely, it is the inversion of the above mentioned pricing operators given the current market state of the underlyings' price and the term structure of derivatives' prices. Let us outline this in case of the Lévy codebook: there the inverse problem corresponds to calculating the time-dependent Lévy triplet L_0 given the price of the underlying S_0 and the term structure of derivatives' prices. Even though the map from model characteristics to prices is usually smooth, it is due to *smoothing* properties, often hard to invert: existence, uniqueness and stability issues (in the sense of Jacques Hadamard - see Definition 1.4.1) appear. Machine learning technology and, in particular, deep learning provide one possible way to fix these issues, which otherwise require sophisticated regularization techniques, by implicit regularization. A non-exhaustive list of these methods and theoretical considerations are in Section 2.7. Among the others, one relevant example is early stopping, see e.g. [HTW19].
In other words: learning the map from derivatives' prices (given the current price of the underlying) to model characteristics L_0 and storing the information in a neural network provides an accurate map satisfying Hadamard's requirements. We shall pursue this approach not in the originally proposed way by solving a supervised learning problem, see, e.g. [Her16], but rather by storing first the information of the pricing operator in a neural network and then inverting this network,

compare here, e.g. [HMT21].
In the very same years, other applications of ML to the same problem were developed. One of these is presented in [Wie+19], where Wiese and co-authors make use of a Generative Adversarial Networks (GANs), see [GBC16] and the references therein, to learn to simulate time series of the codebook introduced by Wissel in [Wis07]. Thanks to the particular choice of the codebook, it is relatively easy to learn new plausible prices that satisfy static arbitrage constraints on discrete grids: as we recalled above in Section 1.4, non-negativity of the local volatility process is the only requirement in this case. On the other hand, the work we present here does not try to get rid of static arbitrage possibilities *alone*, but of dynamic arbitrage as well, thus answering in a more complete and satisfactory way to the need of a realistic equity option market simulator, which was the *raison d'être* of [Wie+19]. In addition, our setting deals with continuous time and strike intervals.

One could also view our current model as an unusually parameterized neural stochastic differential equation (NSDE) model, see, e.g. [CKT20] for details on this concept. NSDEs, i.e. stochastic differential equations with neural network characteristics, are a wonderful concept to construct non-parametric models, but it is quite delicate to write constraint dynamics with neural networks. Therefore we have chosen Consistent Recalibration Models with tangent affine models, where it is easier to express constraints in terms of neural networks for the drift.

The remainder of the chapter is structured as follows. In the next section, we introduce mathematically the concept of a Lévy triplet *codebook*, as shortly alluded to above, and we define consistent recalibration (CRC) models. We also briefly review affine models and embed stochastic volatility affine models in the context of CRC models, outlining some key properties of this codebook. The second section of the chapter is dedicated to one of the building blocks of the whole theory: generalized Hull-White extensions. We do not simply use the Hull-White extension, as it was exploited when first defined, for the calibration at the initial time of the term structure in the interest rate models, but we think of it as a tool that allows for recalibration of the model parameters. Further, we talk about a *generalized* extension, since we are replacing the pure drift addition typical of interest rate models with a Lévy process, which naturally encodes a greater calibration power. To make things clearer, an example is laid out in the third section, where we analyse how the generalized Hull-White extension is added to the classical Heston model in order to get a consistent recalibration model, what we call a generalized Bates model. The same example is important because a very similar version of this model has been implemented numerically. The fourth section is devoted to the CNKK equation: how it is derived, defined and how it can be seen as a generalization of the HJM equation. Section 3.5 is dedicated to the formal definition of CRC models for stochastic volatility affine models with piecewise constant model parameters for pricing stocks' derivatives. Some numerical considerations are also listed, to show what are the most relevant aspects to deal with in case of a concrete implementation. One of these points is the main subject of Section 3.6, where we discuss about calibrating the model using a neural network. Subsections are dedicated to the available literature and to the architecture and training techniques used to achieve the final result. A geometric interpretation is presented in Section 3.7, starting from theory of finite dimensional realizations and its connections with Frobenius' theorem.

Notation. The set $\mathbb{N}_0$ denotes the set of natural numbers with 0 included; $\mathbb{R}^m_+$ the real vectors in $\mathbb{R}^m$ whose components are greater than 0.
With $L(X)$ we denote the set of X-integrable predictable processes for a semimartingale X. If we talk about (X, Y) as a $(m+n)$-semimartingale, we mean that X is an $\mathbb{R}^m$-valued semimartingale and Y is an $\mathbb{R}^n$-valued semimartingale.

Whenever we apply complex logarithm of continuous functions $\mathbb{R}^n \ni u \mapsto f(u) \neq 0$, we use the normalization $\log f(0) = 0$, so that logarithms are uniquely defined.
For the sake of readability, SDE and SPDE will be used as acronyms for stochastic differential equation and stochastic partial differential equation, respectively.

3.1 Consistent Recalibration Models

As already mentioned, we take inspiration from the seminal paper of Kallsen and Krühner [KK15], but we place ourselves in a more general framework that does not necessarily rely on the infinite divisibility of the processes. Let $(\Omega, \mathcal{F}, (\mathcal{F})_{t\geq 0}, \mathbb{Q})$ denote a filtered probability space, where $\mathbb{Q}$ represents a risk-neutral measure so that discounted asset prices are supposed to be $\mathbb{Q}$-martingales. All expectations, if not differently specified, are taken with respect to this probability measure and are denoted by $\mathbb{E}$. We consider an adapted (multivariate) stochastic process $X := (X_t)_{t\geq 0}$ taking values in $\mathbb{R}^n$, which can be considered as logarithm of price processes.
We assume that call options of any strike and maturity are liquidly traded and denote the time t value of a call option with maturity T and strike K by $C_t(T, K)$. Having liquid market prices for all maturities and strikes[1] translates, in mathematical terms, in having the marginal distributions of several (at most n) underlying processes (under the pricing measure). Similar to [KK15], we could at this point require that the given marginal distributions of X are infinitely divisible, that the characteristic functions are absolutely continuous with respect to time and define the *codebook* of our model as a "forward" Lévy exponent and then define dynamics for such (infinite dimensional) codebook. But rather than focusing on the joint behaviour of X and the codebook, whose dynamics are expressed in function of a generic semimartingale M, as done in [KK15], we exploit the intuition behind the choice of such codebook, since it provides easier conditions to avoid dynamic and static arbitrage, but we build around it a new framework.

For this reason, we conveniently define consistent recalibration models as models that keep *consistency*, which means that future realizations will be in a neighbourhood of the current state that can always be reached with positive probability, and that are *analytically tractable*, thus looking as finite factor models instantaneously. This is reached by introducing stochastic parameters, whose dynamics could be extrapolated by market data, and by means of a Hull-White extension, used to compensate the stochastic updates in the parameters, while leaving the marginal distributions of the state variables unchanged.

We start defining the set of functions that is the base for our theory:

Definition 3.1.1 (Γ_n)**.** *The set Γ_n denotes the collection of continuous functions $\eta : \mathbb{R}^n \times \mathbb{R}_{\geq 0} \to \mathbb{C}$ such that there exists a càdlàg process Z with independent increments and finite exponential moments $\mathbb{E}[\exp((1+\varepsilon)\|Z_T\|)] < \infty$ for all $T \geq 0$ and for some $\varepsilon > 0$ satisfying*

$$\mathbb{E}[\exp(\mathrm{i}\,\langle u, Z_T\rangle)] = \exp\left(\mathrm{i}\,\langle u, Z_0\rangle + \int_0^T \eta(u, r)\,dr\right) \tag{3.1}$$

for $u \in \mathbb{R}^n$.

Remark 3.1.2. Requiring that for some $\varepsilon > 0$, $\mathbb{E}[\exp((1+\varepsilon)\|Z_T\|)] < \infty$ implies that we can extend the left hand side of (3.1) to the strip $-\mathrm{i}[0,1]^n \times \mathbb{R}^n$, thus we could choose, for example, $u = -\mathrm{i}$.

[1] In practice, and for our ensuing numerical algorithm, this is never the case.

Remark 3.1.3. All functions $\eta \in \Gamma_n$ are necessarily of Lévy-Khintchine type at the short end ($r = 0$ in (3.1)). Note that by doing so, we are extending the definitions given in [CN12] and [KK15] since we only assume the function η to be of Lévy-Khintchine form at the short end.

Remark 3.1.4. Often, elements in Γ_n are subject to additional no-arbitrage constraints in order to satisfy, for example, the martingale property for both the price processes $S = \exp(X)$ and call options $C_t(T, K)$ for all $T, K > 0$ (see Theorem 3.7 in [KK15]). For instance, if we consider $X = (X^i)_{i=1}^d$ as being a log-price process for some k, we also assume $\exp(X^k)$ is a martingale, which is equivalent to state that $\eta(-\mathrm{i}e_k, r) = 0$ with e_k being the k-th basis vector of $\mathbb{R}^n$, i.e. $\langle e_k, X_T \rangle = X_T^k$. We assume tacitly that such conditions are imposed if necessary. Notice in case of a components of X corresponding to interest rates we do not need to impose such a condition (since we do not need martingality).

We can think of the set Γ_n as a chart, in the language of geometry, or codebook, in the language of mathematical finance, for all liquid market prices at one instant of time. If we want to consider their time evolution, we had better define Γ_n-valued processes:

Definition 3.1.5 (Γ_n-valued semimartingale)**.** *Let $(\Omega, \mathcal{F}, (\mathcal{F}_t)_{t\geq 0}, \mathbb{Q})$ be a filtered probability space. A stochastic process η is called a* Γ_n-valued semimartingale *if $(\eta_t(u,T))_{0\leq t\leq T}$ is a complex-valued semimartingale for $T \geq 0$ and $u \in \mathbb{R}^n$ and if*

$$\big((u, r) \mapsto \eta_t(u, r + t)\big) \in \Gamma_n \,.$$

In particular, all trajectories are assumed to be càdlàg.

Definition 3.1.6 (Regular decomposition)**.** *We say that η allows for a* regular decomposition *with respect to a d-dimensional semimartingale M if there exist predictable processes $(\alpha_t(u,T))_{0\leq t\leq T}$ taking values in $\mathbb{C}$ with $\alpha_t(0,T) = 0$ for all $0 \leq t \leq T$ and $(\beta_t(u,T))_{0\leq t\leq T}$, $\mathbb{C}^d$-valued, with $\beta_t^i(0,T) = 0$ for all i and all $0 \leq t \leq T$ and for $T \geq 0$ and $u \in \mathbb{R}^n$ such that*

$$\eta_t(u,T) = \eta_0(u,T) + \int_0^t \alpha_s(u,T)\, ds + \sum_{i=1}^d \int_0^t \beta_s^i(u,T)\, dM_s^i \tag{3.2}$$

for $0 \leq t \leq T$, and $\left(\sqrt{\int_t^T \|\beta_t(u,r)\|^2\, dr}\right)_{t\geq 0} \in L(M)$.

In view of the two new definitions, we can also generalize the condition expressed in (3.1):

Definition 3.1.7 (Conditional expectation condition)**.** *Let $(\Omega, \mathcal{F}, (\mathcal{F}_t)_{t\geq 0}, \mathbb{Q})$ be a filtered probability space, then we say that a tuple (X, η) of an n-dimensional semimartingale X and of a Γ_n-valued semimartingale η satisfies the* conditional expectation condition *if*

$$\mathbb{E}\left[\exp(\mathrm{i}\,\langle u, X_t\rangle)\middle|\,\mathcal{F}_s\right] = \exp\left(\mathrm{i}\,\langle u, X_s\rangle + \int_s^t \eta_s(u,r)\, dr\right) \tag{3.3}$$

for $0 \leq s \leq t$.

At this point, the link of the whole theory to its discrete-time counterpart exposed in [RT17] becomes even clearer:

Definition 3.1.8 (Forward and process characteristics)**.** *Let X be an adapted semimartingale taking values in $\mathbb{R}^n$ and η a Γ_n-valued semimartingale with $\eta_s(0,t) = 0$ for all $0 \leq s \leq t$ and for*

which the conditional expectation condition (3.3) *is satisfied. Then the process* η *is called* forward characteristic process of X. *Analogously, the process denoted by* κ_s^X *and that coincides with the short end of the forward characteristics of* X, *i.e.* $\eta_{s-}(\cdot, s)$, *with* $\kappa_s^X(0) = 0$ *for all* $s \geq 0$ *is said* (process) characteristic of X.

Remark 3.1.9. Note that both processes are uniquely defined (up to a $d\mathbb{Q} \otimes dt$-nullset):

1. The normalization $\eta_s(0,t) = 0$ for all $0 \leq s \leq t$ ensures that the map $u \mapsto \eta_s(u,t)$ is continuous and uniquely defined through the use of the complex logarithm.

2. Since the adapted process $\left(\exp\left(\mathrm{i}\,\langle u, X_s\rangle - \int_0^s \kappa_r^X(u)\,dr\right)\right)_{s\geq 0}$ is a local martingale (see Theorem 3.1.12 below) and $\kappa_r^X(0) = 0$ for any $r \geq 0$, uniqueness follows from Lemma A.5 in [KK15].

Definition 3.1.10 (Term structure for derivatives)**.** *We call the tuple* (X, η) *of an* n*-dimensional semimartingale* X *and of its* Γ_n*-valued forward characteristic process* η *a* term structure model for derivatives' prices.

In mathematical finance, we are often interested in risk-neutral models, that is models for which, under a suitable probability measure, the price of liquidly traded assets are (local) martingales.

Definition 3.1.11 (Risk-neutral model)**.** *The term structure model for derivatives provided by the tuple* (X, η) *is said to be* risk-neutral *if the corresponding stock prices* S *and all European call option prices* $C(T,K)$ *for* $T \geq 0$ *and* $K > 0$ *are (local) martingales.*

With the following theorems, we are able to characterize which processes η can be considered forward processes, given the existence of a regular decomposition. Discrete versions of the same theorems are given in [RT17], while proofs for the continuous case under examination are in [KK15].

Theorem 3.1.12. *Let* $(\Omega, \mathcal{F}, (\mathcal{F}_t)_{t\geq 0}, \mathbb{Q})$ *be a filtered probability space together with a tuple* (X, η) *of an* n*-dimensional semimartingale* X *and of a* Γ_n*-valued semimartingale* η *satisfying the* conditional expectation condition*, then*

- *the differentiable, predictable characteristic* κ^X *of the* n*-dimensional semimartingale* X *exists and is given by* $\kappa_t^X(u) = \eta_{t-}(u,t)$ *(usually called* short end *or* spot condition*) for* $t \geq 0$ *and* $u \in \mathbb{R}^n$, *i.e. the process*

$$\exp\left(\mathrm{i}\,\langle u, X_t\rangle - \int_0^t \eta_{s-}(u,s)\,ds\right) \tag{3.4}$$

  is a local martingale.

- *If* η *allows for a regular decomposition* (3.2) *with respect to a* d*-dimensional semimartingale* M, *then the* (HJM) drift condition

$$\int_t^T \alpha_t(u,r)\,dr = \eta_{t-}(u,t) - \kappa_t^{(X,M)}\left(u, -\mathrm{i}\int_t^T \beta_t(u,r)\,dr\right) \tag{3.5}$$

  holds for $0 \leq t \leq T$ *and* $u \in -\mathrm{i}[0,1]^n \times \mathbb{R}^n$.

It could be useful for the reader having in mind the following expression for the forward characteristics of the $(n+d)$-semimartingale (X, M): for all t such that $0 \le t \le T$, we have

$$\exp\left(\eta_t^{(X,M)}(u, v;\, T)\right) = \mathbb{E}\left[\exp\left(\mathrm{i}\,\langle u, (X_T - X_t)\rangle + \mathrm{i}\,\langle v, M_T - M_t\rangle\right)\middle|\, \mathcal{F}_t\right]$$

from which it is easier to derive the expression for $\kappa_t^{(X,M)}$ for $0 \le t \le T$.

Theorem 3.1.13. *Let $(\Omega, \mathcal{F}, (\mathcal{F}_t)_{t\ge 0}, \mathbb{Q})$ be a filtered probability space together with a tuple (X, η) of a n-dimensional semimartingale X and a Γ_n-semimartingale η. Furthermore, assume that η allows for a regular decomposition* (3.2) *with respect to a d-dimensional semimartingale M such that the predictable characteristics of X satisfy* (3.4) *and such that the drift condition* (3.5) *holds, then the* conditional expectation condition *holds true.*

Corollary 3.1.14. *Let $(\Omega, \mathcal{F}, (\mathcal{F}_t)_{t\ge 0}, \mathbb{Q})$ be a filtered probability space together with a tuple (X, η) where X is a n-dimensional semimartingale and η, Γ_n-semimartingale, satisfies the* conditional expectation condition*. Moreover, assume that η allows for a regular decomposition* (3.2) *with respect to a d-dimensional semimartingale M and that the processes X and M are locally independent*[2]*, i.e.*

$$\kappa_t^{X,M}(u_1, u_2) = \kappa_t^X(u_1) + \kappa_t^M(u_2) \tag{3.6}$$

for $u_1 \in \mathbb{R}^n$ and $u_2 \in \mathbb{R}^d$. Then

$$\int_t^T \alpha_t(u, r)\, dr = -\kappa_t^M\left(-\mathrm{i}\int_t^T \beta_t(u, r)\, dr\right)$$

for $0 \le t \le T$ and $u \in \mathrm{i}[0,1] \times \mathbb{R}^n$ and, furthermore, the conditional expectation condition (3.3) *rewrites as*

$$\mathbb{E}\left[\exp\left(\int_s^t \eta_{r-}(u, r)\, dr\right)\middle|\, \mathcal{F}_s\right] = \exp\left(\int_s^t \eta_s(u, r)\, dr\right)$$

for $0 \le s \le t$.

Proof. To obtain the new form of the conditional expectation condition is enough to use (3.4). □

Remark 3.1.15 (Risk-neutral model). The two previous theorems basically ratify the equivalence between the conditional expectation condition on one hand, and the short end and drift conditions on the other. In [KK15], since they assume η being of Lévy-Khintchine type for all times, it is possible to show (Theorem 3.7) equivalence with the fact of $S = \exp(X)$ and $C_t(T, K)$ being martingales. This implies a *risk-neutral* model and, by the fundamental theorem of asset pricing, no arbitrage opportunities.
This is not given for free in our settings, but requires additional assumptions (see Remark 3.1.4). For example, requiring that $S = \exp(X)$ is a 1-dimensional martingale is equivalent to the condition $\eta_s(-\mathrm{i}, t) = 0$ for all $0 \le s \le t$. In this case, indeed, we can write

$$\mathbb{E}\left[e^{\mathrm{i}u(X_t - X_s)}\middle|\, \mathcal{F}_s\right] = \exp\left(\int_s^t \eta_s(u, r)\, dr\right)$$

and for $u = -\mathrm{i}$ we have $\mathbb{E}\left[e^{X_t - X_s}\middle|\, \mathcal{F}_s\right] = 1$ for all $0 \le s \le t$. In the following, we will anyway assume this condition whenever needed.

Remark 3.1.16. Forward characteristics encode the term structure of distributions of increments of the stochastic process X, i.e. for $0 \le t \le T$, the distributions of $X_T - X_t$ conditional on the information $\mathcal{F}_t$ at time t. Notice that there is redundant information in processes of forward characteristics, which then translates into the drift conditions (3.5).

[2] See [KK15] for a rigorous definition.

3.1.1 Affine processes

In this subsection, we introduce affine processes and give some important results on their forward characteristic processes. Moreover, since we are mainly interested in affine stochastic volatility models, we will state some properties for their particular case.

Let D be a non empty Borel subset of $\mathbb{R}^d$ to which we associate the set $\mathcal{U} := \{u \in \mathbb{C}^d : \sup_{x\in D} \operatorname{Re}\langle u, x\rangle < \infty\}$.

Definition 3.1.17 (Affine process)**.** *An affine process is a time-homogeneous Markov process $(X_t, \mathbb{P}^x)_{t\geq 0, x\in D}$ with state space D, whose characteristic function is an exponentially affine function of the state vector. This means that its transition kernel p_t satisfies the following:*

- *it is stochastically continuous, i.e. $\lim_{s\to t} p_s(x,\cdot) = p_t(x,\cdot)$ weakly on D for every $t \geq 0$ and $x \in D$, and*
- *its Fourier-Laplace transform has exponential affine dependence on the initial state. This means that there exist functions $\Phi : \mathcal{U} \times \mathbb{R}_{\geq 0} \to \mathbb{C}$ and $\psi : \mathcal{U} \times \mathbb{R}_{\geq 0} \to \mathbb{C}^d$ with*

$$\mathbb{E}_x\left[e^{\langle u, X_t\rangle}\right] = \Phi(u,t)e^{\langle x, \psi(u,t)\rangle}, \tag{3.7}$$

for all $x \in D$, $u \in \mathcal{U}$ and $t \in \mathbb{R}_{\geq 0}$.

Remark 3.1.18. The existence of a filtered probability space $(\Omega, \mathcal{F}, (\mathcal{F}_t)_{t\geq 0})$ is already included by the notion of Markov process (see [KST11]).

Remark 3.1.19. The definition we gave is not the original provided by Duffie *et al.* in [DFS03] but a slightly more general one: the right hand side of (3.7) is equal to $e^{\phi(u,t)+\langle x,\psi(u,t)\rangle}$ as long as we know that $\Phi(u,t) \neq 0$, but this can be shown ([KST13]) and not postulated (as done in [DFS03]). From now on, we assume $\Phi(u,t) = \exp(\phi(u,t))$. A priori we do not even have a unique definition of the functions ψ and ϕ, but we can assume the normalization $\phi(u,0) = 0$ and $\psi^i(u,0) = u$ for all $u \in \mathcal{U}$ and all $i = 1,\dots,d$, which makes the functions unique.

In this subsection we build a generic example for term structure models for derivatives' prices. Therefore, we define an affine stochastic volatility model:

Definition 3.1.20 (Affine stochastic volatility model)**.** *Let us consider a proper convex cone $C \subset \mathbb{R}^m$ (the* stochastic covariance structures*). An* affine stochastic volatility model *is a time-homogenous affine (Markov) process (X, Y) taking values in $\mathbb{R}^n \times C$ relative to some filtration $(\mathcal{F}_t)_{t\geq 0}$ and with state space $D = \mathbb{R}^n \times C$ such that*

- *it is stochastically continuous, that is, $\lim_{s\to t} p_s(x,y,\cdot) = p_t(x,y,\cdot)$ weakly on D for every $t \geq 0$ and $(x,y) \in D$, and*
- *its Fourier-Laplace transform has exponential affine dependence on the initial state. This means that there exist (deterministic) functions $\phi : \mathcal{U} \times \mathbb{R}_{\geq 0} \to \mathbb{C}$ and $\psi_C : \mathcal{U} \times \mathbb{R}_{\geq 0} \to \mathbb{C}^m$ with*

$$\mathbb{E}\left[e^{\langle u, X_t\rangle + \langle v, Y_t\rangle}\middle|\mathcal{F}_s\right] = e^{\phi(u,v,t-s)+\langle u, X_s\rangle + \langle \psi_C(u,v,t-s), Y_s\rangle}, \tag{3.8}$$

for all $(x,y) \in D$, $0 \leq s \leq t$ and $(u,v) \in \mathcal{U}$, where

$$\mathcal{U} := \{(u,v) \in \mathbb{C}^{n+m} \mid e^{\langle u,\cdot\rangle + \langle v,\cdot\rangle} \in L^\infty(D)\},$$

and the normalizations $\phi(u,v,0) = 0$ and $\psi_C^i(u,v,0) = v$ for all $(u,v) \in \mathcal{U}$ and $i = 1,\dots,m$.

Remark 3.1.21. In line with literature on affine processes there is a $\mathbb{C}^{n+m}$-valued function ψ, whose projection onto the X-directions is u, as exemplified in (3.8). Whence we only need the projection in the C-directions, which we denote by ψ_C. This corresponds to a standard assumption if we consider X as a price process: if we move X_s by a quantity x, then also X_t gets shifted by the same amount.

Functions ϕ and ψ_C are important because they allow the introduction of the so-called *functional characteristics* (because of complete characterisation) of the affine process (X, Y). We define

$$F(u,v) := \frac{\partial \phi}{\partial t}(u,v,t)\Big|_{t=0^+}, \qquad R_C(u,v) := \frac{\partial \psi_C}{\partial t}(u,v,t)\Big|_{t=0^+} \tag{3.9}$$

for all $(u, v) \in \mathcal{U}$ and continuous in $(0, 0)$ (see [KST13]). Equations (3.9) are called *Riccati equations*.

More in general, we can also define the *generalized Riccati equations*[3] and prove the following theorem (from [Kel08]):

Theorem 3.1.22. *Suppose that $|\phi(u, w, T)| < \infty$ and $\|\psi_C(u, w, T)\| < \infty$ for some $(u, w, T) \in \mathcal{U} \times \mathbb{R}_{\geq 0}$. Then for all $t \in [0, T]$ and v with $\operatorname{Re} v \leq \operatorname{Re} w$ the derivatives* (3.9) *exist. Moreover, for $t \in [0, T)$, ϕ and ψ_C satisfy the generalized Riccati equations:*

$$\frac{\partial}{\partial t}\phi(u,v,t) = F(u, \psi_C(u,v,t)), \qquad \phi(u,v,0) = 0 \tag{3.10a}$$

$$\frac{\partial}{\partial t}\psi_C(u,v,t) = R_C(u, \psi_C(u,v,t)), \qquad \psi_C(u,v,0) = v. \tag{3.10b}$$

We can also derive the following proposition:

Proposition 3.1.23. *Let (X, Y) be a homogenous affine process taking values in $D = \mathbb{R}^n \times C$, then for $t \leq T$ we have that*

$$\phi(u,0,t) = \int_0^t F(u, \psi_C(u,0,s))\, ds$$

and

$$\psi_C(u,0,t) = \int_0^t R_C(u, \psi_C(u,0,s))\, ds,$$

where $(u, v) \mapsto F(u, v)$ and $(u, v) \mapsto \langle R_C(u, v), y\rangle$ are of Lévy-Khintchine form.

Proof. While the first part automatically comes from the definition of generalized Riccati equations, the second can be found in [Kel08]. □

Corollary 3.1.24. *Let (X, Y) be a homogeneous affine process taking values in $D = \mathbb{R}^n \times C$ and assume that the finite moment condition $\mathbb{E}[\exp((1 + \varepsilon)\|X_t\|)] < \infty$ holds true for some $\varepsilon > 0$, then for $0 \leq t \leq T$*

$$\eta_t(-\mathrm{i}u, T) := F(u, \psi_C(u,0,T-t)) + \langle R_C(u, \psi_C(u,0,T-t)), Y_t\rangle$$

defines a Γ_n-valued semimartingale and the tuple (X, η) satisfies the conditional expectation condition.

[3] The name comes from the fact that they boil down to the well-known Riccati equations when (X, Y) is a diffusion process.

Proof. The proof follows from the previous proposition and simple algebraic operations. For any $0 \leq t \leq T$, we have

$$\begin{aligned}
\mathbb{E}\left[e^{\langle u, X_T\rangle}\middle|\mathcal{F}_t\right] &= e^{\phi(u,0,T-t)+\langle u,X_t\rangle+\langle \psi_C(u,0,T-t),Y_t\rangle} \\
&= e^{\langle u,X_t\rangle+\int_0^{T-t} F(u,\psi_C(u,0,r))\,dr+\left\langle \int_0^{T-t} R_C(u,\psi_C(u,0,r))\,dr, Y_t\right\rangle} \\
&= e^{\langle u,X_t\rangle+\int_0^{T-t} F(u,\psi_C(u,0,r))+\langle R_C(u,\psi_C(u,0,r)),Y_t\rangle\,dr} \\
&= e^{\langle u,X_t\rangle+\int_t^{T} F(u,\psi_C(u,0,r-t))+\langle R_C(u,\psi_C(u,0,r-t)),Y_t\rangle\,dr} \\
&= e^{\langle u,X_t\rangle+\int_t^{T} \eta_t(-\mathrm{i}u,r)\,dr}.
\end{aligned}$$

□

In interest rate theory, where affine models proved to be a powerful tool, Hull-White extensions is realized by making the drift term time dependent and plays the fundamental role of allowing the calibration of an initial yield curve to the prescribed model. This will be the topic of the next section.

We see in the following some applications of such theory.

Example 3.1.25. **Deterministic term structure of forward characteristics:** Deterministic forward term structure models correspond to time-dependent Lévy processes. More precisely, let (X, η) be a tuple satisfying the conditional expectation condition and assume that η is a deterministic, then X is an additive process and $\eta_t(u,T) = \eta_0(u,T)$ is of Lévy-Khintchine form for every $T \geq 0$ (compare, for example, with Definition 3.1.6). A particular example would be any time-dependent Lévy model.

Example 3.1.26. **Interest rate models:** If the process X is one-dimensional, pure-drift and absolutely continuous with respect to Lebesgue measure, then we fall in the case treated in Corollary 3.1.14 and we have

$$\int_t^T \alpha_t(u,r)\,dr = -\kappa_t^M\left(-\mathrm{i}\int_t^T \beta_t(u,r)\,dr\right),$$

but also

$$\mathbb{E}\left[\exp\left(-\int_t^T \eta_{s-}(u,s)\,ds\right)\middle|\mathcal{F}_t\right] = \exp\left(-\int_t^T \eta_t(u,r)\,dr\right), \tag{3.11}$$

from which we obtain

$$uX_t = uX_0 - \int_0^t \eta_{s-}(u,s)\,ds.$$

Equation (3.11) is also well-known in interest rate theory: if we denote with $P(t,T)$ the price of a risk-less zero coupon bond, with $f(t,T)$ the forward rate yield prevailing at t for T and with $r(t)$ the short time interest rate at t, then we have

$$P(t,T) = \mathbb{E}\left[e^{-\int_t^T r(s)\,ds}\middle|\mathcal{F}_t\right] = e^{-\int_t^T f(t,S-t)\,dS}$$

for $0 \leq t \leq T$ and $u \in \mathbb{R}$. Moreover, if we assumed M being a Brownian motion, then we would have $\kappa_t^M(u) = -u^2/2$ and, thus,

$$\int_t^T \alpha_t(u,r)\,dr = -\frac{1}{2}\left(\int_t^T \beta_t(u,r)\,dr\right)^2,$$

from which, differentiating both sides with respect to T, we obtain the well-known HJM drift condition

$$\alpha_t(u,T) = -\beta_t(u,T)\int_t^T \beta_t(u,r)\,dr.$$

Notice that $(\eta_{s-}(u,s))_{s\geq 0}$ is linear in u, since X is pure drift.

3.2 Generalized Hull-White extension

Hull-White extension of Vašíček model was performed adding a time-dependent constant drift to the equation for the short term interest rate r, in order to have a perfect match with the current ($t = 0$) term structure of forward rates and, thus, to enhance calibration.

In this case, we will take a more general approach and will encode the extension, represented by a Lévy process, in the constant part of the affine process (responsible for the state-independent characteristics thereof). In other words, the function F will become consequently time-inhomogeneous, thus modifying the forward characteristics of the process X.

Corollary 3.2.1. *Let $(\widetilde{X}, Y)$ be a time-*inhomogenous*, homogeneous càdlàg affine process taking values in $\mathbb{R}^n \times C$ with time-dependent continuous $T \mapsto F_T$, and assume that the finite moment condition $\mathbb{E}\left[\exp((1+\varepsilon)\|\widetilde{X}_t\|)\right] < \infty$ holds true for some $\varepsilon > 0$, then for $0 \leq t \leq T$*

$$\widetilde{\eta}_t(-\mathrm{i}u, T) := F_T(u, \psi_C(u, 0, T-t)) + \langle R_C(u, \psi_C(u, 0, T-t)), Y_t\rangle$$

defines a Γ_n-valued semimartingale and the tuple $(\widetilde{X}, \eta)$ satisfies the conditional expectation condition.

Remark 3.2.2. Here time-inhomogenous, homogenous affine processes appear as generalization of the approaches in [CN12] and [KK15] (CNKK-approach), since we can calibrate a large variety of (virtually, any) initial term structure into $t \mapsto F_t$.

Remark 3.2.3. Although we are only modifying the forward characteristic process of X, the process Y, which is Markov in its own filtration, remains the same. This keeps the transformation simple and the processes tractable, since it does not affect the stochastic covariance structure.

The above structure increases the calibration properties of the original model. In addition, since the Lévy process is allowed to change over time, we could calibrate it to match market conditions for other instant of times (apart the initial time).

The main consequence of having such a generalized Hull-White extension is that we could compensate fluctuations (i.e. *recalibrations*) in the original model's parameters with a calibration of the Lévy process, so to keep the price/volatility surface unchanged. In other words, we could consider the parameters of the original model as *state variables*. When this is possible, we will talk about a model that satisfies the ***consistent recalibration property***.

A valid question, at this point, would be to know when this is possible. Are there conditions that we could impose or verify to make sure that such a compensating mechanism can always happen?

Let us denote with $(\nu_t^L)_{t\geq 0}$ the Lévy measure of the time-dependent Lévy process L, with p_t and Z_t the set of parameters and state variables belonging to the time-homogeneous model at time[4] t, respectively, and with ν^{p_t, Z_s} the Lévy measure that has the same expressive capability

[4]Since parameters can be considered as state variables, they are allowed to change in time.

as the original model[5], where we made explicit the dependence on the parameters p_t and state variables Z_s, for $s \leq t$.

Proposition 3.2.4. *Let us assume that the stochastic parameter process $(p_t)_{t\geq 0}$ has trajectories, whose total variation is bounded by a deterministic constant, take values on the compact set Θ of admissible parameters. Moreover, assume that $p \mapsto \nu^{p,\cdot}$ is continuously differentiable and that p remains constant whenever Z leaves a prespecified compact set K. Then, if for all $t \geq 0$ we have the non-negativity condition*

$$\nu_t^L \geq \sum_{0\leq s\leq t} \nu_t^{p_s, Z_{s^-}} - \nu_t^{p_{s^-}, Z_{s^-}}, \tag{3.12}$$

the consistent recalibration property holds.

Proof. The proof is done by induction on the jumping times of the parameter process p. For more details, see [RT17]. □

As already mentioned above, this add-on will transform the functional characteristic F in a time-dependent function and it might be worth noticing how this happens in practice.

Using the same notation introduced in [RT17], we can define F_T as it appears in Corollary 3.2.1 adding a new time-dependent function μ:

Definition 3.2.5 (Inc^D)**.** *Let Z be a generic stochastic process with values in the (state) space D, such that all increments ΔZ_s satisfy $z + \Delta Z_s \in D$ for any $z \in D$ and any $s \geq 0$. We denote by Inc^D the set which contains all continuous functions $\mu : \mathcal{U} \times \mathbb{R}_{\geq 0} \to \mathbb{R}$ of the type $\mu(u,t) := \log \mathbb{E}[\exp(\langle u, \Delta Z_t\rangle)]$ for which $\mu(0,t) = 0$ for all $t \geq 0$.*

In other words, we are adding to the "old" F the cumulant generating function of the process $(\Delta Z_s)_{s\geq 0}$, that is $F_s(u,v) := F(u,v) + \mu(u,v,t-s)$ for all $u \in \mathcal{U}$ and $t \geq s \geq 0$. This will become even clearer in the following, when we will specify our consistent recalibration model.

Analogously to what already done, we can define $\widetilde{\phi}$ and $\widetilde{\psi}$ as the time-inhomogeneous versions of ϕ and ψ as solutions to the time-inhomogeneous version of the Riccati equations. In particular, for stochastic volatility affine processes, similarly to Theorem 3.1.22, for $s \leq t$ we can write (compare with [RT17]):

$$\frac{\partial}{\partial t}\widetilde{\phi}(u,v;s,t) = F_t(u, \widetilde{\psi}_C(u,v;s,t)), \qquad \widetilde{\phi}(u,v;0,0) = 0 \tag{3.13a}$$

$$\frac{\partial}{\partial t}\widetilde{\psi}_C(u,v;s,t) = R_C(u, \widetilde{\psi}_C(u,v;s,t)), \qquad \widetilde{\psi}_C(u,v;0,0) = v, \tag{3.13b}$$

with $\widetilde{\psi}_C(u,v;s,t) = \psi_C(u,v;t-s)$. At this point, it is also possible to rewrite the expression for the *forward characteristics* of the time-inhomogeneous process $\widetilde{X}$ as

$$\int_t^T \widetilde{\eta}_t(u,r)\, dr = \widetilde{\phi}(\mathrm{i}u,0;t,T) + \left\langle \widetilde{\psi}_C(\mathrm{i}u,0;t,T), Y_t \right\rangle. \tag{3.14}$$

In particular, for $t = 0$ we recover the characteristic function and we obtain

$$\int_0^T \widetilde{\eta}_0(u,r)\, dr = \widetilde{\phi}(\mathrm{i}u,0;0,T) + \left\langle \widetilde{\psi}_C(\mathrm{i}u,0;0,T), y \right\rangle$$

[5]Here, we mean that the price or volatility surface created by the model and the Lévy measure should be the same.

and, if we denote with $C(y)$ the set of characteristic functions $\widetilde{\eta}_0$, we notice that for any element $\widetilde{\eta}_0 \in C(y)$, there exists (at least) one $\mu \in \mathrm{Inc}^D$ that defines $\widetilde{\eta}$ itself. It is thus possible to establish a surjective function g between Inc^D and $C(y)$. The existence of such g is equivalent to nothing but the Condition (3.12) previously stated since we can consider any jump-time as an initial starting point for the process $(\widetilde{X}, Y)$ due to the Markovianity of the process.

3.3 From Heston to Hull-White extended Bates model

Before introducing the consistent recalibration model more mathematically, let us briefly recall the Heston model, which is an affine stochastic volatility model, for $X = \log(S)$ being the log-return of the underlying price

$$\begin{aligned} &dX(t) = \Big(r - q - \frac{1}{2}V(t)\Big)dt + \sqrt{V(t)}dW_1(t), \;\; X(0) = x_0, \\ &dV(t) = k\left[\theta - V(t)\right]dt + \sigma\sqrt{V(t)}dW_2(t), \;\; V(0) = v_0, \\ &dW_1(t)\,dW_2(t) = \rho\,dt, \qquad \rho \in [-1, 1], \end{aligned} \tag{3.15}$$

and where r and q represent the instantaneous risk-free and dividend yields respectively and are constant, $\theta > 0$ is the long-term mean of the variance, $k > 0$ is the speed of mean-reversion, $\sigma > 0$ represents the instantaneous volatility of the variance process V. In order to ensure positivity of the variance process, we need to satisfy $2k\theta > \sigma^2$ (Feller condition).
For $0 \le t \le T$, we have that η defines a Γ_1-semimartingale:

$$\eta_t(u, T) = F(\mathrm{i}u, \psi_C(\mathrm{i}u, 0, T-t)) + R_C(\mathrm{i}u, \psi_C(\mathrm{i}u, 0, T-t))\,V_t,$$

where C coincides with $\mathbb{R}_{>0}$ and

$$\begin{aligned} F(u_1, u_2) &= k\theta u_2 + (r-q)u_1, \\ R_C(u_1, u_2) &= \frac{1}{2}u_1(u_1 - 1) + \frac{1}{2}\sigma^2 u_2^2 + \sigma\rho u_1 u_2 - k u_2. \end{aligned}$$

The Hull-White extension of the Heston model consists in a generalized version of the so-called Bates model ([Bat88]) in which we add a compensated[6] jump Lévy process L with Lévy measure $\nu(t, dx)$ to the dynamics of the log-return X:

$$dX(t) = \Big(r - q - \frac{1}{2}V(t)\Big)dt + \sqrt{V(t)}dW_1(t) + dL_t. \tag{3.17}$$

The first consequence that should appear obvious is that we are enriching the space of calibrated volatility surfaces, thanks to the Lévy process, while keeping the same dimensions of the state variables. Accordingly, the functional characteristic F will then change to

$$F_t(u_1, u_2) = k\theta u_2 + (r-q)u_1 + \mu_L(u_1, u_2, t), \tag{3.18}$$

where μ_L is the cumulant generating function of L. As we will see below, we can establish a bijective relation between μ_L and the Lévy measure ν_L.

We are talking about a generalized Bates model since the Lévy measure is also allowed to change in time and, as already said, this permits to make also other parameters time dependent.

[6] Compensation is necessary to have a martingale process, as it is often the case in the pricing context.

3.3.1 Consistent recalibration (with words)

Time is mature to explain how the generalized Bates model can be used as a consistent recalibration (CRC) model. Recall that although formally there are only two state variables, now also parameters are free to change in time thanks to the compensation mechanism that the Hull-White extension provides.

Let us start at time $t = t_0$ with a log-price X_{t_0}, a set of parameters $p_{t_0} = (r, q, k, \theta_{t_0}, \sigma_{t_0}, \rho_{t_0})$, an initial variance for the log-price V_{t_0} and the compensated jump Lévy process L_{t_0} which represents the Hull-White extension. Note that some of the parameters are not constant, but change over time and are denoted by the time-index. This particular combination of state variables and parameters fully specifies a particular model $\mathcal{M}_1$ among all possible models $\mathcal{M}_i$ that can represent an implied volatility surface (IVS) without breaking any no-arbitrage constraints and should be able to reflect those market conditions that are summarised by the IVS at time t_0. It is thus natural to write $\text{IVS}_t = \text{IVS}(X_t, V_t; \theta_t, \sigma_t, \rho_t; \{T_i\}, \{K_j\})$ for the volatility surface at time t. The model state variables X and V are thus able to evolve in time until $\mathcal{M}_1$ is able to mirror the market. Eventually, this situation will break at time $t = t_1$ and a new calibration will be necessary.

1. Starting from time t_0, X and V can evolve until $t_1 = t_0 + \Delta t$, where the new volatility surface is given by $\text{IVS}(X_{t_1}, V_{t_1}; \theta_{t_0}, \sigma_{t_0}, \rho_{t_0}; \{T_i\}, \{K_j\})$.

2. Parameters $\Theta_2 := (\theta_{t_0}, \sigma_{t_0}, \rho_{t_0})$ will move to another configuration $(\theta_{t_1}, \sigma_{t_1}, \rho_{t_1})$, but, to enforce a smooth change between the first configuration and the second,

3. Also the Lévy process will be adjusted and will compensate the changes in the parameters $(\theta_t, \sigma_t, \rho_t)$ to reproduce the same IVS. This implies a movement in the parameters Θ_1 of the Lévy process.

4. In this way we can represent realistically the behaviour of the market.

The *recalibration* of the Hull-White extension is also preserving the drift-condition of the forward characteristics, thus ensuring that we do not violate no-arbitrage constraints. The new model $\mathcal{M}_2$ will then specify the evolution in time of the state variables until another recalibration will be needed.

Notice that the evolution of the state variables is determined by the "old" parameters $(\theta_{t_i}, \sigma_{t_i}, \rho_{t_i})$ on the closed interval $[t_i, t_{i+1}]$ and for this reason there is no discontinuity at $t = t_{i+1}$ in the modelled IVS caused by the parameters movement. Further, since we do not know in advance the exact recalibration times t_i, these are random variables (more in Section 3.5).

3.4 CNKK Equation

3.4.1 Quick heuristics

The mathematical formulation that is needed to describe what we sketched above starts from (3.2). If we rewrite the same equation using Musiela's parametrization, defining $x := T - t$, then the map becomes $(u, t, x) \mapsto \eta_t(u, t + x)$ in the new notation. In addition, let us introduce the strongly continuous semigroup $\{S(t) \,|\, t \geq 0\}$ of right shifts, such that for a proper function g this

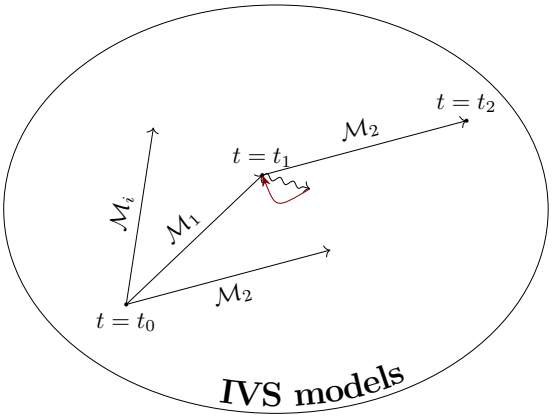


Figure 3.1: At $t = t_1$ the change in the parameters $(\theta_{t_0}, \sigma_{t_0}, \rho_{t_0})$ would cause a jump in the volatility structure (snake arrow), but this is compensated by the change in the Lévy process L (red bent arrow).

is mapped to $S(t)g(u,\cdot) = g(u, t+\cdot)$. Then we can rewrite Equation (3.2) as

$$\eta_t(u, t+x) = S(t)\eta_0(u,x) + \int_0^t S(t-s)\alpha_s(u,t+x)ds + \sum_{i=1}^d \int_0^t S(t-s)\beta_s^i(u,t+x)dM_s^i, \quad (3.19)$$

which can be rewritten in terms of $\theta_t(u,x) := \eta_t(u,t+x)$ and, with abuse of notation, $\alpha_t(u,x) := \alpha_t(u,t+x)$, $\beta_t(u,x) := \beta_t(u,t+x)$ as

$$\theta_t(u,x) = S(t)\theta_0(u,x) + \int_0^t S(t-s)\alpha_s(u,x)ds + \sum_{i=1}^d \int_0^t S(t-s)\beta_s^i(u,x)dM_s^i. \quad (3.20)$$

Finally, the passage to the limit will justify what written in the next subsection.

3.4.2 CNKK SPDE

The framework we will develop in the following will allow a thorough analysis of factor models in the CNKK-approach introduced in [CN12] and [KK15], yet with crucial differences. For example, as already done by Kallsen and Krühner, we assume that the volatility processes β^i of the forward characteristic η are functions of the present state of η itself, i.e. for all $i = 1, \dots, d$

$$\beta_t^i(u,T)(\omega) = \sigma\big(t, \eta_{t^-}(\cdot,\cdot)(\omega)\big)(u,T),$$

but, since we introduce the right-shift operator, we will obtain an SPDE and not simply an SDE.

Definition 3.4.1 (Lévy codebook Hilbert space)**.** *Let G be a Hilbert space of continuous complex-valued functions defined on the strip $-\mathrm{i}[0,1]^n \times \mathbb{R}^n$, i.e. $G \subset C((-\mathrm{i}[0,1]^n) \times \mathbb{R}^n; \mathbb{C})$.*
H is called a Lévy codebook Hilbert space *if H is a Hilbert space of continuous functions $\eta : \mathbb{R}_{\geq 0} \to G$, i.e. $H \subset C(\mathbb{R}_{\geq 0}; G)$ such that*

- *There is a continuous embedding $H \subset C(\mathbb{R}_{\geq 0} \times (-\mathrm{i}[0,1]^n) \times \mathbb{R}^n; \mathbb{C})$,*
- *The shift semigroup $(S_t\eta)(u,x) := \eta(u, t+x)$ acts as strongly continuous semigroup of linear operators on H,*

- *Continuous functions of finite activity Lévy-Khintchine type*

$$(u,t) \mapsto \mathrm{i}\,\langle a(t),u\rangle - \frac{\langle u, b(t)u\rangle}{2} + \int_{\mathbb{R}^n} \big(\exp(\mathrm{i}\,\langle \xi,u\rangle) - 1\big)\,\nu_t(d\xi)$$

lie in H, where a, b, ν are continuous functions defined on $\mathbb{R}_{\geq 0}$ taking values in $\mathbb{R}^n$, the positive-semidefinite matrices on $\mathbb{R}^n$ and the finite positive measures on $\mathbb{R}^n$. This is for example the case for processes with independent increments and finite variation.

Remark 3.4.2. Notice that we do not assume that there are additional stochastic factors outside the considered parametrization of liquid market prices.

Remark 3.4.3. Notice that elements of the Hilbert space H are understood in Musiela parametrization and therefore denoted by a different letter in the sequel. As already written in Subsection 3.4.1, we have the relationship $\eta_t(u,t+x) = \theta_t(u,x)$, with $x := T - t$. In this sense, we also have the equality $\theta_t(u,0) = \kappa_t^X(u)$ for the predictable characteristics of X.

Definition 3.4.4 (CNKK equation)**.** *Let H be a Lévy codebook Hilbert space. We call the following stochastic partial differential equation*

$$d\theta_t = \big(A\theta_t + \mu_{\mathrm{CNKK}}(\theta_t)\big)dt + \sum_{i=1}^{d} \sigma_i(\theta_t)\,dB_t^i \tag{3.21}$$

a CNKK equation $(\theta_0, \kappa, \sigma)$ *with initial term structure θ_0 and characteristics κ and σ, if*

- *$A = \frac{d}{dx}$ is the generator of the shift semigroup on H,*
- *$\sigma_i : U \subset H \to H$, U an open subset of H, are locally Lipschitz vector fields, and*
- *$\mu_{\mathrm{CNKK}} : U \to H$ is locally Lipschitz and satisfies that for all $\eta \in \Gamma_n$ we have*

$$\int_0^{T-t} \mu_{\mathrm{CNKK}}(\theta)(u,r)\,dr = \theta(u,0) - \kappa_\theta\left(u, -\mathrm{i}\int_0^{T-t} \sigma(\theta)(r,u)\,dr\,; 0\right), \tag{3.22}$$

where $(\kappa_\theta)_{\theta\in U}$ is Γ_{n+d}-valued for each $\theta \in \Gamma_n$, such that $\kappa_\theta(u,0;0) = \theta(u,0)$ and $\kappa_\theta(0,v;0) = -\frac{\|v\|^2}{2}$, for $u \in \mathbb{R}^n$, $v \in \mathbb{R}^d$.

Remark 3.4.5. κ_θ is the forward characteristic process associated to the couple (X,B), where X is (still) the log-return price process and B the driving process of θ. Moreover, Equation (3.22) can be seen as a *drift-condition* and it is analogous to Equation (3.5), reformulated under the Musiela's parametrization.

Remark 3.4.6. It is evident how Equations (3.21) and (3.20) relate to each other and how the former can be seen as the limit case of the latter.

Remark 3.4.7. We do not require that all solutions of Equation (3.21) are Γ_n-valued, which would be too strong as a condition and difficult to characterize. In particular, Γ_n is a more general than what we need.

Proposition 3.4.8. *Let θ be a Γ_n-valued solution of a CNKK equation and let X be a semimartingale such that the predictable characteristics satisfy*

$$\kappa_t^{(X,B)}(u,v) = \kappa_{\theta_t}(u,v;t)$$

for $u \in \mathbb{R}^n$, $v \in \mathbb{R}^d$ and $t \geq 0$, then the tuple (X,θ) satisfies the conditional expectation condition.

Proof. The proposition is basically a consequence of Theorem 3.1.13: the drift condition is satisfied by assumption and

$$\exp\left(\mathrm{i}\,\langle u, X_t\rangle - \int_0^t \kappa_{s-}^{(X,B)}(u,v;s)\,ds\right)$$

is a (local) martingale because of the Lévy-Khintchine assumption in Definition 3.4.1 regarding the functions of the Lévy codebook Hilbert space. □

3.4.3 Generalization of the HJM equation

Equation (3.21) is very similar to the famous HJM equation[7], but there are relevant differences, for example both the drift and drift condition are different and, what is more, functions have another argument (a *strike* dimension) that is completely missing in the case of the HJM equation, where only a time-dimension is considered.

We can construct a particular example which corresponds indeed to the HJM equation: let us consider a situation without leverage (where the Brownian motion B is independent of the return process X), assuming that

$$\kappa_\theta(u,v;0) = \theta(u,0) - \frac{\|v\|^2}{2},$$

for $u \in \mathbb{R}^n$, $v \in \mathbb{R}^d$ and $t \geq 0$. Basically, we are looking at functions in a restricted Lévy space, for which the Lévy measure is null. This implies that the CNKK equation is a parameter-dependent HJM equation. In this case, Condition (3.22) can be simplified to

$$\mu_{\mathrm{CNKK}}(\theta)(u,x) = -\sum_{i=1}^{d} \sigma^i(\theta)(u,x) \int_0^x \sigma^i(\theta)(u,s)\,ds,$$

for $x \geq 0$ and $u \in \mathbb{R}^n$ (note the analogies with Example 3.1.26).

Example 3.4.9. **Black-Scholes model ([BS73]):** It might be interesting at this point to see a concrete example coming from a simpler model. If we consider asset prices described by a geometric Brownian motion $dS_t = S_t \sigma dW_t$, where $\sigma > 0$ is constant and W is a standard Brownian motion, then the log-prices X are given by $dX_t = d\log S_t = {}^{\sigma^2}/_2\,dt + \sigma dW_t$. We find that $\eta_t(-\mathrm{i}u, r) = {}^1/_2\,\sigma^2 u(u-1) = F(u)$, in the notation of (3.9). It is easy to see that η is pure-drift and that the extended functional characteristic becomes $F_t = \frac{1}{2}\sigma_t^2 u(u-1) + \mu_L(u,t)$, where μ_L is the cumulant of the generalized Hull-White extension.

From what has been said so far, it is clear how the CNKK equation is in fact a generalization of the HJM equation.

Remark 3.4.10. It is possible to further increase the complexity of the equation, for example considering options on a term structure. In this case, we would need another argument to take into account for both drift and volatility.

Remark 3.4.11. All these considerations are conceivable only because we are dealing with affine processes. In general, for a return price process X, it might not be possible to write the conditional expectation condition and to continue with the following statements.

[7] In the literature, this is also known as HJMM equation, where the last M stands for Musiela.

Remark 3.4.12. It is possible to generalize what has been said in this section for processes driven by infinite dimensional Brownian motions, e.g. in Equation (3.21) we could replace the sum $\sum_{i=1}^{d}$ with $\sum_{i\in\mathbb{N}}$. The theory has been paved in the book by Da Prato and Zabczyk [DZ14], but also Chapter 2 of [Fil01] by Filipović provides a useful and accessible introduction. We continue considering the finite-dimensional case because this does not really create new hurdles to be solved, in contrast to the main obstacle, the drift μ_{CNKK}, for which no explicit expression is available.

3.5 Consistent recalibration (with maths)

We are now ready to face the same considerations we reported above in more rigorous settings. First of all, let us recap the most important equations. For the return process, we have

$$\begin{aligned} dX(t) &= \delta_t^X(X_t, V_t)\,dt + \gamma_t^X(X_t, V_t)\,dW_1(t) + dL_t, \quad X(0) = x_0, \\ dV(t) &= \delta_t^V(X_t, V_t)\,dt + \gamma_t^V(X_t, V_t)\,dW_2(t), \quad V(0) = v_0, \end{aligned} \tag{3.23}$$

with $dW_1(t)\,dW_2(t) = \rho_t\,dt$, for $\rho_t \in [-1, 1]$. Drifts and volatility coefficients are denoted by δ and γ respectively and can be functions of X and V, e.g. $\gamma^X(x, v) = \gamma^V(x, v) = \sqrt{v}$. While for the forward characteristic process, our *codebook*, we report the CNKK SPDE

$$d\theta(t) = \left[A\theta(t) + \mu_{\text{CNKK}}(\theta(t))\right]dt + \sum_{i=1}^{d}\sigma_i(\theta(t))\,dB^i(t), \quad \theta(0) = \theta_0, \tag{3.24}$$

where the dependence among the Brownian motions B^i with $i = 1, \dots, d$ and these with W_j for $j = 1, 2$ is not specified. The key relation that connects the two different systems is given in Corollary 3.2.1 and is the following (rewritten in the Musiela notation):

$$\theta_t(-\mathrm{i}u, x) = F_{t+x}(u, \psi_C(u, 0; x)) + \langle R_C(u, \psi_C(u, 0; x)), V_t\rangle\,. \tag{3.25}$$

Last but not least, we should also remember that parameters are free to move in time. As such, we consider the process p, whose dynamics are exogenously given, but which are confined inside the space of admissible parameters $\Theta \subset \mathbb{R}^M$. In the example described in Subsection 3.3.1 it is defined as

$$p_t = (\theta_t, \sigma_t, \rho_t), \qquad t \geq 0,$$

given the constraints of positivity and Feller condition for σ_t and θ_t and $\rho_t \in [-1, 1]$. This is the reason why we used the subscript t in Equations (3.23) above.
Let us suppose that at time t_0 the model can fit well the market surface given by observed call prices (or, equivalently, of implied volatility surface) $C_{t_0}^{obs}(T_i, K_j)$ for $i = 1, \dots, n$ and $j = 1, \dots, m$. In this condition, the process $(\theta, (X, V))$ are free to evolve in time until the a new calibration is necessary. This is the case when

$$\Delta_{C_{t_0}} := \sum_{i=1}^{n}\sum_{j=1}^{m}\left|C_{t_0}^{\text{model}}(T_i, K_j) - C_{t_0}^{\text{obs}}(T_i, K_j)\right|^2 > \varepsilon, \tag{3.26}$$

where C_t^{model} is the price of a call option at time t given by the model and ε is a threshold fixed a priori.
Thus, we can define the following hitting times: for $i \in \mathbb{N}$,

$$\begin{aligned} \tau_0 &:= \inf\{t > t_0 \,:\, \Delta_{C_t} > \varepsilon\} \\ \tau_{i+1} &:= \inf\{t > \tau_i \,:\, \Delta_{C_t} > \varepsilon\}\,. \end{aligned} \tag{3.27}$$

As already underlined in [KK15], the model price C_t^{model} can be expressed as a measurable function of the codebook θ satisfying Equation (3.24). In particular, since it is a progressively measurable process (it is right-continuous on a complete probability space), we can use Début theorem, which tells us that the sequence $(\tau_i)_{i=\mathbb{N}_0}$ is in fact made by *stopping times*.

Remark 3.5.1. Since we are in the same framework as [DZ14], and indeed we could generalize all results to infinite dimensional Brownian motion, it is worth mentioning that the solution process θ satisfies the strong Markov property.

The strictly increasing sequence $(\tau_i)_{i=\mathbb{N}_0}$ is important since it is at these random (stopping) times that we have to run a *new calibration* procedure for the model. Both the "true" state variables X and V and the parameter process p are allowed to change in order to have $\Delta_{C_{t_0}}$ less than ε again. This is just the only first calibration problem we need to solve. Indeed, in order to compensate the changes caused by the new parameters, we have to modify F_t. In particular, we can calibrate the so-called Hull-White extension part, which enters F_t as the cumulant generating function μ_L of the Lévy process L. This *re-calibration* ensures that we are not breaking the validity of Equation (3.25), while allowing for an "exact" match (in the sense of having $\Delta_{C_t} < \varepsilon$) with the observed data. Once μ_L is recovered, we are able to write down again the equation for the codebook θ. We can summarise this last passage more mathematically by introducing an operator $\mathcal{I}$ such that

$$\mathcal{I} : \Theta \times \mathbb{R}_+^{n+m} \to \mathrm{Inc}^{\mathbb{R}^n \times C}, \qquad \big(p, (C^{obs})\big) \mapsto \mu_L. \tag{3.28}$$

Remark 3.5.2. The cumulant generating function μ_L identifies uniquely L if and only if the process L has finite moments of order n for all $n \in \mathbb{N}$. If we denote with ν_L the Lévy measure associated to L, then this is true if and only if

$$\forall n \in \mathbb{N}, \quad \int_{\|x\| \geq 1} \|x\|^n \, \nu_L(t, dx) < \infty.$$

Eventually, we are now ready to give the definition of Consistent Recalibration (CRC) model with piecewise constant parameters p:

Definition 3.5.3 (Consistent Recalibration Model with Piecewise-constant p)**.** *Let $(\Omega, \mathcal{F}, \mathbb{F}, \mathbb{P})$ be a complete filtered probability space. The quintuple $(\theta, (X, V), p, L, (\tau_i)_{i\in\mathbb{N}_0})$ is called* consistent recalibration model *for equity derivative pricing if for the stochastic processes $(\theta, (X, V), p)$ with values in $H \times (\mathbb{R}^n \times C) \times \Theta$ there exists a jump Lévy process L (with finite moments) such that the following conditions are satisfied for all $n \in \mathbb{N}_0$:*

(i) The Hull-White extension L on $[\tau_n, \tau_{n+1}]$ is determined by calibration to $\theta(\tau_n)$ through μ_L:

$$\begin{aligned} &\theta(\tau_n)(u, 0) = \kappa_{\theta(\tau_n)}(u, 0; 0), \\ &\mu_{L(\tau_n)} = \mathcal{I}\left(p(\tau_n), (X(\tau_n), V(\tau_n)), C^{\mathrm{obs}}_{\tau_n}\right), \end{aligned}$$

and for $t \in [\tau_n, \tau_{n+1}]$ we have

$$L(t) = S(t - \tau_n)L(\tau_n).$$

(ii) The evolution of (X, V) on $[\tau_n, \tau_{n+1}]$ corresponds to the Hull-White extended stochastic volatility affine model determined by the parameters $p(\tau_n)$ and by the process $L(\tau_n)$:

$$(X, V)(t) = \left(X^{\tau_n, X(\tau_n)}, V^{\tau_n, V(\tau_n)}\right)(t) \quad \text{for} \quad t \in [\tau_n, \tau_{n+1}],$$

where $(X^{s,x}, V^{s,v})$ is the unique solution of the system of SDEs (3.23) on $[s,\infty)$ with initial conditions $X(s) = x$ and $V(s) = v$ and with L_t replaced by L_{t-s}. Implicitly, we also assume that all parameters p that enters the model are admissible (with the usual meaning).

(iii) The evolution of θ on $[\tau_n, \tau_{n+1}]$ is determined by X and V according to the prevailing Hull-White extended stochastic volatility model: for $t \in [\tau_n, \tau_{n+1}]$ and $x \in [0, \tau_{n+1} - \tau_n]$

$$\theta(t)(-\mathrm{i}u, x) = F_{\tau_n+x}(u, \psi_C(u, 0; x)) + \langle R_C(u, \psi_C(u, 0; x)), V(t)\rangle\,.$$

For the processes (X, V) and θ, we use the same symbols as in Equations (3.23) and (3.24), with a slight abuse of notation, since these stochastic processes evolve in the intervals $[\tau_n, \tau_{n+1}]$ following the same dynamics, but according to the parameters $p(\tau_n)$ and to the process $L(\tau_n)$. The parameters p remain constant in each interval of the type $[\tau_n, \tau_{n+1})$ and, by construction, $(\theta, (X, V))$ is continuous on every stopping time τ_n.
In this sense, any CRC model can be seen as the concatenation of stochastic volatility affine models with static parameters.

We can now prove the following.

Theorem 3.5.4. *Let $(\theta, (X, V), p, L, (\tau_i)_{i\in\mathbb{N}_0})$ be a consistent recalibration model as in Definition 3.5.3 and $S = \exp(X)$ the discounted price process. Then S and European call (resp. put) option prices $C_t(T, K)$ (resp. $P_t(T, K)$) are (true) martingales on $\mathbb{R}_{\geq 0}$. Moreover, if we denote the payoff function of a call option as $V(y) = (y - K)_+$, then the following pricing formula holds:*

$$C_t(T, K) = \frac{e^{rt}}{2\pi} \int_{\mathrm{i}\mathrm{Im}\,(\tilde{z})+\infty}^{\mathrm{i}\mathrm{Im}\,(\tilde{z})-\infty} \Psi_T(-z)\hat{V}(z)\,dz, \tag{3.29}$$

where $\tau_{n+1} > T \geq \tau_n$ for some $n \in \mathbb{N}$, $\tilde{z}$ belongs to the analytic strip for which we have finite exponential moment (cf. Remark 3.1.2), $\hat{V}$ denote the Fourier transform of V and $\Psi_{T|\tau_n}$ is the characteristic function of X_T, i.e.

$$\Psi_{T|\tau_n}(u) = \mathbb{E}\left[e^{\mathrm{i}\langle u, X_T\rangle} \mid \mathcal{F}_{\tau_n}\right] = e^{\mathrm{i}\langle u, X_{\tau_n}\rangle + \int_0^T \theta_{\tau_n}(u,r)\,dr}. \tag{3.30}$$

Proof. From point *(ii)* of Definition 3.5.3 we have the couple (X, V) is a stochastic volatility model on each interval of the type $[\tau_n, \tau_{n+1}]$, from which martingality of S follows by taking the conditional expectation and noting that, by Definition 3.4.1, we automatically have $\theta(0, x) = 0$ for any $x \geq 0$. We can further extend this result on $\mathbb{R}_{\geq 0}$ because, by our own construction, the concatenation of the entire process is made in a continuous way.
Once it is established that S is a martingale, the same property follows for European call and put options by use of the tower property of the conditional expectation.
The pricing formula (3.29) comes from Fourier pricing, which was initially introduced by Carr and Madan in [CM99], while (3.30) comes from Corollary 3.2.1 where the characteristic process has been expressed in Musiela notation. □

Remark 3.5.5. Having shown that the discounted price process S and derivatives' prices are martingales, we automatically rule out arbitrage possibilities for the so-called consistent recalibration models introduced in Definition 3.5.3.

Remark 3.5.6. Note that the from (3.30) we see that the characteristic process θ encodes information on the conditional expectation of X, which is equivalent, as already noted in [RT17] to the knowledge of the entire derivative-price surface, thanks to Breeden–Litzenberger formulas ([BL78]).

Remark 3.5.7. Equation (3.29) can be generalized to other payoff function and is usually enriched with a dampening factor which is introduced to exploit numerical algorithm ([CM99]). In our case, we will use Fourier-pricing techniques in order to obtain the dataset for a supervised learning algorithm.

3.5.1 Numerical considerations

There are practical remarks that we should consider when dealing with CRC models as defined above in a numerical

- ***Simulations***: As already mentioned in [Har+18], it is worth noting that simulating (X, V) is much easier than simulating the HJM codebook, that is θ, in particular when this is infinite dimensional, as it could be the case also here. In fact, with the approach we are outlining, we do not need to simulate anything from any infinite dimensional distribution.
- ***Drift term***: Even if we wanted to simulate θ solving the SPDE (3.24), we should be able to write down explicitly the drift term $\mu_{\text{CNKK}}(\theta)$, but, apart from some degenerate cases, this is not possible. The only way to overcome this chasm is acknowledging Equation (3.25) as a key relation for the entire construction.
- ***Process p***: If we assume that a piecewise process for the parameters p is given (or obtained through calibration), then CRC models can be simulated following steps (i) to (iii) in Definition 3.5.3.
- ***Operator $\mathcal{I}$***: Last but not least, we have not specified precisely how the operator $\mathcal{I}$ is acting. For the moment, we will consider it as an abstract operator. This is anyway of great relevance because once we are able to recover L (or, alternatively, μ_L), we can obtain θ through Equation (3.25). Otherwise speaking, we could solve SPDE (3.24).

3.6 Deep calibration

3.6.1 An ill-posed inverse problem

If we look more closely to steps $(i) - (iii)$ of Definition 3.5.3, it is possible to realize that the more complex aspect is given by the application of the operator $\mathcal{I}$. This is basically a calibration conditioned on some (new) parameters and state variables whose complexity depends on the distribution of the Lévy process L. In general, this is not a trivial operation, since it consists in solving an *inverse problem* that is *ill-posed* in the sense of Hadamard even for the easiest cases (e.g. Bates model). The inverse problem is ill-posed because of an identifiability issue, which means that the information coming from market data is insufficient to exactly identify the parameters. If we express the quantity ΔC_t of Equation (3.26) as a function of the model parameters ϑ, that is

$$\Delta C_t(\vartheta) = \sum_{i=1}^{n}\sum_{j=1}^{m} \left| C_t^{\text{model}}(\vartheta; T_i, K_j) - C_t^{\text{obs}}(T_i, K_j) \right|^2,$$

we can write the identifiability problem as the fact that the function $\Delta C_t(\vartheta)$ has many local minima. Furthermore, it is in general unclear whether these minima can be reached by the adopted algorithm. For example, Cont and Tankov show in [CT04] that if one had available a set

of call options prices (or, equivalently, implied volatilities) for *all* strikes (in a given time interval!) and a *single* maturity, then it would be possible to deduce all the parameters of the model and, in particular, the Lévy triplet. But in reality this is never the case, since we only know prices for a finite number of strikes and, in addition, we also have observational errors in the data. As a result, we have a serious identification problem, exemplified by the fact that we can obtain the same prices for (infinitely) many combinations of the parameters. The strategy which they begin to develop in [CT04] and complete in [CT06b] is the use of the Kullback-Leibler divergence, also called relative entropy, as a regularizer in order to get a well-posed inverse problem.
To overcome this issue, we decided to follow a different strategy, making use of the implicit regularization present in neural networks.

3.6.2 Learning the inverse map

Andres Hernandez was among the first who tried neural networks (NN) to address calibration tasks in Finance. In [Her16], he showed that a feedforward NN can actually approximate the inverse map given by the pricing formula and obtain the two parameters of the Hull-White interest rate model (a, σ) as output of a NN. Just as a reminder, the Hull-White model consists of the following SDE

$$dr(t) = [\beta(t) - ar(t)]\, dt + \sigma\, dW(t),$$

where a, $\sigma > 0$ and $\beta(t)$ is uniquely determined by the term structure[8]. The greatest achievement that he highlighted in the paper is the possibility of replacing the traditional "slow" and cumbersome calibration procedure with a new straightforward deterministic map, which makes calibration itself a very efficient task, since the core of all calculations is offset to the training phase. In fact, once a NN is trained, its application is extremely cheap from a computational point of view, being the most expensive operations simple matrix vector multiplications.

Despite the good results obtained by Hernandez, learning the map from the prices to the parameters can be critical, since this map is not known in explicit form. In principle, we do not even know if the universal approximation theorem could be applied, because the direct map could not be bijective (thus having a discontinuous inverse function). In general, since the inverse map is not known, we lack control on it and it might well be that a NN learns appropriately the map on the given training sample, but is not able to generalize on out-of-sample data. This is actually what happened when we tried to apply this approach to our problem since our situation is considerably more involved than Hernandez'.

For this reason, it is a better idea to learn the direct (or forward) map from the parameters to the prices/implied volatilities. This is done, for example, by Horvath and coauthors in [HMT21], where they implemented a feedforward NN to directly get the volatilities from the model parameters. Note that for the training of the networks, the data are artificially generated and the grid of strikes and maturities is fixed at the beginning. Again, the most appealing advantage they see in the application of NN is the possibility of enabling live calibration of derivative instruments, since the application of the NN itself only requires milliseconds avoiding the traditional bottleneck of calibration. This allows making use of new models that were before considered too computationally expansive for use, such as the rough volatility models (e.g. rough Heston or rough Bergomi model, which require Monte Carlo algorithms).

Despite the encouraging results found in [HMT21], there are still some inconveniences using

[8]This curve is calibrated through the derivative of the instantaneous forward rate $f(t,T)$ at time $t = 0$, i.e. $\beta(t) = \frac{\partial f(0,t)}{\partial T} + af(0,t) + \frac{\sigma^2}{2a}\left(1 - e^{-2at}\right)$ once the other two parameter a and σ have been calibrated.

this last approach. If on one hand the advantages on the speed side are evident, on the other there are still downsides that are relevant, but not addressed by this kind of solution. Indeed, problems might come by the second step of this procedure, as denoted in [HMT21], which is the real calibration. Even if we have to face a deterministic optimisation problem, this might not be as easy as it seems, in particular for multidimensional models. In these cases, we might need to use a local optimiser to speed up, which usually requires prior knowledge about the solution, or a global optimiser, which might take longer time.

To overcome these issues, we propose a new method that allows learning the inverse map, as already done by Hernandez, even for ill-posed problems. It is our wish to underline that the same idea could be used to learn the inverse map and solve inverse problems in fields other than mathematical finance. Adopting the same approach as Hernandez did not work out in our case, because of identifiability issues: different combinations of parameters in the stochastic volatility Hull-White extended affine model result in the same volatility surface. Thus, our idea is allowing a neural network to decide autonomously which parameters giving as output knowing that these parameters will then have to give rise to the prescribed volatility surface. In this way, we will also be able to learn the operator $\mathcal{I}$ defined in (3.28).

In order to make the system works, we need two neural networks. The first, denoted in the following as NN_1, is a map between parameters and volatilities (basically, the usual pricing function, as learnt in [HMT21]), while the second, called NN_2, maps volatilities to parameters (but is not trained in the usual way); in our case, to the parameters defining the Lévy process L. Then, we compose the two networks, where the first is trained, while the second is not, to obtain a new neural network NN_3 which receives in input volatilities and returns the (same) volatilities:

$$\mathrm{NN}_3 := \mathrm{NN}_2 \circ \mathrm{NN}_1 \,.$$

In other words, NN_3 will learn the identity and, during the training phase, NN_2 will get trained. In this respect, we can see NN_2 as the *inverse* neural network of NN_1. The trick is as simple as that. However, notice that we might not recover exactly the same parameters that gave birth to the original IVS, but an *equivalent*[9] combination that resulted in the same surface through NN_1.

3.6.3 Numerical implementation

In broad terms, the numerical implementation follows the model outlined in Section 3.3, where the process L is a compensated compound Poisson process in which the size of jumps is normally distributed. Since the parameters are allowed to change in time, the mean and variance of the Gaussian distribution for the jump-size of L are time-dependent, while the Poisson rate is considered constant in time. The same holds true for the other parameters belonging to the "Heston" part, with the exception of the interest rate r, the dividend rate q and k, which is the speed of mean reversion for the variance process. We decided to free ourselves from a static framework and to have a variable maturities-strikes grid. More precisely, we used ten different time-to-maturities $\{\tau_i\}_{i=1}^{10}$, with $\tau_1 < \cdots < \tau_{10}$ ranging between 7 and 440 days (extremes included), being more concentrated for short maturities, and thirteen different moneyness $m_1 < \cdots < m_{13}$ ranging between 0.8 and 1.2 (extremes included) in strictly increasing order. One difference with the generalized Bates model described in Section 3.3 is that we have a maturity-dependent jump distribution: mean and variance depend on the maturity in the sense that they are piecewise

[9] We could think of the combinations selected by the neural network as the representatives for an equivalence class, where all members originate the same implied volatility surface.

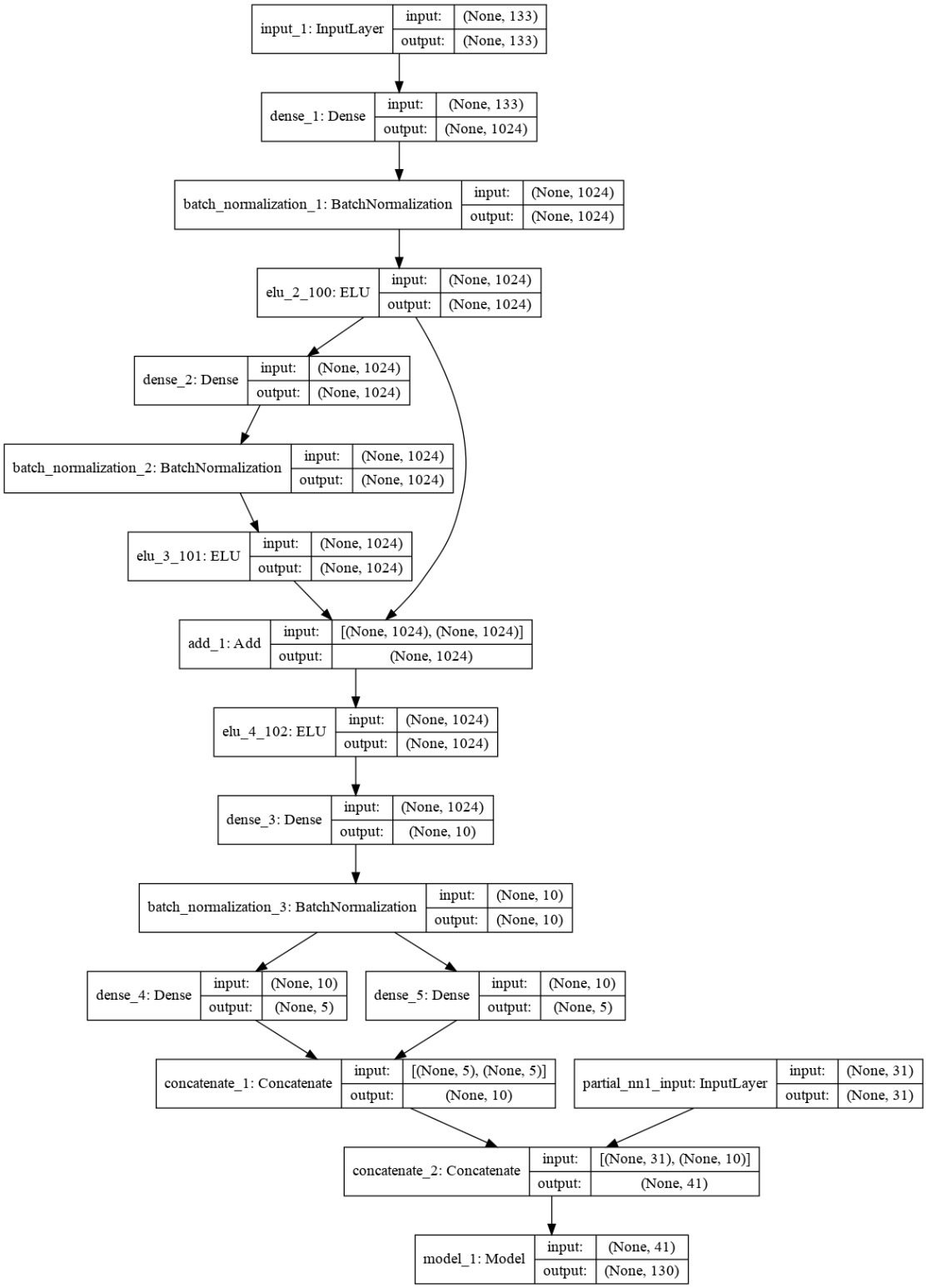


Figure 3.2: Keras representation of NN_3 (with just 2 "main" hidden layers instead of 4 for NN_2 to fit the picture in the page). NN_1 is summarised as model_1 at the very bottom. Between the 2 wanted hidden layers elu_2 and elu_4, it is possible to find another layer (in this sense the nomenclature *1-cell* that we used in the text).

constant within two adjacent time-to-maturities, therefore, the process L is here modelled through 11 parameters: the Poisson rate and then 5 tuples mean-variance for the normal distributions.

The first neural network, i.e. NN_1, is a 1-cell[10] residual feedforward NN (see [He+16] for more information on ResNets) composed by 4 "main" hidden layers with 1024 nodes each. The input layer has dimension 41 and includes

$$r, q, \{\tau_i\}_{i=1}^{10}, \{m_i\}_{i=1}^{13}, v_0, k, \theta, \sigma, \rho, \lambda, \{\nu_i\}_{i=1}^{5}, \{\delta_i\}_{i=1}^{5},$$

where ν_i and δ_i are the mean and standard deviations of the normal distributions for the jump size. The output layer has dimension 130 and includes the entire point-valued volatility surface, denoted as

$$\{\mathrm{IVS}_i\}_{i=1}^{130}.$$

The activation function used for all layers (apart from the output layer) is ELU. This network is trained first with artificially generated data: all parameters are sampled from uniform distributions whose extremes (parameters) are defined a priori (and are kept fixed throughout the process). Then, QuantLib Python routines (see [AB+03]) are used to obtain in an efficient and fast way all the necessary prices. Implied volatilities are then retrieved through the algorithm outlined by Fabien Le Floc'h in `http://chasethedevil.github.io/post/implied-volatility-from-black-scholes-price/` and implemented in Python.

Second, NN_2 is created, but not (immediately) trained. As already explained, this second neural network will be trained only after being composed with the trained NN_1, which will be marked as *non-trainable* in this second phase. This composed NN is called NN_3. In order to learn the operator $\mathcal{I}$, NN_3 will be trained and, as a side result, NN_2 will be also trained. That is to say that NN_3 is just used as a mere tool to arrive to get NN_2 trained as well. Finally, it will be then separated from NN_1. As already said, the goal of NN_3 is basically learning the identity function. Thus, the input of NN_2 is the following:

$$\theta, \sigma, \rho, \{\mathrm{IVS}_i\}_{i=1}^{130},$$

while the output, since we have to learn the Lévy process L, is

$$\{\nu_i\}_{i=1}^{5}, \{\delta_i\}_{i=1}^{5}.$$

From an architectural viewpoint, NN_2 has 4 "main" hidden layers with 1024 nodes each. The activation function is ELU, as for NN_1. While training NN_3 (and, implicitly, NN_2), we have to provide as output the entire implied volatility structure $\{\mathrm{IVS}_i\}_{i=1}^{130}$, while as input the concatenation of the complete input of NN_2, plus the incomplete input of NN_1, that is everything listed above apart from $\{\nu_i\}_{i=1}^{5}, \{\delta_i\}_{i=1}^{5}$, which have to be guessed during the training. To obtain a satisfactory training procedure, we tried also different activation functions for the output layer of NN_2. In the end, the best results were reached using the standard sigmoid function stretched to completely cover the intervals[11] in which ν_i and δ_i were (randomly) extracted. Without this precaution the training process could not converge to a reliable result.
For both training processes, we used the mean squared error on the implied volatilities as loss function, since we were dealing with regression-type tasks (other loss functions were tried, but they gave birth to NNs that were operating more poorly). To obtain better results, it was really helpful also the linear transformation operated on the input and output data: outside of the implied volatilities which were kept unchanged, all other quantities were scaled to reside in the

[10] With this, we mean that between the predefined hidden layers we only find one time the application of the activation function (basically, another hidden layer). The situation might be clearer by looking at Figure 3.2.

[11] Instead of the interval $[0, 1]$ which represents the codomain of the function.

interval $(0, 1)$. Moreover, as it is possible to see from Figure 3.2, we made use of batch normalization ([IS15]), while we avoided drop-out ([Sri+14]). Training has been performed using the ADAM algorithm ([KB15]). The best batch size for both training processes was 1'000 (out of a database made of around 600'000 elements). All hyperparameters have been selected after tuning the networks, using not only manual adjustments, but also other techniques like *randomized search*. The whole neural network architecture was developed using Tensorflow ([Aba+15]) and Keras ([Cho+15]). A schematic representation can be found in Figure 3.3.

3.6.4 Graphical results

In this subsection, we report some of the pictures produced using the model outlined in Section 3.5 with Python and the graphical package Matplotlib[12].
In the first case, plotted figures represent a 3D representation of implied volatility surfaces together with a heat-map reporting the (pointwise) differences between the original implied volatility surface (Original IVS) and the one obtained by application of neural network NN_1 (New IVS), which takes parameters in input as return the IVS on a grid given by 13 moneyness (or strikes) and 10 maturities. The heat-map was produced using the command pcolormesh. All (input) parameters were randomly generated from a uniform distribution, the same used for the generation of training data (but of course not used during the training process). For sake of simplicity, here we calibrated just one couple (ν, δ) for each volatility surface.

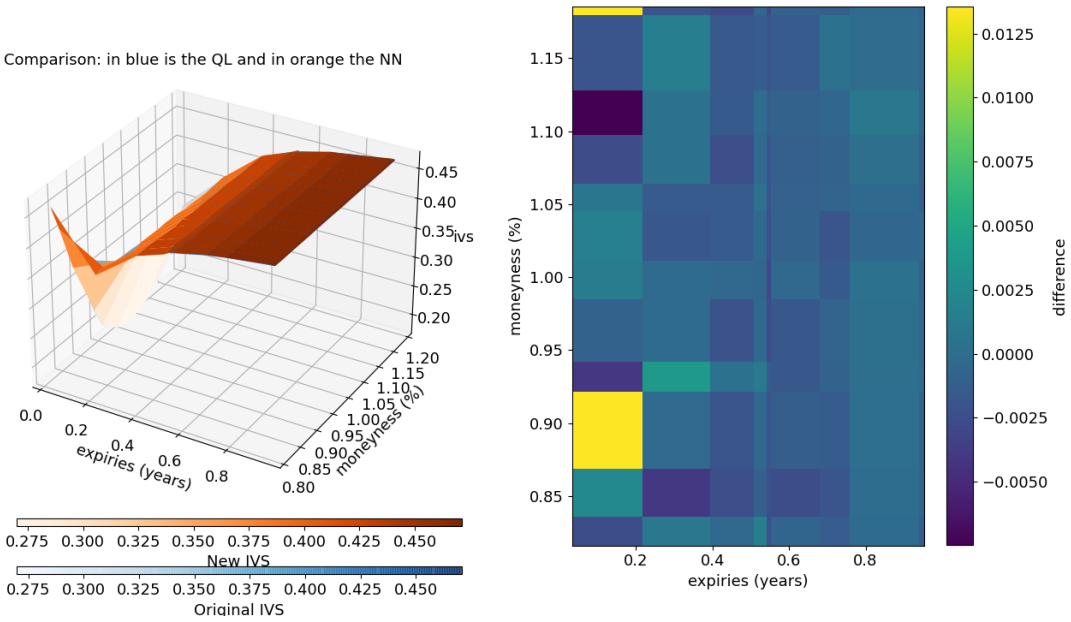


Figure 3.4: Parameters: $S_0 = 100$, $r = 0.0205$, $q = 0.03$, $V_0 = 0.0001$, $\kappa = 7.797$, $\theta = 0.247$, $\sigma = 0.280$, $\rho = 0.042$, $\lambda = 0.081$, $\nu = 0.159$, $\delta = 0.205$

[12] J. D. Hunter, "Matplotlib: A 2D Graphics Environment", Computing in Science & Engineering, vol. 9, no. 3, pp. 90-95, 2007

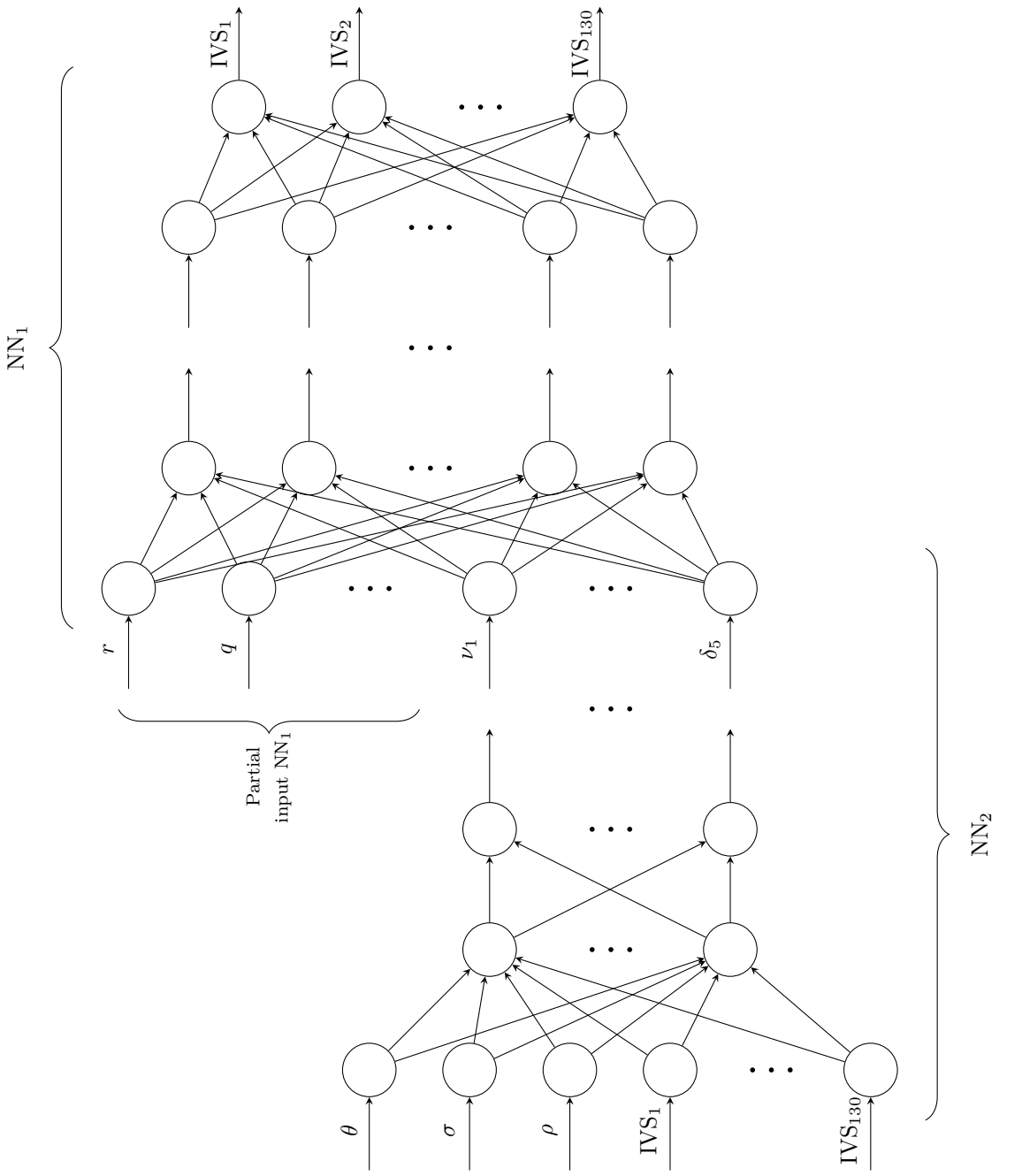


Figure 3.3: Representation of NN_3 as a fully-connected feedforward neural network (residual cells are ignored).

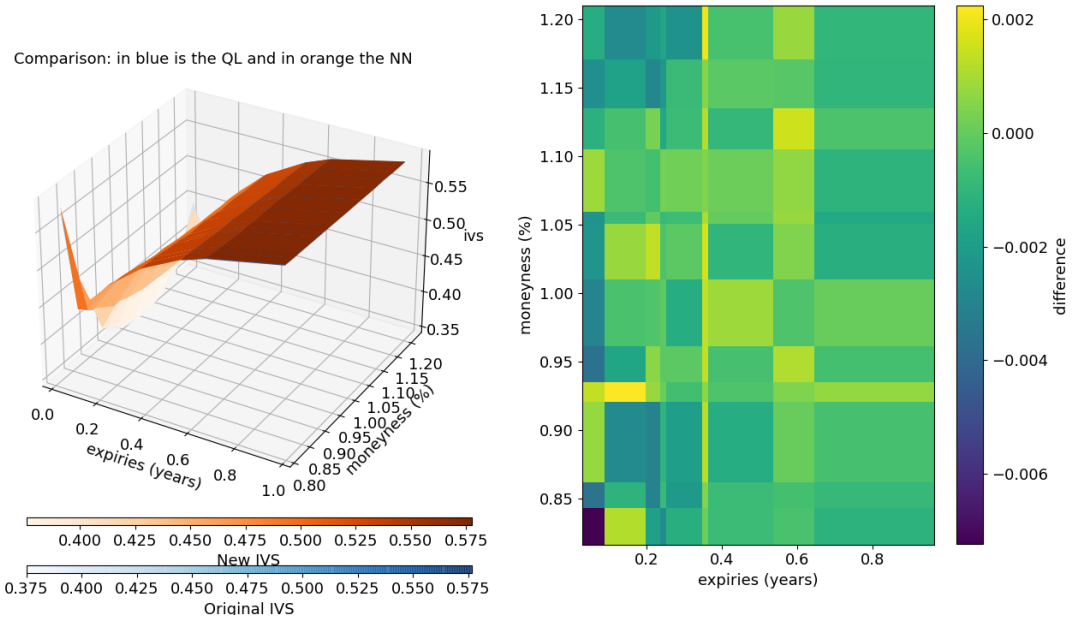


Figure 3.5: Parameters: $S_0 = 100$, $r = 0.0068$, $q = 0.0161$, $V_0 = 0.0951$, $\kappa = 5.421$, $\theta = 0.370$, $\sigma = 0.224$, $\rho = 0.242$, $\lambda = 0.289$, $\nu = 0.087$, $\delta = 0.249$

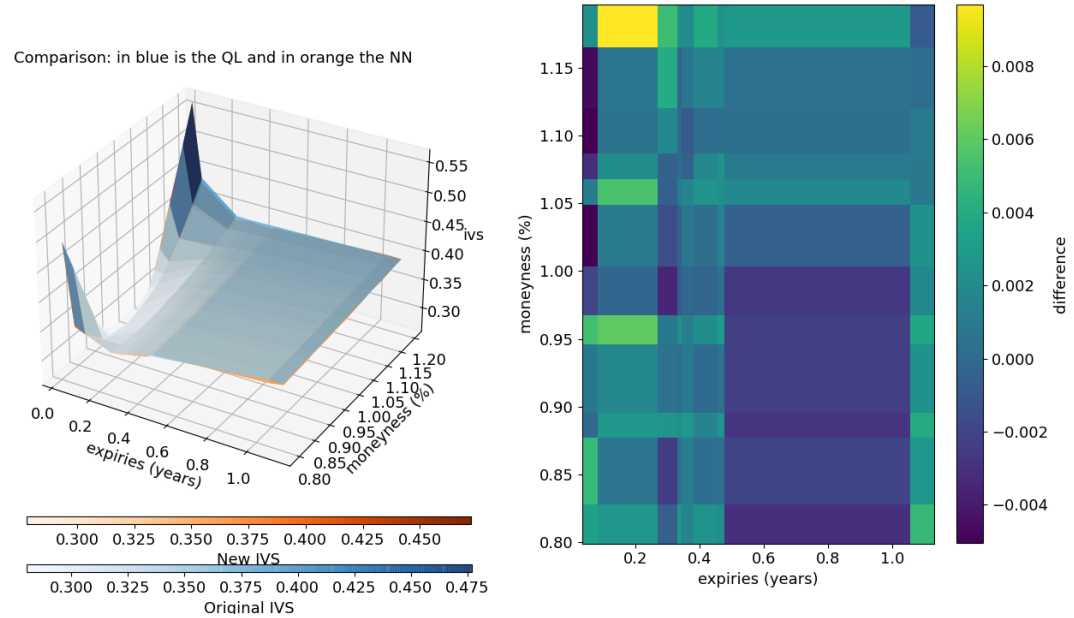


Figure 3.6: Parameters: $S_0 = 100$, $r = 0.0111$, $q = 0.0021$, $V_0 = 0.0552$, $\kappa = 8.698$, $\theta = 0.106$, $\sigma = 0.391$, ρ = -0.12, $\lambda = 0.491$, ν = -0.202, $\delta = 0.287$

On the other hand, in the next figures we will take into consideration the neural network we called in Section 3.6 NN_2. Figures 3.7, 3.8 and 3.9 were basically obtained after one loop of Algorithm 3, in the sense that we started from an IVS generated by a model in which the price process and the variance process evolved in time (step 3 in the algorithm), we let the 3 parameters θ, σ and ρ changing according to exogenous dynamics (essentially, normal noise) and then we exploited NN_2 to recover the jump parameters that would give rise to the same IVS (IVS_{new} in the same algorithm). Note that the 'Original IVS' (in the plots) are obtained for Figures 3.7 and 3.8 in an analytical way, while in Figure 3.9 the 'Original IVS' is the output of NN_1. The fact that the error is zero everywhere, although the starting and derived (by NN_2) parameters are not the same, means that the inversion of the neural network is indeed effective.

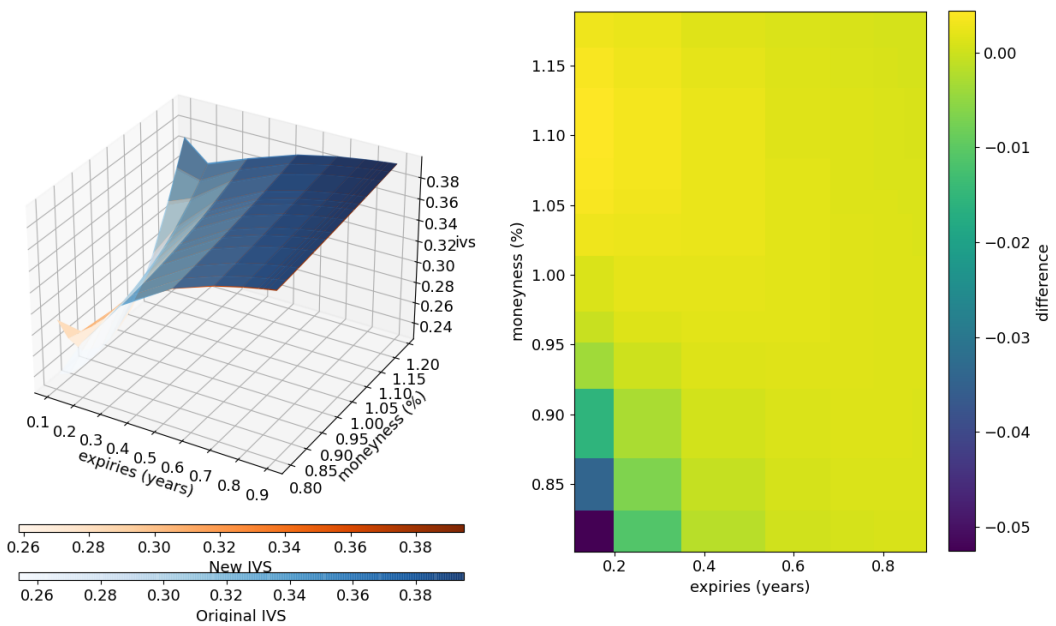


Figure 3.7: Parameters: $S_0 = 99.783$, $r = 0.0521$, $q = 0.0082$, $V_0 = 0.0037$, $\kappa = 6.924$, $\theta = 0.146$, $\sigma = 0.328$, $\rho =$ -0.08, $\lambda = 0.295$, $\nu =$ -0.286, $\delta = 0.211$; after 1 loop and using NN_2 $\theta = 0.142$, $\sigma = 0.339$, $\rho =$ -0.08, $\nu =$ -0.238, $\delta = 0.299$

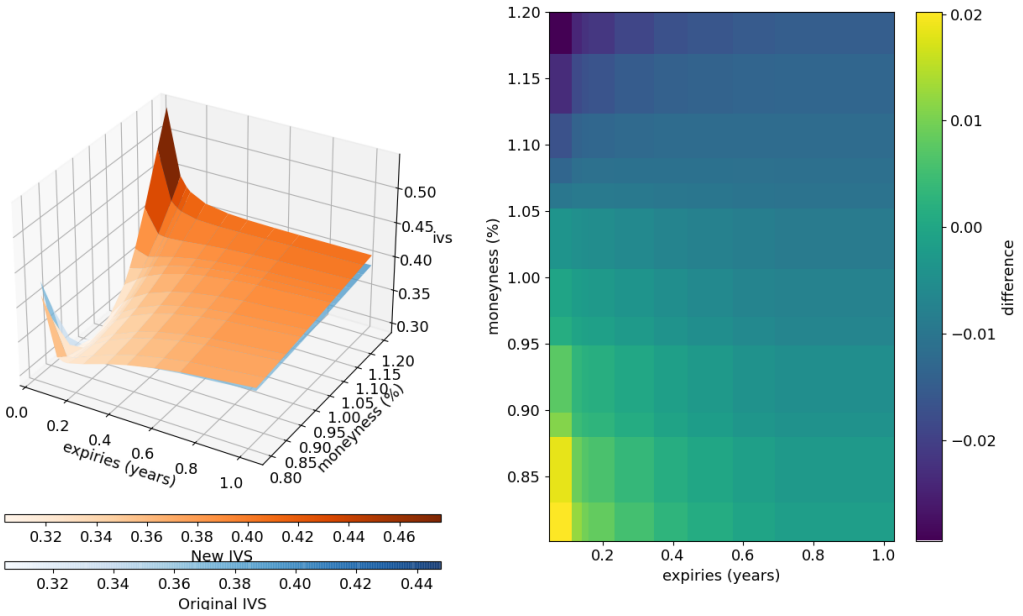


Figure 3.8: Parameters: $S_0 = 100.26$, $r = 0.0111$, $q = 0.0021$, $V_0 = 0.0663$, $\kappa = 8.698$, $\theta = 0.106$, $\sigma = 0.391$, $\rho =$ -0.12, $\lambda = 0.491$, $\nu =$ -0.202, $\delta = 0.287$; after 1 loop and using NN_2 $\theta = 0.102$, $\sigma = 0.408$, $\rho =$ -0.12, $\nu =$ -0.279, $\delta = 0.300$.

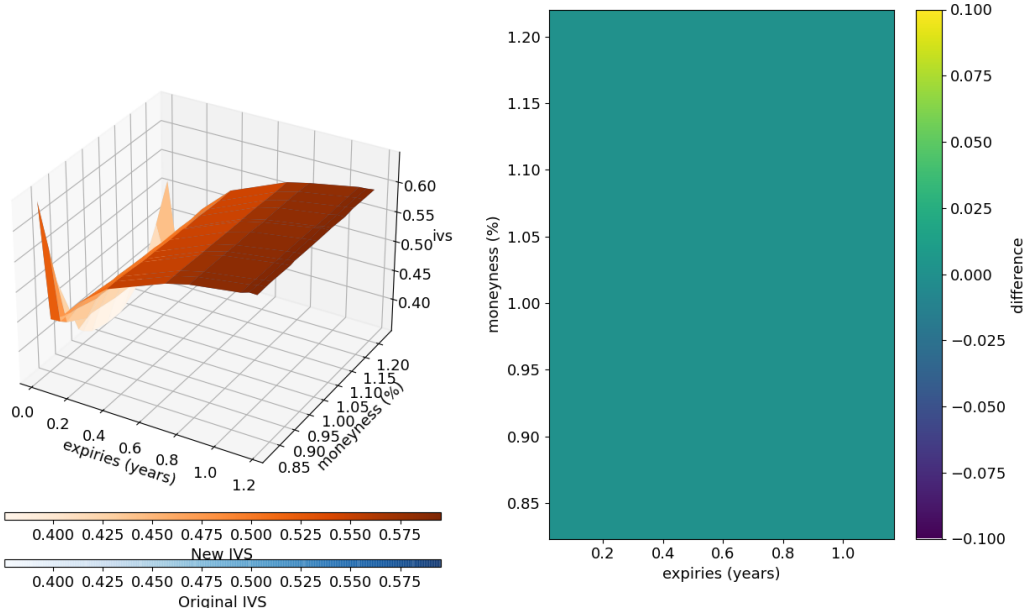


Figure 3.9: Parameters: $S_0 = 100.83$, $r = 0.0068$, $q = 0.0161$, $V_0 = 0.1046$, $\kappa = 5.421$, $\theta = 0.370$, $\sigma = 0.224$, $\rho = 0.242$, $\lambda = 0.289$, $\nu = 0.087$, $\delta = 0.249$; after 1 loop and using NN_2 $\theta = 0.357$, $\sigma = 0.218$, $\rho = 0.251$, $\nu = 0.289$, $\delta = 0.300$.

3.6.5 A side result: moving IVS

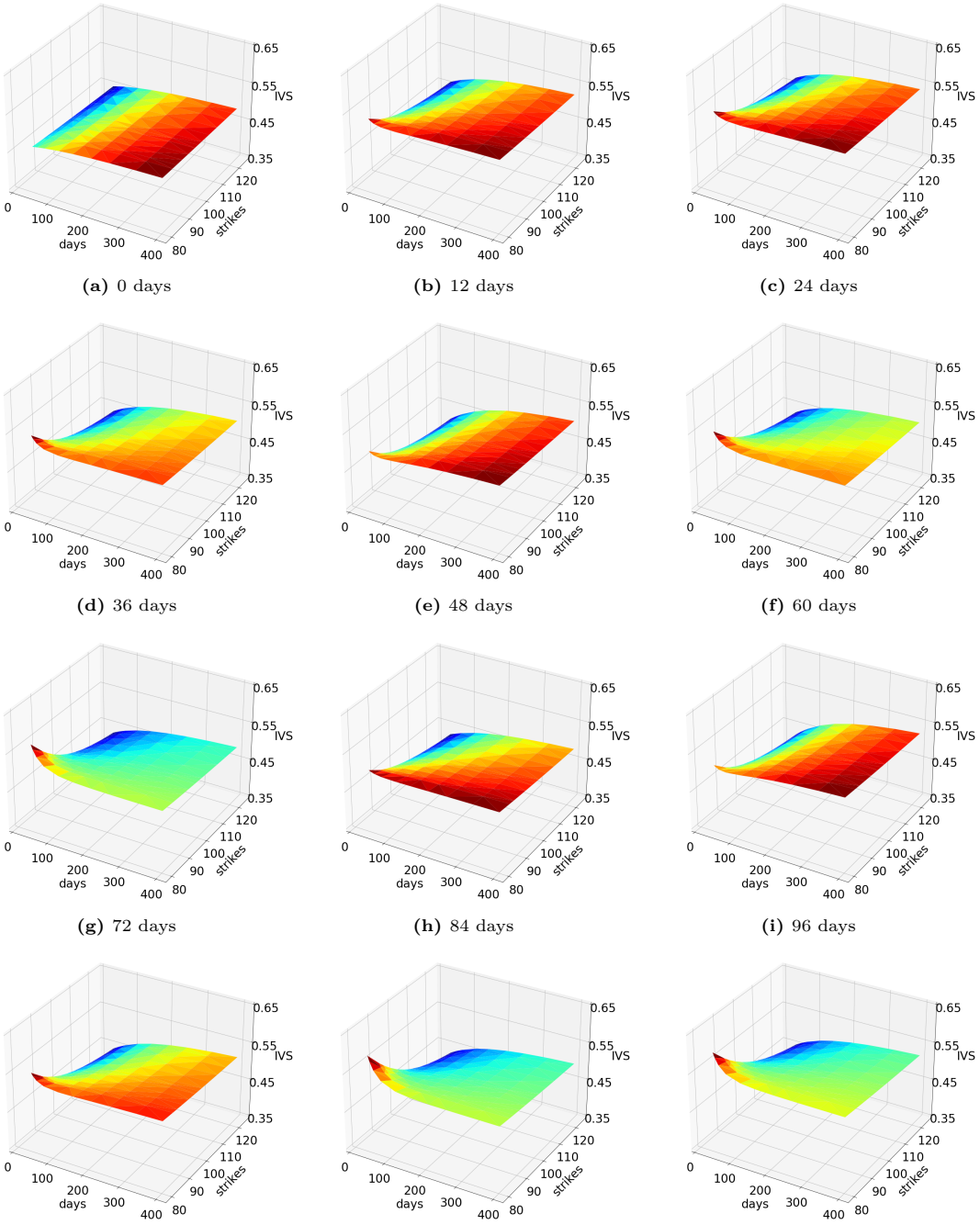
Having a concrete numerical tool that allows the recalibration of our model online and basically in an instantaneous way has another "cheerful" consequence. Let us imagine, for the moment, that the dynamics of the parameters p are known. If this is the case, then we can model for indefinite time the evolution of an implied volatility surface without breaking any arbitrage constraints, neither static nor dynamic ones. To our knowledge, it is the first time that this achieved in an efficient way, one impressive other implementation has been presented in [CMN17]. The algorithm used to accomplish that is outlined in Algorithm 3.

Algorithm 3: No-arbitrage evolution of IVS

1 Pick initial values for the state variables (value of asset X and variance V), the Heston parameters (θ, σ, ρ and k), the jump-frequency $\sim \text{Pois}(\lambda\, dt)$ and the jump-size normal distribution $\sim \mathcal{N}(\nu_i, \delta_i^2)$ for $i = 1, \dots, 5$. Parameters λ and κ remain fixed throughout the procedure;
2 Compute the implied volatility surface (IVS) given the initial values;
3 **Bates step**: update the two state variables X and V in X_{new} and V_{new};
4 Compute the new implied volatility surface IVS_{new} given X_{new} and V_{new};
5 **Heston-parameter step**: update the three parameters θ, σ, ρ according to an exogenously given dynamics and obtain θ_{new}, σ_{new}, ρ_{new};
6 Given IVS_{new} together with θ_{new}, σ_{new}, ρ_{new}, compute the new parameters $(\nu_i^{\text{new}}, \delta_i^{\text{new}})_{i=1}^5$ such that the IVS obtained with X_{new}, V_{new}, ρ_{new}, θ_{new}, σ_{new}, λ, k and $(\nu_i^{\text{new}}, \delta_i^{\text{new}})_{i=1}^5$ equals to IVS_{new});
7 Overwrite the initial parameters with the new parameters (having the sub/super-script *new*);
8 Restart from point 3.

As already written in the algorithm and for our purposes, we decided to initially pick randomly the parameters θ, σ, ρ, but then letting them evolve according to very simple dynamics, namely adding some noise to the current value to get the new one. The variance of the Gaussian noise has been chosen relatively small and values are scaled if they overcome a certain threshold, so that the relative change (with respect to the initial value) could not exceed 5%. In addition, we made sure that the Feller condition was always satisfied and that the values could not exit the natural domains we assigned them. For example, if the correlation ρ were brought outside of the interval $[-1, 1]$, then we would force it to remain inside by collapsing the value to the closest extreme. For both θ and σ the interval $[0.01, 0.5]$ was chosen.
Notice also that Steps 2, 4 and 6 of Algorithm 3 are made by neural networks, NN_1 for 2 and 4, while NN_2 for Step 6.
The output of Algorithm 3 can be found in Figure 3.10, with dt is equal to one day and where only selected days are shown.

Solving the same problem with the desired precision without neural networks would have required an immense computational power, since the inverse problem is notably ill-posed and the regularized inverse problem has to be solved at any point in time along the discretization grid. This is possible on a standard laptop thanks to these techniques. Finally, it is important to underline that we do not break any arbitrage condition because the CNKK drift condition is fully incorporated in the steps of Algorithm 3.



(a) 0 days **(b)** 12 days **(c)** 24 days

(d) 36 days **(e)** 48 days **(f)** 60 days

(g) 72 days **(h)** 84 days **(i)** 96 days

(j) 108 days **(k)** 120 days **(l)** 132 days

Figure 3.10: Evolution of an IVS with initial parameters S_0=100, r=0, q=0, V_0=0.20, κ=4.7, θ=0.25, σ=0.2, ρ=-0.4, λ=0.28, ν=0.01, δ=0.01, maturities=$\{45, 50, 60, 80, 120, 160, 210, 260, 330, 400\}$, moneyness=$\{0.81, 0.83, 0.86, 0.90, 0.94, 0.98, 1, 1.02, 1.06, 1.1, 1.14, 1.17, 1.25\}$.

3.6.6 Python code

In the following, Listing 3.1 shows the implementation of the arbitrage-free evolution for the implied volatility surface in time. One can clearly distinguish the role of the two different neural networks `NN1` and `NN3` that represent the direct map between model parameters and IVS and the inverse map, respectively. A series of pictures is then saved at each instant of time to eventually produce a nice movie-animation.

```
1
2  def arbitrage_free_evol(self, NN1, NN3, transf_lin, seed=1, nb_times=1000, useNN1=
       True, **kwargs):
3      """
4      Arbitrage Free Evolution of the IVS.
5      :param NN1: NN mapping params to ivs
6      :param NN3: NN mapping ivs and Heston params to Levy params
7      :param transf_lin: linear transformation
8      :param seed: optional, seed
9      :param nb_times: optional, number of iterations
10     :param useNN1: optional, use NN1 for direct mapping or analytical mapping?
11     :return: video with ivs that evolves in time without breaking arbitrage
       constraints
12     """
13     np.random.seed(seed)
14
15     # Initial parameters for time 0
16     rd = self.dates_ql[0]
17     bat_obj = self.create_BatesTDJ_model(date = rd)
18     ir = bat_obj.risk_free_rate
19     dr = bat_obj.dividend_rate
20     lambd = bat_obj.lambd
21     kappa = bat_obj.kappa
22     delta_times = np.copy(bat_obj.ttm*365.).astype(int)
23
24     # Input for NN1
25     params = self.combine_params_NN1(bat_obj)
26     # IVS
27     if useNN1:
28         ivs = NN1.predict(np.array(params)).ravel()
29     else:
30         ivs = np.array(self.obtain_vola(bat_obj))
31
32     obs_str, params_str = self.params_string(bat_obj=bat_obj)
33     plot_surface(Z=np.array(ivs), z_label='ivs', main_title='BatesTDJ IVS',
       string1=', '.join(o for o in obs_str), string2=', '.join(p for p in params_str
       ), delta_times=delta_times.tolist(), strikes=bat_obj.strikes, folder='Images',
        counter=0, seed=seed, **kwargs)
34
35     i = 1
36     while i <= nb_times:
37         theta = bat_obj.theta
38         sigma = bat_obj.sigma
39         rho = bat_obj.rho
40         moneyness = bat_obj.moneyness
41         S_new, v_new = bat_obj.bates_tdj_step()
42         # Update moneyness to keep strikes constant
43         moneyness_new = moneyness * S_new / bat_obj.spot_price
44         print('New spot price: ', S_new)
45         bat_obj = self.create_BatesTDJ_model(date=rd, ir=ir, dr=dr, sp=S_new, v0=
       v_new, rho=rho, sigma=sigma, theta=theta, kappa=kappa, lambd=lambd, nu=bat_obj
       .nu, delta=bat_obj.delta, delta_times=delta_times, moneyness=moneyness_new)
46         params = self.combine_params_NN1(bat_obj)
```

```
        print('params :', params)

        # Obtain IVS (there are 2 possibilities)
        input_NN1 = np.array(params)
        if useNN1:
            ivs = NN1.predict(input_NN1, goal='find_iv').ravel()
        else:
            ivs = np.array(self.obtain_vola(bat_obj))

        # Plot the IVS
        obs_str, params_str = self.params_string(bat_obj=bat_obj)
        plot_surface(Z=np.array(ivs), z_label='ivs', main_title='BatesTDJ IVS',
    string1=', '.join(o for o in obs_str), string2=', '.join(p for p in params_str
    ), delta_times=delta_times.tolist(), strikes=bat_obj.strikes, folder='Images',
     counter=i, seed=seed, **kwargs)

        # Update the parameters rho, theta, sigma
        std_dev = 0.05
        noise = np.random.normal(loc=0, scale=std_dev, size=(1, 3))
        # Noise must be proportional to parameter absolute value
        noise[:, 0] = self.__control_noise_level(noise=noise[0, 0], param=theta)
        noise[:, 1] = self.__control_noise_level(noise=noise[0, 1], param=sigma)
        noise[:, 2] = self.__control_noise_level(noise=noise[0, 2], param=rho)
        # Clip so they do not go outside of chosen bounds
        params_old = np.array([theta, sigma, rho])
        params_new = np.clip(params_old+noise, batdj_lhs_min[5:8], batdj_lhs_max
    [5:8]).squeeze()
        print('params_new: ', params_new)

        # Feller condition
        while 2*kappa*params_new[0] <= params_new[1]**2:
            print('Feller condition not satisfied.')
            params_new[0:2] = self.control_Feller(theta, sigma, std_dev)
            # Clip again
            params_new = np.clip(params_new, batdj_lhs_min[5:8], batdj_lhs_max
    [5:8])

        # Input for NN3
        input_NN1 = input_NN1.reshape(1, -1)
        ivs = ivs.reshape(1, -1)
        input_NN3, partial_nn1_input = training_data_NN3(x=input_NN1, y=ivs, save=
    False)
        input_NN3 = transf_batdj_NN3_in2(input_NN3, transf_lin.transform)
        jump_params = NN3.predict(input_NN3).squeeze()
        print('jump_params:', jump_params)
        print('Is S_new the same as bat_obj.spot_price?', S_new == bat_obj.
    spot_price)
        nu_new = jump_params[:nb_js_params].astype(float)
        delta_new = jump_params[nb_js_params:].astype(float)
        bat_obj = self.create_BatesTDJ_model(date=rd, ir=ir, dr=dr, sp=S_new, v0=
    v_new, rho=params_new[2], sigma=params_new[1], theta=params_new[0], kappa=
    kappa, lambd=lambd, nu=nu_new, delta=delta_new, delta_times=delta_times,
    moneyness=moneyness_new)
        i += 1
    print(i)
    # Create video: Need to import pkgs os, glob, subprocess
    video_name = 'IVS_seed'+str(seed)+'_times'+str(nb_times)+'_stddev'+str(std_dev
    )
    if useNN1:
        video_name = video_name + '_NN1'
    video_name = video_name + '.mp4'
    os.chdir('Images')
```

```
98      subprocess.call([
99          # framerate==2 means 2 frames per second
100         'ffmpeg', '-y', '-framerate', '4', '-i', 'file%02d_seed'+str(seed)+'.png',
          '-pix_fmt', 'yuv420p', video_name])
101     for file_name in glob.glob('*.png'):
102         os.remove(file_name)
```

Listing 3.1: Code for implementation of the IVS no-arbitrage evolution. This function is actually a public method of the class that handles Bates models (this is the reason for the `self` among the function arguments).

3.7 Finite dimensional realizations for CNKK equations

For completeness, we formulate a geometric interpretation of CRC models gathering technical material from [FT02] and [FT03] on which we heavily rely. In the sequel, we shall consider the classical problem of finding (minimal) finite dimensional realizations that contain solution, in a mild sense, to the CNKK SPDE (3.21). Let us take into account particular vector fields σ, which only depend on the state of the forward characteristic θ via a tenor $0 \leq x_1, \ldots, x_n$ of times-to-maturity, through continuous linear functionals ℓ. In particular, we would like to find conditions under which these kind of vector fields σ will lead to an affine solution of the CNKK SPDE.

Recall that G is a Hilbert space of continuous complex-valued functions defined on the strip $-\mathrm{i}[0,1]^n \times \mathbb{R}^n$, i.e. $G \subset C((-\mathrm{i}[0,1]^n) \times \mathbb{R}^n; \mathbb{C})$ and H a Lévy codebook Hilbert space as in Definition 3.4.1. Theory is developed on the Fréchet space defined as domain of the operator A^∞:

$$D(A^\infty) := \bigcap_{n \in \mathbb{N}} D(A^n),$$

equipped with the family of seminorms

$$p_n(h) = \sum_{i=0}^{n} \left\| A^i h \right\|_H, \qquad \text{for all } n \in \mathbb{N}_0.$$

Note that in this setting the operator A acts as a *bounded* operator on $D(A^\infty)$ ([FT02]). To specify the mathematical framework, we need some definitions.

Definition 3.7.1 (Local CNKK model)**.** *Let $\mathcal{U}$ be a convex open set in H. A (local) CNKK model in $\mathcal{U}$ is formed by a set of maps $(\sigma_1, ..., \sigma_d) : \mathcal{U} \to H$ such that* (3.21) *admits a unique $\mathcal{U}$-valued (local) solution for every initial term structure $\theta_0 \in \mathcal{U}$.*

Definition 3.7.2 (Finite dimensional realization)**.** *We say that $(\sigma_1, ..., \sigma_d)$ admits an n-dimensional realization around θ_0 if there exists an open neighbourhood $V \ni \theta_0$ in $\mathcal{U} \cap D(A^\infty)$, an open set U in $\mathbb{R}_{\geq 0} \times \mathbb{R}^{n-1}$, and a $\mathcal{C}^\infty$-map $\alpha : U \times V \to \mathcal{U} \cap D(A^\infty)$ such that*

1. *$\forall \theta \in V, \theta \in \alpha(U, \theta)$;*
2. *for every $(z, r) \in U \times V$, $D_z(z, r) : \mathbb{R}^n \to D(A^\infty)$ is injective;*
3. *for all $(z_i, r_i) \in U \times V$, $\alpha(z_1, r_1) = \alpha(z_2, r_2)$ implies $D_z\alpha(z_1, r_1)(\mathbb{R}^n) = D_z\alpha(z_2, r_2)(\mathbb{R}^n)$ (same tangent spaces);*

4. *for every $\theta_\star \in V$ there exists a U-valued diffusion process Z and a stopping time $\tau > 0$ such that*

$$\theta_{t\wedge\tau} = \alpha(Z_{t\wedge\tau}, \theta_\star) \tag{3.31}$$

is the (unique) local solution of (3.21) *with $\theta_0 = \theta_\star$.*

From the above definition, one can recognise that α is actually the parametrization of an n-dimensional submanifold with boundary M_θ in $\mathcal{U} \cap D(A^\infty)$. Thus, we can rewrite Equation (3.31) as $\theta_{t\wedge\tau} \in M_{\theta_\star}$ and we can look for the solution being in a (sub)manifold.

Definition 3.7.3 (Locally invariant manifold)**.** *A manifold M of $\mathcal{U} \cap D(A^\infty)$ is said to be* locally invariant *for Equation* (3.21) *if for every $\theta_\star \in M$ the local solution θ_t of* (3.21) *satisfies $\theta_{t\wedge\tau} \in M_{\theta_\star}$ for some stopping time $\tau > 0$.*

An essential role is played by *Banach maps*:

Definition 3.7.4 (Banach map)**.** *Given a Fréchet space E, a smooth (i.e. C^∞) map $P : E \supset U \to E$ is called a* Banach map *if there exist smooth (not necessarily linear) maps $R : E \supset U \to B$ and $Q : B \supset V \to E$ such that $P = Q \circ R$, where B is a Banach space and U and V are open sets.*

We are now ready for the following definition:

Definition 3.7.5 (Tenor-dependent volatility)**.** *We call the volatility vector fields $\sigma_1, \dots, \sigma_d$ of a CNKK equation* tenor-dependent *if*

- *we have that*

$$\sigma_i(\theta) = \phi_i(\ell(\theta)), \quad 1 \le i \le d,$$

where $\ell \in L(H, G^p)$, for some $p \in \mathbb{N}$, and $\phi_1, \dots, \phi_d : G^p \to D(A^\infty)$ are smooth and pointwise linearly independent maps. Moreover, since it can be shown (Lemma 4.4 in [FT03]) that the drift is also a Banach map, we set

$$\mu_{\text{CNKK}}(\theta) = \phi_0(\ell(\eta)),$$

where $\phi_0 : G^p \to D(A^\infty)$ is smooth. We usually have to assume $\ell_1(\eta) = \eta(0, .)$;

- *for every $q \ge 0$, the map*

$$(\ell, \ell \circ (d/dx), \dots, \ell \circ (d/dx)^q) : D\left((d/dx)^\infty\right) \to G^{p(q+1)}$$

is open.

Before continuing, it might be useful to recall some concepts from geometry.

Definition 3.7.6 (Distribution)**.** *Given a Fréchet space E, a* distribution *on U open subset of E is a collection of vector subspaces $D = \{D_f\}_{f\in U}$ of E. A distribution D on U is said to be* involutive *if for any two locally given vector fields X_1, X_2 with values in D (i.e. $X_i(f) \in D(f)$ for any $f \in U$) the Lie bracket $[X_1, X_2]$ has also values in D.*

Definition 3.7.7 (Weak foliation)**.** *A* weak foliation *$\mathscr{F}$ of dimension n on an open subset U of a Frechét space E is a collection of submanifolds with boundary $\{M_r\}_{r\in U}$, usually called* leaves*, such that*

i) for all $r \in U$ *we have* $r \in M_r$ *with* $\dim(M_r) = n$,

ii) the distribution

$$D(\mathscr{F})(f) := \operatorname{span}\{T_f M_r \,|\, r \in U \text{ with } f \in M_r\}$$

has dimension n *for all* $f \in U$, *i.e. given* $f \in U$ *the tangent spaces* $T_f M_r$ *agree for all* $M_r \ni f$. *This distribution is called* tangent distribution *of* $\mathscr{F}$.

In general, we say that any distribution D *is* tangent *to* $\mathscr{F}$ *if* $D(f) \subset D(\mathscr{F})(f)$ *for all* $f \in U$.

Finally, if define $\mu(\theta) := A\theta + \mu_{\text{CNKK}}(\theta) - \frac{1}{2}\sum_{j=1}^{d} D\sigma_j(\theta)\sigma_j(\theta)$, let us remind the reader about the fact that the set generated by all multiple Lie brackets of the vector $\mu, \sigma_1, \ldots, \sigma_d$ generates a Lie algebra, denoted as D_{LA}. Moreover, let us define D as the distribution given by the same vector fields, that is $D := \operatorname{span}\{\mu, \sigma_1, \ldots, \sigma_d\}$.

All these concepts are extremely important since they enter the formulation of a weak version of Frobenius theorem (see Theorem 3.9 and Proposition 4.8 in [FT03] for a proof), connecting elements from algebraic and geometric theories.

Theorem 3.7.8. *Let* U *be an open set in* $D(A^\infty)$ *and* $\mathscr{F}$ *an* n*-dimensional weak foliation on* U, *for* $n \in \mathbb{N}$. D *is involutive* $\iff$ D *is tangent to* $\mathscr{F}$.

Being involutive is equivalent to saying that $D_{\text{LA}} \subset D(\mathscr{F})$ on U. Therefore, the boundedness of $\dim(D_{\text{LA}})$ is necessary for the existence of a weak foliation on U. If we assume that D_{LA} has constant and finite dimension N_{LA} on the U open and connected set, then we can also prove next theorem (Theorem 4.10 in [FT03]):

Theorem 3.7.9. *Let* U *be an open and connected subset of the Fréchet space* $D(A^\infty)$ *such that* $\dim(D_{\text{LA}}) = N_{\text{LA}}$ *on it. For a tenor dependent volatility structure, there exist linearly independent constant vectors* $\lambda_1, \ldots, \lambda_{N_{\text{LA}}-1}$ *such that for* $1 \le i \le d$ *on* U

$$D_{\text{LA}} = \operatorname{span}\{\mu, \lambda_1, \ldots, \lambda_{N_{LA}-1}\} \text{ and } \sigma_i(\theta) \in \operatorname{span}\{\lambda_1, \ldots, \lambda_{N_{\text{LA}}-1}\}.$$

If we introduce

$$\Sigma := \big\{\theta \in \mathcal{U} \cap D(A^\infty) \,|\, \mu(\theta) \in \operatorname{span}\{\lambda_1, \ldots, \lambda_{N_{\text{LA}}-1}\}\big\},$$

it is clear that we can only expect the results of Theorem 3.7.9 to be true on at most $\mathcal{U} \cap D(A^\infty) \setminus \Sigma$. It can be shown (again [FT03]), that Σ is closed and nowhere dense in $D(A^\infty)$ and that we have $N_{\text{LA}} = \dim(D_{\text{LA}}) \ge \dim(D) = d + 1$ on $\mathcal{U} \cap D(A^\infty) \setminus \Sigma$. This is the reason why all leaves of our foliation will have dimension $N \ge 2$.

But what is the aspect of these leaves? All forward characteristics remain within the finite dimensional manifold with boundary given by

$$\Big\{F_t(u, \psi_C(u, 0; x)) + \langle R_C(u, \psi_C(u, 0; x)), y\rangle \;\Big|\; (t, u, y) \in \mathbb{R}_+ \times \mathbb{R}^n \times C\Big\},$$

as one can see from (3.25), where the model parameters are fixed. Any $\theta \in \mathcal{U} \cap D(A^\infty) \subset H$ forward characteristic will imply a system of (Riccati) ODEs which will intersect the leaf at most one time. But if we are able to change the couple (F_t, R_C), varying the model parameters, we could obtain more intersections. From point 3 in Definition 3.7.1 on has that two leaves can only intersect when the tangent spaces coincide and it is exactly on these intersection points that recalibration takes place. In other words, one could say that it is because of these "collision" that (F_t, R_C) is allowed to change, avoiding remaining constant. In this sense, a CRC model

can be seen as the concatenation of different forward characteristics θ on different leaves (coming from different weak-foliations) identified by the functional characteristics (F_t, R_C) which can evolve in time. Every time this selection is operated, the forward characteristic instantiates as a finite dimensional realization in $\mathcal{U} \cap D(A^\infty) \setminus \Sigma$. The HJM model is in this way *tangent* to the Hull-White extended finite factor affine term structure model.

We conclude the section with one last theorem (Theorem 4.13 in [FT03]).

Theorem 3.7.10. *Let $\sigma_1, \dots, \sigma_d$ be a tenor-dependent volatility structure of a CNKK model. Assume furthermore that for initial values in a large enough subset of Γ_n the local mild solutions θ of the CNKK equation leave leaves of a given foliation with constant dimension $N \geq 2$ locally invariant (finite dimensional realization).*

Then there exist $\lambda_1, \dots, \lambda_{N_{\mathrm{LA}}-1}$ such that $\sigma_i(\theta) \in \operatorname{span}\{\lambda_1, \dots, \lambda_{N_{\mathrm{LA}}-1}\}$. This means in particular that there exists a function $A_t : \mathbb{R}^n \times \mathbb{R}_+$ and an $\mathbb{R}^{N_{\mathrm{LA}}-1}$-valued process Z with $Z_0 = 0$ for which

$$\theta_t(u,x) = A_t(u,x) + \sum_{i=1}^{N_{\mathrm{LA}}-1} \lambda_i(u,x) Z_t^i \tag{3.32}$$

up to some stopping time τ, for $x \geq 0$ and $u \in \mathbb{R}^n$.

The stopping time τ is related to the definition of local mild solution (see [Fil01]).

Remark 3.7.11. The affine character of the representation of the solution process θ in (3.32) is apparent. In particular this representation leads via the conditional expectation formula (in case of global solution of the CNKK equation) to affine factor processes Z and a homogenous, time-inhomogenous affine process (X, Y).

3.8 Summary

In this paper, we tried to set up a new rigorous framework in continuous time for the dynamics of volatility surfaces (or cubes, etc. . .), a so called consistent recalibration models, with applications. To do so, we took inspiration from similar work in discrete time by Richter and Teichmann [RT17] and another paper by Harms *et al.* [Har+18], which builds the theory in continuous time, but focusing on yield curves modelling. With respect to the latter, in our case we have a more complex setting due to the more complex term structure, which is here enriched with a "strike" dimension. This is reflected in what we called CNKK equation, a generalization of the more popular HJM equation, but with considerably more involved drift term. It goes without saying that this made the equation intractable from an analytical point of view.

To overcome this issue, we decided to represent the drift term using neural networks. We therefore proposed a new way of solving the (ill-posed) calibration problem, by exploiting the fact that composition of neural networks is still a neural network and by defining, in this sense, a sort of *inverse* network applying implicit regularization. The same trick could be used for other applications, also in branches other than mathematical finance, to solve inverse problems. The use of neural networks was crucial to make numerical procedures tractable and to get information on the solution of the CNKK SPDE. In this case, we can say that the neural network helped us solving an equation which we could not even write down (explicitly).

Eventually, we could use the same inverse neural network to simulate the evolution in time of an implied volatility surface, in this case generated by a generalized Bates model. To the best of our knowledge, it is the first time this can be achieved for indefinite time without breaking

arbitrage constraints. In this way, we are even implicitly building a realistic equity option market simulator capable of avoiding any form of arbitrage.

It is worth noting that all four points of Box 1.1 have been addressed for the general case of continuous strikes and maturities. The first two points on the absence of static and dynamic arbitrage, respectively, rely on the theoretical considerations by Kallsen and Krühner ([KK15]); the third rests also on results from Section 4.2 in [KK15]. Note, however, that this properties do not cease to hold when moving in the consistent recalibration setting we presented here, by sewing together in a suitable way all the spaced out affine tangent models. Finally, the fourth point counts on machine learning technology, in particular neural networks and their proven flexibility.

Chapter 4

Model Free Deep Hedging

4.1 Which model?

Typically, in financial markets we need closed-form solutions that provide the price of a specific derivative in order to find its value or, of equal importance, to estimate the parameters of a determined model. A single calibration operation might require the evaluation of hundreds, when not thousands, of such contracts. If an analytical formula is not available, then a fast numerical procedure, for example involving numerical integration, is necessary. This explains why affine models became very successful in the past years, in particular thanks to highly efficient Fourier pricing technique developed by Carr and Madan in their seminal paper [CM99]. The requirement for a fast computation is ineluctable and, in fact, it restricts dramatically the set of models we can employ for derivative pricing.
The fitting criterion that is usually chosen is the minimization of the "least-squares error" which can be re-interpreted as the maximization of the likelihood, under the hypothesis that the errors between the model and the market are normally distributed. Moreover calibration is taking place on the most liquid instruments, for which financial data are supposed to be reliable.

Once the best-fitting parameters have been fixed for a specific model, the same model is then used to price other more exotic derivatives, which are usually more illiquid instruments, or also to hedge, by considering the calibrated parameters as the *true* ones. This is justified by the fact that the same pricing measure $\mathbb{Q}$ identified through calibration of the pricing process should be used for the entire market. Resorting to statistical learning lingo, it is clear that this approach completely ignores two kinds of errors (see also [Bac22]):

- **Approximation error**, which does not depend on the selected parameters, but rather on the model pool that was chosen for parametrizing the liquid instruments, regardless whether only underlying asset prices, or asset prices and derivatives (as in *market models*). Note that approximation error is a deterministic function that can be viewed as a consequence of the modelling assumptions. It is thus a consequence of the quant's belief in a model pool. In principle, the error can be reduced if we increase the complexity of the model, in such a way that is able to capture more market features.
- **Estimation error**, which depends, on the other hand, on the specific parameters chosen for the selected model class. In principle, if the model class is able to capture most of the market characteristics, this error could be reduced having more and more observations, but, as opposed to approximation error, it increases if model complexity is augmented. This

kind of error is rather a stochastic function, because it depends on the available data: it could be possible that a different family of financial instrument (randomly sampled by the same underlying distribution) lead to a different set of (calibrated) parameters.

In financial mathematics (see, e.g. [Bur+19]), it is sometimes custom to group these two kinds of errors under the term *model uncertainty* or, with a slight abuse of terminology, *model risk*. In financial jargon, practitioners often used the distinctive - and rather charming - expression *unknown unknown*: we do not simply know which model can best mirror the real world (and we have to live with that). All we can do is gather information from the real world, either in the form of past data or market expectations.
This is clearly different from *financial risk*, which is the risk associated to the real world being stochastic. In other words (and roughly said), we do not know if the price of a stock will increase or decrease on a fixed future day. But in principle, if we believe that our model is reflecting reality, then we can quantify the probability of a particular event. For this reason, this risk is referred to as *known unknown*.

4.1.1 A brief Robust-Finance detour

In order to systematically reduce model uncertainty, different solutions have been developed. One of the most prominent and studied in mathematical finance has been **robust finance**. The archetypal idea is considering a pool of models, usually characterized through different probability measures, and then pricing or hedging looking at min-max solutions of optimization problems.

A classical example may be trying to optimize the utility function computed on the final reward of a portfolio, taking the infimum over a set of probability measure and the supremum over all possible admissible strategies. An illustrative reference is [NN18], where the authors also provide a very well written, detailed survey on the subject.

This approach has become popular after the financial crisis started in 2008. One of the consequence was that researchers started to question the rationality of a unique probability measure used for pricing and hedging, hence opening for more considerations of the so-called Knightian uncertainty, in the sense of Frank Knight [Kni21]. He was among the firsts to point out the difference between what is unknown, but can be quantified, "measured" (in his words), and what is unknown and cannot be exactly estimated, naming the latter *uncertainty* and the former *model risk*.
The purpose of this stream of research is to quantify uncertainty in the choice of pricing models, i.e. probability measures, as opposed to other sources of risk, which can be quantified within a specific pricing model, such as calibration error committed when approximating market elements with model proxies.

More generally, an approach which involves the choice of an a priori set of probability measures, as previously described, is called *quasi-sure*, and has been formalized for a discrete time settings by Bouchard and Nutz in [BN15]. The market is ruled by a set $\mathcal{P}$ of probability measures, not necessarily equivalent, that are given the task of determining which events are relevant and which ones can be ignored (negligible). This formulation is desirable because it merges two different point of views: on one hand, there is no attempt to model the stocks directly, but one sees the distribution of the stocks as partially described by the current prices of the traded options, thus leading to more model-independent methodologies. On the other hand, each $\mathbb{P} \in \mathcal{P}$ is seen as a candidate model on which a suitable selection, a "robust analysis", has to be performed.

Recent developments on robust models for pricing and hedging have been moved forward, for discrete time models, by Burzoni and coauthors, see, for example, [Bur+19] and references

therein. In this and previous works, it is developed a *scenario-based* or *pathwise* approach, in which the agent's beliefs (or models) are equivalent to selecting a set of specific scenarios inside a set of possible scenarios Ω. For instance, if the trader believes that stock prices are continuous, then she will retain all scenarios that describe such continuous trajectories, implicitly admitting Black-Scholes [BS73] and Heston [Hes93] models (to name a few), but discard all the others that omit the desired property. The particular case that entails including all scenarios is often referred to as the model-independent framework. While, for instance, choosing a particular probability measure is a way of operating a selection on the same set Ω.

Both approaches look conceptually attractive from a practical point of view, because they allow interpolation between a complete market model (see Section 4.3.2 for definitions and theorems), where just one measure is present, and a generic "universally acceptable" market, where all probability measures and scenarios, respectively, are allowed. Actually, the two mentioned approaches have been proven to be equivalent by Obłój and Wiesel [OW21] under mild technical assumptions. Moreover, note that both procedures work on "elimination": for [BN15] the concept of equivalence between sets of probability measures is based on having the same negligible (polar) sets[1], and, after all, one needs to remove some probability measures to reduce the entire model complexity; while for [Bur+19] the goal is removing possible trajectories, represented by subsets of Ω, by including additional assumptions. In particular, if a path is deemed impossible by all participants in the market, then this is eliminated.

The problems faced by robust finance are not the only ones in financial markets, even if they are of utter importance. Other incongruities may arise from the flow of events, as time passes. As we all know, observable or reckonable experiences can profoundly influence financial markets and it might be that a calibration executed in the past is not anymore valid for the current situation. This was clearly the case when interest rates began to be negative as a consequence of macro-economic developments[2]. Another more recent example comes from the COVID-19 crisis when futures on oil prices went also negative[3]. These were not completely unforeseeable phenomena, but it might well be possible that the existing models in use in most financial firms at those times were not able to capture these sudden changes.

What to do in this case?

Should we employ the model that was giving the most promising results at the moment of calibration, or should we rather use a new model (maybe coming from a different family of models) which is more suitable for the current situation?

This chapter is organized as follows: in the next section, we revise the work from Dümbgen-Rogers' 'Estimate Nothing', highlighting its main benefits and limitations. Then in Section 4.3 we introduce, for the sake of clarity, the concept of hedging in complete and incomplete markets, laying down the theory that is used in Section 4.4 for Deep Hedging. Moreover, in Section 4.5 we merge Deep Hedging and Estimate Nothing to achieve a robust hedging technique, that is profiting from both approaches and is able to address issues that they singularly entail. We call

[1] Given a measurable space $(\Omega, \mathcal{F})$ and a set of probability measures $\mathcal{P}$ on it, a subset $A \subseteq \Omega$ is said *polar* if $A \subseteq A'$, with $A' \in \mathcal{F}$, and $\mathbb{P}(A') = 0$ for all $\mathbb{P} \in \mathcal{P}$. Any property is said to hold $\mathcal{P}$-*quasi surely* if it holds outside a $\mathcal{P}$-polar set. That is why the name of this approach.

[2] See, for example, `https://www.wsj.com/articles/draghi-says-ecbs-negative-rates-have-been-a-success-1507824716`.

[3] See, for example, `https://www.marketwatch.com/story/oil-prices-went-negative-a-year-ago-heres-what-traders-have-learned-since-11618863839`.

this new methodology Model Free Deep Hedging. Eventually, Section 4.6 provides Python code for the implementation of the crucial aspects of Model Free Deep Hedging.

4.2 Dümbgen-Rogers' 'Estimate Nothing'

As we have seen, in practice, the choice of the model is thus somehow made arbitrarily and, besides such debatable choice, it may even lead to inconsistencies over time.

Box 4.1 – Model choice

This arbitrariness on the model choice depends on a series of constraints that are both of practical and theoretical nature, in the sense that

1. it is heavily dependent on the availability of fast routines for pricing;
2. it normally ignores potential modelling errors, in particular modelling uncertainty, even if this problem can be tackled with robust finance techniques;
3. it depends on a specific instant of time.

Addressing all these three points is not trivial at all. Moritz Dümbgen and Chris Rogers came up in 2014 [DR14] with a brilliant idea building on top of Bayesian methods, probably taking inspiration by the previous work by Bunnin and coauthors [BGR02].

Bayesian inference has a long history and it is common to date back this method to the reverend Thomas Bayes, whose notes were collected by Mr. Price and published posthumously in 1763 [BP63]. The central idea is that we can modify a *prior* beliefs, expressed in mathematical terms with a probability distribution, based on factual experience. The result of such update is another distribution which is usually called *posterior*. This process can then be indefinitely iterated to shape the distribution against observed events. For example, if such a distribution is drawn over different values for the choice of a parameter, then it might be a wise idea to select the parameter as the mode or the mean of the same distribution. This will obviously reduce the amount of information we have on that parameter. For this reason, it might be even wiser to withhold the entire distribution.

The first observation, which was already made in [BGR02], is that Bayesian inference can operate not only by selecting among representative models in one pool, but also among different pools. In fact, it is quite naive to think that the same model - even the same pool of models, such as the stochastic volatility Heston model - can be always employed over time. Common sense suggests that there is no model, among those that provide explicit formulas, that can outperform the others consistently, no matter how complex it is, while it is rather likely that different models will provide the best market description at different times.
At this point, the reader might have already guessed the idea of [DR14]. The authors decided to let many models competing in the market to build a posterior distribution on the models themselves. This distribution is then repeatedly updated to incorporate new information coming from the market. As a consequence, different models will turn out to have a larger posterior, indicating their capabilities to better describe the current market situation. As mentioned before in Section 4.1, "all we can do is gather information from the real world, either in form of past data or market expectations" and this is what is also implemented here. Transition from one instant of time to the next one always comes with a new piece of information.

Models' reaction will be two-fold:

- On one hand, new derivative market prices are observable, for example quoted in the form of an implied volatility surface. This means that in our pool of models, those that are able to reproduce such prices will have a bigger weight compared to those that are not able to reflect them.
- On the other hand, new market prices for the underlyings are observable. Since traditional models always come with price process dynamics, this implies that those models that can better reflect the evolution of the underlying are more likely to be better in imitating the real world.

These two different points of view are exploited at the same time. Note that the first kind of calibration on price surfaces is a standard procedure in mathematical finance, where we wish to find a model that mirrors the current price configuration. The second is rather common in econometrics, where times series are the privileged object of investigation. Moreover, we would like to remark that the second point of view is with respect to the statistical (real world) measure whereas the first point of view is with respect to the pricing measure.

The method has been called 'estimate nothing' because there is no estimation to be done: we just need to specify our pool of models at the beginning and then the same set of models is retained over and over again. Only probabilities on this set are able to change. And eventually, as we will see, no model is selected.

4.2.1 Likelihoods

In order to capture the evolution of market information coming from derivatives and underlyings' prices, we need to introduce a tool from Bayesian statistics, that is likelihood. But before that, let us specify the mathematical settings.

Since the purpose of this methodology is having a consistent model that is possibly executable on computers, we will consider a finite probability space, denoted as $(\Omega, \mathcal{F}, \mathbb{P})$, where $\Omega = \{\omega_1, \dots, \omega_N\}$, the probability measure is positive on all possible scenarios, that is $\mathbb{P}(\{\omega_i\}) > 0$ for $i = 1, \dots, N$, and $\mathcal{F} = 2^\Omega$ is the power set of Ω (discrete σ-algebra).
Observations from the financial market arrive at discrete times $\mathcal{T} := (t_k)_{k\in\mathbb{N}}$, as in reality. The central object of study is a single asset denoted by S, whose log asset is denoted by X, which is consequently discretized in time as $(S_k)_{k\in\mathbb{N}}$. So, if we introduce the flow of information as a stochastic process $I = (I_k)_{k\in\mathbb{N}}$, we can consider the filtration generated by I and set $\mathbb{F} = (\mathcal{F}_k^I)_{k\in\mathbb{N}}$. This means that I_k gathers all market information at time t_k. In our case, we consider information coming from the market under two different data: one is the price of the underlying S and the other is the composed by the set of (derivatives') prices observable in the market, at time t_k, that is P_k^{mkt}. If we consider a classical environment with European options, we can parametrize the set P_k with coordinates given by strikes (or moneyness) and time to maturity. For this reason, we can write $P_k = P_k^{K,T}$ to denote a derivative price P at time t_k with strike K and expiry at time $T > t_k$. The set of strikes Σ_k and maturities $\mathcal{T}_k^M$ characterizing the price surface at t_k can, in principle, vary in time as well, but, for the time being, we will ignore this subtlety, thus removing the index k from Σ_k and $\mathcal{T}_k^M$. Since both sets are finite, we set $\Sigma := \{K_i \in \mathbb{R} : i = 1, \dots, d_K\}$ and $\mathcal{T}^M = \{T_j > 0 : j = 1, \dots, d_T\}$ and, for ease of notation, we consider prices organized in a grid G with dimension $d_K d_T$: $G := \{g_i : i = 1, \dots, d_K d_T\}$, where $g_i = (K_{j_K}, T_{\ell_T})$ for some appropriate j_K, ℓ_T. From now on, we assume that the filtration $\mathbb{F} = (\mathcal{F}_k^I)$ coincides with the filtration generated by $\left(S, \{P^{K_i,T_j}\}_{1\le i\le d_K, 1\le j\le d_T}\right)$.

A crucial assumption that we are making is that the evolution of the asset S is Markovian, even if there is model uncertainty on its exact definition. As previously seen, a standard way of dealing with this uncertainty is considering more models. In our case, again for evident computational reasons, we consider J models establishing a finite pool of model. The models in the pool can reflect our expectations of the market and, in this sense, are concrete instances of our beliefs. It is important to note that such models, in general, will not belong to the same class, specified by the dynamics of the underlying, but different classes are taken into account. More systematically, this means that for every model specifications of the dynamics, we settle different parameter specifications that should span the space of admissible parameters. If we were to use Black-Scholes model as a model class, this would mean considering volatilities inside a plausible interval, e.g. $[0.1, 0.7]$, conveniently discretized.

For every model j, we consider the transition density at time t_k, i.e. $p_{j,k}$ (we take here deliberately a continuous notation with obvious meaning), of the type

$$\begin{aligned} p_{j,k}(x, x') &= \mathbb{P}_j \left(S_{k+1} \in dx' \,|\, S_k = x \right) / dx' \\ &= \mathbb{P}_j \left(S_1 \in dx' \,|\, S_0 = x \right) / dx' = p_j(x, x'), \qquad \text{for } j = 1, \dots, J, \end{aligned} \tag{4.1}$$

because of Markovianity; $\mathbb{P}_j$ denotes the probability with respect to model j.

Remark 4.2.1 (Stochastic volatility models). Note that the transition density considered in (4.1) is not suitable for stochastic volatility models, since the current level of volatility should also be taken into account. The same holds true for models that have other stochastic factors that can evolve randomly in time. One can of course include these into considerations.

Associated to model j, we also have a pricing functional φ_j that, given the spot price of the underlying and the grid G of strikes-maturities, can provide a superimposable grid of prices (or, equivalently, of implied volatilities) for all instants of time t_k. Thus, we have

$$P_k^{(j)} = \varphi_j \left(S_k; \{g_i\}_{i=1}^{d_K d_T} \right), \qquad \text{for } j = 1, \dots, J. \tag{4.2}$$

We can now define the log-likelihood of model j at time t_k as

$$\begin{cases} \ell_0^{(j)} = 0, \\ \ell_k^{(j)} = \beta\, \ell_{k-1}^{(j)} + \log p_j(S_{k-1}, S_k) - \mathcal{Q}(\varphi_j(S_k), P_k^{\text{mkt}}) \end{cases} \tag{4.3}$$

where $\beta \in (0, 1]$ is a parameter that influences the past log-likelihoods, thus decreasing the importance of past observations compared to newer ones; $\mathcal{Q} : \mathbb{R}^{d_K d_T} \times \mathbb{R}^{d_K d_T} \to \mathbb{R}$ is a non-negative definite quadratic form that is supposed to measure the distance between market and model prices. For a discussion on $\mathcal{Q}$, we remind to Section 2.1 in [DR14]. Note that seeing the log-likelihood at time t_k starting from log-likelihood at time t_{k-1} is an instance of Bayesian updating, which is now modified through the factor β.

At this point, we finally summon Bayes' rule to compute the posterior distribution $\pi_k := \pi(t_k)$ at time t_k on the pool of models:

$$\pi_k^{(j)} = \frac{\exp\left(\ell_k^{(j)}\right)}{\sum_{i=1}^{J} \exp\left(\ell_k^{(i)}\right)}, \qquad \text{for } j = 1, \dots, J. \tag{4.4}$$

Remark 4.2.2. The updating iterative process can run indefinitely as long as new information is provided.

Remark 4.2.3. As the authors in [DR14] write, now *everything becomes easy*, in the sense that

- the transition density of S_k is given by $\sum_{j=1}^{J} \pi_k^{(j)} p_j(S_k, \cdot)$,
- the price of any derivative can be re-written as a convex combination of the prices $\gamma_k^{(j)}$ given by model j at time t_k, that is $\sum_{j=1}^{J} \pi_k^{(j)} \gamma_k^{(j)}$,
- the delta-hedge of any derivative can be computed in a similar way: if $\delta_k^{(j)}$ is the delta-hedge for model j at time t_k, then we have $\sum_{j=1}^{J} \pi_k^{(j)} \delta_k^{(j)}$,

It becomes now clear what was stated at the end of the previous section: nothing is estimated, since we only repeatedly update the posterior distribution π.

Remark 4.2.4. There are at least two important differences with the methods from robust finance we saw in Section 4.1.1. In first place, using this methodology, we do not have to choose any model based on our calibration procedure, but it is rather the market that selects the combination of models which better reflects the current conditions. In second place, we do not exclude models: if we removed all models that are not considered plausible, we would not be able to see if their relevance increases in another instant of time.
In this sense, the time aspect of uncertainty can be lifted by this approach.

Remark 4.2.5. Note that the 'Estimate nothing' approach is able to address points 2. and 3. of Box 4.1. Nevertheless, we are quite dependent on fast procedures to provide key quantities (such as prices or Greeks), since the evaluation of all models in the pool might take substantial time.

4.3 Hedging

This section is mainly based on the book by Jeanblanc, Yor and Chesney [JYC12] and on lecture notes from Josef Teichmann [Tei13]. Let us consider a probability space $(\Omega, \mathcal{F}, \mathbb{P})$ with a filtration $\mathbb{F} := (\mathcal{F}_t)_{0 \le t \le T}$ which satisfies the usual conditions. Let us fix a maturity date $T \in (0, +\infty)$. We denote with $S = (S_t^i)_{0 \le t \le T}$ the risky assets, usually modelled as semimartingales[4], for $i = 1, \dots, d$ adapted to $\mathbb{F}$ and with $B = (B_t)_{0 \le t \le T}$ the risk free bank account (or money market account) which evolves according to the deterministic interest rate r_t, i.e. $B_t = \exp(\int_0^t r_s \, ds)$.
With $\varphi = (\varphi_t)_{0 \le t \le T}$ we denote the *trading strategy* or *dynamic portfolio*, made by two components: $\varphi = (\eta, \psi)$. The real-valued process η_t represents the number of unit of the bank account B at time t held by an investor, while ψ_t is $\mathbb{R}^d$-valued and the i^{th} coordinate denotes the quantity of asset S^i held by the same investor at time t. The value process $V(\varphi) = (V_t(\varphi))_{0 \le t \le T}$ is the market-to-market value of the entire portfolio at time t, given by $V_t(\varphi) := \sum_{i=1}^{d} \psi_t^i S_t^i + \eta_t B_t$. Sometimes, the bank account is considered as the asset in position 0 of the portfolio, so to conveniently write $V_t(\varphi) = \langle \varphi_t, S_t \rangle = \sum_{i=0}^{d} \varphi_t^i S_t^i$, with the convention $S_t^0 = B_t$ for all $t \in [0, T]$.
In the following we restrict our attention to *self-financing strategies*:

Definition 4.3.1 (Self-financing strategy)**.** *A* self-financing strategy *is defined as an adapted trading strategy $\varphi = (\eta, \psi)$ such that*

[4] In the following we are assuming that S is locally bounded (note that, for example, all continuous process are). This allows some simplifications such as considering discounted prices processes that are local martingales and not σ-martingales, simplifying the technical settings. To have a better view on the arbitrage theory for general semimartingales in continuous time, we remind the reader to [DS94].

A) *the quantities* $\int_0^T \eta_t \, dB_t$ *and* $\int_0^T \psi_t \, dS_t$[5] *are (almost surely) finite - we can, for example, require that* $\int_0^T |\eta_t| \, dt < +\infty$ *for the first integral;*

B) $V_t(\varphi) = \eta_t B_t + \langle \psi_t, S_t \rangle = \eta_0 B_0 + \langle \psi_0, S_0 \rangle + \int_0^t \eta_u \, dB_u + \int_0^t \psi_u \, dS_u$[5] *(almost surely) for all* $t \in [0, T]$.

Thus, a self-financing strategy does not rely on the withdrawal or injection of money during its execution.

Remark 4.3.2. If we use the bank account as a *numéraire*, we can introduce discounted quantities by dividing with B_t. For the money market account we have $\widetilde{B}_t = B_t/B_t = 1$ for all $t \in [0, T]$, while for the risky assets $\widetilde{S}_t = S_t/B_t$ for $t \in [0, T]$. If we employ discounted quantities, then condition B) of Definition 4.3.1 translates to

$$\widetilde{V}_t(\varphi) = \widetilde{V}_0(\varphi) + \int_0^t \psi_u \, d\widetilde{S}_u \quad \text{(a.s.) for all } t \in [0, T]. \tag{4.5}$$

Remark 4.3.3. In case of self-financing strategy, both η and ψ are predictable processes. Note that predictability of ψ was already tacitly required because of the stochastic behavior of the risky assets.

Remark 4.3.4. We can rewrite condition B) of Definition 4.3.1 in differential form: for all $t \in [0, T]$

$$dV_t(\varphi) = \eta_t \, dB_t + \psi_t \, dS_t,$$

from which we see that changes of value of the portfolio over an infinitesimal time interval are due entirely to changes in value of the assets and not to removal or injection of wealth from outside. If we consider just one risky asset that pays also dividends, we can represent it (into continuous time) as a factor e^{qt}, for $q > 0$, that multiplies S. In this case, we have

$$dV_t(\varphi) = \eta_t \, dB_t + \psi_t(dS_t + qS_t dt).$$

For the time being, we will consider $q = 0$ without loss of generality.

Remark 4.3.5. So far, we have not imposed restrictions on the sign of the trading strategy process φ. This means that *borrowing*, i.e. $\eta_t < 0$, and *short selling*, i.e. $\psi_t^i < 0$ for any $i = 1, \ldots, d$, are allowed.

Remark 4.3.6. Note that there exists a bijection between self-financing trading strategies $\varphi = (\eta, \psi)$ and tuples of the form (V_0, ψ), where $V_0 \in \mathbb{R}$ is the initial value of the portfolio, i.e. $V_0 = V_0(\varphi)$, $V_0 \in L^0(\mathcal{F}_0)$ and ψ predictable and S-integrable. Moreover, for any $t \in [0, T]$, we have

$$\eta_t = \widetilde{V}_t(\varphi) - \sum_{i=1}^d \psi_t^i \widetilde{S}_t^i = V_0 + \int_0^t \psi_u \, d\widetilde{S}_u - \sum_{i=1}^d \psi_t^i \widetilde{S}_t^i.$$

For this reason, in the following we will denote the value of a portfolio as V^ψ meaning that this is obtained through the positions $(\psi_1, \ldots, \psi_d)$ and starting from V_0.

We are now going to show such bijection.

[5]The integral $\int_0^t \psi_u \, dS_u$ denotes *vector stochastic integration*, which may differ from the componentwise stochastic integration $\sum_{i=1}^d \int_0^t \psi_u^i \, dS_u^i$. Since an in-depth study of that topic is beyond the scope of this thesis, we reference to [SC02] for more details. The use of vector stochastic integration is assumed from here on.

Proposition 4.3.7. *Let us define the discount factor* $D_t := (B_t)^{-1} = \exp(-\int_0^t r_u\,du)$. *If* $\varphi = (\eta, \psi)$ *is a self-financing portfolio, then for all* $t \in [0,T]$ *we have*

$$\widetilde{V}(\varphi) = D_t V_t(\varphi) = V_0(\varphi) + \int_0^t \psi_u\, d(D_u S_u).$$

Conversely, if $x > 0$ *and* $\psi = (\psi^1, \dots, \psi^d)$ *is a vector of predictable processes and if* V^ψ *(with the notation of Remark 4.3.6) is the solution of*

$$dV_t^\psi = r_t V_t^\psi\, dt + \psi_t(dS_t - r_t S_t\, dt), \quad V_0^\psi = x, \tag{4.6}$$

then the process $\hat{\psi}_t := (V_t^\psi - \langle \psi_t, S_t\rangle, \psi_t) \in \mathbb{R}^{d+1}$ *for* $t \in [0,T]$ *is a self-financing strategy and* $V_t^\psi = V_t(\hat{\psi})$.

Proof. First of all, note that the discount factor satisfies an ODE of the type $dD_t = -r_t D_t\, dt$. Then, note that the first implication has already been obtained in Remark 4.3.2, in particular this is equivalent to Equation (4.5).

For the second implication, we note that the value of the discounted portfolio implied by (4.6) is

$$\widetilde{V}_t^\psi = x + \int_0^t \psi_u\, d\widetilde{S}_u,$$

while the fraction of wealth invested in the risk-less asset is

$$\eta_t B_t = V_t^\psi - \sum_{i=1}^d \psi_t^i S_t^i.$$

At this point, we can verify condition B) of Definition 4.3.1:

$$\begin{aligned} dV_t &= r_t V_t dt + \psi_t(dS_t - r_t S_t dt) = r_t\left(V_t - \psi_t S_t\right) dt + \psi_t dS_t \\ &= r_t \eta_t B_t\, dt + \psi_t\, dS_t = \eta_t\, dB_t + \psi_t\, dS_t. \end{aligned} \tag{4.7}$$

□

Given this brief introduction, we can finally define what we mean by hedging in financial markets. *Hedging* is the practice of taking a position in one market or investment to offset, balance and protect against the risk adopted by assuming a position in a contrary or opposing market or investment.

Problem 4.3.8 (Hedging)**.** *In mathematical terms, given* $H \in L^0(\mathcal{F}_T)$ *a random measurable payoff at time* T*, can we define a self-financing strategy* (V_0, ψ) *such that* $V_T(\varphi) = H$ $\mathbb{P}$*-almost surely? What is the minimal amount of initial endowment for such trading strategy?*

If there exists such a self-financing trading strategy, we will call it a *replication strategy.*

Let us now define two important process:

Definition 4.3.9 (Cumulative gains G and costs C)**.** *The* cumulative gains/losses *process* G *from a strategy* ψ *is obtained by setting* $V_0 \equiv 0$ *in the definition for the value of a self-financed portfolio* V*: for all* $t \in [0,T]$

$$G_t(\psi) := \int_0^t \psi_u\, d\widetilde{S}_u = 0 + \int_0^t \psi_u\, d\widetilde{S}_u. \tag{4.8}$$

Similarly, we can define the process of cumulative costs *or* cost of trading *C from strategy φ for $0 \leq t \leq T$ as*

$$C_t(\varphi) := V_t(\varphi) - \int_0^t \psi_u \, d\widetilde{S}_u. \tag{4.9}$$

It represents the total cost for trading on $[0, t]$. Note that these costs arise from trading because of the fluctuations in the price process $\widetilde{S}$ and are not due to transaction costs.

Since allowing for self-financing strategy only might lead to arbitrage, we further restrict the set of *admissible strategies* by requiring that $G(\psi)$ must be larger than a certain quantity.

Definition 4.3.10 (Admissible strategy)**.** *Any self-financing strategy for which $G(\psi)$ is uniformly bounded from below, i.e. $G_t(\psi) \geq -a$ $\mathbb{P}$-almost surely for $a \geq 0$ for $0 \leq t \leq T$, is said to be* admissible*.*

Admissibility is needed to avoid the so-called *doubling strategies* or *money-pumps*. Let us consider, for example, an exponential Brownian motion $S = \exp(W_t - t/2)$ on the infinite horizon ($T = +\infty$), with the understanding that $S_{+\infty} = 0$. If we go short at time 0, that is we choose $\psi = -1$, the value of our final portfolio would be $V_{+\infty} = -(S_{+\infty} - S_0) = 1$, which would cause arbitrage. Note that we did not use money at initial time. The name *doubling strategy* derives from the martingale betting strategy system, in which the gambler wins the stake if a coin comes up heads and loses when tails shows up. The strategy had the gambler double the bet after every loss, so that the first win would recover all previous losses plus win a profit equal to the original stake. Since any gambler will almost surely eventually flip heads, the martingale betting strategy is certain to make money for the gambler provided they have infinite wealth (and there is no limit on money earned in a single bet). However, no one has infinite wealth.

In view of Definition 4.3.10, we can say that our goal is to find admissible replication strategies.

Remark 4.3.11. Note that a in Definition 4.3.10 does not depend on the particular scenario $\omega \in \Omega$, but may vary according to ψ.
In the following, when talking about admissible strategies, we will always interpret it as a strategy having the cumulative gain process bounded from below.

Remark 4.3.12. So far we have implicitly made many simplifications (or formulated assumptions):

- The risk-less asset represented by the bank account has always positive value. However, in reality, risk-less assets do not exist.
- We have not explicitly distinguished between discrete or continuous time setting. In particular, in our framework, it is possible to trade continuously in time.
- Prices for buying or selling the risky asset S is given by S alone and there are no trading fees or other types of transaction costs (*frictionless market*).
- There are no constraints on strategies (see also Remark 4.3.5).
- Investors are thought to be "small" investors, that is market prices are not influenced by the investor strategy.

4.3.1 Delta hedging for Black Scholes

In financial markets, people are familiar with a particular type of hedging, called *delta-hedging*. Specifically, delta-hedging mitigates the financial risk of an option by hedging against price

changes in its underlying. It is called in this way because Delta is the first derivative of the option's value with respect to the underlying asset price. This is performed by buying an asset with an inverse price movement. For instance, in order to delta-hedge a vanilla option derivative, the trader should buy the underlying onto which the derivative is written.

To see how delta-hedging originates, we can have a look at the renowned Black-Scholes formula, which granted jointly to Robert C. Merton and Myron S. Scholes a Nobel prize for economics in 1997[6] ([BS73]) "for a new method to determine the value of derivatives". Let us consider the price of a European call option C with maturity T and strike price K at time $t \in [0,T]$

$$C_t(T,K) = S_t e^{-q(T-t)}\Phi(d_+) - e^{-r(T-t)}K\Phi(d_-), \tag{4.10}$$

with

$$d_\pm = \frac{\log\left(\frac{S_t}{K}\right) + \left(r - q \pm \frac{1}{2}\sigma^2\right)(T-t)}{\sigma\sqrt{T-t}}, \tag{4.11}$$

where r is the constant continuous risk-free rate, q is the constant continuous dividend yield, $\Phi(\cdot)$ is the cdf of a standard normal random variable and $T-t$ is the time to maturity. Actually, from Equation (4.10) we see that we already have a replication strategy since the value of the call option is expressed as a linear combination of units of risky asset, the stock S_t, and units of risk-less asset, the money market account B_t. We may in fact set $\eta_t = -e^{-rT}K\Phi(d_-)$ and $\psi_t = e^{-q(T-t)}\Phi(d_+)$ and, hence, rewrite (4.10) as

$$C_t(T,K) = \eta_t e^{rt} + \psi_t S_t = \eta_t B_t + \psi_t S_t.$$

At this point, it is enough to note that the first derivative of the call option under the Black-Scholes setting has a closed-form formula to clarify the name of *delta-hedging*. In this framework, we have indeed

$$\delta_t := \frac{\partial\, C_t(T,K)}{\partial S} = e^{-q(T-t)}\Phi(d_+) \tag{4.12}$$

and thus $C_t(T,K) = \eta_t B_t + \delta_t S_t$, since $\delta_t = \psi_t$ for all $t \in [0,T]$.
We are now left with showing that this trading strategy is also admissible. With this goal in mind, we can proceed as follows. First of all, we recall this version of the Itô representation theorem:

Theorem 4.3.13 (Itô representation theorem)**.** *Any random variable $H \in L^2(\mathcal{F}_T^W, \mathbb{Q})$ admits a unique representation*

$$H = \mathbb{E}^{\mathbb{Q}}[H] + \int_0^T \xi_u\, dW_u \quad \mathbb{Q}\text{-a.s},$$

where ξ is a process in $L^2(W)$, in particular $\int_0^t \xi_u\, dW_u$ is a martingale for any $t \in [0,T]$.

Second, that under the Black-Scholes model the underlying has dynamics given by a geometric Brownian motion: $dS_t = \mu S_t\, dt + \sigma S_t\, d\widetilde{W}_t$, where $\mu \in \mathbb{R}$ is the drift, $\sigma > 0$ the volatility coefficient and $(\widetilde{W}_t)_{0\le t\le T}$ a Brownian motion. After a change of measure and employing an equivalent martingale measure $\mathbb{Q}$, the discounted dynamics become $dS_t = \sigma S_t dW_t$, with W Brownian motion under $\mathbb{Q}$. Actually, in this case, we can even write the Radon-Nikodým derivative explicitly: for $t \in [0,T]$

$$L_t = \left.\frac{d\mathbb{Q}}{d\mathbb{P}}\right|_{\mathcal{F}_t} = \exp\left(-\alpha\widetilde{W}_t - \frac{1}{2}\alpha^2 t\right),$$

where $-\alpha := \frac{r-q-\mu}{\sigma}$ and α is usually called *risk-premium*. In this case, $(S_t)_{0\le t\le T}$ is a square integrable martingale under the measure $\mathbb{Q}$ and the same holds true for discounted call prices

[6] Unfortunately, Fischer Black died in his mid-fifties in 1995.

(otherwise there could arbitrage opportunities). If we set the filtration of the stochastic space as the natural filtration generated by the Brownian motion W, i.e. $\mathbb{F} = \mathbb{F}^W$ (augmented in the usual way), we can use the Itô representation Theorem 4.3.13 for $H = (S_T - Ke^{-rT})_+$ (which lies in L^2) and write

$$H = \mathbb{E}^{\mathbb{Q}}[H] + \int_0^T \xi_u \, dW_u = \mathbb{E}^{\mathbb{Q}}[H] + \int_0^T \psi_u \, dS_u,$$

where we can set $\psi_t \equiv \frac{\xi_t}{\sigma S_t}$. Moreover, since we have $H \geq 0$ (by definition), we also have for any $t \in [0, T]$

$$G_t(\psi) = \int_0^t \psi_u \, dS_u \geq -\mathbb{E}^{\mathbb{Q}}[H],$$

and $\mathbb{E}^{\mathbb{Q}}[H]$ is finite (since for bounded domains we have $L^2 \subset L^1$). Thus, the tuple $(\mathbb{E}^{\mathbb{Q}}[H], \psi)$ defines an admissible strategy (see Definition 4.3.10).

4.3.2 (In)complete markets

To talk about incomplete markets, it is advisable to first speak about complete markets. Roughly speaking, a market is *complete* if any derivative product can be perfectly hedged, i.e. it is the terminal value of a self-financing portfolio.

Definition 4.3.14 (Contingent claim)**.** *A* contingent claim H *is defined as a square integrable F_T-random variable, where $T \in (0, +\infty)$ is a fixed time horizon. We can also write $H \in L^2(\mathcal{F}_T)$.*

Definition 4.3.15 (Attainable claim)**.** *A contingent claim H is said to be* attainable *or* hedgeable *if there exists a predictable process $\varphi = (\varphi_0, \ldots, \varphi_d)$ such that $V_T(\varphi) = H$. The self-financing strategy $\hat{\psi} = (V^{\psi} - \langle \psi, S \rangle, \psi)$ is called the* replicating strategy *(or the* hedging strategy*) of H, and $V_0^{\psi} = h$ is the initial price (see also Remark 4.3.6).*

Remark 4.3.16. The process V^{ψ} is the price process of H.

In some sense, this initial value is an equilibrium price: the seller of the claim H, for instance a bank, agrees to sell the claim at an initial price h if she can construct a portfolio with initial value h and terminal value greater than the claim she has to deliver. The buyer of the claim, for example a private investor, agrees to buy the claim if he is unable to produce the same (or a greater) amount of money while investing h in the financial market.

Definition 4.3.17 (Complete market)**.** *Assume that r is deterministic*[7] *and let $\mathbb{F}^S$ be the natural filtration generated by the underlying price S. The market is said to be* complete *if any contingent claim $H \in L^2(\mathcal{F}_T^S)$ is the value at time T of some self-financing strategy $\hat{\psi}$.*

Remark 4.3.18. For what we saw before in Section 4.3.1, the Black-Scholes model $(S, \mathbb{F} = \mathbb{F}^W)$ can be seen as an example of complete market, since it is arbitrage free and for every contingent claim H we have $H = \mathbb{E}^{\mathbb{Q}}[H] + \int_0^T \psi_u \, dS_u = V(\varphi)$ where φ is an admissible self-financing strategy identified by $(V_0 = \mathbb{E}^{\mathbb{Q}}[H], \psi)$, with S discounted asset price.

In order to guarantee market completeness for liquid and frictionless markets with an arbitrary (finite) number of assets, the so-called second fundamental theorem of asset pricing was introduced.

[7] If r is stochastic, it is possible to work with with the filtration generated by the discounted S, i.e. $(\exp(-\int_0^t r_s \, ds) S_t)_{0 \leq t \leq T}$.

This was first formulated in 1979 by Harrison and Kreps [HK79], but many others worked on it (see, for example, [Bia10] and references therein). The theorem provides necessary and sufficient conditions for a financial market to be complete, and it was firstly studied to address the issue of knowing which contingent claims are spanned by a given set of market securities.

Remark 4.3.19 (**A nice crossover**). We can actually provide a (partial) answer to this problem by taking advantage of the Theorems in Section 2.4, in particular Theorem 2.4.6 and Proposition 2.4.8, which together establish that shallow ReLU neural networks are dense in space of continuous functions (defined on compact sets). If we transpose this claim in the financial setting, we can state that linear combinations of European call options can approximate any contingent claim with continuous payoff (on a compact set).

Theorem 4.3.20 (Second Fundamental Theorem of Asset Pricing)**.** *Let S represent the discounted prices process and assume that the market is arbitrage-free (hence, S is a local martingale). Then there exists a unique equivalent local martingale measure (ELMM) $\mathbb{Q}$ if and only if the market is complete.*

If we denote with $\mathcal{M}^e(S)$ the set of equivalent martingale measures with respect to the numéraire $S^0 = B$, and we assume that $\mathcal{M}^e(S) \neq \varnothing$, then Theorem 4.3.20 states that the market is complete if and only if $\mathcal{M}^e(S) = \{\mathbb{Q}\}$, i.e. $\mathcal{M}^e(S)$ is a singleton. If the market is incomplete (but arbitrage-free), then there are more equivalent (local) martingale measures.

Remark 4.3.21. Note that there can be a complete market with no equivalent martingale measures, but still complete market. This can be the case, for example, if we have a risk-less asset and two geometric Brownian motions with different drifts, but same diffusion coefficient and same driving Brownian motion. In this case, we can build arbitrage opportunities (so there are no ELMMs), but the market is complete because any contingent claim can be replicated as a function of a discounted risky asset (see Comment 2.1.5.3 in [JYC12]).

Remark 4.3.22 (Market completeness for Itô diffusions). Let $(\Omega, \mathcal{F}, \mathbb{P})$ be a probability space. We assume that an n-dimensional Brownian motion W is constructed on this space and we denote by $\mathbb{F}^W$ its (augmented) natural filtration. We assume that in the market there is one risk-less asset, the money market account, which evolves according to a deterministic continuous interest rate r, i.e. $B_t = S^0_t = \exp(\int_0^t r_s\, ds)$, and d risky assets, which we model with a d-dimensional Itô diffusion process:

$$dS^i_t = S^i_t \left(b^i_t\, dt + \sum_{j=1}^{n} \sigma^{ij}_t\, dW^j_t \right), \qquad \text{for } i = 1, \dots, d.$$

We assume that all process r, b^i and σ^{ij} are predictable and satisfy integrability conditions of the type $\int_0^t r_s\, ds < +\infty$, with $r_t > 0$, $\int_0^t |b^i_t|\, dt < +\infty$ a.s. and $\int_0^t (\sigma^{ij}_s)^2\, ds < +\infty$ a.s. for all $t \geq 0$ and all $i = 1, \dots, d$ and $j = 1, \dots, n$. Note that the prices of *all* assets are positive. If we denote with $[\Sigma_t(\omega)]_{ij}$ the element $\sigma^{ij}_t(\omega)$, then we have that the market is complete if and only if the rank of the matrix Σ_t is equal to d for almost all $t \geq 0$ a.s. Moreover, if $d < n$, the market is not complete but arbitrage-free, while if $n < d$ we fall back in the case of Remark 4.3.21 and arbitrage is possible (see Theorem 1.7 in [Bia10] and Section 2.2 in [JYC12]). The standard Black-Scholes framework, for $n = d = 1$, is special case of this broader setting where the market is complete.

Remark 4.3.23. A financial market can be incomplete for different reasons. We already saw in Remark 4.3.22 that one of these could be the fact that the sources of randomness outnumber the financial securities. This is, for example, also the case for stochastic volatility models, such as the celebrated Heston model [Hes93], where the modelled financial assets are a stock with stochastic volatility and a bond with constant interest rate. In this setting, the market is incomplete because

it is not possible to hedge the risk factor associated with stochastic volatility[8]. Another possibility is given by market friction. Continuous-time portfolio strategies accrue transaction costs at every instant the portfolio is rebalanced and, thus, these strategies are effectively forbidden if their costs are infinite. Good articles on this topic are [HN89] and [SSC95]. Further, incompleteness can also be the result of asymmetric information, as shown in [FS91]. Here the hypothesis is that contingent claim H is attainable with respect to a larger filtration than that to which the stock price process S is actually adapted to.

4.3.3 Hedging in incomplete markets

In (arbitrage-free) incomplete markets the basic question turns out to be: how can we obtain a hedging strategy for non-attainable claims? The question is of utter importance since the value of the replication portfolio is then, by arbitrage arguments, the price of the claim at any instant of time t between now and maturity $T < +\infty$. In this case, the concept of replication falls apart not because continuous time trading is actually impossible in reality, but because there are risks that one cannot hedge even by continuous time trading. Let us now consider discounted quantities, so that, for example, S will denote the discounted price process of the underlying. In the following, we denote with $\mathcal{M}_{\mathbb{P}}(S)$ the convex set of equivalent (local) martingale measure with respect to the historical measure $\mathbb{P}$ for the discounted price process S. We additionally assume $\mathcal{M}_{\mathbb{P}}(S) \neq \varnothing$. This assumption is equivalent to the no arbitrage condition 'No Free Lunch with Vanishing Risk' (NFLVR) introduced by Delbaen and Schachermayer [DS94] for locally bounded semimartingales.

Definition 4.3.24 (Viable prices)**.** *The set* $\{E^{\mathbb{Q}}[e^{-\int_0^T r_s\, ds} H] : \mathbb{Q} \in \mathcal{M}_{\mathbb{P}}(S)\}$ *is called the set of* viable prices*.*

If the asset H is traded at a price that is included in the viable prices' set, then we do not have arbitrage opportunities. Therefore, in an incomplete market, for example due to frictions, an agent has to specify an optimality criterion which defines an acceptable 'minimal price' for any position. The intuition behind the criterion is that the hedging strategy should be able to replicate the contingent claim as best as possible.

In the following, we are going to talk about some of the most common ways to hedge in incomplete markets:

- **Super-replication**
  If we fix a payoff H, the idea of super-replication is to find all strategies that can produce at least H, and then select the cheapest one. Let us take $H \in L^0_{\geq 0}(\mathcal{F}_T)$, non-negative measurable with respect to $\mathcal{F}_T$, then we can define the super-replication price as

$$\Pi^{\text{super}}(H) := \inf\left\{ h \in \mathbb{R} : \exists\, \psi \text{ admissible s.t. } h + \int_0^T \psi_t\, dS_t \geq H \ \mathbb{P}-\text{a.s.} \right\}. \tag{4.13}$$

  The interpretation is that $\Pi^{\text{super}}(H)$ will then be the price of the portfolio at time 0, i.e. $\Pi^{\text{super}}(H) = V_0^{\psi}$. Selling the claim H for this prices involves no risks because the strategy $(\Pi^{\text{super}}(H), \psi)$ is a self-financing admissible strategy which produces at least H by time T. In principle, there could be a problem though: since the initial price is defined as

[8] In this case, it is actually possible to "complete" the market by adding, for instance, a derivative on the variance of the underlying asset (more on this topic in [RT97] and references therein). But if we consider non-trivial jump processes, this is not possible, since the drift coefficient of the discounted risky asset that is set to zero when using Girsanov's theorem can be usually nullified by an infinite number of parameter combinations.

an infimum of a set, it is not clear a priori if this can actually be reached. A possible proof, that makes use of Ansel-Stricker Lemma [AS94; DP07] and of optional decomposition, also called Kramkov decomposition ([Kra96]), can be found in [Tei13] and shows that the price is actually attained as a minimum in case $\Pi^{\text{super}}(H) < +\infty$. Moreover, it is shown that it coincides with $\sup_{\mathbb{Q}\in\mathcal{M}_{\mathbb{P}}(S)} \mathbb{E}^{\mathbb{Q}}[H]$ ([EQ95]), so that we have a dual representation.

This approach is maybe the most natural when thinking of a possible hedging strategy for non-attainable claims, but is rather extreme: the seller has no risk from this sale because all risk is now on the buyer side. For this reason, the super-replication price is also called *seller price*. On the other hand, the *buyer price* is defined as $\inf_{\mathbb{Q}\in\mathcal{M}_{\mathbb{P}}(S)} \mathbb{E}^{\mathbb{Q}}[H]$. The interval given by $(\Pi_{\text{buy}}(H), \Pi_{\text{sell}}(H)) = (\inf_{\mathbb{Q}\in\mathcal{M}_{\mathbb{P}}(S)} \mathbb{E}^{\mathbb{Q}}[H], \sup_{\mathbb{Q}\in\mathcal{M}_{\mathbb{P}}(S)} \mathbb{E}^{\mathbb{Q}}[H])$ defines a *no-arbitrage* price interval which coincides with the set of viable prices (Definition 4.3.24.

Remark 4.3.25. Super-hedging strategies usually provide forbiddingly high prices for the seller price, as seen in [EJ97] for a purely discontinuous (jump) price process, or in [BJ00] in case of jump-diffusion price process. Their conclusion for European options is that the interval of viable prices is bounded above by the super-replication price and that for a call option the super-hedging strategy consists in buying the underlying at time 0 and holding it until maturity. But this solution is most cases too expensive and, thus, unrealistic.

- **Quadratic Hedging**
  For a 'guided tour' in quadratic hedging, there exists a distinguished technical report by Schweizer [Sch99b] which we undoubtedly suggest. We will mainly quickly revise the two most important approaches developed in this field.

  ▷ Local Risk Minimization
  The first attempt to hedge a non-attainable claim in incomplete markets is actually due to Föllmer and Sondermann in 1986 ([FS86]), where they considered the case of S being a $\mathbb{P}$-martingale and introduced the concept of *risk-minimizing strategies*, which minimize the risk in a sequential sense. To proceed, we assume that the cost process C is square-integrable and let us define the *risk process* associated to a strategy φ as

  $$R_t(\varphi) := \mathbb{E}^{\mathbb{P}}\left[\left(C_T(\varphi) - C_t(\varphi)\right)^2 \mid \mathcal{F}_t\right], \quad 0 \le t \le T. \tag{4.14}$$

  Clearly, the absolute minimum of $R(\varphi)$ is the zero process and the goal becomes minimizing $R(\varphi)$ in a suitable way. This can be attained if and only if there is a self-financing strategy φ with $V_T(\varphi) = H$. In this sense, attainable claims corresponds to zero-risk. Since we are in an incomplete market settings, we can decide to minimize (4.14) anyway among those strategies for which $V_T(\varphi) = H$, at the cost of giving up on self-financial strategies. This has the (unpleasant) consequence that the portfolio might need injection or withdrawal of wealth in time. This approach is known as *local risk-minimization*. A strategy $\varphi = (\eta, \psi)$ is called *risk-minimizing* (RM) if at any instant of time it minimizes the risk process, i.e. for $0 \le t \le T$ we have $R_t(\varphi) \le R_t(\widetilde{\varphi})$ where $\widetilde{\varphi}$ is an admissible continuation of φ from t on, that is $V_T(\varphi) = V_T(\widetilde{\varphi})$ $\mathbb{P}$-a.s. and $\widetilde{\psi}_s = \psi_s$ for $s \le t$ and $\widetilde{\eta}_s = \eta_s$ for $s < t$. Although RM-strategies with $V_T(\varphi) = H$ will in general not be self-financing, it turns out that any RM-strategies is "self-financing on average" in the sense that the cost process defined in (4.9) is a $\mathbb{P}$-martingale (note that self-financing strategies imply that the cost process $C_t(\varphi)$ is a constant $\mathbb{P}$-a.s.).
  The key result for finding risk-minimizing RM-strategies is the Galtchouk-Kunita-Watanabe decomposition (Proposition 4.14 in Section 3.4 of [KS98] or [KW67] for the original paper). Since the space $\mathcal{I}^2(S) := \{\int \psi \, dS \,:\, \psi \in L^2(S)\}$ of stochastic integrals

is a closed subspace of $\mathcal{M}_0^2(\mathbb{P})$, the space of square integrable $\mathbb{P}$-martingales null at time 0, any $H \in L^2(\mathcal{F}_T, \mathbb{P})$ can be uniquely written as

$$H = \mathbb{E}^{\mathbb{P}}[H] + \int_0^T \psi_u^H \, dS_u + L_T^H, \quad \mathbb{P}\text{-a.s.},$$

for $\psi^H \in L^2(S)$ and some $L^H \in \mathcal{M}_0^2(\mathbb{P})$ for which $L^H I$ is a $\mathbb{P}$-martingale for all $I \in \mathcal{I}^2(S)$. This actually allows to write down the optimal strategy $\varphi^\star$ such that $V_T(\varphi^\star) = H$ $\mathbb{P}$-a.s. as $\psi_t^\star = \psi_t^H$, $V_t^\star := V_t(\varphi^\star) = \mathbb{E}^{\mathbb{P}}[H \,|\, \mathcal{F}_t]$ and $C_t(\varphi^\star) = \mathbb{E}^{\mathbb{P}}[H] + L_t^H$ for $0 \le t \le T$. In this case, we have $R_t(\varphi^\star) \le R_t(\varphi)$ for any other admissible RM-strategy φ (Theorem 2.4 in [Sch99b]) for $0 \le t \le T$.
The generalization to the semimartingale setting was achieved a few years later by Föllmer and Schweizer ([FS91]) with strategies taking the name of *locally* risk minimizing strategies. For an extension that takes into account transaction costs, see Lamberton, Pham and Schweizer [LPS98].

▷ Mean-Variance Hedging
A different type of quadratic hedging has been proposed by Gouriéroux, Laurent and Pham in 1998 ([GLP98]) with the name of *mean-variance hedging*. Useful comparisons between the two approaches is provided in [Sch99b; HPS01] (see also [Sch92] and the more recent publication [Sch10]). The key difference between (locally) risk minimization and mean-variance hedging is that we no longer impose the strict requirement $V_T(\varphi) = H$, but rather require that the self-financing strategy constraint for (V_0, ψ). Another essential difference is that, as we will see, in this case we are rather focusing on the global risk over the entire period $[0, T]$ altogether minimized at time 0, rather then minimizing the risk for every instant of time. The problem they considered concerned $L^2(\mathbb{P})$-minimization of the quadratic error (from which the name *quadratic hedging*) under the $\mathbb{P}$-historical probability measure:

$$\min_{h,\psi} \mathbb{E}^{\mathbb{P}} \left[\left(H - V_T^{h,\psi} \right)^2 \right], \tag{4.15}$$

where $V_T^{h,\psi}$ denotes the portfolio value at time T with initial value $h \in \mathbb{R}$ and ψ self-financing admissible strategy. The solution is the L^2-projection of H on the vector space $h + \int_0^T \psi_u \, dS_u$, with S continuous semimartingale. Let us define with $\mathcal{M}_{\mathbb{P}}^2(S) := \left\{ \mathbb{Q} \in \mathcal{M}_{\mathbb{P}}(S) \,:\, \frac{d\mathbb{Q}}{d\mathbb{P}} \in L^2(\mathbb{P}) \right\} \subseteq \mathcal{M}_{\mathbb{P}}(S)$ the set of all ELMMs with square-integrable density. As for $\mathcal{M}_{\mathbb{P}}(S)$, we assume that $\mathcal{M}_{\mathbb{P}}^2(S) \neq \varnothing$. We denote with $\Psi := \{\psi \in L(S) \,:\, \int_0^T \psi_u \, dS_u \in L^2(\mathbb{P}) \text{ and } \int \psi \, dS_u \in \mathcal{M}(\mathbb{Q}) \; \forall \, \mathbb{Q} \in \mathcal{M}_{\mathbb{P}}^2(S)\}$, where $\mathcal{M}(\mathbb{Q})$ denotes the set of $\mathbb{Q}$-martingales. A *mean-variance* (MV) strategy is a tuple (V_0, ψ) with $V_0 \in \mathbb{R}$ and $\psi \in \Psi$. It is said to be *optimal* if it minimizes the $L^2(\mathbb{P})$-norm difference of (4.15) over all possible MV-strategies. To proceed, we also need the definition of *variance-optimal ELMM* $\widetilde{\mathbb{Q}}$, which is the unique element of $\mathcal{M}_{\mathbb{P}}^2$ that minimizes the Radon-Nikodým derivative:

$$\widetilde{\mathbb{Q}} := \min_{\mathbb{Q} \in \mathcal{M}_{\mathbb{P}}^2} \left\| \frac{d\mathbb{Q}}{d\mathbb{P}} \right\|_{L(\mathbb{P})} = \sqrt{1 + \mathrm{Var}_{\mathbb{P}} \left(\frac{d\mathbb{Q}}{d\mathbb{P}} \right)}.$$

Moreover, [GLP98] shows that for all $0 \le t \le T$ we have

$$\widetilde{Z}_t := \mathbb{E}^{\widetilde{\mathbb{Q}}} \left[\frac{d\widetilde{\mathbb{Q}}}{d\mathbb{P}} \,\middle|\, \mathcal{F}_t \right] = \widetilde{Z}_0 + \int_0^t \widetilde{\zeta}_u \, dS_u,$$

for $\widetilde{\zeta} \in \Psi$. Further, $\widetilde{Z}$ is continuous. At this point, we can again resort to the Galtchouk–Kunita–Watanabe decomposition of H under $\widetilde{\mathbb{Q}}$ with respect to S and define the optimal MV-strategy for the portfolio $V_t^{H,\widetilde{\mathbb{Q}}} := \mathbb{E}^{\widetilde{\mathbb{Q}}}[H \mid \mathcal{F}_t] = \mathbb{E}^{\widetilde{\mathbb{Q}}}[H] + \int_0^t \xi_u^{H,\widetilde{\mathbb{Q}}}\, dS_u + L_t^{H,\widetilde{\mathbb{Q}}}$ for $t \in [0,T]$ and set $V_0^\star := \mathbb{E}^{\widetilde{\mathbb{Q}}}[H]$ and $\psi_t^\star := \xi_t^{H,\widetilde{\mathbb{Q}}} - \widetilde{\zeta}_t \int_0^{t-} 1/\widetilde{Z}_u\, dL_u^{H,\widetilde{\mathbb{Q}}}$ for $0 \le t \le T$.

Remark 4.3.26. One of the criticism of quadratic hedging is that profits and losses are both equally punished when increasing the squared difference in (4.15).

Remark 4.3.27. Actually, mean-variance hedging can be seen as part of a broader class of methods for pricing in incomplete markets. Since in this condition we have at our disposal many equivalent local martingale measures, we just need to select one based on a theoretically grounded criterion, such as the maximization of a utility function. Other examples are [Fri00], where the concept of *minimal entropy* martingale measure is introduced, or again [BF02] for *minimax* measures, just to name but a few.

- **Hedging through convex functions**
  The dual characterization of the super-replication price paves the way to an extension of the convex duality approach to study optimization problems with convex risk measures and, hence, utility functions. Note that also mean-variance hedging can be seen as a special case of hedging through convex function, where the convex function f is simply quadratic, i.e. $f(x) = x^2$. Similarly, the minimal entropy martingale measure of [Fri00] is associated to the convex function $f(x) = x \ln x$, for $x > 0$. For its generality, this approach is thus of great relevance in research and in practice. Standard references, in this case, are the articles by Xu [Xu06], Klöppel and Schweizer [KS07] and book by Föllmer and Schied [FS16] (in discrete times).

  Probably, the first instance of utility functions used for incomplete market pricing and hedging was put forward by Hodges and Neuberger in 1989 [HN89] employing an exponential utility function. If we consider a trader with a specified preference structure described by a (concave) utility function, then the goal is matching the maximal expected utilities that can be gained with and without a particular claim. More precisely, let us consider a utility function U that is a strictly increasing and strictly concave function, that may satisfy some assumptions on its regularity, such as being continuously differentiable, and other asymptotic conditions (e.g. limit Inada conditions; see [KS99] for more details). Then the price of a contingent claim H is defined as the infimum over $h \in \mathbb{R}$ such that

$$\sup_\psi \mathbb{E}\left[U\left(V_T^{x+h,\psi} - H\right)\right] \ge \sup_\psi \mathbb{E}\left[U\left(V_T^{x,\psi}\right)\right], \tag{4.16}$$

  where the supremum is taken over all possible admissible strategies ψ (this definition was already proposed in [HN89]). The agent selling the contingent claim starts with an initial endowment $x + h$ and, using the strategy ψ, he obtains a portfolio with terminal value $V_T^{x+h,\psi}$. Since he he has to deliver the T contingent claim H, its final wealth is $V_T^{x+h,\psi} - H$. Therefore, we require that its expected utility obtained by selling the claim H and hedging (through investment of the additional h) is at least as large as the expected utility when she does not sell the claim. For this reason, the method is often referred to as *indifference pricing*. This method has been studied extensively: we remind to [RE00] for the case of exponential utility of the type $x \mapsto \alpha \exp(-\gamma x)$, with α is only a multiplicative constant (and is chosen equal to 1), while γ is a constant that measures thee degree of *risk-aversion* of the trader whose utility we are considering. Other example are [HK04] and [İS06].

Remark 4.3.28. Despite being appealing from an economic point of view, this methodology has the inherent problematic of calibration for the utility function. For example, how should we estimate the risk-aversion parameter γ in an exponential utility function? For discussion on the topic, we refer to [Rab00].

More recently, other types of utilities have been taken into account. Namely, utility derived from coherent or convex risk measures (note that if ρ is a coherent risk measure, then $U := -\rho$ is a coherent utility function; see [CDK05] for a formal definition). A *coherent* risk measure, as introduced in the seminal paper by Artzner and coauthors [Art+99], is a function $\rho : L^\infty \to \mathbb{R}$ such that for any X, Y random variables

1. $\rho(X + Y) \leq \rho(X) + \rho(Y)$: subadditivity, to express the concept that diversifying reduces the risk;
2. if $\lambda \geq 0$, then $\rho(\lambda X) = \lambda\rho(X)$: positive homogeneity, if we scale the same position, the risk will scale accordingly;
3. if $X \geq Y$, then $\rho(X) \leq \rho(Y)$: monotonicity, i.e. a less risky position requires less cash injection;
4. if $m \in \mathbb{R}$, then $\rho(X + m) = \rho(X) - m$: translation invariance, i.e. adding cash to a position reduces the need for more by the same amount. In particular, $\rho(X+\rho(X)) = 0$, which means that $\rho(X)$ is the least amount of cash we need to add to X to make the position acceptable, in the sense $\rho(X) \leq 0$. This condition is also the crucial condition for *monetary* risk measures (as introduced by Föllmer and Schied [FS02]).

Convex risk measure [FS02] replace the first two points above with

5. $\rho(\lambda X + (1 - \lambda)Y) \leq \lambda\rho(X) + (1 - \lambda)\rho(Y)$, for $\lambda \in [0, 1]$: convexity.

In this framework, let us suppose that the trader starts with a liability $-L$ (bounded below from a constant) that is $\mathcal{F}_T$-measurable and with an initial amount of money equal to $x_0 \in \mathbb{R}$. All quantities are again considered discounted. Using convex risk measure, we can define the *minimal risk* ([Xu06]) as the risk associated to the optimal hedge

$$\rho^{x_0}(-L) := \inf_{\psi} \rho\left(-L + V_T^{x_0,\psi}\right). \tag{4.17}$$

for ψ admissible strategy (see Remark 4.3.11). Such a minimal price is going to be the minimal amount of cash the agent needs to add to her position to implement the optimal hedge and such that the overall position becomes acceptable.
Analogously, she will buy H with the maximal amount which will allow to keep the risk of her portfolio stable. Having defined the minimal risk $\rho^{x_0}(-L)$ for a liability $-L$ with initial capital x_0, we can thus introduce the seller price and the buyer price as follows using the indifference philosophy previously illustrated. For the seller, the goal is again that of finding a price for a contingent claim H, such as a derivative option, by accepting to charge the minimal amount provided that the total risk of her portfolio (after re-balancing the hedging) will not increase from selling the derivative. Analogously, for the buyer: the trader is willing to pay the maximal amount as long as the total risk of the portfolio remains stable.

$$\begin{aligned} \Pi_{\text{sell}}(H) &:= \inf\left\{x \in \mathbb{R} : \rho^{x_0+x}(-L - H) \leq \rho^{x_0}(-L)\right\}, \\ \Pi_{\text{buy}}(H) &:= \sup\left\{x \in \mathbb{R} : \rho^{x_0-x}(-L + H) \leq \rho^{x_0}(-L)\right\}. \end{aligned} \tag{4.18}$$

Under usual assumptions on ρ, e.g the Fatou property[9] and $\rho^0(0) > -\infty$ (see [Xu06] for a discussion on those), we can show that the buyer price is always bounded above by the seller price, i.e. $\Pi_{\text{buy}} \leq \Pi_{\text{sell}}$, that these are also included in the interval defined by the sub- and super-replication strategies and are, thus, arbitrage-free. It is also possible to prove that, in case of complete market, the buyer and seller price coincide with the unique no arbitrage price, as one would expect.
Using convex functions for pricing and hedging brings also a couple of advantages:

i) The definition of risk measure prices reduce to that of utility based prices if we decide that the risk measure is an expected utility. For example, by choosing $\rho(-L) = -\mathbb{E}[U(-L)]$ for a utility function U, the minimal risk becomes

$$\rho^{x_0}(-L) = \inf_{\psi} \rho\left(-L + V_T^{x,\psi}\right) = -\sup_{\psi} \mathbb{E}\left[U(V_T^{x,\psi} - L)\right],$$

while the seller price becomes

$$\begin{aligned}\Pi_{\text{sell}}(H) &= \inf\left\{x \in \mathbb{R} \,:\, \rho^{x_0+x}(-L-H) \leq \rho^{x_0}(-L)\right\} \\ &= \inf\left\{x \in \mathbb{R} \,:\, \sup_{\psi} \mathbb{E}\left[U\left(V_T^{x_0+x,\psi} - H\right)\right] \geq \sup_{\psi} \mathbb{E}\left[U\left(V_T^{x_0,\psi}\right)\right]\right\},\end{aligned}$$

and this actually coincides with the definition we gave in (4.16). Note, however, that risk measures defined throughout utility functions are not translation invariant.

ii) In complete markets, the seller of the claim H is entitled the sum $\mathbb{E}[H]$ at time 0 and with this she is able to set up an admissible self-financing strategy that will exactly cover the final payoff. Thus, the seller ends up with zero risk on his side, as she had started with.
On the other hand, in incomplete markets we start with capital x_0, a liability $-L$ and hence we have a minimal risk $\rho^{x_0}(-L)$. If the trader sells an option H for $\Pi_{\text{sell}}(H)$, then she will set up a strategy starting from $x_0 + \Pi_{\text{sell}}(H)$. After optimal hedging, she will end up having a risk equal to $\rho^{x_0}(L)$, the same she had at the beginning.
In this sense, convex risk pricing extends the idea of "risk preservation" from complete to incomplete markets.

One example of convex and coherent risk measure is expected shortfall (also known as average value at risk) introduced in [Art+99] and studied in detail in [AT02a; AT02b]. Other examples can be found in [FS16].

4.4 Deep Hedging

Deep hedging is the expression used by Buehler, Gonon, Teichmann and Wood to denote a new type of hedging in financial markets based on deep learning, thus making use of neural networks. Their seminal paper [Bue+19] has received in recent years a lot of attention from the mathematical finance community because it allows to address many issues typically associated to this financial problem in a flexible (numerical) way. At the moment, it is even industrially used in one of the most prominent investment banks of the world and has granted Hans Buehler the title of "Quant of the year" in 2022.

[9] If X_n is a sequence of random variables uniformly bounded below from a constant and converging to X almost surely, then $\rho(X) \leq \liminf_{n\to+\infty} \rho(X_n)$.

The main innovation of deep hedging consists, as already said, in its flexibility: being able to handle at the same time any kind of transaction cost, market frictions, liquidity constraints on any type of market scenario for arbitrarily complex portfolios of derivatives, makes this approach extremely attractive from a practical point of view. Moreover, the model appears valuable for the fact that the any information, not only asset prices, but also market signals, news, preferences, can be included in the feature set as input for the neural network. On top of these attractive features, the authors do not even require per se that the market is free of arbitrage (even if these cases are ruled out by a mild condition on the risk measure). It goes without saying that this (new) methodology clearly allows traders to leave the safe shores of analytical and even complete-market models for deep-water, being guided by data alone.

4.4.1 A realistic framework

It is important at this point to specify the framework in which we work. Since we want to have a model that is as realistic as possible and simulate it on our computers, we consider a *discrete* probability space $(\Omega, \mathcal{F}, \mathbb{P})$ where $\Omega := \{\omega_1, \dots, \omega_N\}$ for some $N \in \mathbb{N}$, the probability measure $\mathbb{P}$ is assigning positive weight to all scenarios, i.e. $\mathbb{P}(\{\omega_i\}) > 0$ for all $i = 1, \dots, N$. The σ-algebra $\mathcal{F}$ can be considered to be equal to the power set 2^Ω.

Since in real life hedging is only allowed at discrete time, we fix $T > 0$ a maturity date and then $\mathcal{T} := (t_k)_{k=0}^n$ a sequence of dates denoting when trading is allowed. We also set $t_0 = 0$ and $t_n = T$.
Moreover, we model the available market information through the stochastic process $(I_k)_{k=0}^n$, with values in $\mathbb{R}^r$ for $r \in \mathbb{N}$. I_k denotes the information available at time t_k, for $k = 0, \dots, n$. The filtration generated by I is $\mathbb{F} = (\mathcal{F}_k)_{k=0}^n$, which means that $\mathcal{F}_k$ includes all information accessible in the market up to time t_k. In principle, the process I could model *all* information that are available: this includes derivatives' mid-prices (typically quoted through implied volatilities), market costs, news, trading signals, firms' balance sheets, as well as satellite-image counting cars in parking stores[10]. Every $\mathcal{F}_k$-measurable random variable can be written as a function of $I_0, \dots, I_k$; this is therefore the richest available feature set for any decision taken at time t_k.

The financial market is consisting of d hedging instruments whose mid-prices are denoted with the process $(S_k)_{k=1}^n$, thus S is an $\mathbb{R}^d$-valued stochastic process adapted to the filtration $\mathbb{F}$. As already mentioned, we do not require S being a martingale under an equivalent martingale measure. Consequently, we are not tied to any equivalent probability measure $\mathbb{Q}$ and, in fact, we could even use $\mathbb{P}$ for our computations. This is actually the case, for example, when using historical price trajectories. In principle, using the physical measure $\mathbb{P}$ makes things even more challenging because it is not possible to resort to martingale pricing techniques, which constitutes the standard approach in financial mathematics.
In the original paper [Bue+19], the authors consider every intermediate payment as accrued using a locally risk-free overnight rate. In view of this, they do not consider, as normally expected, a risk-free security, such as a money market account, and, hence, all rates are zero. Our approach slightly moves away from this one and, in order to enforce generality, we introduce another instrument denoted by $B = (B_k)_{k=0}^n$ that mimics the usual bank account and another process which represents the risk-free rate r, such that $B_k = \Pi_{j=0}^{k-1}(1 + r_j(t_{j+1} - t_j))$. Both B and r are also $\mathbb{F}$-adapted. Naturally, all dates t_j are included in $\mathcal{T}$. For the same reason, we will also consider dividend paying stocks, modelling dividends through a factor q that is paid at every instant of time (without loss of generality).

[10] More on this curious and alluring story can be found at `https://newsroom.haas.berkeley.edu/how-hedge-funds-use-satellite-images-to-beat-wall-street-and-main-street/`.

The goal is hedging our portfolio, which is made up of a liability $-Z$, which is a $\mathcal{F}_T$-measurable random variable, and an initial monetary amount $p_0 \in \mathbb{R}$.
Normally, the quantity Z is referred to as a contingent claim in the classical literature. We highlight that this is not strictly the case in our settings. As a matter of fact, we can consider Z as being any payoff, for instance deriving from the sum of different other financial instruments, including OTC derivatives. It is clear, from what written so far, that we are adopting the "seller" point of view, which will be kept throughout. Of course, the same approach would be valid, with opposite sign positions, for the "buyer" perspective.
Notice also that we will only adopt self-financing strategies, for which cash injection or extraction is only allowed at time 0. This is why we start with an initial endowment p_0: here a positive sign implies that money has been added to the portfolio, while a negative sign that money has been withdrawn.

In order to be able to cover the liability $-Z$ at time T, we need to invest the money p_0 in the market. This translates in trading in S using an $\mathbb{R}^d$-valued and $\mathbb{F}$-adapted stochastic process $\delta = (\delta_k)_{k=0}^{n-1}$ with $\delta_k = (\delta_k^1, \dots, \delta_k^d)$. Here, δ_k^i denotes the trader's long or short position, depending on the sign, at time t_k of the i^{th} asset. In addition, we define $\delta_{-1} = 0$, for notational convenience and $\delta_n = 0$, because at time $t_n = T$ the agent will have sold all her investments in order to hedge $-Z$. As will be clearer later, we can also include constraints on δ, for example due to liquidity or trading restrictions (e.g. no short sale allowed, etc...). The set of strategies $\delta \in \mathbb{R}^d$ satisfying all these constraints for all times in $\mathcal{T}$ is denoted with $\mathcal{H}$. Note, however, that the set of trading days actually used for hedging can be a subset of the days $\mathcal{T}$ on which trading is allowed.
At this point, we can rewrite the cumulative gain process from Definition 4.3.9 for discrete time. In [Bue+19], since there are no rates, this becomes

$$G(\delta) = \sum_{k=0}^{n-1} \delta_k \left(S_{k+1} - S_k\right) =: (\delta \cdot S)_T.$$

In case of self-financing strategies within a frictionless market, the value of their portfolio is

$$-Z + p_0 + (\delta \cdot S)_T. \tag{4.19}$$

Since we want to consider a realistic formulation of hedging, we also consider transaction costs. As noted in in [LPS98], it is actually possible to extend the definition of trading costs given in (4.9) to account also for them. This addition is sometimes called *transaction cost process* and is denoted with TC (we will, for ease of notation, just continue with C in the following). The process C can be written as

$$C_T(\delta) = \sum_{k=0}^{n} c_k(\delta_k - \delta_{k-1}). \tag{4.20}$$

Note that the same formula holds if we assume that depositing or withdrawing money from the bank account is free of charge. The function c_k, which we consider fixed once for all k, can acquire different form. A couple of possible choices are:

1) Fixed transaction costs: let us fix $c_k^i > 0$ and a threshold $\varepsilon > 0$, then $c_k(\psi) = \sum_{i=1}^d c_k^i \mathbb{1}_{\{|\psi^i| \geq \varepsilon\}}$;

2) Proportional transaction costs: let us fix $c_k^i > 0$, then $c_k(\psi) = \sum_{i=1}^d c_k^i S_k^i |\psi^i|$.

At this point, the trader's terminal wealth at T, sometimes also called PnL, that is "Profits and Losses", is not any longer (4.19), but becomes

$$\mathrm{PnL}\left(p_0, Z, \delta; T\right) := -Z + p_0 + (\delta \cdot S)_T - C_T(\delta). \tag{4.21}$$

In the following, we also assume that the non-negative $\mathbb{F}$-adapted cost functions are normalized, so that $c_k(0) = 0$, and that they are upper semi-continuous[11]. If we consider $d+1$ assets, with the addition of the bank account B, potential dividend paying assets, and for brevity we write $\mathrm{PnL}_k = \mathrm{PnL}\left(p_0, Z, \delta; k\right)$ and $P_k := \mathrm{PnL}_k + C_k(\delta)$ for the portfolio value without transaction costs, then we can write the evolution of the portfolio value in the following way:

$$\begin{aligned}
&\mathrm{PnL}\left(p_0, Z, \delta; 0\right) = P_0 = p_0, \\
&P_{k+1} = P_k + \left(P_k - \sum_{i=1}^{d} \delta_k^i S_k^i\right) r\, \Delta t + \sum_{i=1}^{d} \delta_k^i \left(S_{k+1}^i - S_k^i + q^i S_k^i \Delta t\right); \\
&\mathrm{PnL}_{k+1} = P_{k+1} - C_{k+1}(\delta); \\
&\mathrm{PnL}_T = -Z + P_T - C_T(\delta).
\end{aligned} \tag{4.22}$$

The interpretation should be clear: the value of the portfolio at the future instant of time t_{k+1} depends on the investments decided in the present, t_k, by investing δ_k^i in asset number i, while leaving the rest in the money market account. It might be useful to compare Equations (4.22) with Remark 4.3.4. In the following and as already done in(4.22), for ease of notation and without loss of generality, we will write Δt for any time interval.

Remark 4.4.1. Notice that we have made no assumptions on the model for S. The settings are extremely flexible. We could simulate S under an equivalent martingale measure or just use historical paths under $\mathbb{P}$ for it.

In an idealized complete market with continuous-time trading, no transaction costs and unconstrained hedging, for any liability $-Z$ there exists a unique replication strategy δ and a fair price $p_0 \in \mathbb{R}$ such that $-Z + p_0 + (\delta \cdot S)_T - C_T(\delta) = 0$ holds $\mathbb{P}$-a.s. As we have seen in Section 4.3.3, in incomplete markets we can and have to choose a particular optimality criterion to be able to find a price for any liability. For its generality and flexibility, the criterion that we will adopt here is using convex measures, as defined in Section 4.3.3. If we denote such a measure with ρ, then our goal becomes finding

$$\rho^{p_0}(-Z) = \inf_{\delta \in \mathcal{H}} \rho\big(-Z + P_T - C_T(\delta)\big) = \inf_{\delta \in \mathcal{H}} \rho(\mathrm{PnL}_T), \tag{4.23}$$

where the infimum is taken on all self-financing strategies that start from p_0 and satisfy all required constraints included in $\mathcal{H}$ (see also the definition of *minimal risk* in (4.17)).

4.4.2 Deep-learning the hedging strategy

To be able to handle hedging in such a realistic model, including restrictions that may act on liquidity, market prices, trading days, getting information from the entire financial market requires a likewise flexible numerical method.
It is no surprise that neural networks are actually suitable for such task. For definitions and theoretical background on neural networks, we remind to Chapter 2, Section 2.3 and following.

[11] A function f is upper semi-continuous at a point x_0 is $\limsup_{x \to x_0} f(x) \leq f(x_0)$. This technical requirement is necessary to prove convergence of the price found by the neural network to the true price, when the number of nodes in the hidden layer grows to infinity (see proof of Proposition 4.3 in [Bue+19].

The novelty of [Bue+19] comes from the fact that the network is not used in a completely supervised learning way with labeled data, but rather as a model for a functional to minimize. Consequently, it becomes natural to think of the neural network as an instrument to represent the convex risk measure ρ applied to the PnL at final time T, as described in the Equalities (4.23).

The main intuition is that of reproducing the entire portfolio evolution in time following Equations 4.22, so that

- the input layer will contain the input actually provided at the beginning of the strategy, that is the initial capital $p_0 \in \mathbb{R}$, the initial amount of money invested in the (risky) assets $\delta_0 \in \mathbb{R}^d$;
- the output layer returns the final value of the portfolio, that is P_T and then we subtract the liability Z and the costs due to the transactions $C_T(\delta)$ to reach PnL_T;
- the loss function of the neural network coincides with the chosen convex risk measure ρ applied to PnL_T. Notice that this choice is financially grounded;
- there is actually a neural network for all trading day t_k and these are of course connected so that information and strategies can unroll and evolve from time $t_0 = 0$ to time $t_n = T$.

This final point is maybe that most unusual one and it is worth spending some words on it. Since it is clear from (4.22) that there is a recurrent structure inside the evolution of the portfolio variable P, and that the input at time t_{k+1} is the output obtained at time t_k, it raises as natural the idea of considering a chain of neural networks that forms a larger one. Inside each of these day-networks, there are some layers designated for the computation of the strategy δ_k, that is then passed at the next network (in the chain). Note that these networks that are computing the strategy for time t_k do not have a loss function. In the training procedure, all weights of the neural-chain are then adjusted at the same time to minimize (4.23). This methodology, which is chaining many different networks, has been called *semi-recurrent*.

In principle, the information at time t_k that is available in the market is generated by $I_0, \dots, I_k$. Therefore, the strategy δ_k should be dependent on this information only: all other relevant input to determine δ_k, such δ_{k-1} or S_k among the others, is also a function of $I_0, \dots, I_k$. This naturally translates in machine learning language: in every day in which hedging is allowed, the associated δ_k should be a function of $I_0, \dots, I_k$ modelled through a neural network:

$$\delta_k = F_k(I_0, \dots, I_k) \quad \text{for } k = 0, \dots, n-1$$

for some neural network F_k to be defined. Note that these networks modelling the strategies are the only variable part in our chain of networks. In other words, these networks F_k are the only ones that present weights (and biases) and that are trained while minimization of (4.23) takes place. All other operations implemented through networks, more specifically, by layers of networks, are achieved through "fixed" functions - not dependent on trainable weights - determined by Equations 4.22. In light of this consideration, we can re-write our goal 4.23 in the following way:

$$\rho^{p_0}(-Z) = \inf_{\delta \in \mathcal{H}_{\mathcal{N}}} \rho\big(-Z + P_T - C_T(\delta)\big) \tag{4.24}$$

where $\mathcal{H}_{\mathcal{N}}$ is the new set of considered strategies. We can define it as

$$\mathcal{H}_{\mathcal{N}} := \left\{ (\delta_k)_{k=0}^{n-1} \in \mathcal{H} \,:\, \delta_k = F_k(I_0, \dots, I_k), F_k \in \mathcal{N}^{\sigma}_{r(k+1),d},\, k = 0, \dots, n-1 \right\}, \tag{4.25}$$

with $\mathcal{N}^{\sigma}_{r(k+1),d}$, as introduced in Definition 2.3.1, is a neural network with input dimension $r(k+1)$, output dimension d and activation function σ. Further, the set $\mathcal{H}_{\mathcal{N}}$ defined in (4.25) can again be re-written as

$$\mathcal{H}_{\mathcal{N}} = \left\{ (\delta_k^{\xi})_{k=0}^{n-1} \in \mathcal{H} \,:\, \delta_k^{\xi} = F^{\xi_k}(I_0, \dots, I_k), \xi_k \in \Xi_{r(k+1),d},\, k = 0, \dots, n-1 \right\} \tag{4.26}$$

where the new set Ξ_{d_1,d_2} denotes all parameters associated with a neural network whose input and output dimensions are d_1 and d_2, respectively. On purpose we leave unspecified the number of layers and the number of nodes in hidden layers as this is free to vary (as well as other architectures' choices). The only requirement is having, at least, shallow neural networks with a non-linear activation function, such as a sigmoid, to make universal approximation in the sense of Theorem 2.4.6 possible. If we set the parameter space $\Xi := \Pi_{k=0}^{n-1} \Xi_{r(k+1),d}$, then we can rewrite our optimization goal as

$$\rho^{p_0}(-Z) = \inf_{\delta \in \Xi} \rho\big(-Z + P_T - C_T(\delta) \big) \tag{4.27}$$

This should clarify what we previously wrote, that all weights in the neural network chain are optimized to solve problem (4.23).

Note that in the original article [Bue+19] the authors also proved that using shallow networks, as one would expect, is enough: if the hidden layer dimension is arbitrarily large, it is possible to approximate the true time 0 price of the liability arbitrarily well.

Remark 4.4.2 (Markovianity). If S is an $\mathbb{F}$-adapted Markov process under the measure $\mathbb{P}$ and the liability/contingent claim $-Z$ is of the form $Z := g(S_T)$ for a pay-off function $g : \mathbb{R}^d \to \mathbb{R}$, then we could rewrite the optimal strategy as $\delta_k = f_k(I_k, \delta_{k-1})$ for some function $f_k : \mathbb{R}^r \times \mathbb{R}^d \to \mathbb{R}^d$.

Remark 4.4.3 (Recurrent structure). This framework becomes truly recurrent when we decide that $\xi_k = \xi_0$ for all k. This was the approach also used in [HTŽ21] where they applied deep hedging to non-Markovian rough models. In this case, hidden states of the recurrent neural network (RNN) are passed on to the next trading day and other output variables are passed to the loss function of the network.

Remark 4.4.4. If we leave p_0 unspecified, we can recover the price of the liability $-Z$, according to the principle specified by the convex function ρ.

4.5 Model Free Deep Hedging

As already presented in Remark 4.2.3, we can use the approach developed in [DR14] to hedge derivatives. This can be done employing the posterior distribution defined in (4.4) using the j^{th} model log-likelihood computed as in (4.3). Our endeavour is thus to merge the two approaches presented in Section 4.2 and Section 4.4.

From now on, we will not use the index j to denote models belonging to the pool fixed a priori, but rather θ, which denotes the parameters for the a particular model. Moreover, we suppose that the posterior distribution π is already available. This usually entails a certain training period, sometimes called *burn in phase*, before the probability function is actually "ready" to be used. In addition, the fact that we start with $\ell_0^{(\theta)} = 0$ in (4.3) for all models, implies that we do not impose any particular view on the market. Otherwise said, we adopt a *non-informative* prior distribution.

After the burn in phase, we can simply ignore all time passed, since all required information is summarized in π and in all $\ell^{(\theta)}$, therefore we suppose to be at time $t = 0$. Let us start by defining with:

- π the posterior mixture distribution obtained by the Estimate Nothing approach.
- S^{Hist} the whole path of the historical trajectories, whose dynamics are unknown. We can for example think of it as the S&P500 index.
- S_θ^{Sim} the entire set of all simulations on which the deep hedging neural network has been trained. Normally, we need a fan of simulations for each θ representing a model of our pool. Note that the simulations used to train the network are then discarded.
- NN is the deep hedging neural network used to learn the optimal strategy. It is implemented in a semi-recurrent fashion as described in Section 4.4.2. Read also Remark 4.5.2 below.
- P_θ the price of the selected derivative at time $t = 0$ computed using the model indexed by θ (see Remark 4.5.1 below).
- $\delta_{t,\theta} := \text{NN}_\delta(S_{t,\theta}^{\text{Sim}})$ the delta of the derivative computed by the trained neural network NN on the model simulations $S_{t,\theta}^{\text{Sim}}$. The time index t refers to the subset of instant of times in which trading is performed.
- F the payoff-function of the derivative, so that $F(S^{\text{Hist}})$ represents H, a (random) payoff at time T, or Z, a (random) liability that we need to pay back. In principle, we can also consider path-dependent payoff functions, such as for Asian options.

Then the goal is computing

$$\int_\Theta P_\theta \, \pi(d\theta) + \left(\int_\Theta \delta_\theta \, \pi(d\theta) \cdot S^{\text{Hist}} \right)_T - F\left(S^{\text{Hist}}\right). \tag{$\star$}$$

Equation ($\star$) represents the PnL at final time T, weighted by our posterior distribution on the strategy side, hence conceiving a mixture model. Actually, we would expect Equation ($\star$) to have mean value "close" to zero and restrained variance to acquire confirmation that hedging and pricing worked well.

Remark 4.5.1. There might be different ways for computing the price P_θ for all different models. For example, if a closed-form formula were available, we could simply use it by inputting the parameters θ. Another possibility would be using the neural network trained for deep hedging (the same used to get the delta's $\delta_{t,\theta}$ as side-products), by leaving p_0 unspecified (Remark 4.4.4). In this case, the simulations S_θ^{Sim} are used as input for NN.

Remark 4.5.2. One of the main novelties we introduced is considering the neural network not only as a map acting the price process alone, but also on the parameters θ that generated the trajectories with which the network has been trained.
In this way, once the network is fully trained, we can employ it to instantaneously evaluate prices and strategies for a number of models, without the need of having more pricers and hedgers at the same time. The employ of a machine learning algorithm as neural network, that can offset the training time with instantaneous evaluations once training is finished, empower the approach by Dümbgen and Rogers in the sense of enabling more complex models that would be normally out of sight because of lengthy computations (see Box 4.1 and Remark 4.2.5).

Remark 4.5.3 (**Model Free Deep Hedging**). Most importantly, we solved one of the few deficiencies of Deep Hedging as developed in [Bue+19], that is model-dependency.

Remark 4.5.4. This proposal is not the first attempt of creating a bond between robust finance and deep hedging. A first trial has been advanced by Lütkebohmert, Schmidt and Sester in

[LSS21]. The chosen Markovian class of models is called *generalized affine process* and includes many notorious examples, such as Black-Scholes ([BS73]), Cox-Ingersoll-Ross (CIR) ([CIR85]) or Vašíček models ([Vaš77]). The dynamics of the continuous semimartingale S are described as unique strong solution to the SDE

$$dS_t = (b_0 + b_1 S_t)\, dt + (a_0 + a_1 S_t)^\gamma\, dW_t,$$

for $a_i, b_i \in \mathbb{R}$, $i = 0, 1$, $\gamma \in [1/2, 1]$ suitably selected and with $S_0 = x \in E$ for the state space E (either $\mathbb{R}$ or $\mathbb{R}_+$). In this case, model uncertainty is modelled by allowing the parameters to take value in a set $\Theta := \Pi_{i=1}^5 [\underline{\theta_i}, \overline{\theta_i}]$, where the underline and overline denotes the minimum and maximum, respectively, that the five parameters (a_0, a_1, b_0, b_1, γ) of the model can reach.

4.5.1 Numerical implementation

In order to verify the goodness of the idea, we numerically implemented on Python, using Tensorflow [Aba+15] as an API to realize neural networks.

To simulate the market environment, we decided to adopt a Bates model as true/historical model, while a set of candidate Merton models as our pool of models. The derivative under investigation is an at-the-money European option, whose value was computed with the help of QuantLib [AB+03]. For a description of Bates model, we remind to Section 3.3. While Merton model is a jump-diffusion which is basically made by geometric Brownian motion and a compound Poisson process. The pool of models is indexed by the parameter θ which represents the diffusion coefficients. The model dynamics are

$$d \log S_t = \left(r - q - \frac{1}{2}\theta^2 - \lambda m \right) dt + \theta\, dW_t + \Upsilon dN_t, \tag{4.28}$$

where $\Upsilon \sim \mathcal{N}(\nu, \delta^2)$, $m = \exp(\nu + \frac{1}{2}\delta^2) - 1$ and $dN_t \sim \text{Pois}(\lambda\, dt)$. The interval chosen for θ is $[0.1, 0.7]$, discretized with a step equal to 0.04.
The deep hedging network has a size which depends on the maturity of the derivative, a European ATM call option with 30-days expiry, but each network implementing the non-linear function was shallow, with just 16 nodes in the hidden layer. The initial learning rate to train the network with ADAM ([KB15]) was 0.01. The batch-size was chosen to be 256. For every parameter-combination, 10'000 paths were generated to form the training set of the deep hedging and the duration of the network could vary, but usually 15/20 epochs were sufficient to obtain good results.
The burn in phase for π can be easily adapted, but in the following we will see results obtained after 500 steps. The functional $\mathcal{Q}$ in Equation 4.3 we used was the square Euclidean distance on implied volatility surfaces (since we did not notice a remarkable improvement using the method proposed in [DR14]).
Prices in Equation ($\star$) were computed using the neural network trained for deep hedging.
All deep-hedging algorithms have been trained using a mean-variance hedging strategy.

4.5.2 Graphical results

In Figure 4.1 one can see two different point of views (different angles) on the same implied volatility estimation by means of the 'Estimate nothing' method. On the left part of the pictures, we can see the two implied volatility surfaces: in blue there is the one associated to (historical) Bates model, while in orange the one obtained by the mixture model using only Merton models, where the value of the diffusion coefficient is takes place in the interval $[0.1, 0.7]$. Below, in

Figure 4.2 one can see the likelihood on the parameter θ after a rather long burn in phase, appropriately normalized. In this case, the value of θ are concentrating around 0.4.

In Figures 4.3 one can see the output of Deep Hedging. We compare different histograms where Deep Hedging has been used to train a neural network on Merton paths. The second plot is just an enlargement of the first.

Finally, in Figure 4.4, we can see the histogram representing Equation $\star$. In orange, we can see the histograms originating from applying Deep Hedging on Bates trajectories, while in blue we see the bars coming from Deep Hedging applied on Bates model. From the picture, we can appreciate the goodness of the proposal algorithm. In fact, the mean of the PnL is 0.076, very close to zero, while its variance is 5.74, slightly larger than the variance provided by Deep Hedging on Bates, which is 5.5. We would like to recall that no minimization was operated in Equation $\star$ to reach these results. Results for this simulation are summarized in Table 4.1.

Model	Mean of ($\star$)	Variance of ($\star$)	Call Price
MF-DH	0.0765	5.747	5.513
DH-Bates	0.0023	5.521	5.412

Table 4.1: The table collects key numbers for the simulation described in Section 4.5.1. In the first row, the results obtained by Model Free Deep Hedging, while in the second row the results from applying Deep Hedging on the Bates model. Monte Carlo price: 5.524, Fourier-based Bates price: 5.408. Initial conditions: $S_0 = 118.52$, $V_0 = 0.1485$, $r = q = 0$, $\kappa = 0.8$, $\theta = 0.22$, $\sigma = 0.15$, $\rho = -0.53$, $\lambda = 0.15$, $\nu = -0.29$, $\delta = 0.30$, $dt = 1/365$.

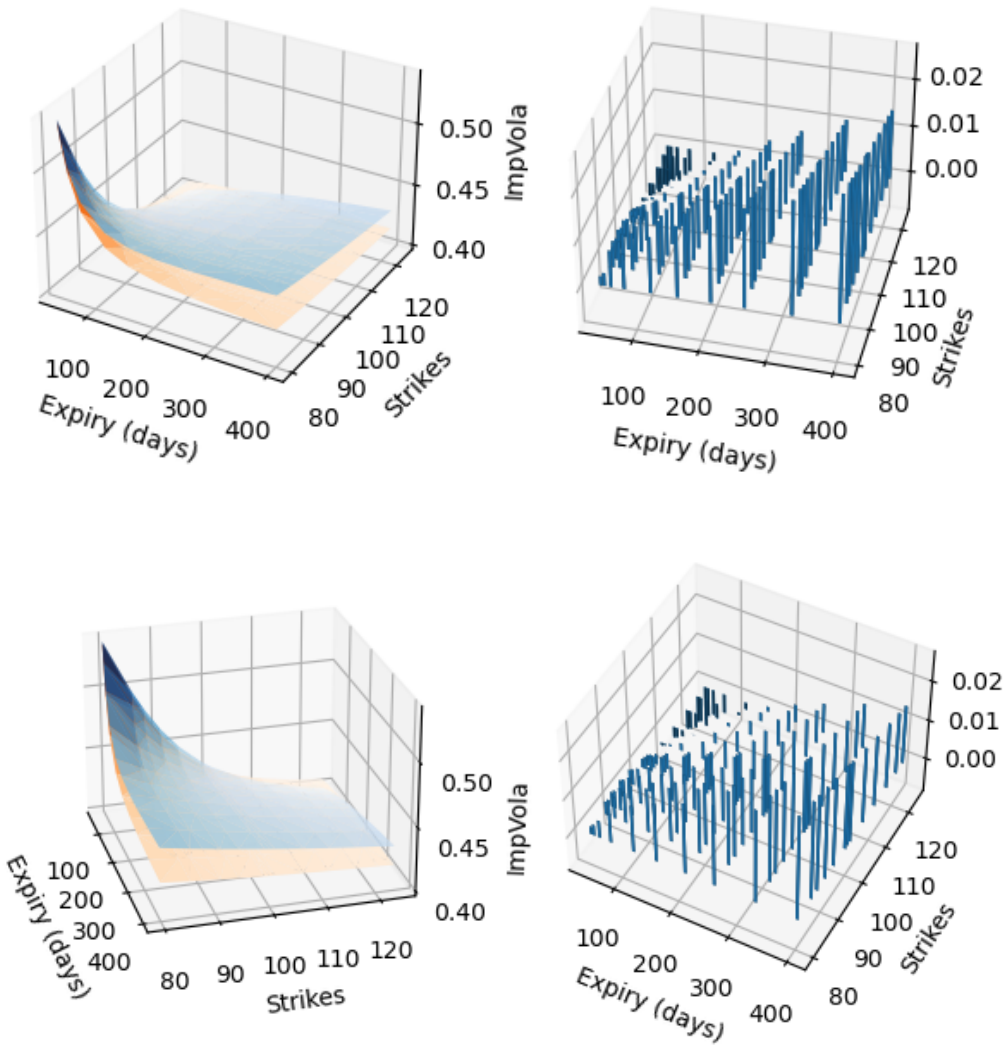


Figure 4.1: Output of the algorithm proposed in [DR14] by Dümbgen and Rogers. On the left, we can see the two IVS (in blue the target IVS, while in orange the mixture model IVS). On the right the pointwise difference between the two IVS.

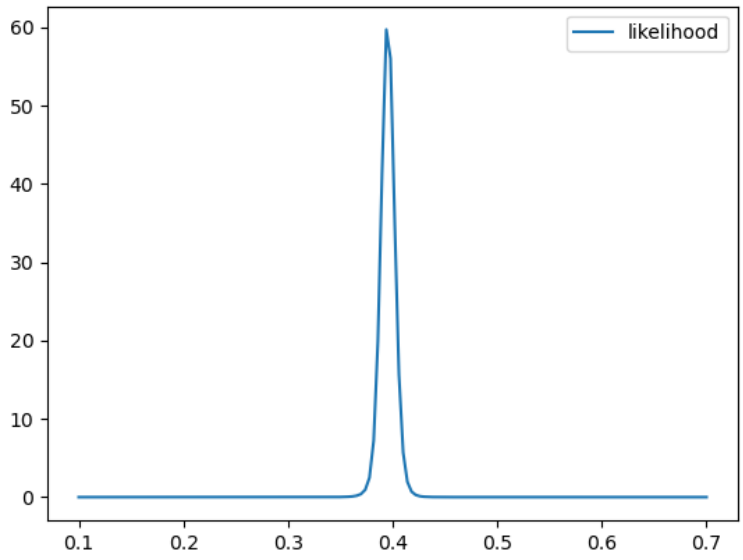


Figure 4.2: Final likelihood on the diffusion coefficient θ of Equation 4.28.

4.6 Python code

This section collects parts of the code needed to produce the results of this chapter.

In the Listing 4.1, we can read the code to compute the density π obtained following the 'Estimate Nothing' approach. The code starts from a non-informative prior inside the function `bayesian_update_EN` and this gets updated using a Bayesian updating principle `iter` times. At every instant of time, a new object of class `Merton` is created that is used to estimate the best fitting diffusion coefficient parameter exploiting both the density function of the underlying process S, `likelihood1`, and the L^2-difference in the prices, `likelihood2`. These are then summed, weighting the second variable with a log-likelihood coefficient `ll2_coeff`. Inside the function `compute_mixture_ivs`, the resulting density π is normalized by dividing for its L^1-norm and the final implied volatility surface is calculated as mixture model using the density π.

```
1
2  import numpy as np
3  import pickle
4  from tqdm import tqdm
5  from scipy.integrate import simpson
6  from scipy.stats import norm
7  from matplotlib import pyplot as plt
8  from utils import read_input_data, min_vola, max_vola
9  from vola_class import volaSurface
10 from merton_model import Merton, L2_difference
11 from bates import jump_params_length
12
13 option_type_str = 'call'
14
15 def apply_mask(pi, threshold=1e-3):
```

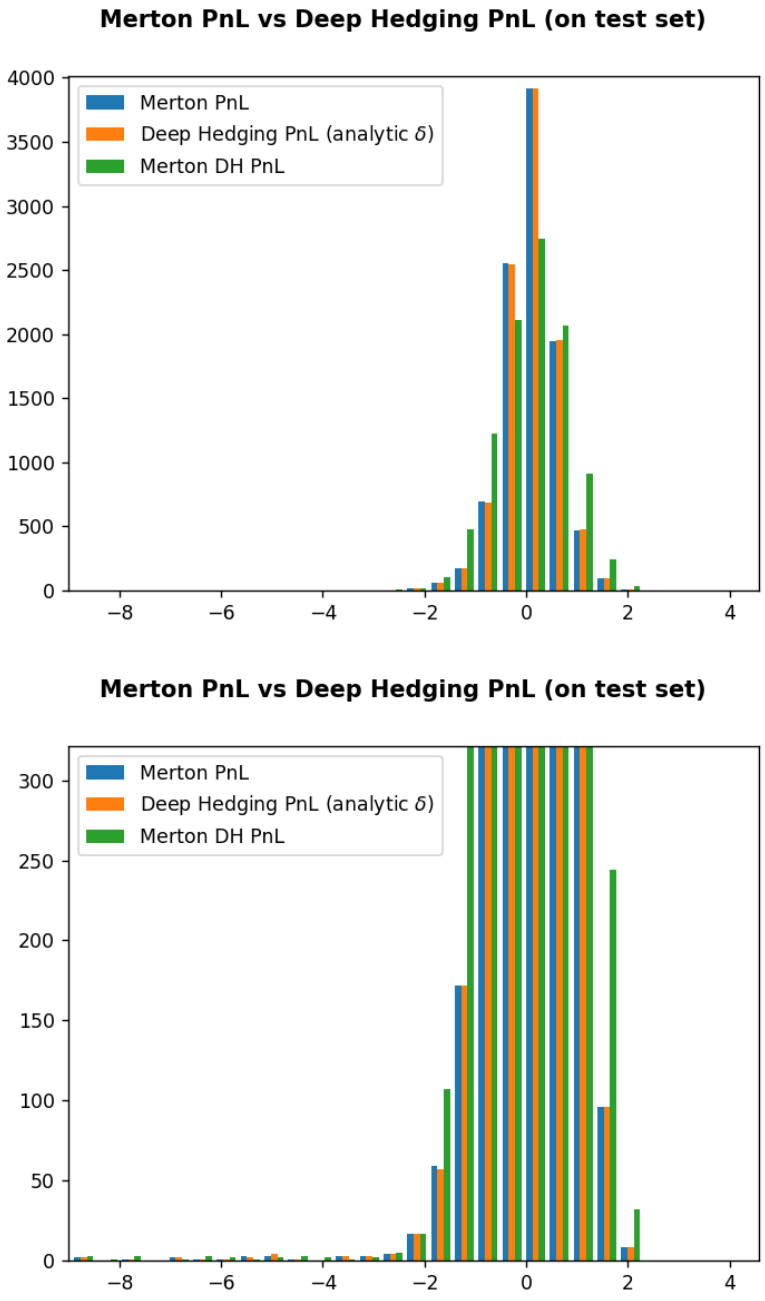


Figure 4.3: Output of the algorithm Deep Hedging from [Bue+19] used to train a neural network on the Merton model. 'Merton PnL' stands for the PnL computed by analytical formulas; 'Deep Hedging PnL (analytical δ)' stands for the PnL computed with analytical δ, but using the price found by the network; 'Merton PnL' stands for the PnL entirely computed via Deep Hedging. The second plot is just an enlargement of the first.

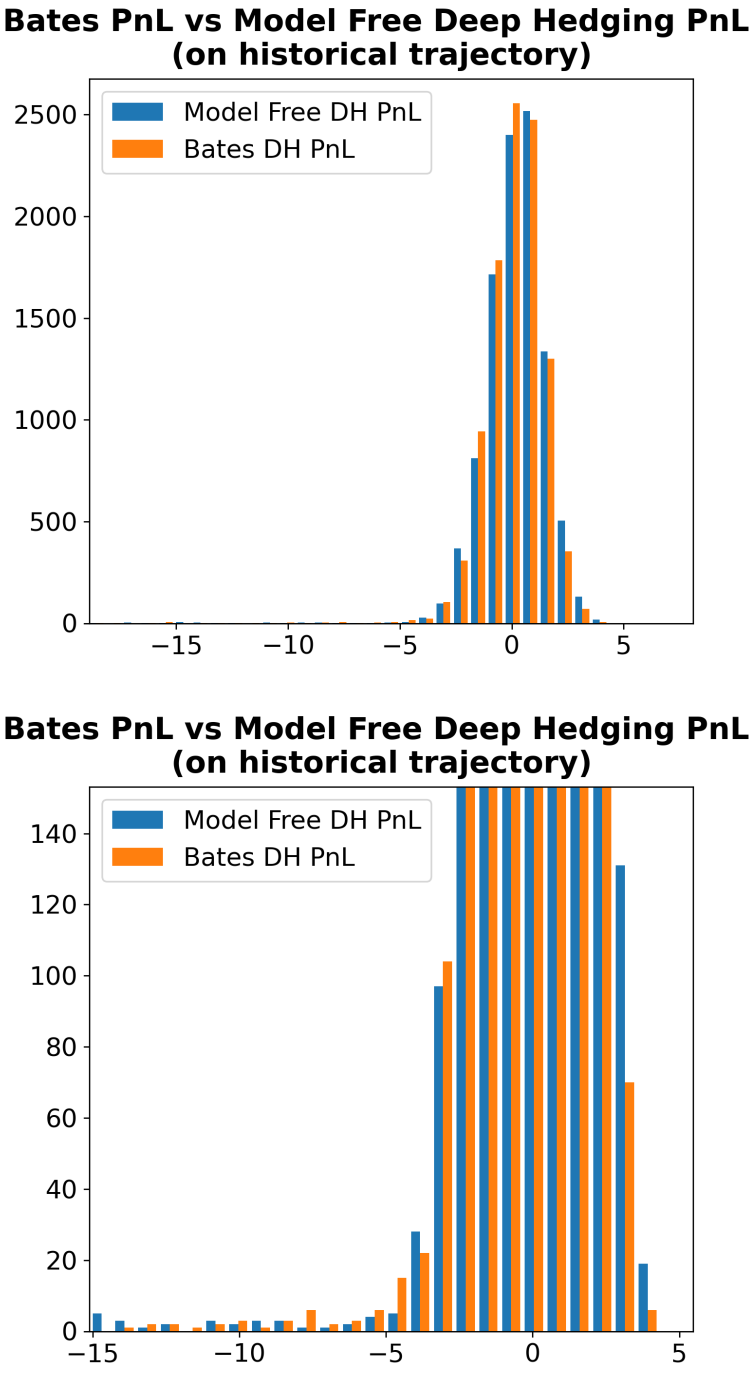


Figure 4.4: Output of the algorithm Model Free Deep Hedging proposed in this thesis. 'Mean-price Model Free DH PnL' stands for the PnL computed following Equation $\star$; 'Bates DH PnL' stands for the PnL entirely computed via Deep Hedging on Bates trajectories. The second plot is just an enlargement of the first.

```
    """
    Filtering to speed up computations.
    :param pi: EN posterior density
    :param threshold: limit below which weights are disregarded
    :return: filtered values of x and y on which I will compute the mixture ivs
    """
    mask = pi['y'] >= threshold
    pi['x'] = pi['x'][mask]
    pi['y'] = pi['y'][mask]
    return pi


# In this case, the mixture ivs is computed only with merton models where the diffusion coefficient is allowed to change
def compute_mixture_ivs(pi, merton_obj, target_ivs, ivs_struct, method=2, op_type=option_type_str):
    """
    Compute the weighted implied volatility of mixture Merton model and return the mixture ivs and the difference
    between the target_ivs and the mixture ivs.
    :param pi: dictionary, density distribution with keys x and y
    :param merton_obj: merton model object
    :param target_ivs: implied volatility surface which I want to approximate
    :param ivs_struct: implied volatility surface which I want to approximate
    :param method: 1: linear combination of IVS, 2 linear combination of prices
    :param op_type: 'call' or 'put'?
    :return: 1) ivs: implied volatility surface coming from the mixture model
             2) L2 difference between (model) "ivs" and "target_ivs"
    """
    assert 'x' in pi and 'y' in pi, '[compute_mixture_ivs] Mixture density is missing "x" or "y"!'
    ivs = np.zeros_like(target_ivs)
    S0 = merton_obj.S0
    pi['y'] /= pi['y'].sum()  # Normalize
    pi['x'] = np.concatenate((pi['x'], np.array([pi['x'][-1]+0.1])))  # add last point for dx
    if method == 1:  # 1) linear combination of IVS
        for j, x in enumerate(pi['x'][:-1]):
            prices = merton_obj.Merton_ivs_pricer(S0=S0, sigma=x, ivs_struct=ivs_struct, op_type=op_type)
            tmp_ivs = volaSurface.get_vola(S0, merton_obj.r, merton_obj.q, ivs_struct=ivs_struct, price_vec=prices)
            ivs += pi['y'][j] * np.array(tmp_ivs)
    elif method == 2:  # 2) linear combination of prices from which we derive the IVS
        prices = np.zeros_like(target_ivs)
        for j, x in enumerate(pi['x'][:-1]):
            tmp_prices = merton_obj.Merton_ivs_pricer(S0=S0, sigma=x, ivs_struct=ivs_struct, op_type=op_type)
            prices += pi['y'][j]*np.array(tmp_prices)*pi['const']*(pi['x'][j+1]-x)
        ivs = volaSurface.get_vola(S0, merton_obj.r, merton_obj.q, ivs_struct=ivs_struct, price_vec=prices)
    print('mixture ivs', ivs)
    print('target_ivs', target_ivs)
    print('diff', ivs-target_ivs)
    return ivs, L2_difference(ivs, target_ivs), pi


def bayesian_update_EN(name_params_list, name_volas_list, l12_coeff=-0.15, iter=10, ivs_plot=False, method=2, beta=0.95, points=150, save_for_later=True, seed_str='', mask=False):
    """
```

```
    Continuous Bayesian update of posterior likelihood on the diffusion coefficient of the Merton model following the
    "Estimate Nothing" approach.
    :param name_params_list: name of the list of parameters to load
    :param name_volas_list: name of the list of volatility structures to load
    :param ll2_coeff: log-likelihood2-coefficient to balance the transition density
    :param iter: number of iteration until
    :param ivs_plot: plot IVS?
    :param method: 1: linear combination of IVS, 2 linear combination of prices (from which we get ivs)
    :param beta: beta of EN, used to "forget" older contributions
    :param points: number of points to use for the posterior distribution
    :param save_for_later: want to save the (filtered) posterior distribution and the initial conditions?
    :param seed_str: string of integer number --> should be the same as in name_params_list and name_volas_list
    :param mask: want to filter EN-pdf (based on "y" values)?
    :return: 1) final posterior likelihood on diffusion coefficient; 2) model ivs;
             3) L2 difference between model ivs and the loaded ivs selected by "
    iter"
    """
    assert 0 < beta <= 1, '[bayesian_update_EN] "beta" is supposed to be in (0,1]'
    p_list, v_list = read_input_data(name_params_list, name_volas_list)
    assert len(p_list) == len(v_list), '[bayesian_update_EN] Loaded object from AFE have different lengths'
    mjs_pos, sjs_pos = jump_params_length(p_list)
    MAX_N_TIMES = len(p_list)-1
    xx = np.linspace(min_vola, max_vola, points)
    tot_log_likelihood = [np.zeros_like(xx)]
    for i in tqdm(range(max([1, min([iter, MAX_N_TIMES])]))):
        merton = Merton(S0=p_list[i][0], sigma=np.sqrt(p_list[i][1]), r=p_list[i][2], q=p_list[i][3], jump_intensity=p_list[i][8], mean_jump_size=p_list[i][mjs_pos], stdev_jump_size=p_list[i][sjs_pos])
        likelihood1 = merton.trans_density_sigma(fPrice=p_list[i+1][0], iPrice=merton.S0, dt=1./365)
        likelihood2 = merton.vola_diff_sigma(obs_vola=np.array(v_list[i]['ivs']), S0=merton.S0, ivs_struct=v_list[i], op_type=option_type_str)
        yy1 = [likelihood1(x) for x in xx]
        yy2 = [likelihood2(x) for x in xx]
        yy1 = np.log(yy1); yy2 = np.log(yy2)
        fig = plt.figure()
        plt.plot(xx, yy1, label='log-likelihood1'); plt.plot(xx, ll2_coeff*yy2, label='ll2_coeff * log-likelihood2')
        plt.plot(xx, yy1 + ll2_coeff*yy2, label='sum log-likelihoods')
        plt.plot(xx, beta*tot_log_likelihood[i]+yy1+ll2_coeff*yy2, label='sum log-likelihoods with past')
        plt.legend(); #plt.show()
        fig.savefig("en_likelihoods_in_time/seed_"+seed_str+"_ll2_coeff"+str(ll2_coeff)+"_t"+str(i)+".png")
        tot_log_likelihood.append(beta*tot_log_likelihood[i] + yy1 + ll2_coeff*yy2)
    tot_likelihood = np.exp(tot_log_likelihood[-1])  # get last element, but we might need more
    # Normalize likelihood to have density 1
    area = simpson(tot_likelihood, xx)
    tot_likelihood = tot_likelihood / area
    print('tot_likelihood:\n', tot_likelihood)
    fig = plt.figure()
    plt.plot(xx, tot_likelihood, label='likelihood')
    plt.legend();  # plt.show()
    fig.savefig("en_likelihoods_in_time/seed_"+seed_str+"_ll2_coeff"+str(ll2_coeff
```

```
    )+"_tFinal"+str(i)+".png")
    pi = {'x': xx, 'y': tot_likelihood, 'const': tot_likelihood.sum()}
    if mask:
        pi = apply_mask(pi)
    m_ivs, ivs_L2diff, pi = compute_mixture_ivs(pi, merton, target_ivs=np.array(
    v_list[i]['ivs']), ivs_struct=v_list[i], method=method)
    print('L2 diff:', ivs_L2diff)
    if ivs_plot:
        v_list[i]['m_ivs'] = m_ivs
        print(v_list[i])
        plt.close("all")
        volaSurface.show_vola(ivs_struct=v_list[i], one_plot=False)
    if save_for_later:
        string_name = 'EN_post_plus_i_cond_seed'+seed_str+'_iter'+str(iter)+'
    _points'+str(points)
        string_name = string_name+'_mask.pl' if mask else string_name+'_no_mask.pl
    '
        with open(string_name, 'wb') as obj:
            pickle.dump([pi, p_list[i]], obj)
    return tot_likelihood, m_ivs, ivs_L2diff, pi



if __name__ == "__main__":
    ## Bates model - EN
    # seed_str, iterations = '3', 50
    seed_str, iterations = '28', 500
    name_params_list = 'params_list_seed'+seed_str+'_times'+str(iterations)+'
    _stddev0.05.pl'
    name_volas_list = 'volas_list_seed'+seed_str+'_times'+str(iterations)+'
    _stddev0.05.pl'
    # Merton model
    # name_params_list = 'params_list_seed1116750_times50_stddev0.05_merton.pl'
    # name_volas_list = 'volas_list_seed1116750_times50_stddev0.05_merton.pl'
    print(name_params_list)
    print(name_volas_list)
    bayesian_update_EN(name_params_list, name_volas_list, ivs_plot=True, method=2,
     iter=500, seed_str=seed_str, mask=False)
```

Listing 4.1: Code for the computation of the density π following the 'Estimate Nothing' approach.

Listing 4.2 contains the code for the definition of the class and function responsible for neural network implementation making use of Tensorflow ([Aba+15]). The most important function is `Deep_Hedging_Model`, which effectively implements deep hedging. This is realized by receiving in input the financial parameters of the model, the price at initial time of the underlying, i.e. S_0, the information we will use inside the network as input for the neurons. The two latter inputs are then also provided at every instant of time `j`, that is t_j, throughout the developing of the hedging strategy. The class `StrategyLayer`, on the other hand, is responsible for implementing the (non-linear) function g that maps the underlying S_{t_j}, the financial parameters p and, possibly, the delta at previous time $\delta_{t_{j-1}}$, to the current delta δ_{t_j}: $\delta_{t_j} = g(S_{t_j}, p, \delta_{t_{j-1}})$. Quite interestingly, the function `Delta_SubModel` is used to extract from the deep hedging neural network the portion of it that is providing as output the delta itself. This is the reason why the argument `days_from_today` is needed: it is used to exactly know where to interrupt the trained network so that it can return the corresponding δ. Finally, the purpose of the function `prepare_input_delta_NN` is used to aggregate different datasets, namely the input for the non-linear function g.

```
from tensorflow.keras.layers import Input, Dense, Concatenate, Subtract, Lambda,
    Add, Dot, BatchNormalization, Activation, LeakyReLU
from tensorflow.keras.models import Model, load_model
```

```
from tensorflow.keras.initializers import he_normal, Zeros, he_uniform,
    TruncatedNormal
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau
import tensorflow.keras.backend as K_back
import tensorflow as tf
from eu_options import EuropeanCall

def softplus(x, t=2):
    return tf.math.log(1+tf.math.exp(t*x))/t


class StrategyLayer(tf.keras.layers.Layer):
    def __init__(self, n=None, m=None, nodes=None, use_BN=None, kernel_initializer
    ="he_uniform", bias_initializer="he_uniform", act_hidden="relu", act_output="
    linear", delta_constraint=None, transaction=None):
        """
        Build NN that gives as output the strategy (delta). The input depends on
    the type of chosen strategy.
        :param n: number of hidden layers for strategy
        :param m: dimension of process
        :param nodes: number of neurons for hidden layers (apart from last one)
        :param use_BN: use batch normalization?
        :param kernel_initializer: random matrix initialization
        :param bias_initializer: random vector initialization
        :param act_hidden: activation function for dense layers
        :param act_output: activation function for output layer
        :param delta_constraint: tuple (delta_min, delta_max) to apply to output
    layer
        :param transaction: time ---> important for naming
        """
        super(StrategyLayer, self).__init__(name="delta_"+str(transaction))
        assert n > 1, "[StrategyLayer]: At least two layers are needed"
        self.n = n
        self.m = m
        self.nodes = nodes
        self.use_BN = use_BN
        self.act_hidden = act_hidden
        self.act_output = act_output
        self.kernel_initializer = kernel_initializer
        self.bias_initializer = bias_initializer
        self.intermediate_dense = [None for _ in range(self.n)]
        self.intermediate_BN = [None for _ in range(self.n)]
        self.delta_constraint = delta_constraint
        for j in range(self.n):
            self.intermediate_dense[j] = Dense(self.nodes, kernel_initializer=self
    .kernel_initializer, bias_initializer=self.bias_initializer, use_bias=(not
    self.use_BN))
        if self.use_BN:
            for j in range(self.n):
                self.intermediate_BN[j] = BatchNormalization()
        self.output_dense = Dense(self.m, kernel_initializer=self.
    kernel_initializer, bias_initializer=self.bias_initializer, use_bias=True)

    def call(self, input):
        for j in range(self.n):
            if j == 0:
                output = self.intermediate_dense[j](input)
            else:
                output = self.intermediate_dense[j](output)
            if self.act_hidden == "leaky_relu":
                output = LeakyReLU()(output)
            elif self.act_hidden == "softplus":
```

```
57                 output = softplus(output)
58             else:
59                 output = Activation(self.act_hidden)(output)
60             if self.use_BN:  # batch normalization
61                 output = self.intermediate_BN[j](output, training=True)
62         output = self.output_dense(output)
63         if self.act_output == "leaky_relu":
64             output = LeakyReLU()(output)
65         elif self.act_output in ("sigmoid", "tanh", "hard_sigmoid"):
66             if self.delta_constraint is not None:  # delta constraints for B&S
    model
67                 output = Activation(self.act_output)(output)
68                 delta_min, delta_max = self.delta_constraint
69                 output = Lambda(lambda x: (delta_max-delta_min)*x+delta_min)(
    output)
70             else:
71                 output = Activation(self.act_output)(output)
72         return output
73
74
75 def Deep_Hedging_Model(N=None, n=None, fpd=None, m=1, nodes=16, r=0.0, q=0.0, dt
    =1./365, initial_wealth=0.0, final_cost=False, strategy_type=None, use_BN=None
    , kernel_initializer="he_uniform", bias_initializer="he_uniform", act_hidden="
    relu", act_output="linear", delta_constraint=None, cost_type="proportional",
    epsilon=0.0, **kwargs):
76     """
77     Returns neural network for Deep Hedging.
78     :param N: number of transactions (usually 1 per day)
79     :param n: number of hidden layers for strategy
80     :param fpd: financial model parameters dimension
81     :param m: dimension of process
82     :param nodes: number of neurons for hidden layers (apart from last one)
83     :param r: risk-less interest rate
84     :param q: dividend rate
85     :param dt: time difference of the discretized process
86     :param initial_wealth: initial amount of wealth, default is 0
87     :param final_cost: commission fees at last transaction?
88     :param strategy_type: "simple" or "semi_recurrent"
89     :param use_BN: use batch normalization?
90     :param kernel_initializer: random matrix initialization
91     :param bias_initializer: random vector initialization
92     :param act_hidden: activation function for dense layers
93     :param act_output: activation function for output layers
94     :param delta_constraint: tuple (delta_min, delta_max) to apply to output layer
95     :param cost_type: "constant" or "proportional"
96     :param epsilon: fixed amount or proportional amount of costs
97     :return: tensorflow model (neural network)
98     """
99     assert N > 0, "[Deep_Hedging_Model]: Number of transactions needs to be at
    least 1"
100     monitoring_steps = kwargs.get('fixing_days', [N])
101     # Price and available information (input for NN with previous time strategy)
102     fin_params = Input(shape=(fpd,), name="financial_model_params")
103     price = Input(shape=(m,), name="price_0")
104     strategy_input = Input(shape=(m,), name="information_set_0")
105     FV_factor = 1 + (r-q)*dt  # np.exp((r-q)*dt)
106     inputs = [fin_params, price, strategy_input]
107     wealth = Lambda(lambda p: 0*p+initial_wealth, name="wealth_0")(price)  # TRICK
108     for j in range(N):
109         if strategy_type == "simple":  # standard FNN
110             helper1 = strategy_input
111         elif strategy_type == "recurrent":  # (semi-)recurrent NN
```

```
            if j == 0:
                strategy = Lambda(lambda x: x*0.0)(price)  # Strategy at t = -1 should be 0
            helper1 = Concatenate()([strategy_input, strategy])
        helper2 = Concatenate(name="add_fin_params_"+str(j))([helper1, fin_params])
        strategy_layer = StrategyLayer(n=n, m=m, nodes=nodes, use_BN=use_BN, kernel_initializer=kernel_initializer, bias_initializer=bias_initializer, act_hidden=act_hidden, act_output=act_output, delta_constraint=delta_constraint, transaction=j)
        strategy_helper = strategy_layer(helper2)
        if j == 0:
            delta_strategy = strategy_helper  # strategy_-1 is set to 0
        else:  # delta_strategy = strategy_{t+1} - strategy_t
            delta_strategy = Subtract(name="diff_strategy_"+str(j))([strategy_helper, strategy])
        if cost_type == "proportional":  # Proportional transaction cost
            absolute_changes = Lambda(lambda x: K_back.abs(x), name="absolute_change_"+str(j))(delta_strategy)
            costs = Dot(axes=1)([absolute_changes, price])
            costs = Lambda(lambda x: epsilon*x, name="cost_"+str(j))(costs)
        elif cost_type == "constant":
            costs = Lambda(lambda x: epsilon+x*0.0)(price)
        if j > 0:
            wealth = Lambda(lambda x: x*FV_factor)(wealth)
        wealth = Subtract(name="wealth-cost_"+str(j))([wealth, costs])
        # Money invested in the bank account:  w_{t+1} = w_t - delta_strategy*price_t
        mult = Dot(axes=1)([delta_strategy, price])
        wealth = Subtract(name="wealth-risky_asset_"+str(j))([wealth, mult])
        # New input variables
        price = Input(shape=(m,), name="price_"+str(j+1))
        strategy_input = Input(shape=(m,), name="information_set_"+str(j+1))
        strategy = strategy_helper
        if j != N-1:
            inputs += [price, strategy_input]
        else:  # j == N-1
            inputs += [price]
    # Final time-step
    wealth = Lambda(lambda x: x*FV_factor)(wealth)
    if final_cost:
        if cost_type == "proportional":  # proportional transaction cost
            absolute_changes = Lambda(lambda x: K_back.abs(x), name="absolute_change_"+str(j+1))(strategy)
            costs = Dot(axes=1)([absolute_changes, price])
            costs = Lambda(lambda x: epsilon*x, name="cost_"+str(j+1))(costs)
        elif cost_type == "constant":
            costs = Lambda(lambda x: epsilon+x*0.0)(price)
        wealth = Subtract(name="wealth-cost_"+str(j+1))([wealth, costs])
    # Bank account for the final period
    mult = Dot(axes=1)([strategy, price])  # delta_strategy = strategy_t
    wealth = Add()([wealth, mult])  # now we add (instead of subtracting)
    payoff = Input(shape=(m,), name="payoff_final_t")  # add payoff at time T
    inputs += [payoff]  # final input to be inserted
    wealth = Subtract(name="wealth_"+str(j+1))([wealth, payoff])  # remove payoff (=liability)
    return Model(inputs=inputs, outputs=wealth)


def Delta_SubModel(dhen_obj, model=None, days_from_today=None, **kwargs):
    if dhen_obj.nn_params['strategy_type'] == "simple":
        inputs = [Input(dhen_obj.proc_dim, ), Input(dhen_obj.fpd, )]
```

```
164         intermediate_inputs = Concatenate()(inputs)
165     elif dhen_obj.nn_params['strategy_type'] == "recurrent":
166         inputs = [Input(dhen_obj.proc_dim, ), Input(dhen_obj.proc_dim, ), Input(
    dhen_obj.fpd, )]
167         intermediate_inputs = Concatenate()(inputs)
168     outputs = model.get_layer("delta_"+str(days_from_today))(intermediate_inputs)
169     return Model(inputs, outputs)
170
171
172 def prepare_input_delta_NN(dhen_obj, derivative, days_from_today,
    strat_input_range, fin_params):
173     assert dhen_obj.obs_params['dt'] == derivative.dt, "[prepare_input_delta_NN]
    dt needs to be the same"
174     if dhen_obj.nn_params['strategy_type'] == 'simple':
175         input_submodule = [strat_input_range, fin_params]
176     elif dhen_obj.nn_params['strategy_type'] == 'recurrent':  # TRICK (SEE BELOW)
    - otherwise need loop over days
177         # TRICK: compute the delta at previous time using the analytical formula
178         S_range_minus1 = dhen_obj.obtain_S_range(days_from_today=days_from_today)
179         tau_minus1 = (dhen_obj.nn_params['N']-(days_from_today-1))*derivative.dt
180         d1_minus1 = EuropeanCall.d1_func(S_range_minus1, derivative.K, derivative.
    r, derivative.q, derivative.imp_vola, tau_minus1)
181         model_delta_minus1 = EuropeanCall.delta_func(d1_minus1, derivative.q,
    tau_minus1)
182         input_submodule = [strat_input_range, model_delta_minus1, fin_params]
183     return input_submodule
```

Listing 4.2: Code for definition of the neural network class and function used for deep hedging. Parts of the function `Deep_Hedging_Model` are inspired to the code of Yu Man Tam `https://github.com/YuMan-Tam/deep-hedging`.

The heart of the code resides in Listing 4.3, where the class `DH_EN` is implemented. The goal of this class is to keep under one umbrella the different aspects involved in model free deep hedging. In this case, with an instance of this class we can create a deep hedging neural network, using the method `create_NN`, create a sample of simulations from a specified financial model, e.g. Black-Scholes (geometric Brownian motion), Merton (geometric Brownian motion with jumps) or Bates, with the method `obtain_paths`, using these simulations to train the network previously created, thanks to `compile_and_train_NN`.
Eventually, the function `get_mixed_prices_deltas` is able to combine an instance of the class `DH_EN` and the density π obtained as shown in Listing 4.1 to actually test model free deep hedging. In this function, the argument `F_trajectories` stands for the trajectories linked to Bates model, that is supposed to be the observed model, `derivative` is any derivative, e.g. an European call option, and `dhen_obj` an instance of `DH_EN`.

```
1  from tensorflow.keras.models import load_model
2  from tensorflow.keras.optimizers import Adam
3  import tensorflow.keras.backend as K_back
4  import tensorflow as tf
5  import numpy as np
6  from sklearn.model_selection import train_test_split
7  from merton_model import Merton
8  from LaTex_NN import Deep_Hedging_Model, Delta_SubModel, prepare_input_delta_NN
9  from bates import BatesTDJ, jump_params_length
10 from tqdm import tqdm
11 import os
12
13 def train_test_split_for_list(data=None, test_size=None):
14     # Split simulated data into training and testing sample
15     x_train = []
16     x_test = []
17     for x in data:
```

```
18          tmp_x_train, tmp_x_test = train_test_split(x, test_size=test_size, shuffle
    =False)
19          x_train += [tmp_x_train]
20          x_test += [tmp_x_test]
21      return x_train, x_test
22
23  def create_input_strategy(input_type, S, init_price, axis=1):
24      # Choose from: "S", "log_S", "normalized_log_S" (by S0)
25      if input_type == "S":
26          output = np.stack(S, axis=axis)
27      elif input_type == "log_S":
28          output = np.stack((np.log(S)), axis=axis)
29      elif input_type == "normalized_log_S":
30          output = np.stack((np.log(S/init_price)), axis=axis)
31      return output
32
33  def create_total_input(process, strategy_input, financial_parameters, payoff,
    time_steps):
34      # Put together financial_parameters, underlying, strategy_input and final-time
     payoff
35      x_all = [financial_parameters]
36      for j in range(time_steps):
37          x_all += [process[j, :, None]]
38          x_all += [strategy_input[j, :, None]]
39      x_all += [process[time_steps, :, None]]
40      x_all += [payoff[:, None]]
41      return x_all
42
43  def mean_pred(y_pred):
44      return K_back.mean(y_pred, axis=-1)
45
46  def mse(wealth=None):
47      return K_back.mean(K_back.square(wealth), axis=-1)
48
49  def set_seeds(seed):
50      np.random.seed(seed)
51      tf.random.set_seed(seed)
52
53  def log_returns_GBM(vola, r, q, n_samples, m, N, dt):
54      init_time = np.zeros((n_samples, m))
55      return np.stack(([init_time] + [(r-q-vola**2/2)*dt + vola*np.random.normal(0,
    np.sqrt(dt), (n_samples, m)) for _ in range(N)]))
56
57  def proc_from_log_returns(init_price, log_returns):
58      return init_price*np.exp(np.cumsum(log_returns, axis=0))
59
60  def exclude_fin_model(fin_model_list):
61      exclude = np.random.choice(len(fin_model_list), 1)[0]
62      test_fin_model = fin_model_list.pop(exclude)
63      return fin_model_list, test_fin_model
64
65
66  class DH_EN:
67      def __init__(self, str_model, fin_model_list, obs_params, nn_params,
    payoff_func, n_samples=10**4, time_steps=20, proc_dim=1, **kwargs):
68          """
69          Class used to generate a Neural Network (NN) model that is intimately
    connected with a precise financial model,
70          where all parameters of the financial part are specified at the beginning.
71          :param str_model: string denoting the financial model ---> used for
    internal dispatch
72          :param fin_model_list: list of parameters (as dictionaries) needed for the
```

```
     financial model
        :param obs_params: observable financial quantities (S0, r, q, ...)
        :param nn_params: parameters needed for NN model
        :param payoff_func: payoff function, i.e. liability to hedge
        :param n_samples: number of samples of the financil model to simulate
        :param time_steps: number of time steps for simulations (at the moment, it is also the number of transactions)
        :param proc_dim: process dimension of the financial model
        """
        assert all(k in nn_params for k in ('N', 'n', 'strategy_type')), 'Specify all elements of NN!'
        assert all(k in obs_params for k in ('S0', 'dt', 'r', 'q')), 'Specify all observables financial quantities!'
        self.str_model = str_model
        if 'test_fin_model' in kwargs.keys():
            self.test_fin_model = kwargs.get('test_fin_model', {'sigma': 0.2})
            self.fin_model_list = fin_model_list
        else:
            self.fin_model_list, self.test_fin_model = exclude_fin_model(fin_model_list)
        self.n_models = len(self.fin_model_list)
        self.fpd = len(self.fin_model_list[0])  # ASSUME all models belong to the same family
        self.obs_params = obs_params
        self.nn_params = nn_params
        self.payoff_func = payoff_func
        self.n_samples = n_samples
        self.time_steps = time_steps
        self.proc_dim = proc_dim
        self.NN = None
        self.db = None
        self._dispatch = {'GBM': self._generate_GBM, 'Merton': self._generate_Merton, 'Bates': self._generate_Bates}
        self._paths = None

    def obtain_paths(self, **kwargs):
        if 'obs_dict_list' in kwargs.keys():
            obs_dict_list = kwargs['obs_dict_list']
            if self._paths is None:
                self._paths = np.concatenate([self.generate_sample(fin_model, **obs_dict_list[k]) for k, fin_model in enumerate(self.fin_model_list)], axis=1)
        else:
            if self._paths is None:
                self._paths = np.concatenate([self.generate_sample(fin_model) for fin_model in self.fin_model_list], axis=1)

    def _generate_GBM(self, fin_model_params, **kwargs):
        assert 'sigma' in fin_model_params, 'sigma is needed for GBM!'
        sigma = fin_model_params['sigma']
        S0 = kwargs.get('S0', self.obs_params['S0'])
        dt = self.obs_params['dt']
        r = self.obs_params.get('r', 0)
        q = self.obs_params.get('q', 0)
        log_returns = log_returns_GBM(sigma, r, q, self.n_samples, self.proc_dim, self.time_steps, dt)
        return proc_from_log_returns(S0, log_returns)

    def _generate_Merton(self, fin_model_params, **kwargs):
        assert 'sigma' in fin_model_params, 'sigma is needed for Merton!'
        assert 'lambd' in fin_model_params, 'lambda is needed for Merton!'
        sigma = fin_model_params['sigma']
        lambd = fin_model_params['lambd']
```

```
125         nu = fin_model_params.get('nu', 0)
126         delta = fin_model_params.get('delta', 0)
127         S0 = kwargs.get('S0', self.obs_params['S0'])
128         dt = self.obs_params['dt']
129         r = self.obs_params.get('r', 0)
130         q = self.obs_params.get('q', 0)
131         log_returns = Merton.log_increments(r, q, sigma, lambd, nu, delta, dt,
    self.n_samples, self.time_steps, self.proc_dim)
132         return proc_from_log_returns(S0, log_returns)
133 
134     def _generate_Bates(self, fin_model_params, **kwargs):
135         assert 'sigma' in fin_model_params, 'sigma is needed for Bates!'
136         assert 'lambd' in fin_model_params, 'lambda is needed for Bates!'
137         sigma = fin_model_params['sigma']  # for Bates this is the vol of vol
138         lambd = fin_model_params['lambd']
139         nu = fin_model_params.get('nu', 0)
140         delta = fin_model_params.get('delta', 0)
141         kappa = fin_model_params.get('kappa', 0)
142         theta = fin_model_params.get('theta', 0)
143         rho = fin_model_params.get('rho', 0)
144         V0 = fin_model_params.get('V0', 0.1)
145         S0 = kwargs.get('S0', self.obs_params['S0'])
146         dt = self.obs_params['dt']
147         r = self.obs_params.get('r', 0)
148         q = self.obs_params.get('q', 0)
149         log_returns = BatesTDJ.log_increments(S0, V0, r, q, kappa, theta, sigma,
    rho, lambd, nu, delta, dt, self.n_samples, self.time_steps, self.proc_dim)
150         return proc_from_log_returns(S0, log_returns)
151 
152     def generate_sample(self, fin_model_dict, **kwargs):
153         return self._dispatch[self.str_model](fin_model_dict, **kwargs)
154 
155     def create_NN(self, **kwargs):
156         path = kwargs.get('path', '')
157         if 'nn_models' in path:
158             self.NN = load_model(path)
159             self.NN.trained = True
160         else:
161             nodes = self.nn_params.get('nodes', int(16))
162             r = self.obs_params.get('r', 0.0)
163             q = self.obs_params.get('q', 0.0)
164             wealth0 = self.nn_params.get('initial_wealth',0.0)
165             final_cost = self.nn_params.get('final_cost', True)
166             use_BN = self.nn_params.get('use_BN', True)
167             act_hidden = self.nn_params.get('act_hidden', 'leaky_relu')
168             act_output = self.nn_params.get('act_output', 'sigmoid')
169             delta_constraint = self.nn_params.get('delta_constraint', None)
170             cost_type = self.nn_params.get('cost_type', 'proportional')
171             epsilon = self.nn_params.get('epsilon', 0.0)
172             self.NN = Deep_Hedging_Model(N=self.nn_params['N'], n=self.nn_params['
    n'], fpd=self.fpd, m=self.proc_dim, nodes=nodes, r=r, q=q, dt=self.obs_params[
    'dt'], initial_wealth=wealth0, final_cost=final_cost, strategy_type=self.
    nn_params['strategy_type'], use_BN=use_BN, act_hidden=act_hidden, act_output=
    act_output, delta_constraint=delta_constraint, cost_type=cost_type, epsilon=
    epsilon, **kwargs)
173             self.NN.trained = False
174         return self.NN
175 
176     def compile_and_train_NN(self, epochs: int, batch_size: int, lr, loss, metric=
    None, metric_name: str = '',
177                              callbacks: list = None, save=False, **kwargs):
178         assert self.NN is not None, 'You need to create NN before!'
```

```
        assert self.db is not None and 'x_train' not in kwargs, 'Create or provide a complete database before!'
        if self.NN.trained:
            print("Model already trained"); return None
        if metric is not None and metric_name == '':
            AssertionError('If metric is given, provide "metric_name" also!')
        optimizer = kwargs.get('optimizer', Adam(learning_rate=lr))
        self.NN.add_loss(loss)
        self.NN.add_metric(metric, name=metric_name)
        self.NN.compile(optimizer=optimizer)
        x_train = kwargs.get('x_train', self.db['x_train'])
        # print("x_train", x_train)
        x_test = kwargs.get('x_test', self.db['x_test'])
        self.NN.fit(x=[x_train], batch_size=batch_size, epochs=epochs, validation_data=[x_test], callbacks=callbacks, verbose=1)
        path = kwargs.get('path', 'nn_models/NN')
        if save and not os.path.exists(os.path.join(os.getcwd(), path)):
            self.NN.trained = True
            self._save_NN(path)

    def _save_NN(self, path):
        self.NN.save(path)

    @staticmethod
    def process_data(payoff_function, S_0, trajectories, strat_input_info):
        payoff_T = payoff_function(trajectories)  # Payoff of the option
        prices = np.stack(trajectories, axis=1)  # Trading set
        strat_input = create_input_strategy(input_type=strat_input_info, S=trajectories, init_price=S_0)
        return payoff_T, prices, strat_input

    @staticmethod
    def melt_Bates_params(fin_params_dict_list):
        return [np.concatenate((np.array((fin_params_dict['sigma'], fin_params_dict['lambd'], fin_params_dict['theta'], fin_params_dict['kappa'], fin_params_dict['rho'], fin_params_dict['V0'])),
                                fin_params_dict['nu'], fin_params_dict['delta']))
    for fin_params_dict
                in fin_params_dict_list]


    def prepare_sample_for_NN_input(self, strat_input_info, **kwargs):
        self.obtain_paths(**kwargs)
        S = np.squeeze(self._paths).T  # now one column = one instant in time and one row = one simulation
        if 'obs_dict_list' in kwargs.keys():
            obs_dict_list = kwargs.get('obs_dict_list', None)
            assert self.n_models == len(obs_dict_list), "[prepare_sample_for_NN_input] lenght problem"
            S0_vec = np.array([obs_dict_list[k]['S0'] for k in range(self.n_models) for _ in range(self.n_samples)])
        else:
            S0_vec = np.tile(self.obs_params['S0'], self.n_samples*self.n_models)
        S0_mat = np.tile(S0_vec.reshape(-1, 1), self.time_steps+1)
        payoff_T, prices, strat_input = DH_EN.process_data(self.payoff_func, S0_mat, S, strat_input_info)
        if self.str_model == "Bates":
            fin_model_array = DH_EN.melt_Bates_params(self.fin_model_list)
            financial_parameters = np.tile(fin_model_array, (self.n_samples, 1))
        else:
            financial_parameters = np.concatenate([np.tile(np.array([*fin_model_dict.values()]), (self.n_samples,1)) for fin_model_dict in self.
```

```
    fin_model_list], axis=0)
230         x_train = create_total_input(process=prices, strategy_input=strat_input,
    financial_parameters=financial_parameters, payoff=payoff_T, time_steps=self.
    nn_params['N'])
231         # Generation of test data
232         S_test = np.squeeze(self.generate_sample(self.test_fin_model)).T
233         payoff_T_test, prices_test, strat_input_test = DH_EN.process_data(self.
    payoff_func, self.obs_params['S0'], S_test, strat_input_info)
234         if self.str_model == "Bates":
235             fin_model_array_test = DH_EN.melt_Bates_params([self.test_fin_model])
236             fin_params_test = np.tile(fin_model_array_test, (self.n_samples, 1))
237         else:
238             fin_params_test = np.tile(np.array([*self.test_fin_model.values()]), (
    self.n_samples, 1))
239         x_test = create_total_input(process=prices_test, strategy_input=
    strat_input_test, financial_parameters=fin_params_test, payoff=payoff_T_test,
    time_steps=self.nn_params['N'])
240         db = {}
241         db['x_train'], db['x_test'] = x_train, x_test
242         db['S_train'], db['S_test'] = S, S_test
243         db['option_payoff_train'], db['option_payoff_test'] = payoff_T,
    payoff_T_test
244         self.db = db
245         return db
246
247     def evaluate_NN(self, batch_size_evaluate, **kwargs):
248         assert self.NN is not None, 'Need to create (and train) a NN before'
249         x_test = kwargs.get('x_test', self.db['x_test'])
250         verbose = kwargs.get('verbose', 1)
251         eval_result = self.NN.evaluate(x_test, batch_size=batch_size_evaluate,
    verbose=verbose)
252         if verbose == 1:
253             print('Loss function on x_test', eval_result)
254         return eval_result
255
256     def obtain_S_range(self, days_from_today, intervals=101, **kwargs):
257         S_test = kwargs.get('S_test', self.db['S_test'])
258         min_S = S_test[:, days_from_today].min()
259         max_S = S_test[:, days_from_today].max()
260         return np.linspace(min_S, max_S, intervals)
261
262
263 def create_comb_simulation(comb, dhen_obj, icond):
264     if dhen_obj.str_model == "GBM":  # Simulate GBM from initial conditions
265         log_ret_comb = log_returns_GBM(comb, icond['r'], icond['q'], dhen_obj.
    n_samples, m=1, N=dhen_obj.time_steps, dt=icond['dt'])
266         fin_params_comb = np.tile(np.array([comb]), (dhen_obj.n_samples, 1))
267     elif dhen_obj.str_model == "Merton":
268         log_ret_comb = Merton.log_increments(icond['r'], icond['q'], comb, icond['
    lambd'], icond['nu'], icond['delta'], icond['dt'], dhen_obj.n_samples,
    dhen_obj.time_steps, m=1)
269         fin_params_comb = np.tile(np.array([comb, icond['lambd'], icond['nu'],
    icond['delta']]), (dhen_obj.n_samples,1))
270     return log_ret_comb, fin_params_comb
271
272
273 def input_NN_list(post_density, icond, dhen_obj, strat_input_type):
274     x_list, price_list, strat_input_list, fin_params_list = [], [], [], []
275     for j, comb in enumerate(tqdm(post_density['x'][:-1], desc="Computing input
    for DH NN:")):
276         log_ret_comb, fin_params = create_comb_simulation(comb, dhen_obj, icond)
277         S_test_comb = np.squeeze(proc_from_log_returns(icond['S0'], log_ret_comb))
```

```
    .T
278         payoff_T, prices, strat_input = DH_EN.process_data(dhen_obj.payoff_func,
    icond['S0'], S_test_comb, strat_input_type)
279         x_list += [create_total_input(process=prices, strategy_input=strat_input,
    financial_parameters=fin_params, payoff=payoff_T, time_steps=dhen_obj.
    time_steps)]
280         price_list += [prices]
281         strat_input_list += [strat_input]
282         fin_params_list += [fin_params]
283     inp_dict = {"x_list": x_list, "price_list": price_list, "strat_input_list":
    strat_input_list, "fin_params_list": fin_params_list}
284     return inp_dict
285
286
287 def get_mixed_prices_deltas(post_density, icond, dhen_obj, F_trajectories,
    derivative, strat_input_type, **kwargs):
288     inp_dict = kwargs.get('inp_dict', {})
289     if not inp_dict:
290         inp_dict = input_NN_list(post_density, icond, dhen_obj, strat_input_type)
291     x_l, price_l = inp_dict["x_list"], inp_dict["price_list"]
292     strat_input_l, fin_params_l = inp_dict["strat_input_list"], inp_dict["
    fin_params_list"]
293     strat_input = create_input_strategy(input_type=strat_input_type, S=
    F_trajectories, init_price=icond['S0'])
294     if 'dhen_F' in kwargs.keys():
295         dhen_F = kwargs.get('dhen_F', None)
296         payoff_T_F, prices_F, strat_input_F = DH_EN.process_data(dhen_F.
    payoff_func, icond['S0'], F_trajectories, strat_input_type)
297         fin_params_F = dhen_F.test_fin_model.copy()
298         fin_params_F['sigma'] = icond['sigma']
299         fin_params_F = DH_EN.melt_Bates_params([fin_params_F])
300         fin_params_F = np.tile(fin_params_F, (dhen_F.n_samples, 1))
301         x_F = create_total_input(process=prices_F, strategy_input=strat_input_F,
    financial_parameters=fin_params_F, payoff=payoff_T_F, time_steps=dhen_F.
    time_steps)
302
303     delta_derivatives = []
304     for j, comb in enumerate(tqdm(post_density['x'][:-1], desc="Computing deltas
    from DH NN:")):
305         deltas = []
306         for day in range(dhen_obj.time_steps):
307             input_sub_model = prepare_input_delta_NN(dhen_obj, derivative, day,
    strat_input[day].reshape(-1, 1), fin_params_l[j])
308             sub_model = Delta_SubModel(dhen_obj=dhen_obj, model=dhen_obj.NN,
    days_from_today=day)
309             deltas += [np.squeeze(sub_model(input_sub_model))]
310         delta_derivatives.append(deltas)
311     mix_deltas = np.squeeze(delta_derivatives)
312
313     price_derivative_l, price_derivative_an_l = [], []
314     for j, comb in enumerate(tqdm(post_density['x'][:-1], desc="Computing prices
    as in DH NN:")):
315         price_derivative_an_l += [Merton.Merton_pricer(icond['S0'], derivative.K,
    icond['r'], icond['q'], comb, icond['lambd'], icond['nu'], icond['delta'],
    derivative.T[0, 0])]
316         price_derivative_l += [-np.squeeze(dhen_obj.NN(x_l[j]))]
317     price_derivative = np.squeeze(price_derivative_l)
318     for j, comb_y in enumerate(post_density['y']):
319         mix_deltas[j] = mix_deltas[j] * comb_y
320         price_derivative[j] = price_derivative[j] * comb_y
321         price_derivative_an_l[j] = price_derivative_an_l[j] * comb_y
322     # Now I have to sum up over different comb, but for the same days
```

```
323     mix_prices = np.sum(price_derivative, axis=0)  # returns array of dimension "
        n_sample"
324     mix_deltas = np.sum(mix_deltas, axis=0)  # returns array of dimension "(N,
        n_sample)"
325     return mix_prices, mix_deltas, price_derivative, delta_derivatives
326
327
328 # Read initial condition
329 def convert_list2dict(p_list):
330     mjs_pos, sjs_pos = jump_params_length([p_list])
331     return {'S0': p_list[0], 'V0': p_list[1], 'r': p_list[2], 'q': p_list[3], '
        kappa': p_list[4], 'theta': p_list[5], 'sigma': p_list[6], 'rho': p_list[7], '
        lambd': p_list[8], 'nu': p_list[mjs_pos], 'delta': p_list[sjs_pos]}
```

Listing 4.3: Code for definition of the class that coordinates the deep hedging approach with the 'Estimate Nothing' parameter distribution.

Bibliography

[Aba+15] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu and Xiaoqiang Zheng. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: `https://www.tensorflow.org/` (cit. on pages 82, 122, 130).

[AT02a] Carlo Acerbi and Dirk Tasche. 'Expected Shortfall: A Natural Coherent Alternative to Value at Risk'. In: *Economic Notes by Banca Monte dei Paschi di Siena SpA* 31.2 (Dec. 2002), pp. 379–388. DOI: `10.1111/1468-0300.00091` (cit. on page 115).

[AT02b] Carlo Acerbi and Dirk Tasche. 'On the coherence of expected shortfall'. In: *Journal of Banking & Finance* 26.7 (July 2002), pp. 1487–1503. ISSN: 0378-4266. DOI: `10.1016/S0378-4266(02)00283-2`. URL: `https://www.sciencedirect.com/science/article/pii/S0378426602002832` (cit. on page 115).

[ALS19a] Zeyuan Allen-Zhu, Yuanzhi Li and Zhao Song. 'A Convergence Theory for Deep Learning via Over-Parameterization'. In: *Proceedings of the 36th International Conference on Machine Learning*. Ed. by Kamalika Chaudhuri and Ruslan Salakhutdinov. Vol. 97. Proceedings of Machine Learning Research. PMLR, Sept. 2019, pp. 242–252. URL: `https://proceedings.mlr.press/v97/allen-zhu19a.html` (cit. on page 48).

[ALS19b] Zeyuan Allen-Zhu, Yuanzhi Li and Zhao Song. 'On the Convergence Rate of Training Recurrent Neural Networks'. In: *Proceedings of the 33rd International Conference on Neural Information Processing Systems*. Red Hook, NY, USA: Curran Associates Inc., 2019 (cit. on page 49).

[AB+03] Ferdinando Ametrano, Luigi Ballabio et al. *QuantLib - a free/open-source library for quantitative finance*. 2003. URL: `http://quantlib.org/` (cit. on pages 81, 122).

[AS94] Jean-Pascal Ansel and Christophe Stricker. 'Couverture des actifs contingents et prix maximum'. fr. In: *Annales de l'I.H.P. Probabilités et statistiques* 30.2 (1994), pp. 303–315. URL: `http://www.numdam.org/item/AIHPB_1994__30_2_303_0/` (cit. on page 111).

[Aro+19] Sanjeev Arora, Nadav Cohen, Wei Hu and Yuping Luo. 'Implicit Regularization in Deep Matrix Factorization'. In: *Advances in Neural Information Processing Systems*. Ed. by H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox and R. Garnett. Vol. 32. Curran Associates, Inc., 2019. URL: https://proceedings.neurips.cc/paper/2019/file/c0c783b5fc0d7d808f1d14a6e9c8280d-Paper.pdf (cit. on page 52).

[Art+99] Philippe Artzner, Freddy Delbaen, Jean-Marc Eber and David Heath. 'Coherent Measures of Risk'. In: *Mathematical Finance* 9.3 (July 1999), pp. 203–228. DOI: 10.1111/1467-9965.00068 (cit. on pages 114, 115).

[Bac22] Francis Bach. 'Learning theory from first principles - draft'. In: (Feb. 2022). URL: https://www.di.ens.fr/~fbach/ltfp_book.pdf (cit. on pages 19, 97).

[Bar+19] Peter L. Bartlett, Nick Harvey, Christopher Liaw and Abbas Mehrabian. 'Nearly-tight VC-dimension and Pseudodimension Bounds for Piecewise Linear Neural Networks'. In: *Journal of Machine Learning Research* 20.63 (Apr. 2019), pp. 1–17. URL: http://jmlr.org/papers/v20/17-612.html (cit. on page 42).

[Bat88] David S. Bates. *Pricing Options Under Jump-Diffusion Processes*. Rodney L. White Center for Financial Research Working Papers 37-88. Wharton School Rodney L. White Center for Financial Research, Oct. 1988. URL: https://www.biz.uiowa.edu/faculty/dbates/papers/chapter3.pdf (cit. on page 69).

[BP63] Thomas Bayes and Richard Price. 'LII. An essay towards solving a problem in the doctrine of chances. By the late Rev. Mr. Bayes, F. R. S. communicated by Mr. Price, in a letter to John Canton, A. M. F. R. S'. In: *Philosophical Transactions of the Royal Society of London* 53 (1763), pp. 370–418. DOI: 10.1098/rstl.1763.0053. URL: https://royalsocietypublishing.org/doi/abs/10.1098/rstl.1763.0053 (cit. on page 100).

[BPS22] Mathias Beiglböck, Gudmund Pammer and Walter Schachermayer. 'From Bachelier to Dupire via optimal transport'. en. In: *Finance and Stochastics* 26 (Jan. 2022), pp. 59–84. ISSN: 0949-2984. DOI: 10.3929/ethz-b-000522579 (cit. on page 16).

[Bel+19] Mikhail Belkin, Daniel Hsu, Siyuan Ma and Soumik Mandal. 'Reconciling modern machine-learning practice and the classical bias-variance trade-off'. In: *Proceedings of the National Academy of Sciences* 116.32 (2019), pp. 15849–15854. DOI: 10.1073/pnas.1903070116. eprint: https://www.pnas.org/doi/pdf/10.1073/pnas.1903070116. URL: https://www.pnas.org/doi/abs/10.1073/pnas.1903070116 (cit. on pages 50, 51, 53).

[BJ00] Nadine Bellamy and Monique Jeanblanc. 'Incompleteness of markets driven by a mixed diffusion'. In: *Finance and Stochastics* 4.2 (Feb. 2000), pp. 209–222. DOI: 10.1007/s007800050012 (cit. on page 111).

[BF02] Fabio Bellini and Marco Frittelli. 'On the Existence of Minimax Martingale Measures'. In: *Mathematical Finance* 12.1 (Jan. 2002), pp. 1–21. DOI: 10.1111/1467-9965.00001 (cit. on page 113).

[BC12] Amel Bentata and Rama Cont. 'Mimicking the marginal distributions of a semi-martingale'. In: *arXiv: Probability* (2012) (cit. on page 17).

[Ber+21] Julius Berner, Philipp Grohs, Gitta Kutyniok and Philipp Petersen. *The Modern Mathematics of Deep Learning*. 2021. DOI: 10.48550/ARXIV.2105.04026. URL: https://arxiv.org/abs/2105.04026 (cit. on page 31).

[Bia10] Francesca Biagini. 'Second Fundamental Theorem of Asset Pricing'. In: *Encyclopedia of Quantitative Finance* (May 2010). Ed. by R. Cont, pp. 1623–1628. DOI: 10.1002/9780470061602.eqf04008 (cit. on page 109).

[BS14] Monica Bianchini and Franco Scarselli. 'On the Complexity of Neural Network Classifiers: A Comparison Between Shallow and Deep Architectures'. In: 25.8 (2014), pp. 1553–1565. DOI: 10.1109/TNNLS.2013.2293637 (cit. on page 41).

[BS73] Fischer Black and Myron Scholes. 'The Pricing of Options and Corporate Liabilities'. In: *Journal of Political Economy* 81.3 (1973), pp. 637–654. ISSN: 00223808, 1537534X. URL: http://www.jstor.org/stable/1831029 (cit. on pages 14, 73, 99, 107, 122).

[Böl+19] Helmut Bölcskei, Philipp Grohs, Gitta Kutyniok and Philipp Petersen. 'Optimal Approximation with Sparsely Connected Deep Neural Networks'. In: *SIAM J. Math. Data Sci.* 1.1 (2019), pp. 8–45. URL: https://doi.org/10.1137/18M118709X (cit. on pages 42, 43, 46).

[BN15] Bruno Bouchard and Marcel Nutz. 'Arbitrage and duality in nondominated discrete-time models'. In: *The Annals of Applied Probability* 25.2 (Apr. 2015), pp. 823–859. DOI: 10.1214/14-AAP1011 (cit. on pages 98, 99).

[BV04] Stephen Poythress Boyd and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, Mar. 2004. ISBN: 9780521833783. URL: https://web.stanford.edu/%5C%7Eboyd/cvxbook/ (cit. on page 24).

[BL78] Douglas T. Breeden and Robert H. Litzenberger. 'Prices of State-Contingent Claims Implicit in Option Prices'. In: *The Journal of Business* 51.4 (Oct. 1978), pp. 621–651. ISSN: 00219398, 15375374. URL: http://www.jstor.org/stable/2352653 (cit. on pages 16, 76).

[Bue+19] Hans Buehler, Lukas Gonon, Josef Teichmann and Ben Wood. 'Deep hedging'. In: *Quantitative Finance* 19.8 (Feb. 2019), pp. 1271–1291. DOI: 10.1080/14697688.2019.1571683. URL: https://doi.org/10.1080/14697688.2019.1571683 (cit. on pages 8, 115–121, 126).

[BGR02] Oliver F. Bunnin, Yike Guo and Yuhe Ren. 'Option pricing under model and parameter uncertainty using predictive densities'. In: *Statistics and Computing* 12 (Jan. 2002), pp. 37–44. DOI: 10.1023/A:1013116204872 (cit. on page 100).

[Bur+19] Matteo Burzoni, Marco Frittelli, Zhaoxu Hou, Marco Maggis and Jan Obłój. 'Pointwise Arbitrage Pricing Theory in Discrete Time'. In: *Mathematics of Operations Research* 44.3 (Apr. 2019), pp. 1034–1057. DOI: 10.1287/moor.2018.0956. URL: https://doi.org/10.1287/moor.2018.0956 (cit. on pages 98, 99).

[Car07] René Carmona. 'HJM: A unified approach to dynamic models for fixed income, credit and equity markets'. English (US). In: *Paris-Princeton Lectures on Mathematical Finance 2004*. Lecture Notes in Mathematics. Germany: Springer Verlag, 2007, pp. 1–50. ISBN: 3540733264. DOI: 10.1007/978-3-540-73327-0_1 (cit. on pages 10, 12, 13, 16).

[CMN17] René Carmona, Yi Ma and Sergey Nadtochiy. 'Simulation of Implied Volatility Surfaces via Tangent Lévy Models'. In: *SIAM Journal on Financial Mathematics* 8.1 (2017), pp. 171–213. DOI: 10.1137/15M1015510. eprint: https://doi.org/10.1137/15M1015510. URL: https://doi.org/10.1137/15M1015510 (cit. on pages 12, 17, 58, 87).

[CN09] René Carmona and Sergey Nadtochiy. 'Local volatility dynamic models'. In: *Finance and Stochastics* 13 (2009), pp. 1–48 (cit. on page 16).

[CN11] René Carmona and Sergey Nadtochiy. 'Tangent Models as a mathematical framework for Dynamic Calibration'. In: *Finance at Fields*. Ed. by Matheus R. Grasselli and Lane P. Hughston. World Scientific Publishing Co. Pte. Ltd., 2011. Chap. 6, pp. 151–179. DOI: 10.1142/S0219024911006280 (cit. on pages 6, 12, 17, 18).

[CN12] René Carmona and Sergey Nadtochiy. 'Tangent Lévy market models'. In: *Finance and Stochastics* 16.1 (Jan. 2012), pp. 63–104. ISSN: 0949-2984 (cit. on pages 6, 12, 13, 17, 18, 61, 67, 71).

[CT06a] René Carmona and Michael Tehranchi. 'Interest Rate Models: an Infinite Dimensional Stochastic Analysis Perspective'. In: Jan. 2006. ISBN: 978-3-540-27065-2. DOI: 10.1007/b138563 (cit. on pages 5, 12).

[CM99] Peter Carr and Dilip B. Madan. 'Option valuation using the fast Fourier transform'. In: *Journal of Computational Finance* 2 (1999), pp. 61–73 (cit. on pages 76, 77, 97).

[Cau47] Augustine-Louis Cauchy. 'Méthode générale pour la résolution des systèmes d'équations simultanées'. In: *C.R. Acad. Sci. Paris* 25 (Oct. 1847), pp. 536–538. URL: https://ci.nii.ac.jp/naid/10026863174/en/ (cit. on page 24).

[Che+18] Ricky T. Q. Chen, Yulia Rubanova, Jesse Bettencourt and David Duvenaud. 'Neural Ordinary Differential Equations'. In: *Proceedings of the 32nd International Conference on Neural Information Processing Systems*. NIPS'18. Montréal, Canada: Curran Associates Inc., 2018, pp. 6572–6583 (cit. on pages 47, 48).

[CDK05] Patrick Cheridito, Freddy Delbaen and Michael Kupper. 'Coherent and convex monetary risk measures for unbounded càdlàg processes'. In: *Finance and Stochastics* 9.3 (July 2005), pp. 369–387. ISSN: 0949-2984. DOI: 10.1007/s00780-004-0150-7 (cit. on page 114).

[Cho+15] François Chollet et al. *Keras*. https://github.com/fchollet/keras. 2015 (cit. on page 82).

[Coh+21] Alain-Sam Cohen, Rama Cont, Alain Rossier and Renyuan Xu. 'Scaling Properties of Deep Residual Networks'. In: *ArXiv* abs/2105.12245 (2021) (cit. on page 48).

[CT04] Rama Cont and Peter Tankov. 'Non-parametric calibration of jump–diffusion option pricing models'. In: *Journal of Computational Finance* 7 (2004), pp. 1–49 (cit. on pages 77, 78).

[CT06b] Rama Cont and Peter Tankov. 'Retrieving Lévy Processes from Option Prices: Regularization of an Ill-posed Inverse Problem'. In: *SIAM Journal on Control and Optimization* 45.1 (2006), pp. 1–25. DOI: 10.1137/040616267. eprint: https://doi.org/10.1137/040616267. URL: https://doi.org/10.1137/040616267 (cit. on page 78).

[CIR85] John C. Cox, Jonathan E. Ingersoll and Stephen A. Ross. 'A Theory of the Term Structure of Interest Rates'. In: *Econometrica* 53.2 (Mar. 1985), pp. 385–407. ISSN: 00129682, 14680262. DOI: 10.2307/1911242. URL: http://www.jstor.org/stable/1911242 (cit. on pages 10, 122).

[Cré03] Stephane Crépey. 'Calibration of the Local Volatility in a Generalized Black–Scholes Model Using Tikhonov Regularization'. In: *SIAM Journal on Mathematical Analysis* 34.5 (Apr. 2003), pp. 1183–1206. DOI: 10.1137/S0036141001400202. URL: https://doi.org/10.1137/S0036141001400202 (cit. on page 16).

[CKT20] Christa Cuchiero, Wahid Khosrawi and Josef Teichmann. 'A Generative Adversarial Network Approach to Calibration of Local Stochastic Volatility Models'. In: *Risks* 8.4 (2020). ISSN: 2227-9091. DOI: 10.3390/risks8040101. URL: https://www.mdpi.com/2227-9091/8/4/101 (cit. on pages 58, 59).

[CKT16] Christa Cuchiero, Irene Klein and Josef Teichmann. 'A new perspective on the fundamental theorem of asset pricing for large financial markets'. In: *Theory Probab. Appl.* 60.4 (2016), pp. 561–579. ISSN: 0040-585X (cit. on page 7).

[CLT15] Christa Cuchiero, Martin Larsson and Josef Teichmann. 'Deep neural networks, generic universal interpolation, and controlled ODEs'. en. Ithaca, NY, 2019-08-15 (cit. on page 48).

[Cyb89] George Cybenko. 'Approximation by superpositions of a sigmoidal function'. In: *Mathematics of Control, Signals, and Systems (MCSS)* 2.4 (Dec. 1989), pp. 303–314. ISSN: 0932-4194. DOI: 10.1007/BF02551274. URL: http://dx.doi.org/10.1007/BF02551274 (cit. on page 32).

[DZ14] Giuseppe Da Prato and Jerzy Zabczyk. *Stochastic Equations in Infinite Dimensions*. 2nd ed. Encyclopedia of Mathematics and its Applications. Cambridge University Press, 2014. DOI: 10.1017/CBO9781107295513 (cit. on pages 12, 74, 75).

[DP07] Marzia De Donno and Maurizio Pratelli. 'On a Lemma by Ansel and Stricker'. In: *Lecture Notes in Mathematics*. Vol. 1899. Séminaire de Probabilités XL. Springer, 2007, pp. 411–414 (cit. on page 111).

[DS94] Freddy Delbaen and Walter Schachermayer. 'A general version of the fundamental theorem of asset pricing.' In: *Mathematische Annalen* 300.3 (Sept. 1994), pp. 463–520. DOI: 10.1007/BF01450498 (cit. on pages 103, 110).

[Don93] David L. Donoho. 'Unconditional Bases Are Optimal Bases for Data Compression and for Statistical Estimation'. In: *Applied and Computational Harmonic Analysis* 1.1 (1993), pp. 100–115. ISSN: 1063-5203. DOI: https://doi.org/10.1006/acha.1993.1008. URL: https://www.sciencedirect.com/science/article/pii/S1063520383710080 (cit. on pages 43, 44).

[Dot78] L. Uri Dothan. 'On the term structure of interest rates'. In: *Journal of Financial Economics* 6.1 (Mar. 1978), pp. 59–69. ISSN: 0304-405X. DOI: 10.1016/0304-405X(78)90020-X. URL: https://www.sciencedirect.com/science/article/pii/0304405X7890020X (cit. on page 10).

[Doz16] Timothy Dozat. 'Incorporating Nesterov Momentum into Adam'. In: *Workshop track - ICLR 2016*. Feb. 2016 (cit. on page 30).

[Du+19] Simon Du, Jason Lee, Haochuan Li, Liwei Wang and Xiyu Zhai. 'Gradient Descent Finds Global Minima of Deep Neural Networks'. In: *Proceedings of the 36th International Conference on Machine Learning*. Ed. by Kamalika Chaudhuri and Ruslan Salakhutdinov. Vol. 97. Proceedings of Machine Learning Research. PMLR, Sept. 2019, pp. 1675–1685. URL: https://proceedings.mlr.press/v97/du19c.html (cit. on page 48).

[DHS11] John Duchi, Elad Hazan and Yoram Singer. 'Adaptive Subgradient Methods for Online Learning and Stochastic Optimization'. In: *Journal of Machine Learning Research* 12.61 (July 2011), pp. 2121–2159. ISSN: 1532-4435. URL: http://jmlr.org/papers/v12/duchi11a.html (cit. on page 30).

[DFS03] Durrel Duffie, Damir Filipović and Walter Schachermayer. 'Affine Processes and Applications in Finance'. In: *The Annals of Applied Probability* 13.3 (2003), pp. 984–1053. ISSN: 10505164. URL: http://www.jstor.org/stable/1193233 (cit. on page 64).

[DR14] Moritz Dümbgen and Leonard Christopher Gordon Rogers. 'Estimate nothing'. In: *Quantitative Finance* 14.12 (Nov. 2014), pp. 2065–2072. ISSN: 1469-7688. DOI: 10.1080/14697688.2014.951678. URL: https://doi.org/10.1080/14697688.2014.951678 (cit. on pages 6–8, 100, 102, 103, 120, 122, 124).

[Dup94] Bruno Dupire. 'Pricing with a Smile'. In: *Risk Magazine* 7 (1994), pp. 18–20 (cit. on page 16).

[EJ97] Ernst Eberlein and Jean Jacod. 'On the range of options prices (*)'. In: *Finance and Stochastics* 1.2 (Apr. 1997), pp. 131–140. DOI: 10.1007/s007800050019 (cit. on page 111).

[Eck+21] Stephan Eckstein, Gaoyue Guo, Tongseok Lim and Jan Obłój. 'Robust pricing and hedging of options on multiple assets and its numerics'. In: *SIAM J. Financial Math.* 12.1 (2021), pp. 158–188 (cit. on page 6).

[EQ95] Nicole El Karoui and Marie-Claire Quenez. 'Dynamic Programming and Pricing of Contingent Claims in an Incomplete Market'. In: *SIAM Journal on Control and Optimization* 33.1 (Jan. 1995), pp. 29–66. DOI: 10.1137/S0363012992232579 (cit. on page 111).

[Elb+21] Dennis Elbrächter, Dmytro Perekrestenko, Philipp Grohs and Helmut Bölcskei. 'Deep neural network approximation theory'. In: *IEEE Transactions on Information Theory, invited feature paper* 67.5 (May 2021). URL: http://www.nari.ee.ethz.ch/pubs/p/deep-it-2019 (cit. on pages 38, 40).

[ES16] Ronen Eldan and Ohad Shamir. 'The Power of Depth for Feedforward Neural Networks'. In: *29th Annual Conference on Learning Theory*. Ed. by Vitaly Feldman, Alexander Rakhlin and Ohad Shamir. Vol. 49. Proceedings of Machine Learning Research. Columbia University, New York, New York, USA: PMLR, 23–26 Jun 2016, pp. 907–940. URL: https://proceedings.mlr.press/v49/eldan16.html (cit. on page 40).

[Fil01] Damir Filipović. *Consistency Problems for Heath-Jarrow-Morton Interest Rate Models*. Lecture Notes in Mathematics. 1760. Springer, Jan. 2001. ISBN: 978-3-540-41493-3. DOI: 10.1007/b76888. URL: http://infoscience.epfl.ch/record/148492 (cit. on pages 5, 74, 94).

[Fil09] Damir Filipović. *Term-Structure Models: A Graduate Course*. Springer Finance, 2009. URL: http://infoscience.epfl.ch/record/148491 (cit. on page 11).

[FT02] Damir Filipović and Josef Teichmann. 'On Finite-Dimensional Term Structure Models'. In: (Jan. 2002). URL: http://infoscience.epfl.ch/record/148504 (cit. on page 91).

[FT03] Damir Filipović and Josef Teichmann. 'Existence of invariant manifolds for stochastic equations in infinite dimension'. In: *Journal of Functional Analysis* 197.2 (Feb. 2003), pp. 398–432. ISSN: 0022-1236. DOI: https://doi.org/10.1016/S0022-1236(03)00008-9. URL: https://www.sciencedirect.com/science/article/pii/S0022123603000089 (cit. on pages 91–94).

[FS02] Hans Föllmer and Alexander Schied. 'Convex measures of risk and trading constraints'. In: *Finance and Stochastics* 6.4 (Oct. 2002), pp. 429–447. DOI: 10.1007/s007800200072 (cit. on page 114).

[FS16] Hans Föllmer and Alexander Schied. *Stochastic Finance: An Introduction in Discrete Time*. Fourth Edition. De Gruyter, 2016. ISBN: 9783110463453. DOI: doi:10.1515/9783110463453 (cit. on pages 113, 115).

[FS91] Hans Föllmer and Martin Schweizer. 'Hedging of contingent claims under incomplete information'. In: ed. by M.H.A. Davis and R.J. Elliott. Stochastics Monographs. Vol. 5. Gordon and Breach, London/New York, Jan. 1991, pp. 389–414 (cit. on pages 110, 112).

[FS86] Hans Föllmer and Dieter Sondermann. 'Hedging of non-redundant contingent claims'. In: *Contributions to Mathematical Economics*. Ed. by A. Mas-Colel W. Hildenbrand. Elsevier Science Publisher (North Holland), 1986. Chap. 12, pp. 205–223 (cit. on page 111).

[Fri00] Marco Frittelli. 'The Minimal Entropy Martingale Measure and the Valuation Problem in Incomplete Markets'. In: *Mathematical Finance* 10.1 (Jan. 2000), pp. 39–52. DOI: 10.1111/1467-9965.00079 (cit. on page 113).

[GT21] Matteo Gambara and Josef Teichmann. *Consistent Recalibration Models and Deep Calibration*. 2021. arXiv: 2006.09455 (cit. on page 57).

[GBD92] Stuart Geman, Elie Bienenstock and René Doursat. 'Neural Networks and the Bias/Variance Dilemma'. In: *Neural Computation* 4.1 (Jan. 1992), pp. 1–58. DOI: 10.1162/neco.1992.4.1.1 (cit. on page 23).

[Gie+20] Patryk Gierjatowicz, Marc Sabate-Vidales, David Šiška, Lukasz Szpruch and Žan Žurič. 'Robust pricing and hedging via neural SDEs'. In: (2020) (cit. on page 6).

[GBC16] Ian Goodfellow, Yoshua Bengio and Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016 (cit. on pages 29–31, 59).

[GLP98] Christian Gouriéroux, Jean Paul Laurent and Huyên Pham. 'Mean-Variance Hedging and Numéraire'. In: *Mathematical Finance* 8.3 (July 1998), pp. 179–200. URL: https://EconPapers.repec.org/RePEc:bla:mathfi:v:8:y:1998:i:3:p:179-200 (cit. on page 112).

[Gra11] Alex Graves. 'Practical Variational Inference for Neural Networks'. In: *Advances in Neural Information Processing Systems*. Ed. by J. Shawe-Taylor, R. Zemel, P. Bartlett, F. Pereira and K. Q. Weinberger. Vol. 24. Curran Associates, Inc., 2011. URL: https://proceedings.neurips.cc/paper/2011/file/7eb3c8be3d411e8ebfab08eba5f49632-Paper.pdf (cit. on page 51).

[Gre52] Ulf Grenander. 'On empirical spectral analysis of stochastic processes'. In: *Arkiv för Matematik* 1.6 (Aug. 1952), pp. 503–531 (cit. on page 23).

[Gro15] Philipp Grohs. 'Optimally Sparse Data Representations'. In: *Harmonic and Applied Analysis: From Groups to Signals*. Ed. by Stephan Dahlke, Filippo De Mari, Philipp Grohs and Demetrio Labate. Cham: Springer International Publishing, 2015, pp. 199–248. ISBN: 978-3-319-18863-8. DOI: 10.1007/978-3-319-18863-8_5. URL: https://doi.org/10.1007/978-3-319-18863-8_5 (cit. on page 43).

[GRK20] Ingo Gühring, Mones Raslan and Gitta Kutyniok. 'Expressivity of Deep Neural Networks'. In: *ArXiv* abs/2007.04759 (2020) (cit. on page 31).

[Gun+18] Suriya Gunasekar, Jason Lee, Daniel Soudry and Nathan Srebro. 'Characterizing Implicit Bias in Terms of Optimization Geometry'. In: *Proceedings of the 35th International Conference on Machine Learning*. Ed. by Jennifer Dy and Andreas Krause. Vol. 80. Proceedings of Machine Learning Research. PMLR, Oct. 2018, pp. 1832–1841. URL: https://proceedings.mlr.press/v80/gunasekar18a.html (cit. on page 51).

[GH14] Julien Guyon and Pierre Henry-Labordère. *Nonlinear option pricing*. Chapman & Hall/CRC Financial Mathematics Series. CRC Press, Boca Raton, FL, 2014, pp. xxxviii+445. ISBN: 978-1-4665-7033-7 (cit. on page 5).

[Gyö86] Istvan Gyöngy. 'Mimicking the one-dimensional marginal distributions of processes having an Itô differential'. In: *Probability Theory and Related Fields* 71.4 (Dec. 1986), pp. 501–516. DOI: 10.1007/bf00699039. URL: %5Curl%7Bhttps://doi.org/10.1007%2Fbf00699039%7D (cit. on page 17).

[Had02] Jacques Hadamard. 'Sur les problèmes aux dérivés partielles et leur signification physique'. In: *Princeton University Bulletin* 13 (1902), pp. 49–52 (cit. on page 16).

[HMD16] Song Han, Huizi Mao and William J. Dally. 'Deep Compression: Compressing Deep Neural Network with Pruning, Trained Quantization and Huffman Coding'. In: *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*. Ed. by Yoshua Bengio and Yann LeCun. 2016. URL: http://arxiv.org/abs/1510.00149 (cit. on page 43).

[HR19] Boris Hanin and David Rolnick. 'Complexity of Linear Regions in Deep Networks'. In: *Proceedings of the 36th International Conference on Machine Learning*. Ed. by Kamalika Chaudhuri and Ruslan Salakhutdinov. Vol. 97. Proceedings of Machine Learning Research. PMLR, Sept. 2019, pp. 2596–2604. URL: https://proceedings.mlr.press/v97/hanin19a.html (cit. on page 41).

[Har+18] Philipp Harms, David Stefanovits, Josef Teichmann and Mario Valentin Wüthrich. 'Consistent recalibration of yield curve models'. In: *Mathematical Finance* 28.3 (2018), pp. 757–799. DOI: https://doi.org/10.1111/mafi.12159. URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/mafi.12159 (cit. on pages 7, 77, 94).

[HK79] John Michael Harrison and David Marc Kreps. 'Martingales and arbitrage in multiperiod securities markets'. In: *Journal of Economic Theory* 20.3 (Feb. 1979), pp. 381–408. ISSN: 0022-0531. DOI: https://doi.org/10.1016/0022-0531(79)90043-7 (cit. on page 109).

[Has+22] Trevor Hastie, Andrea Montanari, Saharon Rosset and Ryan J. Tibshirani. 'Surprises in high-dimensional ridgeless least squares interpolation'. In: *The Annals of Statistics* 50.2 (Apr. 2022), pp. 949–986. DOI: 10.1214/21-AOS2133 (cit. on page 50).

[He+15] Kaiming He, Xiangyu Zhang, Shaoqing Ren and Jian Sun. 'Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification'. In: *2015 IEEE International Conference on Computer Vision (ICCV)*. 2015, pp. 1026–1034. DOI: 10.1109/ICCV.2015.123 (cit. on page 48).

[He+16] Kaiming He, Xiangyu Zhang, Shaoqing Ren and Jian Sun. 'Deep Residual Learning for Image Recognition'. In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016, pp. 770–778. DOI: 10.1109/CVPR.2016.90 (cit. on pages 46, 47, 81).

[HJM92] David Heath, Robert Jarrow and Andrew Morton. 'Bond Pricing and the Term Structure of Interest Rates: A New Methodology for Contingent Claims Valuation'. In: *Econometrica* 60.1 (1992), pp. 77–105. URL: https://EconPapers.repec.org/RePEc:ecm:emetrp:v:60:y:1992:i:1:p:77-105 (cit. on pages 10, 13).

[HPS01] David Heath, Eckhard Platen and Martin Schweizer. 'A Comparison of Two Quadratic Approaches to Hedging in Incomplete Markets'. In: *Mathematical Finance* 11.4 (Oct. 2001), pp. 385–413. URL: https://EconPapers.repec.org/RePEc:bla:mathfi:v:11:y:2001:i:4:p:385-413 (cit. on page 112).

[HTW19] Jakob Heiss, Josef Teichmann and Hanna Wutte. 'How implicit regularization of Neural Networks affects the learned function - Part I'. In: *CoRR* abs/1911.02903 (2019). arXiv: 1911.02903. URL: http://arxiv.org/abs/1911.02903 (cit. on pages 54, 55, 58).

[Her16] Andres Hernandez. 'Model calibration with neural networks'. In: *Available at SSRN 2812140* (2016) (cit. on pages 58, 78).

[Hes93] Steven L. Heston. 'A Closed-Form Solution for Options with Stochastic Volatility with Applications to Bond and Currency Options'. In: *Review of Financial Studies* 6 (Apr. 1993), pp. 327–343. ISSN: 0893-9454. DOI: 10.1093/rfs/6.2.327 (cit. on pages 99, 109).

[HL86] Thomas S. Y. Ho and Sang-Bin Lee. 'Term Structure Movements and Pricing Interest Rate Contingent Claims'. In: *The journal of finance.* 41.5 (Dec. 1986). ISSN: 0022-1082. DOI: 10.2307/2328161 (cit. on page 10).

[HN89] Stewart D. Hodges and Anthony Neuberger. 'Optimal replication of contingent claims under transaction costs'. In: *Review Futures Market* 8 (Nov. 1989), pp. 222–239. ISSN: 0898-011X (cit. on pages 110, 113).

[Hoe+99] Jennifer A. Hoeting, David Madigan, Adrian E. Raftery and Chris T. Volinsky. 'Bayesian model averaging: a tutorial'. In: *Statist. Sci.* 14.4 (1999). With comments by M. Clyde, David Draper and E. I. George, and a rejoinder by the authors, pp. 382–417. ISSN: 0883-4237 (cit. on page 8).

[Hor91] Kurt Hornik. 'Approximation capabilities of multilayer feedforward networks'. In: *Neural networks* 4.2 (Mar. 1991), pp. 251–257 (cit. on page 36).

[HSW90] Kurt Hornik, Maxwell Stinchcombe and Halbert White. 'Universal approximation of an unknown mapping and its derivatives using multilayer feedforward networks'. In: *Neural Networks* 3.5 (1990), pp. 551–560. ISSN: 0893-6080. DOI: https://doi.org/10.1016/0893-6080(90)90005-6. URL: https://www.sciencedirect.com/science/article/pii/0893608090900056 (cit. on pages 35, 36).

[HMT21] Blanka Horvath, Aitor Muguruza and Mehdi Tomas. 'Deep learning volatility: a deep neural network perspective on pricing and calibration in (rough) volatility models'. In: *Quantitative Finance* 21.1 (2021), pp. 11–27. DOI: 10.1080/14697688.2020.1817974. URL: https://doi.org/10.1080/14697688.2020.1817974 (cit. on pages 59, 78, 79).

[HTŽ21] Blanka Horvath, Josef Teichmann and Žurič, Žan. 'Deep Hedging under Rough Volatility'. In: *Risks* 9.7 (July 2021). ISSN: 2227-9091. DOI: 10.3390/risks9070138. URL: https://www.mdpi.com/2227-9091/9/7/138 (cit. on page 120).

[Hua+16] Gao Huang, Yu Sun, Zhuang Liu, Daniel Sedra and Kilian Q. Weinberger. 'Deep Networks with Stochastic Depth'. In: *Computer Vision – ECCV 2016*. Ed. by Bastian Leibe, Jiri Matas, Nicu Sebe and Max Welling. Cham: Springer International Publishing, 2016, pp. 646–661. ISBN: 978-3-319-46493-0 (cit. on page 47).

[HK04] Julien Hugonnier and Dmitry Olegovich Kramkov. 'Optimal Investment with Random Endowments in Incomplete Markets'. In: *The Annals of Applied Probability* 14.2 (May 2004), pp. 845–864. ISSN: 10505164. URL: http://www.jstor.org/stable/4140431 (cit. on page 113).

[HW90] John Hull and Alan White. 'Pricing Interest-Rate-Derivative Securities'. In: *Review of Financial Studies* 3.4 (1990), pp. 573–592. URL: https://www.jstor.org/stable/2962116 (cit. on page 10).

[İS06] Aytaç İlhan and Ronnie Sircar. 'Optimal Static–Dynamic Hedges for Barrier Options'. In: *Mathematical Finance* 16.2 (Apr. 2006), pp. 359–385. DOI: 10.1111/j.1467-9965.2006.00275.x (cit. on page 113).

[IS15] Sergey Ioffe and Christian Szegedy. 'Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift'. In: *Proceedings of the 32nd International Conference on Machine Learning*. Ed. by Francis Bach and David Blei. Vol. 37. Proceedings of Machine Learning Research. Lille, France: PMLR, July 2015, pp. 448–456. URL: https://proceedings.mlr.press/v37/ioffe15.html (cit. on page 82).

[JYC12] Monique Jeanblanc, Marc Yor and Marc Chesney. *Mathematical Methods for Financial Markets*. Springer Finance. Springer London, 2012. ISBN: 9781447125242. URL: https://books.google.ch/books?id=Ypd7uAAACAAJ (cit. on pages 103, 109).

[KK15] Jan Kallsen and Paul Krühner. 'On a Heath-Jarrow-Morton approach for stock options'. In: *Finance and Stochastics* 19.3 (July 2015). Copyright - Springer-Verlag Berlin Heidelberg 2015, pp. 583–615. DOI: 10.1007/s00780-015-0263-1 (cit. on pages 6, 12, 17, 18, 57, 60–63, 67, 71, 75, 95).

[KS98] Ioannis Karatzas and Steven Eugene Shreve. *Brownian Motion and Stochastic Calculus*. Second Edition. Graduate Texts in Mathematics (113). Springer New York, 1998. ISBN: 9780387976556 (cit. on page 111).

[KNS16] Hamed Karimi, Julie Nutini and Mark Schmidt. 'Linear Convergence of Gradient and Proximal-Gradient Methods Under the Polyak-Łojasiewicz Condition'. In: *European Conference on Machine Learning and Knowledge Discovery in Databases - Volume 9851*. ECML PKDD 2016. Riva del Garda, Italy: Springer-Verlag, 2016, pp. 795–811. ISBN: 9783319461274. DOI: 10.1007/978-3-319-46128-1_50. URL: https://doi.org/10.1007/978-3-319-46128-1_50 (cit. on page 24).

[Kel08] Martin Keller-Ressel. *Affine processes: theory and applications in finance*. Citeseer, 2008 (cit. on page 65).

[KST11] Martin Keller-Ressel, Walter Schachermayer and Josef Teichmann. 'Affine processes are regular'. en. In: *Probability Theory and Related Fields* 151.3-4 (Dec. 2011). Received 18 June 2009, Revised 22 March 2010, Published online 30 June 2010. It was possible to publish this article open access thanks to a Swiss National Licence with the publisher, pp. 591–611. ISSN: 0178-8051. DOI: 10.3929/ethz-b-000028228 (cit. on page 64).

[KST13] Martin Keller-Ressel, Walter Schachermayer and Josef Teichmann. 'Regularity of affine processes on general state spaces'. In: *Electronic journal of probability* 18 (2013), pp. 1–17 (cit. on pages 64, 65).

[Kel72] Hans G. Kellerer. 'Markov-Komposition und eine Anwendung auf Martingale.' In: *Mathematische Annalen* 198 (1972), pp. 99–122. URL: http://eudml.org/doc/162296 (cit. on page 16).

[KB15] Diederik P. Kingma and Jimmy Ba. 'Adam: A Method for Stochastic Optimization'. In: *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*. Ed. by Yoshua Bengio and Yann LeCun. Jan. 2015. URL: http://arxiv.org/abs/1412.6980 (cit. on pages 30, 82, 122).

[KS07] Susanne Klöppel and Martin Schweizer. 'Dynamic utility indifference valuation via convex risk measures'. In: *Mathematical Finance* 17.4 (Oct. 2007), pp. 599–627. DOI: https://doi.org/10.1111/j.1467-9965.2007.00317.x (cit. on page 113).

[Kni21] Frank H. Knight. *Risk, Uncertainty and Profit*. Boston, MA: Houghton Mifflin Co, 1921 (cit. on page 98).

[Kra96] Dmitry Olegovich Kramkov. 'Optional decomposition of supermartingales and hedging contingent claims in incomplete security markets'. In: *Probability Theory and Related Fields* 105 (Dec. 1996), pp. 459–479. DOI: 10.1007/BF01191909 (cit. on page 111).

[KS99] Dmitry Olegovich Kramkov and Walter Schachermayer. 'The Asymptotic Elasticity of Utility Functions and Optimal Investment in Incomplete Markets'. In: *The Annals of Applied Probability* 9.3 (Aug. 1999), pp. 904–950. ISSN: 10505164. DOI: 10.1214/aoap/1029962818. URL: http://www.jstor.org/stable/2667287 (cit. on page 113).

[KW67] Hiroshi Kunita and Shinzo Watanabe. 'On Square Integrable Martingales'. In: *Nagoya Mathematical Journal* 30 (Aug. 1967), pp. 209–245. DOI: 10.1017/S0027763000012484 (cit. on page 111).

[LPS98] Damien Lamberton, Huyên Pham and Martin Schweizer. 'Local Risk-Minimization Under Transaction Costs'. In: *Mathematics of Operations Research* 23.3 (Aug. 1998), pp. 585–612. DOI: 10.1287/moor.23.3.585 (cit. on pages 112, 117).

[Les+93] Moshe Leshno, Vladimir Ya. Lin, Allan Pinkus and Shimon Schocken. 'Multilayer feedforward networks with a nonpolynomial activation function can approximate any function'. In: *Neural Networks* 6.6 (Jan. 1993), pp. 861–867. ISSN: 0893-6080. DOI: https://doi.org/10.1016/S0893-6080(05)80131-5. URL: https://www.sciencedirect.com/science/article/pii/S0893608005801315 (cit. on page 36).

[LRP16] Laurent Lessard, Benjamin Recht and Andrew Packard. 'Analysis and Design of Optimization Algorithms via Integral Quadratic Constraints'. In: *SIAM Journal on Optimization* 26.1 (Jan. 2016), pp. 57–95. DOI: 10.1137/15M1009597. eprint: https://doi.org/10.1137/15M1009597. URL: https://doi.org/10.1137/15M1009597 (cit. on page 30).

[LSS21] Eva Lütkebohmert, Thorsten Schmidt and Julian Sester. 'Robust deep hedging'. In: *arXiv* (2021) (cit. on pages 6, 122).

[MP43] Warren Mcculloch and Walter Pitts. 'A Logical Calculus of Ideas Immanent in Nervous Activity'. In: *Bulletin of Mathematical Biophysics* 5 (Dec. 1943), pp. 127–147 (cit. on page 30).

[MRT18] Mehryar Mohri, Afshin Rostamizadeh and Ameet Talwalkar. *Foundations of Machine Learning*. 2nd ed. Adaptive Computation and Machine Learning. Cambridge, MA: MIT Press, 2018. 504 pp. ISBN: 978-0-262-03940-6 (cit. on page 42).

[Mon17] Guido F. Montúfar. 'Notes on the number of linear regions of deep neural networks'. In: *SampTA*. Mar. 2017 (cit. on page 41).

[Mon+14] Guido F. Montúfar, Razvan Pascanu, Kyunghyun Cho and Yoshua Bengio. 'On the Number of Linear Regions of Deep Neural Networks'. In: *Advances in Neural Information Processing Systems*. Ed. by Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence and K. Q. Weinberger. Vol. 27. Curran Associates, Inc., Dec. 2014. URL: https://proceedings.neurips.cc/paper/2014/file/109d2dd3608f669ca17920c511c2a41e-Paper.pdf (cit. on page 41).

[Nes83] Yurii Nesterov. 'A method for solving the convex programming problem with convergence rate $\mathcal{O}(1/k^2)$'. In: *Proceedings of the USSR Academy of Sciences* 269 (1983), pp. 543–547. (Translated from: Soviet Mathematics Doklady 27 372–376, Zbl 0535.90071 - Original in Russian) (cit. on page 30).

[Nes14] Yurii Nesterov. *Introductory Lectures on Convex Optimization: A Basic Course*. 1st Edition. Springer Publishing Company, Incorporated, 2014. ISBN: 1461346916 (cit. on page 30).

[NN18] Ariel Neufeld and Marcel Nutz. 'Robust Utility Maximization with Lévy Processes'. In: *Mathematical Finance* 28.1 (Jan. 2018), pp. 82–105. DOI: https://doi.org/10.1111/mafi.12139. URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/mafi.12139 (cit. on page 98).

[OW21] Jan Obłój and Johannes Wiesel. 'A unified Framework for Robust Modelling of Financial Markets in Discrete Time'. In: *Finance and Stochastics* 25.3 (June 2021), pp. 427–468. DOI: 10.1007/s00780-021-00454-7 (cit. on page 99).

[PV18] Philipp Petersen and Felix Voigtlaender. *Optimal approximation of piecewise smooth functions using deep ReLU neural networks*. 2018. arXiv: 1709.05289 [math.FA] (cit. on page 40).

[Pin99] Allan Pinkus. 'Approximation theory of the MLP model in neural networks'. In: *Acta Numerica* 8 (Jan. 1999), pp. 143–195. DOI: 10.1017/S0962492900002919 (cit. on page 36).

[Pol84] David Pollard. *Convergence of stochastic processes*. Springer series in statistics. New York; Berlin; Tokyo: Springer, 1984. ISBN: 0387909907 (cit. on page 42).

[Pol63] Boris Teodorovich Polyak. 'Gradient methods for the minimisation of functionals'. In: *USSR Computational Mathematics and Mathematical Physics* 3.4 (1963), pp. 864–878. ISSN: 0041-5553. DOI: https://doi.org/10.1016/0041-5553(63)90382-3. URL: https://www.sciencedirect.com/science/article/pii/0041555363903823. (Translated from: Zh. Vych. Mat. 3, No. 4, 643-653, 1963 - Original in Russian) (cit. on page 26).

[Pol64] Boris Teodorovich Polyak. 'Some methods of speeding up the convergence of iteration methods'. In: *USSR Computational Mathematics and Mathematical Physics* 4.5 (1964), pp. 1–17. ISSN: 0041-5553. DOI: https://doi.org/10.1016/0041-5553(64)90137-5. URL: https://www.sciencedirect.com/science/article/pii/0041555364901375. (Translated from: Zh. Vych. Mat., 4, No. 5, 791-803, 1964 - Original in Russian) (cit. on page 30).

[Rab00] Matthew Rabin. 'Risk Aversion and Expected-Utility Theory: A Calibration Theorem'. In: *Econometrica* 68.5 (Sept. 2000), pp. 1281–1292. ISSN: 00129682, 14680262. DOI: 10.1111/1468-0262.00158 (cit. on page 114).

[RC20] Noam Razin and Nadav Cohen. 'Implicit Regularization in Deep Learning May Not Be Explainable by Norms'. In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2020 (cit. on page 52).

[RT17] Anja Richter and Josef Teichmann. 'Discrete Time Term Structure Theory and Consistent Recalibration Models'. In: *SIAM Journal on Financial Mathematics* 8.1 (2017), pp. 504–531. DOI: 10.1137/15M1007434. URL: https://doi.org/10.1137/15M1007434 (cit. on pages 7, 57, 61, 62, 68, 76, 94).

[RM51] Herbert Robbins and Sutton Monro. 'A Stochastic Approximation Method'. In: *The Annals of Mathematical Statistics* 22.3 (Sept. 1951), pp. 400–407. ISSN: 00034851. URL: http://www.jstor.org/stable/2236626 (cit. on page 28).

[RT97] Marc Romano and Nizar Touzi. 'Contingent Claims and Market Completeness in a Stochastic Volatility Model'. In: *Mathematical Finance* 7 (Oct. 1997), pp. 399–412. DOI: 10.1111/1467-9965.00038 (cit. on page 110).

[Rop10] Michael Roper. 'Arbitrage free implied volatility surfaces'. In: (2010) (cit. on pages 13, 14).

[Ros57] Frank Rosenblatt. *The perceptron - A perceiving and recognizing automaton*. Tech. rep. 85-460-1. Ithaca, New York: Cornell Aeronautical Laboratory, Jan. 1957 (cit. on page 31).

[RE00] Richard Rouge and Nicole El Karoui. 'Pricing Via Utility Maximization and Entropy'. In: *Mathematical Finance* 10.2 (Apr. 2000), pp. 259–276. DOI: 10.1111/1467-9965.00093 (cit. on page 113).

[RW20] Johannes Ruf and Weiguan Wang. 'Neural Networks for Option Pricing and Hedging: A Literature Review'. In: *SSRN Electronic Journal* 4.1 (June 2020), pp. 1–46. DOI: 10.2139/ssrn.3486363. URL: https://papers.ssrn.com/sol3/papers.cfm?abstract_id=3486363 (cit. on page 8).

[RH19] Navid Azizan Ruhi and Babak Hassibi. 'Stochastic Gradient/Mirror Descent: Minimax Optimality and Implicit Regularization'. In: *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019. URL: https://openreview.net/forum?id=HJf9ZhC9FX (cit. on page 52).

[Sch99a] Philipp J. Schönbucher. 'A Market Model for Stochastic Implied Volatility'. In: *Philosophical Transactions: Mathematical, Physical and Engineering Sciences* 357.1758 (1999), pp. 2071–2092. ISSN: 1364503X. URL: http://www.jstor.org/stable/55251 (cit. on page 15).

[Sch92] Martin Schweizer. 'Mean-Variance Hedging for General Claims'. In: *The Annals of Applied Probability* 2.1 (Feb. 1992), pp. 171–179. DOI: 10.1214/aoap/1177005776 (cit. on page 112).

[Sch99b] Martin Schweizer. *A guided tour through quadratic hedging approaches*. SFB 373 Discussion Papers 1999, 96. Humboldt University of Berlin, Interdisciplinary Research Project 373: Quantification and Simulation of Economic Processes, Nov. 1999 (cit. on pages 111, 112).

[Sch10] Martin Schweizer. 'Mean–Variance Hedging'. In: *Encyclopedia of Quantitative Finance*. Ed. by R. Cont. John Wiley & Sons, Ltd, May 2010, pp. 1177–1181. ISBN: 9780470061602. DOI: https://doi.org/10.1002/9780470061602.eqf14025 (cit. on page 112).

[SW08a] Martin Schweizer and Johannes Wissel. 'Arbitrage-free market models for option prices: The multi-strike case'. In: *Finance and Stochastics* 12 (May 2008), pp. 469–505. DOI: 10.1007/s00780-008-0068-6 (cit. on pages 16, 17).

[SW08b] Martin Schweizer and Johannes Wissel. 'Term Structures of implied volatilities: absence of arbitrage and existence results'. In: *Mathematical Finance* 18.1 (Jan. 2008), pp. 77–114. DOI: https://doi.org/10.1111/j.1467-9965.2007.00323.x. URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1467-9965.2007.00323.x (cit. on pages 15, 16).

[SB14] Shai Shalev-Shwartz and Shai Ben-David. *Understanding Machine Learning - From Theory to Algorithms*. Cambridge University Press, June 2014, pp. I–XVI, 1–397. ISBN: 978-1-10-705713-5 (cit. on page 19).

[SC02] Albert Nikolayevich Shiryaev and Alexander Semenovich Cherny. 'Vector Stochastic Integrals and the Fundamental Theorems of Asset Pricing'. In: *Proc. Steklov Inst. Math.* 237 (Jan. 2002), pp. 6–49 (cit. on page 104).

[SSC95] Halil Mete Soner, Steven Eugene Shreve and Jakša Cvitanić. 'There is no Nontrivial Hedging Portfolio for Option Pricing with Transaction Costs'. In: *The Annals of Applied Probability* 5.2 (May 1995), pp. 327–355. ISSN: 10505164. URL: http://www.jstor.org/stable/2245301 (cit. on page 110).

[Sri+14] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever and Ruslan Salakhutdinov. 'Dropout: A Simple Way to Prevent Neural Networks from Overfitting'. In: *Journal of Machine Learning Research* 15.56 (June 2014), pp. 1929–1958. URL: http://jmlr.org/papers/v15/srivastava14a.html (cit. on page 82).

[Ste17] Mark F. J. Steel. 'Model Averaging and its Use in Economics'. In: *arXiv* (2017) (cit. on page 8).

[Sto48] Marshall Harvey Stone. 'The Generalized Weierstrass Approximation Theorem'. In: *Mathematics Magazine* 21.4 (Apr. 1948), pp. 167–184. ISSN: 0025570X, 19300980. URL: http://www.jstor.org/stable/3029750 (cit. on pages 37, 38).

[Sut+13] Ilya Sutskever, James Martens, George Dahl and Geoffrey Hinton. 'On the importance of initialization and momentum in deep learning'. In: *Proceedings of the 30th International Conference on Machine Learning*. Ed. by Sanjoy Dasgupta and David McAllester. Vol. 28. Proceedings of Machine Learning Research 3. Atlanta, Georgia, USA: PMLR, 17–19 Jun 2013, pp. 1139–1147. URL: https://proceedings.mlr.press/v28/sutskever13.html (cit. on page 30).

[Tei13] Josef Teichmann. *Mathematical finance (extended from lectures of Fall 2012 by Martin Schweizer transcribed by Peter Gracar and Thomas Hille)*. 2013. URL: https://metaphor.ethz.ch/x/2020/hs/401-4889-00L/src/lecturenotesMF20191212.pdf (cit. on pages 103, 111).

[Tel16] Matus Telgarsky. 'Benefits of Depth in Neural Networks'. In: *29th Annual Conference on Learning Theory*. Ed. by Vitaly Feldman, Alexander Rakhlin and Ohad Shamir. Vol. 49. Proceedings of Machine Learning Research. Columbia University, New York, New York, USA: PMLR, 23–26 Jun 2016, pp. 1517–1539. URL: https://proceedings.mlr.press/v49/telgarsky16.html (cit. on pages 37, 39).

[VC71] Vladimir Naumovich Vapnik and Alexey Yakovlevich Chervonenkis. 'On the Uniform Convergence of Relative Frequencies of Events to Their Probabilities'. In: *Theory of Probability and its Applications* 16.2 (July 1971), pp. 264–280. DOI: 10.1137/1116025. Translated by B. Seckler from: Dokl. Akad. Nauk. 181 (4): 781. 1968. - Original in Russian (cit. on page 41).

[VS21] Gal Vardi and Ohad Shamir. 'Implicit Regularization in ReLU Networks with the Square Loss'. In: *Proceedings of Thirty Fourth Conference on Learning Theory*. Ed. by Mikhail Belkin and Samory Kpotufe. Vol. 134. Proceedings of Machine Learning Research. PMLR, 15–19 Aug 2021, pp. 4224–4258. URL: https://proceedings.mlr.press/v134/vardi21b.html (cit. on page 53).

[Vaš77] Oldrich Vašíček. 'An equilibrium characterization of the term structure'. In: *Journal of Financial Economics* 5.2 (Nov. 1977), pp. 177–188. ISSN: 0304-405X. DOI: https://doi.org/10.1016/0304-405X(77)90016-2. URL: https://www.sciencedirect.com/science/article/pii/0304405X77900162 (cit. on pages 10, 11, 122).

[VWB16] Andreas Veit, Michael Wilber and Serge Belongie. 'Residual Networks Behave like Ensembles of Relatively Shallow Networks'. In: *Proceedings of the 30th International Conference on Neural Information Processing Systems*. NIPS'16. Barcelona, Spain: Curran Associates Inc., 2016, pp. 550–558. ISBN: 9781510838819 (cit. on page 47).

[Vla+19] Mariia Vladimirova, Jakob Verbeek, Pablo Mesejo and Julyan Arbel. 'Understanding Priors in Bayesian Neural Networks at the Unit Level'. In: *Proceedings of the 36th International Conference on Machine Learning*. Ed. by Kamalika Chaudhuri and Ruslan Salakhutdinov. Vol. 97. Proceedings of Machine Learning Research. PMLR, Sept. 2019, pp. 6458–6467. URL: https://proceedings.mlr.press/v97/vladimirova19a.html (cit. on page 51).

[Wie+19] Magnus Wiese, Lianjun Bai, Ben Wood and Hans Buehler. 'Deep Hedging: Learning to Simulate Equity Option Markets'. In: *SSRN Electronic Journal* (Jan. 2019). DOI: 10.2139/ssrn.3470756 (cit. on page 59).

[Wis07] Johannes Wissel. 'Arbitrage-free market models for option prices'. en. In: *NCCR Finrisk Working Paper Series* (2007). Ed. by Financial Valuation National Centre of Competence in Research and Risk Management, p. 428 (cit. on pages 16, 17, 59).

[Xu06] Mingxin Xu. 'Risk measure pricing and hedging in incomplete markets'. In: *Annals of Finance* 2.1 (Feb. 2006), pp. 51–71. DOI: 10.1007/s10436-005-0023-x (cit. on pages 113–115).

[Yar17] Dmitry Yarotsky. 'Error bounds for approximations with deep ReLU networks'. In: *Neural Networks* 94 (2017), pp. 103–114. ISSN: 0893-6080. DOI: https://doi.org/10.1016/j.neunet.2017.07.002. URL: https://www.sciencedirect.com/science/article/pii/S0893608017301545 (cit. on pages 37, 38, 40).

[Zei12] Matthew D. Zeiler. *ADADELTA: An Adaptive Learning Rate Method*. 2012. DOI: 10.48550/ARXIV.1212.5701. URL: https://arxiv.org/abs/1212.5701 (cit. on page 30).

[Zha+16] Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht and Oriol Vinyals. 'Understanding deep learning requires rethinking generalization'. In: *Communications of the ACM* 64 (Nov. 2016). DOI: 10.1145/3446776 (cit. on page 48).

Acknowledgements

When I started more than four years ago, I did not know what to expect from a doctorate. I knew it would not be easy, as other good friends embarked on the same adventure and I could see their dedication, but I just wanted to get back to doing one of the things I liked the most: studying mathematics.
I still remember the fatigue of the first months, when I had to resume studying mathematical topics that I had not seen for two years by then (it seems little, but for a precise and exact science, it is a lot), and when I had to learn to divide my time between study and work, which I had kept in part-time mode.
If I gritted my teeth and managed to make it toward the end, I owe it mainly to the people who stood by me during these years and gave me the strength to recover from every fall or disappointment that, inevitably, study and research entail.

Fortunately, there have not only been the hard times. On the contrary, I have found out that doing research is intimately connected with getting to know the people with whom you work, from whom you learn, and with whom, in the end, you spend some of the best moments of your life. In fact, that is the only way I can sum up this period. As one of the best ever, for me. I wish everyone that inebriating feeling you get when you manage to solve a problem, perhaps with a friend or colleague, after days spent on it. That is also why we like mathematics, after all. And that is also what creates strong ties among people, I believe. New good friends and new beautiful discoveries in mathematics. What else could I ask for?

In any case, this section is dedicated to all the people I have crossed on my path during the past years and made this journey so special and unique to me. To those that were already there and did not leave, those who were there and are gone, those who only appeared momentarily, and those who I hope will stay (as long as possible).

But before continuing, just a couple of words on the current situation. Now it is easy for me to remember all the good moments I passed with my friends, but some periods were very tough. In particular, the world has been turned upside down over the past few years by a pandemic that has overturned our lives, at least partially, and is not quite over yet. As I said, I have been fortunate to have all these people by my side, but for some it has been harder than for others. So my thoughts go, first of all, to Matti. You were a brilliant mathematician, never ordinary, and it was always interesting and fun talking to you. For what it is worth, it was a pleasure to have met you and to have been colleagues together.

When I first came some years ago, I was thinking of finding very intelligent and diligent people. As you could easily have guessed, it came out that I was right. But not only I found that. I met new friends. Almost a new family. And that was, I am sure, the biggest gift I received from this experience.

I will now undertake the task of thanking all the people who are important to me. I hope I don't forget anyone. But if I do, I apologize: the last few weeks have been very busy (to put it mildly). I hope I could meet everyone's expectations.
Both as a friend, for most of you, and as a student, as a Ph.D. candidate.

The first person I would like to thank is Professor Teichmann who accepted me as Ph.D. student. I still remember when we met for the first time. I was telling him that I was no genius, but rather a hardworking guy, and he replied that this was not the way mathematics has been developing in the last decades and that the image of a standalone mathematician is almost secluded to history.
I already told you how much I enjoyed our talks, discussions and the work together. I learned a lot from all of them. I tried to squeeze every single word you said to become a good mathematician. But from you I not only got mathematics and brilliant ideas. I think the most important lesson you transmitted was about freedom. You taught me that freedom is not only what the others give us, but it is also what we can take from ourselves. You taught that we should always go for the most difficult targets, without fear; that daring is allowed. I hope I can keep this lesson in my mind for the rest of my life.
Secondly, I also learnt that being a good runner is positively correlated with being a good mathematician, as the SOLA run testifies. And I hope that next time we will aim for even better results.

I would also like to thank Professor Nadtochiy for accepting the invitation to come to Zurich as a co-advisor for my thesis. We have never met in person, but in the first years of my Ph.D. I worked on your research and it is therefore an honour to have you here.
Last but not least, I would also like to thank Professor Schweizer for being the chairman of the committee. This time, I must say that in the last year of research I have worked on and taken inspiration from many of your articles on hedging in incomplete markets. I can only renew my admiration.

Many thanks to Beatrice and Dylan and also Thorsten: you have always been kind, available and down-to-earth. I enjoyed all the talks and chats together. Thanks also Celeste for being so sweet, sensible and always ready to help and support others. I think everybody likes you. For being the "latina" of the group. We will manage to organize that karaoke night sooner or later.

Thanks also to Jean-Luc, whose name every time tricks my brain and I try to speak French, and especially to Denise. It is a pity that because of the Covid we saw less of each other. You have always been supportive over the years whenever I had questions of an administrative or bureaucratic nature. And thanks also for the nice messages and emails. I will keep them.

Time has come to thank the people that allowed me to pursue this path in practice. During these years, I have continued working for Finma and they accepted the idea that I could work less for them in order to deepen my studies. It was not an obvious decision. In particular, I need to thank the people that sponsored me in first instance, such as Michael and Birgit. Michael played a prominent role in this process and I will always be grateful for it. Actually, he is also the reason why I could find a job at Finma. He was always supportive and honest, and a rare example of mathematical intelligence and exquisite human qualities. I am sure he will have a great career.
I would also like to thank the other colleagues of V-RIM. In particular Franziska, for the little talks in Italian and her help whenever I could not fully understand what was going on in German.

Christoph, for the always-interesting discussions, themes and the inquisitive mathematical questions; the knowledge of reinsurance and for the current project on machine learning. There is always so much to learn from you. François, for the support shown to Italian as an official Swiss language, the projects together and for the stories on his family and for having a kindred spirit. Irina, for her heart-felt friendship, which is one of the most precious things, the (double) visit in Parma and for the long chats on everything. Laurent, for what he could teach me and the many brilliant solutions. Johannes, because he is the most reliable colleague one can have, and he is always ready to help. Susi, for being extremely supportive, comprehensive, humble and being a perfect manager. Falk, for his correct and thoughtful leadership; it is great that we will have more time to work together. Andreas and Paul for the projects together and their precious guidance and expertise. Thorsten for his Italian and the memories of the Italian doctorate in Rome. Angela, for her knowledge of Japan and non-life insurance. Camille, because despite now being a successful manager, she is always treating me as an old good friend. Rainer, for the holidays suggestions and because he willingly speaks English. Urs, for his availability. And also the new colleagues: Patrick, Jérôme, and the others. And also the colleagues who left: Laura, for her help and friendship and for the organization of many dinners and trips; Jeta, for her being a book-lover and for being a sort of sister-colleague; Daniel, for his optimistic views (good luck with Loco!); Fernando, for the fun together; Stephan, for his friendship. I have learnt so much from all of you. I can only say "thank you". I hope I was a good colleague, at least as good as you were with me.

I now need to thank all the friends I met at ETH. In principle, it is possible to study mathematics from home, but if I tried to come to the university anytime I could, it is just because you were always there. For the first time in Zurich, I could find a group of friends with whom to hang out, relax, joke around. I will not try to recall all the good moments spent together as they are simply too many. I already know I will miss them a lot, and I hope I will be able to stay with you all for a while longer. So, let us start.

First of all, I would like to mention my two extra-ordinary office mates. Since you joined, I have got to know you better. I am glad that you 'happened' to be with me. Florian R., because although he is very young, he already has a promising career as a mathematician. For all the FIM Teas we went to to pick up some food and for the confidences and comparisons on latest news.
Moritz, because now a glance is enough for us to understand each other and to know what we are thinking. And for the hearty laughter inside and outside the walls of the ETH. Sooner or later we will enjoy a sauna together and I'll be able to see if your progress in Swiss-German is finally good for something.

Martin, because he has been basically at my side from the beginning. For being a loyal Ph.D. mate. Always ready to offer his help when needed, despite being incredibly busy with his jobs. This is how one defines friends, I guess. I am looking forward to going together on a wedding this summer. It will be fun!
Hanna, because she is my Ph.D. sister, for being the person with whom I have always shared doubts and questions about our future. For being one of the pillars of the group. For the afternoons spent together making exercises for the course on reinforcement learning. Hopefully, I will disturb you again in your office for a coffee while you are having a call. And, hopefully, you will again forgive me.
Daria, for chatting together, talking about maths and anything else. For your kindness and your care. You are one of the strongest people I know: proud, thoughtful, never afraid to express your

opinion. For this I really admire you. I would never pass up a chat with you. I promise I will read all books you suggested me.
Syang, who shares with me not only the advisor, but also the expedient of doing his Ph.D. and working for a company in the meantime. We also studied together for different courses. We became friends because of the work we did together on (randomized) signatures. It was really nice to be able to count on you. I hope that in our future work together we will find the same chemistry that made us progress in the first project.
Erdinç, because "there is no two without three", because you helped Syang and me in the projects with signatures. I hope there will still be a chance to work together and to visit you in Turkey. I wish you the best in your academic career.
Florian K., because I still remember when we first met: it was during my first year of my PhD and I had to follow you and other friends of yours for a project on interest rates. I could see even then that you were the most interested in the subject and that my task as supervisor would turn out to be easier than expected. Thank you also for your Italian, which is far better than my German. I hope we will see each other often, perhaps on holiday in Italy.
Thanks also to Robert, who always managed to organise the group's teaching activities impeccably. For the stories about how his parents met, on which it would be possible to make a film, for his accounts of the weeks in Africa and for his being German, Greek and American at the same time.
Calypso, because I also shared many moments of this journey with her: worries, hopes and, finally, the defense itself. It is always nice to see how we could be friends and dialogue despite having different opinions. It is even nicer to know that even if we do not talk for a while, you are always ready to help me out. So, thank you for rereading the thesis and hunting down mistakes and forgetfulness. I won't forget.
Chong, for the dinners together and the long talks about the world and China. Thank you for your honesty and accuracy.
Zhouyi, because although we did not get to know each other too well, we had our moments of fun and jokes together.
Jakob H., because I still have to understand if he is smarter or faster (in running). Because, in addition to the great performance, he managed amazingly well the SOLA run; for having read a small subsection of the thesis. And also because he is one of the few I know who can actually eat more than me while remaining paper-thin.
David, for the two questions I had and his ready answers, for having explained what are the rules for the rankings in chess tournaments and for being the greatest chess player I know. And it is likely that he is also better than the reader.
Jakob W., for his frankness, spirit and ability to find a good joke in any situation. For the hikes and dinners together.

Special mention must be made of the two Italian guys I met when I arrived: Vincenzo and Matteo. You were the first with whom I realised that I had not just found new colleagues, but real friends. And that thought came as a pleasant surprise. Even though you left ETH a few years ago, we practically talk every day on a Signal group (with a dubious name) and I think will continue for long. I hope our bond will not be affected by time. Thank you also for helping me in this last period with the re-reading of some parts of the thesis and, especially to Matteo, for a couple of last-minute questions. We will definitely meet again for Vincenzo's wedding all together and I am looking forward to it.

Many thanks to Nikolay, another clear example of the friendships that can be born within these walls. For your loyalty, friendship, the long chats at the bus stop waiting for the night-bus to pass. For our common interests, from mathematics to our respective cultures. For the many invitations to visit you in Australia, which I hope I will soon accept, and because I hope we can

also meet somewhere in Italy even before. For what you taught me about reinforcement learning and for a Sunday afternoon spent together writing code to try to solve a problem. I will never forget the feeling of victory we felt at the end. I wish you and Maria all the best for the future, wherever it may be.

Anastasis and Behnoosh, because they are a wonderful couple. For all the times we hung out together, for the jokes and laughter. To Anastasis, for making me discover that 'Coolio' is not just a rapper's name, but his favourite expression (or so it seems). To Behnoosh for telling me about Iran and suggesting to relatives and friends to go and visit Italy once you are in Europe.
Andy and Fenghui, because they too are an amazing couple. I hope distance will not create problems in the future. To Andy for being a British gentleman in Zurich, for always encouraging me. For being so kind despite dealing with rough stuff all day long. For his marvellous course. For making me appreciate the sense of competitiveness between Oxford and Cambridge. To Fenghui for sharing his sense of wonder with the rest of us. I have to admit: it was pretty hilarious. Thank you for your sincerity and exuberance. For being such a good sporty.
All four of you will be leaving soon. I shall miss you. But I hope we can talk and see each other around the world once in a while.

Thanks to the others who have already left, but left some of their wisdom behind.
John, for the conference we went to together in Naples, for the good time we spent together there as well as here. For your infinite kindness and humility. For speaking at least six languages fluently, but never bragging about it. For being a great actuary, but always finding time for others. Maybe we can even meet at work.
Daniel B., for the lunches spent talking about politics, Europe and our home countries. For the afternoons spent in the gym and for contacting me to talk about a possible job. I hope you will also contact me in the future. And that we will organise more dinners together.
Andrea, because I still haven't managed to understand how a "Argauer" can be an ardent Lazio fan, even though he is the calmest and most placid person I know. For learning Italian like it was his first language.
Sara, for her kindness and cheerfulness. It was a pity we did not manage to go to London together for the conference. I hope to see you again in Pisa. For being a rare example of a Swiss person moving to Italy for work.
Xi, for your helpfulness and practicality. The first day I started my PhD we went to eat together at the Asian canteen with Vincenzo. Thank you for explaining to me a bit how this microcosm worked.
Michel, for introducing me to the tasty world of Belgian beers, for all the times we came home on the same bus, for the dinners at the Alehouse talking about energy, finance, history and more.
Special thanks to Wahid as well. It is a pity I can no longer come to your office at weekends, knowing you were always there to work, to take a break and talk about politics and maths. I am sure you miss us at least as much as we miss you. I got to know you better since our first meeting in Lausanne. At first I thought you were a bit difficult, but I actually found out that you are tender than a loaf of bread. I hope you will have in London the career you deserve after all your hard-work.
Giulia, because you were only with us for the duration of one summer, but it was like a hurricane sweeping over us. Because you are brilliant and imaginative. I am sure we will still see each other from time to time somewhere. Maybe we will even get to work together on something.
Philippe, for the interesting talks, for telling us about Canada, the US and Citadel. And for his great performance in the SOLA run.
Ariel, for being a nice mate in Oberwolfach.
Thanks also to those I have known less, but who will always be part of my image over these years.

So I would like to remember Marvin, Lukas, Balint and Gudmund, but also Cosimo, Thomas and David.

I am sure that the group will remain 'strong' and close in the years to come. This is mainly due to the rookies (newcomers). Marco, for all the jokes we play on each other, for his Italian with dialectal influences, for the trip to Sarnen and for teaching us that we should never go to Trashwalden. When you look at me, watch your eyes not to look down, or you pay for the move. I still have some good music for you to discover.
Daniel K., for being a worthy co-co-driver on the trip to Sarnen. For chatting and for his good mood. For his friendliness.
Songyan, who has only just arrived, but already has his hands on in many projects and I am sure he will complete them all.
Chiara R., for her sweetness, sensitivity, helpfulness and friendship, encouraged by the common language. You are so strong. Thank you also for all the chats during coffee breaks in your office. For being a scout, for opening up at the end and becoming more friends. I really hope our friendship will be long-lasting.
I wish all of you the best for your Ph.D. path and hope you will enjoy as much as I did.

Finally, the other friends that are somehow linked to this Ph.D. years.
Marco, for being one of my best friends. For having shared short holidays and long talks. I hope his new Taiwanese adventure can bring him all the satisfaction he deserves. But I also hope he finds the time to return to Europe from time to time. For making me realise what it means to do a Ph.D. and for hosting and showing me Geneva. For coming to Parma to celebrate the passing years. I told you I would give you the hospitality of a king, and you replied that it was that of an emperor.
Chiara G., because she picked me up in a very difficult time and made me find the right path again. For her fierce fights for climate. As I have already written to her, who would have thought that since the distant summer of 2008 we would still be in touch? Because she is the reason I ended up first in Lausanne and then in Zurich. Because she is a true friend and she showed me that. I hope you will find your own new way.
Fiona, because she is so nice, so funny, also polyglot and always a pleasure to talk to. It was so nice that we met again after the EPFL years. Thanks also for the lunch you offered me at your place!

Of course, there are plenty of other friends who deserve to be mentioned. Especially those I meet again in Parma, my home-town. Knowing that I will see them again makes the return journey much more pleasant and lighter. Since there are too many of you, I won't try to list you all, because I would surely make some mistakes. And also because time is running out and the text risks becoming boring.

We are approaching the finale.
Eventually, I would also like to thank Francesca, who is my best friend and partner. I know that sometimes it is not easy to stay with me and that there can be ups and downs. But we can always solve everything together, and this is what counts the most. For your freckles, your eyes and your mind, which have intrigued me from the very first day. Because you give meaning to the passing of time.

This might also be the last time I can publicly thank my family, so I definitely want to take

this opportunity to say thank you to my mother Caterina and my father Aldo. Needless to say, without them I would not be where I am now. Nor would I be what I am now. Their advice has always been among the most valuable and sought-after, and I think it will continue to be like that for a long time to come. I would especially like to thank them for all the sacrifices they have made to give me all the opportunities I have had. It may sound obvious, but it really is not. One day, I only hope to be as exemplary a parent as they have been.
Thanks also to my sister Benedetta for all the times she put up with me, maybe more than she could, certainly more than she had to. I have always tried to be a good example for you, but I see that now you are starting to set it for me. Now you have graduated, you have also started working and have become a young woman. Know that you can always count on your big brother.
Finally, I would like to mention my grandparents, Maria Luisa and Anselmo, who are not here anymore, and Rita and Giovanni. Your lives are for me the clearest instance of what it means to be good people. You have always taught me everything through examples. For this I will be eternally grateful.

I always tried to be the best version of myself.
If I managed and you are still reading, it is likely that I also have to thank you.

Matteo Zurich, 18.05.2022