UNIVERSITÀ DEGLI STUDI DI MODENA E REGGIO EMILIA

Corso di Laurea Magistrale in Ingegneria Informatica
Dipartimento di Ingegneria "Enzo Ferrari"

Reti generative per la modellazione e stima di serie temporali in ambito fintech

Relatore:
Prof. Simone Calderara

Candidato:
Matteo Casolari

Tutore Aziendale
Axyon AI SRL
Ing. Daniele Grassi

Anno Accademico 2017-2018

Abstract

Dalla loro introduzione nel 2014, si è osservato un crescente interesse nelle applicazioni del modello di Generative Adversarial Network (GAN) in un gran numero di task di computer vision, a partire dalla generazione e recupero di immagini fino ad arrivare a problemi text-to-image. Tuttavia, è stato fatto relativamente poco lavoro nel campo della generazione di serie temporali con le GAN, in particolare nell'ambito di dati finanziari.

In questa tesi, si propone un modello di GAN che genera serie temporali finanziarie realistiche. Il particolare scopo è quello di generare andamenti di prezzi azionari, per simulare un gran numero di scenari economici in condizioni reali o controllate.

La novità del modello proposto, rispetto al modello originale delle GAN, consiste nell'architettura: il generator è una rete sequence-to-sequence, allenata a generare la parte finale di una serie temporale azionaria condizionatamente alla parte iniziale fornita come input. Il discriminator, invece, è una rete convoluzionale, allenata per determinare se la serie univariata completa presentata in input sia reale o sia generata dal generator.

Indice

Elenco delle figure

Elenco delle tabelle

Capitolo 1

Introduzione

Il mercato azionario è una rete libera di transazioni che consiste nell'aggregazione di compratori e venditori di azioni. Gioca una parte di grande importanza nella crescita economica di ogni paese e per questo motivo sia i governi, che le industrie e addirittura le banche centrali controllano attentamente gli avvenimenti che lo riguardano. La funzione primaria del mercato azionario è quella di fornire capitali d'investimento per aziende, alle quale viene quindi offerta una valida alternativa di finanziamento ai prestiti bancari per espansioni e nuove attività commerciali. Un mercato azionario forte e in crescita è indice di un'economia solida e prosperosa. Oltre a fornire strumenti finanziari alle aziende, il mercato azionario funge anche da piattaforma comune per permettere agli investitori di far fruttare i propri risparmi o quelli dei propri clienti. Nonostante i mercati azionari esistano da secoli, solo nell'ultimo periodo si è passato da metodi di investimento "manuali", costituiti dalla presenza fisiche di alcune persone che utilizzano le proprie conoscenze per decidere come capitalizzare le proprie risorse, a metodi "automatici", dove si impiegano algoritmi e grandi quantità di dati per individuare quali siano le azioni su cui investire e quanto investire. Con il crescente sviluppo dei campi del *machine learning* e *deep learning*, la naturale tendenza è quello di utilizzare queste nuove tecnologie per estrarre informazioni dai dati passati e prevedere il prezzo di indici azionari. Il ruolo dell'uomo quindi viene limitato al solo livello metodologico, lasciando il compito di operare scelte strategiche educate ad algoritmi "intelligenti". L'utilizzo di reti neurali ha preso sempre più piede grazie alla loro flessibilità e capacità di analizzare una gran quantità di informazioni eterogenee.

Le *Generative Adversarial Network* sono una delle ultime innovazioni nel campo del *deep learning*, e sono destinate a prendersi sempre più parte della scena. Secondo lo stesso Yann LeCun, uno dei principali ricercatori nel campo nonché il primo ad aver realizzato reti convoluzionali come le conosciamo oggi, l'idea di utilizzare due reti che competono è assolutamente geniale: "Generative Adversarial Networks is the most interesting idea in the last ten years in machine learning". Dati i precedenti successi del *deep learning* applicato all'analisi di serie finanziarie e data l'innovazione epocale apportata dall'invenzione delle GAN allo stesso campo, si ritiene di grande interesse lo studio e l'applicazione di queste al contesto di *stocks analysis*.

1.1 Scopo della tesi

Lo scopo principale della tesi è quello di ricerca. Nel corso della tesi quindi si esplorerà la possibilità di utilizzare reti con una struttura ispirata da quella delle GAN per affrontare problemi di previsione e analisi dell'andamento di serie temporali finanziarie. In parte poiché il paradigma GAN è di recente invenzione, in parte poiché questo non è stato inizialmente pensato per la generazione di trend azionari, ma piuttosto di immagini, nel campo finanziario non si è ancora approfondito nella pratica quanto queste reti possano aiutare la formulazione di strategie. Axyon AI, l'azienda presso la quale è stato svolto il progetto legato alla scrittura della tesi, ha come *core idea* quella di investire molto sull'innovazione tecnologica, seguendo il principio secondo il quale solamente tramite nuove tecnologie all'avanguardia si possa avere una differenziazione rispetto ai competitor a livello nazionale e internazionale.

Le *Generative Adversarial Networks* sono formate da una rete *generator*, che ha lo scopo di creare esempi realistici, e compete con una rete *discriminator*, che cerca di separare gli esempi "reali" da quelli "sintetici", come dettagliatamente mostrato nella sezione 2.2.4. I possibili scenari di utilizzo della rete allenata con successo sui dati a disposizione sono:

- Utilizzare il *generator* per stabilire la direzionalità del titolo analizzato: creando un certo numero di scenari futuri di prezzo, si può estrarre la convinzione rispetto l'andamento del titolo del modello calcolando la media di queste, e prendere decisioni di conseguenza.

- Utilizzare il *generator* per acquisire proprietà statistiche sul titolo analizzato: invece che utilizzare solo la media per stabilire se un titolo nei prossimi giorni cresca o cali, è possibile trarre ulteriori informazioni statistiche utili agli analisti, come la volatilità prevista o gli intervalli di confidenza della previsione.

- Utilizzare il *discriminator* come *anomaly detector*: dato che il *discriminator* allenato è in grado di stimare la realisticità di una certa sequenza, questa caratteristica può essere utilizzata per individuare delle anomalie all'interno della stessa. In tal modo è possibile riuscire a riconoscere eventuali errori o incorrettezze nei dati di input.

- Utilizzare il *discriminator* per valutare altri algoritmi: sempre per lo stesso principio, il *discriminator* può essere utilizzato per giudicare la realisticità dei segnali prodotti da altri algoritmi o modelli, fungendo quindi da metrica a sé stante.

Tutti questi vari use cases sono utili per poter espandere e migliorare StocksAnalyst, uno dei prodotti dell'azienda ospitante, descritto brevemente in sezione 1.2.1.

1.2 Axyon AI SRL

Axyon AI è un'azienda di Modena che offre soluzioni di *deep learning* per aiutare banche e fondi di investimento ad ottimizzare processi di business, ridurre i rischi ed incrementare i ricavi utilizzando modelli predittivi tecnologicamente avanzati e ad alto tasso di successo.

Axyon AI si pone l'obiettivo di portare il *deep learning* e l'AI al cuore delle istituzioni finanziarie, e per fare questo si avvale di:

- Tecnologia: viene utilizzata una piattaforma di *deep learning* proprietaria costruita specificatamente per risolvere problemi nel campo finanziario.
- Esperienza: sono state completate con successo *Proof Of Concepts* in diversi campi, dal *credit risk* alla gestione patrimoniale, dalla previsione del tasso di rinuncia al *fraud detection*. Per ogni campo sono stati battuti tutti i *benchmark* del cliente, ogni volta.
- Partners: l'azienda nel tempo ha stretto forti legami col mondo della ricerca, avendo collaborazioni con l'"Università degli Studi di Modena e Reggio Emilia", e dell'industria, essendo sostenuta direttamente da ING e avvalendosi di importanti partnership con Nvidia e IBM.

Al momento Axyon AI vende due prodotti completi basati su algoritmi di AI: SynFinance, per il mercato dei prestiti sindacati, e StocksAnalyst, per l'*asset allocation*. Essendo il progetto legato alla tesi orientato all'evoluzione e sviluppo del secondo prodotto, in seguito verrà analizzato più in dettaglio in cosa consiste.

1.2.1 StocksAnalyst

Axyon StocksAnalyst è un motore basato su AI, pronto per l'uso, progettato per fondi di investimento e banche, che permette di integrare le strategie degli asset manager con le predizioni generate da algoritmi proprietari di Axyon AI basati sulla tecnologia del *deep learning*, in breve tempo e con costi contenuti. In particolare, StocksAnalyst:

- Integra diverse sorgenti di dati (*data sources*), inclusi dati non strutturati (e.g. notizie) e, qualora vengano forniti, anche dati di proprietà del cliente (e.g. indici proprietari del cliente).
- Utilizza le più avanzate tecniche di AI e *Deep Learning* (DL), incluse *Fully Connected Neural Networks* (FCNNs), *Convolutional Neural Networks* (CNNs), *Recurrent Neural Networks* (RNNs) e *Genetic Algorithms* (GAs).
- Fornisce predizioni via web, API o direttamente integrandosi con i sistemi del cliente. Genera *insights* sulle predizioni che permettono di interpretare e capire le logiche dell'AI sottostanti alla generazione dei segnali, auto-apprese dagli algoritmi elaborando i dati ad essi forniti.

1.3 Organizzazione della tesi

I vari capitoli che compongono la tesi vengono mostrati di seguito, insieme a una breve descrizione degli argomenti trattati negli stessi:

- Capitolo 2: vengono introdotti i concetti, le teorie e le tecnologie a cui si farà riferimento nel corso della tesi. È suddiviso in due sezioni: nella prima viene analizzato prevalentemente l'aspetto economico e finanziario, nella seconda vengono mostrati i fondamenti del *deep learning* e le tipologie di reti utilizzate.
- Capitolo 3: viene trattata la metodologia che ha portato allo sviluppo del progetto legato alla tesi analizzando passo passo ogni step del processo: a partire dalla definizione del problema, per continuare con le analisi preliminari, la scelta degli iperparametri, fino ad arrivare alla descrizione delle problematiche affrontate.
- Capitolo 4: viene mostrata in maniera dettagliata l'architettura del modello realizzato e dei suoi componenti, insieme alle tecnologie che sono state utilizzate a supporto dello sviluppo.
- Capitolo 5: vengono riportati tutti i test e i risultati ottenuti. Inizialmente viene descritta la metodologia adottata per costruire il dataset, il processo di training e il valore di tutti gli iperparametri utilizzati, per garantire la riproducibilità dei test effettuati. Successivamente vengono descritte le baseline utilizzate e confrontate con la rete implementata mostrando dapprima risultati quantitativi e poi qualitativi.
- Capitolo 6: vengono riassunte le principali caratteristiche del lavoro prodotto, traendone le conclusioni ed esaminando gli obiettivi raggiunti. Vengono inoltre proposte alcune possibilità di ampliamento e di utilizzi alternativi del modello analizzato nel corso della tesi.

Capitolo 2

Background teorico

In questo capitolo verranno introdotti i concetti, le teorie e le tecnologie a cui si farà riferimento nel corso di tutta la tesi. La sezione (2.1) sarà focalizzata sull'aspetto economico, introducendo l'ambiente in cui è inserito il progetto. Sarà descritto il mercato azionario, gli strumenti utilizzati per analizzarlo e alcune delle teorie fondamentali che lo governano. Verranno introdotti anche i singoli indici azionari utilizzati come dati in questa tesi. Nella sezione (2.2) sarà introdotto il *deep learning*, analizzando le tipologie di reti che verranno utilizzate nei modelli testati. Le GAN, l'argomento principale della tesi, verranno introdotte nella sezione (2.2.4)

2.1 Stock Analysis

Per *stock analysis* si intende l'esaminazione e la valutazione del mercato azionario. Può assumere la forma di analisi di singoli titoli, settori o aree più ampie. È un metodo utile al supporto del processo decisionale di compravendita di azioni, in quanto aiuta gli investitori a compiere scelte informate, tentando di guadagnare un vantaggio dall'analisi dei dati di mercato attuali e passati. Viene anche chiamato *market analysis* o *equity analysis*, ed è il campo dell'economia finanziaria in cui è possibile far rientrare tutte le tecniche di *machine learning* per lo studio dell'andamento dei prezzi dei mercati finanziari nel tempo, quindi anche il progetto legato alla tesi.

In questa sezione si descriverà brevemente cosa sono i mercati azionari e com'è possibile guadagnare investendo su azioni. Saranno inoltre introdotte le tecniche di analisi e alcune delle più importanti teorie finanziarie, fino a presentare i titoli azionari che verranno analizzati.

2.1.1 Mercato azionario

Il mercato azionario è l'aggregazione di compratori e venditori di azioni, ovvero titoli rappresentativi di quote di società. Viene inteso come una rete libera di transazioni economiche piuttosto che una struttura fisica o un'entità discreta. Il principale motivo per cui le azioni esistono è quello di strumento finanziario per investimenti dell'impresa: rinunciando

a quote di società le imprese che cedono azioni ne ricevono benefici finanziari, raccogliendo capitale, e operativi, favorendo l'ingresso di nuovi soci privati nell'impresa. Gli investitori, a loro volta, acquistando azioni ottengono il diritto di votare per il consiglio di amministrazione, di percepire i dividendi nel caso in cui vengano erogati ai proprietari di azioni e, soprattutto, di rivendere le azioni in borsa.

Il mercato azionario si divide in mercato primario e in mercato secondario. Sul mercato primario sono collocate le azioni di nuova emissione, sul mercato secondario, invece, sono negoziati i titoli già in circolazione. La suddivisione è doverosa in quanto l'impresa di riferimento riceve un compenso solo per le azioni acquistate nel mercato primario, tutti gli scambi che avvengono sul mercato secondario sono tra investitori e non portano alcun capitale all'impresa. Se le azioni della società in questione risultano redditizie per gli investitori, la richiesta di queste subirà verosimilmente un aumento, facendone salire di conseguenza anche il prezzo. Questo processo porta a un guadagno sia per l'investitore, che si ritrova a possedere beni che valgono più di quanto aveva pagato, sia per l'impresa, in quanto ha la possibilità di immettere nuove azioni nel mercato primario ad un prezzo più alto. Tuttavia, come per ogni tipo di investimento, c'è una inversa proporzionalità tra ritorno e rischio. Gli investimenti sul mercato azionario sono tra i più redditizi se fatti con lungimiranza, ma sono anche tra quelli a più alto coefficiente di rischio. Come è possibile che la società su cui si ha investito si espanda grandemente, è anche possibile che questa vada in crisi o addirittura fallisca, portando il valore delle azioni acquistate a valori più bassi di quelli di acquisto, o addirittura a zero.

Mercato rialzista e mercato ribassista

Due dei concetti base del mercato azionario sono il mercato rialzista (o *bull market*, mercato toro) e il mercato ribassista (o *bear market*, mercato orso). Per mercato rialzista si intende un mercato in cui il prezzo delle azioni ha una tendenza all'aumento. Questo è il mercato in cui la maggioranza degli investitori sperano, in quanto il maggior numero degli investitori azionari sono compratori, invece che venditori allo scoperto. Per mercato ribassista si intende un mercato in cui il prezzo delle azioni ha una tendenza alla diminuzione.

Nel caso in cui il mercato sia rialzista, il metodo in cui un investitore può guadagnare è abbastanza lineare: acquistando azioni della società che crede possa crescere di più in quel periodo. Per esempio, se un investitore crede che il prezzo delle azioni della società "A" crescerà rispetto a quello attuale, che è di 10€ l'una, l'investitore può decidere di comprarne 100 spendendo 1000€. Se in un futuro il prezzo dell'azione dovesse arrivare a 20€ l'una, le può rivendere ricavandoci 2000€, con un guadagno complessivo di 1000€.

Gli investitori possono riuscire a guadagnare anche nel caso in cui il mercato sia ribassista, attraverso la vendita allo scoperto. La vendita allo scoperto (o *short selling*) è un'operazione finanziaria che consiste nel chiedere in prestito azioni che l'investitore non possiede a una banca o un intermediario finanziario, per poi rivenderle istantaneamente. Se il prezzo di quelle azioni cala, l'investitore può generare del profitto ricomprando lo stesso numero di azioni vendute inizialmente e saldando il proprio debito con l'intermediario finanziario. Per esempio, se un investitore crede che il prezzo delle azioni della società "B" calerà rispetto a quello attuale, che è di 20€ l'una, l'investitore può decidere di chiedere

in prestito 100 azioni ad un intermediario e di rivenderle subito ricavandoci 2000€. Se in futuro il prezzo dell'azione dovesse calare fino a 10€ l'una, l'investitore può ricomprare 100 azioni della stessa società ed estinguere il debito con l'intermediario, spendendo 1000€. Il guadagno complessivo sarà quindi di 1000€.

Value investing e growth investing

Il *value investing* e il *growth investing* sono due diverse tecniche di investimento che differiscono nell'approccio al mercato e nella scelta delle azioni su cui investire. Nella ricerca delle azioni perfette su cui investire, le variabili fondamentali da prendere in considerazione sono:

- Rischio
- Potenziale di crescita
- Prezzo

Solitamente c'è un bilanciamento di questi fattori. Sarà quindi molto difficile per un investitore trovare delle azioni che abbiano alto potenziale di crescita, con rischio limitato e a basso prezzo. La differenziazione tra le tecniche di *value investing* e *growth investing* si basa su quali di queste variabili concentrarsi per la scelta delle azioni su cui investire.

I *value investors*, ovvero gli investitori che seguono l'approccio di *value investing*, si concentrano principalmente su azioni a limitato rischio e bassa valutazione. Tendono, pertanto, ad acquistare azioni di società che offrono introiti stabili e prevedibili, con pochi rischi, e a valutazioni che si può aspettare di vedere crescere. L'analisi, quindi, si concentra sulla valutazione di eventuali minacce per l'azienda su cui si sta investendo e sulla ricerca di azioni poco costose se rapportate al reale valore di quota di società che si sta acquisendo.

I *growth investors*, ovvero gli investitori che seguono l'approccio di *growth investing*, si concentrano primariamente sul potenziale di crescita delle azioni. Le azioni ad alto potenziale di crescita solitamente sono sovrapprezzo, in quanto tale proprietà viene considerata nella valutazione globale dell'azione. L'analisi, quindi, si concentra sulla valutazione della possibilità di effettiva crescita dell'azienda su cui si sta investendo.

In definitiva, i *growth investors* investono sperando nel cambiamento (in positivo) del valore proprie azioni, i *value investors* investono auspicando la conservazione del valore delle proprie azioni.

2.1.2 Analisi tecnica e analisi fondamentale

L'analisi tecnica e l'analisi fondamentale sono le principali scuole di pensiero nell'approccio ai mercati azionari. Entrambi i metodi sono di *stock analysis*, in quanto sono usati per studiare e prevedere possibili andamenti di prezzi di azioni, ma differiscono sostanzialmente per la tipologia di dati e per le metodologie utilizzate per portare a termine l'analisi.

L'analisi tecnica sfrutta principalmente i dati relativi all'andamento del mercato nel passato, analizzati tramite metodi grafici, statistici o algoritmici. I principi fondamentali

su cui si basa sono quello di validità dei trend e di ripetitività della storia. Per il principio di validità dei trend, nell'analisi del grafico del prezzo di un indice, nonostante sia possibile una locale fluttuazione del prezzo, è probabile che l'andamento globale sia di tipo monotono, legando di fatto l'andamento futuro con quello passato. Per il principio di ripetitività della storia il comportamento degli investitori si ripete nel tempo. Quindi, nel caso in cui si dovesse ripetere una situazione simile ad una presentatasi in passato, gli investitori si comporteranno in maniera simile, portando l'andamento del mercato ad un risultato simile, e quindi prevedibile.

L'analisi fondamentale utilizza le caratteristiche economico-finanziarie intrinseche della società in esame al fine di determinare il valore effettivo (*fair value*) di questa, e quindi anche delle relative azioni. Le variabili maggiormente considerate sono lo stato patrimoniale, il conto corrente e il rendiconto finanziario, unitamente alla valutazione dell'eventuale impatto di eventi micro e macroeconomici sulla società presa in esame. Il principio su cui si basa l'analisi fondamentale è quello della correttezza dei mercati. Per questo principio le azioni sul mercato che hanno un prezzo che è diverso dal proprio reale valore (maggiore o minore) tenderanno ad assumere il valore "corretto" in un tempo finito.

2.1.3 Teorie finanziarie

I principi fondamentali dell'analisi tecnica e fondamentale non sono tuttavia universalmente accettati. Nel corso degli anni una parte della comunità scientifica ha condotto vari studi volti a confutare le tesi a sostegno della prevedibilità del mercato, producendo teorie di assoluta importanza per il campo dell'economia finanziaria: l'ipotesi dei mercati efficienti e la teoria del *random walk*.

Ipotesi dei mercati efficienti (EMH)

Secondo l'ipotesi dei mercati efficienti di Eugene Fama [19] , nei mercati considerati efficienti i prezzi delle azioni riflettono pienamente tutte le informazioni a disposizione riguardanti la stessa. Nel caso in cui si operi in mercati di quel tipo, nessuna elaborazione di informazioni sul passato o sul presente conferisce alcun vantaggio effettivo rispetto alla semplice conoscenza del prezzo, rendendo quindi gli sforzi compiuti dall'analisi tecnica e fondamentale completamente inutili. Lo sviluppatore dell'ipotesi dei mercati efficienti introduce anche una formalizzazione per quanto riguarda il livello di efficienza del mercato:

1. Efficienza in forma debole: i prezzi delle azioni riflettono tutta l'informazione contenuta nella serie storica dei prezzi stessi. In questo caso tutte le informazioni sfruttate dall'analisi tecnica sono aggregate nel prezzo, rendendola di fatto inefficace. Tuttavia, qualche forma di analisi fondamentale ha la possibilità di portare risultati utili.

2. Efficienza in forma semi-forte: i prezzi delle azioni riflettono l'informazione contenuta nella serie storica degli stessi e anche in qualsiasi altro dato disponibile al pubblico. Questa forma implica che sia l'analisi tecnica che quella fondamentale non possano portare esiti favorevoli, a meno che non si disponga di informazioni private.

3. Efficienza in forma forte: i prezzi delle azioni riflettono l'informazione contenuta nella serie storica degli stessi e anche in qualsiasi altro dato, sia pubblico che privato. Non è possibile formulare alcun tipo di strategia con rendimento superiore a quello del mercato.

Teoria del Random Walk

La teoria della passeggiata aleatoria [20], o del *random walk*, afferma che i prezzi delle azioni variano seguendo una *random walk*, ovvero in maniera completamente casuale e senza alcuna correlazione sequenziale. Questa teoria è completamente coerente e in stretta correlazione con l'ipotesi dei mercati efficienti, e sottolinea a sua volta l'impossibilità di prevedere l'andamento dei mercati azionari. Il modello di *random walk* che più spesso viene utilizzato è in ambito finanziario è quello gaussiano. Nel *random walk gaussiano*, ogni valore di una serie temporale di valori continui $p(t)$, che nel nostro caso rappresenta il prezzo di un'azione, è legato al valore precedente tramite la relazione:

$$p(t) = p(t-1) + x(t) \tag{2.1}$$

Dove $p(t-1)$ è il valore del prezzo all'istante di tempo precedente $t-1$, e $x(t)$ è il valore assunto dalla variabile aleatoria $X \sim \mathcal{N}(\mu, \sigma)$ al tempo t. La densità di probabilità della variabile aleatoria $X \sim \mathcal{N}(\mu, \sigma)$, detta *gaussiana* o *normale*, è:

$$f(x, \mu, \sigma^2) = \frac{1}{\sigma\sqrt{2\pi}} e^{-(x-\mu)^2/2\sigma^2} \tag{2.2}$$

L'assunzione che viene spesso fatta è che $x(t)$ sia rumore bianco o *white noise*, e quindi sia campionato da una gaussiana con media 0, $X \sim \mathcal{N}(0, \sigma)$:

$$f(x, \sigma^2) = \frac{1}{\sigma\sqrt{2\pi}} e^{-x^2/2\sigma^2} \tag{2.3}$$

Secondo questa legge matematica, il prezzo di un azione all'istante di tempo t è completamente scorrelato da qualsiasi altra informazione che non sia il prezzo all'istante di tempo precedente $t-1$, e varia da questo in maniera assolutamente randomica, con la probabilità del 50% positivamente e del 50% negativamente.

Prove empiriche

L'effettiva validità dei risultati apportati dall'analisi tecnica e fondamentale è ancora un argomento di discussione. Secondo i sostenitori dell'ipotesi dei mercati efficienti, il guadagno degli investitori di maggiore successo non è dato dalla loro abilità nel prevedere trend di mercato o dall'adozione di tecniche avanzate, ma semplicemente dal caso. Non avendo alcuna reale possibilità di avere informazioni utili a supporto delle proprie scelte, ogni investitore opera assolutamente a caso e ha la stessa probabilità di prendere la decisione giusta di un altro che usa il lancio di una moneta per decidere. Gli investitori di successo sono tali solo perché, tra milioni di casi di investitori, è probabile che ve ne siano alcuni che effettuano casualmente una serie di scelte giuste d'investimento.

Non essendo possibile arrivare alla formulazione di un teorema di tipo matematico, negli anni la comunità scientifica si è spesa tanto nel supportare la validità di analisi tecnica e analisi fondamentale con svariate prove empiriche [25, 2, 21]. Tuttavia, molti degli studi empirici soffrono del fenomeno chiamato *data snooping bias* [18]. I risultati sperimentali sono ampiamente positivi, ma poco generalizzabili: cambiando la società, il tasso di cambio o l'indice da prevedere, le performance delle regole di trading o degli algoritmi sviluppati risultano non essere più così buone.

In definitiva, nonostante non vi sia una visione unitaria sulla completa efficacia dell'analisi in questione, è universalmente riconosciuto come questa possa essere di grande aiuto se usata come supporto decisionale per investitori. L'emozionalità che caratterizza l'investitore in quanto essere umano può portare a decisioni errate dovute ad irrazionale ottimismo o pessimismo. L'analisi tecninca o fondamentale, insieme alla gestione del rischio, o *risk management*, sono due strumenti molto importanti per sopprimere o quantomeno limitare l'aspetto emotivo nelle decisioni di carattere finanziario.

2.1.4 Indici azionari

Un indice azionario, o *stock index*, è una misurazione di una sezione del mercato azionario. Il suo valore viene calcolato a partire dal prezzo delle azioni appartenenti al paniere di titoli azionari rappresentato. Un paniere di titoli azionari è un insieme di strumenti finanziari opportunamente scelti per rappresentare l'effettivo andamento di una sezione di mercato. A seconda del tipo di scelta del paniere si possono avere diversi tipi di indici. Gli indici più conosciuti sono quelli nazionali, che rappresentano l'andamento del mercato azionario all'interno di una determinata nazione e sono formati dalle più importanti società della stessa. Tra questi, i più importanti sono l'americano S&P 500, il giapponese Nikkei 225 e l'inglese FTSE 100. Allargando il campo geograficamente, possiamo trovare indici regionali come il FTSE Developed Europe e il FTSE Developed Asia Pacific Index, rispettivamente per Europa e Asia Pacifica, fino ad arrivare ai mondiali MSCI World e S&P Global 100. All'interno delle singole nazioni, esistono vari indici specializzati in alcuni settori di mercato, come ad esempio gli americani Wilshere US REIT, che racchiude le principali società del settore immobiliare, e Dow Jones U.S. Technology Index, che segue esclusivamente aziende nell'industria tech, oppure il recentissimo FTSE Italian Brands, dedicato alle società italiane conosciute in tutto il mondo. Esistono, inoltre, titoli che si focalizzano su società di determinate dimensioni, oppure con una certa struttura manageriale, per aziende ad alto tasso di crescita e addirittura specifici per società che usano Linux come sistema operativo.

Su indicazione dell'azienda ospitante, gli indici azionari analizzati sono FTSE MIB, DAX 30, FTSE 100, S&P 500, in quanto più interessanti per l'integrazione con la piattaforma già esistente. Questi indici vengono analizzati nel periodo che va dal 1° gennaio 2000 al 31 aprile 2018.

FTSE MIB

Il FTSE MIB (*Financial Times Stock Exchange Milano Indice di Borsa*) è il più importante indice italiano, nonché l'indice di riferimento per la Borsa Italiana. È composto dalle 40 società italiane di maggior capitalizzazione, anche con sede all'estero.

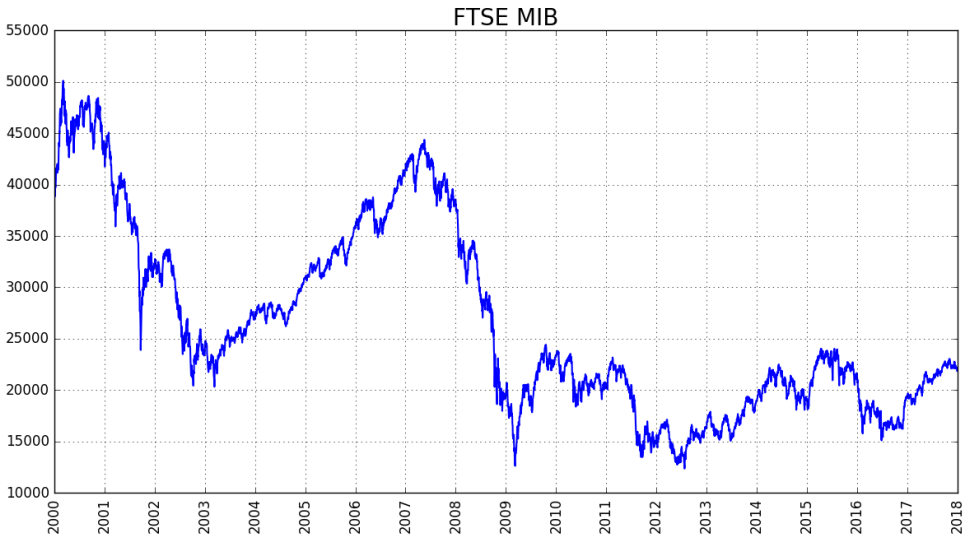


Figura 2.1: Andamento del FTSE MIB nel periodo dal 2000 al 2018.

FTSE 100

Il FTSE 100 (*Financial Times Stock Exchange 100*) è il più importante indice del Regno Unito, ed è composto dalle 100 società più capitalizzate della *London Stock Exchange*. Le sole società presenti nel FTSE 100 rappresentano circa l'81% di tutto il mercato azionario nazionale.

DAX 30

Il DAX 30 (*Deutsche AktienindeX 30*) è l'indice azionario nazionale tedesco. È costituito dalle 30 più importanti compagnie del paese, secondo la *Deutsche Börse* (Borsa Tedesca). Il DAX 30 ha due versioni, l'indice di performance e l'indice di prezzo, a seconda che i dividendi siano valutati oppure no. Nonstante l'indice di prezzo abbia un metodo di calcolo più simile a quello degli altri indici nazionali, l'indice di performance è quello più conosciuto, ed è quindi quello che verrà analizzato.

S&P 500

L'S&P 500 (*Standard & Poor's 500*) è un indice azionario americano che si basa sulla capitalizzazione delle 500 più grandi società con sede negli USA. Insieme al *Dow Jones*

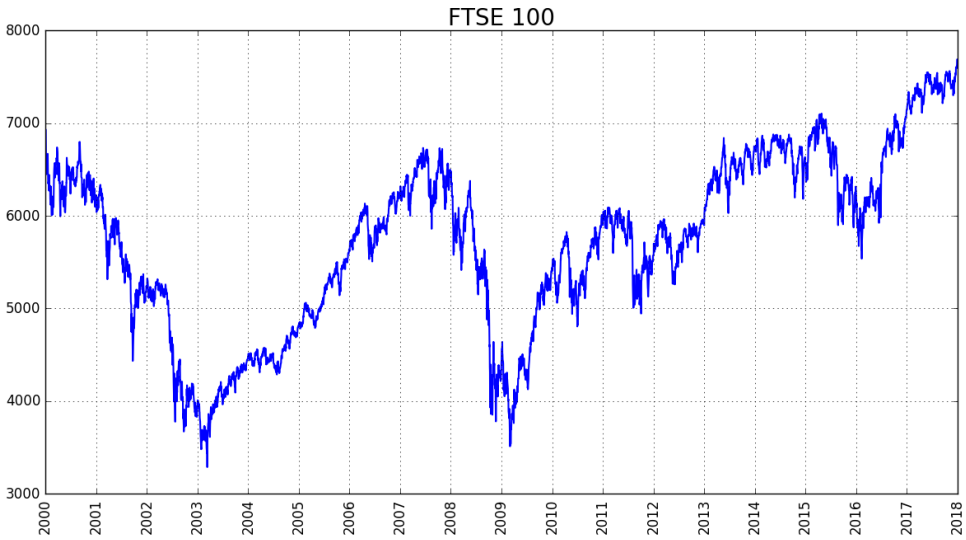


Figura 2.2: Andamento del FTSE 100 nel periodo dal 2000 al 2018.

Figura 2.3: Andamento del DAX 30 nel periodo dal 2000 al 2018.

Industrial Average, è uno dei più importanti indici nazionali, nonché uno dei più seguiti a livello globale. Quello che differenzia l'S&P 500 dal Dow Jones è il sistema con cui vengono pesati i titoli nella media: il primo è *value weighted*, ovvero ha il peso di ogni titolo proporzionale alla capitalizzazione di borsa della società, il secondo è *price weighted*, e pesa ogni titolo a seconda del prezzo. Il Down Jones, quindi, non tiene conto delle effettive

dimensioni delle società dei titoli di cui è composto, risultando meno rappresentativo del reale andamento del mercato.

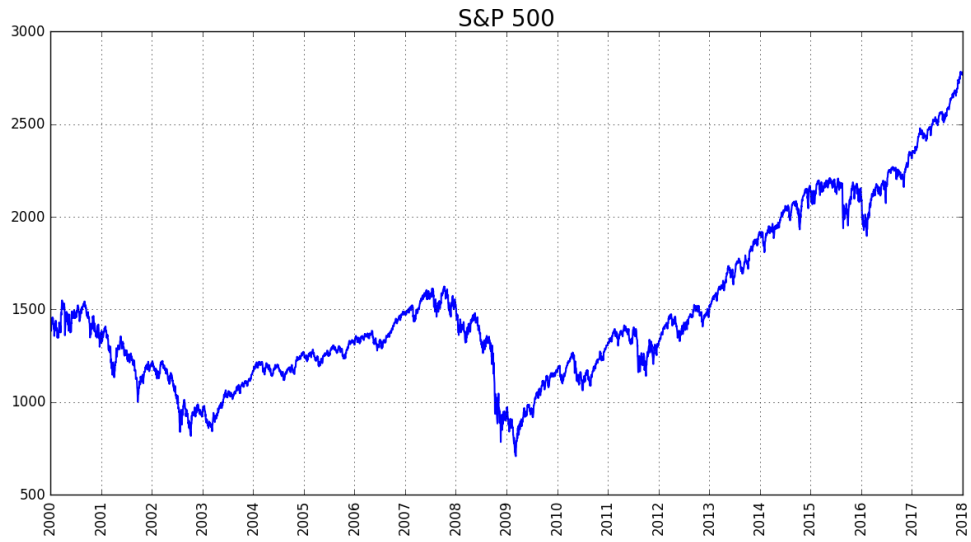


Figura 2.4: Andamento del S&P 500 nel periodo dal 2000 al 2018.

2.2 Deep learning

Il *Deep Learning* (DL) è il campo appartenente alla branca dell'Intelligenza Artificiale che utilizza algoritmi ispirati dalla struttura e dal funzionamento del cervello umano per risolvere problemi di classificazione, regressione e clustering. Per capire meglio quale sia il significato del termine, è necessario fare chiarezza sulle differenze tra quelli che vengono erroneamente considerati suoi sinonimi: Intelligenza Artificiale (AI) e *Machine Learning* (ML).

Per Intelligenza Artificiale si intende la capacità di alcune macchine di portare a termine compiti complessi ed assumere comportamenti tipici di un umano, mostrando quindi una forma di intelligenza sintetica. Il *machine learning* è il campo che studia algoritmi in grado di risolvere un problema senza essere programmati esplicitamente, ma imparando da un gran numero di esempi che gli vengono mostrati. Il ML è semplicemente uno degli approcci possibili per l'AI. Se, come algoritmi di ML, si utilizzano reti neurali (approfondite nella sezione 2.2.1) con un

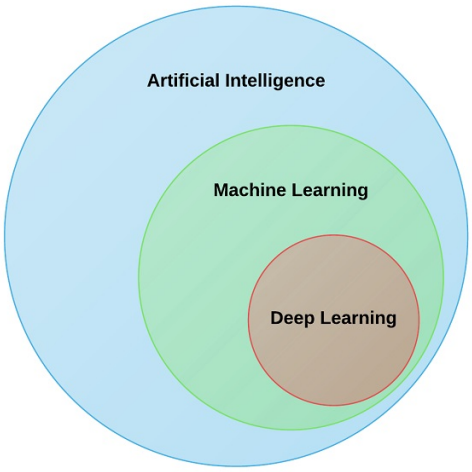


Figura 2.5: Relazione tra Intelligenza Artificiale, Machine Learning e Deep Learning

numero elevato di strati, allora si sta lavorando nel campo del *Deep Learning*.

Nonostante le basi teoriche siano state gettate negli anni '40, è solo nell'ultimo decennio che il DL ha ottenuto sempre più attenzione da parte della comunità scientifica. I modelli realizzati hanno raggiunto lo stato dell'arte nella classificazione di immagini, testi, suoni e dati strutturati, spesso raggiungendo o superando il livello umano. La ragione per cui si è assistito solo nell'ultimo periodo a una crescita, è che siano venute a verificarsi solo recentemente le condizioni adatte allo sviluppo di algoritmi di DL:

- Aumento della quantità di dati disponibili: grazie alla recente digitalizzazione della società, si ha a disposizione una quantità di dati enormemente superiore a quella del passato. Ogni attività umana eseguita su computer, cellulari e dispositivi digitali crea dati, che poi possono essere accumulati e utilizzati da questi algoritmi. La peculiarità degli algoritmi di DL è quella di riuscire a sfruttare il grande volume di dati a disposizione in maniera molto più efficiente degli algoritmi di ML classico.

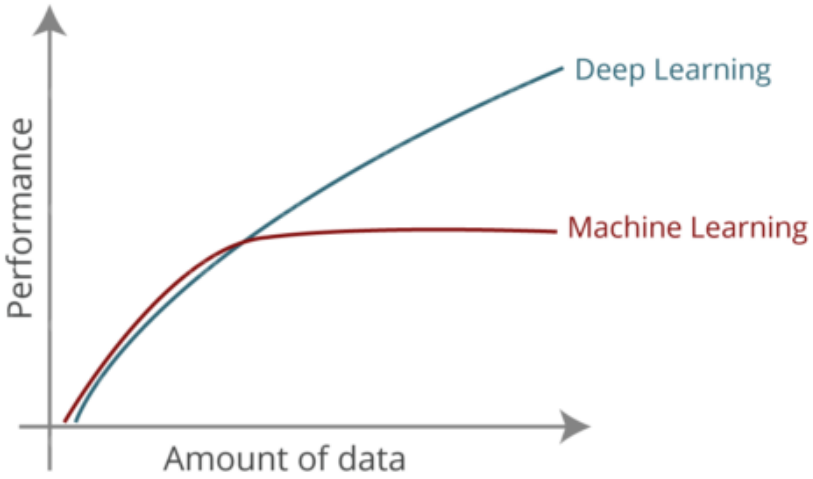


Figura 2.6: Prestazioni di Machine Learning e Deep Learning al variare della quantità di dati impiegati.

- Aumento della potenza di calcolo utilizzabile: l'avvento delle GPU ha ridotto grandemente il tempo necessario per processare i dati e utilizzarli per allenare le reti. La velocità computazionale è di vitale importanza specialmente nel processo di ottimizzazione dell'algoritmo. Con una velocità di training superiore è possibile testare più

architetture differenti, e quindi avere più probabilità di trovare quella che meglio si adatta al problema in analisi.

L'idea chiave che sta alla base del DL è quella di manipolare i dati grezzi in modo da ottenere rappresentazioni sempre più astratte di questi. Nonostante per un essere umano sia facile determinare l'identità di un oggetto a partire da una sua immagine, è molto più difficile descrivere rigorosamente i passaggi attraverso i quali è arrivato a quella conclusione. Il problema è che la funzione che mappa un insieme di pixel a l'identità di un oggetto è molto complessa. La soluzione che il DL dà a questo tipo di problemi è quella di suddividere questa funzione in una serie di sottofunzioni più semplici, attraverso le quali i vari layers del modello trasformano una rappresentazione in un'altra più astratta. Per risolvere l'esempio in questione, dati i pixel di immagine, il primo layer si occuperà di indentificare i bordi e i contorni delle figure. Dati i bordi, il secondo layer si dedicherà all'individuazione forme note, come spigoli, curve o cerchi. Dati gli output del secondo layer, il terzo layer potrà riconoscere parti di oggetto tramite l'aggregazione di forme note, come ruote di automobili, arti umani o finestre. L'ultimo layer, quindi, può determinare l'entità dell'oggetto a partire dal riconoscimento di varie parti caratteristiche. Più è alto il numero di layer, più complessa può essere la funzione che viene rappresentata.

In questa sezione si introducono le reti neurali e le tipologie architetturali utilizzate, con un approfondimento finale sulla classe di algoritmi alla quale appartiene il modello sviluppato.

2.2.1 Artificial Neural Networks

Le reti neurali artificiali (o *Artificial Neural Networks*, abbreviato ANN) sono modelli matematici ispirati alla struttura e al funzionamento del cervello animale. Sono costituite da un insieme connesso di elementi semplici chiamati percettroni, che rappresentano una versione artificiale dei neuroni biologici. Il modello del percettrone, proposto nel 1943 da Warren S. McCulloch e Walter Pitts [22] e realizzato nel 1958 da Frank Rosenblatt [30], consiste in un nodo che riceve in input un vettore di lunghezza n di segnali d'ingresso $\boldsymbol{x} = x_1, x_2, \ldots, x_n$, con $x_i \in \mathbb{R}\ \forall\ i = 1, 2, \ldots, n$ e ha un segnale in uscita $y \in \mathbb{R}$. Il singolo percettrone pesa l'input tramite un vettore di pesi $\boldsymbol{w} = w_1, w_2, \ldots, w_n$, con $w_i \in \mathbb{R}\ \forall\ i = 1, 2, \ldots, n$, vi somma un peso particolare b chiamato bias, e determina l'uscita tramite una funzione di attivazione f. L'intero funzionamento può essere schematizzato dalla formula:

$$y = f\left(\sum_{i=1}^{n} w_i x_i + b\right) \tag{2.4}$$

Nonostante la funzione di attivazione f che più si avvicina a quella del neurone biologico sia quella binaria (2.5), nella pratica si è notato come questa non favorisca il processo di apprendimento nelle reti neurali. La funzione di attivazione nettamente più utilizzata nel campo è la ReLU (*Rectified Linear Unit*)(2.6) [24], ma non è raro trovare anche Leaky ReLU (Rectified Linear Unit) (2.7) [11], sigmoidi (2.8) o TanH (Tangent Hyperbolic) (2.9)

$$f(x) = \begin{cases} 0, & \text{if } x < 0 \\ 1, & \text{if } x \geq 0 \end{cases} \tag{2.5}$$

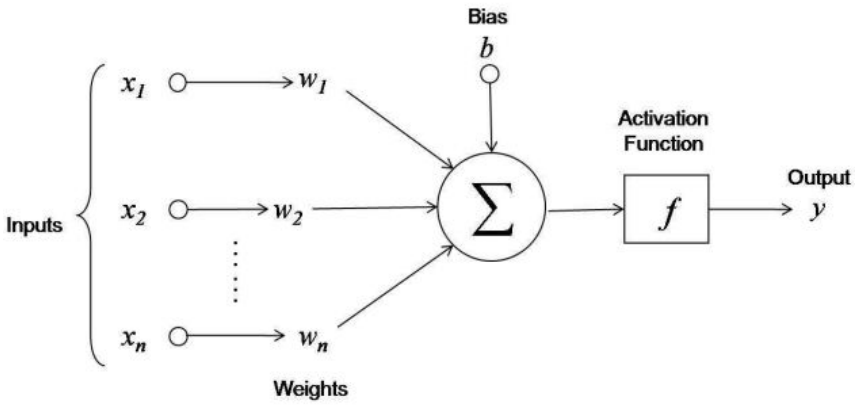


Figura 2.7: Percettrone

$$f(x) = \begin{cases} 0, & \text{if } x < 0 \\ x, & \text{if } x \geq 0 \end{cases} \tag{2.6}$$

$$f(x) = \begin{cases} \alpha x, & \text{if } x < 0 \\ x, & \text{if } x \geq 0 \end{cases} \tag{2.7}$$

$$f(x) = \sigma(x) = \frac{1}{1 + e^{-x}} \tag{2.8}$$

$$f(x) = tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \tag{2.9}$$

Le ANN sono architetture costituite da un numero variabile di percettroni, organizzati a strati. La struttura complessiva ha n ingressi, anche chiamati *feature*, che descrivono matematicamente l'input della rete. Esempi di *feature* possono essere i valori dei pixel di un'immagine da classificare, parametri di un macchinario da controllare oppure i prezzi di un'azione nei giorni precedenti a quello da prevedere. Le m uscite sono i valori che la rete ha prodotto, e possono essere le probabilità che l'input appartenga a una certa classe oppure alcuni valori predetti. Il primo e l'ultimo strato sono "speciali", sono sempre presenti e vengono chiamati *input layer* e *output layer*. I neuroni nell'*input layer* sono n, e ognuno ha un solo ingresso, che costituisce uno degli ingressi della rete. Allo stesso modo, i neuroni dell'output layer sono m, e ognuno ha una sola uscita, che coincide con una delle uscite della rete. Gli altri strati vengono chiamati *hidden layer*, o strati nascosti, in quanto non sono direttamente visibili dall'esterno del modello. Ogni neurone di un *hidden layer* ha come input tutte le uscite dei neuroni nel layer precedente, e trasmette la sua uscita a tutti i neuroni dello strato successivo. Nonostante sia possibile fare in modo che un neurone riceva in input solo alcuni degli output dello strato precedente, quella descritta è la tipologia più diffusa di layers, e viene chiamata *fully connected*, o completamente connessa. Il numero di *hidden layer* di una rete ne determina la profondità: più è alto il numero di *hidden layer* più la rete è profonda, o meglio *deep*.

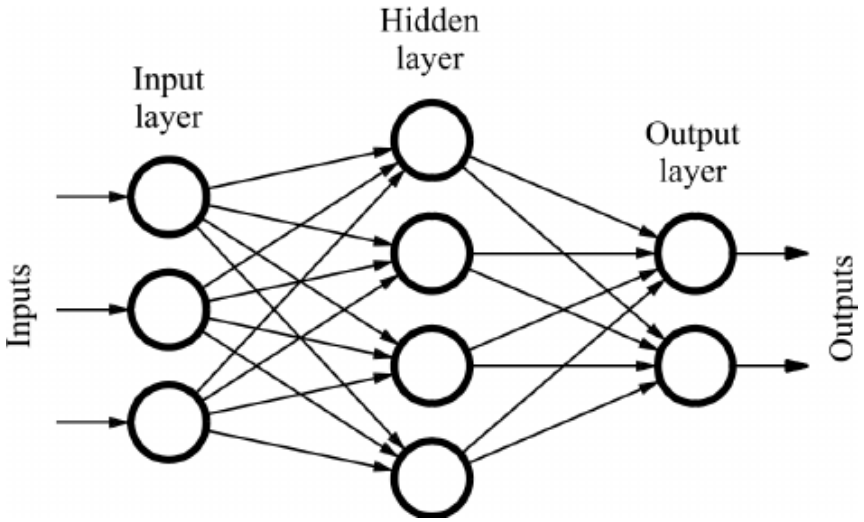


Figura 2.8: Artificial Neural Network

Il processo di apprendimento si riduce alla modifica dei pesi dei percettroni perché producano l'output desiderato. In questo processo si presentano iterativamente vari input con output noto, si osserva il valore prodotto dalla rete, e si cambiano i pesi dei percettroni di ogni in modo tale da avvicinare l'output a quello noto. Il metodo matematico per l'aggiornamento dei pesi è chiamato *backpropagation* [36], e si riferisce alla propagazione all'indietro, dall'uscita all'ingresso, dell'errore commesso dalla rete. Il metodo classico per allenare una rete neurale è quello di dividere il dataset a disposizione in train e test. La prima parte di dataset è quella che iterativamente viene mostrata alla rete per aggiornare i pesi. Una volta che la rete è "allenata", si utilizza la seconda parte di dataset per verificarne l'effettiva capacità. Nel caso in cui la rete sia in grado di generalizzare con successo, allora otterrà buoni risultati anche in caso di input che non siano mai stati presentati alla stessa.

Nonostante le ANN siano state ispirate da funzionamento del cervello umano, i punti in comune non sono così tanti. La analogia tra i due non è altro che una spiegazione semplificata di come le ANN operino, che può essere appetibile per chi deve spiegarla ad un pubblico in quanto comprensibile e suggestiva, ma che trova pochi riscontri nella realtà. Quello che è certo è che non è ancora perfettamente chiaro il funzionamento dei singoli neuroni biologici e soprattutto del meccanismo che porta il sistema complessivo ad imparare comportamenti e memorizzare concetti, nonostante lo studio del sistema nervoso animale abbia compiuto passi da gigante negli ultimi anni. La motivazione per la quale le reti neurali hanno riscosso grande successo nell'ambito scientifico, piuttosto che per l'effettiva somiglianza con il cervello animale, è per la loro capacità di imparare funzioni complesse. Per il teorema dell'approssimazione universale [5] una rete neurale con un solo *hidden layer* avente un numero sufficiente di neuroni è in grado di approssimare una qualsiasi funzione $\mathbb{R}^n \to \mathbb{R}^m$, quindi è possibile creare per ogni dataset una rete che mappi perfettamente ogni input con ogni output.

2.2.2 Convolutional Neural Networks

Le reti *fully connected* hanno la certezza matematica di poter approssimare qualsiasi funzione, tuttavia richiedono un gran numero di pesi. Spesso questo svantaggio ne rende impraticabile l'uso esclusivo per molte delle tipologie di input più interessanti. Per fare un esempio numerico, si supponga di avere a disposizione la serie storica giornaliera dei valori di un indice azionario. Ad ogni valore, si decide di affiancare 14 valori di indicatori tecnici ritenuti rilevanti allo scopo. La serie storica viene quindi suddivisa in intervalli di 100 giorni, e lo scopo della rete è quello di determinare il valore degli ultimi 30 giorni date le informazioni sui primi 70. L'*input layer* è quindi costituito da $70 \times 15 = 1050$ neuroni. Com'è spesso ragionevole supporre, si vuole che il numero dei neuroni nel primo *hidden layer* sia superiore a quello dell'*input layer*, e per semplicità si sceglie 1500. Dato che ogni neurone dell'*input layer* contribuisce con un certo peso alla sommatoria di ogni neurone nel primo hidden layer, il numero totale di pesi impiegato risulterà $1050 \times 1500 = 1575000$. È verosimile considerare una rete con almeno 3 *hidden layer*, che per semplicità vengono supposti tutti con lo stesso numero di neuroni. Per ognuna delle 2 connessioni tra gli *hidden layer* sono quindi necessari 1500 x 1500 = 2250000 pesi. Per la connessione finale il numero di pesi necessario è $1500 \times 30 = 45000$. In base a questo veloce calcolo numerico, per realizzare una semplice rete neurale sono necessari $1575000 + 2 \times 2250000 + 45000 = 6120000$ pesi, poco più di 6 milioni. Questa stima numerica è inoltre molto ottimista, dato che spesso si desidera realizzare reti molto più *deep*. Oltre a problematiche di occupazione della RAM, il grandissimo numero di pesi influenza negativamente anche l'intero processo di training, in quanto rende necessario un enorme volume di dati e una grande quantità di tempo, rendendo di fatto la rete ingestibile.

Un altro problema legato all'utilizzo di layer *fully connected* è quello della perdita della correlazione spaziale dei dati di input. L'uscita di ogni neurone dipende dal valore di attivazione di tutti i neuroni dello strato precedente, ma in realtà i valori degli indicatori nei primi giorni sono debolmente correlati con quelli negli ultimi giorni. Quello che i layer *fully connected* non riescono a sfruttare è il principio di coerenza spaziale, che molto spesso vige per serie temporali, immagini, video e dati testuali. Dati "vicini" spazialmente o temporalmente sono molto più correlati di dati "lontani". Basti pensare ai pixel di un'immagine, frame di un video, o parole in un testo.

Queste problematiche hanno portato alla nascita delle reti neurali convoluzionali (o *Convolutional Neural Network*, abbreviato CNN). Le CNN sono reti neurali ispirate da un modello neurofisiologico della corteccia visiva degli animali, e sono costituite da un insieme di layer convolutivi posti in sequenza. Inizialmente sono state introdotte da Kunihiko Fukushima [8], e sono state allenate utilizzando la backpropagation solo qualche anno dopo da Yann LeCun [16]. I layer di una CNN applicano l'operatore di cross-correlazione sulla matrice che gli viene presentata in input. Questa operazione presenta una sottile differenza con la convoluzione nell'indicizzazione dei filtri, è stata ritenuta necessaria pertanto una differenziazione per rigore matematico. Tuttavia nel campo dell'AI i due termini vengono considerati sinonimi ed è consuetudine utilizzare entrambi.

Come mostrato in figura (2.9), per costruire la matrice di output si fa scorrere lungo

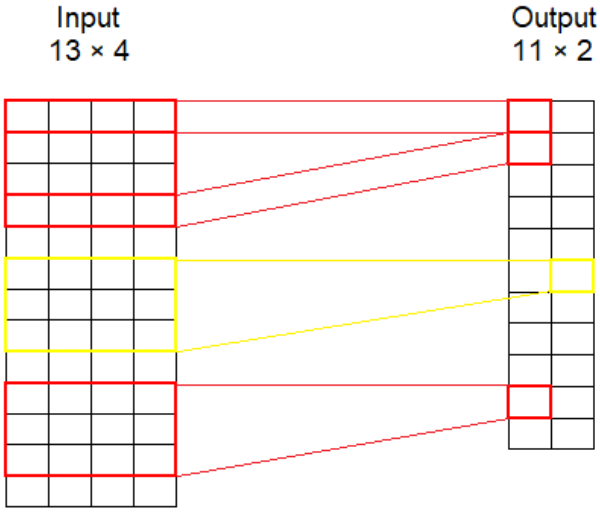


Figura 2.9: Convolutional Neural Network

un asse dalla matrice di input un filtro. Alla somma della matrice che risulta dalla moltiplicazione delle due viene poi applicata la funzione di attivazione per ottenere il valore di un singolo elemento dell'output. Più formalmente si ha:

$$y_{a,b} = f\Big(\sum_{i=0}^{s_f-1}\sum_{j=0}^{n^{(l)}-1} w_{i,j}^{b} x_{a+i,j} + b^{b}\Big) \tag{2.10}$$

Dove la matrice di input x è di dimensione $m^{(l)} \times n^{(l)}$, f è la funzione di attivazione, $w_{i,j}^{b}$ è il peso nella posizione i, j del b-simo filtro, b^{b} è il bias del b-simo filtro e $s_f \times n^{(l)}$ è la dimensione dei filtri utilizzati. Si noti inoltre che la matrice di output è di dimensione $m^{(l+1)} \times n^{(l+1)}$, quindi $b \in [0, n^{(l+1)} - 1]$, dove $n^{(l+1)}$ è il numero di filtri, e $a \in [0, m^{(l+1)} - 1]$, dove $m^{(l+1)}$ è dato da:

$$m^{(l+1)} = m^{(l)} - s_f + 1 \tag{2.11}$$

Il layer descritto è quello che verrà utilizzato e viene chiamato di tipo 1D, in quanto la convoluzione "scorre" solo lungo un asse. È il tipo di layer che viene utilizzato nel caso in cui si stia lavorando con serie temporali o testi. Molto più noti sono i layer convolutivi 2D e 3D, nei quali la convoluzione "scorre" lungo rispettivamente 2 e 3 assi, e vengono utilizzati per gestire immagini e video.

La condivisione dei pesi non è un aspetto positivo solamente per contenerne il numero degli stessi, ma introduce anche robustezza alle traslazioni. I vari filtri impareranno diversi pattern, e scorrendo su tutti i dati di input si attiveranno positivamente nel caso di riconoscimento, indipendentemente dalla zona di input in cui è stato riconosciuto. Supponiamo

quindi di avere una rete con lo scopo di determinare se una certa sequenza di dati ingresso sia vera o falsa. Quello che alcuni filtri possono imparare è a riconoscere pattern anomali, segnalando agli strati superiori una alta probabilità che la sequenza in input sia falsa, indipendentemente da dove questi pattern siano posizionati all'interno della sequenza originale. Se utilizzassimo una rete *fully connected*, il concetto di pattern anomalo dovrebbe essere ripetuto svariate volte, rendendo difficile il training e introducendo di fatto una grande ridondanza. Nelle reti convolutive questa necessità di ripetizione viene sostituita dalla capacità di scorrere dei filtri.

2.2.3 Long Short Term Memory

Le reti che sono state analizzate fino a questo momento appartengono alla categoria delle reti neurali feed-forward (o *Feed Forward Neural Network*, abbreviato FFNN), ovvero reti neurali dove le connessioni tra i vari neuroni non formano cicli, ma conducono direttamente dall'input all'output. Un'altra categoria di reti neurali che spesso vengono utilizzate per la gestione di serie temporali sono le reti neurali ricorrenti (o *Recurrent Neural Network*, abbreviato RNN). Le reti appartenenti a questa categoria hanno connessioni che formano grafi diretti, e che quindi presentano anche cicli al loro interno. Le RNN sono state esplicitamente pensate per la gestione di dati con una dimensione sequenziale, dato che permettono di elaborare ogni elemento della serie temporale singolarmente, mantenendo tuttavia l'ordine nei dati di input. Per fare ciò è necessario introdurre uno stato che rappresenti tutta la serie storica precedente. Ogni nodo della rete avrà come ingresso il valore dello stato nel time step precedente, e il valore dell'input in quel time step. Dopo un'elaborazione, fornirà in uscita il valore aggiornato dello stato e il valore dell'output per quel time step. Matematicamente si avrà:

$$\begin{aligned} s^t &= f(x^t \cdot U + s^{t-1} \cdot W + b_s) \\ o^t &= f(s^t \cdot V + b_o) \end{aligned} \tag{2.12}$$

Dove x^t è il vettore di input di dimensione s_x al time step t, s^t è il vettore di stato di dimensione s_s al time step t e o^t è il vettore di output di dimensione s_o al time step t. La funzione di attivazione, che spesso è sigmoidale o una TanH, è denotata dalla lettera f. Le matrici dei pesi U (di dimensione $s_x \times s_s$), V (di dimensione $s_s \times s_s$) e W (di dimensione $s_s \times s_o$) mappano rispettivamente il vettore di input sul vettore di stato, il vettore di stato precedente su quello attuale e il vettore di stato attuale sul vettore di output. I vettori b_s e b_o sono i bias. Si noti che le matrici dei pesi e i vettori di bias sono unici per il nodo e non variano al variare del time step in input.

Analogamente alle reti feed-forward, anche le RNN vengono allenate utilizzando l'algoritmo di backpropagation. In particolare, utilizzando la versione "srotolata" della rete, visibile nella parte destra della figura (2.10), è possibile propagare l'errore all'indietro, dall'ultimo time step al primo, ovvero da destra verso sinistra. Nonostante teoricamente il metodo non presenti alcun difetto, nella pratica è inutilizzabile in quanto soffre del problema conosciuto in letteratura come *vanishing/exploding gradient problem*. Questa falla è stato scoperto da Sepp Hochreiter [12], ed è legato all'intrinseca profondità delle RNN.

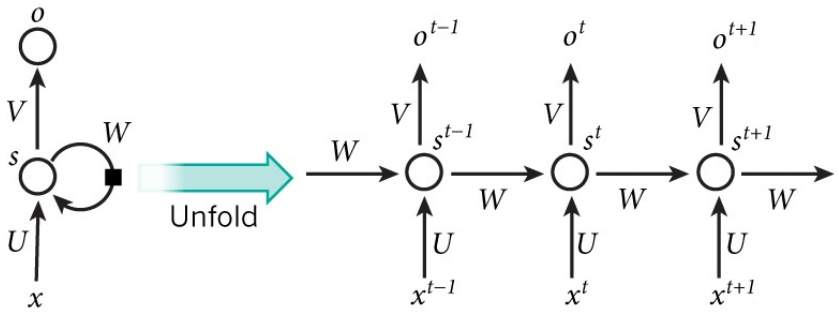


Figura 2.10: Recurrent Neural Network

Mano a mano che l'errore viene propagato all'indietro, questo tenderà in alcuni casi ad aumentare eponenzialmente (*exploding*), portando i pesi dei primi nodi ad essere molto instabili. In altri casi la tendenza dell'errore è quella di annullarsi (*vanishing*), impedendo ai pesi dei primi nodi di cambiare imparando effettivamente qualcosa.

Come soluzione al problema del *vanishing/exploding gradient*, lo stesso Sepp Hochreiter, insieme a Jürgen Schmidhuber, propose l'architettura *Long Short Term Memory* (LSTM) [13]. Le unità di tipo LSTM sono concettualmente simili ai nodi delle RNN in quanto a loro volta processano i dati temporali time step per time step salvando l'informazione sul passato in una cella di memoria o stato, ma si differenziano per l'utilizzo di un *input gate*, un *output gate* e un *forget gate* per l'accesso e la modifica a tale cella. In particolare, in funzione dell'output della cella precedente e dell'input del time step attuale, l'*input gate* decide quanto dell'elaborazione dell'attuale input far aggiornare la memoria, il *forget gate* decide quanto far incidere il valore dello stato precedente su quello attuale e l'*output gate* decide quanto far incidere il valore attuale della cella sull'output. Le formule che governano una cella LSTM sono:

$$
\begin{aligned}
f^t &= \sigma(x^t \cdot W_f + o^{t-1} \cdot U_f + b_f) \\
i^t &= \sigma(x^t \cdot W_i + o^{t-1} \cdot U_i + b_i) \\
\tilde{o}^t &= \sigma(x^t \cdot W_{\tilde{o}} + o^{t-1} \cdot U_{\tilde{o}} + b_{\tilde{o}}) \\
s^t &= f^t * s^{t-1} + i^t * tanh(x^t \cdot W_s + o^{t-1} \cdot U_s + b_s) \\
o^t &= \tilde{o}^t * tanh(s^t)
\end{aligned}
\tag{2.13}
$$

Dove x^t è il vettore di input di dimensione s_x al time step t. I vettori f^t, i^t, $\tilde{o}^t$, s^t, o^t sono rispettivamente l'attivazione del *forget gate*, *input gate* e *output gate*, il vettore dello stato e il vettore di output. Sono tutti di dimensione s_s e riferiti al time step t. Le matrici dei pesi W e U mappano rispettivamente il vettore di input e il vettore di output predecente sull'attivazione del *gate* associato o sullo stato attuale. Le matrici W hanno dimensione $s_x \times s_s$ e quelle U $s_s \times s_s$. I vettori b sono i bias, e hanno dimesione s_s. È opportuno evidenziare che le matrici dei pesi e i vettori di bias sono unici per il nodo e non variano

al variare del time step in input. Si noti che col simbolo "$\cdot$" si intende il prodotto scalare e col simbolo "$*$" la moltiplicazione elemento per elemento.

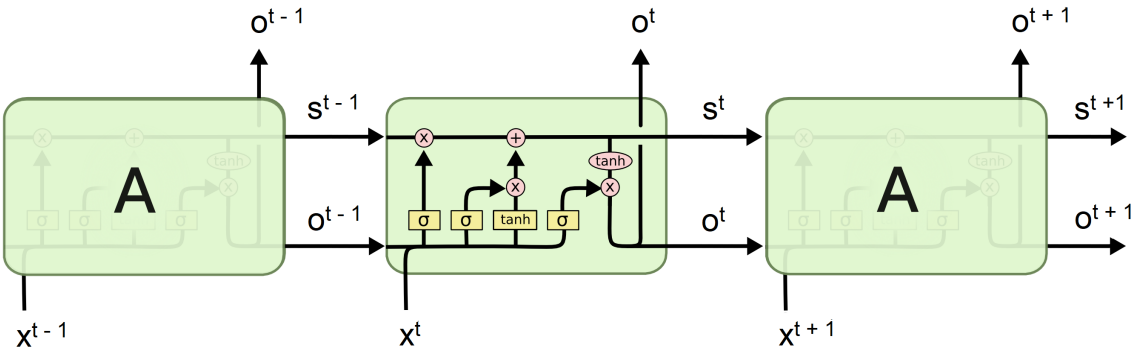


Figura 2.11: Unità di tipo Long Short Term Memory

La presenza di celle di tipo *gated* come quelle LSTM, non solo aumenta la capacità della rete di memorizzare informazioni riguardanti time step molto più addietro, ma risolve parzialmente anche il problema del *vanishing/exploding gradient*, aiutando il flusso dell'errore all'indietro nei time step.

2.2.4 Generative Adversarial Networks

Le *Generative Adversarial Network* (GAN) sono una classe di algoritmi di ML introdotte nel 2014 da Ian Goodfellow [10], e consistono in due reti neurali che competono in un gioco a somma zero. Lo scopo ultimo di questo tipo di reti è quello di generare esempi originali di dati campionati dalla distribuzione di un dataset. Per esempio, se come dataset di partenza si ha un insieme di fotografie di gatti, si vuole creare una rete che generi immagini di gatti che non siano la copia di uno già proposto. Per fare ciò vengono implementate due reti neurali:

- Il *Generator* (G), che ha il compito di generare campioni di dati nella maniera più realistica possibile per far sì che il *discriminator* non riesca a distinguerli da quelli appartenenti dataset originale.

- Il *Discriminator* (D), che ha il compito di determinare se un campione appartiene al dataset originale o è stato generato dal *generator*, cercando quindi di non essere ingannato da questo.

Entrambe le reti vengono allenate allo stesso tempo e il miglioramento di una forza l'altra a migliorarsi a sua volta. In altre parole, il processo di training di D e G consiste in un gioco minimax, descrivibile matematicamente tramite il valore:

$$\min_G \max_D V(D,G) = \mathbb{E}_{x\sim p_{data}(x)}[\log D(x)] + \mathbb{E}_{z\sim p_z(z)}[\log(1 - D(G(z)))] \tag{2.14}$$

Dove x è un generico esempio campionato dal dataset con probabilità p_{data} e z è il rumore campionato dalla distribuzione p_z. Con $G(z)$ si indica il campione generato da G dato l'input z e con $D(x)$ si indica la probabilità stimata da D che il campione x appartenga al dataset originario.

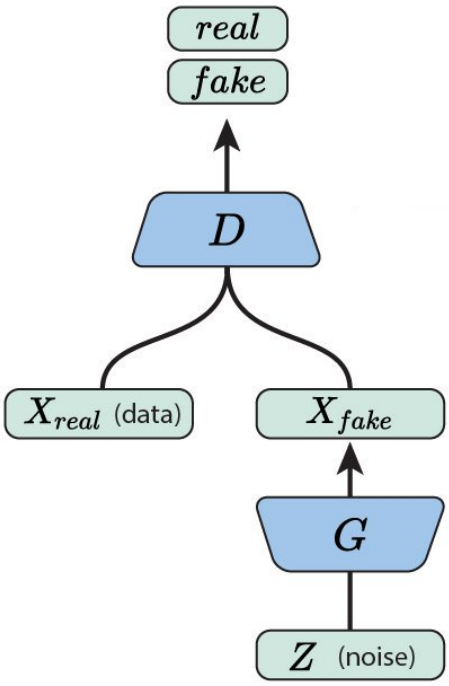


Figura 2.12: Generative Adversarial Network

Le GAN sono state applicate con successo per risolvere diversi problemi di *computer vision*, come la generazione di immagini [10, 31, 23], la manipolazione di immagini [27] e problemi di tipo *text-to-image* [29]. Nel campo delle serie temporali si sono impiegate le GAN per ricostruire frame mancanti di video [28], generare segnali medici [7], frasi [37, 38] o dialoghi [17]. La gran quantità di possibili applicazioni delle GAN dimostra quanto la struttura su cui si basano sia flessibile e possa essere utilizzata per molteplici scopi. Quello che ci si propone in questa tesi è di esplorare i possibili utilizzi delle GAN nella generazione di dati finanziari.

Capitolo 3

Stocks Analysis e GANs

In questo capitolo si analizzerà a fondo come sia possibile utilizzare delle GAN per analizzare il prezzo di indici azionari. Ci si concentrerà in particolare sul processo che ha accompagnato lo sviluppo del progetto legato alla tesi: a partire dalla definizione del problema, enunciata nella sezione 3.1, si è proseguito con l'analisi di fattibilità iniziale, presentata in sezione 3.2. Nella sezione 3.3 si mostrerà il metodo con cui si sono scelti gli iperparametri della rete e infine, nella sezione 3.4 si illustreranno le scelte progettuali effettuate e le soluzioni alle problematiche che si sono dovute affrontare nello sviluppo del progetto.

3.1 Definizione del problema

Quello che si è visto più spesso nell'impostazione dei problemi che vengono risolti con le GAN è parecchio differente dall'approccio utilizzato in questa tesi. La motivazione fondamentale è perché si cerca di utilizzare un algoritmo di tipo *unsupervised* per risolvere un problema di regressione, ovvero di carattere *supervised*.

L'apprendimento di tipo supervisionato (o *supervised learning*), è una tecnica di *machine learning* che si pone l'obiettivo di imparare la funzione che mappa dall'input all'output. Esempi di problemi di *supervised learning* sono:

- Classificazione: a partire dalle *feature*, si vuole sapere la classe dell'esempio analizzato. Per esempio, data un'immagine di un animale domestico, si vuole sapere se questo sia un cane, un gatto, un criceto etc...
- Regressione: a partire dalle *feature*, si vuole sapere il valore di una variabile legata a questi. Per esempio, si vuole stimare il prezzo di una casa date le dimensioni, la città, anno, etc...

L'apprendimento di tipo non supervisionato (o *unsupervised learning*), è una tecnica di *machine learning* che si pone l'obiettivo di imparare la struttura e la distribuzione dei dati presentati in input, non avendo a disposizione l'output. Esempi di problemi di *unsupervised learning* sono:

- *Clustering*: si vuole suddividere tutto il dataset in un certo numero di gruppi, dove elementi simili stanno in gruppi simili. Per esempio, date le caratteristiche di ogni animale, si vuole introdurre una nuova categorizzazione del regno animale.

- *Anomaly detection*: si vuole individuare gli esempi che non sono conformi a quelli presenti nel dataset. Per esempio, dati un insieme di transizioni finanziarie, si vogliono trovare quelle di carattere fraudolento.

Il problema che si vuole affrontare è quello di generare il valore di un indice azionario per 30 giorni, avendo a disposizione il valore dello stesso indice, insieme a 14 valori di indicatori tecnici, per i 70 giorni precedenti. Per una descrizione più rigorosa della costruzione del dataset si veda la sezione 5.1. I valori dell'indice azionario generati dovranno essere più verosimili possibile. È necessario che la serie generata sia indistinguibile dalla serie originale, ovvero che non presenti artificiali andamenti lineari o eccessivi picchi giornalieri, e che presenti un trend simile a quello originale. Il problema descritto è di tipo *supervised*, in quanto si hanno a disposizione dati sui giorni passati come input e si hanno i valori dell'indice nei giorni successivi come output desiderato. Le GAN, tuttavia, sono utilizzate spesso nel contesto *unsupervised*, in quanto si chiede indirettamente al *generator* di imparare la distribuzione di un dataset per generare esempi originali campionati da questa.

La soluzione a questa apparente incongruenza è quella di introdurre un *context* al *generator*, di fatto utilizzando delle *Context-Conditional GAN* (CCGAN) [6]. I dati in input costituiscono il contesto dato al generator per creare i dati di output. Per una descrizione più rigorosa dell'architettura del modello si veda la sezione 4.1. Può sembrare una forzatura della struttura delle GAN, ma quello che si cerca di fare è di trovare una rete che, elaborando dati relativi ai giorni precedenti, riesca ad imparare la distribuzione dell'andamento del titolo nei giorni successivi. Nonostante questo poi non porti alla generazione di serie temporali simili all'originale, si vuole esplorare la possibilità che mediamente il trend di queste segua quello della serie originale.

3.2 Analisi di fattibilità

Prima di procedere con il training del modello *end-to-end*, si è ritenuta necessaria verificare la fattibilità del progetto. In questa prima fase quindi si è stressato particolarmente il *discriminator*, in quanto c'erano ragionevoli motivi per sospettare che questo potesse non essere in grado di svolgere il suo compito. Se l'evoluzione dei prezzi dei titoli analizzati dovesse realmente seguire una *random walk*, come affermato dall'omonima teoria, allora per il *discriminator* non ci sarebbe stato alcun modo di distinguere una sequenza casuale da una vera. Un'analisi dettagliata della teoria del *random walk* è presentata nella sezione 2.1.3. Se questo scenario dovesse presentarsi, i risultati per la GAN sarebbero devastanti. Il *generator* rimarrebbe bloccato in una soluzione subottimale, in quanto gli basterebbe imparare a generare *random walk* per battere sempre il *discriminator*. Il modello divergerebbe in fretta, impedendo sia a G che a D di imparare una qualche funzione utile.

Per provare la capacità operativa del *discriminator*, e per fornire di fatto una controprova alla teoria del *random walk*, si sono eseguiti vari test sui dati a disposizione. In particolare, per ogni titolo a disposizione, si è allenato un *discriminator* a distinguere i dati reali dai dati generati da una *random walk gaussiana*, con parametri μ e σ calcolati a partire dai dati originali. In figura 3.1 è possible vedere i risultati dell'esperimento. Per

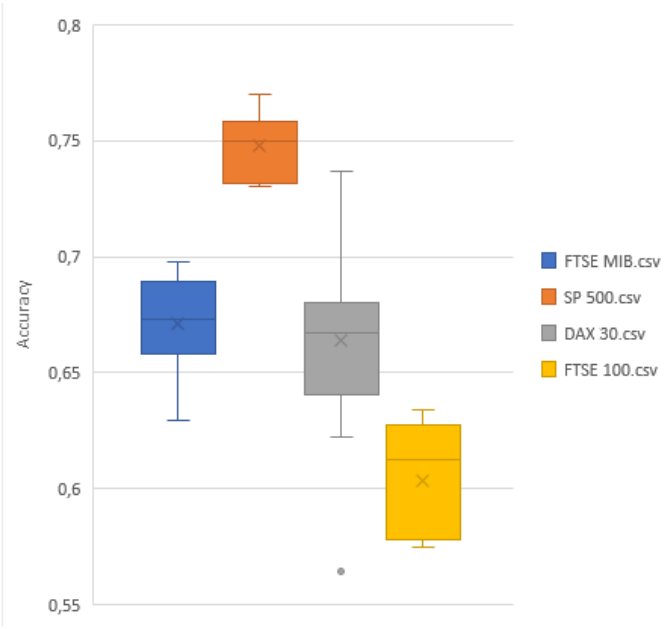


Figura 3.1: Accuracy del discriminator nel validation set nel distinguere random walk da dati originali.

ognuno degli indici analizzati è stato diviso il dataset come spiegato nella sezione 5.1 e sono state compiute 10 prove, calcolando per ognuna l'*accuracy* del *discriminator* nel *validation set*. Quello che si evince da questa prova iniziale è che D è sufficientemente abile nel discriminare serie temporali reali da *random walk*. In nessuno degli esperimenti l'*accuracy* è risultata meno del 50%, e per tutti e 4 gli indici la mediana supera il 60%. Quello che è più interessante da analizzare è come le prestazioni di D varino a seconda dell'indice analizzato. Oltre a provare l'effettiva capacità di D, questo esperimento può essere utilizzato con successo per costruire una sorta di "indice di imprevedibilità" per il mercato analizzato. Secondo quanto risulta, il FTSE 100 è molto più imprevedibile dell'S&P 500, in quanto è molto più difficile per il *discriminator* distinguerlo da una *random walk*.

Per esplorare meglio le capacità del *discriminator*, si sono costruiti diversi generatori di *random walk*, e si è esplorata la capacità di questo di distinguere sequenze generate con diversi metodi da quelle reali. I diversi generatori implementati sono:

- *Random walk generator*: è il generatore utilizzato anche per la prima prova. Per costruire ognuna delle serie prima calcola media μ e deviazione standard σ del valore reale dell'indice in tutto il periodo analizzato. La serie generata sarà quindi

campionata da una gaussiana con media μ e deviazione standard σ.

- *Alpha random walk generator*: il generatore si comporta esattamente come il *random walk generator*, ma solo per $(100 - \alpha)\%$ dei valori finali. Gli $\alpha\%$ valori iniziali sono presi da una serie reale.

- *Real random walk generator*: il generatore costruisce ogni serie campionando casualmente valori all'interno di tutta la serie reale.

- *Alpha real random walk generator*: il generatore si comporta esattamente come il *real random walk generator*, ma solo per $(100 - \alpha)\%$ dei valori finali. Gli $\alpha\%$ valori iniziali sono presi da una serie reale.

- *Conditional random walk generator*: Per costruire ognuna delle serie prima calcola media μ e deviazione standard σ del valore dell'indice in una delle serie reali. La serie generata sarà quindi campionata da una gaussiana con media μ e deviazione standard σ.

- *Alpha conditional random walk generator*: è il generatore che crea i dati più realistici. Calcola la media μ e deviazione standard σ degli $\alpha\%$ valori iniziali, e li usa per generare i restanti valori campionando da una gaussiana di media μ e deviazione standard σ.

Si noti che i valori dell'indice utilizzati non sono il valore reale, ma la normalizzazione della differenza finita del valore di un giorno rispetto al giorno prima. In figura 3.2 è possible vedere i risultati dell'esperimento. Per ogni tipo di generatore è stato diviso l'indice FTSE MIB come spiegato nella sezione 5.1 e sono state compiute 10 prove, calcolando per ognuna l'*accuracy* del *discriminator* nel *validation set*. I risultati di questa seconda prova sono in linea con quelli ottenuti nella prima prova. L'introduzione dei generatori di tipo alpha è stata fatta per simulare quello che il *discriminator* dovrà analizzare quando verrà inserito nel contesto della GAN. Nell'esempio si è studiato il caso in cui $\alpha = 50\%$. Essendo costituiti in parte da serie reali, il *discriminator* avrà più difficoltà nel differenziarli da queste. Dai risultati ottenuti si evince come non ci sia molta differenza nell'utilizzo di *random walk gaussiane* con parametri calcolati sull'intero periodo o su una sola serie. Generare serie campionando casualmente da valori reali, tuttavia, genera esempi meno difficili da distinguere da quelli reali. Questa è un ulteriore riprova di come dati di giorni conseguenti non siano completamente scorrelati. È opportuno specificare che per entrambi i grafici i dati mostrati sono relativi al *validation set*, ovvero che le sequenze reali e casuali che il *discriminator* analizza non sono state mostrate in input nella fase di training. Per ulteriori dettagli si veda la sezione 5.1.

I risultati ottenuti possono essere usati efficacemente come controprova alla teoria del *random walk* e indicano che il modello proposto può essere allenato con successo a generare serie più informative di semplici serie casuali.

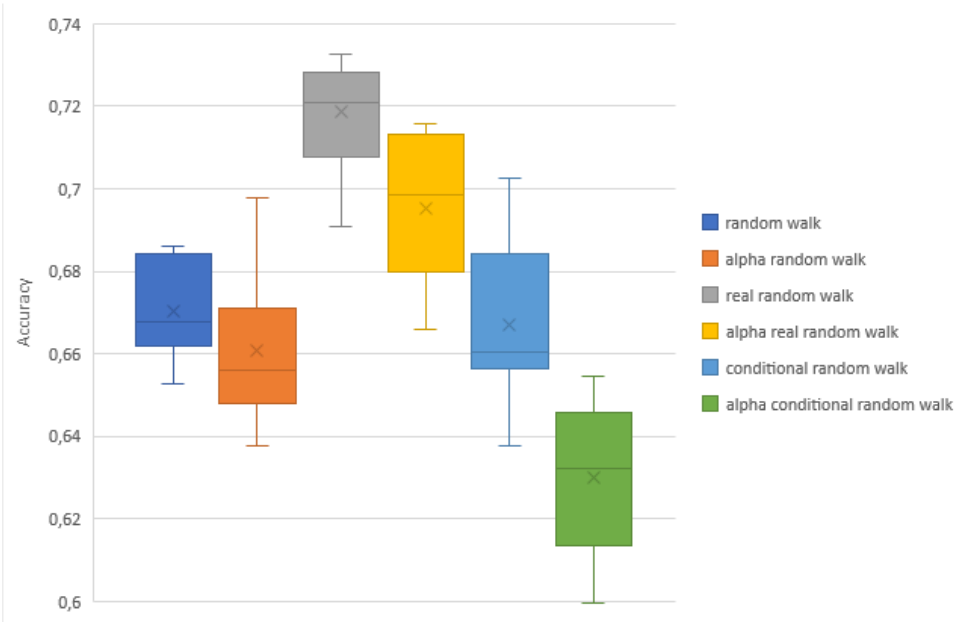


Figura 3.2: Accuracy del discriminator nel validation set del FTSE MIB a seconda del generatore di random walk

3.3 Hyperparameter tuning

Per *hyperparameter tuning* si intende il processo di selezione del set di valori ottimali degli iperparametri della rete. Un iperparametro è un qualsiasi parametro della rete che non può essere allenato con la *backpropagation*, ma che deve essere settato prima che il processo di apprendimento inizi. Esempi di iperparametri sono: il numero di layers della rete, le dimensioni di ogni layer e i parametri dell'ottimizzatore, di regolarizzazione e di costruzione del dataset. La descrizione esaustiva del set di iperparametri analizzati con il relativo valore ottimale è presente nella sezione 5.2.2.

3.3.1 Come allenare una GAN

Le GAN, e in generale ogni rete composta da sottoreti che collaborano o competono, sono particolarmente difficili da allenare con successo. Lo scenario che spesso si presenta è quello di una convergenza prematura a una soluzione subottimale, con una rete che vince ampiamente sull'altra. Indipendentemente da quale rete dovesse imporsi sull'altra, la mancanza di un valido avversario porterebbe la rete prevaricatrice a non avere necessità e stimoli a migliorarsi ulteriormente, impedendo al *generator* di produrre risultati utili. Nel corso di vari studi, la comunità scientifica ha sperimentato diverse *good practices* per favorire la stabilità delle reti. La maggior parte di queste sono state raccolte da Soumith Chintala in un workshop al NIPS [3]. Di seguito verranno elencate quelle che sono state utilizzate anche nel progetto della tesi:

1. Normalizzare l'input: modificare l'input della rete per fare in modo che abbia media $\mu = 0$ e deviazione standard $\sigma = 1$.

2. Modificare la loss: invece che utilizzare la formula (2.14) come *objective function*, ovvero quella presente nel paper originale [10], quello che si consiglia di fare è di utilizzarne una versione modificata:

$$\begin{aligned} \max_D V(D,G) &= \mathbb{E}_{x\sim p_{data}(x)}[\log D(x)] + \mathbb{E}_{z\sim p_z(z)}[\log(1 - D(G(z)))] \\ \max_G V(D,G) &= \mathbb{E}_{z\sim p_z(z)}[\log D(G(z))] \end{aligned} \tag{3.1}$$

   Praticamente utilizzando target 1 per dati provenienti dal dataset e 0 per quelli provenienti dal *generator* quando si sta allenando il *discriminator*, e scambiando i target quando si allena il *generator*.

3. Evitare i gradienti sparsi: invece che utilizzare ReLU, impiegare LeakyReLU. Per il *downsampling* nelle reti convolutive preferire l'utilizzo di *stride* al *pooling*.

4. Utizzare il metodo Adam [14]: il metodo di ottimizzazione Adam può essere utilizzato sia per il *generator* che per il *discriminator*. Per quanto riguarda i suoi parametri, sono ampiamente utilizzati 0.0002 come *learning rate* α, 0.9 per β_1 e 0.999 per β_2. Queste impostazioni dell'ottimizzatore sono state ritenute valide sorprendentemente per quasi tutti le diverse applicazioni, e raramente si è ritenuto necessario modificarle.

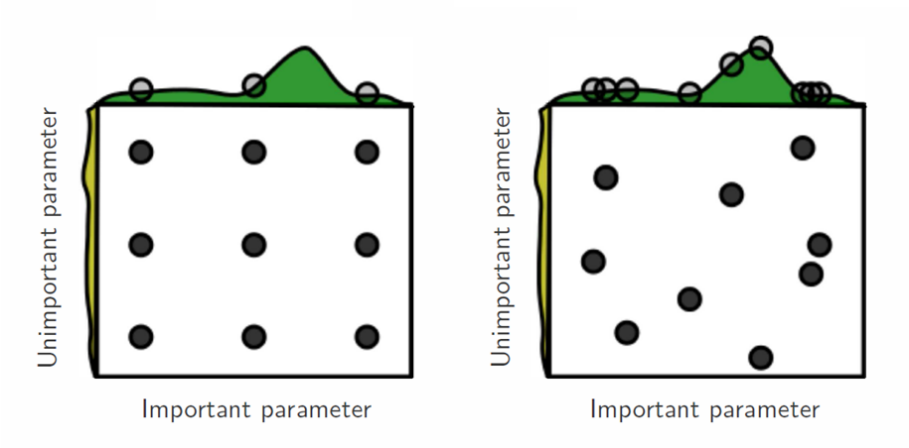
5. Non bilanciare il training tramite statistiche: nel caso in cui una delle due reti prevarichi, intuitivamente potrebbe risultare una buona scelta quella di allenare la rete più "debole" più dell'altra, aiutandola a raggiungere l'altra e bilanciando il training. Quello che è stato dimostrato empiricamente è che questo metodo non funziona, o perlomeno che è molto difficile trovare un'euristica e stabilire quanto una rete debba essere allenata più dell'altra.

3.3.2 Grid search e random search

Per quanto riguarda l'insieme di iperparametri specifici della rete in esame, per i quali non è presente alcuna *good practice*, è necessario il portare a termine in prima persona il processo di *hyperparameter tuning*. Quello che praticamente si fa è esplorare lo spazio degli iperparametri, campionando ripetutamente diverse configurazioni per poi valutarle e decidere alla fine quale di queste sia la migliore. Il campionamento dei valori dei diversi iperparametri può essere fatto tramite:

1. *Grid search*, figura 3.3a: consiste in una ricerca esaustiva all'interno di ogni iperparametro, muovendosi di step di distanza predefinita. Quello che si forma è una griglia n-dimensionale, dove n è il numero di iperparametri da ottimizzare.

2. *Random search*, figura 3.3b: per ogni configurazione, ogni iperparametro viene campionato in maniera casuale tra tutti i suoi valori possibili.



(a) (b)

Figura 3.3: Hyperparameter tuning di due iperparametri tramite: grid search (a), random search (b)

Il metodo che si tende ad utilizzare di più nell'ambito di *deep learning*, e che è stato utilizzato per esplorare anche lo spazio degli iperparametri del modello analizzato, è il *random search*. Nonstante il *grid search* permetta di esplorare in maniera più sistematica tutto lo spazio, il numero di configurazioni da analizzare cresce esponenzialmente con il numero di iperparametri. Supponendo di avere 10 iperparametri, e di suddividere ognuno di questi in 5 intervalli, quello che risulta dalla *grid search* è che bisogna esplorare almeno 5^{10} configurazioni, ovvero quasi 10 milioni. Questa valutazione è molto ottimista, in quanto di solito quello che si vuole fare è esplorare più di 5 valori per ogni iperparametro, e vi sono più di 10 iperparametri. Nella pratica quello che si sarebbe costretti a fare è di limitare grandemente il numero di intervalli in cui suddividere ogni iperparametro, calando di fatto la precisione dell'esplorazione. Utilizzando una *random search*, invece, il numero di configurazioni da esplorare è puramente arbitrario. Inoltre, come è possibile vedere in figura 3.3, la *random search* funziona molto meglio della *grid search* nel caso in cui ci sia un iperparametro che più di altri incide sulle performance della rete. Campionando casualmente si proverà un valore diverso dell'iperparametro dominante per ogni configurazione analizzata, invece che procedere a passi discreti, di fatto evitando di sprecare tempo cambiando valori di iperparametri meno incisivi.

Per ogni configurazione analizzata, quello che si fa è di allenare la rete mostrandole esempi presi dal *training set*, per poi valutare la metrica del modello ottenuto sul *validation set*. La metrica utilizzata, meglio descritta nella sezione 3.4.1, è l'errore percentuale all'ultimo giorno.

3.4 Problematiche affrontate e soluzioni

In questa sezione si analizzeranno tutte le problematiche e le decisioni progettuali che si sono affrontate nel corso del progetto di tesi, analizzando per ognuna eventuali alternative e soluzioni. In primo luogo, si parlerà della metrica scelta per valutare la qualità di un modello, poi si introdurrà il classico problema di *trade-off* tra *underfitting* e *overfitting* e infine si parlerà del problema della variabilità delle soluzioni, tipico delle GAN.

3.4.1 Scelta della metrica

Dopo aver costruito il dataset, creato il modello e averlo allenato, il passo successivo è quello di stabilire quanto questo sia abile nello svolgere il compito a cui è stato assegnato, e per questo si utilizza una metrica. La metrica è semplicemente uno scalare che indica quanto bene (o quanto male) la rete stia facendo sul dataset in esame, al fine di confrontare due diversi modelli e stabilire il migliore dei due. Essendo l'unico parametro su cui la scelta del modello migliore verrà effettuata, la decisione della metrica è di delicata importanza e deve rappresentare in maniera più completa possibile quello che ci si aspetta dal modello.

Uno dei casi più famosi nell'ambito del *machine learning* di scelta errata di metrica è l'*accuracy paradox*. Per *accuracy* si intende il numero di predizioni corrette diviso il numero totale di predizioni effettuate dal modello, e viene utilizzata principalmente per problemi di classificazione. Nonostante l'*accuracy* venga spesso utilizzata come metrica standard, per alcuni problemi non risulta la metrica più opportuna. Pertanto può capitare che, tra due modelli, venga scelto come migliore quello con *accuracy* minore. Si supponga di dover creare un rivelatore di incendi, che analizzando i dati forniti dai sensori all'interno di un ambiente, debba segnalare preventivamente un incendio nel caso se ne stia per verificare uno. Come dataset si hanno a disposizione tutti i campionamenti raccolti in un anno dai sensori installati nei vari siti di interesse, con il corrispettivo valore booleano che rappresenta il verificarsi un incendio. Si supponga quindi di aver 100000 campioni e che si siano verificati effettivamente solo 5 incendi. Uno dei modelli che in termini di *accuracy* ha prestazioni migliori in questo tipo di problemi è il modello che da come output sempre falso. Nel caso specifico, la sua *accuracy* sarebbe 99.995% nonostante questo sia totalmente inutile. Quello che nella realtà si fa è preferire modelli con più alta tendenza a far scattare l'allarme. In questo modo spesso si scelgono modelli con minor *accuracy* e con il rischio di falso allarme, ma con più alta probabilità che l'allarme suoni nel caso di incendio, che è il motivo per cui sono stati implementati in primo luogo. Una metrica che è può meglio rappresentare modelli in problemi come questi e spesso si utilizza nel caso di classi sbilanciate è l'*F1-score*.

Macroscelta: realisticità o efficacia?

La prima scelta che si è dovuto effettuare riguardo alla metrica è di carattere generale. È necessario premiare la realisticità o l'efficacia del modello nel produrre scenari di mercato?

- Per realisticità si intende la capacità della rete di creare scenari più verosimili possibili, ed è la proprietà che spesso si ricerca se si stanno utilizzando delle GAN. Una

metrica di questo tipo è l'*inception score* [31], che utilizza un classificatore anticipatamente allenato (la rete *inception*, appunto) per valutare quanto i sample generati siano realistici e vari. Più l'output della rete *inception* è dominato da una classe più l'esempio è realistico, e più la distribuzione dell'output su tutti i campioni generati è uniforme più la rete fornisce esempi variegati.

- Per efficacia si intende la capacità della rete di creare scenari più vicini possibile a quello reale, ed è la proprietà che spesso si ricerca per modelli di tipo *supervised*. Una metrica di questo tipo è l'errore quadratico medio (o *Mean Squared Error*, abbreviato MSE), che quantifica quanto il campione generato si discosta da quello originale:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y_i})^2 \tag{3.2}$$

  Dove n è il numero di valori previsti nello scenario generato, y_i è il valore reale al giorno i e $\hat{y_i}$ è il valore previsto al giorno i.

Nonostante spesso nello studio delle GAN si premi i modelli in base alla realisticità dei sample generati, quello che si è deciso di fare è di premiare i modelli più efficaci nel generare campioni. La motivazione principale che ha portato a questa scelta è che l'*inception score* si basa su una rete di grandi dimensioni anticipatamente allenata, *state of the art* nella classificazione di immagini. Sfortunatamente non ci sono o comunque non è possibile avere accesso a questo tipo di reti per la gestione di dati finanziari. Inoltre, il problema in esame è di tipo *supervised* e lo scopo ultimo della rete è quello di creare scenari che si avvicinino a quello reale, per guadagnare maggiori informazioni sull'andamento futuro del mercato. Si noti che quella di cui si parla in questa sezione è la metrica con cui scegliamo gli iperparametri, non la *loss*, ovvero non è l'errore che verrà utilizzato per aggiornare i pesi della rete, ma è solo un valore che ci indica le performance di una rete col set di iperparametri analizzato.

Microscelta: come definire la qualità di un sample?

Stabilito che la metrica dovesse premiare l'efficacia della soluzione proposta, è stato necessario decidere, data la serie reale e quella generata, come praticamente calcolare la vicinanza tra le due. Le possibilità analizzate sono:

1. MSE: già introdotta nella sezione precendente e descritta dalla formula (3.2). Prende in considerazione tutta la serie generata e ne calcola una distanza punto per punto da quella originale.

2. Direzionalità finale (FD): prende in considerazione solamente il valore finale della serie generata, e di questo valuta se la direzione corrisponde con la direzione della serie reale. Sinteticamente stabilisce se la serie ha predetto correttamente se il valore del titolo crescerà o calerà. La formula del valore di una serie sarà quindi:

$$FD = \begin{cases} 0, & \text{if } (y_n - y_1)(\hat{y_n} - y_1) \leq 0 \\ 1, & \text{if } (y_n - y_1)(\hat{y_n} - y_1) > 0 \end{cases} \tag{3.3}$$

La notazione è coerente con quella di formula (3.2).

3. Errore finale % (FE): prende in considerazione solamente il valore finale della serie generata, e di questo ne calcola la distanza con il valore finale della serie reale in termini percentuali. La formula del valore di una serie sarà quindi:

$$FE = 100 \ \frac{|y_n - \hat{y_n}|}{y_1}\% \tag{3.4}$$

La notazione è coerente con quella di formula (3.2).

4. Differenza di volatilità % (VE): prende in considerazione tutta la serie generata e calcola quanto la volatilità di questa differisca dalla volatilità della serie originale in termini percentuali. La formula del valore di una serie sarà quindi:

$$VE = 100 \ \frac{|\sigma - \hat{\sigma}|}{\sigma}\% \tag{3.5}$$

Dove σ è la deviazione standard della serie di valori reali e $\hat{\sigma}$ è la deviazione standard di quella generati.

Nonostante il MSE sia lo standard nei problemi di regressione, non è stato ritenuto idoneo per il particolare problema e per l'utilizzo della GAN. Il problema principale è che la metrica pesa equivalentemente tutti i giorni della serie, risultando avere poca enfasi nei confronti dei giorni finali, che sono ritenuti più importanti per il problema analizzato. Quello che inoltre può accadere è che le casuali fluttuazioni della serie reale o generata penalizzino la distanza anche di una soluzione con buona qualità. La FD è una metrica migliore della MSE, poiché è più sintetica, si concentra solo sugli aspetti che maggiormente ci interessano. Quello che però capita di osservare è una particolare insensibilità all'*overfit*. La serie generata verrà considerata valida anche nel caso in cui porti a valori esageratamente alti o bassi, a patto che abbia previsto in maniera giusta la direzione. Invece che l'FD, la metrica che si è deciso utilizzare è il FE. Mantenendo tutti gli aspetti positivi della FD, introduce anche la necessità di maggior precisione della rete e scongiura il rischio di serie sproporzionate misurando la distanza del valore finale. Sperimentalmente, la VE risulta poco utile nello stabilire la qualità di una serie, e si è inoltre rivelta poco stabile al cambiamento degli iperparametri. Inoltre, quello che concettualmente si sta cercando di fare utilizzando questa metrica è di rendere la deviazione standard della serie generata simile a quella della serie reale, premiando di fatto la realisticità invece che l'efficacia. Nonostante non sia stata considerata nel processo di *hyperparameter tuning*, può essere interessante utilizzarla per verificare il realismo delle serie prodotte al termine del processo.

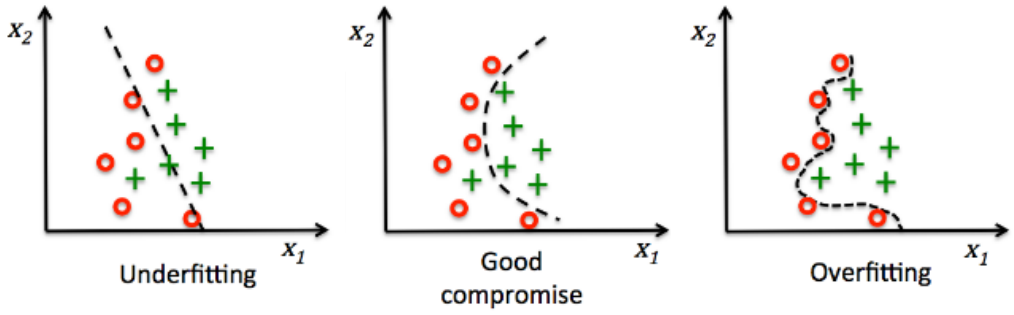
3.4.2 Underfitting e overfitting

L'*underfitting* e l'*overfitting* sono due problemi noti nel campo del *machine learning*, e vengono spesso citati insieme in quanto sono altamente correlati. Spesso prendere decisioni per limitare l'uno comporta un accrescimento dell'altro e per raggiungere una sufficiente generalizzazione al di fuori del *training set* è necessario trovare un compromesso che minimizzi entrambi i fenomeni.

- Per *underfitting* si intende quando una rete non riesce a cogliere adeguatamente la struttura all'interno dei dati sui quali si sta allenando, spesso perché non ha sufficiente complessità. Un esempio si ha in figura 3.4a.

- L'*overfitting* è il fenomeno per il quale capita che il modello impari troppo bene i dati in ingresso e non sia in grado di generalizzare su altri esempi presi dalla stessa distribuzione. Spesso è dovuto a un'eccessiva capacità rappresentazionale. Un modello che soffre di *overfitting* ha erroneamente incluso parte del noise nella propria rappresentazione dei dati. Un esempio si ha in figura 3.4c.

Quello che si ricerca in un modello è il giusto compromesso tra i due, che permetta una corretta generalizzazione, come in figura 3.4b.



(a) (b) (c)

Figura 3.4: Linea di separazione tra le classi nel caso in cui vi sia: underfitting (a), un buon compromesso tra underfitting e overfitting (b), overfitting (c)

Nel processo di creazione e *tuning* della rete si sono incontrati entrambi i fenomeni. Nella parte iniziale dell'esperimento, si è assistito a un *underfitting*, prima del *generator* e poi del *discriminator*. In entrambi i casi è bastato aumentare le dimensioni della rete per renderla sufficientemente potente, semplicemente aumentando il numero di layers e di neuroni per ogni layer.

Quello che si è notato in seguito ai cambiamenti apportati è stato il sorgere di *overfitting* da parte del *discriminator*. In particolare, si ritiene questo abbia "imparato a memoria" i dati presenti nel *training set*, non lasciando alcuna possibilità al *generator* di ingannarlo. Utilizzando tecniche di regolarizzazione come *dropout* [32] e *L2 regularization* [15] si è cercato di limitare la potenza di D, ma senza esito positivo. La soluzione al problema dell'*overfitting* che è stata implementata è quella dell'*early stopping* [26]. In particolare, invece che allenare la rete per un numero prefissato di epoche, la si allena fino a che questo porta un miglioramento e ci si ferma quando si sta cominciando a perdere capacità di generalizzare. Per raggiungere questo obiettivo si è deciso di analizzare le *loss* del *generator* e del *discriminator*. Quello che si nota analizzando la figura 3.5, è che il training della rete si può suddividere in tre fasi:

1. La fase utile: è la prima fase, e nell'esempio di figura va dall'inizio fino a circa l'epoca 500. In questa fase G impara effettivamente a generare sample realistici, e D impara a distinguere i campioni falsi.

2. La fase inutile: è la fase intermedia e si posiziona tra le altre due. In questa non vi sono cambiamenti rilevanti nelle due *loss*, pertanto si ha una fase in cui le due reti apportano piccoli cambiamenti per tentare di sovrastare l'altra, ma senza imparare qualcosa di utile.

3. La fase dannosa: è l'ultima fase, e nell'esempio in figura va da circa l'epoca 1500 fino alla fine. In questa fase comincia l'*overfitting* di D, la sua *loss* comincia a calare e quella di G ad aumentare. L'andamento continua in modo costante e porta la rete alla divergenza.

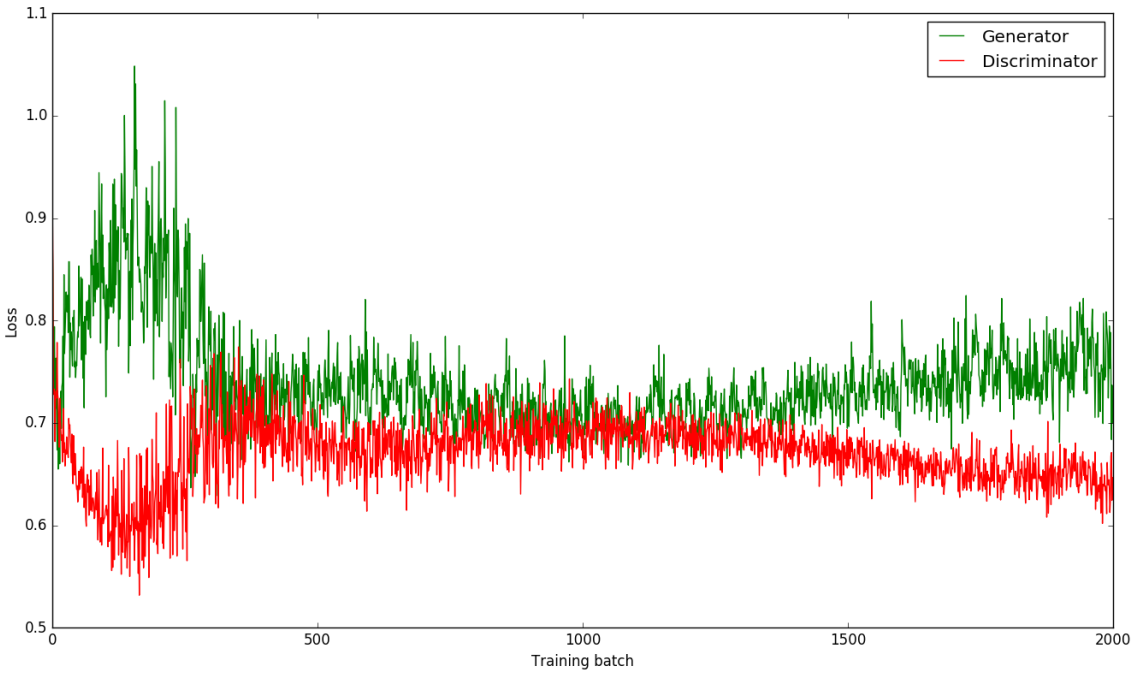


Figura 3.5: Loss del generator e del discriminator per le prime 2000 epoche

L'*early stopping* viene utilizzata per terminare il *training* a cavallo tra la prima fase e la seconda. Nella pratica, se per un certo numero di epoche la differenza in valore assoluto delle due *loss* rimane al di sotto di una determinata percentuale, vuol dire che si è raggiunta la seconda fase, e che quindi l'algoritmo può fermarsi. I dettagli sui parametri dell'*early stopping* possono essere trovati nella sezione 5.2.2.

3.4.3 Scarsa variabilità

Uno dei problemi più famosi dell'utilizzo delle GAN è quello della variabilità delle soluzioni, che spesso viene anche chiamato *mode collapse*. In molti dataset la distribuzione di probabilità presenta un gran numero di picchi, dove si addensano sottogruppi di campioni. Può capitare a una GAN che il *generator* produca campioni provenienti da solo uno di questi sottogruppi, limitando di fatto la variabilità del risultato prodotto.

Si supponga di avere un dataset contenente foto di gatti. Nonostante la grande quantità di immagini a disposizione, i soggetti fotografati sono molti meno. Quello che il *generator* potrebbe imparare a fare è generare immagini che rappresentano solo uno dei gatti presenti. Questa soluzione è una "scorciatoia" per G, in quanto il suo compito viene facilitato dato che si deve occupare solo di rappresentare al meglio solamente uno dei soggetti proposti, e lo scopo di ingannare D non ne risente in alcun modo. Ancora più grave è se questo invece che imparare a generare un solo gatto dovesse imparare a generare perfettamente una singola immagine presente nel dataset.

Nel nostro caso il contesto aiuta a scongiurare il sorgere di questo problema. Cambiando il *context* di input è possibile ottenere diversità nell'output. Tuttavia, quello che si è notato è che, fornendo in input lo stesso contesto, l'output tende ad avere poca variabilità. In figura 3.6 sono mostrati alcuni esempi nei quali si è osservato *mode collapse* e altri in cui questo problema non è presente.

Generatori multipli

Come prima soluzione, si è esplorata la possibilità di utilizzare generatori multipli. In questa ottica ci si limita ad accettare che G si dedicherà solamente a una regione della distribuzione dell'input. Per ovviare a questo problema, si creano n *generator*, e li si allena in maniera congiunta. Così facendo ognuno di questi collasserà su un modo, ma avendone diversi si coprirà tutta la distribuzione di input. La soluzione è intuitiva e semplice da realizzare, ma il parametro n è difficile da regolare: nel caso in cui sia troppo piccolo la nostra rete avrà un esplorazione parziale dello spazio di input, l'aumento eccessivo tuttavia, porta a un maggiore sforzo computazionale e una maggiore difficoltà nell'allenare D. Per questi motivi si è deciso di proseguire con la ricerca di altre soluzioni.

Noise moltiplicativo

La seconda soluzione che si è implementata è quella del noise moltiplicativo. Come è possibile vedere in figura 4.2, nella GAN originale il noise gaussiano viene introdotto in maniera additiva, sommandolo all'*encoding* del *generator*. Si è quindi esplorata la possibilità invece di moltiplicarlo. Analizzando il modello, meglio descritto nella sezione 4.1, si è arrivati alla conclusione che la motivazione per cui non si ha una grande variabilità nelle serie di uscita è che il noise gaussiano a media $\mu = 0$ e deviazione standard $\sigma = 1$ non abbia un impatto sufficiente sull'*encoding*. Per ovviare a questo problema abbiamo esplorato l'ipotesi che moltiplicando il noise invece che sommarlo, si potesse avere limpatto necessario. In seguito a questo cambiamento non si sono notati miglioramenti nella variabilità delle soluzioni, e

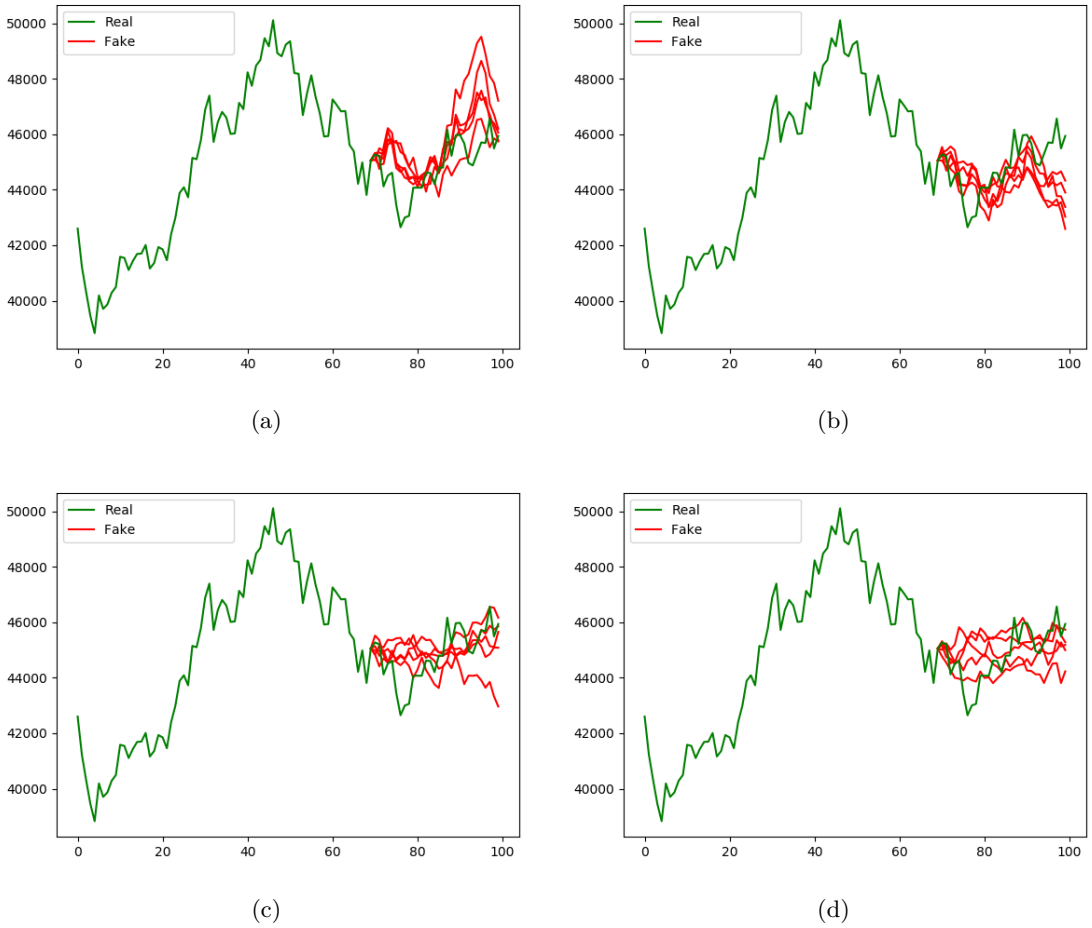


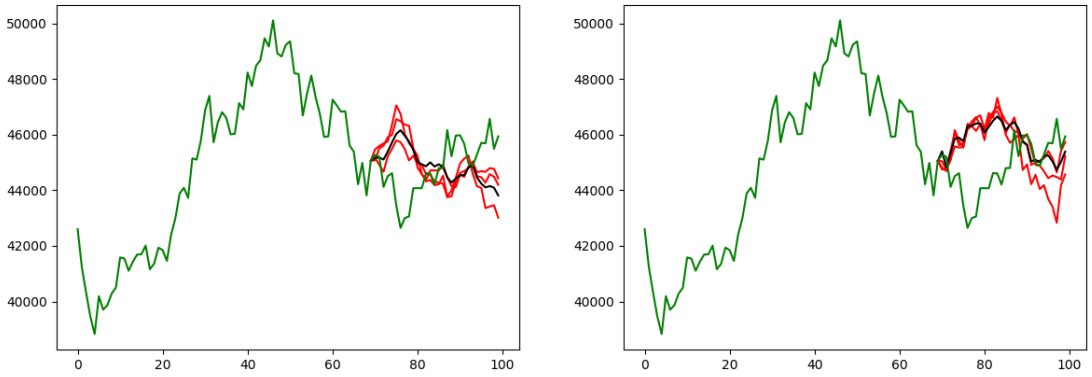
Figura 3.6: Due grafici nel quale è presente il fenomeno di mode collapse (a, b), e due in cui non è presente (c, d). La linea in verde rappresenta l'andamento reale del titolo per 100 giorni, le linee rosse indicano l'output del generator per gli ultimi 30 giorni, calcolato ricevendo come input informazioni sui primi 70.

si è inoltre assistito ad un peggioramento delle performance e della qualità dei risultati prodotti.

Loss adversarial e supervised

Nonostante la prova del noise di tipo moltiplicativo non sia andata a buon fine, questa ci ha comunque dato buone indicazioni sul problema di cui stava soffrendo la rete. Se il fatto che il *generator* non produca esempi con sufficiente variabilità non dipende dal noise in input, allora è probabile che dipenda dalla rete in sé. L'architettura del *generator* e dei suoi componenti è meglio descritta nella sezione 4.1.1.

Quello che si aspetta succeda è che il *decoder* ignori completamente l'input portando tutti i suoi pesi a 0, e producendo l'output solo tramite i bias. Questo comportamento è spiegato dal fatto che le reti neurali naturalmente tendono ad opporsi alle fonti di rumore, e che il *discriminator*, specialmente nelle prime epoche, ha difficoltà nel riconoscere una *random walk* dalla serie originale. Per provare questa teoria, è stata disegnata una semplice prova: settando l'input del *decoder* a 0, è possibile vedere quale sia il suo output nel caso in cui agiscano solo i bias. Il risultato, mostrato in figura 3.7, è perfettamente allineato con quello che ci si aspetta. È possibile notare come la serie di valori generati con solo il bias sia simile a quelle generate con l'input normale.



(a) (b)

Figura 3.7: Due grafici ottenuti da una rete allenata con sola loss supervised (a, b). La linea in verde rappresenta l'andamento reale del titolo per 100 giorni, le linee rosse indicano l'output del generator per gli ultimi 30 giorni con l'input normale, mentre quella nera nel caso si consideri l'input del decoder nullo.

La soluzione al problema è quella che è stata implementata anche nel paper di riferimento [6]: combinare alla *loss* di tipo *adversarial* della GAN, una *loss* di tipo *supervised*. Il motivo per cui questa soluzione risolve il problema è che il *generator*, tentando di rendere il proprio output simile alla serie reale, sarà spinto ad utilizzare i dati su cui è condizionato. La *loss* utilizzata, la sua formula e le implicazioni che ha sulla rete sono dettagliatamente analizzate nella sezione 4.1.3.

Capitolo 4

Modello

In questo capitolo si analizzerà nel dettaglio l'architettura del modello realizzato e di tutti i suoi componenti, insieme alle tecnologie che sono state utilizzate a supporto dello sviluppo. Nella sezione 4.1 si mostra l'architettura del modello, partendo da una visione globale per arrivare ad un'analisi dei singoli componenti. Nella sezione 4.2 verranno introdotti i framework utilizzati.

4.1 Architettura

Come anticipato nella sezione 2.2.4, l'architettura di una GAN è composta da due reti: il *discriminator* (D) e il *generator* (G). G riceve in input un vettore di noise e produce in uscita un campione, e il suo compito è quello di mappare lo spazio di input in quello del dataset. D riceve in input un vettore campionato dal dataset o creato da G e produce in uscita un singolo valore rappresentante la probabilità l'input appartenga al dataset. Nel nostro caso, tuttavia, l'architettura si discosta leggermente da quella originale in quanto abbiamo avuto la necessità di introdurre un contesto per G. Questo, quindi, oltre al noise in ingresso, ha anche una matrice di contesto, che consiste nel prezzo dell'indice in questione per i giorni precedenti, insieme ad alcuni indicatori tecnici opportunamente selezionati. L'architettura è mostrata in figura 4.1.

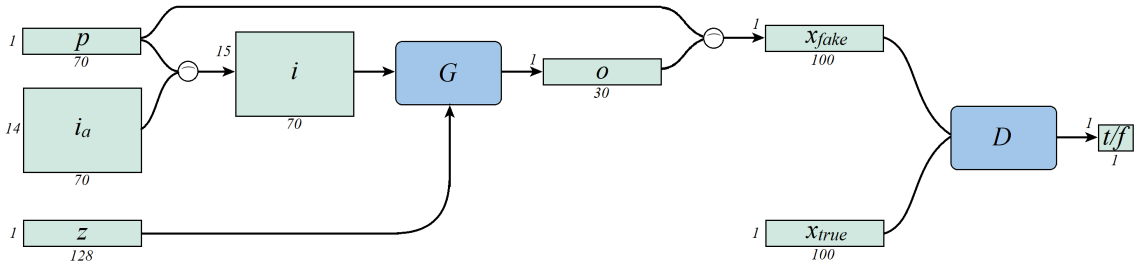


Figura 4.1: Schema a blocchi dell'architettura dell'intero modello

Dove:

- Il *generator* G: è la rete del *generator*, ed è dettagliatamente analizzata nella sezione 4.1.1.
- Il *discriminator* D: è la rete del *discriminator*, ed è dettagliatamente analizzata nella sezione 4.1.2.
- Il simbolo "$\frown$": corrisponde all'operatore di "concatenazione".
- Il prezzo p (1, 70): è il vettore costituito dai valori del prezzo dell'indice analizzato per 70 giorni.
- Le info aggiuntive i_a (14, 70): è la matrice costituita dai 14 valori degli indicatori tecnici selezionati, per ognuno dei 70 giorni.
- Il noise z (1, 128): è il vettore di noise in input, con media $\mu = 0$ e deviazione standard $\sigma = 1$.
- L'input di G i (15,70): è la matrice di contesto che viene data a G per produrre i valori di uscita. Corrisponde alla concatenazione di p con i_a lungo l'asse 0.
- L'output di G o (1,30): è il vettore di output generato, e corrisponde a uno scenario possibile per i 30 giorni successivi secondo G.
- Il campione generato x_{fake} (1,100): concatenando lungo l'asse 1 il prezzo per i primi 70 giorni p con il prezzo per i 30 successivi generato da G o si ha il vettore che costituisce una serie completa "fake".
- Il campione reale x_{true} (1,100): è il vettore che costituisce una serie di 100 valori di prezzo presenti nel dataset.
- L'output di D *t/f* (1,1): è la probabilità data da D che il vettore in input appartenga al dataset.

Lo schema in figura serve a dare un'idea di come il modello in generale funzioni e delle dimensionalità dei dati in input e output. Nelle sezioni seguenti si descriverà in dettaglio la struttura interna dei due modelli, dedicando una particolare attenzione anche al tipo di *loss* che si è implementata per il training di G e di D.

4.1.1 Generator

Il *generator* G ha il compito di fornire una sequenza temporale di prezzi in uscita data una sequenza temporale di informazioni in ingresso, e l'architettura che naturalmente si presta a meglio a questo tipo di problemi è quella di tipo *sequence to sequence* [33]. G sarà quindi formato a sua volta da due reti, che lavorano insieme per trasformare una sequenza nell'altra: un *encoder* E condensa le informazioni fornite in input in un singolo vettore chiamato *encoding* e un *decoder* De, che utilizza questo vettore come contesto per formare

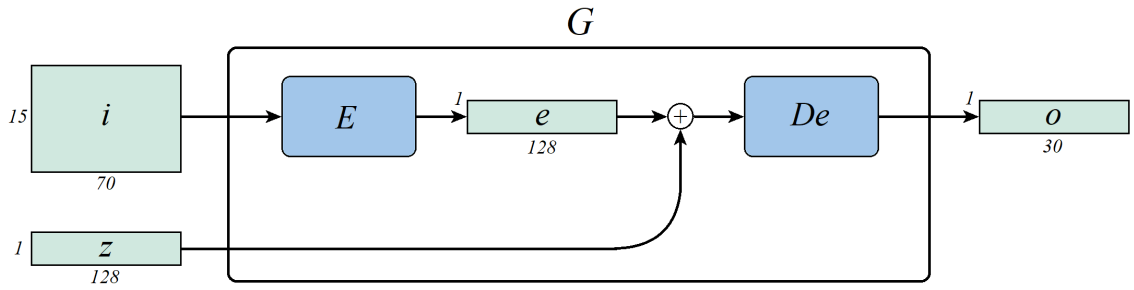


Figura 4.2: Schema a blocchi dell'architettura del generator

la sequenza di uscita. A differenza del paper originale, nel nostro caso all'*encoding* viene sommato il noise. Lo schema a blocchi è mostrato in figura 4.2.

Dove:

- L'input di G i, l'output o, e il noise z coincidono con quelli presenti anche in figura 4.1.
- L'*encoder* E: è la rete dell'*encoder*, ed è costituita da una rete ricorrente con unità LSTM. La sua particolare architettura verrà mostrata in seguito.
- Il *decoder* De: è la rete del *decoder*, ed è costituita da una rete neurale di tipo *fully connected*. La sua particolare architettura verrà mostrata in seguito.
- L'*encoding* e: è il vettore che rappresenta tutte le informazioni presenti nell'input condensate da E, e al quale viene sommato il noise per avere il contesto per De.

L'*encoder* E ha il compito di analizzare i dati relativi ai 70 giorni precedenti alla previsione e codificare tutta l'informazione estratta in un singolo vettore. Per fare ciò si è scelto di mostrare in ingresso di un'unità LSTM di volta in volta le informazioni riguardante un solo giorno, e utilizzare la concatenazione dell'uscita e dello stato all'ultimo istante di tempo come rappresentazione compatta di tutta la serie in ingresso. Con una versione a più unità di quella proposta è stato possibile anche codificare l'informazione presente in una frase scritta in linguaggio naturale [33]. Nella figura 4.3 è possibile vedere una rappresentazione grafica della rete implementata. In questa, con i^n si indica in vettore a 15 dimensioni rappresentante le informazioni del giorno n (la n-esima colonna di i), con s^n e o^n i vettori 64-dimensionali prodotti dall'unità LSTM e rappresentanti rispettivamente lo stato e l'uscita dell'unità. Il vettore e è quello mostrato in figura 4.2. Ulteriori iperparametri della rete sono presenti in sezione 5.2.2.

Quello che differenzia l'architettura implementata da quella proposta nel paper originale [33], oltre alla presenza del noise, è la struttura del *decoder*. Invece che utilizzare un'altra rete LSTM, si è deciso di utilizzare una rete neurale *fully connected* o *dense*. Le motivazioni che hanno portato a questa decisione sono sia teoriche, dato che si è preferito conferire una maggiore capacità rappresentativa essendo limitate le dimensionalità di input e di output, che pratiche, in quanto si è notata una maggiore variabilità di scenari

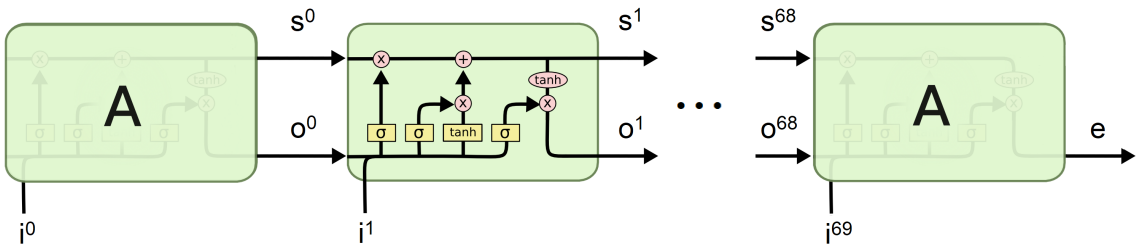


Figura 4.3: Architettura dell'encoder

utilizzando una rete *dense*. Lo schema della rete implementata è mostrato in figura 4.4 e ulteriori iperparametri della rete sono presenti in sezione 5.2.2.

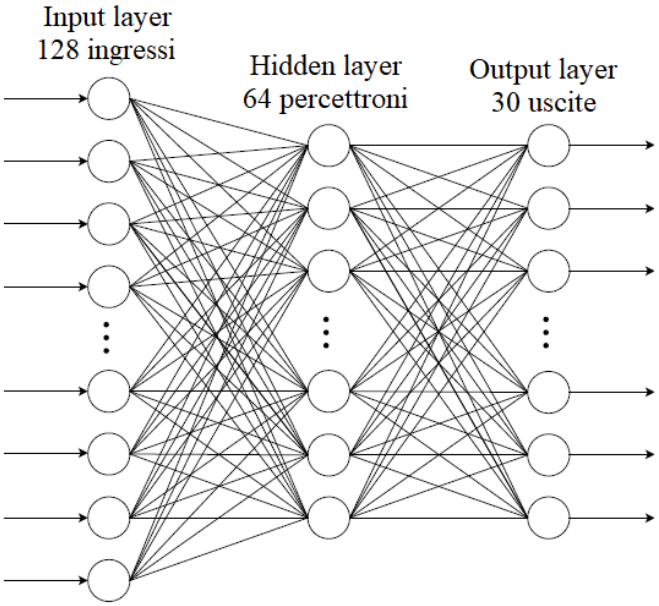


Figura 4.4: Architettura del decoder

4.1.2 Discriminator

Il *discriminator* D ha il compito di stabilire se la serie che gli viene presentata in input appartiene al dataset o è generata da G. Per fare ciò si è deciso di implementare una rete convolutiva che analizzi la serie in input e che produca in output un singolo valore rappresentante la probabilità dell'input di appartenere al dataset. Questa rete presenta come ultimo strato un layer *fully connected*, che permette alla rete di sfruttare le informazioni estratte dai vari filtri per stimare il proprio obbiettivo. In figura 4.5 vengono mostrati

i dettagli dei layer della rete, mentre ulteriori iperparametri sono descritti nella sezione 5.2.2.

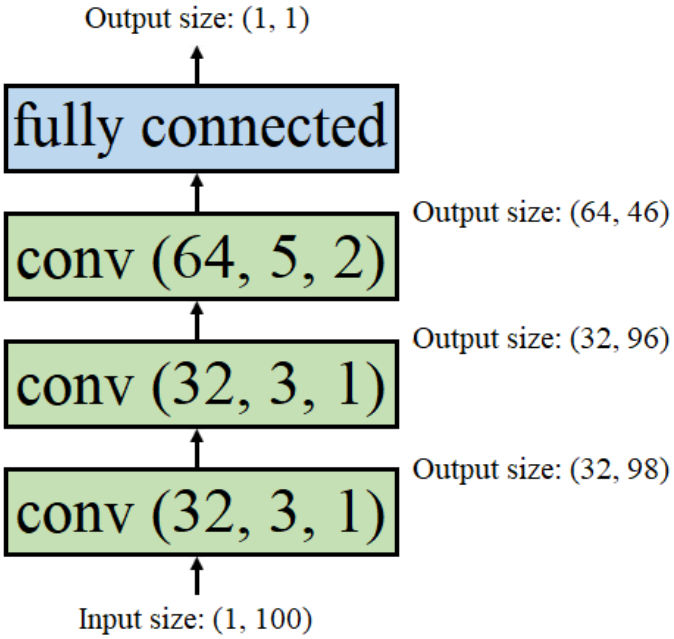


Figura 4.5: Architettura del discriminator. conv(32, 3, 1) denota un layer convolutivo con 32 filtri, ciascuno di dimensione 3 e stride 1. Ogni layer è seguito da una operazione di ReLU.

4.1.3 Loss

Nel *machine learning*, la *loss* è quel valore che stabilisce quando un singolo esempio prodotto si discosti dal valore desiderato e che viene utilizzato dal processo di *backpropagation* per modificare i pesi della rete. Essendo la grandezza che l'ottimizzatore cerca di diminuire, è di grande importanza ed è quella su cui è basato tutto il training della rete.

Come già anticipato nella sezione 3.4.3, la *loss* che si è implementato, in particolare per il training del *generator*, combina una parte *supervised* (*mean squared error* o MSE) con una *adversarial*. La *loss* del *discriminator*, tuttavia, non viene modificata. Le formule che descrivono le due *objective function* sono quindi:

$$\begin{aligned}
\max_D V(D,G) &= \mathbb{E}_{x\sim p_{data_x}(x)}[\log D(x)] + \mathbb{E}_{z\sim p_z(z),i,o\sim p_{data_{i,o}}(i,o)}[\log(1-D(G(z,i)))] \\
\max_G V(D,G) &= \alpha\mathbb{E}_{z\sim p_z(z),i,o\sim p_{data_{i,o}}(i,o)}[\log D(G(z,i))] + \\
&\quad -(1-\alpha)\mathbb{E}_{z\sim p_z(z),i,o\sim p_{data_{i,o}}(i,o)} \parallel o - G(z,i) \parallel_2
\end{aligned} \tag{4.1}$$

Dove $\alpha \in [0,1]$ rappresenta la frazione di *loss* totale attribuita alla *loss adversarial*.

Durante il processo di *hyperparameter tuning* ci siamo trovati a regolare anche il parametro α di funzione (4.1). I risultati sono mostrati nella figura 4.6.

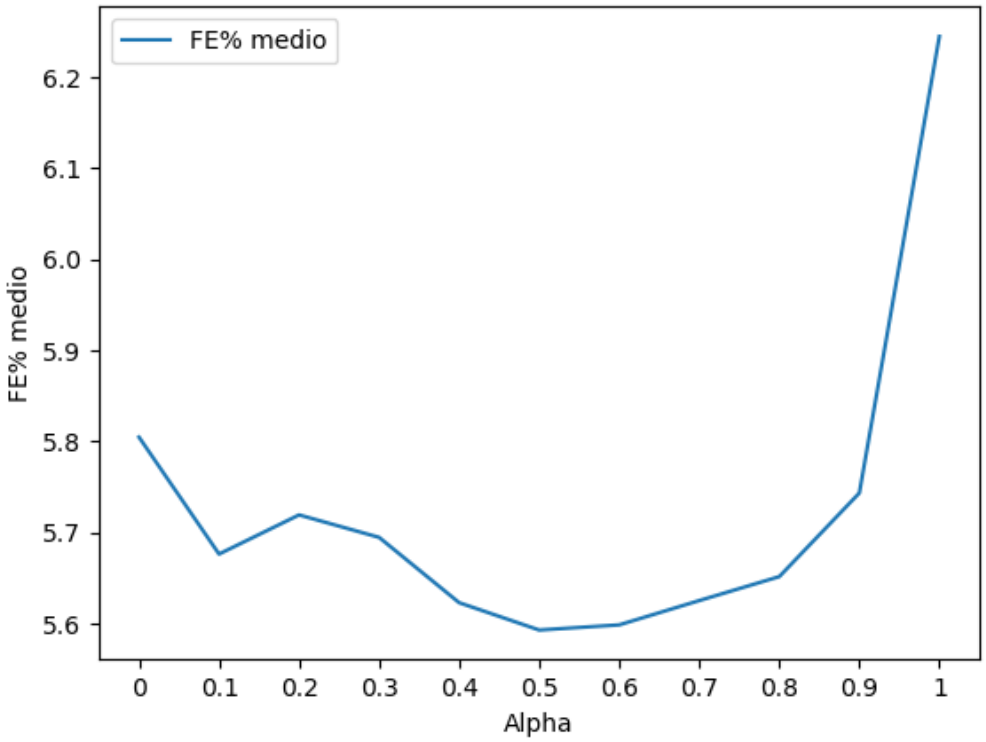


Figura 4.6: Tuning del parametro α. Lungo l'asse x si ha il valore di α, mentre l'asse y rappresenta il valore della metrica utilizzata ottenuto in diverse prove (si noti che la metrica è un errore, quindi più è piccola e meglio è).

Nel grafico mostrato, per $\alpha = 1$ si ha una *loss* completamente *adversarial*, e per $\alpha = 0$ una *loss* completamente *supervised*. Il valore ottimale dell'iperparametro tuttavia è 0,5, che corrisponde a un peso equivalente per entrambe le *loss*. Quello che si è provato in questa fase è che, per il problema analizzato, l'inserimento di una parte di *loss adversarial* funge da regolarizzatore per la rete. Gli esempi prodotti dalla rete non sono solamente più realistici, ma risultano anche più "vicini" a quelli originali rispetto a una rete con *objective* puramente *supervised*.

4.2 Librerie e frameworks

Invece che progettare e realizzare un sistema di *machine learning* da zero, quello che spesso si preferisce è utilizzare alcune librerie o frameworks come supporto alla programmazione del codice. I benefici che derivano dall'utilizzo di un framework sono:

- Codice più facile da gestire da un punto di vista di *software engineering*, in quanto più affidabile, *debuggabile* e robusto.
- Prodotti più facilmente scalabili a grandi dataset, con la possibilità di girare su hardware eterogeneo.
- Maggiore accessibilità anche a chi non è esperto di *machine learning*, programmazione, matematica e *parallel computing*.

In questa sezione vengono mostrate le librerie utilizzate per modellare le reti durante lo sviluppo della tesi.

4.2.1 Tensorflow

Creato dal team di Google Brain, TensorFlow [1] è una libreria *open source* per il calcolo numerico e per il *machine learning* su larga scala. TensorFlow utilizza Python per fornire una comoda API per la creazione di applicazioni e modelli, mentre esegue le operazioni interne nel più efficiente linguaggio C++. Questa libreria permette agli sviluppatori di creare dei grafi attraverso i quali far scorrere i dati. Ogni nodo del grafo rappresenta una operazione matematica, che viene eseguita in maniera ottimizzata per avere minor tempo di esecuzione e utilizzo di memoria possibile. Questa struttura, inoltre, facilita anche il calcolo delle derivate, di essenziale importanza nel processo di *backpropagation*. La presenza di un grafo permette al framework di mappare la computazione in modo da sfruttare al massimo architetture come le GPU (*Graphical Processing Unit*), portando un importante *speedup* complessivo.

4.2.2 Keras

Keras [4] è una libreria *open source* di alto livello scritta in Python, per la progettazione e implementazione di reti neurali. È stata sviluppata con il principale scopo di consentire una rapida sperimentazione di modelli. Utilizza software di terze parti (TensorFlow, CNTK e Theano) come *backend*, ovvero utilizza le funzionalità di queste librerie per fornire un'interfaccia di programmazione semplice e intuitiva, in maniera totalmente trasparente al programmatore.

Tutti i modelli sviluppati per il progetto legato alla tesi sono stati implementati utilizzando Keras, con TensorFlow come backend.

Capitolo 5

Risultati

In questo capitolo vengono mostrati tutti i test e i risultati ottenuti. Per garantire la riproducibilità dei risultati ottenuti, nella sezione 5.1 si è descritto dettagliatamente come è stato costruito il dataset, a partire dal metodo con il quale si sono calcolati gli indicatori tecnici fino ad arrivare al processo di creazione dei singoli esempi. Sempre con l'intento di assicurare la riproducibilità delle prove effettuate, in sezione 5.2 si è mostrata la metodologia utilizzata per il training delle reti e i valori di tutti gli iperparametri riguardanti il training e le due reti. In sezione 5.3 si introducono le baseline utilizzate per la valutazione del modello.

Una volta presentate tutte le informazioni utili per il setup delle prove, in sezione 5.4 vengono confrontate le performance della rete e delle baseline. Infine, nella sezione 5.5 si mostrano svariati esempi di scenari prodotti dalla rete, e l'analisi di come le due reti migliorino durante il training.

5.1 Dataset

Il dataset è parte fondamentale del progetto di *machine learning*, e per dataset si intende l'insieme di dati che vengono mostrati all'algoritmo in fase di training, che sono utilizzati per l'*hyperparameter tuning* in fase di *validation* e infine quelli usati in fase di test. Il reperimento, la gestione e suddivisione dei dati hanno un profondo impatto non solo su quello che il modello imparerà, ma anche sulla effettiva facilità che il modello avrà di imparare. È opportuno quindi fare molta attenzione alla fase di preparazione dei dati.

Nella sezione 5.1.1 verranno descritti gli indicatori tecnici scelti per affiancare il prezzo nell'insieme di informazioni riguardanti l'andamento del titolo nei giorni precedenti alla previsione. Nella sezione 5.1.2, invece, verrà spiegato in dettaglio come si è gestito l'intero dataset, dividendolo in *training*, *validation* e *test set* e suddividendo ognuno di questi insiemi a sua volta in sottoserie da proporre in ingresso alla rete.

5.1.1 Indicatori tecnici

Come anticipato in sezione 3.1, oltre al valore del prezzo dell'indice analizzato, si è deciso di calcolare il valore di 14 indicatori tecnici, per ognuno dei 70 giorni precedenti alla previsione. Alcuni di questi indicatori non sono altro che manipolazioni o misure aggregate del valore del prezzo nei giorni precedenti e si possono identificare come *features* create appositamente per l'analisi tecnica. Altri, invece, sono valori aggiuntivi, non strettamente correlati al prezzo nei giorni precedenti. In generale, tutti forniscono informazioni aggiuntive utili al *generator* per generare in maniera più accurata i possibili scenari futuri. Gli indicatori tecnici scelti sono:

1. *Price open*: il prezzo di apertura è il prezzo a cui un certo indice viene scambiato al momento dell'apertura del mercato. Dipende certamente dal prezzo del giorno precedente, ma anche dagli ordini di acquisto ricevuti prima dell'apertura del mercato.

2. *Price low*: il prezzo minimo è il valore minimo raggiunto dal prezzo all'interno del giorno analizzato.

3. *Price high*: il prezzo massimo è il valore massimo raggiunto dal prezzo all'interno del giorno analizzato.

4. *Price close*: il prezzo di chiusura è l'ultimo prezzo di un prodotto negoziabile nel momento in cui il mercato chiude. Rappresenta la valutazione più aggiornata di un indice prima che le negoziazioni ricomincino il giorno successivo.

5. *Volume*: rappresenta la quantità di azioni trattate durante il giorno, ed è un indicatore di liquidità dell'indice. Più è elevato il volume degli scambi e più è liquido lo strumento.

6. *Chaikin A/D Oscillator* (5, 10): questo indicatore finanziario si basa sul momento di Accumulation/Distribution (AD). L'AD calcola la posizione dell'indice analizzato a partire dal prezzo di chiusura come frazione della gamma giornaliera moltiplicando per il volume. Espresso matematicamente si ha:

$$AD = cum\left(\frac{(C-L)-(H-C)}{(H-L)}V\right) \tag{5.1}$$

   Dove *cum* indica la somma cumulativa totale, C è il prezzo di chiusura, H è il prezzo massimo, L è il prezzo minimo e V è il volume giornaliero. L'indicatore viene quindi calcolato dalla differenza tra la media mobile esponenziale (o *Exponential Moving Average*, abbreviato EMA) a 5 giorni e l'EMA a 10 giorni dell'AD. Questo indicatore permette di individuare accumulazioni e distribuzioni all'interno dell'ultimo periodo di tempo, e dà la possibilità di osservare il flusso di liquidità in ingresso e in uscita dal titolo.

7. *Chaikin A/D Oscillator* (10, 20): corrisponde all'indicatore descritto nel punto precedente, ma calcolato dalla differenza tra l'EMA a 10 giorni e l'EMA a 20 giorni dell'AD.

8. *Average Directional Movement Index Rating* (5): questo indicatore finanziario (ADXR) quantifica il momento dell'*average directional movement index* (ADX). L'ADX è un altro indicatore che rappresenta la forza e robustezza dell'eventuale trend in atto. La formula che esprime matematicamente l'ADX è:

$$ADX(t) = \frac{13ADX(t-1) + DX(t)}{14} \tag{5.2}$$

Dove:

$$\begin{aligned} DX(t) &= 100\ \frac{PDI(t) - MDI(t)}{PDI(t) + MDI(t)} \\ PDI(t) &= max(H(t) - H(t-1), 0) \\ MDI(t) &= max(L(t) - L(t-1), 0) \end{aligned} \tag{5.3}$$

Con max che corrisponde all'operatore di massimo, $H(t)$ al prezzo massimo del giorno t e $L(t)$ al prezzo minimo al giorno t. L'indicatore in esame corrisponde alla media tra l'ADX al giorno corrente e l'ADX di 5 giorni prima. La misura che ne deriva è quindi una versione più *smooth* dell'ADX.

9. *Average Directional Movement Index Rating* (10): corrisponde all'indicatore descritto nel punto precedente, ma calcolato facendo la media tra l'ADX al giorno corrente e l'ADX di 10 giorni prima

10. *Money Flow Index* (5): il MFI è un indicatore che utilizza sia il prezzo che il volume per misurare la pressione alla vendita o all'acquisto del titolo analizzato. Risulterà quindi positivo quando il trend è di crescita e negativo quando il trend è di declino. Il valore ottenuto viene poi normalizzato per andare da 0 a 100. La formula, in particolare, è:

$$MFI = 100 - \frac{100}{1 + \frac{\sum_{i=t-n}^{t} PMF(i)}{\sum_{i=t-n}^{t} NMF(i)}} \tag{5.4}$$

Dove:

$$\begin{aligned} PMF(t) &= \begin{cases} MF(t), & \text{if } MF(t) > MF(t-1) \\ 0, & otherwise \end{cases} \\ NMF(t) &= \begin{cases} MF(t), & \text{if } MF(t) \leq MF(t-1) \\ 0, & otherwise \end{cases} \\ MF(t) &= \frac{H(t) + L(t) + C(t)}{3}\ V(t) \end{aligned} \tag{5.5}$$

Con $n = 5$ che corrisponde al periodo analizzato in giorni, $H(t)$, $L(t)$, $C(t)$ e $V(t)$ rispettivamente al prezzo massimo, minimo, di chiusura e al volume di azioni trattate nel giorno t.

11. *Money Flow Index* (10): corrisponde all'indicatore descritto nel punto precedente, ma calcolato nel caso in cui il periodo analizzato $n = 10$.

12. *Absolute Price Oscillator* (5, 10): questo indicatore finanziario mostra la differenza in valore assoluto tra due EMA del prezzo del titolo in analisi, una a 5 giorni e l'altra a 10 giorni. Utilizzando medie mobili, l'indicatore elimina i trend di lungo e corto termine e mostra efficacemente l'eccesso di acquisto e vendita a medio termine.

13. *Absolute Price Oscillator* (10, 20): corrisponde all'indicatore descritto nel punto precedente, ma calcolato tra l'EMA a 10 giorni e l'EMA a 20 giorni del prezzo del titolo in analisi.

14. *Aroon Oscillator* (5):è un indicatore che, come l'ADXR, ha come obiettivo quello di misurare la forza del trend attuale e la probabilità che questo continui. Per fare ciò utilizza le misure intermedie *Aroon Up* (AU) e *Aroon Down* (AD), che valutano il trend in base al numero di giorni passati dall'ultimo massimo o minimo della serie e che sommano sempre a 100.

$$\begin{aligned} AU &= 100\ \frac{5 - d_M}{5} \\ AD &= 100\ \frac{5 - d_m}{5} \end{aligned} \tag{5.6}$$

Dove $d_M \in [0, 5]$ e $d_m \in [0, 5]$ rappresentano il numero di giorni che separano quello corrente rispettivamente dal massimo e il dal minimo degli ultimi 5 valori. L'AO è semplicemente la differenza tra AU e AD.

5.1.2 Split, overlap e normalizzazione

Nel *machine learning*, si costruiscono algoritmi che compiono predizioni e prendono decisioni guidate dai dati presentati come esempi, dei quali costruiscono un modello matematico. Invece che proporre tutti i dati a propria disposizione come input per allenare il modello, spesso si sceglie di dividere il dataset in 3 parti: il *training set*, il *validation set* e il *test set*. Il *training set* è il dataset che si propone ripetutamente al modello, e che questo utilizza per modificare i propri parametri attraverso il processo di *backpropagation*. Quando il modello è allenato, gli si presenta il *validation set*, ovvero una serie nuova di dati. Tutti gli iperparametri vengono regolati per ottimizzare le *performance* sul *validation set*. Per evitare che il procedimento abbia favorito gli iperparametri che danno buoni risultati solo sul *validation set*, e non i migliori in assoluto, si valuta la capacità del modello migliore nel *validation set* sul *test set*, un ulteriore dataset che si usa solo sul modello finale. Il *test set* viene utilizzato anche per operare confronti tra diverse architetture e modelli.

I dati a disposizione vanno dal 31/12/1999 al 31/04/2018, fornendo quindi in totale informazioni per esattamente 4758 giorni. Si noti che i dati non sono presenti le giornate di sabato, domenica e festivi, in quanto la borsa in quei giorni è chiusa. Essendo il periodo considerato lo stesso per tutti i 4 diversi indici, coincidono anche le dimensionalità dei dati, che risultano uguali indipendentemente dal fatto che il dataset in analisi sia il FTSE MIB,

il FTSE 100, il DAX30 o lo S&P 500. Per ognuno dei 4 indici si ha, quindi, una tabella di dimensioni 15×4758 di dati *raw*, ovvero dati ancora da elaborare prima di poterli dare in input al modello.

La prima fase consiste nell'elaborazione dei dati per renderli più facilmente gestibili dal modello. Per quanto riguarda il solo prezzo, si esegue un'operazione di differenziazione finita (FD), ovvero si sostituisce ogni valore con la differenza tra lo stesso e il valore precedente:

$$p(t) = \begin{cases} 0, & \text{if } t = 0 \\ p(t) - p(t-1), & \text{if } t \neq 0 \end{cases} \tag{5.7}$$

Di seguito è mostrato un esempio per meglio capire come l'operatore applicato cambia i dati. Nel caso dei primi 5 valori del FTSE MIB si ha:

$$FD([41226, 40314, 39452, 38835, 40194]) = [0, -912, -862, -617, 1359] \tag{5.8}$$

Come anticipato anche nella sezione 3.3.1, nel training delle GAN invece che fornire in ingresso i dati originali, spesso si preferisce fornirne una versione normalizzata. Per normalizzazione si intende quel processo che porta i dati in ingresso ad avere media $\mu = 0$ e deviazione standard $\sigma = 1$. Per fare ciò, si calcola μ e σ per righe, si sottrae μ a ogni elemento e lo si divide per σ. La formula per aggiornare i valori è:

$$x_{i,j} \leftarrow \frac{x_{i,j} - \mu_i}{\sigma_i} \tag{5.9}$$

Dove:

$$\begin{aligned} \mu_i &= \frac{1}{N} \sum_{j=0}^{N-1} x_{i,j} \\ \sigma_i &= \sqrt{\frac{1}{N} \sum_{j=0}^{N-1} (x_{i,j} - \mu_i)^2} \end{aligned} \tag{5.10}$$

Questa scelta, che viene spesso presa nell'ambito del *machine learning*, serve sostanzialmente a facilitare l'apprendimento del modello poichè rende la fase di training meno sensibile alla scala delle diverse *features*. Introducendo il vincolo di avere tutte media $\mu = 0$ e deviazione standard $\sigma = 1$, la scala non inciderà più in alcun modo in quanto sarà completamente uniformata tra tutte le *features*.

Dopo aver elaborato il dataset, è opportuno suddividerlo in *training set*, *validation set* e *test set*. La suddivisione dei dati da assegnare a ogni dataset è una scelta progettuale. Spesso quello che si tende a fare è di estendere il *training set* il più possibile, poiché un maggior numero di esempi in fase di allenamento permettono di creare modelli più grandi, oltre che favorire una corretta generalizzazione. Tuttavia, è opportuno evitare di limitare *validation* e *test set* in maniera eccessiva. In tal caso, infatti, si rischierebbe di rendere poco significativa la valutazione di questi. La suddivisione che si è adottata è quindi quella di 80% per il *training set* e 10% sia per il *validation* che per il *test set*. Una rappresentazione grafica di questa divisione è mostrata in figura 5.1.

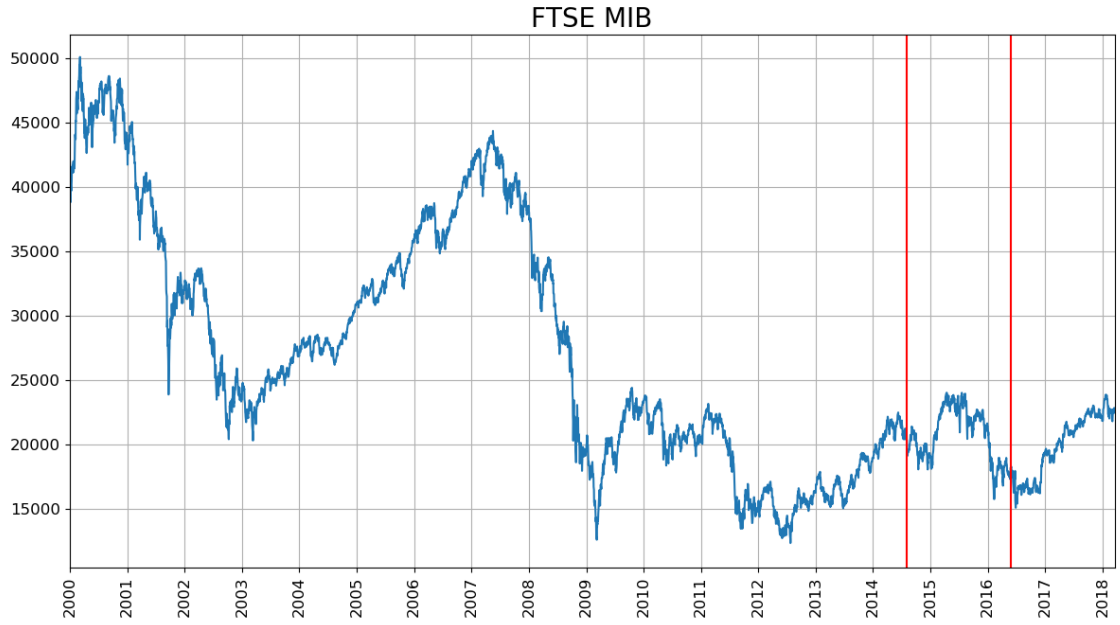


Figura 5.1: Andamento del FTSE MIB nel periodo a disposizione. Le linee in rosso caratterizzano la suddivisione tra training e validation set (quella a sinistra) e tra validation e test set (quella a destra).

La tabella di dimensioni 15×4758 viene quindi suddivisa in tre: una di dimensioni 15×3806 per il *training* e due di dimensioni 15×476 per *validation* e *test*.

Nell'ultima fase, ci si è occupati di suddividere le macroserie di *training*, *validation* e *test* create nelle fasi precedenti in serie di dimensioni 100, da poter dare al modello in input. La soluzione più semplice e intuitiva è quella di dividerle in serie consecutive, senza intersezione. Sfortunatamente, questa soluzione non è applicabile, in quanto si otterrebbe un numero troppo ridotto di serie. Per esempio, utilizzando questa strategia si otterrebbero solamente 4 serie complete per il *validation*. La soluzione adottata è quella di utilizzare una certa percentuale $\alpha\%$ di *overlap*. In questo modo, la serie i-esima è formata per il $\alpha\%$ dai valori della serie $(i-1)$-esima e per il restante da valori nuovi. Un esempio esplicativo del funzionamento dell'*overlap* è mostrato in figura 5.2. Oltre ad essere utile per rendere il numero di esempi accettabile, l'introduzione dell'*overlap* trova giustificazione anche nel merito di *data augmentation*, una pratica molto diffusa nel *machine learning*. Inserire una certa quantità di ridondanza nei dati di *training* si è scoperto spesso non essere deleterio ma utile, in quanto funge da regolarizzatore e aumenta il grado di generalizzazione. Anche in questo caso la scelta del parametro α è progettuale. Si cerca di massimizzarne il valore per aumentare le dimensioni del dataset, ma un valore troppo alto introdurrebbe un'eccessiva correlazione dei dati in ingresso. Il valore che è stato scelto per α è quindi 95%.

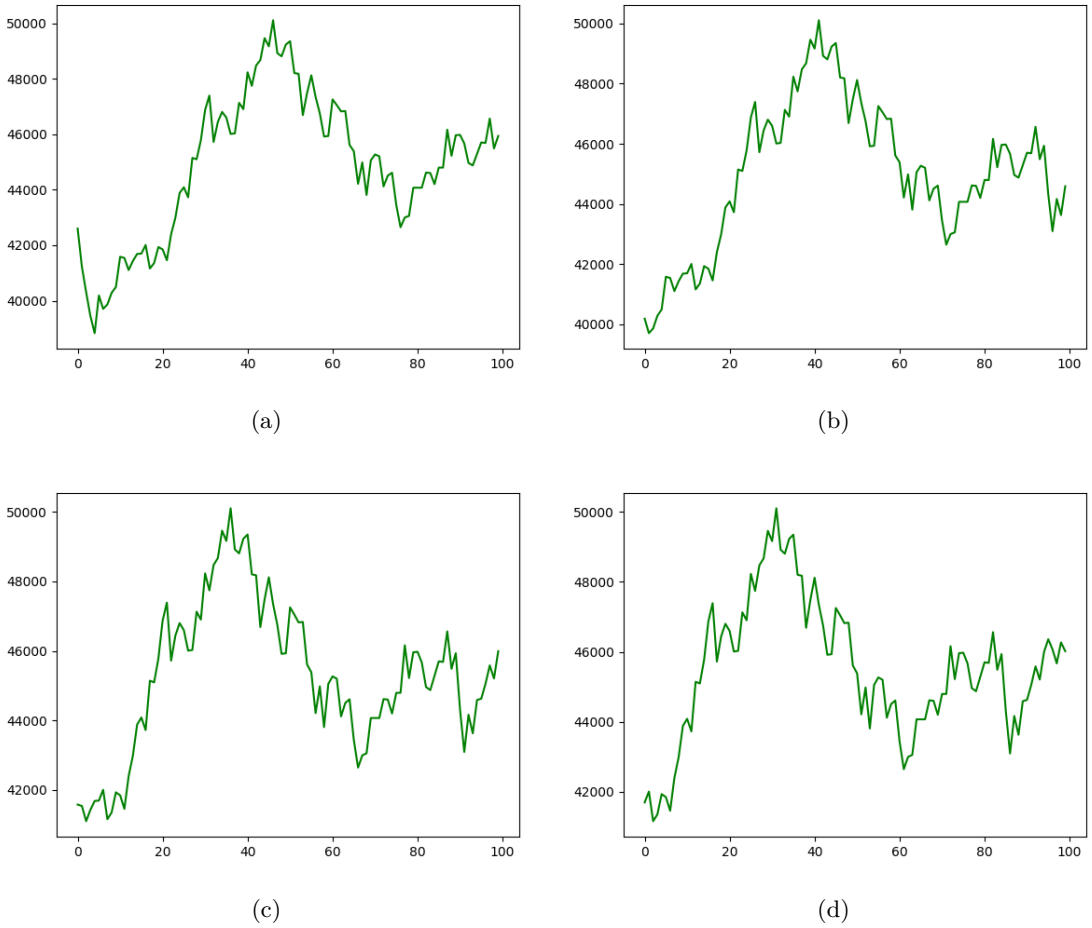


Figura 5.2: I grafici dell'andamento del FTSE MIB per le prime 4 serie, con overlap al 95%.

Tutte le dimensionalità prese in considerazione vengono riassunte nella seguente tabella:

	Dimension
Raw data	15×4758
Raw training data	15×3806
Raw validation data	15×476
Raw test data	15×476
Dataset training	$742 \times 15 \times 100$
Dataset validation	$76 \times 15 \times 100$
Dataset test	$76 \times 15 \times 100$

Tabella 5.1: Tabella delle dimensionalità del dataset

5.2 Training e parametri di training

La fase di training è la fase più importante della costruzione del modello, in quanto è quella nella quale si modificano i parametri della rete per adattarli ai dati presentati in input, ed è anche quella computazionalmente più onerosa. Quello che spesso si fa per allenare una rete neurale è suddividere tutto il dataset in gruppi di esempi di dimensione fissa, chiamati *mini-batch*. Si propone un gruppo alla volta alla rete, e per ognuno si calcola la *loss* e si utilizza l'ottimizzatore per aggiornare i pesi. Quando si è mostrato in input tutto il dataset a disposizione, si dice che si è terminata un'epoca. Se non si è implementata una qualche forma di *early stopping*, dopo un certo numero di epoche l'algoritmo si ferma, e la rete è considerata allenata.

Tuttavia, il metodo con cui si allenano le GAN è un po' diverso, e viene mostrato in sezione 5.2.1. Tutti gli iperparametri per la costruzione del dataset, dei modelli e per la regolazione del training sono elencati nella sezione 5.2.2.

5.2.1 Processo di training

Dato che all'interno della GAN vi sono due reti e che queste reti hanno due *loss* distinte, non è possibile allenarle in maniera completamente *end-to-end*, come si fa di solito con le altre reti neurali. È, quindi, opportuno allenare le due reti separatamente. Data la dimensione del *mini-batch* n, gli step che vengono eseguiti ripetutamente sono:

- Il *discriminator* D viene allenato costruendo il *mini-batch* a partire da $\frac{n}{2}$ esempi presi a caso dal dataset e altri $\frac{n}{2}$ esempi sempre presi a caso dal dataset, ma nei quali il *generator* ha completato gli ultimi 30 valori della serie. Ai primi viene data una label 1 e ai secondi 0. In base a questi dati in input e output si è allenato D.

- Il *generator* G viene allenato in maniera *end-to-end*. Si disabilita la modifica dei pesi di D, e si fa generare a G la porzione finale di n serie prese a caso dal dataset. A tutte viene data una label di 1, poiché in tal modo G cercherà di modificare i propri pesi per rendere l'output di D vicino a 1, ingannandolo. La *loss* (*adversarial*) viene quindi propagata attraverso D senza modificarne i pesi e riversata su G. La *loss supervised*, invece, è calcolata semplicemente tra le serie generate da G e le serie originali.

Invece che chiamare un'epoca una passata completa del dataset, si identifica come epoca la singola coppia di *mini-batch* presentata a G e D.

Si noti che il metodo con cui si è allenato le due reti è coerente con l'*objective* mostrato nella sezione 4.1.3.

5.2.2 Iperparametri

Come anticipato nella sezione 3.3, per alcuni iperparametri sono stati scelti valori di default poiché generalmente questi funzionano bene in tutti i tipi di reti, per alcuni si è deciso il valore in base a scelte progettuali e altri invece sono stati esplorati e il loro valore è stato scelto in base a quello che ha portato migliori risultati nel *validation set*. In questa sezione

ci si occuperà di mostrare tutti gli iperparametri della rete con i relativi valori ottimali utilizzati, indipendentemente da come questi siano stati scelti.

Iperparametri globali

Gli iperparametri globali sono:

- *Batch size* = 32. Numero di esempi presenti in un singolo step di training, ovvero la dimensione del *mini-batch* citato in sezione 5.2.1.
- *Early stopping max epochs* = 2000. Numero massimo di epoche dell'algoritmo di training.
- *Early stopping min epochs* = 150. Numero minimo di epoche dell'algoritmo di training.
- *Early stopping patience* = 10. Numero di epoche consecutive per cui si deve presentare la condizione di terminazione prima di terminare effettivamente l'algoritmo. Ulteriori informazioni sul tipo di *early stopping* implementato sono nella sezione 3.4.2.
- *Early stopping treshold* = 15%. Differenza percentuale tra le *loss* del *generator* e del *discriminator* sotto la quale scatta la condizione di terminazione. Ulteriori informazioni sul tipo di *early stopping* implementato sono nella sezione 3.4.2.
- *Evaluate metric* = 'fe'. Errore finale %, metrica utilizzata per valutare le configurazioni durante l'*hyperparameter tuning*. In sezione 3.4.1 viene descritta dettagliatamente, insieme a tutte le possibili alternative analizzate.
- *Evaluate samples* = 500. Parametro che indica il numero di scenari n prodotto per ogni esempio. Invece che limitarsi a creare un singolo vettore di lunghezza 30, per valutare la rete su un singolo esempio le si fanno generare n scenari e ne si valuta l'andamento medio.
- *Noise mean* = 0. Media μ della gaussiana da cui è campionato il noise in input al *generator*.
- *Noise std* = 1. Deviazione standard σ della gaussiana da cui è campionato il noise in input al *generator*.

Iperparametri del dataset

Gli iperparametri del dataset sono:

- *Dataset feature number* = 15. Numero di *features* per ogni giorno che vengono mostrate alla rete come input. Una di queste è il prezzo dell'indice, le altre 14, invece, sono mostrate nella sezione 5.1.1.

- *Dataset overlap* = 95%. Parametro *overlap* della creazione del dataset, citato in sezione 5.1.2.
- *Dataset split* = [0.8, 0.1, 0.1]. Percentuali, di cui si è parlato in sezione 5.1.2, in cui viene suddiviso il dataset. Nel nostro caso implica che l'80% dei dati venga riservato per il *training set*, il 10% per il *validation set* e 10% per il *test set*.
- *Series given length* = 70. Numero di giorni dati in input al *generator* per effettuare la previsione.
- *Series missing length* = 30. Numero di giorni generati dal *generator*.

Iperparametri del generator

Gli iperparametri del generator sono:

- *Adversarial weight* = 0.5. Peso della parte adversarial della loss, che coincide col parametro α di formula (4.1).
- *Alpha* = 0.2. Coefficiente angolare della parte negativa di tutte le Leaky ReLU del *generator*, ovvero il parametro α di formula (2.7).
- *Decoder bias initializer* = 'zeros'. Inizializzatore per il vettore di bias del *decoder*. Implica che tutti i bias della rete neurale *fully connected* vengano inizializzati a 0.
- *Decoder hidden layer size* = 64. Dimensione dell'*hidden layer* della rete *fully connected* del *decoder*.
- *Decoder kernel initializer* = 'glorot uniform' [9]. Inizializzatore per la matrice di pesi del *decoder*. Implica che tutti i pesi della rete neurale *fully connected* vengano inizializzati campionando da una distribuzione uniforme in [-l, l]:

  $$f(x) = \begin{cases} \frac{1}{2l}, & \text{if } -l \leq x \leq l \\ 0, & otherwise \end{cases} \tag{5.11}$$

  Con:

  $$l = \sqrt{\frac{6}{n_{out} + n_{in}}} \tag{5.12}$$

  Dove n_{out} è il numero di output e n_{in} il numero di input del particolare layer che si sta inizializzando.
- *Encoder activation* = 'tanh'. Funzione di attivazione utilizzata per i valori di stato e output dell'unità LSTM dell'*encoder*. L'espressione matematica della TanH è mostrata nell'equazione (2.9).
- *Encoder bias initializer* = 'zeros'. Inizializzatore per i vettori di bias dell'unità LSTM dell'*encoder*. Implica che tutti i bias della rete vengano inizializzati a 0.

- *Encoder dropout* = 0.0. Parametro del regolarizzatore *dropout*, ovvero la frazione di neuroni da spegnere per la trasformazione dell'input. Se posto a 0 implica che non ci sia alcun *dropout*.

- *Encoder kernel initializer* = 'glorot uniform' [9]. Inizializzatore per le matrici di pesi dell'unità LSTM dell'*encoder*. Implica che tutti i pesi delle reti vengano inizializzati campionando da una distribuzione uniforme in [-l, l], come in formula (5.11).

- *Encoder recurrent activation* = 'hard sigmoid'. Funzione di attivazione utilizzata per tutti i *gate* dell'unità LSTM dell'*encoder*. È una versione lineare della sigmoide, che ha formula (2.8), e viene utilizzata per velocizzare le operazioni in quanto non necessita alcun calcolo esponenziale.

- *Encoder recurrent dropout* = 0.0. Parametro del regolarizzatore *dropout*, ovvero la frazione di neuroni da spegnere per la trasformazione dell'input nelle uscite dei vari *gate* dell'unità LSTM dell'*encoder*. Se posto a 0 implica che non ci sia alcun *dropout*.

- *Encoder states* = 64. Dimensione dello stato e dell'output dell'unità LSTM dell'*encoder*

- *Noise handling* = 'add'. Metodo con cui si inserisce il noise nel modello del *generator*. Le possibili alternative sono 'add', ovvero sommandolo, e 'mul' ovvero moltiplicandolo. Una descrizione più accurata dei due metodi è presente nella sezione 3.4.3.

- *Optimizer* = 'adam'. Metodo di ottimizzazione utilizzato per aggiornare i pesi del *generator*.

- *Optimizer* $\beta_1 = 0.9$. Parametro β_1 dell'ottimizzatore Adam.

- *Optimizer* $\beta_2 = 0.999$. Parametro β_2 dell'ottimizzatore Adam.

- *Optimizer decay* = 0.0. Tasso di decadimento del *learning rate* per ogni aggiornamento. Se posto a 0 implica che non ci sia alcun decadimento.

- *Optimizer learning rate* = 0.0002. *Learning rate* iniziale dell'ottimizzatore Adam. Nel paper originale [14] è indicato con la lettera α, ma non è raro trovarlo indicato anche con la lettera η.

Iperparametri del discriminator

Gli iperparametri del discriminator sono:

- *Alpha* = 0.2. Coefficiente angolare della parte negativa di tutte le Leaky ReLU del *discriminator*, ovvero il parametro α di formula (2.7).

- *Convolution bias initializer* = 'zeros'. Inizializzatore per i vettori di bias di tutti i layer della rete convoluzionale. Implica che tutti i bias vengano inizializzati a 0.

- *Convolution filters* = [32, 32, 64]. Numero di filtri dei layer della rete convoluzionale. La notazione a lista indica che il primo layer ha 32 filtri, il secondo 32 e il terzo 64, come mostrato in figura 4.5.
- *Convolution filters size* = [3, 3, 5]. Dimensione dei filtri della rete convoluzionale. La notazione a lista indica che il primo layer ha filtri di dimensione 3, il secondo di dimensione 3 e il terzo di dimensione 5, come mostrato in figura 4.5.
- *Convolution kernel initializer* = 'glorot uniform' [9]. Inizializzatore per le matrici di pesi dei vari layer convolutivi. Implica che tutti i pesi vengano inizializzati campionando da una distribuzione uniforme in [-l, l], come in formula (5.11).
- *Convolution padding* = 'valid'. Tipo di *padding* dei layer della rete. Il termine 'valid' indica sostanzialmente un'assenza di *padding*.
- *Convolution strides* = [1, 1, 2]. Stride operato nei layer della rete convoluzionale. La notazione a lista indica che il primo e il secondo layer hanno *stride* 1 (ovvero *no stride*), il terzo invece ha *stride* 2, come mostrato in figura 4.5.
- *Optimizer* = 'adam'. Metodo di ottimizzazione utilizzato per aggiornare i pesi del *discriminator*.
- *Optimizer* $\beta_1 = 0.9$. Parametro β_1 dell'ottimizzatore Adam.
- *Optimizer* $\beta_2 = 0.999$. Parametro β_2 dell'ottimizzatore Adam.
- *Optimizer decay* = 0.0. Tasso di decadimento del *learning rate* per ogni aggiornamento. Se posto a 0 implica che non ci sia alcun decadimento.
- *Optimizer learning rate* = 0.0002. *Learning rate* iniziale dell'ottimizzatore Adam. Nel paper originale [14] è indicato con la lettera α, ma non è raro trovarlo indicato anche con la lettera η.

5.3 Baseline

Nel *machine learning*, per *baseline* si intende un metodo che utilizza euristiche, statistiche o semplici reti neurali per risolvere lo stesso problema che si sta analizzando. La *baseline* viene utilizzata principalmente come base per il confronto con il modello creato, che spesso risulta essere più complesso.

Quando al termine della creazione e allenamento del modello con la configurazione di iperparametri migliori si testano le sue prestazioni sul *test set*, spesso non è facile interpretare il valore risultante, poiché la metrica potrebbe essere difficilmente comprensibile o potremmo avere a che fare con problemi intrinsecamente complessi. Se nel nostro caso, per esempio, le performance del modello sul *test set* dovessero risultare *average* $fe\% = 5\%$, non avremmo alcun modo di sapere se questi risultati siano buoni oppure no. La nostra rete ha appreso qualcosa di utile? Per rispondere a questa domanda si confronta questo

risultato con quello di altri modelli semplici per capire in confronto a questi, quanto sia più (o meno) utile.

Nel nostro caso, per *baseline* si intende un modello che produca una stima di valori del prezzo di un titolo azionario per 30 giorni, utilizzando i valori dell'indice nei 70 giorni precedenti o statistiche su questi. Per garantire una buona variabilità, si sono implementati diversi modelli di *baseline*: il modello martingala, due versioni del modello lineare e il modello *random walk*. Di seguito ognuno di questi verrà descritto dettagliatamente. Il confronto tra i risultati del modello implementato e quelli delle varie *baseline* è mostrato nella sezione 5.4.

5.3.1 Modello martingala

Data una variabile aleatoria $x(t)$ rappresentante il valore di un titolo azionario al tempo t, e data la sequenza $S = \{x(0), x(1), \dots\}$ di $x(t)$ discreta nel tempo, se il valore del titolo è modellabile con una martingala [35], allora avremo che:

$$\mathbb{E}(x(t)|x(1), x(2), ..., x(t-1)) = x(t-1) \tag{5.13}$$

Quindi, secondo il modello martingala, il miglior valore da poter dare al prezzo nello step successivo è semplicemente l'ultimo valore assunto disponibile. È opportuno notare come la modellazione in martingala delle sequenze di prezzi di indici azionari sia completamente coerente con l'ipotesi dei mercati efficienti e la teoria del *random walk*, già introdotte nella sezione 2.1.3.

Il modo con cui abbiamo modellato il martingala è assumendo per ognuno dei 30 valori futuri quello con maggiore probabilità, ovvero l'ultimo disponibile:

$$p(t) = p(69), t \in [70, 99] \tag{5.14}$$

Dove $p(t)$ è il prezzo del titolo nella serie analizzata al tempo t, con $t \in [0, 99]$. In figura 5.3 è possible vedere il comportamento del modello.

5.3.2 Modello lineare

Data una variabile aleatoria $x(t)$ rappresentante il valore di un titolo azionario al tempo t, e data la sequenza $S = x(0), x(1), \dots$ di $x(t)$ discreta nel tempo, se il valore del titolo è approssimabile ad un modello lineare, allora avremo che:

$$\mathbb{E}(x(t)|x(1), x(2), ..., x(t-1)) = x(t-1) + b \tag{5.15}$$

Dove b è il parametro che indica il coefficiente angolare della retta su cui giacciono i valori previsti. Il modello lineare è coerente con una versione modificata della teoria del *random walk*. Questa versione sostiene che invece che a *white noise*, il prezzo dei titoli azionario possa essere associato all'andamento di una *random walk with drift*.

Il modello verrà quindi assumerà i valori:

$$p(t) = p(69) + (t - 69)b, t \in [70, 99] \tag{5.16}$$

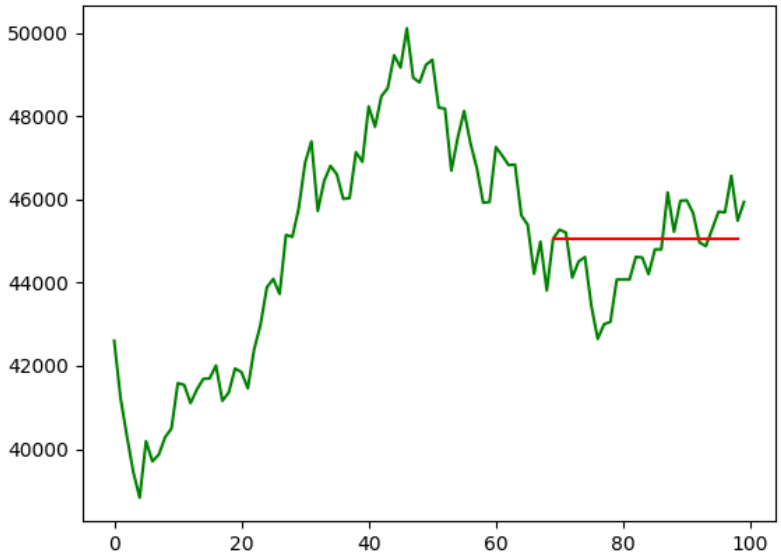


Figura 5.3: Grafico dell'output del modello martingala. La linea in verde rappresenta l'andamento reale del titolo per 100 giorni, la linea rossa indica il valore assunto dalle previsioni del modello martingala per gli ultimi 30 giorni.

Dove $p(t)$ è il prezzo del titolo nella serie analizzata al tempo t, con $t \in [0, 99]$. Di seguito si mostrano due tipi di modelli lineari, differenziabili dal modo in cui si è calcolato il parametro b.

Linearità locale

Per linearità locale si intende linearità il cui valore b è calcolato considerando solo la serie in analisi. I valori previsti dal modello quindi giaceranno sulla retta passante per il primo valore della serie disponibile (0) e per l'ultimo (69). Si ha quindi:

$$b = \frac{p(69) - p(0)}{69} \tag{5.17}$$

E la previsione del prezzo sarà quindi data da:

$$p(t) = p(69) + (t - 69)\frac{p(69) - p(0)}{69}, t \in [70, 99] \tag{5.18}$$

Dove $p(t)$ è il prezzo del titolo nella serie analizzata al tempo t, con $t \in [0, 99]$.

Questo modello è giustificato dal fatto che si stia cercando di proseguire il trend assunto dalla sequenza per i primi valori. In figura 5.4 viene mostrato un esempio di ouput.

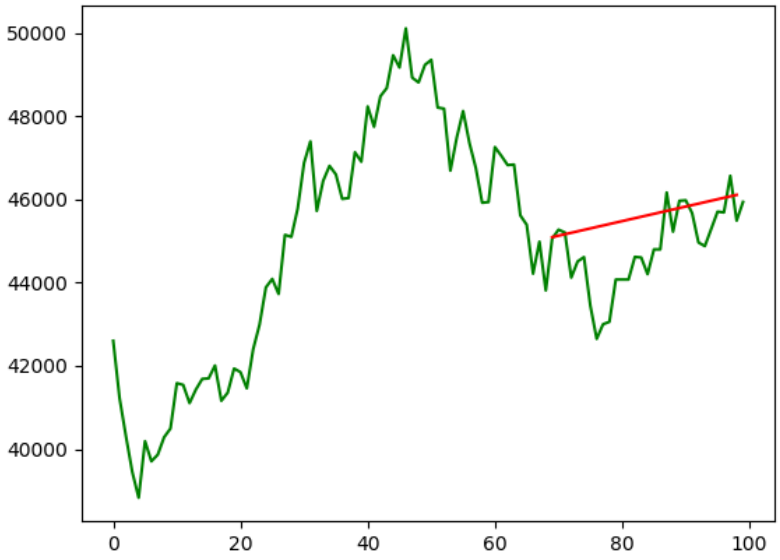


Figura 5.4: Grafico dell'output del modello lineare, con linearità locale. La linea in verde rappresenta l'andamento reale del titolo per 100 giorni, la linea rossa indica il valore assunto dalle previsioni del modello per gli ultimi 30 giorni.

Linearità globale

Per linearità globale si intende linearità il cui valore b è calcolato considerando tutta la serie azionaria a disposizione del titolo in questione, dal 31/12/1999 al 31/04/2018. I valori previsti dal modello quindi giaceranno sulla retta passante per il primo valore della serie in assoluto (0) e per l'ultimo in assoluto (4757). Si ha quindi:

$$b = \frac{p(4757) - p(0)}{4757} \tag{5.19}$$

E la previsione del prezzo sarà quindi data da:

$$p(t + t_0) = p(69 + t_0) + (t - 69)\frac{p(4757) - p(0)}{4757}, t \in [70, 99] \tag{5.20}$$

Dove $p(t)$ è il prezzo del titolo nella serie analizzata al tempo t, con $t \in [0, 4757]$, e t_0 è il punto d'inizio nella serie totale della serie correntemente analizzata. Dato che il periodo considerato è molto esteso, quello che si nota è che tipicamente il valore assoluto di b è molto piccolo. Le predizioni del modello lineare a linearità globale quindi sono molto simili a quelle del modello martingala. Tuttavia, possiamo considerarle come una versione leggermente migliorata, dato che considera anche l'andamento globale del titolo. A differenza del martingala, quindi, le previsioni del modello sono ripetitive all'interno del dataset di uno stesso titolo, ma variano a seconda di quale titolo si stia considerando. Un

esempio esplicativo è mostrato in figura 5.5. Nel periodo analizzato, il FTSE MIB (5.5a) è l'unico ad essere complessivamente calato, mentre il DAX 30 (5.5c) e l'S&P 500 (5.5d) sono cresciuti e il FTSE 100 (5.5b) è rimasto pressoché invariato.

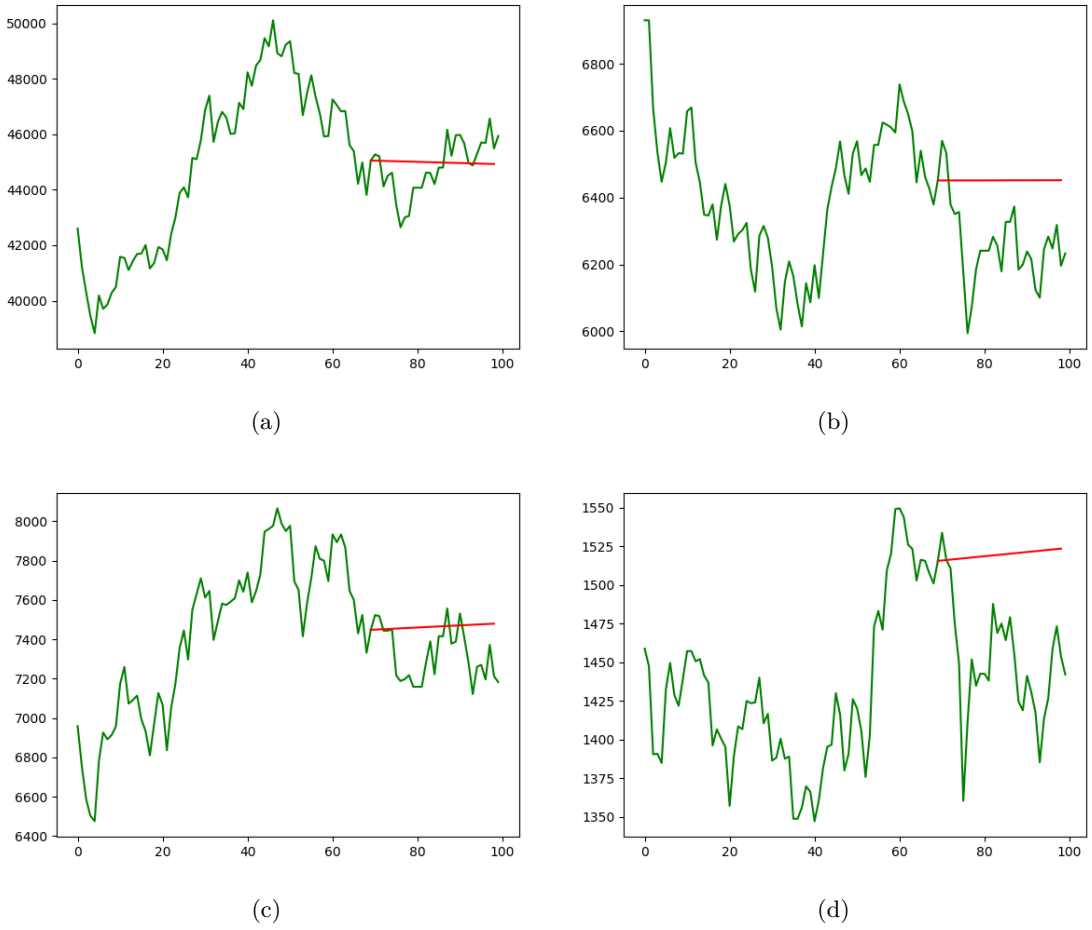


Figura 5.5: Grafico dell'output del modello lineare con linearità locale per la prima serie del FTSE MIB (a), FTSE 100 (b), DAX 30 (c) e S&P 500 (d). La linea in verde rappresenta l'andamento reale del titolo per 100 giorni, la linea rossa indica il valore assunto dalle previsioni del modello per gli ultimi 30 giorni.

5.3.3 Modello random walk

Il modello *random walk*, a differenza dei modelli precedentemente analizzati, è un modello che tenta di ricreare la serie futura campionando da una distribuzione, invece che seguire trend predefiniti. Dopo aver eseguito l'operazione di differenziazione finita sui primi 70 valori, come descritto nella sezione 5.1.2, di questi ne calcola la media μ e la deviazione

standard σ. A questo punto, per generare i successivi valori, non fa altro che campionare da una gaussiana con media μ e deviazione standard σ, precedentemente calcolate. L'assunzione maggiore che fa questo modello è che la distribuzione della differenza finita dei prezzi dei titoli azionari sia modellabile con una gaussiana. Un esempio di output del modello è mostrato in figura 5.6.

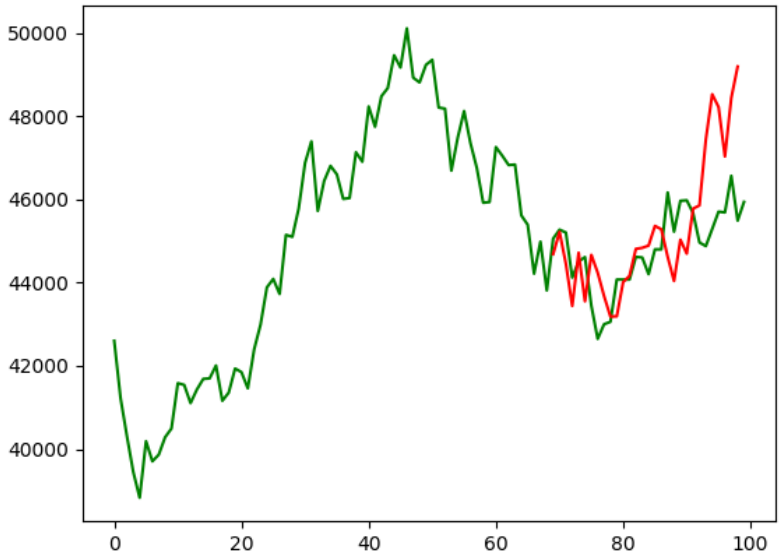


Figura 5.6: Grafico dell'output del modello random walk. La linea in verde rappresenta l'andamento reale del titolo per 100 giorni, la linea rossa indica il valore assunto dalle previsioni del modello per gli ultimi 30 giorni.

5.4 Risultati quantitativi

Per risultati quantitativi si intende tutti quei valori strettamente numerici che vengono estrapolati direttamente dal modello generato. Nel nostro contesto, come risultati quantitativi si considerano le performance del modello realizzato e delle *baseline* sul *test set*. Nelle analisi che seguiranno, le *baseline* che si sono considerate sono il modello martingala (o *Martingale Model*, MM), il modello a linearità locale (o *Locally Linear Model*, LLM), il modello a linearità globale (o *Globally Linear Model*, GLM) e il modello a *random walk* (o *Random Walk Model*, RWM). Il modello realizzato è definito dalla sigla GAN.

La prima analisi effettuata riguarda la metrica che ha guidato l'*hyperparameter tuning*, ovvero l'FE%, descritto in sezione 3.4.1. Tale metrica la si è calcolata per tutti gli esempi presenti nel *test set* di tutti i 4 dataset a disposizione (FTSE MIB, FTSE 100, DAX 30 e S&P 500), esaminando i valori generati da ogni baseline e quelli generati dal *generator*.

Come valore finale si è considero il valor medio degli FE% ottenuti. I risultati sono mostrati nella tabella 5.2.

FE %				
Model	FTSE MIB	FTSE 100	DAX 30	S&P 500
MM	4,00	2,78	3,50	2,56
LLM	4,33	3,35	3,91	2,42
GLM	4,24	2,78	**3,41**	2,36
RWM	5,67	4,28	5,13	2,95
GAN	**3,98**	**2,71**	3,51	**2,01**

Tabella 5.2: Confronto in termini di FE% del modello con le baseline, per tutti gli indici a disposizione

Nella seconda analisi quello che si è quantificato sono le prestazioni del modello valutando un'altra metrica presa in considerazione in sezione 3.4.1, ovvero l'FD%. Il processo di valutazione coincide esattamente con quello della prima analisi. I risultati sono mostrati nella tabella 5.3.

FD %				
Model	FTSE MIB	FTSE 100	DAX 30	S&P 500
MM	0,0	0,0	0,0	0,0
LLM	59,2	36,8	56,2	72,4
GLM	34,2	**53,9**	**64,5**	**78,9**
RWM	44,7	42,1	55,3	60,5
GAN	**60,5**	46,0	**64,5**	**78,9**

Tabella 5.3: Confronto in termini di FD% del modello con le baseline, per tutti gli indici a disposizione

I risultati ottenuti sono soddisfacenti. Considerando la metrica principale, e quella che ha guidato tutto il processo di creazione e tuning degli iperparametri, il nostro modello ha ottenuto performance migliori di tutte le baseline in tutti gli indici, eccezion fatta per il DAX 30. Se si considera una metrica secondaria come l'FD%, tuttavia, la GAN ottiene performance leggermente peggiori.

5.5 Risultati qualitativi

Per risultati qualitativi si intendono tutti i risultati che non sono utili per un confronto numerico, ma che forniscono solo in maniera visuale un'idea della qualità dei sample generati. Nel nostro caso, come risultati qualitativi si considerano i grafici che mostrano i valori prodotti dal *generator* e dal *discriminator* in diverse fasi del training della rete. In queste analisi, pertanto, non si sono utilizzate in alcun modo le *baseline* considerate nella parte di risultati quantitativi.

La prima analisi proposta è lo studio di come l'input dato al *generator* influenzi il suo output. Per fare ciò si sono mostrati in input al modello del *generator*, già allenato, sequenze con diverse caratteristiche, e gli si è chiesto di generare un certo numero di diversi scenari. Come è possibile vedere in figura 5.7, le traiettorie generate mostrano un alto grado di correlazione pur mantenendo sufficiente variabilità, se condizionate sulla stessa sequenza in input: si veda, per esempio 5.7b. Se l'input del *generator* cambia anche di poco, tuttavia, il trend medio delle serie cambia a sua volta: si veda, per esempio 5.7a e 5.7b. In queste due figure le serie di prezzi iniziali sono molto simili all'occhio umano, ma gli output del *generator* sono parecchio diversi. Un'altra proprietà che è possibile osservare confrontando, ad esempio, 5.7a e 5.7d, è che il grado di volatilità delle serie prodotte dal *generator* è tanto più grande quanto è più grande il grado di volatilità delle serie in ingresso. Questo testimonia un tentativo del *generator* di riprodurre in output proprietà statistiche dell'input proposto.

Training del generator

Come seconda prova si è analizzato il processo di training del *generator*. Per fare ciò si confronta come questo completi una singola serie in ingresso mentre è nella fase di training. I grafici mostrati in figura 5.8 presentano due tendenze interessanti: in primo luogo, la varianza delle serie generate diminuisce significativamente con il progredire delle epoche. Poi, il *generator* diventa sempre più abile a generare serie realistiche agli occhi del *discriminator*. Questo aspetto è facile da individuare se si considera che l'opacità delle serie generate in rosso è proporzionale al valore di realisticità dato dal *discriminator* alla stessa.

Training del discriminator

Come ultima prova si è analizzato il processo di training del *discriminator*. Per fare ciò si è scelta una serie e un insieme di scenari futuri generati casualmente tramite il modello *random walk*, già discusso in sezione 5.3.3. Periodicamente durante il training si sono proposte al *discriminator* queste serie e si è verificato come questo le valutasse. Si è osservato come il *discriminator* diminuisse il valore di queste serie progressivamente, a partire da quelle che più si differenziavano da quella originale, fino ad arrivare al punto in cui il valore attribuito a tutte le serie è 0, a parte per quelle veramente simile all'originale. Le considerazioni fatte sono possibili in quanto si considera che l'opacità delle serie generate in marrone è proporzionale al valore di realisticità dato dal *discriminator* alla stessa. Si noti che queste serie non sono mai state viste dal *discriminator* in fase di training, poiché non sono state generate dal *generator* del modello ma da un altro modello.

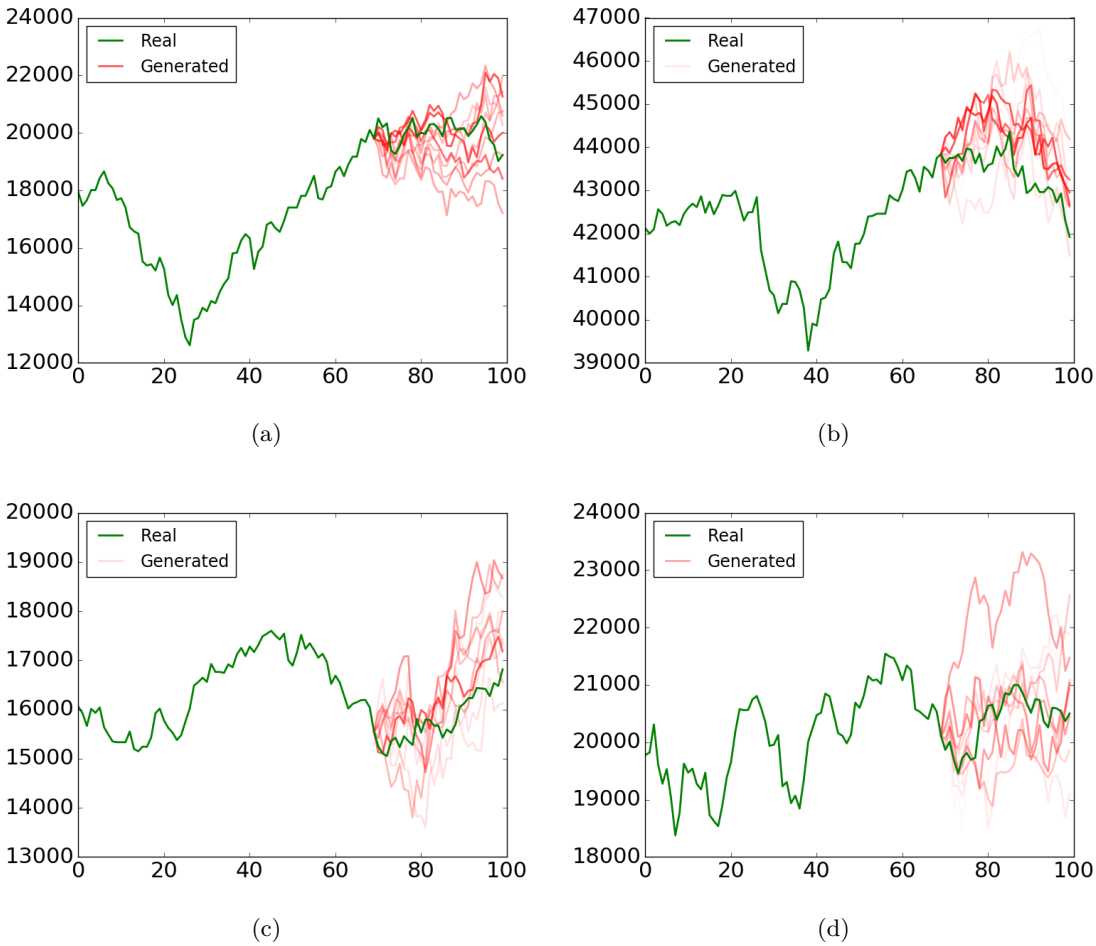


Figura 5.7: Diverse serie di prezzi azionari completate da un generator allenato. Come serie in ingresso si sono scelte: una serie regolare (a), una decrescente (b), una crescente (c) e una ad alta volatilità (d). I grafici sono stati ottenuti svolgendo 10 simulazioni per ogni serie in input. Si noti che l'opacità di ogni serie generata (in rosso) è proporzionale all'ouput del discriminator quando l'input è la stessa, ovvero più è trasparente più è facile da riconoscere come falsa.

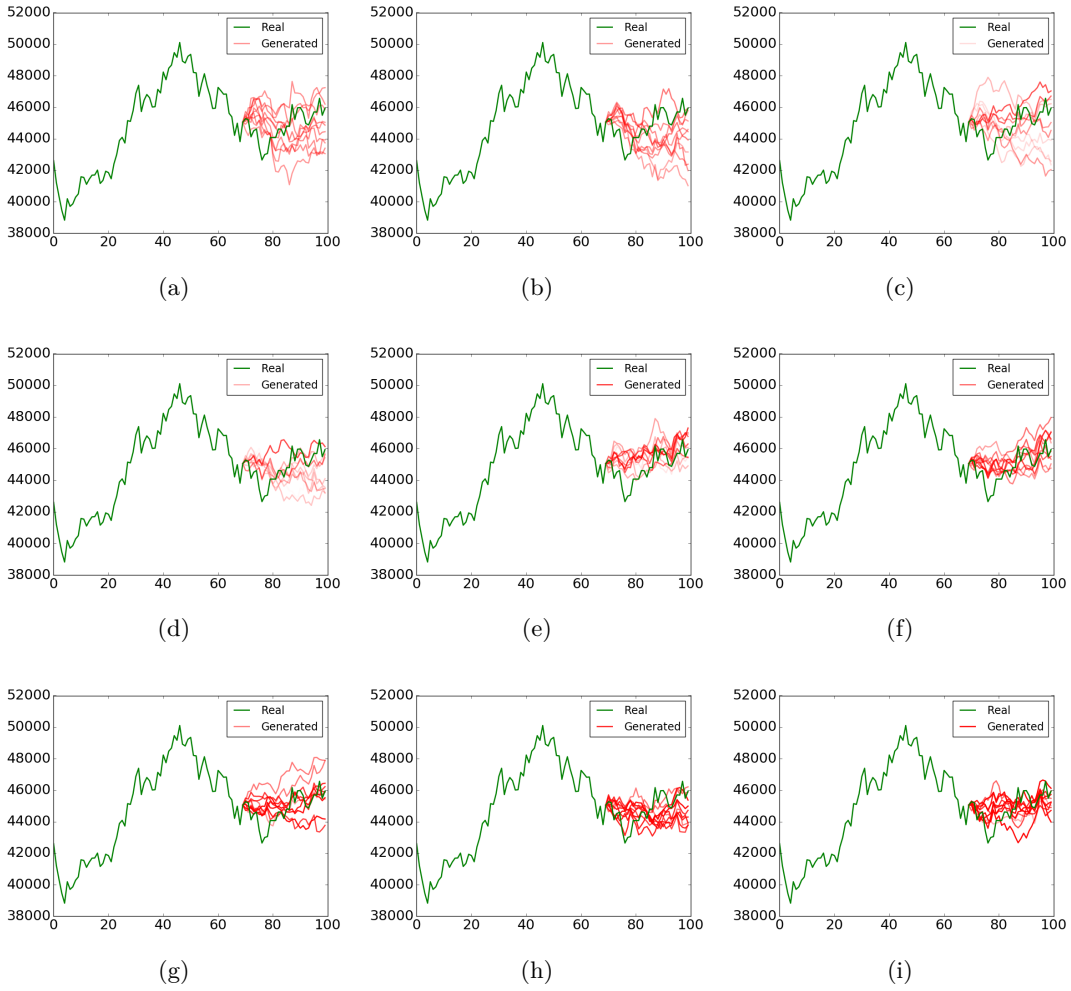


Figura 5.8: Una serie azionaria completata dal modello durante il processo di training. I grafici sono stati ottenuti svolgendo 10 simulazioni ogni 150 epoche. Si noti che l'opacità di ogni serie generata (in rosso) è proporzionale all'ouput del discriminator quando l'input è la stessa, ovvero più è trasparente più è facile da riconoscere come falsa.

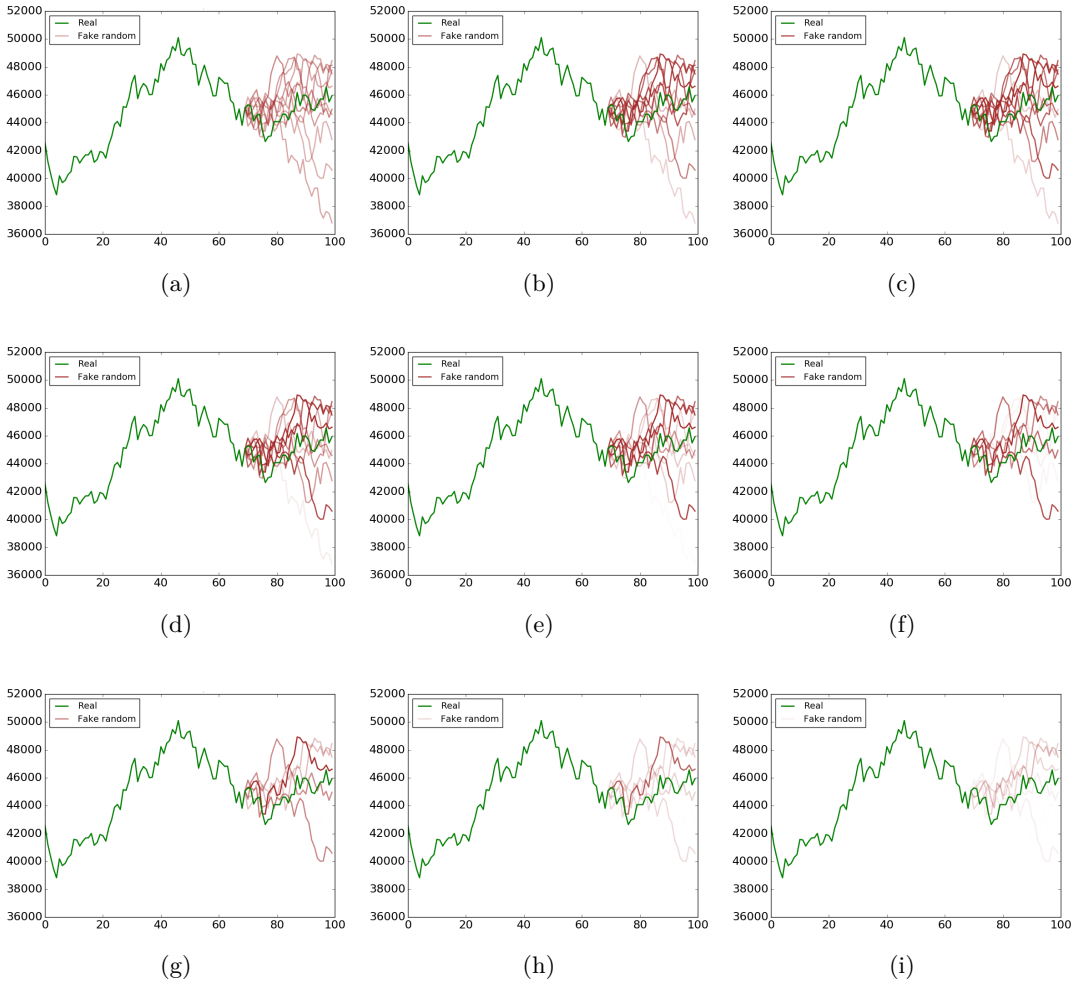


Figura 5.9: Una serie azionaria completata per 10 volte dal modello random walk all'inizio del processo di training. Il test consiste nell'esaminare come varia il comportamento del discriminator mantenendo costanti le seire. I grafici sono stati ottenuti svolgendo il test ogni 200 epoche. Si noti che l'opacità di ogni serie generata (in marrone) è proporzionale all'ouput del discriminator quando l'input è la stessa, ovvero più è trasparente meglio questo l'ha riconosciuta come falsa.

Capitolo 6

Conclusioni e sviluppi futuri

In questo capitolo si traggono le conclusioni sul lavoro prodotto e sugli obiettivi che sono stati raggiunti. Allo scopo, nella sezione 6.1 si ricapitolano brevemente le motivazioni e gli obiettivi del progetto. In seguito, in sezione 6.2, si presentano svariate possibilità di sviluppi futuri: si propongono modifiche al modello costruito, nuovi modelli e utilizzi alternativi delle parti del modello creato.

6.1 Conclusioni

Lo scopo principale della tesi è quello di esplorare la possibilità di utilizzare reti con una struttura ispirata da quella delle GAN per affrontare problemi di previsione e analisi dell'andamento di serie temporali finanziarie. Essendo un campo di ricerca totalmente innovativo, il progetto è stato strutturato per affrontare le sfide relative, a partire dallo studio della letteratura, fino alla valutazione dei risultati comparativi con le baseline che sono state studiate e implementate. Pertanto, gli obiettivi che si sono raggiunti durante lo sviluppo del progetto legato alla tesi sono:

- Studio della letteratura: si è partiti approfondendo l'ambito economico in cui il progetto è situato, analizzando il funzionamento del mercato azionario e gli strumenti e teorie già presenti nel campo. In seguito, si è esaminata la letteratura a disposizione riguardante genericamente il *deep learning* e in particolare le *Generative Adversarial Networks*.

- Test preliminari di fattibilità: dopo aver studiato le principali teorie economiche sull'andamento dei mercati, e aver appreso come queste si opponessero alla realizzabilità del modello pensato, si è messa in discussione l'effettiva fattibilità del progetto. Si è messo a punto un test che stressasse la parte di modello ritenuta deficitaria, e si sono raccolti i risultati.

- Implementazione del modello: una volta stabilita la fattibilità del progetto, si è proceduto con lo sviluppo e l'implementazione del modello proposto. Utilizzando il server aziendale, si è esplorato accuratamente lo spazio degli iperparametri, per raggiungere la configurazione che rendesse il modello creato il più performante possibile.

- Implementazione baselines: per riuscire ad avere un'idea sulla qualità dell'output del modello, si sono pensate e implementate varie baselines, ovvero modelli semplicistici che trovano giustificazione in modelli statistici dell'andamento del prezzo dei titoli azionari, con il solo scopo di fungere da confronto.

- Raccolta e valutazione dei risultati: nella fase finale si sono prodotti i risultati, volti a visualizzare qualitativamente e quantitativamente come il modello costruito si comportasse su dataset non ancora osservati. I test qualitativi, più importanti in quanto attestano l'effettiva efficacia del modello costruito, sono stati affrontati effettuando un confronto delle performance del modello con quelle delle varie baseline implementate. I test qualitativi, invece, si sono limitati ad un'analisi degli scenari prodotti dalla rete se condizionata da diverse sequenze di input o durante la fase di training.

Nonostante il progetto fosse particolarmente ambizioso, l'esito finale si può definire positivo. Durante lo svolgimento del progetto, si sono presentate diverse problematiche, che sono state tutte risolte con successo. Anche se tutte le baseline si possono considerare battute sui dataset considerati, vi sono ampi margini di miglioramento per la rete implementata, o altre tipologie reti che possono essere utilizzate allo scopo. Inoltre, vi sono diversi utilizzi alternativi della stessa che non sono stati considerati durante il progetto. Questi sviluppi futuri sono mostrati in dettaglio nella sezione che segue.

6.2 Sviluppi futuri

Durante lo sviluppo del progetto, non sono mancati i momenti in cui si è pensato a modi alternativi per realizzare il modello in analisi e teorizzato proposte di ulteriori utilizzi della rete realizzata. In seguito viene quindi riportata una riflessione sulle opportunità di estensione futura del progetto legato alla tesi.

Utilizzi alternativi

L'utilizzo della rete che si è analizzato nella fase di valutazione ed esposizione dei risultati è solamente il primo esposto nella fase di introduzione (sezione 1.1). In particolare, quello che abbiamo analizzato è la capacità del *generator* di stabilire la direzionalità del titolo analizzato. Per ogni serie da analizzare, si fanno generare dal *generator* un certo numero di scenari futuri e si valuta la veridicità dell'andamento della media di questi. Nonostante risulti molto rilevante come use case, quello che si potrebbe fare con la GAN non si limita solo a questo. In seguito si mostrano i possibili ulteriori utilizzi che la rete potrebbe avere:

- Utilizzare il *generator* per acquisire proprietà statistiche sul titolo analizzato: invece che utilizzare solo la media per stabilire se un titolo nei prossimi giorni cresca o cali, è possibile trarre ulteriori informazioni statistiche utili agli analisti, come la volatilità prevista o gli intervalli di confidenza della previsione. Oltre a fornire supporto decisionale, la rete può essere quindi anche utilizzata nel processo di analisi dei rischi.

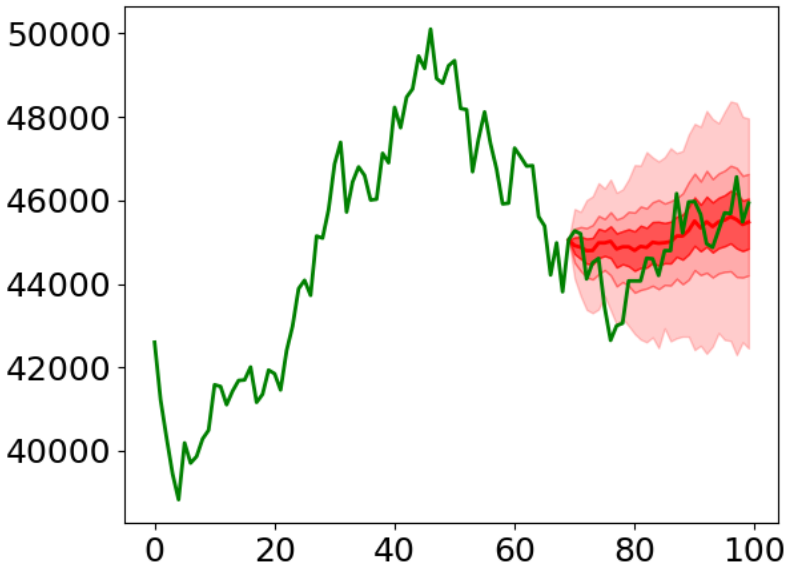


Figura 6.1: Grafico degli intervalli di confidenza dati dal generator. La linea in verde rappresenta l'andamento reale del titolo per 100 giorni. Al modello sono state fatte generare 10000 previsioni per gli ultimi 30 giorni, e la linea in rosso è la mediana di queste. Assumendo la distribuzione dei dati gaussiana, si sono costruiti gli intervalli di confidenza a σ, 2σ, e $\simeq 3\sigma$, ovvero il 68simo, 95simo e 100simo percentile. Per ognuno di questi si è colorato l'intervallo con un una opacità di rosso sempre minore.

In figura 6.1 viene mostrato un esempio serie con i propri intervalli di confidenza, calcolati in base alle serie generate dal *generator*.

- Utilizzare il *discriminator* come *anomaly detector*: è possibile utilizzare il discriminator per identificare all'interno della serie di dati reali movimenti di mercato anomali o eventuali errori.

- Utilizzare il *discriminator* per valutare altri algoritmi: è possibile utilizzare il *discriminator* per giudicare la realisticità dei segnali prodotti da altri algoritmi o modelli. In tal modo è possibile creare una vera e propria metrica di realisticità, utilizzabile per svariati scopi.

- Utilizzare il *discriminator* per scegliere i migliori sample del *generator* da considerare: invece che utilizzare la media di tutti gli scenari generati dal *generator* per avere indicazioni sulla direzionalità dell'indice, è possibile selezionare solo quelli più verosimili, ovvero quelli per i quali l'output del *discriminator* superi una certa soglia. Un'altra ipotesi è quella di utilizzare una media pesata data dal valore del *discriminator*, invece che eliminare quelli meno realistici.

Modifiche al modello

Durante il corso di tutta l'attività svolta si sono dovute compiere diverse scelte progettuali, sia riguardo la costruzione del modello che la scelta delle impostazioni del dataset. Nonostante le scelte fatte abbiano portato a risultati complessivamente positivi, non è da escludere l'eventualità che le alternative fossero più valide. Pertanto, vi sono alcune modifiche al modello e al dataset che si possono proporre:

- Utilizzare altri tipi di dati: oltre alla possibilità di utilizzare altri indici azionari (come il Dow Jones, il Nasdaq o il Nikkei 225), potrebbe essere utile testare l'efficacia dell'architettura utilizzata su altre tipologie di dati finanziari, come per esempio singole azioni (tipo Apple, Oracle o General Electric) oppure di dati relativi al forex (come EUR/USD, USD/JPY o GBP/USD).

- Utilizzare altri intervalli di tempo: i dati a disposizione per l'indice in analisi sono giornalieri, ovvero l'intervallo di tempo tra un dato e quello successivo è di 24 ore (festivi a parte). Un'alternativa esplorabile è quella di campionare i dati con maggiore frequenza, ovvero ogni ora oppure ogni minuto.

- Utilizzare altre architetture per il *generator*: questo punto è quello su cui è possibile spaziare in maniera più ampia. Le architetture che si potrebbero implementare allo scopo potrebbero andare da una *sequence to sequence* più classica, composta da due reti ricorrenti come nel paper originale [33], fino ad arrivare ad architetture più inusuali, nelle quali si segue l'idea alla base della teoria del *random walk* facendo generare al *generator* solo la media μ e deviazione standard σ della gaussiana da cui verranno campionati i valori futuri. Oltre a questo tipo di soluzioni, è opportuno considerare anche modelli di tipo autoregressivo, che stanno avendo molto successo nel campo dell'elaborazione di serie temporali, come per esempio la rete WaveNet [34].

Bibliografia

[1] Martín Abadi et al. «Tensorflow: A System for Large-Scale Machine Learning.» In: *OSDI*. Vol. 16. 2016, pp. 265–283.

[2] Gunduz Caginalp e Mark DeSantis. «Nonlinearity in the Dynamics of Financial Markets». In: *Nonlinear Analysis: Real World Applications* 12.2 (2011), pp. 1140–1151.

[3] Soumith Chintala et al. *How to Train a GAN? Tips and Tricks to Make GANs Work*. 2016.

[4] François Chollet. *Keras*. 2015.

[5] Balázs Csanád Csáji. «Approximation with Artificial Neural Networks». In: *Faculty of Sciences, Etvs Lornd University, Hungary* 24 (2001), p. 48.

[6] Emily Denton, Sam Gross e Rob Fergus. «Semi-Supervised Learning with Context-Conditional Generative Adversarial Networks». In: *arXiv preprint arXiv:1611.06430* (2016).

[7] Cristóbal Esteban, Stephanie L. Hyland e Gunnar Rätsch. «Real-Valued (Medical) Time Series Generation with Recurrent Conditional GANs». In: *arXiv preprint arXiv:1706.02633* (2017).

[8] Kunihiko Fukushima e Sei Miyake. «Neocognitron: A Self-Organizing Neural Network Model for a Mechanism of Visual Pattern Recognition». In: *Competition and Cooperation in Neural Nets*. Springer, 1982, pp. 267–285.

[9] Xavier Glorot e Yoshua Bengio. «Understanding the Difficulty of Training Deep Feedforward Neural Networks». In: *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*. 2010, pp. 249–256.

[10] Ian J. Goodfellow et al. «Generative Adversarial Networks». In: *arXiv:1406.2661 [cs, stat]* (giu. 2014). arXiv: 1406.2661 [cs, stat].

[11] Kaiming He et al. «Delving Deep into Rectifiers: Surpassing Human-Level Performance on Imagenet Classification». In: *Proceedings of the IEEE International Conference on Computer Vision*. 2015, pp. 1026–1034.

[12] Sepp Hochreiter. «Untersuchungen Zu Dynamischen Neuronalen Netzen». In: *Diploma, Technische Universität München* 91.1 (1991).

[13] Sepp Hochreiter e Jürgen Schmidhuber. «Long Short-Term Memory». In: *Neural computation* 9.8 (1997), pp. 1735–1780.

[14] Diederik P. Kingma e Jimmy Ba. «Adam: A Method for Stochastic Optimization». In: *arXiv preprint arXiv:1412.6980* (2014).

[15] Anders Krogh e John A. Hertz. «A Simple Weight Decay Can Improve Generalization». In: *Advances in Neural Information Processing Systems*. 1992, pp. 950–957.

[16] Yann LeCun et al. «Handwritten Digit Recognition with a Back-Propagation Network». In: *Advances in Neural Information Processing Systems*. 1990, pp. 396–404.

[17] Jiwei Li et al. «Adversarial Learning for Neural Dialogue Generation». In: *arXiv preprint arXiv:1701.06547* (2017).

[18] Andrew W. Lo e A. Craig MacKinlay. «Data-Snooping Biases in Tests of Financial Asset Pricing Models». In: *The Review of Financial Studies* 3.3 (1990), pp. 431–467.

[19] Burton G. Malkiel e Eugene F. Fama. «Efficient Capital Markets: A Review of Theory and Empirical Work». In: *The journal of Finance* 25.2 (1970), pp. 383–417.

[20] Burton Gordon Malkiel. *A Random Walk down Wall Street: The Time-Tested Strategy for Successful Investing*. 9th ed. OCLC: ocm72798896. New York: W. W. Norton, 2007. ISBN: 978-0-393-06245-8.

[21] Kim man Lui e Terence TL Chong. «Do Technical Analysts Outperform Novice Traders: Experimental Evidence». In: *Economics Bulletin* 33.4 (2013), pp. 3080–3087.

[22] Warren S. McCulloch e Walter Pitts. «A Logical Calculus of the Ideas Immanent in Nervous Activity». In: *The bulletin of mathematical biophysics* 5.4 (1943), pp. 115–133.

[23] Mehdi Mirza e Simon Osindero. «Conditional Generative Adversarial Nets». In: *arXiv preprint arXiv:1411.1784* (2014).

[24] Vinod Nair e Geoffrey E. Hinton. «Rectified Linear Units Improve Restricted Boltzmann Machines». In: *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*. 2010, pp. 807–814.

[25] Cheol-Ho Park e Scott H. Irwin. «The Profitability of Technical Analysis: A Review». In: (2004).

[26] Lutz Prechelt. «Automatic Early Stopping Using Cross Validation: Quantifying the Criteria». In: *Neural Networks* 11.4 (1998), pp. 761–767.

[27] Alec Radford, Luke Metz e Soumith Chintala. «Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks». In: *arXiv:1511.06434 [cs]* (nov. 2015). arXiv: 1511.06434 [cs].

[28] MarcAurelio Ranzato et al. «Video (Language) Modeling: A Baseline for Generative Models of Natural Videos». In: *arXiv preprint arXiv:1412.6604* (2014).

[29] Scott Reed et al. «Generative Adversarial Text to Image Synthesis». In: *arXiv preprint arXiv:1605.05396* (2016).

[30] Frank Rosenblatt. «The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain.» In: *Psychological review* 65.6 (1958), p. 386.

[31] Tim Salimans et al. «Improved Techniques for Training Gans». In: *Advances in Neural Information Processing Systems*. 2016, pp. 2234–2242.

[32] Nitish Srivastava et al. «Dropout: A Simple Way to Prevent Neural Networks from Overfitting». In: *The Journal of Machine Learning Research* 15.1 (2014), pp. 1929–1958.

[33] Ilya Sutskever, Oriol Vinyals e Quoc V. Le. «Sequence to Sequence Learning with Neural Networks». In: *Advances in Neural Information Processing Systems*. 2014, pp. 3104–3112.

[34] Aäron Van Den Oord et al. «WaveNet: A Generative Model for Raw Audio.» In: *SSW*. 2016, p. 125.

[35] Jean Ville. *Etude Critique de La Notion de Collectif*. Gauthier-Villars Paris, 1939.

[36] P.J. Werbos. *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences*. Harvard University, 1975.

[37] Lantao Yu et al. «SeqGAN: Sequence Generative Adversarial Nets with Policy Gradient.» In: *AAAI*. 2017, pp. 2852–2858.

[38] Yizhe Zhang et al. «Adversarial Feature Matching for Text Generation». In: *arXiv preprint arXiv:1706.03850* (2017).