POLITECNICO DI MILANO
Corso di Laurea Magistrale in Ingegneria Matematica
Dipartimento di Matematica

Una tecnica image-based di Deep Learning per la calibrazione di superfici di volatilità

Relatore: Daniele Marazzina

Tesi di Laurea di:
Ariel Nacamulli, matricola 912992

Anno Accademico 2019-2020

A chi mi ha sempre sostenuto,
a chi ha creduto in me,
dedico a voi questo traguardo

Sommario

Nell'industria finanziaria, come in altri settori, la scelta dei modelli in uso dipende in larga misura dalla loro usabilità. I modelli attuali non hanno necessariamente le proprietà più adatte a descrivere i mercati, quanto invece sono quelli con i costi computazionali più facilmente sostenibili, e dunque più facilmente utilizzabili. Lo scopo di questa tesi è studiare un metodo capace di ridurre drasticamente i costi computazionali di calibrazione dei modelli più complessi al fine di rendere fruibili anche i modelli normalmente più dispendiosi dal punto di vista dei tempi computazionali. Tale risultato è possibile grazie all'impiego delle reti neurali. Nella tesi è anche mostrato come la tecnica può essere facilmente estesa con successo ad altri modelli portando l'esempio di due modelli basati su processi auto similari.

Ringraziamenti

In primis, vorrei ringraziare il prof. Daniele Marazzina non solo per aver accettato di farmi da Relatore, ma soprattutto per la disponibilità e i preziosi consigli.

Vorrei ringraziare i miei genitori per avermi sostenuto emotivamente e materialmente durante i miei studi, incoraggiandomi e aiutandomi nelle scelte più difficili. Grazie a mio fratello per la compagnia, la possibilità di confronto e la ritrovata motivazione a dare sempre il massimo.

Grazie a Dana che mi ripete ogni giorno quanto crede in me. Grazie per avermi dato una carica di fiducia per ciò che faccio.

Grazie alla mia famiglia, ai miei nonni, Miriam, Bruno, Mariella e Pacifico, che hanno sempre creduto nelle mie capacità e festeggiato i miei successi. Grazie ai miei zii e cugini, lontani o vicini.

Grazie alla mia seconda famiglia, gli amici del Gruppone, sempre presenti nei momenti importanti.

Grazie agli amici dell'UGEI, che per me è stata una palestra di vita come nient'altro. Grazie per la vostra amicizia.

Vorrei infine ringraziare i miei amici del Politecnico, grazie per aver condiviso con me anche solo una parte di questo percorso.

Indice

Sommario **1**

Ringraziamenti **3**

1 Introduzione **9**
1.1 Inquadramento generale . . . 9
1.2 Breve descrizione del lavoro svolto . . . 10
1.3 Struttura della tesi . . . 12

2 Stato dell'arte **13**

3 Nozioni di Finanza Quantitativa **17**
3.1 Nozioni generali . . . 17
3.1.1 Ipotesi di base . . . 17
3.1.2 Derivati . . . 18
3.2 Modelli in esame . . . 24
3.2.1 Heston . . . 25
3.2.2 Rough Bergomi . . . 25
3.2.3 Bergomi . . . 25
3.2.4 Modelli auto similari . . . 26
3.3 Tecniche di pricing . . . 27
3.3.1 Monte Carlo . . . 27
3.3.2 Metodo di Carr-Madan . . . 28

4 Nozioni di Artificial Intelligence **31**
4.1 Artificial Neural Networks . . . 31
4.2 Concetti di base . . . 31
4.3 Struttura della rete neurale . . . 31
4.3.1 Layers . . . 32

4.3.2 Neurons . . . 32
4.3.3 Learning . . . 33
4.3.4 Algoritmo di back-propagation . . . 34
4.3.5 Parametri di calibrazione . . . 35
4.4 Utilizzo della Rete Neurale . . . 36

5 Progetto logico della soluzione del problema 39
5.1 Pricing e calibrazione con reti neurali . . . 39
5.1.1 Inverse mapping . . . 39
5.1.2 Utilizzo del pricing . . . 40
5.2 Dataset . . . 42
5.3 Rete neurale . . . 43
5.3.1 Rete a 3 hidden layer . . . 44
5.3.2 Rete a 4 hidden layer . . . 45
5.3.3 Parametri di ottimizzazione . . . 46
5.4 Problema di minimizzazione . . . 47

6 Implementazione della soluzione 49
6.1 Creazione del dataset . . . 49
6.1.1 Funzioni di pricing . . . 49
6.1.2 Volatilità implicita . . . 51
6.1.3 Costruzione del dataset . . . 51
6.2 Rete neurale . . . 52
6.2.1 Calibrazione dei parametri . . . 53

7 Risultati e commenti 55
7.1 Analisi sul numero di hidden layer . . . 55
7.1.1 Rough Bergomi . . . 56
7.1.2 Rough Bergomi con approssimazione costante a tratti . 57
7.2 Modelli con rete a 3 hidden layers . . . 59
7.2.1 Rough Bergomi . . . 59
7.2.2 VGSSD . . . 61
7.3 Modelli con rete a 4 hidden layers . . . 62
7.3.1 Rough Bergomi . . . 64
7.4 Commenti . . . 66

8 Direzioni future di ricerca e conclusioni 69
8.1 Conclusioni 69
8.1.1 Limiti del lavoro 70
8.2 Direzioni future di ricerca 70
8.2.1 Scelta del modello più adatto 70
8.2.2 Un unico metodo di calibrazione 74

A Risultati per i modelli non riportati 79
A.1 Analisi sull'architettura utilizzata 79
A.2 Modelli con rete a 3 hidden layers 80
A.2.1 1-Factor Bergomi 80
A.2.2 Heston 82
A.2.3 NIGSSD 84
A.3 Modelli con rete a 4 hidden layers 87
A.3.1 1-Factor Bergomi 87

B Codice utilizzato 91
B.1 Creazione del dataset 91
B.1.1 Pricing Carr-Madan 91
B.1.2 Pricing Monte Carlo 93
B.1.3 Analytic 95
B.1.4 Pricing Complessivo 95
B.1.5 Volatilità Implicita 97
B.1.6 Script di generazione 98
B.2 Implementazione del processo di calibrazione in due passi . . . 100
B.2.1 Passo 1: rete neurale 100
B.2.2 Passo 2: calibrazione 103

C Composizione del Dataset 107
C.1 rBergomi 107
C.2 1-Factor Bergomi 108
C.3 Heston 108
C.4 VGSSD 109
C.5 NIGSSD 109
C.6 rBergomi Piecewise Constant 110
C.7 1-Factor Bergomi Piecewise Constant 111

D Rete neurale per l'individuazione del modello a minimo errore di calibrazione **113**
D.1 Struttura della rete neurale 113

Capitolo 1

Introduzione

1.1 Inquadramento generale

Con l'avvento dei moderni computer, l'innovazione finanziaria va di pari passo con l'evoluzione tecnologica. Pertanto è naturale che gli strumenti del campo dell'intelligenza artificiale trovino applicazione nell'affrontare i problemi della finanza moderna. Nell'industria finanziaria, come in altri settori, la scelta dei modelli in uso dipende in larga misura dalla loro usabilità. È chiaro che i modelli che hanno a disposizione solo funzioni di pricing lente siano esclusi dall'uso pratico a favore di modelli con pricing più veloci, che sono quindi adottati. In questo senso arriva l'aiuto delle *Deep Neural Network* (o *Artificial Neural Network* o, semplicemente, *Neural Network*, in italiano rete neurale), un potente strumento efficace in numerosi campi di applicazioni di intelligenza artificiale, come il riconoscimento immagini o l'elaborazione del linguaggio naturale. Per dare un'idea di cosa siano le reti neurali, possiamo vederle come un approssimatore universale, uno strumento capace, con una certa dose di abilità dell'utilizzatore, di approssimare arbitrariamente qualsiasi funzione non lineare. In particolare le recenti applicazioni delle reti neurali permettono di eseguire l'approssimazione offline di funzioni di pricing particolarmente dispendiose in termini di tempo computazionale. Questo permette di eseguire la successiva calibrazione dei modelli sui dati di mercato in pochi millisecondi, superando il limite causato dalla lentezza dei pricing di questi modelli. Lo scopo di questa tesi è di presentare la tecnica menzionata, applicata a diversi modelli di pricing. In particolare presenterò

un'applicazione su modelli finanziari derivati dalla famiglia dei processi auto similari. Questi modelli hanno il vantaggio di identificare una superficie di volatilità con solo quattro parametri. La tecnica di pricing utilizzata per il pricing dei derivati modellizzati con processi auto similari è quella sviluppata utilizzando la *Fast Fourier Transform*, che garantisce un pricing abbastanza rapido. Tuttavia, il metodo che presenterò assicura un pricing ancora più veloce, riducendo significativamente, di conseguenza, il tempo di calibrazione. Come sarà dettagliato nei prossimi capitoli, viene impiegata una struttura di rete neurale abbastanza semplice per imparare la mappa tra i parametri del modello e la superficie di volatilità. Con questa conoscenza, diventa molto più veloce calibrare i parametri sulle superfici di volatilità.

1.2 Breve descrizione del lavoro svolto

Le reti neurali trovano applicazione in numerosi campi della finanza. Esempi rilevanti nei sottocampi della valutazione derivati e della calibrazione delle superfici di volatilità si trovano, per esempio, nei lavori di Bayer e Stemper [4] e Stone [45] che presentano un metodo per calibrare i modelli cosiddetti stochastic rough volatility, di Hernández [24], che introduce la possibilità di superare il problema del costo computazionale del pricing con una tecnica di calibrazione alternativa, e in Buehler, Gonon, Teichmann e Wood [9] dove si introduce un framework per la costruzione di un portafoglio di copertura di derivati in presenza di frizioni di mercato, tutti utilizzando tecniche di deep learning. Altre tecniche moderne, come il reinforcement learning, sono impiegate nella risoluzione di problemi di calibrazione, come in Spiegeleer, Madan, Reyners e Schoutens [44], a sostegno del fatto che le tecniche impiegate nel cosiddetto campo dell'intelligenza artificiale possono essere usate per affrontare problemi di calibrazione.

L'idea di questa tesi è di esplorare questo campo, cercando di riprodurre i passi principali di questa forma alternativa di calibrazione, seguendo principalmente il lavoro proposto da Horvath, Muguruza e Tomas [30], con lo scopo di aggiungere nuovi membri al paniere di modelli a cui questa tecnica è applicata. In particolare, il lavoro di Carr, Geman, Madan e Yor [10] serve allo scopo. Nell'articolo, sono presentati sei modelli di pricing, basati su processi auto similari di incrementi indipendenti con legge auto decomponibile,

capaci di sintetizzare il pricing delle opzioni europee. Di questi modelli, due sono presentati in questa tesi.

Affrontando il problema, diventano visibili alcune peculiarità della tecnica, come ad esempio come affrontare la calibrazione degli iperparametri delle reti neurali, come preprocessare i dati o come produrre un dataset di training affidabile. Queste e altre questioni, che saranno ampiamente discusse nei prossimi capitoli, derivano dai principi di base della progettazione delle reti neurali introdotti in Goodfellow, Bengio e Courville [23], insieme a una buona dose di senso finanziario, proveniente dalle intuizioni di Horvath, Muguruza e Tomas [30] e Carr, Geman, Madan e Yor [10].

Un tema centrale che è importante discutere è che la tecnica presentata segue l'idea di mappare direttamente i parametri del modello in superfici di volatilità implicita. L'idea non è nuova, come già introdotto in Gatheral [19] e Gatheral e Jacquier [20], esaminando esplicitamente una rappresentazione parametrica delle superfici di volatilità implicita. Pur supportati da questa idea, la metodologia presentata procede in modo molto più tradizionale, con una composizione in due passi di due mappe diverse, in primo luogo, mappando i parametri in una "superficie di prezzo", cioè una griglia di prezzi, e quindi convertendo i prezzi nella relativa volatilità implicita di Black-Scholes.

Il lavoro svolto nella tesi vuole essere un ulteriore tassello a supporto della tecnica di calibrazione con reti neurali presentata. Infatti, le architetture delle reti neurali sono di solito molto specifiche per il problema, nel senso che sono costruite per affrontare una specifica sfaccettatura del compito per cui sono utilizzate, pertanto, mancano di riusabilità. Tuttavia, i modelli di pricing con approssimativamente lo stesso numero di parametri sembrano avere una complessità simile e possono quindi essere affrontati utilizzando una metodologia di costruzione della rete precisamente definita. In questo senso, il codice di questo lavoro è implementato in Python 3.7, ed è disponibile qui `https://github.com/arielNacamulli/NNCalibration_Surface`. Un limite evidente è che la tecnica dipende dalla qualità dei dati disponibili, siccome una rete neurale, come altre tecniche di apprendimento automatico, funziona bene solo se supportata da dati di buona qualità. Un ulteriore limite è che il metodo è stato testato su un numero relativamente piccolo di modelli, tutti all'interno di un range specifico in termini di numero di parametri (4-5 parametri, oltre a due modelli da 11 parametri), e, anche se sulla carta le funzioni implementate producono un output, il fatto che l'output possa essere

adeguato per un modello completamente diverso deriva solo da una deduzione empirica e non può essere dimostrato per qualsiasi scenario. Tuttavia, con il sostegno dell'idea in Hornik, Stinchcombe e White [28] e Hornik, Stinchcombe e White [26], l'esistenza di un'architettura di rete neurale adatta ad ogni modello finanziario di pricing non è in discussione. Il problema che è ancora aperto, e che questa tesi vuole contribuire ad affrontare, è se sia possibile costruire una metodologia per costruire reti neurali e impostare iperparametri in modo che l'unico passaggio tra un dataset ben costruito e l'identificazione di una mappatura tra parametri del modello e superficie di volatilità sia la pressione di un pulsante.

1.3 Struttura della tesi

La tesi è organizzata come segue:

Nel Capitolo 2 è presentato lo stato dell'arte, per quanto riguarda alcuni modelli di pricing, l'utilizzo di metodi di intelligenza artificiale in finanza e le tecniche impiegate nei problemi di calibrazione.

Nel Capitolo 3 sono introdotte le principali nozioni di finanza quantitativa utili per lo scopo della tesi.

Nel Capitolo 4 sono introdotte le principali nozioni dei metodi di intelligenza artificiale utilizzati nella tesi.

Nel Capitolo 5 è descritto l'approccio proposto e sono dettagliati i passaggi fondamentali del metodo.

Nel Capitolo 6 è presentato il codice utilizzato nella soluzione del problema, con un focus sui passaggi principali.

Nel Capitolo 7 sono presentati e commentati i risultati numerici.

Nel Capitolo 8 sono suggeriti i potenziali lavori di ricerca futuri e vengono illustrate le conclusioni.

Capitolo 2

Stato dell'arte

Questa sezione ha lo scopo di presentare lo stato dell'arte per quanto riguarda le tecniche numeriche impiegate nei problemi di calibrazione della superficie di volatilità e i principali argomenti che sfiorano questa tematica.

Per decenni la scelta del modello finanziario da adottare è dipesa fortemente dalla tecnica di pricing disponibile per il modello. Di conseguenza, in assenza di un pricing veloce per il modello, questo viene escluso dalle applicazioni industriali. Un esempio di questo fatto sono i modelli *rough volatility*, che, nonostante vantaggiose caratteristiche, come presentato in Gatheral, Jaisson e Rosenbaum [21], in cui è evidenziato come i modelli di questa classe, supportati da dati ad alta frequenza, siano particolarmente coerenti con le serie storiche, o come riportato in Gatheral, Jaisson e Rosenbaum [21], dove è analizzato il modello rough Bergomi, mostrando la coerenza del modello con i dati storici, anche per periodi di forte stress sui mercati. Tuttavia questi modelli, basandosi sul pricing Monte Carlo che è computazionalmente costoso, risultano limitati nelle loro applicazioni industriali.
Una parte dello studio è rivolta proprio ad un modello rough volatility, in particolare il modello rough Bergomi, descritto in 3.2.2. La principale caratteristica dei modelli *rough volatility* è il processo aleatorio che guida la dinamica, ovvero un moto browniano frazionario (*fractional Brownian motion (fBm)*), una generalizzazione del tipico Wiener process, in cui gli incrementi non sono necessariamente indipendenti. Altri modelli analizzati sono i modelli 1-Factor Bergomi e Heston, come descritti rispettivamente in Heston [25] e Bergomi [5], entrambi già analizzati in Horvath, Muguruza e Tomas [30]. Sono infine studiati due modelli auto similari. I modelli presentati in

Carr, Geman, Madan e Yor [10], due dei quali sono presentati in questo studio, hanno il notevole vantaggio di individuare una superficie di volatilità con unicamente quattro parametri, tre dei quali, come in altri modelli ([38, 2]), individuano lo smile, e sono quindi relativi alla varianza, alla skewness e alla curtosi. L'ultimo parametro, che rappresenta l'effetto dell'orizzonte temporale sulla distribuzione, è la vera novità del modello. Tutti questi modelli sono presentati brevemente nel Capitolo 3.

Al contrario di quanto avvenuto per i modelli rough volatility, la diffusione di altri modelli si deve soprattutto alle vantaggiose tecniche di pricing disponibili. Si pensi alla convenienza computazionale del pricing basato su FFT [11] che è in buona parte responsabile della popolarità di quei modelli che si basano su questa tecnica, come il modello di Heston [25]. Ancora più sorprendente è il duraturo successo del modello Black-Scholes (e Merton) [7]. Nonostante le numerose imperfezioni del modello, non ultima la sua incapacità di replicare la volatilità di mercato, la semplice e computazionalmente efficiente formula chiusa gli ha assicurato una solida presenza nelle applicazioni industriali, tanto da rendere il modello uno standard di mercato. Il concetto di volatilità implicita di Black-Scholes, cioè il valore di volatilità che rende il prezzo del modello di Black-Scholes uguale al prezzo di mercato, è un punto cruciale anche in questo lavoro.

La particolarità del metodo presentato è quella di fornire un mapping diretto tra i parametri del modello scelto e la superficie di volatilità associata, utile per la fase di calibrazione. Pertanto, sebbene tutti i modelli considerati [25, 5, 29, 10] siano nei fatti dei modelli di pricing, il metodo non coinvolge direttamente i prezzi associati alle diverse combinazioni di parametri, quanto piuttosto la volatilità implicita di questi prezzi. Il primo passo dell'analisi è quindi quello di sviluppare una funzione che, una volta scelti i parametri, individua una superficie di volatilità. L'idea di un modello parametrico sulla volatilità implicita non è nuovo. Ne è un esempio lo studio già presente in Gatheral [19], in cui è proposta una rappresentazione parametrica della superficie di volatilità implicita, e nel lavoro seguente Gatheral e Jacquier [20], in cui la tematica è approfondita con il supporto di analisi numeriche. Oltre ad essere già stata sperimentata, l'idea di mappare i parametri direttamente in una superficie di volatilità permette di mantenere la trattazione più generale, esulando da parametri quali il fattore di sconto e il prezzo spot.

Il metodo presentato fa uso di un metodo computazionale relativamente moderno: le reti neurali. Le reti neurali sono un potente strumento utilizzato nei sistemi di intelligenza artificiale, in campi come il riconoscimento immagini o l'elaborazione del linguaggio naturale (Natural Language Processing, NPL) [23], e sono brevemente discusse nel Capitolo 4. Essendo un valido metodo computazionale, trova impiego in diversi campi della finanza quantitativa (e.g. in Luo, Wu e Wu [37], Alonso e Srivastava [1] e Sirignano, Sadhwani e Giesecke [43]). In Hornik, Stinchcombe e White [28] è dimostrato l'Universal approximation theorem, ovvero la capacità di una famiglia di reti neurali di approssimare una funzione (supportata anche da quanto mostrato in Shaham, Cloninger e Coifman [41] e Hornik [27]) e analogamente in Hornik, Stinchcombe e White [26] è presentato l'Universal approximation theorem for derivatives, un teorema analogo per le derivate della funzione da approssimare. Questi teoremi affermano fortemente la capacità delle reti neurali di rappresentare una qualsiasi funzione non lineare, proprietà che ha fatto si che nel campo della valutazione delle opzioni esistessero applicazioni già dagli anni 1990, essendo l'idea già presente in Malliaris e Salchenberger [39]. In tempi più moderni, tecniche analoghe sono utilizzate anche per il pricing di derivati più complessi, come le opzioni americane, per cui in Gaspar, Lopes e Sequeira [18] è presentata una tecnica risolutiva. Relativamente al tema specifico della calibrazione delle superfici di volatilità implicita, esistono già numerosi studi. L'idea presente in Hernández [24], che sarà ripresa nel corso di questa tesi, è quella di utilizzare la rete neurale come approssimatore, al fine di ottenere una calibrazione rapida. Non è tuttavia l'unica tecnica alternativa utilizzata per la calibrazione della volatilità implicita, come si vede in Spiegeleer, Madan, Reyners e Schoutens [44]. Per quanto concerne i modelli rough volatility, i lavori presentati in Stone [45] e Bayer e Stemper [4] forniscono chiari argomenti a supporto dell'idea che metodi di deep learning possono essere efficaci nella calibrazione di questi modelli.

Capitolo 3

Nozioni di Finanza Quantitativa

Prima di illustrare il lavoro svolto, è necessario introdurre alcuni concetti che verranno utilizzati nel corso della tesi. Questo e il prossimo capitolo saranno dedicati all'introduzione dei principali concetti utilizzati rispettivamente nei campi della finanza quantitativa e dell'intelligenza artificiale.

3.1 Nozioni generali

3.1.1 Ipotesi di base

In questa sezione sono riassunte brevemente le principali ipotesi necessarie per definire il quadro teorico. Queste sono ipotesi di base per quasi tutto il lavoro nella finanza quantitativa e sono, per la maggior parte, ipotesi molto blande. Lo scopo di questa sezione è di dare una panoramica della teoria, una spiegazione più dettagliata può essere trovata su Bjork [6] o Hull [31].

Asset Price and Portfolio

Indicheremo il prezzo di un asset i al tempo t con S_t^i (o semplicemente S_t, se si considera un solo asset). Se inteso in uno specifico scenario futuro ω_j, utilizzeremo $S_t^i(\omega_j)$. L'ipotesi di base è che i prezzi degli asset siano (strettamente) positivi.

Con questa notazione, un portafoglio h di N asset è definito come un vettore $h = [h^1, ..., h^N]$, dove h^i rappresenta l'ammontare investito nell'asset

i. Quindi, il valore del portafoglio h al tempo t è

$$V_t^h = \sum_{i=1}^{N} h^i S_t^i. \tag{3.1}$$

Non arbitraggio

Un'assunzione di base è la condizione di non arbitraggio. Ciò significa che non deve essere possibile alcun arbitraggio definito come

$$V_0^h = 0, \ P(V_1^h \geq 0) = 1, \ P(V_1^h > 0) > 0. \tag{3.2}$$

In parole povere, ciò significa che è impossibile ottenere un profitto sicuro senza correre alcun rischio. Questa ragionevole assunzione permette di dimostrare alcuni teoremi di base su cui è costruito il quadro teorico. Qui l'idea di nostro interesse è che i prezzi che vediamo sui mercati finanziari devono essere privi di arbitraggio. Questo ci permette di definire un ultimo concetto di base.

Calibrazione

L'ipotesi di non arbitraggio è fondamentale nell'individuazione di un modello capace di prezzare gli strumenti finanziari. Affinché non si presentino incongruenze, i prezzi a modello devono rispecchiare le reali condizioni di mercato. Generalmente, i modelli di pricing dipendono da uno o più parametri. Questi parametri rappresentano, in un certo senso, le condizioni di mercato. Infatti, il processo di calibrazione consiste nella messa a punto dei parametri del modello, utilizzando i prezzi di mercato come dati. L'obiettivo è quello di replicare i prezzi di mercato o, se le specifiche del modello non lo permettono, di minimizzare una qualche forma di distanza tra i prezzi di mercato e quelli del modello. Una descrizione più dettagliata del processo si trova nel Capitolo 5.

Nei paragrafi seguenti sono presentati alcuni strumenti finanziari e i modelli necessari per prezzarli.

3.1.2 Derivati

I derivati finanziari sono contratti il cui prezzo dipende dai movimenti di mercato di altri asset. In modo più formale, come presentato in [6],

Definizione 3.1.1. *Un derivato finanziario è una qualsiasi variabile stocastica X della forma $X = \phi(Z)$, dove Z è la variabile stocastica che guida il processo di prezzo delle azioni.*

Ci concentreremo in particolare sulle opzioni. Le opzioni in finanza sono contratti che danno il diritto, ma non l'obbligo, di eseguire qualche scambio di denaro e/o di beni. Le opzioni possono avere le caratteristiche più varie, le opzioni di base (conosciute anche come "opzioni vanilla") sono le opzioni call e put europee.

Opzioni vanilla

Come menzionato, le opzioni call e put sono le opzioni di base e maggiormente scambiate sui mercati. I seguenti esempi, come il lavoro svolto in questa tesi, si riferiscono al mercato azionario. Tuttavia lo stesso vale anche per altre tipologie di asset. Un'opzione call (europea) è un contratto che dà il diritto (ma non l'obbligo) di comprare una certa azione, ad una scadenza definita (chiamata "maturity"), pagando un prezzo definito (chiamato "prezzo di esercizio" o "strike"). Spesso indicheremo la maturity come T e lo strike come K. Con questa notazione, il payoff di un'opzione call è

$$[S_T - K]^+ \tag{3.3}$$

dove $[\,\cdot\,]^+$ rappresenta l'operazione di parte positiva.

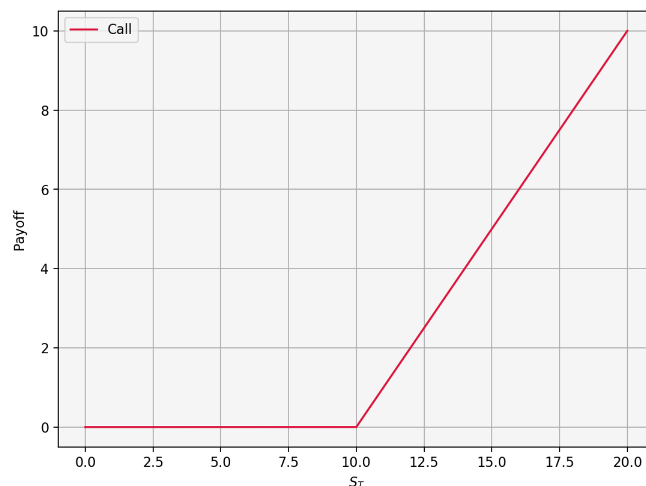


Figura 3.1: Rappresentazione illustrativa del payoff di una call, avente $K = 10$

Viceversa, un'opzione put è un contratto che permette al proprietario di vendere lo stock al prezzo di esercizio. Quindi, il payoff è

$$[K - S_T]^+. \tag{3.4}$$

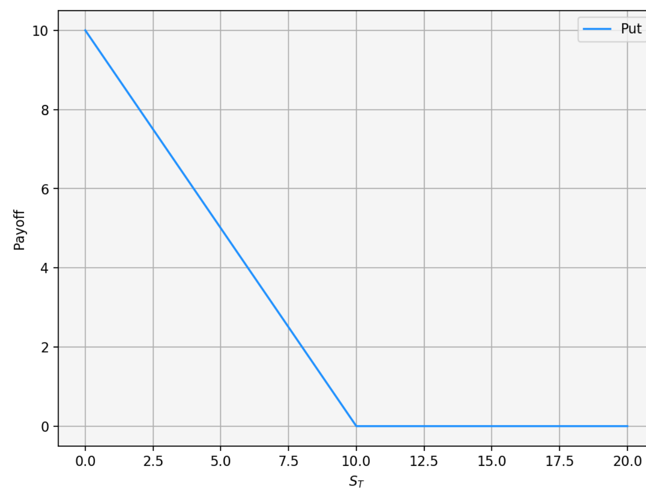


Figura 3.2: Rappresentazione illustrativa del payoff di una put, avente $K = 10$

Un'opzione call il cui valore sottostante è al di sopra dello strike è chiamata *in-the-money* (ITM), viceversa si dice *out-of-the-money* (OTM). Un'opzione put il cui valore del sottostante è al di sopra dello strike si chiama *out-of-the-money* (OTM), viceversa si dice *in-the-money* (ITM).

Per semplicità indicherò il prezzo della call e della put rispettivamente come $c(x, t)$, $p(x, t)$, dove t è il momento in cui si valuta l'opzione e x è il prezzo del sottostante al tempo t. Se necessario, per maggiore chiarezza, queste scritture potrebbero essere integrate con parametri specifici (ad esempio $c(x, t; T, K)$ rappresenta un'opzione call con scadenza in T, con strike K, valutata al tempo t). Intuitivamente, si può mostrare graficamente che, chiamata T la maturity delle opzioni, K lo strike, $c(S_T, T) - p(S_T, T) + K = S_T$.

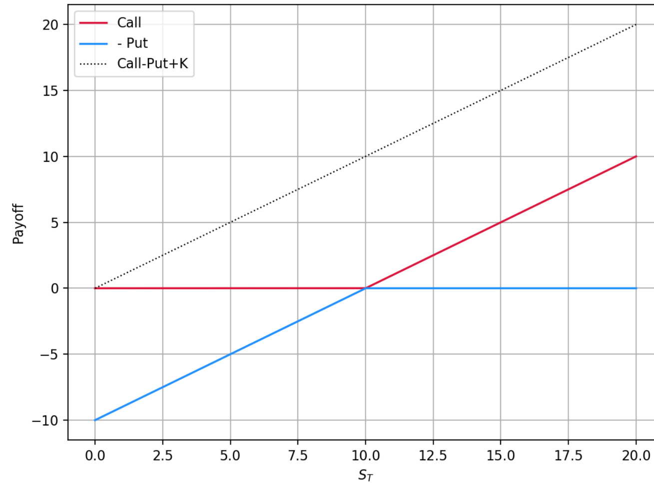


Figura 3.3: Rappresentazione grafica dell'uguaglianza $c(S_T, T) - p(S_T, T) + K = S_T$, *con* $K = 10$

Si può intuire la seguente proposizione

Proposizione 3.1.1 (Put-Call Parity)**.** *Data una call (europea) e una put (europea) con strike price* K *e maturity* T*, vale la seguente formula:*

$$c(S_t, t) - p(S_t, t) = S_t - Ke^{-r(T-t)}. \tag{3.5}$$

Modello di Black-Scholes

Uno dei primi modelli introdotti, e tutt'oggi utilizzato per il pricing dei derivati, è il modello di Black-Scholes [7]. Si basa sull'assunzione che la dinamica del sottostante (che non paga dividendi, anche se questa ipotesi può essere allentata con piccoli cambiamenti nel risultato) segua un moto Browniano geometrico (Geometric Brownian Motion, o GMB) che può essere modellizzato come

$$\begin{cases} S_0 = s \\ dS_t = \mu S_t dt + \sigma S_t dW_t \end{cases} \tag{3.6}$$

con σ chiamata "volatilità" del sottostante e W_t processo di Wiener. Spesso, la dinamica di un'azione sotto il modello di Black-Scholes è riportata nella

sua cosiddetta formulazione neutrale al rischio

$$dS_t = rS_t dt + \sigma S_t dW_t^{\mathbb{Q}} \tag{3.7}$$

che segue dall'assunzione che per il mercato valga il non arbitraggio.
Si può dimostrare che un'opzione call può essere prezzata come

$$c(S,t) = S\mathcal{N}(d_1) - Ke^{-r(T-t)}\mathcal{N}(d_2) \tag{3.8}$$

con

$$d_1 = \frac{log(\frac{S}{K}) + (r - \frac{\sigma^2}{2})(T-t)}{\sigma\sqrt{T-t}},$$
$$d_2 = d_1 - \sigma\sqrt{T-t}.$$

Il modello si basa su ipotesi molto forti, come la volatilità σ costante (detta anche "flat"), e un tasso di interesse privo di rischio r costante. Per questo motivo nel corso del tempo sono stati studiati e implementati modelli più sofisticati. Tuttavia, questo modello rimane il punto di riferimento nella misura in cui i prezzi delle opzioni vanilla sono espressi in termini di volatilità di Black-Scholes (cioè la volatilità che rende il prezzo del modello uguale al prezzo di mercato, attraverso un processo di calibrazione).

Superficie di volatilità

Nel modello di Black-Scholes, volatilità σ tale da rendere il prezzo di modello uguale al prezzo di mercato, è detta volatilità implicita (di Black-Scholes). Dato che generalmente per uno stesso sottostante sono scambiate sul mercato opzioni con caratteristiche contrattuali diverse (in termini di maturity e strike), esistono contemporaneamente diversi valori di volatilità implicita, ognuno associato ad una diversa combinazione dei parametri contrattuali. Ciò che ne risulta è una corrispondenza, per ogni maturity e strike considerato, tra la coppia (T, K) e la relativa volatilità implicita. Tale corrispondenza può essere rappresentata da una superficie, come in Figura 3.4.

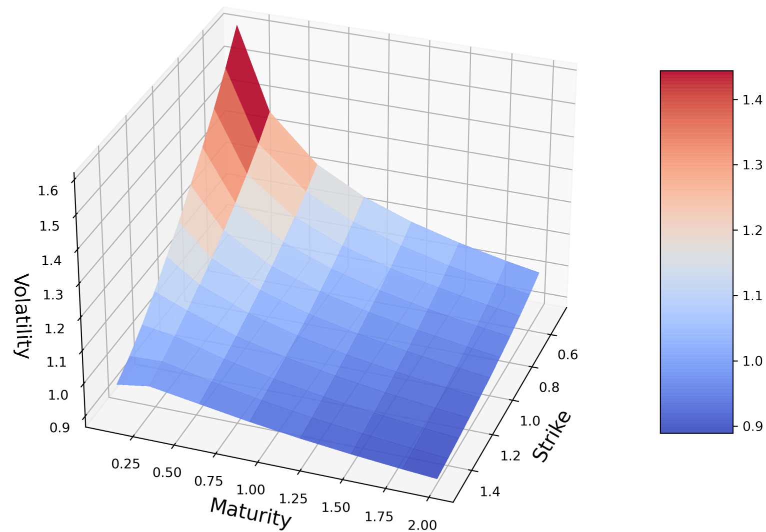


Figura 3.4: Rappresentazione di una generica superficie di volatilità

Si osservi che, dato che non è possibile invertire la funzione di pricing di Black-Scholes analiticamente rispetto alla volatilità σ, la volatilità implicita è generalmente calcolata tramite un qualche metodo di ottimizzazione non lineare.

Calibrazione di una superficie di volatilità

Data la nozione di calibrazione introdotta nella Sezione 3.1.1 e il modello di pricing di Black-Scholes descritto nella Sezione 3.1.2, è il momento di presentare come calibrare una superficie di volatilità, che è l'argomento principale della tesi. Calibrare una superficie di volatilità significa, generalmente,

risolvere il problema di minimizzazione

$$\min_{\theta \in \Theta} \delta\big(P^{Mkt}(\zeta), P(\mathcal{M}(\theta), \zeta)\big) \tag{3.9}$$

dove θ è l'insieme delle possibili combinazioni di parametri, ζ rappresenta i dettagli contrattuali delle opzioni, $\mathcal{M}$ rappresenta il modello selezionato, P e P^{Mkt} sono, rispettivamente, il prezzo delle opzioni sotto il modello selezionato e il prezzo di mercato delle opzioni. δ è una misura adeguata di distanza, tipicamente il Mean Squared Error (MSE):

$$\text{MSE}(x, y) = \frac{1}{n} \sum_{i=1}^{n} (x_i - y_i)^2 \tag{3.10}$$

Un'importante osservazione sui dati di mercato. Nella pratica, la calibrazione deve essere eseguita sulle opzioni maggiormente liquide, che sono quelle vanilla. Tuttavia, poiché sia le opzioni put che quelle call sono tipicamente disponibili per lo stesso strike e la stessa scadenza, solo la più liquida delle due viene usata per la calibrazione; tipicamente si tratta delle le opzioni OTM. Come discusso alla fine della sottosezione relativa al modello di Black-Scholes nella Sezione 3.1.2 , spesso i prezzi di mercato sono espressi in termini di volatilità di Black-Scholes. Questo significa che la selezione dei contratti più liquidi è fatta a monte. Inoltre, ai fini della calibrazione della volatilità, è più semplice, e ugualmente corretto, considerare unitario il prezzo iniziale sottostante e ipotizzare il tasso di interesse nullo, riducendo a un modello incentrato sulla volatilità. Un valore con i corretti parametri di prezzo iniziale e fattore di sconto può essere facilmente ricavato da questa forma semplificata.

3.2 Modelli in esame

In questa sezione introduciamo brevemente alcuni modelli più avanzati utilizzati in questa tesi. Per i modelli descritti anche in Horvath, Muguruza e Tomas [30] seguirò lo stesso standard nell'introdurli.

3.2.1 Heston

Il modello di Heston è presentato in Heston [25] e descritto dal sistema

$$
\begin{aligned}
dS_t &= \sqrt{V_t} S_t dW_t^{[1]}, \quad S_0 = s_0 \\
dV_t &= \kappa(\theta - V_t)dt + \sigma\sqrt{V_t} dW_t^{[2]}, per\ t > 0, V_0 = \xi_0
\end{aligned}
\tag{3.11}
$$

essendo $W^{[1]}$ e $W^{[2]}$ processi di Wiener con parametro di correlazione $\rho \in [-1, 1]$, $\kappa, \theta, \sigma, \xi_0 > 0$ e $2\kappa\theta > \sigma^2$. Nel framework presentato, come in [30], il modello è indicato con $\mathcal{M}^{Heston}(\theta)$, con $\theta = (\kappa, \theta, \sigma, \rho, \xi_0) \in \Theta^{Heston} \subset \mathbb{R}^5$.

3.2.2 Rough Bergomi

Il modello di rough Bergomi è presentato in Horvath, Jacquier e Muguruza [29] ed è descritto dal sistema

$$
\begin{aligned}
dX_t &= -\frac{1}{2} V_t dt + \sqrt{V_t} dW[1]_t, \quad X_0 = 0 \\
V_t &= \xi_0(t)\mathcal{E}\left(\sqrt{2H}\nu \int_0^t (t-s)^{H-1/2} dW_s^{[2]}\right), V_0 = v_0 > 0
\end{aligned}
\tag{3.12}
$$

dove $H \in (0, 1)$ è il parametro di Hurst, $\nu > 0$, $\mathcal{E}(\ \cdot\ )$ l'esponenziale stocastico di Doléans-Dade [14], $\xi_0(\ \cdot\ ) > 0$ la curva di varianza forward iniziale, e $W^{[1]}$ e $W^{[2]}$ processi di Wiener con parametro di correlazione $\rho \in [-1, 1]$. In questo framework, come in [30], il modello è indicato come $\mathcal{M}^{rBergomi}(\theta)$, con $\theta = (\xi_0, \nu, \rho, H) \in \Theta^{rBergomi} \subset \mathbb{R}^n$ per qualche $n \in \mathbb{N}$, in funzione del primo parametro. Si noti che, per limitazioni numeriche, $\xi_0(\cdot\ ) > 0$ è approssimato da una funzione costante a tratti.

3.2.3 Bergomi

Il modello di Bergomi a n fattori, come presentato in [5], è descritto dal sistema

$$
\begin{aligned}
dX_t &= -\frac{1}{2} V_t dt + \sqrt{V_t} dW_t^{[0]}, \quad X_0 = 0 \\
V_t &= \xi_0(t)\mathcal{E}\left(\eta_i \sum_{i=1}^{n} \int_0^t \exp(-\kappa_i(t-s) dW_s^{[i]})\right), V_0 = v_0 > 0
\end{aligned}
\tag{3.13}
$$

dove $\eta_1, ..., \eta_n > 0$, $W^{[0]}$ è un processo di Wiener, $W = (W^{[1]}, ..., W^{[n]})$ è un processo di Wiener correlato di dimensione n, $\mathcal{E}(\ \cdot\ )$ l'esponenziale stocastico, $\xi_0(\ \cdot\ ) > 0$ la curva di varianza forward iniziale, e $W^{[0]}$ e W hanno un parametro di correlazione $\rho\ in[-1, 1]$. In particolare, verrà utilizzato il modello Bergomi a 1 fattore. Nel mio framework, come in [30], il modello è indicato con $\mathcal{M}^{1FBergomi}(\theta)$, con $\theta = (\xi_0, \beta, \eta, \rho) \in \Theta^{1FBergomi} \subset \mathbb{R}^n$ per qualche $n \in \mathbb{N}$ dipendente dal primo parametro. Si noti che, per limitazioni numeriche, $\xi_0(\cdot\ ) > 0$ è approssimato da una funzione costante a tratti.

3.2.4 Modelli auto similari

Gli ultimi modelli analizzati, introdotti in Carr, Geman, Madan e Yor [10] sono basati su un processo additivo auto similare $Y(t)$ per descrivere il prezzo forward. In particolare, abbiamo che

$$F_t(T) = F_0(T)\frac{e^{Y(t)}}{\mathbb{E}_0[e^{Y(t)}]} = F_0(T)e^{f_t}.$$

Inoltre $Y(t)$ è uguale in legge a $t^\gamma X(1)$. I modelli in esame sono due, e si distinguono per la diversa scelta del processo $X(t)$.

VGSSD

Il processo Variance Gamma self-Decomposable (VGSSD), utilizza come processo $X(t)$ un processo variance gamma [38]. Ne risulta una funzione caratteristica uguale a

$$\Phi_Y(u) = (1 - iu\theta\nu t^\gamma + \frac{1}{2}\sigma^2\nu u^2 t^{2\gamma})^{-\frac{1}{\nu}}$$

dove $(\sigma, \nu, \theta, \gamma)$ sono i parametri caratteristici del modello che guidano, rispettivamente, la varianza, la skewness, la curtosi in eccesso e l'effetto dell'orizzonte temporale sulla distribuzione.

NIGSSD

Il processo Normal Inverse Gaussian self-Decomposable (NIGSSD), utilizza come processo $X(t)$ un processo normal inverse Gaussian [38]. Ne risulta una funzione caratteristica uguale a

$$\Phi_Y(u) = exp\Bigg(- \sigma\Bigg(\sqrt{\frac{\nu^2}{\sigma^2} + \frac{\theta^2}{\sigma^4} - \Big(\frac{\theta}{\sigma^2} + iut^\gamma\Big)^2} - \frac{\nu}{\sigma}\Bigg)\Bigg)$$

dove $(\sigma, \nu, \theta, \gamma)$ sono i parametri caratteristici del modello che guidano, rispettivamente, la varianza, la skewness, la curtosi in eccesso e l'effetto dell'orizzonte temporale sulla distribuzione.

3.3 Tecniche di pricing

Affinché il modello possa associare un'opzione ad un prezzo, sono state studiate tecniche numeriche. Questa sezione sarà dedicata alle tecniche di pricing utilizzate a margine dello studio.

3.3.1 Monte Carlo

I metodi Monte Carlo sono i più utilizzati nell'industria finanziaria. Il metodo si basa sulla simulazione numerica di vari scenari, basati sulle proprietà della dinamica neutrale al rischio del sottostante. Per i vari scenari, viene calcolato il payoff alla scadenza e quindi il payoff medio viene attualizzato per ottenere il prezzo dell'opzione. In formule:

$$\text{Discount} \cdot \mathbb{E}^{\mathbb{Q}}\Big[\text{Payoff in } T\Big] \tag{3.14}$$

dove $\mathbb{E}^{\mathbb{Q}}$ rappresenta il valore atteso sotto la misura neutrale al rischio. In pratica, prendiamo (3.7) come dinamica di esempio. Per un'opzione call con strike K e maturity T. Anche il tasso di interesse r, il valore spot S_0, la volatilità σ e un numero di scenari da calcolare N, devono essere definiti. Di seguito uno pseudo-codice di alto livello per il calcolo del prezzo dell'opzione:

Algorithm 1 B&S Monte Carlo

```
1: function CallPrice(S_0, K, T, σ, r)
2:     Z = vettore di N estrazioni casuali da una normale standard
3:     calcola S_T^i = S_0 · exp((r − σ²/2)T + σ√T Z^i) per ogni scenario i
4:     calcola payoff^i = [S_T^i − K]^+ per ogni scenario i
5:     price = e^{−rT} · mean(payoff)
```

Questo mostra il principale vantaggio del metodo Monte Carlo, e la ragione per cui è ampiamente utilizzato: la sua semplicità di implementazione. Infatti, anche per payoff più complicati è di solito abbastanza facile implementare l'algoritmo Monte Carlo adatto. Tuttavia, ci sono alcuni svantaggi.

Il primo e più importante (nonché l'argomento principale della tesi) è l'alto costo computazionale. Infatti, per calcolare un prezzo è necessario simulare N scenari, il che può essere costoso se N è grande (per dare un'idea, i valori tipici vanno da 10^5 a 10^7). Questo ci porta al prossimo inconveniente, che è il fatto che si tratta di un'approssimazione del prezzo. L'errore sull'approssimazione diminuisce come $\frac{1}{\sqrt{N}}$. Il numero corretto di simulazioni deve essere identificato in base al livello di precisione richiesto. Quindi, come spesso accade, c'è un trade-off tra accuratezza e tempo di calcolo.

Per quanto riguarda l'argomento principale della tesi, che è la calibrazione della superficie di volatilità, il metodo Monte Carlo è semplicemente troppo costoso a livello computazionale per affrontare il problema. Questo è dovuto al fatto che, se si devono calibrare P punti di una superficie, e ogni calibrazione richiede M iterazioni, il numero totale di scenari da simulare sarà $N \cdot M \cdot P$, che, data la dimensione di questi numeri, diventa computazionalmente non realizzabile.

3.3.2 Metodo di Carr-Madan

Come menzionato nella Sezione 3.1.2, la calibrazione della volatilità è tipicamente eseguita sulle opzioni vanilla. Quindi, è sufficiente avere un metodo di pricing veloce solo per queste opzioni. Questo metodo si basa sulla seguente formulazione alternativa della dinamica delle azioni

$$S_t = e^{rt+X_t} \tag{3.15}$$

dove X_t è un processo con una funzione caratteristica Φ nota, come riportato su Cont e Tankov [13].

Nel lavoro di Carr e Madan [11], gli autori introducono un metodo che sfrutta la Fast Fourier Transform (FFT) per calcolare i prezzi delle opzioni vanilla. Il metodo si basa sulla seguente formulazione semi-analitica dell'opzione call:

$$C(k) = \left[1 - e^{k-rT}\right]^+ + \frac{1}{2\pi} \int_{-\infty}^{+\infty} e^{-ivk+ivrT} \frac{\Phi_{X_T}(v-i) - 1}{iv(1+iv)} dv \tag{3.16}$$

dove, k è la *moneyness* dell'opzione, cioè $\log\left(\frac{S_0}{K}\right)$, $\Phi_{X_T}(\,\cdot\,)$ la funzione caratteristica di X_T, $\Phi_X(z) = \mathbb{E}\left[e^{iz\cdot X}\right]$. Il punto focale è nell'ultima parte dell'e-

quazione 3.16. Infatti, la parte comprendente l'integrale può essere scritta come

$$\begin{aligned} z_k &= \frac{1}{2\pi} \int_{-\infty}^{+\infty} e^{-ivk} \zeta_T(v) dv \\ \zeta_T(v) &= e^{ivrT} \frac{\Phi_{X_T}(v-i) - 1}{iv(1+iv)}. \end{aligned} \tag{3.17}$$

Ne risulta che z_k coincide con la trasformata inversa di Fourier di $\zeta_T(v)$, che può essere calcolata numericamente con il metodo FFT, avente complessità $\mathcal{O}(n \log n)$ invece della complessità $\mathcal{O}(n^2)$ genericamente richiesta per calcolare numericamente l'integrale, dove n è il numero di suddivisione dell'intervallo di integrazione, e che deve essere una potenza di 2 per il corretto funzionamento del metodo FFT. Il vantaggio computazionale, rispetto ai metodi Monte Carlo, è dovuto al fatto che n, nella FFT, non deve essere grande come N nei metodi Monte Carlo, ma si possono ottenere approssimazioni sufficientemente precise con un paio di ordini di grandezza in meno.

Capitolo 4

Nozioni di Artificial Intelligence

4.1 Artificial Neural Networks

La presente sezione introdurrà i concetti di base delle Reti Neurali Artificiali (ANN) che saranno utilizzate per affrontare il problema presentato nella tesi. In particolare, la ANN utilizzata è una cosiddetta Deep feedforward Neural Network. Il testo di riferimento per questo paragrafo è Goodfellow, Bengio e Courville [23].

4.2 Concetti di base

L'obiettivo delle Deep feedforward Neural Network, note anche come MLP (multilayer perceptrons), è quello di approssimare una funzione f^* attraverso una mappatura $y = f(x; \theta)$, dove x è l'input e θ i parametri che la MLP deve imparare al fine di ottenere la miglior possibile approssimazione della funzione.

4.3 Struttura della rete neurale

In questo paragrafo sono presentati i principali componenti che costituiscono una rete neurale.

4.3.1 Layers

La rete neurale che studiamo è chiamata, per l'appunto, una rete, perché è tipicamente una composizione incrociata di più funzioni. Per esempio, si possono avere 3 funzioni $f^{(1)}, f^{(2)}, f^{(3)}$, collegate a cascata, $y = f(x) = f^{(3)}(f^{(2)}(f^{(1)}(x)))$. In questo caso, $f^{(1)}$ è chiamato il primo *layer*, $f^{(2)}$ il secondo e $f^{(3)}$, essendo quello finale, il layer di *output*. Il primo e il secondo layer, non avendo un output usato direttamente come approssimazione, sono chiamati *hidden layer* (layer nascosti). Il numero di layer rappresenta la "profondità" di una ANN.

4.3.2 Neurons

Il fatto che si chiami rete neurale è dovuto al fatto che l'intera struttura imita, per quello che è la nostra conoscenza, la mente umana.

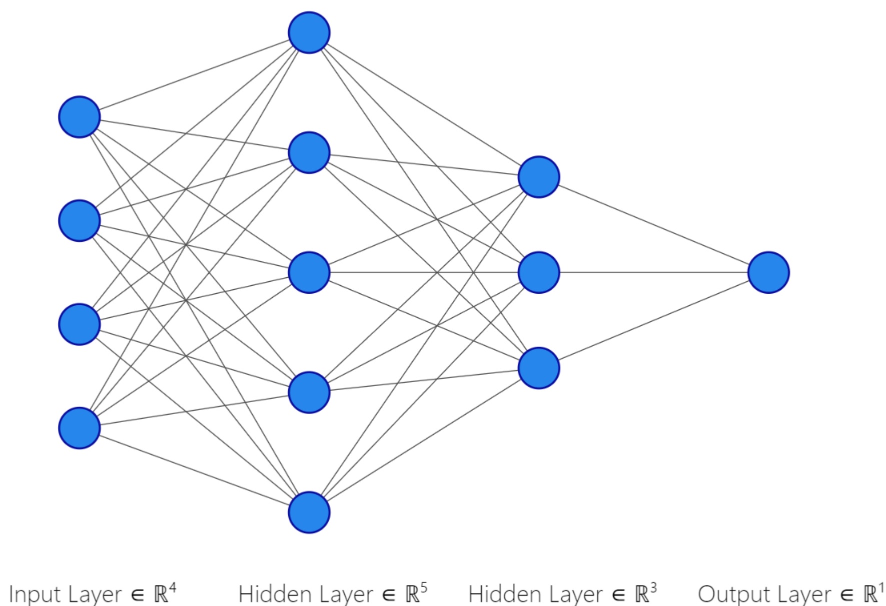


Figura 4.1: Rappresentazione di una rete neurale (estremamente) semplice

Come presentato nel paragrafo precedente, ogni output di un hidden layer è utilizzato come input per il layer successivo. In pratica, ciò che accade è che ogni strato ha un certo numero (N) di *neurons* (o *nodi*). In ogni nodo, viene eseguita un'operazione semplice utilizzando come dati di input, gli output provenienti da tutti i nodi del layer precedente. L'output del nodo i del layer

l è

$$x \mapsto \sigma_l(w_i^T x + b_i)$$

dove ω_i^T è il vettore dei pesi associati al nodo, b_i il suo parametro di *bias*, $\sigma_l(\,\cdot\,)$ la *funzione di attivazione.* Una funzione di attivazione è la stessa per tutti i nodi di un layer, ma può cambiare per layer differenti. Lo scopo della funzione di attivazione è quello di trasformare un'espressione altrimenti lineare, introducendo non linearità. Ci sono diverse funzioni di attivazione comunemente utilizzate nella pratica. Quella in uso per i layer nascosti, nel lavoro presentato, è la *Elu*, $\sigma_{Elu}(x) = \alpha(e^x - 1)$ [12], mentre per il layer di output viene utilizzata una funzione di attivazione lineare $\sigma_{Linear}(x) = cx$. Mentre l'uso dell'attivazione lineare per il layer di output è necessario poiché i valori di output, nel nostro esempio, coprono virtualmente tutti i valori di $\mathbb{R}$, la scelta della funzione di attivazione Elu è più delicata e diventerà chiara più avanti in questo capitolo.

4.3.3 Learning

Ciò che rende possibile l'uso di una ANN come approssimatore è il processo di apprendimento (o *learning*, in inglese). Infatti, ogni ANN ha bisogno di addestrarsi (*training process*) su un set di dati adeguato, al fine di apprendere le caratteristiche principali della funzione approssimata. Il processo, nella pratica, consiste nel raccogliere i dati della mappatura originale $y = f^*(x)$, realizzando un dataset che viene poi suddiviso in un insieme di *training* $X^{train} = \{x_i, f^*(x_i)\}_{i=1,\dots,M}$ e un insieme di *testing*, quest'ultimo è la parte di dataset su cui si testa la bontà dell'approssimazione. Il processo di apprendimento è il processo attraverso il quale la ANN viene calibrata, ovvero è risolto

$$\hat{w} = \underset{w \in W}{\operatorname{argmin}} \, \mathcal{L}\big(\{f(w, x_i)\}_{i=1}^M, \{f^*(x_i)\}_{i=1}^M\big) \tag{4.1}$$

dove $\mathcal{L}$ è la *loss function*, una funzione che dà una misura dell'errore di approssimazione. Nel nostro uso, la funzione di loss $\mathcal{L}$ è scelta uguale all'MSE, già definito in (3.10). Il vero tema è come risolvere il problema di minimizzazione. Il metodo in uso è l'algoritmo di retro-propagazione, o *back-propagation* (in realtà, la sua generalizzazione moderna).

4.3.4 Algoritmo di back-propagation

Questa sezione dà un'idea di come funziona l'algoritmo di retro-propagazione. Schematicamente, l'algoritmo è strutturato come segue:

Algorithm 2 Back-propagation algorithm

1: **for** d in data **do**
2: ***Forwards Pass***
3: Partendo dal layer di input partendo da d calcola il risultato, passando l'informazione avanti lungo la rete, calcolando la funzione di attivazione dei nodi di ciascun layer.
4: ***Backward Pass***
5: Calcola le derivate della loss function rispetto all'output dell'ultimo layer
6: **for** layer in reversedLayers **do** (partendo dall'ultimo layer)
7: Calcola le derivate della funzione di loss rispetto agli input dei nodi del layer successivo
8: Calcola le derivate della funzione di loss rispetto ai pesi tra il layer di output e il layer
9: Calcola le derivate della funzione di loss rispetto all'attivazione del layer
10: Aggiorna i pesi.

Il metodo utilizzato come riferimento per l'ottimizzazione (in particolare, per la fase di aggiornamento dei pesi), è il *Gradient Descent* (GD), o una sua evoluzione.

Definizione 4.3.1 (Gradient Descent)**.** *L'algoritmo iterativo Gradient Descent (GD) con funzione obiettivo $\mathcal{L}$ è caratterizzato dalla seguente regola di aggiornamento*

$$w_{n+1} = w_n - \alpha \cdot \nabla \mathcal{L}(\{F(w_n, x_j)\}_{j=1}^{M}, \{F^*(x_j)\}_{j=1}^{M}) \tag{4.2}$$

dove w_0 è la soluzione iniziale e $\alpha > 0$ è il passo di discesa.

In particolare, il metodo *Stochastic Gradient Descent* (SGD) è un metodo iterativo che differisce dal metodo GD per il fatto che il gradiente è calcolato in maniera approssimata, poiché valutato su un sottoinsieme del campione totale. L'algoritmo SGD si basa sul concetto di *batch size* (o `batch_size`, in termini informatici).

Definizione 4.3.2 (Batch size). *Dato un training set X^{train}, in forma di coppie input-output $\{x, F^*(x)\}$, chiamiamo il sottoinsieme*

$$\{x_j, F^*(x_j)\}_{j \in \mathcal{U}(\{i,\dots,m\})} := X^{batch}_{i,m} \subseteq X^{train}$$

l'i-esimo batch, con $\mathcal{U}(\{i, \dots, m\})$ distribuzione uniforme (discreta) e m dimensione di batch (o batch size).

Definizione 4.3.3 (Stochastic Gradient Descent). *L'algoritmo iterativo Stochastic Gradient Descent (SGD) con funzione obiettivo $\mathcal{L}$ è caratterizzato dalla seguente regola di aggiornamento*

$$w_{n+1} = w_n - \alpha \cdot \nabla \mathcal{L}\Bigg(\Big\{F\Big(w_n, (X^{batch}_{n,m})_i\Big)\Big\}_{j=1}^{m}, \Big\{F^*\Big((X^{batch}_{n,m})_i\Big)\Big\}_{j=1}^{m}\Bigg) \quad (4.3)$$

dove w_0 è la soluzione iniziale e $\alpha > 0$ è il passo di discesa.

Il vantaggio di questo metodo è quello di inserire una componente di aleatorietà che permette di superare il limite del Gradient Descent classico, carente nell'ottimizzazione di funzioni non convesse. Il metodo utilizzato nel presente lavoro è l'algoritmo Adam (ADAptive Moment estimation), presentato in Kingma e Ba [32], che è una successiva evoluzione del SGD.

4.3.5 Parametri di calibrazione

Al fine di ottenere il giusto trade-off tra costo computazionale e precisione della stima, occorre regolare alcuni parametri. Elenchiamo i principali di seguito.

- **Batch size**: come da Definizione 4.3.2, viene generalmente scelto un valore pari ad una potenza di 2, in quanto in questo modo è ottimizzata la parallelizzazione del calcolo;
- **Epochs**: numero di iterazioni che devono essere svolte dall'algoritmo;
- **Early Stopping**: possibilità di interruzione anticipata prima di arrivare al numero di iterazioni previste. L'arresto è associato ad un indicatore, genaralmente il valore di loss. Se la loss smette di decrescere ci si arresta anticipatamente. Generalmente è impostato un parametro di pazienza (*patience*) che indica quante iterazioni consecutive senza una diminuzione della loss bisogna attendere prima di arrestare il processo.

Generalmente è associato un *callback*, ovvero, in caso di interruzione anticipata, viene richiamata la combinazione di pesi associata alla loss minima trovata.

4.4 Utilizzo della Rete Neurale

La struttura delle reti neurali permette loro di essere degli eccellenti strumenti di approssimazione non lineare. In particolare, i seguenti due teoremi forniscono un valido supporto all'utilizzo come approssimatore.

Teorema 4.4.1 (Universal approximation theorem (Hornik, Stinchcombe e White [28])). *Sia* $\mathcal{NN}^{\sigma}_{d_0,d_1}$ *l'insieme delle reti neurali con funzione di attivazione* $\sigma : \mathbb{R} \mapsto \mathbb{R}$, *dimensione di input* $d_0 \in \mathbb{N}$ *e dimensione di output* $d_1 \in \mathbb{N}$. *Allora, se* σ *è continua e non costante,* $\mathcal{NN}^{\sigma}_{d_0,1}$ *è denso in* $L^p(\mu)$ *per ogni misura finita* μ.

Teorema 4.4.2 (Universal approximation theorem for derivatives (Hornik, Stinchcombe e White [26])). *Data* $f^* \in \mathcal{C}^n$ *e* $f : \mathbb{R}^{d_0} \to \mathbb{R}$ *(con il significato visto in 4.2) e* $\mathcal{NN}^{\sigma}_{d_0,1}$ *l'insieme delle reti neurali con funzione di attivazione* $\sigma : \mathbb{R} \mapsto \mathbb{R}$, *dimensione di input* $d_0 \in \mathbb{N}$ *e dimensione di output 1. Allora, se* σ *è non costante e* $\sigma \in \mathcal{C}^n(\mathbb{R})$, *allora,* $\mathcal{NN}^{\sigma}_{d_0,1}$ *approssima arbitrariamente* f *e ogni sua derivata fino all'ordine* n.

Pertanto, la scelta di una funzione di attivazione *Elu* assicura la possibilità di approssimare le derivate della funzione di pricing, oltre alla funzione stessa. Un'indicazione sulla numerosità dei dati e sulla struttura della rete è data dal seguente teorema.

Teorema 4.4.3 (Estimation bounds for Neural Network (Barron [3])). *Sia* $\mathcal{NN}^{\sigma}_{d_0,d_1}$ *l'insieme delle reti neurali con funzione di attivazione* $\sigma(x) = \frac{e^x}{e^x+1}$, *dimensione di input, dimensione di input* $d_0 \in \mathbb{N}$ *e dimensione di output* $d_1 \in \mathbb{N}$. *Allora,*

$$\mathbb{E}\,\|f^* - f\| \leq \mathcal{O}\left(\frac{C_f^2}{n}\right) + \mathcal{O}\left(\frac{nd_0}{N}\log N\right)$$

dove n *è il numero di nodi,* N *la dimensione del training set e* C_f *è il momento primo assoluto della Fourier magnitude distribution.*

Di questa decomposizione dell'errore, il primo termine rappresenta la complessità del modello, cioè un alto numero di nodi n riduce il primo errore. Il secondo termine rappresenta la varianza, si noti che un numero elevato di nodi n deve essere sostenuto da un dataset ampio, per non incorrere in un secondo errore elevato. Questa decomposizione affronta i due tipi di errori più comuni che si affrontano lavorando con le reti. Quando il numero di nodi n è piccolo, e quindi il primo errore è elevato, incorriamo nell'errore di underfitting, ovvero il modello non è sufficientemente complesso da cogliere tutti gli aspetti della funzione da approssimare. Viceversa, se n è troppo elevato e non è sostenuto da un dataset sufficientemente ampio, si incorre nell'errore di overfitting, ovvero il modello performa estremamente bene nell'approssimare il dataset, ma non è in grado di approssimare i valori della funzione non appartenenti al dataset.
Infine, il seguente teorema (enunciato informalmente) supporta l'utilizzo di una rete con layer multipli.

Teorema 4.4.4 (Power of depth of Neural Network (Eldan e Shamir [15])). *Esiste una funzione su $\mathbb{R}^d$ semplice (approssimativamente radiale), esprimibile da una rete feedforward con 3 layer, piccola, che non può essere espressa oltre una certa accuratezza da nessuna rete con 2 layer, a meno di essere esponenziale nel numero di nodi.*

Capitolo 5

Progetto logico della soluzione del problema

In questo capitolo è ripercorso il processo che porta alla calibrazione in ogni sua fase, dalla costruzione del dataset, al training della rete neurale e alla calibrazione.

5.1 Pricing e calibrazione con reti neurali

In questa sezione sono presentate alcune tecniche di pricing e di calibrazione tramite reti neurali.

5.1.1 Inverse mapping

Il primo approccio, proposto in Hernández [24], è quello di utilizzare una mappa inversa che, avendo in input le superfici di volatilità, restituisca in output i relativi parametri, ovvero

$$\mathcal{P}^{-1}\big(\Sigma_{BS}^{MKT}, \{k_i\}_{i=1}^{n}, \{T_j\}_{j=1}^{m}\big) \tag{5.1}$$

dove $\mathcal{P}^{-1}$ è la mappa inversa, Σ_{BS}^{MKT} è la volatilità di mercato, $\{k_i\}_{i=1}^{n}$ gli strike e $\{T_j\}_{j=1}^{m}$ le maturity relative. Questo modello, tuttavia, non è utilizzato nel presente lavoro. Il motivo di tale scelta è dato dagli svantaggi di questo metodo, come riportato in Hernández [24]. Infatti, la mancanza di controllo sulla funzione $\mathcal{P}^{-1}$ e la successiva incapacità di predire quanto correttamente sia replicata la superficie di volatilità sono due svantaggi rilevanti. Inoltre, il

modello ha mostrato una scarsa capacità nel replicare i dati esterni al dataset di training.

5.1.2 Utilizzo del pricing

L'idea alternativa si basa sull'approssimazione di una mappa di pricing attraverso una rete neurale F,

$$\tilde{P}(\mathcal{M}(\theta), \zeta) \approx F(\theta, \zeta) \tag{5.2}$$

e quindi risolvere il problema di calibrazione in un secondo passaggio. Nel nostro caso, invece di approssimare una mappa che ha come output prezzi, approssimiamo una mappa che ha come output le relative volatilità implicite. Si tratta quindi di un approccio a due passi:

i. **Learn**: $\tilde{P}(\mathcal{M}(\theta), \zeta) = \tilde{F}(\theta, \zeta)$;

ii. **Calibrate**: $\hat{\theta} = \text{argmin}_{\theta \in \Theta} \delta(\tilde{F}(\theta, \zeta), \mathcal{P}^{MKT}(\zeta))$.

Apprendimento puntuale

La tecnica più naturale è quella di seguire i due passi sopraindicati come segue.

i. Si fa imparare la mappa

$$F^*(\theta, T, k) = \sigma_{BS}^{\mathcal{M}(\theta)}(T, k)$$

attraverso la rete $\tilde{F}(\theta, T, k) = F(\theta, \hat{w}, T, k)$, avente pesi $\hat{w}$, dove

$$\begin{aligned} F^* : \Theta \times [0, T_{max}] \times [k_{min}, k_{max}] &\to \mathbb{R} \\ (\theta, T, k) &\mapsto F^*(\theta, T, k) \end{aligned} \tag{5.3}$$

e

$$\hat{w} = \underset{w \in \mathbb{R}^n}{\text{argmin}} \sum_{i=1}^{N_{Train}} \big(F(\theta_i, w, T_i, k_i) - F^*(\theta, T_i, k_i)\big)^2 \tag{5.4}$$

dove $\theta_i \in \Theta$, $T_i \in [0, T_{max}]$,$k_i \in [k_{min}, k_{max}]$ $\forall i = 1 \dots N_{Train}$, con N_{Train} numerosità del training set. $\sigma_{BS}^{\mathcal{M}(\theta)}(T, k)$ rappresenta la volatilità implicita di Black-Scholes del modello $\mathcal{M}$ con parametri θ.

ii. Si risolve il problema di minimizzazione

$$\hat{\theta} := \underset{\theta \in \Theta}{\operatorname{argmin}} \sum_{i=1}^{n} \sum_{j=1}^{m} \left(\tilde{F}(\theta, T_i, k_j) - \sigma_{BS}^{MKT}(T_i, k_j) \right)^2. \tag{5.5}$$

Tecniche che seguono questa logica possono essere trovate in Bayer e Stemper [4], Spiegeleer, Madan, Reyners e Schoutens [44] e Ferguson e Green [16].

Apprendimento image-based

La tecnica utilizzata nel presente lavoro, presente anche in Horvath, Muguruza e Tomas [30], chiamata *image-based* poiché apprende la superficie di volatilità per intero, come se fosse un'immagine, richiede un accorgimento aggiuntivo. Le maturity e gli strike devono essere fissati a priori, definendo $\Delta := \{k_i, T_j\}_{i=1,j=1}^{n,m}$ griglia delle maturity e degli strike. Tutte le superfici di volatilità devono essere costruite sulla base di questa griglia. Il metodo è strutturato come segue.

i. Si fa imparare la mappa

$$F^*(\theta, T, k) = \left\{ \sigma_{BS}^{\mathcal{M}(\theta)}(T_i, k_j) \right\}_{i=1,j=1}^{n,m}$$

attraverso la rete $\tilde{F}(\theta) = F(\theta, \hat{w})$, avente pesi $\hat{w}$, dove

$$\begin{aligned} F^* : \Theta &\to \mathbb{R}^{n \times m} \\ \theta &\mapsto F^*(\theta) \end{aligned} \tag{5.6}$$

e

$$\hat{w} = \underset{w \in \mathbb{R}^n}{\operatorname{argmin}} \sum_{u=1}^{N_{Train}} \sum_{i=1}^{n} \sum_{j=1}^{m} \left(F(\theta_u, w)_{i,j} - F^*(\theta)_{i,j} \right)^2. \tag{5.7}$$

ii. Si risolve il problema di minimizzazione

$$\hat{\theta} := \underset{\theta \in \Theta}{\operatorname{argmin}} \sum_{i=1}^{n} \sum_{j=1}^{m} \left(\tilde{F}(\theta)_{i,j} - \sigma_{BS}^{MKT}(T_i, k_j) \right)^2. \tag{5.8}$$

5.2 Dataset

La buona riuscita di un modello che ha una prima fase di training su un dataset dipende fortemente dalla qualità del dato disponibile.

I dataset utilizzati hanno una riga per ogni superficie. Ogni riga è composta da due parti: i primi valori sono i parametri, gli ultimi la superficie di volatilità associata. La griglia Δ utilizzata ha 8 valori di maturity e 11 valori di strike.

Per quanto riguarda i modelli rough Bergomi, Bergomi 1-Factor e Heston, sono stati utilizzati i dataset analizzati in Horvath, Muguruza e Tomas [30][1]. Come già mostrato dagli autori, il dataset è appropriato per affrontare il problema.

Per quanto riguarda i due modelli auto similari invece, non essendo disponibile un dataset analogo, è stato generato *ad hoc*. La tecnica di seguito riportata è riproducibile anche per altri modelli di pricing. La strutturazione del dataset è svolta secondo i seguenti passaggi.

i. **Individuazione intervalli**: per ogni parametro θ_u del modello è individuato un valore massimo θ_u^{max} e uno minimo θ_u^{min}. Tali valori sono scelti basandosi sui valori tipici riportati da Carr, Geman, Madan e Yor [10];

ii. **Suddivisione intervalli**: per ogni intervallo $[\theta_u^{min}, \theta_u^{max}]$ viene presa una discretizzazione regolare di $N_{interval}$ elementi. Per questo lavoro si è scelto $N_{interval} = 18$, giudicato come un numero di suddivisioni sufficiente ad ottenere una buona granularità del dato, senza pesare troppo dal punto di vista computazionale;

iii. **Pricing**: per ogni possibile combinazione di parametri della discretizzazione è calcolato il prezzo tramite la formula (3.16) per ogni valore della griglia Δ. Sono utilizzati i valori $S_0 = 1$ e $r = 0$. Dato che alcune combinazioni di parametri non sono possibili perché genererebbero una soluzione non analitica, queste combinazioni sono rimosse dal dataset a priori. Avendo un totale di 4 parametri, le combinazioni possibili sono $N_{interval}^4$ ($18^4 = 104,976$);

[1] Disponibili sul sito `https://github.com/amuguruza/NN-StochVol-Calibrations/tree/master/Data`.

iv. **Volatilità implicita**: per ogni prezzo è calcolata la volatilità implicita di Black-Scholes. Anche in questo caso ipotizziamo $S_0 = 1$ e $r = 0$.

Così facendo si ottiene un dataset ampio, capace di coprire le necessità di questo studio. Tuttavia la discretizzazione dei parametri, così come la scelta dei parametri contrattuali, è arbitraria e può essere aggiustata a seconda delle necessità. Si tenga presente che è possibile utilizzare alcune delle classiche tecniche di interpolazione o estrapolazione tipicamente usate per coprire i valori di maturity e strike mancanti.

In fase di messa a punto della rete neurale il dataset viene suddiviso in una parte di training e una parte di validation, al fine di ridurre il rischio di overfitting. Inoltre, prima di essere trasmesso alla rete neurale per la fase di addestramento, il dataset passa per una fase di preprocessing. In particolare i dati sono riscalati al fine di velocizzare la fase di training, non dovendo la rete apprendere l'ordine di grandezza dei dati utilizzati. Per quanto riguarda i parametri, essendo definiti con un limite inferiore ed uno superiore, sono riscalati come

$$\theta^{scaled} = \frac{\theta - \frac{\theta_{max}+\theta_{min}}{2}}{\frac{\theta_{max}-\theta_{min}}{2}}$$

utilizzando una funzione specificamente implementata per lo scopo. Relativamente alle volatilità implicite, in quanto potenzialmente distribuite su tutto $\mathbb{R}^+$, sono normalizzate come

$$\sigma_{t,k}^{scaled} = \frac{\sigma_{t,k} - \mu(\sigma_{t,k})}{std(\sigma_{t,k})},$$

dove $\mu(\sigma_{t,k})$ e $std(\sigma_{t,k})$ sono rispettivamente la media e la deviazione standard del punto della griglia di volatilità con maturity t e strike k.

5.3 Rete neurale

Di seguito sono descritte le reti neurali per i diversi modelli. Le reti utilizzate hanno 3 o 4 layer. Il limite inferiore è dovuto a quanto riportato nel Teorema 4.4.4. Il limite superiore è basato sul dato empirico. Siccome le architetture proposte si sono rivelate efficaci, non si è ritenuto necessario aggiungere un ulteriore livello di complessità.

5.3.1 Rete a 3 hidden layer

Per tutti i modelli ad esclusione dei modelli rough Bergomi e 1-Factor Bergomi in cui è utilizzata un'approssimazione costante a tratti della varianza forward (e quindi con un più elevato grado di complessità) è stata utilizzata una rete (feedforward) con le seguenti caratteristiche:

- numero di hidden layer = 3;
- nodi per ogni hidden layer = 32;
- dimensione di input = n_{params}, numero di parametri del modello;
- dimensione di output = 8×11;
- funzione di attivazione hidden layer = Elu;
- funzione di attivazione output layer = linear.

La scelta di questa struttura segue dai teoremi già presentati nel Capitolo 4. In particolare, relativamente alla scelta della funzione di attivazione, per il Teorema 4.4.1, riportato in Hornik, Stinchcombe e White [28], e il Teorema 4.4.2, riportato in Hornik, Stinchcombe e White [26], si è scelta la funzione di attivazione Elu. Da quanto descritto in Eldan e Shamir [15] e riassunto nel Teorema 4.4.4 la rete neurale utilizzata è composta da almeno 3 layer. La scelta di 32 nodi per gli hidden layer è basata sul lavoro già visto in Horvath, Muguruza e Tomas [30], in cui ne sono usati 30. Tuttavia la scelta di utilizzarne 32, essendo una potenza di 2, è dovuta al fatto che in questo modo è più facilmente ottimizzabile rispetto all'utilizzo della GPU del computer, solitamente impiegata per la fase di training delle reti neurali. Data la struttura, i parametri della rete da calibrare sono

$$(n_{params}+1)\times 32+(32+1)\times 32+(32+1)\times 32+(32+1)\times 88 = 32n_{params}+5048.$$

In figura 5.1 una rappresentazione semplificata della rete descritta.

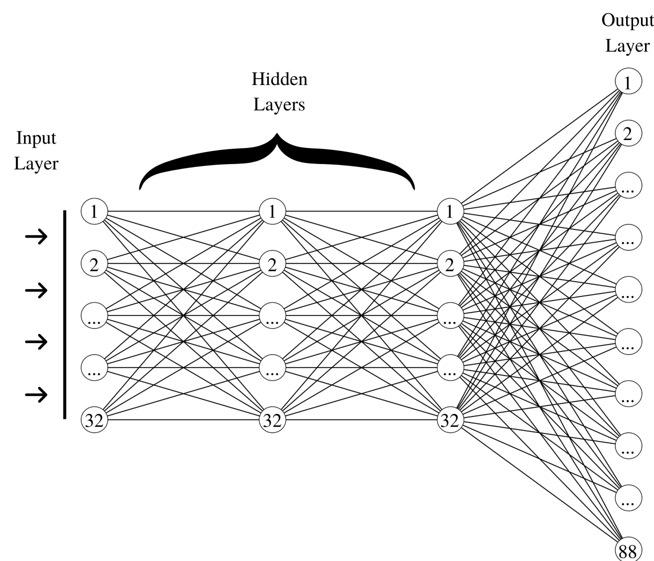


Figura 5.1: Rappresentazione semplificata della rete a 3 hidden layer utilizzata

5.3.2 Rete a 4 hidden layer

Per i modelli rough Bergomi e 1-Factor Bergomi in cui è utilizzata un'approssimazione costante a tratti della varianza forward è stata utilizzata una rete con le seguenti caratteristiche:

- numero di hidden layer = 4;
- nodi per ogni hidden layer = 32;
- dimensione di input = n_{params}, numero di parametri del modello;
- dimensione di output = 8×11;
- funzione di attivazione hidden layer = Elu;
- funzione di attivazione output layer = linear.

L'unica differenza con la rete descritta nel paragrafo precedente è l'introduzione di un ulteriore hidden layer per incrementare la complessità del modello, necessaria per affrontare un modello più strutturato. La maggior complessità si traduce in un maggior numero di parametri della rete da calibrare. In questo caso sono

$$(n_{params} + 1) \times 32 + 3 \times (32 + 1) \times 32 + (32 + 1) \times 88 = 32n_{params} + 6104.$$

In figura 5.2 una rappresentazione della rete descritta.

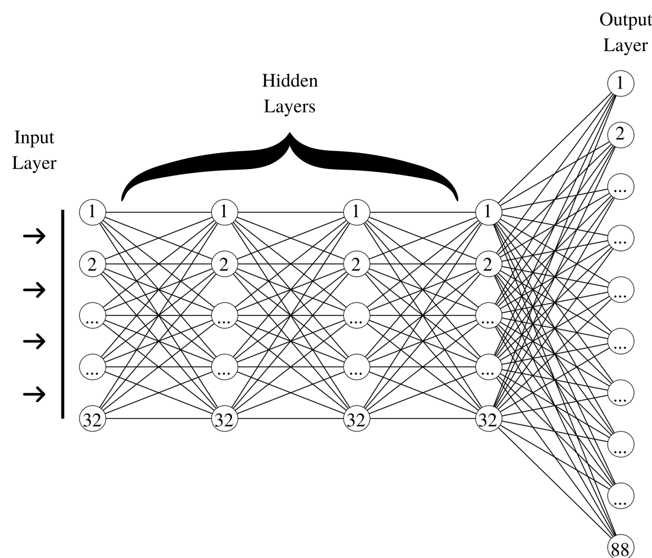


Figura 5.2: Rappresentazione semplificata della rete a 4 hidden layer utilizzata

5.3.3 Parametri di ottimizzazione

In aggiunta all'architettura della rete, è necessario definire alcuni parametri necessari affinché la compilazione sia più efficiente e precisa possibile. In particolare, per entrambe le reti:

- **Epochs**: sono state impostate 1000 epoch;
- **Callback**: è stato impostato un early stopping sulla loss del set di validazione, con parametro *patience* = 150;

- **Batch size**: è stato utilizzato un valore di $batch_size = 128$.

I parametri sopracitati sono descritti nel Capitolo 4.3.5 e i relativi valori sono scelti su base empirica, trovando un giusto trade-off tra precisione e costo computazionale.

5.4 Problema di minimizzazione

Per risolvere i problemi di minimizzazione (5.5) e (5.8), è necessario scegliere un metodo risolutivo. Come suggerito in Horvath, Muguruza e Tomas [30], sono stati presi in considerazione metodi diversi. Nel Capitolo 6 sono riportati i risultati dell'analisi sull'efficienza dei diversi metodi nella risoluzione del problema presentato. Gli algoritmi utilizzati sono:

- ***Levenberg-Marquardt*** **[34, 40]**: un metodo utilizzato primariamente per la minimizzazione di problemi del tipo

$$\hat{\theta} = \operatorname*{argmin}_{\theta \in \Theta} \sum_{i=1}^{n} \big(f(\theta, x_i) - y_i\big)^2;$$

- ***BFGS*** **[8, 17, 22, 42]**: un metodo di Quasi-Newton basato sul precondizionamento del gradiente per individuare la direzione di discesa;

- ***L-BFGS-B*** **[46]**: una versione a memoria limitata del precedente metodo, particolarmente efficiente per i problemi con un gran numero di variabili;

- ***SLSQP*** **[33]**: una versione di un metodo di Sequential Quadratic Programming, metodo di minimizzazione utilizzato per i problemi dove la funzione obiettivo e i vincoli sono due volte differenziabili.

Capitolo 6

Implementazione della soluzione

Nel presente capitolo sono sono presentati i principali script e funzioni utilizzati nella soluzione, con un dettaglio sulle scelte implementative e sulle principali librerie utilizzate. Come il progetto logico, anche il codice si compone di due parti principali. La prima parte è dedicata alla realizzazione dei dataset necessari. La seconda parte è invece relativa alla soluzione del problema di calibrazione in due passi vista nel Capitolo 5. Il codice è sviluppato in linguaggio Python 3, in ambiente Jupyter. Una descrizione del contenuto del dataset è riportata in Appendice C.

6.1 Creazione del dataset

In questa sezione si ripercorrono le principali fasi e le scelte relative alla costruzione del dataset per i due modelli auto similari. Dato che il processo è il medesimo è presentato in forma generale.

6.1.1 Funzioni di pricing

Il primo passo per la creazione del dataset è il pricing delle opzioni call (europee) per tutte le possibili combinazioni di parametri di modello e contrattuali.

Per i modelli NIGSSD e VGSSD visti nel Capitolo 3.2.4 è possibile calcolare il prezzo delle opzioni vanilla sia utilizzando un pricing Monte Carlo, sia utilizzando un metodo, molto più rapido, basato sulle FFT. I metodi basati su quest'ultima tecnica sono primariamente due: il metodo di Carr-Madan

[11] e la formula di Lewis, presentata in Lewis [35]. Quest'ultima formula ha il limite di non essere analitica per alcune combinazioni di parametri (è definita una *analyticity strip*, una striscia di analiticità, supportati dal teorema presentato in Lukacs [36]). Pertanto è stato preferito il metodo di Carr-Madan come funzione di pricing. Tuttavia, per come sono strutturate le funzioni caratteristiche dei due modelli, ci sono delle combinazioni di parametri per cui la funzione caratteristica stessa non è analitica, pertanto questi casi specifici sono trattati a parte.

La funzione implementata in Python per il pricing di un'opzione call è

```
CallPrices_CM(phiY, r, moneyness, TTM, methodParams),
```

dove `phyY` è la funzione caratteristica del modello, `r` è il tasso di interesse risk-free, `moneyness` è la moneyness dell'opzione, definita come $\log \frac{S_0}{K}$, `TTM` è il *time-to-maturity* dell'opzione, ovvero il tempo restante fino alla maturity dell'opzione e `methodParams` è un dizionario contenente i paramtri di suddivisione necessari al metodo FFT. Restituisce come output il prezzo dell'opzione call (europea) associata. Al fine di rendere il codice il più riutilizzabile possibile, tale funzione è in grado di prezzare un'opzione call (europea) per qualsiasi modello per cui sia utilizzabile il metodo di Carr-Madan. Si osservi che la funzione è implementata solo per il calcolo di un'opzione call, tuttavia l'opzione put con gli stessi parametri contrattuali può essere facilmente calcolata tramite la put-call parity (3.5).

Per supportare il calcolo delle opzioni, sono state implementate le funzioni di calcolo del prezzo tramite metodo di Monte Carlo, rispettivamente per il modello VGSSD e NIGSSD:

- `priceVGSSD_MC(moneyness,TTM,sigma,nu,theta,gamma,N_MC)`
- `priceNIGSSD_MC(moneyness,TTM,sigma,nu,theta,gamma,N_MC)`

dove `sigma`, `nu`, `theta` e `gamma` sono i parametri del modello e `N_MC` il numero di simulazioni Monte Carlo. Restituiscono come output il prezzo dell'opzione call (europea) associata. Queste funzioni sono state aggiunte a supporto, entrando in azione solo nel caso in cui il pricing tramite `CallPrices_CM` risulti in un valore non valido. Tuttavia nel corso del processo di creazione del dataset non sono state chiamate in causa, funzionando correttamente la funzione primaria di pricing. Dato che il metodo di Monte Carlo consiste nel simulare la dinamica del sottostante definita dal rispettivo modello, non è stato possibile unificare il processo sotto un'unica funzione.

6.1.2 Volatilità implicita

Per il calcolo della volatilità implicita è stata utilizzata la libreria py_vollib ed in particolare la funzione

`implied_volatility(price, S, K, t, r, flag)`

dove `price` è il prezzo dell'opzione, `S` è il prezzo del sottostante, `K` lo strike, `t` il *time-to-maturity*, `r` il tasso risk-free, `flag` una variabile per indicare se la variabile `price` è relativa al prezzo di una call o di una put. Per il calcolo della volatilità implicita sono state implementate altre funzioni, non altrettanto efficienti e pertanto non utilizzate.

6.1.3 Costruzione del dataset

Una volta definite le funzioni di pricing e di calcolo della volatilità implicita, è possibile costruire il dataset. Di seguito è riportato uno pseudo-codice dello script preposto allo scopo.

Algorithm 3 Costruzione Dataset

1: dataset = *Vettore contenente il futuro dataset, inizialmente vuoto*
2: **for** θ in Θ **do** % *tutte le possibili combinazioni di parametri*
3: volatility_temp = *Vettore che conterrà le volatilità implicite per una specifica combinazione*
4: **for** T in Maturities **do**
5: **for** K in Strikes **do**
6: **if** θ combinazione analitica **then**
7: price = CallPrices_CM(...)
8: iv = implied_volatility(price,...)
9: volatility_temp $\leftarrow$ iv
10: **else**
11: pass
12: dataset $\leftarrow$ volatility_temp
13: Salva il dataset

Al termine del processo i dataset sono salvati in un file gzip chiamato "*NOME_MODELLO*TrainSet.txt.gz", sostituendo a *NOME_MODELLO* l'opportuna denominazione del modello. È opportuno osservare che questa è

la fase più computazionalmente costosa dell'intero processo, richiedendo per ogni modello circa 10 ore di computazione con la macchina a mia disposizione (processore Intel(R) Core(TM) i5-7300U CPU @ 2.60GHz, RAM 8 GB, sistema operativo Windows 10). Questa elaborazione è richiesta *una tantum* ed è svolta "offline", non in fase di mercato.

6.2 Rete neurale

Per la costruzione della rete neurale è stata utilizzata la libreria Keras, una risorsa per la costruzione di reti neurali più o meno complesse. In particolare sono state utilizzate le seguenti:

- `Sequential`: classe di modello di rete neurale in cui i layer sono posizionati in sequenza. Il modello è inizialmente vuoto;
- `add`: metodo per aggiungere un layer al modello;
- `Dense`: classe di layer. Definisce un layer regolare densamente connesso[1].

Una volta costruita la rete desiderata è possibile avviare la fase di training. Il tempo computazionale di questa fase è di circa 14 minuti con la macchina a mia disposizione[2] (e senza utilizzare la GPU per velocizzare il calcolo), tuttavia anche questo processo è da svolgere *una tantum*, poiché è possibile salvare i pesi ottimi. In Figura 6.1 è rappresentato il profilo di loss dei training e validation set al passare delle epoch per il modello VGSSD. Gli altri modelli analizzati evidenziano un grafico analogo. L'andamento tipico evidenzia che la loss è giunta a convergenza, poiché il profilo sembra appiattirsi, e che non sembra esserci overfitting, dato che il profilo del validation set segue quello del training set.

[1] Ovvero i layer visti nel Capitolo 4.3.1. Esistono altre tipologie di layer non utili allo scopo di questa tesi e per questo non trattate.

[2] Misurato sul modello VGSSD, gli altri modelli riportano tempistiche analoghe.

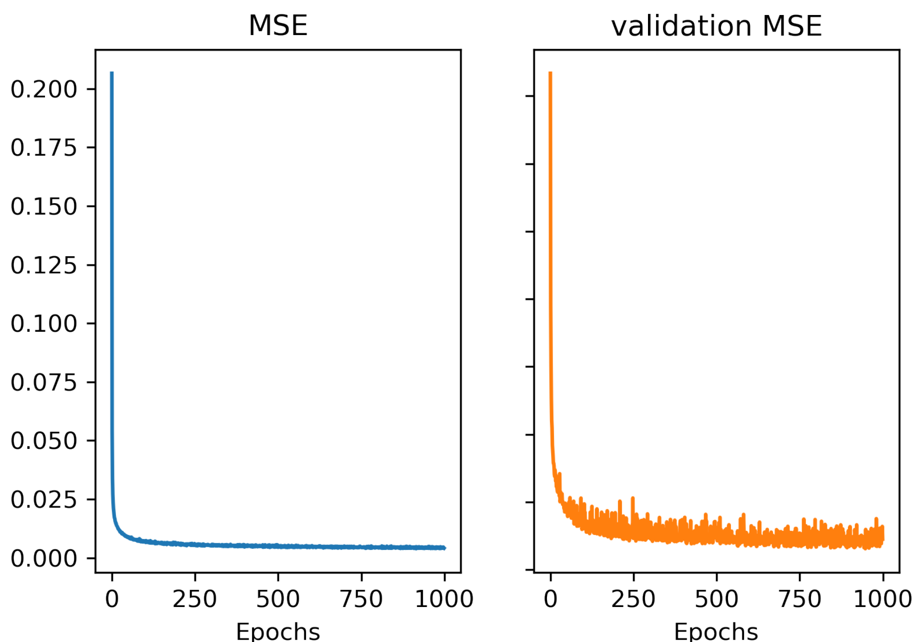


Figura 6.1: Andamento della loss (MSE) al passare delle epochs

Poiché la rete costruita è semplice, partendo dai pesi calibrati è possibile implementare la rete sfruttando la libreria NumPy, come suggerito in Horvath, Muguruza e Tomas [30], ottenendo un notevole vantaggio computazionale. I risultati di seguito riportati sono relativi al modello VGSSD, ma gli altri modelli si comportano in maniera similare. Valutare una singola combinazione di parametri ha un costo medio di 42μs $\pm 3.44\mu$s per l'implementazione con NumPy, di 47.3ms $\pm 1.89\mu$s per il passaggio in Keras. L'implementazione in NumPy risulta essere circa 3 ordini di grandezza più veloce. La funzione implementata per la creazione e calibrazione della rete neurale è chiamata `GeneralizedNNCalibration`.

6.2.1 Calibrazione dei parametri

Per l'ultima fase, in cui avviene la calibrazione dei parametri, sono confrontati i tempi computazionali degli algoritmi di minimizzazione considerati al fine di selezionare il metodo ottimale. Il risultato riportato è relativo al modello VGSSD, ma gli altri modelli si comportano in maniera similare. Il risultato è rappresentato nel grafico in Figura 6.2.

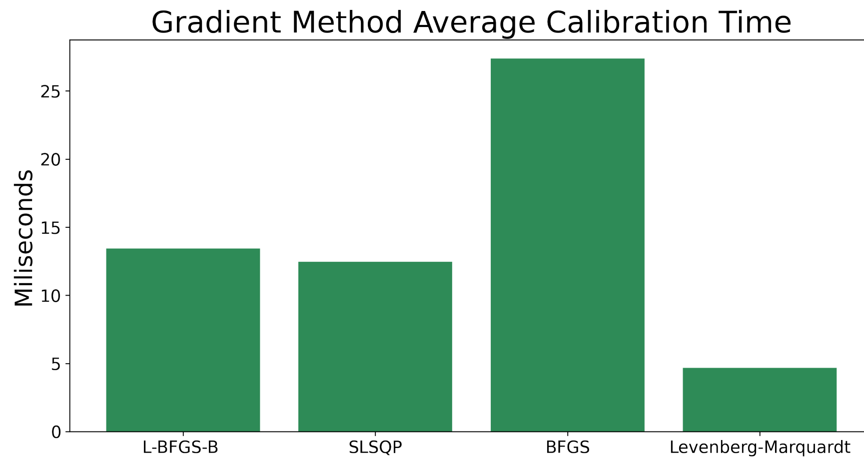


Figura 6.2: Tempo medio di calibrazione utilizzando algoritmi diversi

L'algoritmo Levenberg-Marquardt risulta essere il più vantaggioso, con un costo medio di circa 5ms. Tutti i modelli hanno tuttavia un tempo di esecuzione non elevato.

Capitolo 7

Risultati e commenti

Per supportare la scelta dell'architettura delle reti neurali utilizzate sono state svolte alcune analisi per stabilire quale fosse il numero adeguato di hidden layer da utilizzare. Una volta confermata la struttura sono svolte analisi sulla qualità del metodo. Per ognuno dei modelli presentati è stato svolto l'intero processo di calibrazione e sono state effettuate alcune verifiche sulla bontà della stima ottenuta. Poiché l'analisi svolta è la stessa per tutti i modelli, sono qui presentati due modelli con modellistica della rete neurale a 3 hidden layer e uno con modellistica a 4 hidden layer. I risultati per gli altri modelli sono riportati in Appendice A. Per entrambi i tipi di analisi è stato utilizzato lo stesso dataset, descritto in Appendice C. Per ognuno dei modelli il dataset è suddiviso randomicamente in una parte di training (70%) e una di test (30%). In fase di addestramento della rete neurale il training set è suddiviso in una parte di training vera e propria (90%) e una di validation (10%).

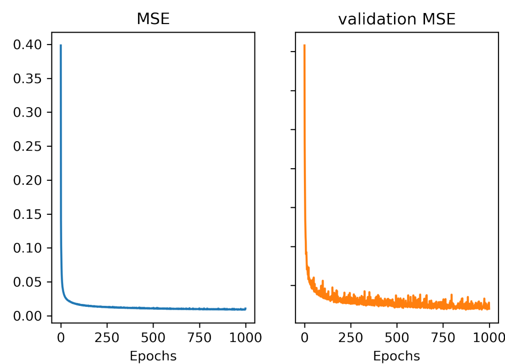
7.1 Analisi sul numero di hidden layer

Per ognuno dei modelli utilizzati è stata svolta un'analisi sull'apprendimento della mappa tra parametri del modello e superficie di volatilità implicita. Per i modelli più semplici (categorizzati come modelli con rete a 3 hidden layer), sono confrontate due reti, una a 3 e una a 4 hidden layer. Per i restanti modelli, oltre alle due reti già menzionate, è valutata anche una rete a 5 hidden layer. Siccome i risultati per i diversi modelli sono simili, sono qui riportati i risultati per i modelli rough Bergomi con approssimazione

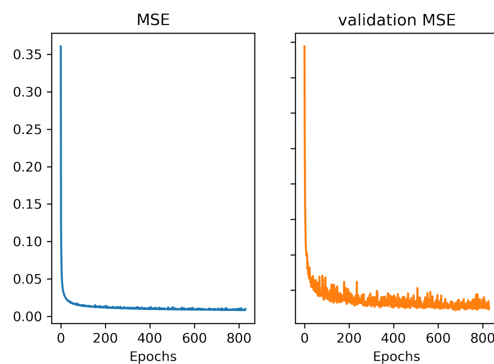
costante e rough Bergomi con approssimazione costante a tratti, i risultati per i restanti modelli sono disponibili in Appendice A.

7.1.1 Rough Bergomi

Dall'analisi dell'andamento della loss (MSE) del training e del validation set in Figura 7.1 si può osservare come per nessuna delle due strutture l'evoluzione di questa misura possa far pensare a un overfitting o ad un mancato raggiungimento della convergenza dell'errore, siccome il trend delle loss è allineato per i due set, è decrescente e mostra un forte appiattimento.



(a) Rete a 3 hidden layer



(b) Rete a 4 hidden layer

Figura 7.1: Andamento della loss al passare delle epoch

Il valore di loss raggiunto dalle due reti è consistente con il risultato ottenuto sul test set, per il quale si ottiene un valore di loss di 0.0098 per la

rete a 3 layer e di 0.0103 per la rete a 4 layer. Si riporta in Tabella 7.1 un riepilogo dei risultati ottenuti.

Tabella 7.1: Valori di loss per le diverse architetture

Rete a 3 Layer			Rete a 4 Layer		
Training	Validation	Test	Training	Validation	Test
0.0088	0.0090	0.0098	0.0089	0.0087	0.0103

Essendo i valori vicini, non si considera esserci motivo di complicare il modello con una rete a 4 o più layer.

7.1.2 Rough Bergomi con approssimazione costante a tratti

Dall'analisi dell'andamento della loss (MSE) del training e del validation set in Figura 7.2 si può osservare come per nessuna delle tre strutture l'evoluzione di questa misura possa far pensare a un overfitting o ad un mancato raggiungimento della convergenza dell'errore, mostrando le loss lo stesso comportamento dell'esempio precedente.

Il valore di loss raggiunto dalle due reti è consistente con il risultato ottenuto sul test set, sebbene il risultato lievemente maggiore su quest'ultimo set evidenzi un leggero overfitting. I risultati sono riportati in Tabella 7.2.

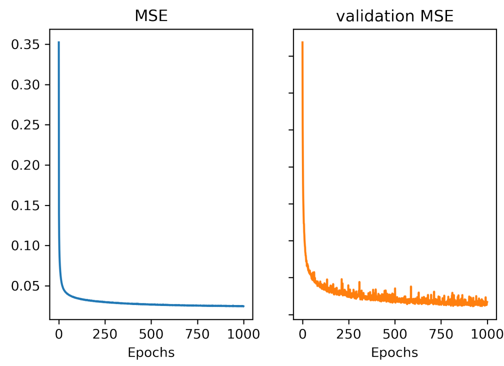
Tabella 7.2: Valori di loss per le diverse architetture

Rete a 3 Layer			Rete a 4 Layer		
Training	Validation	Test	Training	Validation	Test
0.0240	0.0239	0.0269	0.0210	0.0209	0.0224

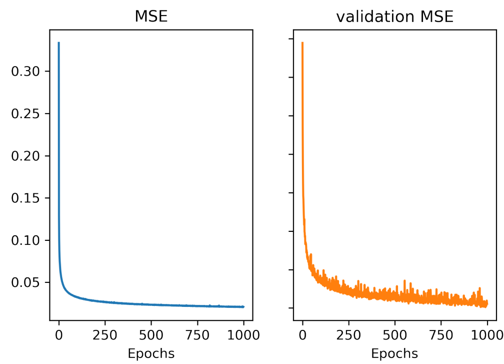
Rete a 5 Layer		
Training	Validation	Test
0.0199	0.0194	0.0211

La differenza di loss tra le reti a 3 e a 4 hidden layer è considerata non trascurabile, c'è motivo in questo caso di propendere per un modello più complesso. Essendo invece i valori della loss per la rete a 4 e a 5 hidden layer vicini, non c'è motivo di complicare il modello con una rete a 5 o più layer.

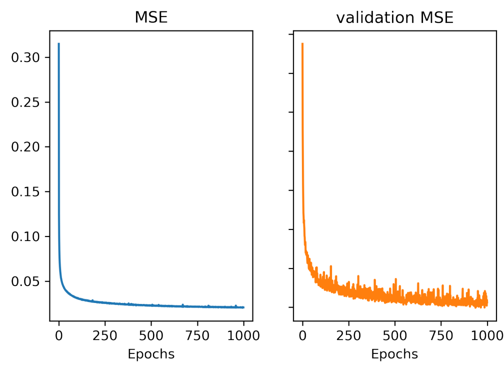
Le analisi svolte confermano l'adeguatezza delle architetture delle reti neurali utilizzate.



(a) Rete a 3 hidden layer



(b) Rete a 4 hidden layer



(c) Rete a 5 hidden layer

Figura 7.2: Andamento della loss al passare delle epoch

7.2 Modelli con rete a 3 hidden layers

Di seguito sono riportati i risultati relativi ai modelli approssimati dalla rete neurale a 3 hidden layer.

7.2.1 Rough Bergomi

Il primo modello analizzato è il modello rough Bergomi con approssimazione costante (o *flat*) della forward variance, rappresentata dal parametro ξ_0. Riportiamo di seguito il grafico relativo all'errore nella stima della volatilità per i diversi punti della griglia Δ. In termini relativi[1] sono rappresentati in Figura 7.3 l'errore medio, la deviazione standard dell'errore e l'errore relativo massimo per ogni punto della griglia.

Figura 7.3: Errore relativo tra dati di input e stima della rete neurale: errore medio, deviazione standard, errore massimo

Si osserva in Figura 7.3 che l'errore medio si mantiene sotto lo 0.6%, mentre si registra per un punto della griglia Δ un errore massimo in termini relativi di circa 16%, sebbene per il resto dei punti la misura di errore si mantenga più limitata. Si osserva inoltre che le differenze maggiori si registrano per punti periferici della griglia, soprattutto per valori bassi di maturity. In Figura 7.4 è rappresentato un confronto tra i dati di input e l'approssimazione generata dalla rete neurale. La superficie rappresentata è esemplificativa, le restanti superfici del campione mostrano caratteristiche analoghe.

Analogamente, in Figura 7.5 sono rappresentati gli *smile* di volatilità relativi alle diverse scadenze considerate, per i dati di input e per l'approssimazione della rete neurale. Anche in questo caso le curve, relative ad una stessa superficie, sono prese a scopo esemplificativo.

[1] Ovvero è valutata la quantità $\frac{\sigma^{NN}-\sigma^{\mathcal{M}}}{\sigma^{\mathcal{M}}}$.

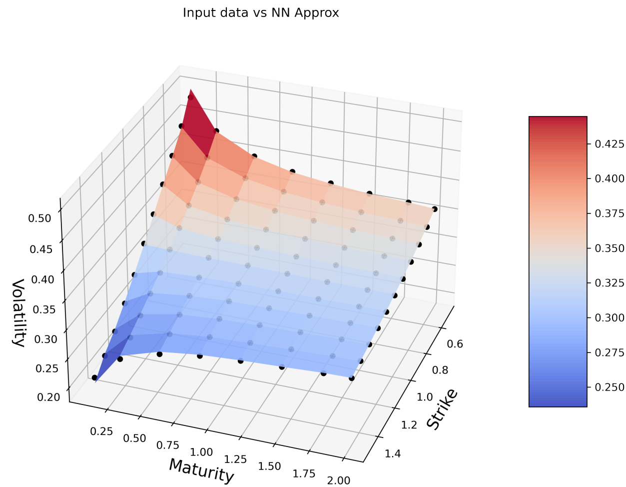


Figura 7.4: Rappresentazione di una superficie di volatilità. La superficie rappresenta i dati relativi al modello di pricing, i punti disegnati in nero i risultati dell'approssimazione della rete neurale

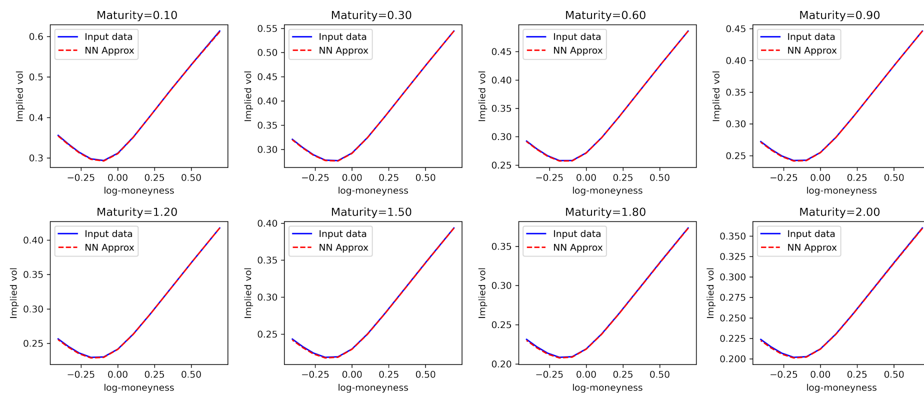


Figura 7.5: Smile relativi alle diverse maturity considerate. Sono confrontati i dati di input (in blu) con l'approssimazione generata dalla rete neurale (in rosso).

Le rappresentazioni in Figura 7.4 e in Figura 7.5 evidenziano come la

rete neurale sia in grado di approssimare efficacemente curve e superfici nel problema in esame, non essendoci evidenze di particolari scostamenti.

In Figura 7.6 è invece rappresentata la misura dell'errore relativo per i diversi parametri del modello, calcolata confrontando i parametri calibrati tramite l'algoritmo Levenberg-Marquardt con quelli originali del test set.

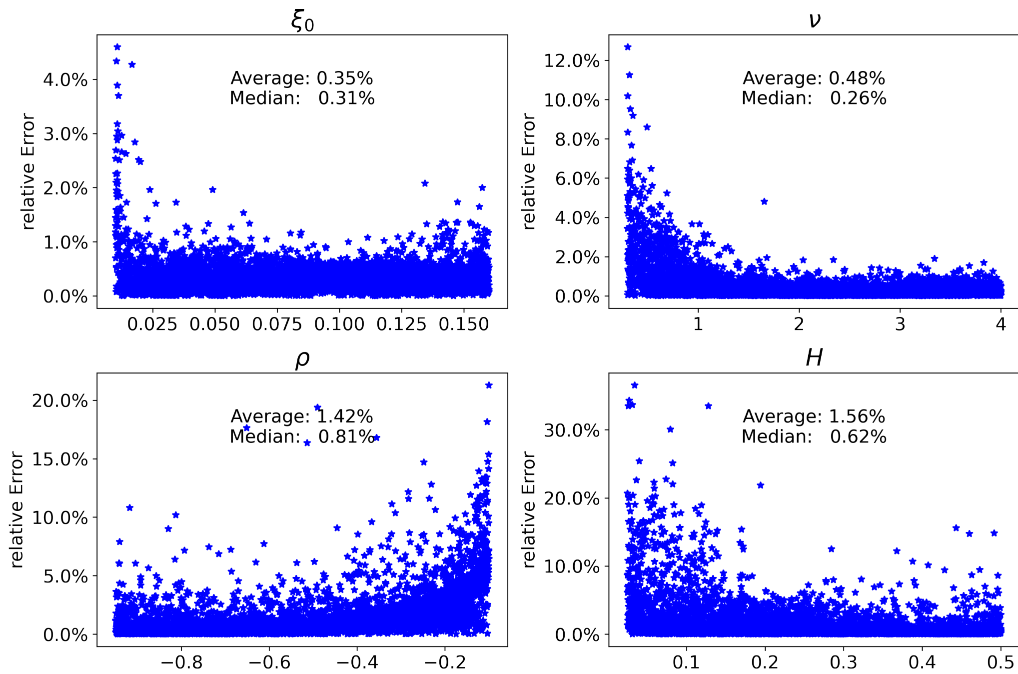


Figura 7.6: Errore relativo dei diversi parametri del modello rough Bergomi

Si osserva un'ottima approssimazione dei parametri calibrati rispetto agli originali. L'errore relativo, calcolato sul test set, ha il valore medio sotto 1.60% e il valore mediano sotto 0.9% per tutti i parametri del modello. Inoltre le maggiori differenze si riscontrano per valori dei parametri prossimi ai limiti massimi e minimi dei parametri.

7.2.2 VGSSD

L'analisi è ripetuta per i due modelli auto similari: qui è riportato il modello VGSSD, mentre il modello NIGSSD è disponibile in Appendice A.

Si osserva in Figura 7.7 che l'errore medio si mantiene sotto lo 0.25%, mentre si registra per un punto della griglia Δ un errore massimo in termini

Figura 7.7: Errore relativo tra dati di input e stima della rete neurale: errore medio, deviazione standard, errore massimo

relativi di circa 14%, sebbene per il resto dei punti la misura di errore si mantenga ampiamente più limitata.

Come per il modello rough Bergomi, in Figura 7.8 è rappresentato un confronto tra i dati di input e l'approssimazione generata dalla rete neurale. Anche in questo caso la superficie è scelta a scopo esemplificativo.

Inoltre, in Figura 7.9 sono rappresentati gli *smile* di volatilità relativi alle diverse scadenze considerate, per i dati di input e per l'approssimazione della rete neurale.

Come per il modello precedente, le rappresentazioni in Figura 7.8 e in Figura 7.9 evidenziano come la rete neurale sia in grado di approssimare efficacemente curve e superfici nel problema in esame.

Infine in Figura 7.10 è riportata la misura dell'errore relativo per i diversi parametri del modello, calcolata confrontando i parametri calibrati con quelli originali, rispetto al test set.

Si osserva che l'errore relativo varia per i diversi parametri, con un massimo di 5.33% per il parametro θ. Relativamente ai valori mediani dell'errore, questi sono notevolmente più limitati, con un valore massimo di 0.35% per il parametro ν. Tale differenza è spiegata dal fatto che l'errore non è distribuito in maniera uniforme, ma alcuni valori, specie al limite dell'intervallo di definizione dei parametri, sono particolarmente elevati.

7.3 Modelli con rete a 4 hidden layers

Per quanto riguarda i modelli con approssimazione costante a tratti della curva forward variance, la metodologia di analisi è la stessa, sebbene siano

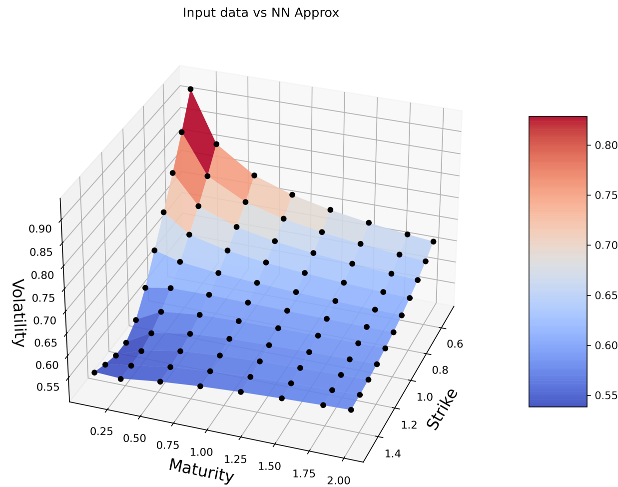


Figura 7.8: Rappresentazione di una superficie di volatilità. La superficie rappresenta i dati relativi al modello di pricing, i punti disegnati in nero i risultati dell'approssimazione della rete neurale

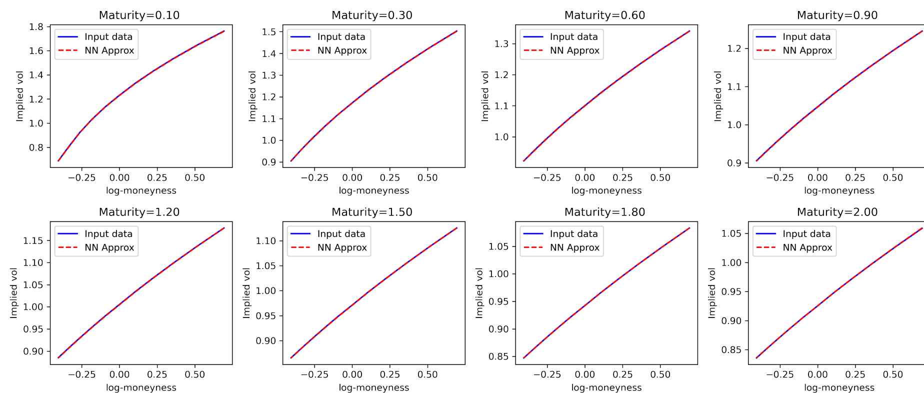


Figura 7.9: Smile relativi alle diverse maturity considerate. Sono confrontati i dati di input (in blu) con l'approssimazione generata dalla rete neurale (in rosso).

in gioco 11 parametri di modello (di cui 8 per l'individuazione della curva

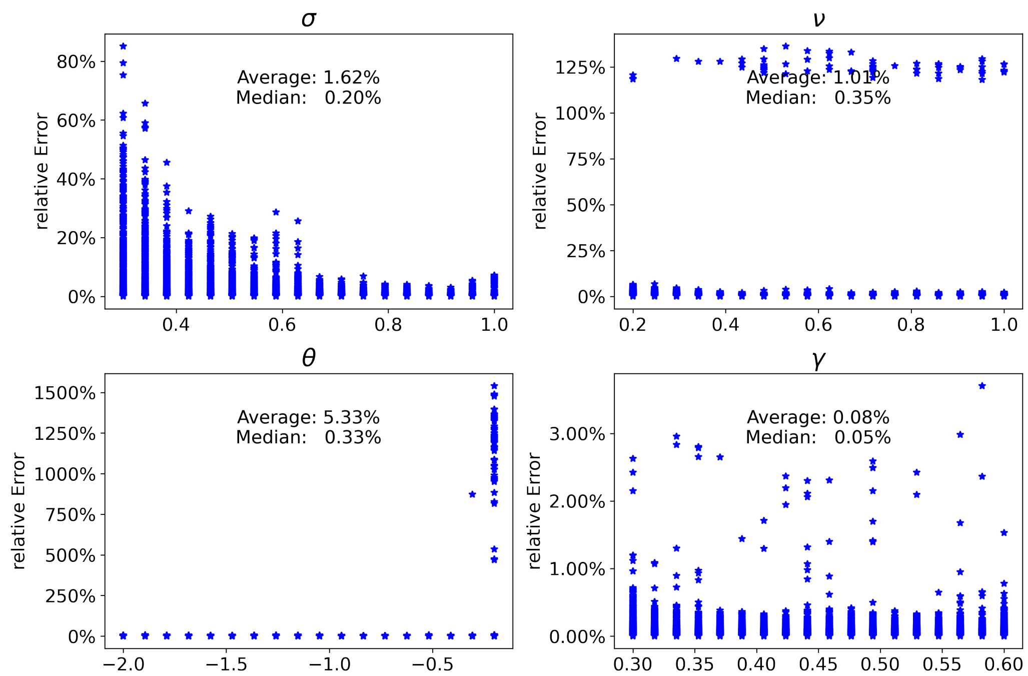


Figura 7.10: Errore relativo dei diversi parametri del modello VGSSD

costante a tratti). Come esempio è riportato il modello rough Bergomi.

7.3.1 Rough Bergomi

Di seguito l'errore di stima relativo per la volatilità, relativamente ai diversi punti della grid Δ.

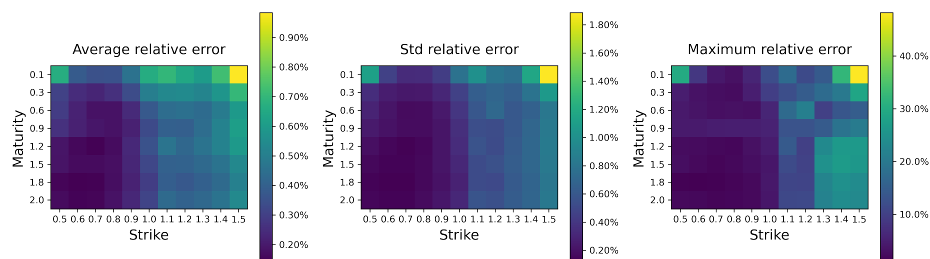


Figura 7.11: Errore relativo tra dati di input e stima della rete neurale: errore medio, deviazione standard, errore massimo

Si osserva in Figura 7.11 che l'errore medio si mantiene sotto l'1.0%, mentre si registra per un punto della griglia Δ un errore massimo in termini

relativi di circa 40%, sebbene per il resto dei punti la misura di errore si mantenga più limitata. Una differenza maggiore poteva essere prevedibile, essendo il modello più strutturato.

Come per i modelli a 3 hidden layer, in Figura 7.12 è rappresentato un confronto tra i dati di input e l'approssimazione generata dalla rete neurale. Anche in questo caso la superficie è presa a scopo esemplificativo.

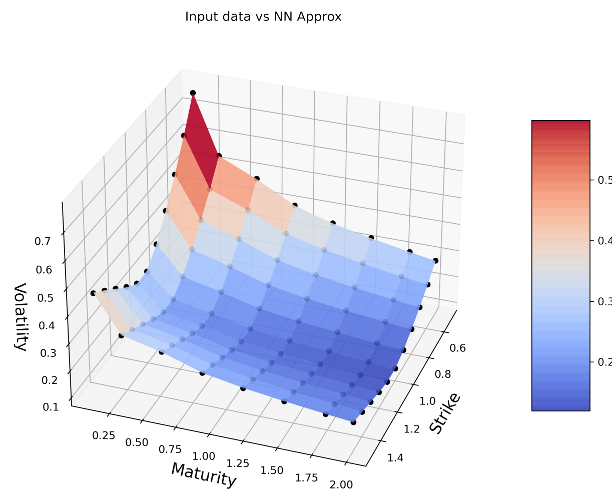


Figura 7.12: Rappresentazione di una superficie di volatilità. La superficie rappresenta i dati relativi al modello di pricing, i punti disegnati in nero i risultati dell'approssimazione della rete neurale

Inoltre, in Figura 7.13 sono rappresentati gli *smile* di volatilità relativi alle diverse scadenze considerate, per i dati di input e per l'approssimazione della rete neurale. Anche in questo caso le curve sono prese a scopo esemplificativo.

Le rappresentazioni in Figura 7.12 e in Figura 7.13 evidenziano come la rete neurale sia in grado di approssimare efficacemente curve e superfici nel problema in esame, sebbene si registrino dei lievi scostamenti nella replicazione degli smile di volatilità.

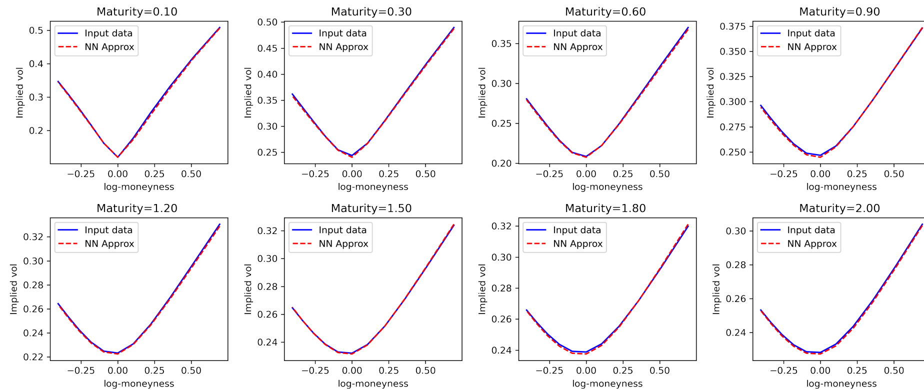


Figura 7.13: Smile relativi alle diverse maturity considerate. Sono confrontati i dati di input (in blu) con l'approssimazione generata dalla rete neurale (in rosso).

Infine, in Figura 7.14 è riportata la misura dell'errore relativo per i diversi parametri del modello, calcolata sul test set confrontando i parametri calibrati con quelli originali.

Come già visto per il modello VGSSD, gli errori relativi presentano valori di media e mediana non allineati e una distribuzione non uniforme. Rispetto ai modelli visti in precedenza si registra una minore qualità della stima, con errori relativi medi che raggiungono un picco di 11.94% per il parametro ξ_8 (uno dei parametri che individua la curva costante a tratti) e un valore mediano massimo di 4.28%, sempre per il parametro ξ_8.

7.4 Commenti

Dalla misura degli errori dei modelli risultano alcune evidenze. In primo luogo ci sono degli aspetti comuni da segnalare. Infatti, si può osservare che per tutti i modelli gli errori maggiormente significativi si presentano nei punti limite della griglia Δ, in particolare per il valore minimo di maturity e i valori estremi di moneyness. Questo può essere dovuto al fatto che, utilizzando un metodo image-based, per tale combinazione l'approssimazione è imprecisa avendo meno punti adiacenti rispetto ad un punto centrale della griglia. Un ulteriore dettaglio comune a tutti i modelli è il fatto che l'errore rilevato dal confronto dei parametri originali con quelli calibrati è eterogeneo sulle diverse discretizzazioni dei parametri, presentando errori maggiori per i valori limite dei parametri stessi. Questo è probabilmente dovuto al fatto che uscendo

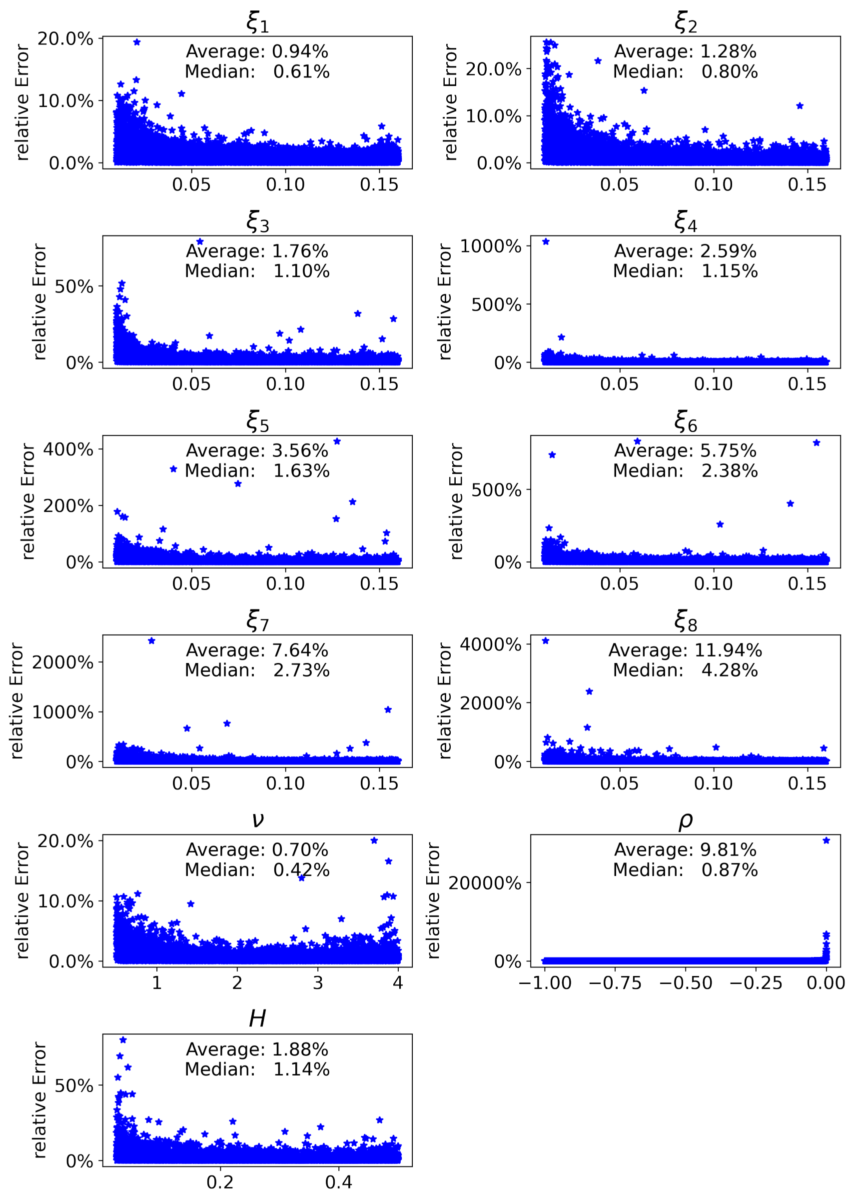


Figura 7.14: Errore relativo dei diversi parametri del modello rBergomi

dall'intorno preimpostato per i parametri l'approssimazione diventa meno precisa, essendo la mappa ben approssimata solo per i valori dei parametri compresi nei limiti prestabiliti. Tale errore può essere ridotto ampliando l'intervallo di definizione, andando a comprendere un intervallo più ampio rispetto ai valori dei parametri solitamente riscontrati nel mercato.

Venendo alle differenze dei vari modelli, sebbene la stima può essere ritenuta soddisfacente per tutti i modelli, sia dal punto di vista dell'approssimazione della superficie di volatilità, sia per quanto riguarda l'errore di calibrazione dei parametri rispetto ai valori originali, i modelli più semplici, stimati da una rete a 3 hidden layer, risultano meglio approssimati. Qualora fosse necessaria ai fini di uno sviluppo industriale una maggior precisione della stima, tale differenza può essere limata indagando in maniera più approfondita i modelli più complessi, con una ricerca più dettagliata della rete più adatta ad imparare la mappa tra parametri e superficie di volatilità per i modelli più complessi, andando quindi a operare ulteriori migliorie alla rete scelta.

Capitolo 8

Direzioni future di ricerca e conclusioni

In questo capitolo sono esposte le conclusioni e ipotizzate delle possibili future direzioni di ricerca.

8.1 Conclusioni

Come confermato dalle analisi riportate nel Capitolo 7 e nell'Appendice A, il metodo presentato, basato sull'utilizzo di una rete neurale, si dimostra efficace nell'approssimazione della mappa tra parametri del modello e la relativa superficie di volatilità. Inoltre, tale approssimazione della mappa risulta sufficientemente adeguata per ottenere una calibrazione dei parametri di modello vicina ai parametri originali. Il Teorema 4.4.1 e il Teorema 4.4.2, che dimostrano la capacità della rete neurale di approssimare una qualsiasi funzione entro certi limiti (ma certamente comprendendo il mapping tra parametri e volatilità), sono la base teorica su cui è possibile affermare che per un qualsiasi modello di pricing è possibile utilizzare una tecnica analoga a quella presentata in questa tesi. L'evidenza empirica suggerisce che per i modelli aventi una complessità fino a 4 o 5 parametri la rete neurale a 3 hidden layer ipotizzata in questa tesi sia adeguata per ottenere una stima sufficiente. Questa tuttavia è un'ipotesi sostenuta unicamente da dati empirici e non può, per il momento, essere generalizzata ad ogni modello di questa complessità. Tuttavia, se questa ipotesi dovesse essere confermata, la funzione di strutturazione e training della rete neurale `GeneralizedNNCalibration` per-

metterebbe di risolvere il problema di calibrazione di qualsiasi modello della complessità menzionata, permettendo di aggirare la barriera dovuta agli alti costi computazionali di alcuni modelli, aprendo all'utilizzo di modelli dotati di tecniche di pricing non computazionalmente efficienti.

8.1.1 Limiti del lavoro

Il lavoro riportato ha certamente delle limitazioni, che elenchiamo di seguito.

In primo luogo, i dataset generati potrebbero essere ulteriormente ampliati, aumentando la granularità delle suddivisioni e allargando l'intervallo di definizione dei parametri di modello. Tale approfondimento tuttavia richiederebbe un elevato costo computazionale, non sostenibile dalla macchina disposizione durante la stesura di questo lavoro.

Inoltre, la creazione dei dataset è un'operazione computazionalmente costosa, richiedendo di svolgere il processo di pricing numerose volte. In particolare, se il numero di combinazioni analizzate è molto esteso, tale processo può essere estremamente dispendioso e possibile solo con una strumentazione tecnologica adeguata. Tuttavia, qualora fosse presa la decisione di adottare questo metodo in un'applicazione industriale, è generalmente possibile contare su una strumentazione all'altezza.

Infine, nella stima dei modelli più complessi approssimati con una rete a 4 hidden layer, l'idea di generalizzare il processo è più remota. Questo è dovuto al fatto che la stima è generalmente più imprecisa rispetto agli altri modelli analizzati. Inoltre, la gran numerosità di parametri implica che per ottenere una sufficiente granularità nella suddivisione degli intervalli di definizione dei parametri sia richiesto un costo computazionale enormemente più alto.

8.2 Direzioni future di ricerca

8.2.1 Scelta del modello più adatto

Diverse condizioni di mercato si traducono in diverse superfici di volatilità. In generale, non è vero che ciascun modello è ugualmente adatto per la calibrazione di ogni superficie di volatilità. A seconda delle proprie caratteristiche, un modello sarà più o meno adatto a seconda delle diverse condizioni di mercato. Si riporta di seguito una rapida analisi per affrontare questo tema. Sulla scia di quanto introdotto nella sezione finale di Horvath, Muguruza e

Tomas [30], è stata costruita ed allenata una rete neurale[1] capace di riconoscere da quale modello sono costruite le diverse superfici presenti nel dataset. In questa fase l'analisi si è limitata ai soli modelli approssimati da una rete a 3 hidden layer. La rete neurale è allenata partendo dal dataset già presentato per i modelli selezionati, il quale è suddiviso in un 70% di train (a sua volta diviso in un 80% di train effetivo e un 20% di validation) e un 30% di test. I risultati della rete sul test set mostrano un'ottima capacità predittiva, con un'accuracy di 99.03% con una confusion matrix rappresentata in Figura 8.1.

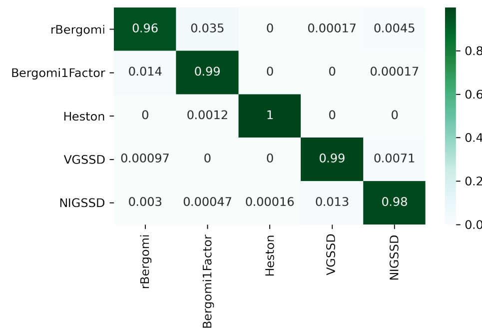


Figura 8.1: Confusion matrix del problema di individuazione del modello generatore

L'ottima misura della stima prodotta può suggerire un'applicazione. Infatti, tale tecnica potrebbe essere uno strumento utile per rispondere alla domanda "Qual è il modello più adatto alle attuali condizioni di mercato?", se si riuscisse a mostrare che il modello individuato dalla rete neurale coincide con il modello che ha un minor errore di calibrazione. Per fare ciò sono stati svolti alcuni test. Per ogni superficie di volatilità implicita nel test set, grazie alla rapidità del processo di calibrazione descritto in questa tesi, è stato individuato il modello con il minimo errore di calibrazione. Lo stesso processo è stato svolto anche su delle superfici di volatilità ricavate a partire da quelle del test set inserendo una componente di rumore in tre modi diversi, elencati di seguito.

1. **Shift Parallelo:** è applicato uno shift parallelo a tutte le superfici del campione, $\sigma_{t,k}^{shift} = \sigma_{t,k}^{base} + \epsilon, \forall t, k$, con $\epsilon = 10^{-4}$;

2. **Rumore gaussiano 1:** è applicato un rumore gaussiano a tutte le superfici del campione, $\sigma_{t,k}^{rand} = \sigma_{t,k}^{base} + \epsilon \cdot z, \forall t, k$, con $\epsilon = 10^{-4}$ e dove

[1] In Appendice D è presente una descrizione della rete utilizzata.

z è estratto da una distribuzione normale standard. Per semplicità di calcolo sono stati estratti 88 valori dalla distribuzione normale standard e sono stati applicati a tutto il campione;

3. **Rumore gaussiano 2:** è applicato un rumore gaussiano a tutte le superfici del campione, come visto al punto precedente, ma utilizzando $\epsilon = 10^{-2}$

Per ciascuno dei quattro campioni (il test set originale e i tre con l'aggiunta del rumore), è stata utilizzata la rete neurale per misurare se questa fosse in grado di identificare i modelli a minimo errore di calibrazione. Il risultato dell'applicazione al test set originale mostra un'accuracy pari a 95.71%, con una confusion matrix riportata in Figura 8.2.

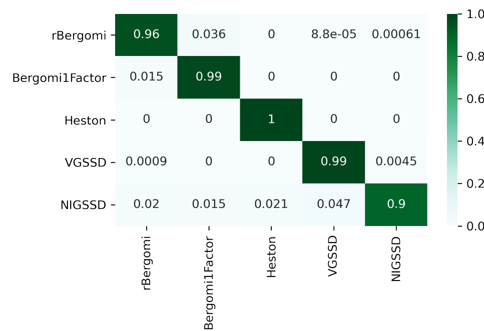
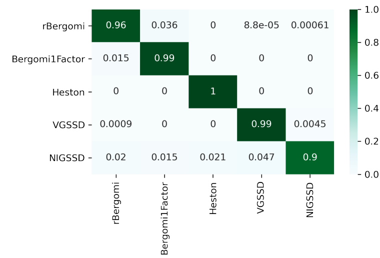


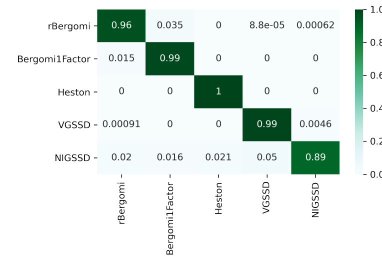
Figura 8.2: Confusion matrix del problema di individuazione del modello a minimo errore

Per i tre modelli con rumore le accuracy risultano essere rispettivamente 95.69%, 95.49% e 40.12% e le confusion matrix sono riportate in Figura 8.3.

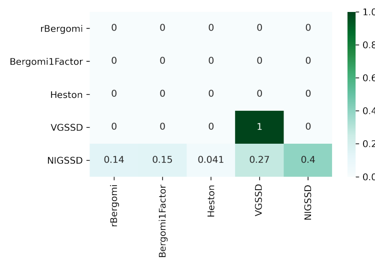
Dal risultato ottenuto sul test set originale si osserva in prima analisi che il modello generatore risulta essere, con ottima approssimazione, il modello con minor errore di calibrazione. La non perfetta corrispondenza può essere dovuta alle piccole approssimazioni svolte per giungere al processo di calibrazione. Dai risultati sui campioni con l'aggiunta di rumore si evidenzia che per i primi due campioni non ci sono sostanziali differenze, la rete neurale è in grado di categorizzare correttamente le superfici di volatilità. Sebbene ampiamente più precisa della scelta casuale, per quanto riguarda l'ultimo campione, la stima è imperfetta. Questo è primariamente dovuto al fatto che per la quasi totalità delle superfici dell'ultimo campione il modello che meglio



(a) Shift Parallelo



(b) Rumore gaussiano 1



(c) Rumore gaussiano 2

Figura 8.3: Andamento della loss al passare delle epoch

stima risulta essere il NIGSSD, il che potrebbe essere dovuto alla maggiore qualità dell'approssimazione tramite rete neurale a 3 hidden layer utilizzata, come potrebbe essere causata da una maggiore versatilità del modello.

In conclusione, questa analisi preliminare mostra come sia possibile allenare una rete neurale per riconoscere il modello da cui è generata una superficie di volatilità e come questo stesso metodo sia utile per identificare superfici di volatilità che differiscono di un piccolo rumore da quelle generate da uno dei modelli in esame. Il metodo non risulta al momento efficace per superfici che sono distanti da quelle descritte nel campione. Si potrebbe indagare se ampliando il numero di modelli in esame e ampliando il campione di calibrazione dei singoli modelli utilizzati tale metodo può risultare efficace. Il risultato finale potrebbe essere un processo di calibrazione di una superficie di volatilità in due passi. Come primo passo la rete neurale introdotta in questo capitolo stabilisce quale modello è il maggiormente adatto a calibrare la superficie di volatilità scelta, come secondo passo viene calibrato il modello individuato come descritto in questa tesi.

8.2.2 Un unico metodo di calibrazione

L'applicazione del metodo ai due modelli auto similari ha allargato il paniere di modelli per cui questa tecnica ha successo, con l'applicazione di una stessa rete neurale. L'obiettivo finale di questa tecnica dovrebbe essere quello di riuscire a approssimare la mappa in maniera autonoma, senza che sia richiesto ulteriore *fine tuning* sulla rete utilizzata. Questo significa che semplicemente dal dataset dato in input il metodo dovrebbe essere in grado di costruire la rete neurale architettonicamente più adatta, ottenendo la miglior approssimazione dopo la fase di training. Questo vorrebbe dire avere già implementata una tecnica di calibrazione di superfici di volatilità universalmente valida, per qualsiasi modello esistente o di nuova creazione. Questo permetterebbe di superare definitivamente il problema del costo computazionale nell'operare quotidianamente la calibrazione delle superfici di volatilità, consentendo di concentrarsi sulle buone caratteristiche dei modelli, piuttosto che sul costo computazionale.

Bibliografia

[1] Miquel Noguer i Alonso e Sonam Srivastava. *Deep Reinforcement Learning for Asset Allocation in US Equities*. 2020. arXiv: `2010.04404 [q-fin.PM]`.

[2] O. Barndorff-Nielsen. "Hyperbolic Distributions and Distributions on Hyperbolae". In: *Scandinavian Journal of Statistics* 5.3 (1978), pp. 151–157.

[3] Andrew Barron. "Approximation and Estimation Bounds for Artificial Neural Networks". In: *Machine Learning* 14 (1994), pp. 115–133.

[4] Christian Bayer e Benjamin Stemper. *Deep calibration of rough stochastic volatility models*. 2018. arXiv: `1810.03399 [q-fin.PR]`.

[5] Lorenzo Bergomi. *Stochastic Volatility Modeling*. Chapman & Hall/CRC, 2015.

[6] Tomas Bjork. *Arbitrage Theory in Continuous Time*. Oxford: Oxford University Press, 2009. ISBN: 9780199574742.

[7] Fischer Black e Myron Scholes. "The Pricing of Options and Corporate Liabilities". In: *Journal of Political Economy* 81.3 (1973), pp. 637–654.

[8] C. G. Broyden. "The Convergence of a Class of Double-rank Minimization Algorithms 1. General Considerations". In: *IMA Journal of Applied Mathematics* 6.1 (mar. 1970), pp. 76–90.

[9] H. Buehler, L. Gonon, J. Teichmann e B. Wood. "Deep hedging". In: *Quantitative Finance* 19.8 (2019), pp. 1271–1291.

[10] Peter Carr, Hélyette Geman, Dilip Madan e Marc Yor. "Self - Decomposability and Option Pricing". In: *Mathematical Finance* 17 (dic. 2006).

[11] Peter Carr e Dilip Madan. "Option Valuation Using the Fast Fourier Transform". In: *Journal of Computational Finance* 2 (1998), pp. 61–73.

[12] Djork-Arné Clevert, Thomas Unterthiner e Sepp Hochreiter. *Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs).* 2016. arXiv: `1511.07289 [cs.LG]`.

[13] Rama Cont e Peter Tankov. "Financial modelling with jump processes Chapman & Hall/CRC". In: (gen. 2004).

[14] C. Doléans-Dade. "Quelques applications de la formule de changement de variables pour les semimartingales". In: *Zeitschrift für Wahrscheinlichkeitstheorie und Verwandte Gebiete* 16 (1970), pp. 181–194.

[15] Ronen Eldan e Ohad Shamir. *The Power of Depth for Feedforward Neural Networks*. 2016. arXiv: `1512.03965 [cs.LG]`.

[16] Ryan Ferguson e Andrew Green. *Deeply Learning Derivatives*. 2018. arXiv: `1809.02233 [q-fin.CP]`.

[17] R. Fletcher. "A new approach to variable metric algorithms". In: *The Computer Journal* 13.3 (gen. 1970), pp. 317–322.

[18] Raquel M. Gaspar, Sara D. Lopes e Bernardo Sequeira. "Neural Network Pricing of American Put Options". In: *Risks* 8.3 (2020).

[19] Jim Gatheral. "A parsimonious arbitrage-free implied volatility parameterization with application to the valuation of volatility derivatives". In: 2004.

[20] Jim Gatheral e Antoine Jacquier. "Arbitrage-free SVI volatility surfaces". In: *Quantitative Finance* 14.1 (2014), pp. 59–71.

[21] Jim Gatheral, Thibault Jaisson e Mathieu Rosenbaum. "Volatility is rough". In: *Quantitative Finance* 18.6 (2018), pp. 933–949.

[22] Donald Goldfarb. "A Family of Variable-Metric Methods Derived by Variational Means". In: *Mathematics of Computation* 24.109 (1970), pp. 23–26.

[23] Ian Goodfellow, Yoshua Bengio e Aaron Courville. *Deep Learning*. `http://www.deeplearningbook.org`. MIT Press, 2016.

[24] Andrés Hernández. "Model Calibration with Neural Networks". In: *Neuroeconomics eJournal* (2016).

[25] Steven Heston. "A Closed-Form Solution for Options with Stochastic Volatility with Applications to Bond and Currency Options". In: *Review of Financial Studies* 6 (1993), pp. 327–343.

[26] K. Hornik, M. Stinchcombe e H. White. "Universal approximation of an unknown mapping and its derivatives using multilayer feedforward networks". In: *Neural Networks* 3 (1990), pp. 551–560.

[27] Kurt Hornik. "Approximation capabilities of multilayer feedforward networks". In: *Neural Networks* 4.2 (1991), pp. 251–257.

[28] Kurt Hornik, Maxwell Stinchcombe e Halbert White. "Multilayer feedforward networks are universal approximators". In: *Neural Networks* 2.5 (1989), pp. 359–366.

[29] Blanka Horvath, Antoine Jacquier e Aitor Muguruza. *Functional central limit theorems for rough volatility.* 2019. arXiv: 1711.03078.

[30] Blanka Horvath, Aitor Muguruza e Mehdi Tomas. "Deep learning volatility: a deep neural network perspective on pricing and calibration in (rough) volatility models". In: *Quantitative Finance* (2020), pp. 11–27.

[31] John Hull. *Options, Futures, and Other Derivatives.* Upper Saddle River, NJ: Prentice Hall, 2008.

[32] Diederik P. Kingma e Jimmy Ba. *Adam: A Method for Stochastic Optimization.* 2017. arXiv: 1412.6980 [cs.LG].

[33] Dieter Kraft. "Algorithm 733: TOMP–Fortran modules for optimal control calculations". In: 20.3 (set. 1994), pp. 262–281.

[34] Kenneth Levenberg. "A Method for the Solution of Certain Non-Linear Problems in Least Squares". In: *Quarterly of Applied Mathematics* 2.2 (1944), pp. 164–168.

[35] Alan Lewis. "A Simple Option Formula for General Jump-Diffusion and Other Exponential Levy Processes". In: *SSRN Electronic Journal* (mag. 2002).

[36] Eugene Lukacs. "A Survey of the Theory of Characteristic Functions". In: *Advances in Applied Probability* 4.1 (1972), pp. 1–38.

[37] Cuicui Luo, Desheng Wu e Dexiang Wu. "A deep learning approach for credit scoring using credit default swaps". In: *Engineering Applications of Artificial Intelligence* 65 (2017), pp. 465–470.

[38] Dilip B. Madan, Peter P. Carr e Eric C. Chang. "The Variance Gamma Process and Option Pricing". In: *Review of Finance* 2.1 (apr. 1998), pp. 79–105.

[39] Mary Malliaris e Linda Salchenberger. “A neural network model for estimating option prices”. In: *Appl. Intell.* 3 (set. 1993), pp. 193–206.

[40] Donald W. Marquardt. “An Algorithm for Least-Squares Estimation of Nonlinear Parameters”. In: *Journal of the Society for Industrial and Applied Mathematics* 11.2 (1963), pp. 431–441.

[41] Uri Shaham, Alexander Cloninger e Ronald R. Coifman. “Provable approximation properties for deep neural networks”. In: *Applied and Computational Harmonic Analysis* 44.3 (mag. 2018), pp. 537–557.

[42] D. F. Shanno. “Conditioning of Quasi-Newton Methods for Function Minimization”. In: *Mathematics of Computation* 24.111 (1970), pp. 647–656.

[43] Justin Sirignano, Apaar Sadhwani e Kay Giesecke. *Deep Learning for Mortgage Risk*. 2018. arXiv: `1607.02470 [q-fin.ST]`.

[44] Jan Spiegeleer, Dilip Madan, Sofie Reyners e Wim Schoutens. “Machine learning for quantitative finance: fast derivative pricing, hedging and fitting”. In: *Quantitative Finance* 18 (lug. 2018), pp. 1–9.

[45] Henry Stone. *Calibrating rough volatility models: a convolutional neural network approach*. 2019. arXiv: `1812.05315 [q-fin.CP]`.

[46] Ciyou Zhu, Richard H. Byrd, Peihuang Lu e Jorge Nocedal. “Algorithm 778: L-BFGS-B: Fortran Subroutines for Large-Scale Bound-Constrained Optimization”. In: *ACM Trans. Math. Softw.* 23.4 (dic. 1997), pp. 550–560.

Appendice A

Risultati per i modelli non riportati

Questa appendice è dedicata a riportare i risultati numerici per i modelli non trattati nel Capitolo 7. La prima parte è relativa all'analisi sulla scelta della migliore architettura della rete neurale da utilizzare, la seconda è relativa alle analisi sulla qualità della stima.

A.1 Analisi sull'architettura utilizzata

Si riporta di seguito la Tabella A.1 che riassume i valori di loss ottenuti sul test set per i modelli restanti, escluso il modello 1-Factor Bergomi con approssimazione costante a tratti della curva di forward variance, che è riportato in Tabella A.2.

Tabella A.1: Valori di loss per i diversi modelli in esame

Modello	Rete a 3 Layer			Rete a 4 Layer		
	Training	Validation	Test	Training	Validation	Test
1-Factor Bergomi	0.0086	0.0072	0.0068	0.0072	0.0070	0.0064
Heston	0.0251	0.0255	0.0242	0.0236	0.0254	0.0246
VGSSD	0.0047	0.0032	0.0031	0.0050	0.0034	0.0034
NIGSSD	0.0046	0.0036	0.0031	0.0051	0.0033	0.0030

Tabella A.2: Valori di loss per le diverse architetture

Rete a 3 Layer			Rete a 4 Layer		
Training	Validation	Test	Training	Validation	Test
0.0201	0.0202	0.0214	0.0187	0.0180	0.0179

Rete a 5 Layer		
Training	Validation	Test
0.0177	0.0172	0.0186

A.2 Modelli con rete a 3 hidden layers

Di seguito sono riportati i risultati relativi ai restanti modelli approssimati dalla rete neurale a 3 hidden layer.

A.2.1 1-Factor Bergomi

Ripetiamo le analisi con il modello 1-Factor Bergomi con approssimazione della forward variance ξ_0 costante. Riportiamo di seguito le stesse analisi già presentate per il modello 1-Factor Bergomi.

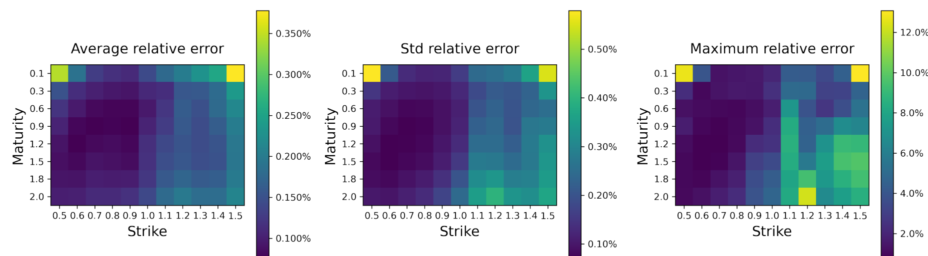


Figura A.1: Errore relativo tra dati di input e stima della rete neurale: errore medio, deviazione standard, errore massimo

Si osserva in Figura A.1 che l'errore medio si mantiene sotto lo 0.4%, mentre si registra per un punto della griglia Δ un errore massimo in termini relativi di circa 12%, sebbene per il resto dei punti la misura di errore si mantenga più limitata. In Figura A.2 è rappresentato un confronto tra i dati di input e l'approssimazione generata dalla rete neurale.

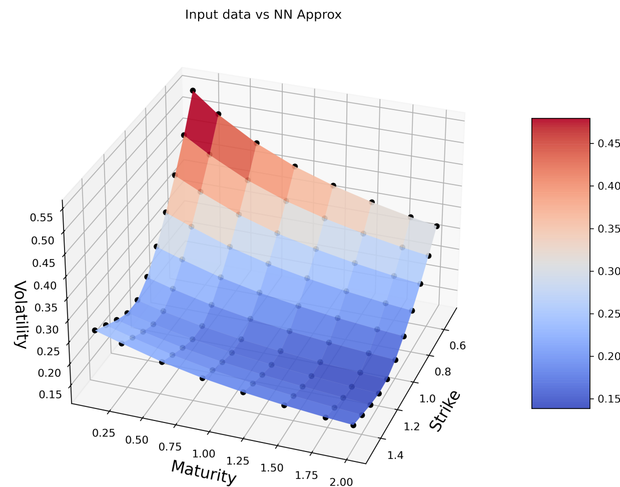


Figura A.2: Rappresentazione di una superficie di volatilità. La superficie rappresenta i dati relativi al modello di pricing, i punti neri i risultati dell'approssimazione della rete neurale

Analogamente, in Figura A.3 sono rappresentati gli *smile* di volatilità relativi alle diverse scadenze considerate, per i dati di input e per l'approssimazione della rete neurale.

Le rappresentazioni in Figura A.2 e in Figura A.3 evidenziano come la rete neurale sia in grado di approssimare efficacemente curve e superfici nel problema in esame. In Figura A.4 la misura dell'errore relativo per i diversi parametri del modello, calcolata confrontando i parametri calibrati tramite l'algoritmo Levenberg-Marquardt con quelli originali.

Si osserva che, valutando sul test set, l'errore relativo ha il valore medio sotto 5.40% e il valore mediano sotto 1.0% per tutti i parametri. Inoltre, per tutti i parametri, le maggiori differenze si riscontrano per valori dei parametri prossimi ai valori limite.

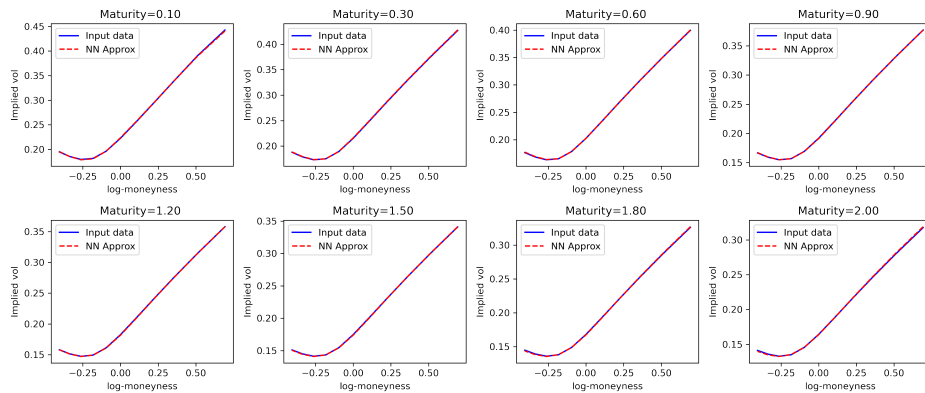


Figura A.3: Smile relativi alle diverse maturity considerate. Sono confrontati i dati di input (in blu) con l'approssimazione generata dalla rete neurale (in rosso).

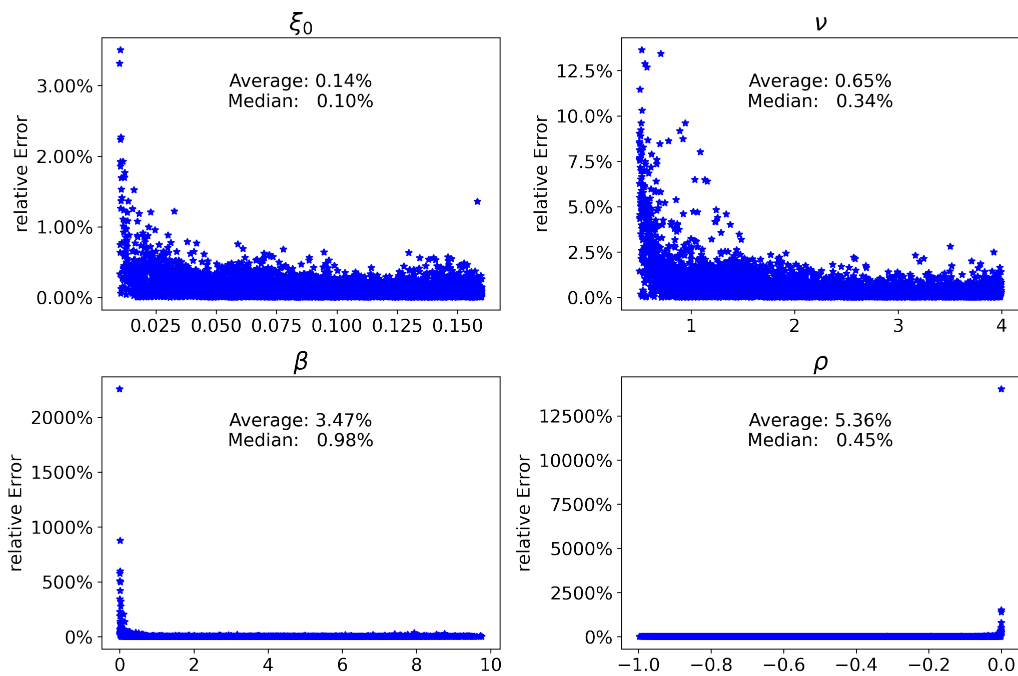


Figura A.4: Errore relativo dei diversi parametri del modello 1-Factor Bergomi

A.2.2 Heston

L'analisi è ripetuta per il modello di Heston a 5 parametri.

Si osserva in Figura A.5 che l'errore medio si mantiene sotto 0.8%, mentre si registra per un punto della griglia Δ un errore massimo in termini relativi

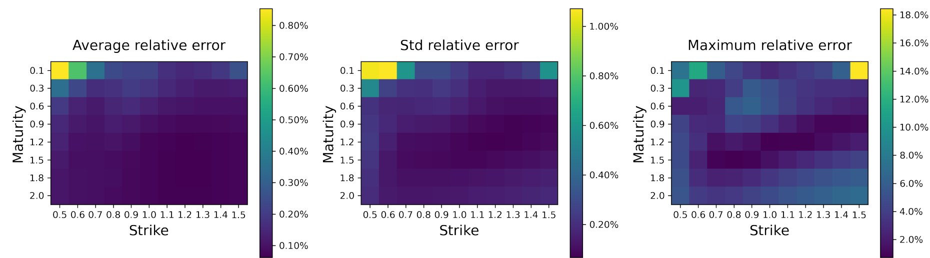


Figura A.5: Errore relativo tra dati di input e stima della rete neurale: errore medio, deviazione standard, errore massimo

di circa 18%, sebbene per il resto dei punti la misura di errore si mantenga più limitata. In Figura A.6 è rappresentato un confronto tra i dati di input e l'approssimazione generata dalla rete neurale.

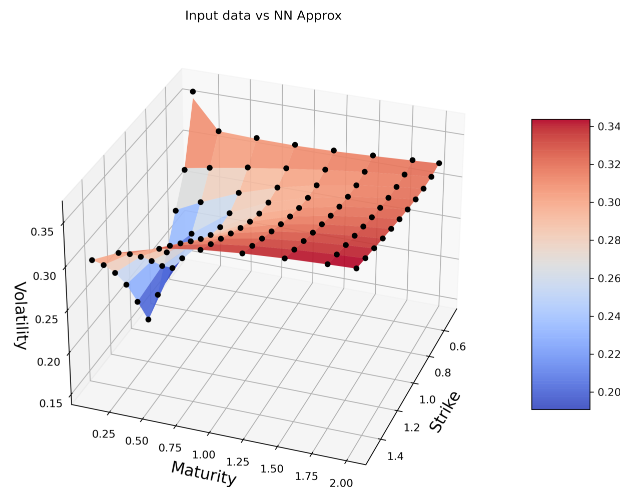


Figura A.6: Rappresentazione di una superficie di volatilità. La superficie rappresenta i dati relativi al modello di pricing, i punti neri i risultati dell'approssimazione della rete neurale

Analogamente, in Figura A.7 sono rappresentati gli *smile* di volatilità

relativi alle diverse scadenze considerate, per i dati di input e per l'approssimazione della rete neurale. Anche in questo caso le curve sono prese a scopo esplicativo.

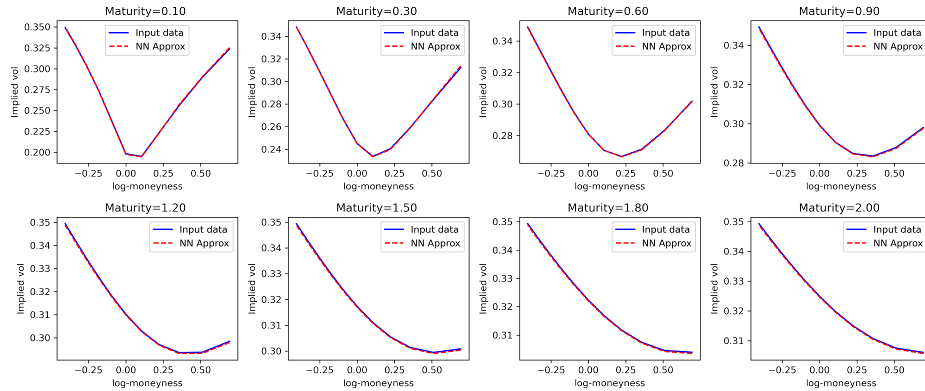


Figura A.7: Smile relativi alle diverse maturity considerate. Sono confrontati i dati di input (in blu) con l'approssimazione generata dalla rete neurale (in rosso).

Le rappresentazioni in Figura A.6 e in Figura A.7 evidenziano come la rete neurale sia in grado di approssimare efficacemente curve e superfici nel problema in esame. In Figura A.8 la misura dell'errore relativo per i diversi parametri del modello, calcolata confrontando i parametri calibrati tramite l'algoritmo Levenberg-Marquardt con quelli originali.

Si osserva che l'errore relativo varia per i diversi parametri, con un massimo di 15.10% per il parametro ξ_0. Relativamente ai valori mediani dell'errore, questi sono notevolmente più limitati, con un valore massimo di 2.49% per il parametro ξ_0. Tale differenza è spiegata dal fatto che l'errore non è distribuito in maniera uniforme, ma alcuni valori, specie al limite dell'intervallo di definizione dei parametri, sono particolarmente elevati.

A.2.3 NIGSSD

L'analisi è ripetuta per il restante modello auto similare.

Si osserva in Figura A.9 che l'errore medio si mantiene sotto lo 0.6%, mentre si registra per un punto della griglia Δ un errore massimo in termini relativi di circa 40%, sebbene per il resto dei punti la misura di errore si mantenga più limitata. In Figura A.10 è rappresentato un confronto tra i dati di input e l'approssimazione generata dalla rete neurale. La superficie non è stata scelta, ma è presa a scopo esemplificativo.

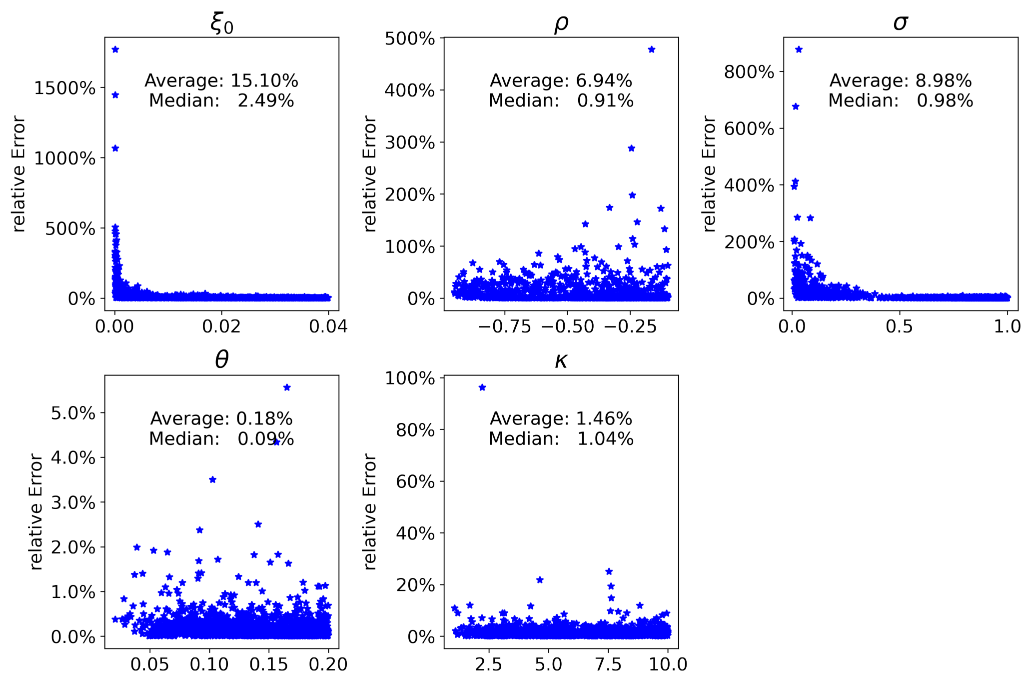


Figura A.8: Errore relativo dei diversi parametri del modello Heston

Figura A.9: Errore relativo tra dati di input e stima della rete neurale: errore medio, deviazione standard, errore massimo

Analogamente, in Figura A.11 sono rappresentati gli *smile* di volatilità relativi alle diverse scadenze considerate, per i dati di input e per l'approssimazione della rete neurale. Anche in questo caso le curve sono prese a scopo esplicativo.

Le rappresentazioni in Figura A.10 e in Figura A.11 evidenziano come la rete neurale sia in grado di approssimare efficacemente curve e superfici nel problema in esame. In Figura A.12 la misura dell'errore relativo per i diversi parametri del modello, calcolata confrontando i parametri calibrati tramite

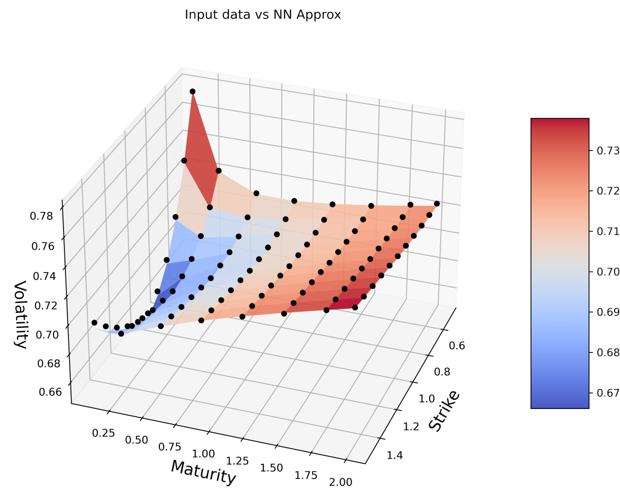


Figura A.10: Rappresentazione di una superficie di volatilità. La superficie rappresenta i dati relativi al modello di pricing, i punti neri i risultati dell'approssimazione della rete neurale

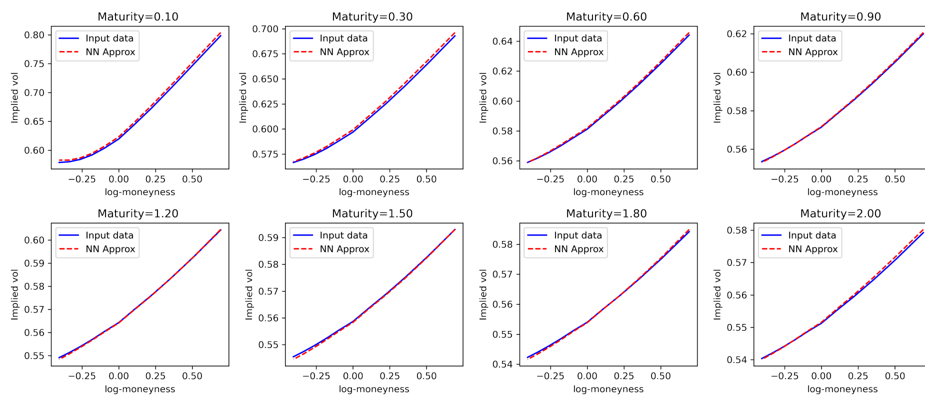


Figura A.11: Smile relativi alle diverse maturity considerate. Sono confrontati i dati di input (in blu) con l'approssimazione generata dalla rete neurale (in rosso).

l'algoritmo Levenberg-Marquardt con quelli originali.

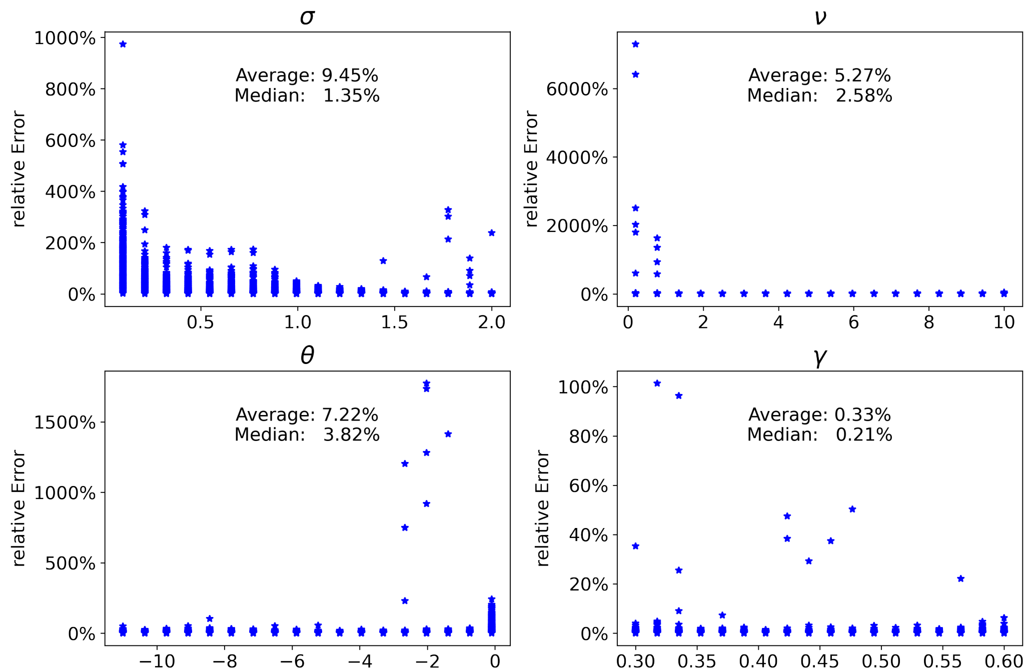


Figura A.12: Errore relativo dei diversi parametri del modello NIGSSD

Si osserva che l'errore relativo varia per i diversi parametri, con un massimo di 9.45% per il parametro σ. Relativamente ai valori mediani dell'errore, questi sono più limitati, con un valore massimo di 3.82% per il parametro θ. Tale differenza è spiegata dal fatto che l'errore non è distribuito in maniera uniforme, ma alcuni valori, specie al limite dell'intervallo di definizione dei parametri, sono particolarmente elevati.

A.3 Modelli con rete a 4 hidden layers

Di seguito sono riportati i risultati per il rimanente modello con curva forward variance approssimata con una funzione costante a tratti.

A.3.1 1-Factor Bergomi

Si osserva in Figura A.13 che l'errore medio si mantiene sotto lo 0.6%, mentre si registra per un punto della griglia Δ un errore massimo in termini relativi di circa 14%, sebbene per il resto dei punti la misura di errore si mantenga più limitata. In Figura A.14 è rappresentato un confronto tra i dati di input

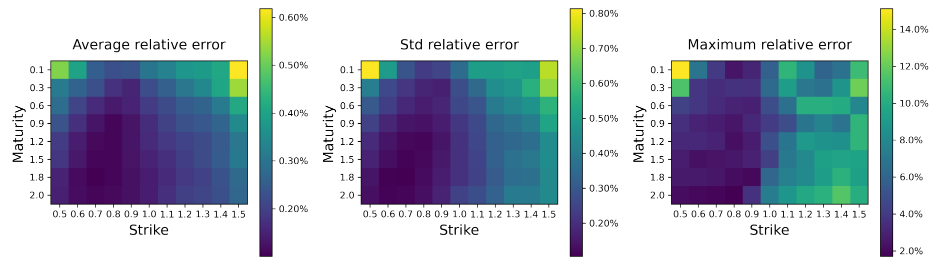


Figura A.13: Errore relativo tra dati di input e stima della rete neurale: errore medio, deviazione standard, errore massimo

e l'approssimazione generata dalla rete neurale. La superficie non è stata scelta, ma è presa a scopo esemplificativo.

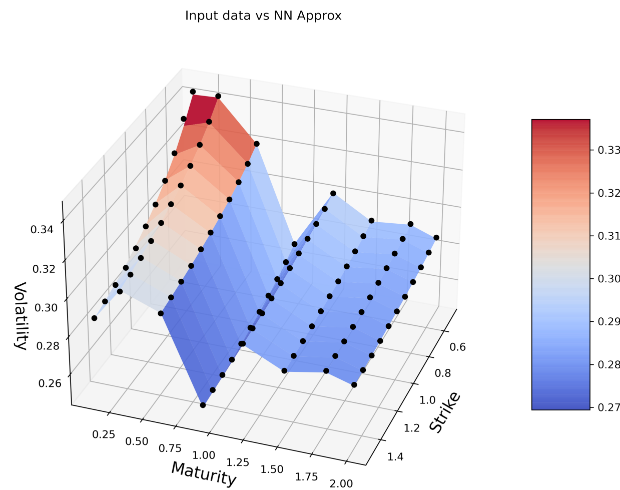


Figura A.14: Rappresentazione di una superficie di volatilità. La superficie rappresenta i dati relativi al modello di pricing, i punti neri i risultati dell'approssimazione della rete neurale

Analogamente, in Figura A.15 sono rappresentati gli *smile* di volatilità relativi alle diverse scadenze considerate, per i dati di input e per l'approssi-

mazione della rete neurale. Anche in questo caso le curve sono prese a scopo esplicativo.

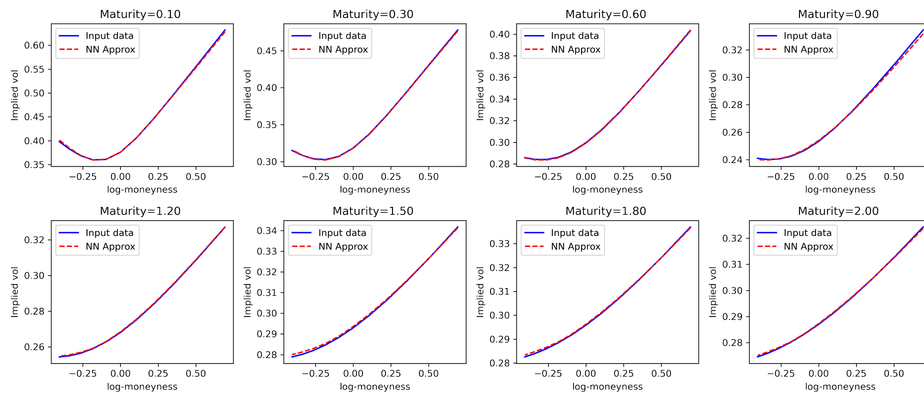


Figura A.15: Smile relativi alle diverse maturity considerate. Sono confrontati i dati di input (in blu) con l'approssimazione generata dalla rete neurale (in rosso).

Le rappresentazioni in Figura A.14 e in Figura A.15 evidenziano come la rete neurale sia in grado di approssimare efficacemente curve e superfici nel problema in esame. In Figura A.16 la misura dell'errore relativo per i diversi parametri del modello, calcolata confrontando i parametri calibrati tramite l'algoritmo Levenberg-Marquardt con quelli originali per il test set.

Si osserva che l'errore relativo varia per i diversi parametri, con un massimo di 5.65% per il parametro β. Relativamente ai valori mediani dell'errore, questi sono notevolmente più limitati, con un valore massimo di 1.48% per lo stesso parametro. Tale differenza è spiegata dal fatto che l'errore non è distribuito in maniera uniforme, ma alcuni valori, specie al limite dell'intervallo di definizione dei parametri, sono particolarmente elevati.

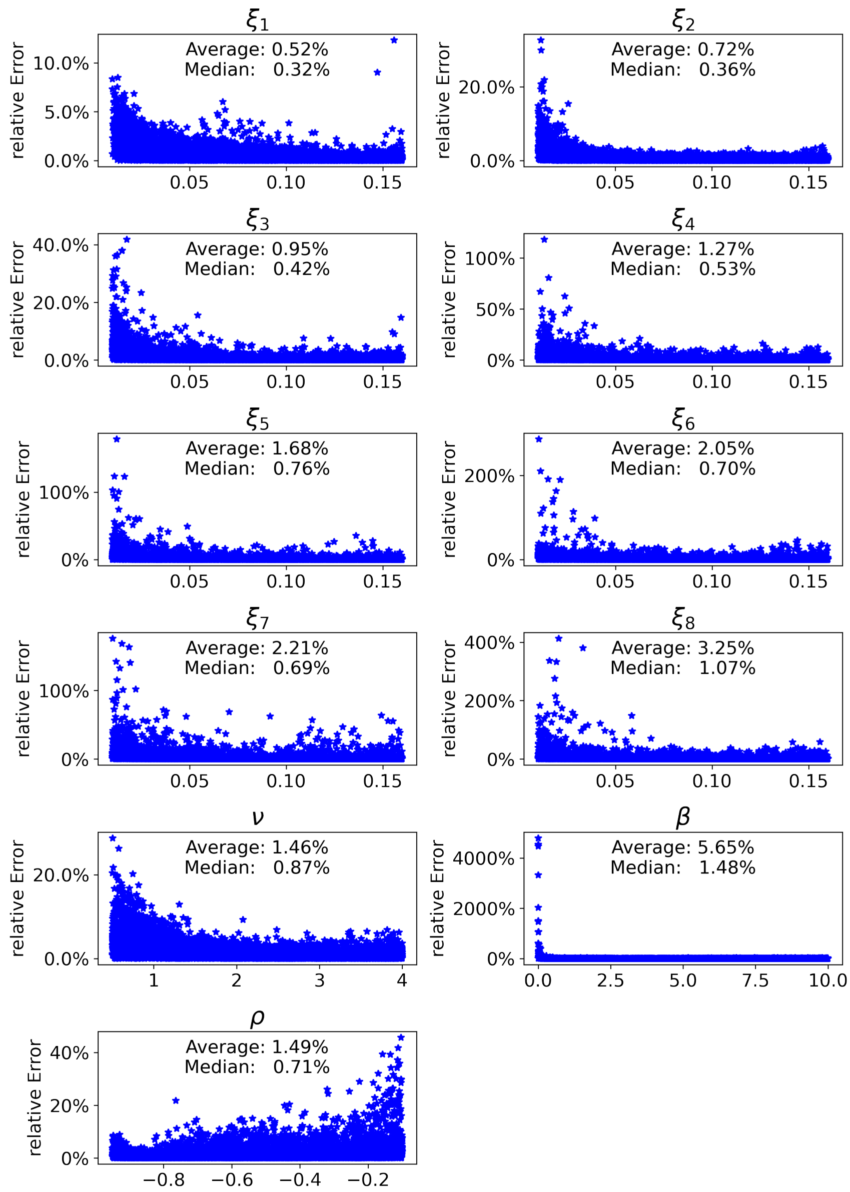


Figura A.16: Errore relativo dei diversi parametri del modello 1-Factor Bergomi

Appendice B

Codice utilizzato

Questa appendice è destinata a raccogliere le principali funzioni utilizzate nel corso del lavoro. Una versione completa è disponibile al link `https://github.com/arielNacamulli/NNCalibration_Surface`.

Nota: Poiché alcune linee di codice risultano troppo ampie per le dimensioni della pagina, sono riportate su più righe

B.1 Creazione del dataset

B.1.1 Pricing Carr-Madan

Di seguito la funzione di pricing utilizzata e le sotto-funzioni associate per il metodo di Carr-Madan.

CallPrices_CM

La funzione riceve in input i parametri descritti più avanti, restituendo come output il prezzo corrispondente.

```
import numpy as np

def CallPrices_CM(phiY, r, moneyness, TTM, methodParams):

    phi = lambda y: np.exp(-1j*y*np.log(phiY(-1j)))*phiY(y)
    fun = lambda y: np.exp(1j*r*y*TTM)*(phi(y-1j)-1)/
        (1j*y*(1j*y+1));
```

```
    f = lambda y,z: fun(y)*np.exp(-1j*z*y)
    I = FFTMethod_CM(f, methodParams, moneyness)

    prices = I/(2*np.pi) + (1-np.exp(-moneyness-r*TTM)).clip(0)

    return prices
```

dove:

`phiY` = funzione caratteristica del modello scelto;

`r` = tasso di interesse;

`moneyness` = moneyness del contratto;

`TTM` = time to maturity del contratto;

`methodParams` = parametri di modello necessari alla funzione FFT.

FFTMethod_CM

Funzione di calcolo di integrali tramite il metodo FFT. L'output del modello è il valore dell'integrale così calcolato.

```
def FFTMethod_CM(f, methodParams, moneyness):

    x1=methodParams['x1']
    xN=methodParams['xN']
    dx=methodParams['dx']
    z1=methodParams['z1']
    zN=methodParams['zN']
    dz=methodParams['dz']
    M=methodParams['M']

    N=2**M;
    x=np.arange(x1,xN+dx,dx)
    z=np.arange(z1,zN+dz,dz)
    vect=np.linspace(0,N-1,N)

    fj=np.exp(-1j*z1*vect*dx)*f(x,0) # define fj
```

```
    FFT=np.fft.fft(fj)
    Integrals=np.real(dx*np.exp(-1j*x1*z)*FFT)

    I = np.interp(np.exp(-moneyness), np.exp(z), Integrals)

    return I
```

dove:

`f` = funzione integranda, dipendente dalla moneyness e dalla variabile di integrazione dell'integrale complesso;

`numericalParams` = dizionario contenente i parametri numerici necessari per il calcolo:

- `x1` = punto iniziale della griglia di integrazione;
- `xN` = punto finale della griglia di integrazione;
- `dx` = passo della griglia di integrazione;
- `z1` = punto iniziale della griglia di moneyness;
- `zN` = punto finale della griglia di moneyness;
- `dz` = passo della griglia di moneyness;
- `M` = $(N = 2^M)$ N numero di passi del metodo FFT;

`moneyness` = moneyness per cui è calcolato l'integrale.

B.1.2 Pricing Monte Carlo

La funzione riceve in input i parametri descritti più avanti, restituendo come output il prezzo Monte Carlo corrispondente. Sono riportate le funzioni di pricing rispettivamente per i modelli VGSSD e NIGSSD.

```
def priceVGSSD_MC(moneyness, TTM, sigma, nu, theta, gamma,
N_MC = int(1e7)):

    g = np.random.randn(N_MC)
    G = np.random.gamma(1/nu, nu, N_MC)

    CF_norm = (1 - theta*nu*TTM**(gamma) - 0.5*sigma**2*nu*TTM
```

```
    **(2*gamma))**(-1/nu)
    Y_t = TTM ** gamma * (theta * G + sigma * np.sqrt(G) * g)
    f_t = -np.log(CF_norm) + Y_t
    Ft = np.exp(f_t)
    payoff = (Ft - np.exp(-moneyness)).clip(0)
    prices = np.mean(payoff)

    return prices

def priceNIGSSD_MC(moneyness, TTM, sigma, nu, theta, gamma,
N_MC = int(1e7)):

    g = np.random.randn(N_MC)
    G = np.random.wald(1/nu, 1, N_MC)

    CF_norm = np.exp(-sigma * (np.sqrt(nu**2/sigma**2 + theta**2
    /sigma**4-(theta/sigma**2 + TTM**gamma)**2) - nu/sigma))
    Y_t = TTM ** gamma * (theta * G + sigma * np.sqrt(G) * g)
    f_t = -np.log(CF_norm) + Y_t
    Ft = np.exp(f_t)

    payoff = (Ft - np.exp(-moneyness)).clip(0)
    prices = np.mean(payoff)

    return prices
```

dove:

`moneyness` = moneyness del contratto;

`TTM` = time to maturity del contratto;

`sigma` = parametro σ;

`nu` = parametro ν;

`theta` = parametro θ;

`gamma` = parametro γ;

`N_MC` = numero di simulazioni Monte Carlo.

B.1.3 Analytic

Per ogni modello è implementa una funzione che valuta l'analiticità della combinazione di parametri assegnata per il modello di pricing scelto. La funzione restituisce in output un booleano, vero se la combinazione è analitica, falso altrimenti.

```
def AnalyticVG(sigma,nu,theta,gamma,T):
    return (1 - theta*nu*T[0]**(gamma) - 0.5*sigma**2*
    nu*T[0]**(2*gamma)) > 0 and (1 - theta*nu*T[-1]**(gamma)
    - 0.5*sigma**2*nu*T[-1]**(2*gamma)) > 0


def AnalyticNIG(sigma,nu,theta,gamma,T):
    return (nu**2/sigma**2 + theta**2/sigma**4-(theta/sigma
    **2 + T[0]**gamma)**2) > 0 and (nu**2/sigma**2+
    theta**2/sigma**4-(theta/sigma**2 + T[-1]**gamma)**2) > 0
```

B.1.4 Pricing Complessivo

La funzione riceve in input i parametri descritti più avanti, restituendo come output il prezzo corrispondente. Sono riportate le funzioni di pricing rispettivamente per i modelli VGSSD e NIGSSD. Le funzioni richiamano, in ordine di priorità, i metodi Carr-Madan e Monte Carlo. Quando un metodo non rientra in una soglia di accettabilità viene stampato un messaggio utile per la diagnostica dell'errore. Nell'utilizzo pratico non si sono riscontrati casi in cui uno dei metodi fallisse.

```
def PriceVG(moneyness, TTM, sigma, nu, theta, gamma):

    phiY = lambda y: (1 - 1j*y*theta*nu*TTM**gamma + 0.5*sigma\
    **2*nu*y**2*TTM**(2*gamma))**(-1/nu)
    price = CallPrices_CM(phiY, 0, moneyness, TTM, methodParams)

    if price>0 and price < 1e2:
        return price
    elif abs(price)<1e-4 :
        return 0
```

```
    else:
        print("price_CM = "+str(price)+" TTM ="+str(TTM)+
        " moneyness ="+str(moneyness)+" params: ("+str(sigma)
        +", "+str(nu)+", "+str(theta)+", "+str(gamma)+")")
        price = priceVGSSD_MC(moneyness, TTM, sigma, nu, theta,\
        gamma)

    if price>0 and price < 1e2:
        return price
    else:
        print("price_MC = "+str(price)+" TTM ="+str(TTM)+
        " moneyness ="+str(moneyness)+" params: ("+str(sigma)+
        ", "+str(nu)+", "+str(theta)+", "+str(gamma)+")")
    return 0

def PriceNIG(moneyness, TTM, sigma, nu, theta, gamma):

    phiY = lambda y: np.exp(-sigma*(np.sqrt(nu**2/sigma**2
    + theta**2/sigma**4-(theta/sigma**2+1j*y*TTM**gamma)**2)
    -nu/sigma))
    price = CallPrices_CM(phiY, 0, moneyness, TTM, methodParams)

    if price>0 and price < 1e2:
        return price
    elif abs(price)<1e-4:
        return 0
    else:
        print("price_CM = "+str(price)+" TTM ="+str(TTM)+
        " moneyness ="+str(moneyness)+" params: ("+str(sigma)+
        ", "+str(nu)+", "+str(theta)+", "+str(gamma)+")")
        price = priceNIGSSD_MC(moneyness, TTM, sigma, nu, theta,\
        gamma)

    if price>0 and price < 1e2:
        return price
    else:
        print("price_MC = "+str(price)+" TTM ="+str(TTM)+
```

```
        " moneyness ="+str(moneyness)+" params: ("+str(sigma)+
        ", "+str(nu)+", "+str(theta)+", "+str(gamma)+")")
    return 0
```

dove:

moneyness = moneyness del contratto;

TTM = time to maturity del contratto;

sigma = parametro σ;

nu = parametro ν;

theta = parametro θ;

gamma = parametro γ;

B.1.5 Volatilità Implicita

Di seguito la funzione di calcolo della volatilità implicita. L'output della funzione è la volatilità implicita corrispondente.

```
def volaFinder(model,moneyness, TTM, sigma, nu, theta, gamma):
    if model == "VG":
        return impVol(PriceVG(moneyness, TTM, sigma, nu, theta,\
        gamma), np.exp(-moneyness), TTM)
    else:
        return impVol(PriceNIG(moneyness, TTM, sigma, nu, theta,\
        gamma), np.exp(-moneyness), TTM)

from py_vollib.black_scholes.implied_volatility
import implied_volatility as iv
def impVol(price,K,TTM):
    try:
        vol = iv(price,1,K,TTM,0,'c')
    except:
        vol = 0
    return vol
```

dove:

`model` = string corrisponente al modello scelto;

`moneyness` = moneyness del contratto;

`TTM` = time to maturity del contratto;

`sigma` = parametro σ;

`nu` = parametro ν;

`theta` = parametro θ;

`gamma` = parametro γ;

`price` = prezzo dell'opzione call;

`K` = strike del contratto.

B.1.6 Script di generazione

Lo script permette di costruire il dataset, eliminando gli elementi per cui la combinazione di parametri non è analitica ed infine salvare il dataset così costruito. Lo script in esempio è relativo al modello VGSSD, un codice analogo è stato costruito anche per il modello NIGSSD. Nello script considerato il risultato finale è la creazione del dataset salvato nel file `VGSSDTrainSet.txt.gz`.

```
import gzip

num_scenarios = param_steps**4

idx = 0

idxes = [0]*num_scenarios; idx_time=0
training_vec = np.zeros((num_scenarios,maturities_dim*
strikes_dim+4))

for sigma in param_vec[0]:
    for nu in param_vec[1]:
        for theta in param_vec[2]:
            for gamma in param_vec[3]:
```

```
                idx_time+=1
                disp=str(idx_time)+"/"+str(num_scenarios)
                print(disp,end="\r")

                if AnalyticVG(sigma,nu,theta,gamma,maturities):
                    temp_vec = np.array([sigma,nu,theta,gamma] +
                    [volaFinder("VG",moneyness[k], maturities[t],
                    sigma, nu, theta, gamma)
                    for t in range(maturities_dim)
                    for k in range(strikes_dim)])

                    training_vec[idx]=temp_vec
                    idxes[idx_time-1]=1
                    idx+=1

dat=[]

def keepTheValue(el):
    for i in el:
        if i == 0:
            return False
    return True

for i in range(len(training_vec)):
    if(keepTheValue(training_vec[i])):
        dat.append(training_vec[i])

f = gzip.GzipFile("VGSSDTrainSet.txt.gz", "w")
np.save(file=f, arr=dat)
f.close()
```

dove:

`param_steps` = numero di suddivisioni dell'intervallo di definizione dei parametri;

`moneyness` = lista delle possibili moneyness;

`maturities` = lista dei possibili time to maturity;

`strikes_dim` = dimensione del vettore moneyness;

`maturities_dim` = dimensione del vettore TTM;

`param_vec` = lista di liste, per ogni parametro sono elencati i diversi valori considerati;

B.2 Implementazione del processo di calibrazione in due passi

Di seguito è riportato il codice relativo al processo in due passi.

B.2.1 Passo 1: rete neurale

Di seguito è presentata la funzione per la costruzione della rete neurale (è riportato l'esempio di una rete a 3 hidden layer).

```
import numpy as np
import gzip
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler

def GeneralizedNNCalibration(model_name,n_parameters,strikes,
maturities,random_state=42,plot_history=False,verbose=-1):

    f = gzip.GzipFile('Dataset/' + model_name +
    'TrainSet.txt.gz',"r")
    dat = np.load(f)
    f.close()
    xx = dat[:, :n_parameters]
    yy = dat[:, n_parameters:]

    from sklearn.model_selection import train_test_split
    X_train, X_test, y_train, y_test = train_test_split(
    yy, xx, test_size=0.15, random_state=42)

    ub = np.max(xx,axis=0)
```

```
lb = np.min(xx,axis=0)

scale2 = StandardScaler()
scale2.fit(X_train)

[x_train_transform, x_test_transform] =
[scale2.transform (X_train), scale2.transform (X_test)]

y_train_transform = np.array([myscale(y,ub,lb)
for y in y_train])
y_test_transform = np.array([myscale(y,ub,lb)
for y in y_test])

# %% Construct the NN

from keras.layers import Dense
from keras.models import Sequential
from keras.callbacks import EarlyStopping

model = Sequential()

model.add(Dense(32,activation = 'elu',input_shape=(n_params,)))
model.add(Dense(32,activation = 'elu'))
model.add(Dense(32,activation = 'elu'))

model.add(Dense(88,activation = 'linear'))

try:
    model.load_weights('ModelWeights/' + model_name +
    'NNWeights.h5')

except:
    model.compile(loss = root_mean_squared_error,
    optimizer = "adam", metrics = ['acc'])

    earlystop = EarlyStopping(monitor='val_loss',
    mode='min', patience=150, restore_best_weights=True)
```

```
        model_history = model.fit(y_train_transform,
        x_train_transform, batch_size=128,
        validation_split=0.1, epochs = 1000, verbose=verbose,
        shuffle=1,callbacks=[earlystop])

        if plot_history:
            plt.figure(1,figsize=(14,4))
            fig, axs = plt.subplots(1, 2)
            axs[0].plot(model_history.history['loss'])
            axs[0].set_title('MSE')
            axs[1].plot(model_history.history['val_loss'],
            'tab:orange')
            axs[1].set_title('validation MSE')

            for ax in axs.flat:
                ax.set(xlabel='Epochs')

            for ax in axs.flat:
                ax.label_outer()
            plt.savefig('CreatedImages/'+model_name+
            'PerformanceHistory.png', dpi=300)

        model.save_weights('ModelWeights/' + model_name +
        'NNWeights.h5')

    NNParameters=[]
    for i in range(len(model.layers)):
        NNParameters.append(model.layers[i].get_weights())

    return (model,NNParameters)
```

in cui è usata la funzione ausiliaria `myscale` vista anche in Horvath, Muguruza e Tomas [30].

```
def myscale(x,ub,lb):
    res = np.zeros (len(ub))
    for i in range (len(ub)):
```

```
        res[i] = (x[i] - (ub[i] + lb[i]) * 0.5) * 2 /
        (ub[i] - lb[i])

    return res
```

nelle quali:

`model_name` = stringa con il nome di uno dei modelli disponibili;

`n_parameters` = numero di parametri del modello scelto;

`strikes` = lista dei possibili strike;

`maturities` = lista dei possibili time to maturity;

`random_state` = valore per riprodurre le simulazioni pseudo-randomiche;

`plot_history` = booleano per definire se rappresentare l'andamento della loss nel corso delle diverse epoch;

`verbose` = imposta il parametro verbose del metodo `fit`;

`x` = nella funzione `myscale`, vettore dei parametri da riscalare;

`ub` = nella funzione `myscale`, vettore dei limiti superiori dei parametri da riscalare;

`lb` = nella funzione `myscale`, vettore dei limiti inferiori dei parametri da riscalare.

B.2.2 Passo 2: calibrazione

Sono qui presentate le funzioni ausiliarie utili al passo di calibrazione, similmente a come utilizzate in Horvath, Muguruza e Tomas [30].

NumPy Neural Network

Implementazione della rete neurale tramite la libreria NumPy.

```
NumLayers=3 # o 4, a seconda delle necessità

def elu(x):
    #Careful function ovewrites x
    ind=(x<0)
    x[ind]=np.exp(x[ind])-1
    return x

def eluPrime(y):
    # we make a deep copy of input x
    x=np.copy(y)
    ind=(x<0)
    x[ind]=np.exp(x[ind])
    x[~ind]=1
    return x

def NeuralNetwork(x):
    input1=x
    for i in range(NumLayers):
        input1=input1@NNParameters[i][0]+
        NNParameters[i][1]
        #Elu activation
        input1=elu(input1)
    #The output layer is linear
    i+=1

    return input1@NNParameters[i][0]+NNParameters[i][1]

def NeuralNetworkGradient(x):
    input1=x

    grad=np.eye(len(x))
    #Propagate the gradient via chain rule
    for i in range(NumLayers):
        input1=input1@NNParameters[i][0]+
        NNParameters[i][1]
        grad=grad@NNParameters[i][0]
```

```
        #Elu activation
        grad*=eluPrime(input1)
        input1=elu(input1)

    grad= grad@NNParameters[i+1][0]
    return grad
```

in cui `num_params` e `NNParameters`, rappresentanti rispettivamente il numero di parametri del modello scelto e i pesi calibrati della rete neurale, sono definiti esternamente. In queste funzioni `x` e `y` rappresentano, per le prime due funzioni, la variabile su cui applicare la funzione di attivazione. Per le restanti due funzioni `x` contiene i parametri di modello passare alla rete neurale per effettuare un forward pass.

Funzione di costo

Funzioni di costo e relativo gradiente, utili per il metodo Levenberg - Marquardt.

```
def CostFuncLS(x,sample_ind):
    return (NeuralNetwork(x)-x_test_transform[sample_ind])
def JacobianLS(x,sample_ind):
    return NeuralNetworkGradient(x).T
```

dove `x` rappresenta la soluzione con cui è calcolata la superficie di volatilità da confrontare con la superficie di volatilità da calibrare. `sample_ind` rappresenta l'indice, relativamente a `x_test_transform`, che individua la superficie di volatilità da calibrare. `x_test_transform` è definito esternamente e rappresenta, in questo caso, il vettore contenente le superfici di volatilità del campione test. La funzione può essere resa più generica proprio modificando il modo in cui la volatilità da calibrare viene passata.
Si può quindi definire lo script che permette di concludere il processo di calibrazione.

Calibrazione

```
import scipy
init=np.zeros(num_params)
I=scipy.optimize.least_squares(CostFuncLS,init,
JacobianLS,args=(i,),gtol=1E-10)
```

dove la variabile `num_params` è precedentemente definita e individua il numero di parametri del modello scelto.

Appendice C

Composizione del Dataset

In questo capitolo è descritto il dataset utilizzato, suddiviso per i diversi modelli.

C.1 rBergomi

Il dataset per il modello rBergomi con forward variance costante è contenuto nel file "rBergomiTrainSet.txt.gz". Tale file individua una matrice di 40,000 righe e 92 colonne. Le prime 4 colonne rappresentano i diversi dei parametri del modello. In particolare, i parametri sono, nell'ordine, ξ_0, ν, ρ e H. Le restanti 88 colonne rappresentano le volatilità corrispondenti relative ai valori di strike `[0.5, 0.6, 0.7, 0.8, 0.9, 1.0, 1.1, 1.2, 1.3, 1.4, 1.5]` e di maturity `[0.1, 0.3, 0.6, 0.9, 1.2, 1.5, 1.8, 2.0]`. Sono di seguito riportate le prime righe del dataset.

```
         0        1         2         3         4         5         6  \
0  0.098927  3.37706 -0.829254  0.427445  0.714599  0.615925  0.528064
1  0.103535  1.53753 -0.352904  0.051939  0.616537  0.537771  0.466459
2  0.050898  1.91066 -0.790343  0.252989  0.582935  0.489448  0.414863

         7         8         9  ...        82        83        84        85  \
0  0.443762  0.361911  0.283117  ...  0.270653  0.229636  0.191644  0.155748
1  0.401086  0.343417  0.301457  ...  0.340072  0.324531  0.312012  0.302151
2  0.346190  0.277154  0.208671  ...  0.278279  0.243750  0.211964  0.182199

        86        87        88        89        90        91
0  0.121722  0.092911  0.082011  0.086221  0.094085  0.102366
1  0.294688  0.289376  0.285934  0.284043  0.283379  0.283648
2  0.154309  0.130050  0.115401  0.111916  0.114064  0.118330
```

C.2 1-Factor Bergomi

Il dataset per il modello 1-Factor Bergomi con forward variance costante è contenuto nel file "Bergomi1FactorTrainSet.txt.gz". Tale file individua una matrice di 40,000 righe e 92 colonne. Le prime 4 colonne rappresentano i diversi dei parametri del modello. In particolare, i parametri sono, nell'ordine, ξ_0, ν, β e ρ. Le restanti 88 colonne rappresentano le volatilità corrispondenti relative ai valori di strike `[0.5, 0.6, 0.7, 0.8, 0.9, 1.0, 1.1, 1.2, 1.3, 1.4, 1.5]` e di maturity `[0.1, 0.3, 0.6, 0.9, 1.2, 1.5, 1.8, 2.0]`. Sono di seguito riportate le prime righe del dataset.

```
          0         1        2         3         4         5         6  \
0  0.057580  2.536480  3.38008 -0.097682  0.411719  0.355430  0.308328
1  0.142071  0.569957  3.59444 -0.444563  0.413882  0.400267  0.389794
2  0.092923  0.532748  2.64730 -0.133424  0.321594  0.313281  0.307493

          7         8         9  ...        82        83        84        85  \
0  0.271002  0.243832  0.230615  ...  0.253039  0.244335  0.238143  0.234063
1  0.381427  0.374517  0.368717  ...  0.381442  0.379007  0.376932  0.375131
2  0.303376  0.300494  0.298560  ...  0.306913  0.305714  0.304779  0.304036

         86        87        88        89        90        91
0  0.231741  0.230834  0.231019  0.232012  0.233581  0.235548
1  0.373543  0.372127  0.370852  0.369694  0.368635  0.367661
2  0.303437  0.302950  0.302552  0.302225  0.301956  0.301735
```

C.3 Heston

Il dataset per il modello Heston è contenuto nel file "HestonTrainSet.txt.gz". Tale file individua una matrice di 12,000 righe e 93 colonne. Le prime 5 colonne rappresentano i diversi dei parametri del modello. In particolare, i parametri sono, nell'ordine, ξ_0, ρ, σ, θ e κ. Le restanti 88 colonne rappresentano le volatilità corrispondenti relative ai valori di strike `[0.5, 0.6, 0.7, 0.8, 0.9, 1.0, 1.1, 1.2, 1.3, 1.4, 1.5]` e di maturity `[0.1, 0.3, 0.6, 0.9, 1.2, 1.5, 1.8, 2.0]`. Sono di seguito riportate le prime righe del dataset.

```
          0         1         2         3         4         5         6  \
0  0.016840 -0.124106  0.876229  0.164721  9.580288  0.351275  0.321272
1  0.015854 -0.266165  0.140586  0.165832  7.190494  0.303108  0.235170
2  0.011428 -0.337197  0.864070  0.052406  8.424803  0.334141  0.247264

          7         8         9  ...        83        84        85        86  \
0  0.294827  0.272889  0.258973  ...  0.391454  0.391977  0.392578  0.393221
```

```
1  0.235567  0.237021  0.239069  ...  0.392158  0.392608  0.393003  0.393357
2  0.215370  0.183602  0.152755  ...  0.207557  0.209541  0.212320  0.215543

         87        88        89        90        91        92
0  0.393885  0.394556  0.395226  0.395891  0.396546  0.397191
1  0.393676  0.393969  0.394238  0.394487  0.394720  0.394938
2  0.218976  0.222469  0.225933  0.229315  0.232585  0.235730
```

C.4 VGSSD

Il dataset per il modello VGSSD è contenuto nel file "VGSSDTrainSet.txt .gz". Tale file individua una matrice di 103,537 righe e 92 colonne. Le prime 4 colonne rappresentano i diversi dei parametri del modello. In particolare, i parametri sono, nell'ordine, σ, ν, θ e γ. Le restanti 88 colonne rappresentano le volatilità corrispondenti relative ai valori di strike `[0.5, 0.6, 0.7, 0.8, 0.9, 1.0, 1.1, 1.2, 1.3, 1.4, 1.5]` e di maturity `[0.1, 0.3, 0.6, 0.9, 1.2, 1.5, 1.8, 2.0]`. Sono di seguito riportate le prime righe del dataset.

```
    0    1    2         3         4         5         6         7         8  \
0  0.3  0.2 -2.0  0.300000  1.616226  1.548826  1.488570  1.433508  1.382345
1  0.3  0.2 -2.0  0.317647  1.566500  1.499571  1.439655  1.384824  1.333795
2  0.3  0.2 -2.0  0.335294  1.518354  1.451905  1.392337  1.337744  1.286854

         9  ...        82        83        84        85        86        87  \
0  1.332401  ...  0.700970  0.686257  0.673141  0.661261  0.649938  0.640280
1  1.283897  ...  0.707511  0.692787  0.679664  0.667781  0.656457  0.646801
2  1.237002  ...  0.714105  0.699369  0.686239  0.674353  0.663027  0.653374

         88        89        90        91
0  0.630863  0.622013  0.613649  0.605705
1  0.637388  0.628543  0.620187  0.612252
2  0.643965  0.635126  0.626777  0.618851
```

C.5 NIGSSD

Il dataset per il modello NIGSSD è contenuto nel file "NIGSSDTrainSet.txt .gz". Tale file individua una matrice di 84,196 righe e 92 colonne. Le prime 4 colonne rappresentano i diversi dei parametri del modello. In particolare, i parametri sono, nell'ordine, σ, ν, θ e γ. Le restanti 88 colonne rappresentano le volatilità corrispondenti relative ai valori di strike `[0.5, 0.6, 0.7, 0.8, 0.9, 1.0, 1.1, 1.2, 1.3, 1.4, 1.5]` e di maturity `[0.1, 0.3, 0.6,`

`0.9, 1.2, 1.5, 1.8, 2.0]`. Sono di seguito riportate le prime righe del dataset.

```
    0    1     2         3         4         5         6         7         8  \
0  0.1  0.2 -11.0  0.300000  7.331715  7.155124  7.002559  6.867821  6.746863
1  0.1  0.2 -11.0  0.317647  7.246174  7.068467  6.914880  6.779190  6.657339
2  0.1  0.2 -11.0  0.335294  7.161502  6.982681  6.828072  6.691431  6.568684

         9  ...        82        83        84        85        86        87  \
0  6.633901  ...  2.081954  2.052857  2.027324  2.004540  1.982912  1.965132
1  6.543573  ...  2.089270  2.060240  2.034768  2.012039  1.990454  1.972730
2  6.454109  ...  2.096607  2.067644  2.042233  2.019559  1.998018  1.980350

         88        89        90        91
0  1.947804  1.931729  1.916727  1.902657
1  1.955448  1.939415  1.924454  1.910422
2  1.963112  1.947122  1.932202  1.918208
```

C.6 rBergomi Piecewise Constant

Il dataset per il modello rBergomi con forward variance costante a tratti è contenuto nel file “rBergomiTermStructureTrainSet.txt.gz”. Tale file individua una matrice di 80,000 righe e 99 colonne. Le prime 11 colonne rappresentano i diversi dei parametri del modello. In particolare, i primi 8 parametri $[\xi_1, \xi_2, \xi_3, \xi_4, \xi_5, \xi_6, \xi_7, \xi_8]$ sono relativi alla curva di forward variance costante a tratti, mentre i restanti parametri sono, nell'ordine, ν, ρ e H. Le restanti 88 colonne rappresentano le volatilità corrispondenti relative ai valori di strike `[0.5, 0.6, 0.7, 0.8, 0.9, 1.0, 1.1, 1.2, 1.3, 1.4, 1.5]` e di maturity `[0.1, 0.3, 0.6, 0.9, 1.2, 1.5, 1.8, 2.0]`. Sono di seguito riportate le prime righe del dataset.

```
         0         1         2         3         4         5         6  \
0  0.098927  0.136640  0.138692  0.137088  0.103535  0.067657  0.054630
1  0.081997  0.068918  0.135412  0.060609  0.107226  0.065236  0.153573
2  0.088072  0.111832  0.118095  0.097303  0.090606  0.123792  0.025886

         7         8         9  ...        89        90        91        92  \
0  0.018507  1.090630 -0.477665  ...  0.336030  0.315785  0.298846  0.284788
1  0.031053  3.480350 -0.473608  ...  0.262625  0.222439  0.185964  0.153264
2  0.081040  0.745329 -0.736918  ...  0.334085  0.319215  0.306216  0.294707

         93        94        95        96        97        98
0  0.273362  0.264395  0.257713  0.253086  0.250223  0.248802
1  0.127649  0.118807  0.126766  0.140106  0.154004  0.167282
2  0.284433  0.275211  0.266914  0.259448  0.252741  0.246736
```

C.7 1-Factor Bergomi Piecewise Constant

Il dataset per il modello 1-Factor Bergomi con forward variance costante a tratti è contenuto nel file "1FactorTermStructureTrainSet.txt.gz". Tale file individua una matrice di 80,000 righe e 99 colonne. Le prime 11 colonne rappresentano i diversi dei parametri del modello. In particolare, i primi 8 parametri $[\xi_1, \xi_2, \xi_3, \xi_4, \xi_5, \xi_6, \xi_7, \xi_8]$ sono relativi alla curva di forward variance costante a tratti, mentre i restanti parametri sono, nell'ordine, ν, β e ρ. Le restanti 88 colonne rappresentano le volatilità corrispondenti relative ai valori di strike `[0.5, 0.6, 0.7, 0.8, 0.9, 1.0, 1.1, 1.2, 1.3, 1.4, 1.5]` e di maturity `[0.1, 0.3, 0.6, 0.9, 1.2, 1.5, 1.8, 2.0]`. Sono di seguito riportate le prime righe del dataset.

```
          0         1         2         3         4         5         6  \
0  0.025886  0.081040  0.037950  0.120538  0.042483  0.030283  0.058621
1  0.101960  0.145352  0.024892  0.155471  0.107971  0.035636  0.063723
2  0.153842  0.107919  0.105259  0.159295  0.097278  0.072155  0.081205

          7        8         9  ...        89        90        91        92  \
0  0.032451  2.53707  5.373730  ...  0.267971  0.255481  0.244581  0.234916
1  0.122603  1.85271  9.025980  ...  0.310215  0.306274  0.302935  0.300053
2  0.103527  1.63767  0.384254  ...  0.352696  0.325622  0.302005  0.281400

         93        94        95        96        97        98
0  0.226252  0.218435  0.211362  0.204963  0.199186  0.193992
1  0.297532  0.295301  0.293310  0.291519  0.289898  0.288422
2  0.263656  0.248811  0.236972  0.228176  0.222261  0.218850
```

Appendice D

Rete neurale per l'individuazione del modello a minimo errore di calibrazione

In questa sezione è descritta la rete neurale utilizzata per risolvere il problema di classificazione, utile per individuare il modello più adatto per una specifica superficie di volatilità.

D.1 Struttura della rete neurale

La rete neurale utilizzata è strutturata come segue.

- **Layer Convolutional** a 32 filtri con kernel di misura 3×3;
- **Layer Max Pooling** di misura 2×2;
- **Layer Flatten**;
- **Layer Dense** a 32 nodi, con funzione di attivazione *relu*;
- **Layer Dense** a 32 nodi, con funzione di attivazione *relu*;
- **Layer Dense** a 5 nodi, con funzione di attivazione *softmax*;

Sono di seguito brevemente introdotti alcuni elementi utilizzati. Per una visione più completa si rimanda a Goodfellow, Bengio e Courville [23].

- **Convolutional Layer**, layer per processare i dati conservando le loro caratteristiche spaziali;
- **Max pooling Layer**, layer che restituisce una matrice contenente i valori massimi delle sottomatrici, della dimensione indicata, della matrice data in input;
- **Flatten**, layer che trasforma in vettore la matrice ricevuta in input;
- **relu**, funzione di attivazione definita come $f(x) = x^+$;
- **softmax**, funzione di attivazione definita come $f(x)_i = \frac{e^{x_i}}{\sum_{j=1}^{K} e^{x_j}}$ per ogni i, con K numero di nodi.

Tale rete neurale è sviluppata utilizzando la libreria Keras, che mette a disposizione gli elementi appena descritti.