UNIVERSITÀ DEGLI STUDI DI MODENA E REGGIO EMILIA

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

Estensione e industrializzazione di modelli di reti generative applicati alla modellazione e previsione di serie temporali finanziarie

Relatore:
Prof. Simone Calderara

Candidato:
Landi Roberto

Tutore aziendale
Axyon AI SRL
Ing. Daniele Grassi

Anno accademico 2017/2018

Abstract

La previsione del mercato azionario è da sempre un problema aperto nel mondo della finanza. In questa tesi viene proposto un modello, per la generazione di serie temporali finanziarie che fa uso della tecnologia GAN, Generative Adversarial Network.

A partire dalla loro introduzione nel mondo del machine learning, le GAN hanno dimostrato la loro efficienza nella generazione di immagini e in altri lavori legati al mondo della visione artificiale, mentre sulle serie temporali pochi lavori sfruttano questa tecnologia, in particolare, non risultano in letteratura, lavori riguardanti la finanza.

Per definizione, le GAN sono composte da due reti neurali che competono tra di loro. Nel modello proposto, per la parte di generazione degli scenari futuri, viene usato un modello sequence-to-sequence con relativo meccanismo di attention.

Le sequenze generate, vengono fornite in ingresso a una seconda rete neurale che utilizzando layers convolutivi e un layer fully-connected finale ha il compito di determinare se una sequenza possa essere considerata come reale o come generata.

Avendo un elevato numero di scenari vero-simili per una sequenza finanziaria data in ingresso, sarà possibile fare previsioni e studiare con maggiore semplicità l’andamento futuro dei prezzi azionari.

Indice generale

1. Introduzione....5
1.1 Scopo della tesi....6
1.2 Axyon AI....7
1.3 Strumenti utilizzati....7
1.3.1 Pycharm....7
1.3.2 Tensorflow....8
1.3.3 Keras....8
1.4 Struttura della tesi....8
2. Quadro teorico....10
2.1 Deep Learning....10
2.1.1 Neural Network....12
2.1.2 Convolutional Neural Networks....15
2.1.3 Long Short Term Memory....17
2.1.4 Generative Adversarial Networks....19
2.1.5 Modello Sequence-to-Sequence....21
2.1.6 Attention....22
2.2 Mercato azionario....23
2.2.1 Mercato al rialzo....24
2.2.2 Mercato al ribasso....24
2.2.3 Teoria dei mercati efficienti....25
2.2.4 Teoria del Random Walk....25
2.2.5 Indici azionari....26
3. GAN per l’analisi del mercato azionario....32
3.1 Definizione del problema....32
3.2 Studio di fattibilità....33
3.2.1 Discriminatore....33
3.2.2 Generatore....37
3.3 Metrica....38
3.4 Problematiche riscontrate....40
3.4.1 Overfitting....40
3.4.2 Mode Collapse....41
4. Modello....44
4.1 Architettura globale....44
4.2 Generatore....45
4.2.1 Encoder....46
4.2.2 Decoder....47
4.3 Discriminatore....48
4.4 Loss....49
4.5 Ottimizzatore....50
5. Esperimenti e risultati....51
5.1 Dataset....51

5.1.1 Variazione percentuale....52
5.1.2 Split....52
5.1.3 Normalizzazione....53
5.1.4 Overlap....54
5.1.5 Indicatori tecnici....55
5.2 Metodo di Addestramento....59
5.2.1 Iperparametri di training....60
5.3 Baseline....64
5.3.1 Modello Random Walk....64
5.3.2 Sequence-to-Dense GAN....64
5.4 Risultati Quantitativi....66
5.5 Risultati Qualitativi....67
6. Conclusioni....73
6.1 Sviluppi futuri....74
Ringraziamenti....76
Bibliografia....77

Indice delle illustrazioni

Illustrazione 1: Relazione tra Intelligenza artificiale, Machine Learning e Deep Learning mostrata graficamente....12
Illustrazione 2: Confronto tra ML e DL rispetto ai dati utilizzati....13
Illustrazione 3: Esempio funzionamento classificatore lineare....15
Illustrazione 4: Funzionamento neurone....16
Illustrazione 5: Esempio architettura di una rete neurale....17
Illustrazione 6: Funzionamente Convolutional Neural Network....18
Illustrazione 7: Rete neurale ricorrenti con cicli al suo interno....19
Illustrazione 8: Struttura interna di una cella LSTM....20
Illustrazione 9: Esempio Architettura GAN....22
Illustrazione 10: Esempio modello Sequence-to-Sequence....23
Illustrazione 11: Andamento FTSE MIB....32
Illustrazione 12: Andamento FTSE 100....33
Illustrazione 13: Andamento S&P 500....33
Illustrazione 14: Andamento CAC 40....34
Illustrazione 15: Andamento S&P/STX 60....34
Illustrazione 16: Andamento SMI 20....35
Illustrazione 17: Andamento DAX 30....35
Illustrazione 18: Andamento IBEX 35....36
Illustrazione 19: Andamento STOXX 50....36
Illustrazione 20: Andamento TOPIX....37
Illustrazione 21: Risultato del primo esperimento....43
Illustrazione 22: Risultati secondo esperimento mono....45
Illustrazione 23: Risultati del secondo esperimento agnostico....46
Illustrazione 24: Andamento delle loss e della likelihood durante il training....50
Illustrazione 25: Esempio di Mode Collapse....54
Illustrazione 26: Rappresentazione grafica del modello proposto....56
Illustrazione 27: Rappresentazione grafica del encoder all'interno del generatore....58
Illustrazione 28: Architettura del decoder all'interno del generatore....59
Illustrazione 29: Schema a blocchi dell'architettura del discriminatore....61
Illustrazione 30: Spiegazione grafica di come funziona SGD con e senza momentum. 62
Illustrazione 31: Rappresentazione grafica split dataset FTSE MIB....65
Illustrazione 32: Decoder del generatore nel modello GAN sequence-to-dense....77
Illustrazione 33: Sequenze generate con il modello random walk....80
Illustrazione 34: Sequenze generate con il modello GAN Sequence-to-Dense....81
Illustrazione 35: Sequenze generate con il modello GAN agnostico....82
Illustrazione 36: Analisi delle sequenze generate durante il training....83
Illustrazione 37: Rappresentazione grafica del vettore di attention....84
Illustrazione 38: Risultato finale che mostra gli intervalli di confidenza del generatore e vettore di attention....86

Indice delle tabelle

Tabella 1: Risultati studio fattibilità sul generatore.....37
Tabella 2: Dimensioni dei dataset di training, validation e test di un singolo indice considerando il numero di giorni presi in considerazione.....52
Tabella 3: Dimensioni dei dataset complessivi di training, validation e test considerando il numero di giorni presi in considerazione.....53
Tabella 4: Numero di sequenze contenute nel Training set, Validation set e Test set.....54
Tabella 5: Risultati quantitativi in termini di likelihood calcolati sulle diverse baseline proposte.....66
Tabella 6: Risultati quantitativi in termini di FE% calcolati sulle diverse baseline proposte.....67

1. Introduzione

Grazie alla sempre più numerosa disponibilità di dati che ci viene messa a disposizione dall'era informatica in corso e alla crescente potenza computazionale ormai alla portata di chiunque, sta spopolando l'utilizzo del *machine learning* e del *deep learning* non solo nel campo della visione artificiale ma in ogni settore dove sia presente una significativa quantità di dati, come ad esempio, nella finanza dove è disponibile una alta concentrazione di dati numerici.
Una delle idee più interessanti nel mondo del *deep learning* è stata introdotta nel 2014 dal ricercatore americano Ian J. Goodfellow ovvero *le Generative Adversiarial Network* (GAN).
Le GAN sfruttano due reti neurali che competono tra di loro, per generare dati (immagini, testi, serie numeriche ecc.) il più reali possibile, campionandoli da una distribuzione che viene imparata dal modello durante l'addestramento.

In finanza il mercato azionario è un luogo, non necessariamente fisico, dove vengono negoziati i titoli di azionari. È da sempre analizzato con molta attenzione da governi, banche e industrie perché rispecchia la salute economica di un paese, chiaramente se un mercato azionario è in costante crescita esso rappresenta un'economia solida.

Il mercato azionario quindi rappresenta un'economia di mercato libero, dove le società hanno la possibilità di accedere a nuovi capitali, senza dover richiedere prestiti bancari in cambio della cessione di una porzione della titolarità della società stessa a terzi interessati.

Il mercato azionario non solo fornisce nuovi strumenti finanziari alle aziende ma permette agli investitori di incrementare i loro capitali se investono su azioni redditizie.
La bravura di un investitore sta proprio nello scegliere su quali azioni investire, usando le proprie conoscenze per effettuare decisioni, siccome non esiste un modello che permette di generare un profitto sicuro e quindi di prevedere quali saranno gli andamenti del mercato.

L'idea di questa tesi è studiare la possibilità di applicare il *deep learning*, in particolare la tecnologia GAN, alla finanza, estendendo un modello già in possesso della azienda ospitante, per generare scenari futuri il più vero-simili possibile analizzando la serie storica dei prezzi e degli indicatori tecnici che si hanno a disposizione per ogni indice azionario.

1.1 Scopo della tesi

Il progetto legato a questa tesi è stato svolto presso l'azienda Axyon AI, che basa i suoi principali prodotti sull'uso del *deep learning.* Lo scopo di questa tesi è quello di utilizzare la tecnologia GAN per analizzare e prevedere l'andamento di serie temporali finanziare.

Prima di questo progetto l'azienda Axyon AI aveva già a disposizione un modello GAN per affrontare questo problema ma con alcuni limiti.

Il modello precedente però era in grado di trattare un singolo indice azionario per volta con la necessità di avere *n* modelli paralleli nel caso si fosse interessati a generare scenari appartenenti ad *n* indici diversi. Il primo obbiettivo è stato di studiare la possibilità di estendere la capacità del modello, in modo da gestire un numero arbitrario di indici contemporaneamente.

Come successivamente verrà approfondito, le GAN sono formate da due reti neurali che competono tra loro ovvero: il D*iscriminator,* che ha il compito di determinare se le serie generate sono reali oppure no ed il G*enerator,* che deve generare serie il più realistiche possibile. Proprio sul G*enerator* si sono concentrati i maggiori sforzi con l'obbiettivo di utilizzare una architettura puramente S*equence-to-Sequence* in simbiosi con una delle più innovative tecnologie in campo *machine learning:* l'A*ttention*.

Un ulteriore obbiettivo è stato quello di ricercare una metrica adatta, in grado di racchiudere in modo significativo la bontà o meno, delle serie generate dalla GAN.
Un modello di questo tipo, potrebbe trovare vari utilizzi nella finanza, ovvero:

- Essere usato per avere un indicazione su quale sia la direzionalità più probabile di un indice azionario nel immediato futuro. Oltre che avere una stima sulla volatilità dell'indice e dare possibilità di effettuare diverse analisi tecniche.
- Utilizzare il *generator* per generare serie complete. È norma comune in finanza prendere serie del passato e riordinare i prezzi delle sequenze in modo casuale per testare nuove strategie di investimento. Generando serie più realistiche rispetto a un riordinamento casuale si può avere una stima più precisa di quanto una strategia di investimento sia affidabile.
- Utilizzare il *discriminator* come rilevatore di anomalie per le serie reali mettendo in evidenza quindi situazioni particolari o critiche.

Lo scopo della tesi è quindi di ricerca, e l'idea finale è andare a integrare con una GAN funzionante IRIS, uno dei prodotti di Axyon AI o industrializzare un prodotto nuovo a se stante.

1.2 Axyon AI

Axyon AI, è un'azienda modenese che usa il *deep learning* per fornire soluzioni su misura nel settore finanziario. Axyon AI, mette a disposizione delle banche e dei fondi di investimento strumenti per ottimizzare i processi di business.

In particolare vende due prodotti:

- IRIS → È uno strumento basato su AI che permette di integrare le strategie degli *asset manager* con predizioni generate da algoritmi proprietari che fanno uso delle principali tecniche di *deep learning*. IRIS fa uso di vari dati del mercato finanziario e utilizza tecniche innovative come l'analisi del sentimento, oltre a utilizzare dati forniti direttamente dal cliente, per generare le sue predizioni. Il modello proposto in questa tesi potrebbe andare a integrare questo software per dare un ulteriore supporto alle strategie degli *asset manager*.
- SynFinance → È uno strumento che offre supporto al mercato dei prestiti sindacati e tramite un motore AI fornisce ad esempio una stima di quanto una banca possa essere interessata o meno a partecipare a un particolare tipo di finanziamento.

1.3 Strumenti utilizzati

Verranno di seguito descritti i principali strumenti di sviluppo utilizzati per la realizzazione del progetto a livello implementativo, che hanno permesso di avere un codice semplice e gestibile pur trattando un problema molto complesso come la progettazione di GAN.

1.3.1 Pycharm

Pycharm è un ambiente di programmazione usato principalmente per il linguaggio *Python*. Fornisce un'assistenza alla programmazione segnalando tempestivamente eventuali errori nel codice, permette un *debug* semplice e intuitivo, supporta lo sviluppo per *Google App Engine* oltre ad avere integrato al suo interno il supporto a G*it* per il controllo e l'integrazione delle diverse versioni di un progetto.

L'autore di Pycharm, JetBrains mette a disposizione una licenza gratuita per gli studenti universitari.

1.3.2 Tensorflow

TensorFlow (1) è un software *open source* per il *machine learning* che fornisce moduli testati e ottimizzati utili nella realizzazione di algoritmi per diversi compiti di apprendimento.

Sviluppato da Google Brain in C++ fornisce una comoda API in Python e supporta l'esecuzione dei programmi su GPU. Tensorflow è una libreria pensata per la computazione numerica attraverso *flow graph. Un flow graph* è un grafo diretto aciclico che rappresenta una computazione, dove ogni nodo è un operazione matematica che prende in input zero o più tensori e restituisce zero o più tensori.

Questa struttura è estremamente importante nel processo di *backpropagation,* siccome facilita il calcolo delle derivate.

1.3.3 Keras

Keras (2) è una libreria *open source* per Python, è progettata per dare un'astrazione a un livello superiore di altre librerie che vanno ad operare per esempio su tensori a basso livello, supportando quindi come *back-end* note librerie quali TensorFlow o Theano.

Keras fornisce un'interfaccia semplice e intuitiva anche per componenti molto complicati, si è scelto quindi di sviluppare il progetto tramite Pycharm utilizzando per i modelli di *deep learning* la libreria Keras con Tensorflow come *backend.*

1.4 Struttura della tesi

In seguito verrà mostrata la struttura in capitoli della tesi e una breve descrizione per ognuno di essi:

- Capitolo 1: Introduzione

- Capitolo 2: Richiamo alle definizioni e ai concetti fondamentali di m*achine learning e deep learning* utilizzati durante lo svolgimento del progetto di tesi, contestualizzazione del lavoro svolto nell'ambito finanziario con introduzione degli indici azionari utilizzati.

- Capitolo 3: Definizione del problema nel dettaglio, studio di fattibilità approfondito per *discriminator* e *generator,* scelta della metrica e esposizione delle problematiche riscontrate con la corrispondente soluzione adottata.

- Capitolo 4: Presentazione del modello proposto, con approfondimento dei vari dettagli dell'architettura, analisi del G*enerator,* del D*iscriminator,* descrizione della *loss function* e dell'ottimizzatore utilizzato.

- Capitolo 5: Dettagli sulla composizione del *dataset,* descrizione degli iper parametri utilizzati per il modello, presentazione delle *baseline* utilizzati con relativi risultati quantitativi e qualitativi.

- Capitolo 6: Conclusioni e discussione finale sugli obbiettivi raggiunti durante lo svolgimento del progetto, applicazioni possibili del progetto e illustrazione di alcune idee su possibili sviluppi futuri.

2. Quadro teorico

Questo capitolo si suddivide in due parti, nella prima parte (2.1) verranno introdotti gli elementi principali del *deep learning* con una analisi più approfondita delle tecniche utilizzate durante il progetto svolto.

La seconda parte (2.2) sarà invece centrata sul mondo della finanza per capire in che contesto verrà utilizzato il modello realizzato e verranno introdotti gli indici azionari che sono stati utilizzati per l'apprendimento.

2.1 Deep Learning

Il *Deep Learning* (DL) o apprendimento profondo fa riferimento agli algoritmi ispirati alla struttura e al funzionamento del cervello umano. È una sotto categoria del *Machine Learning* (ML) o apprendimento automatico che ha sua volta è una branca dell'intelligenza artificiale.

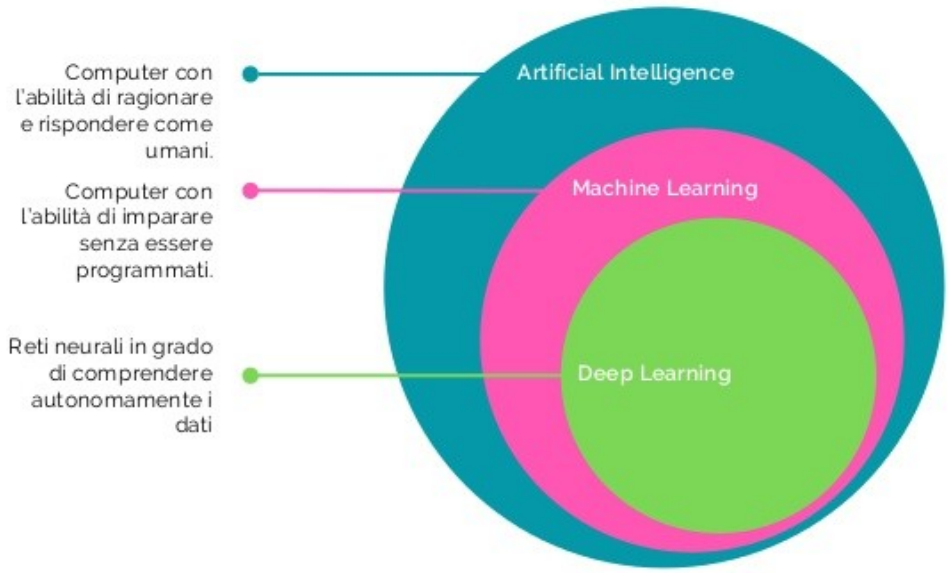

Illustrazione 1: Relazione tra Intelligenza artificiale, Machine Learning e Deep Learning mostrata graficamente.

Quindi viene chiamata intelligenza artificiale la capacità che hanno oggetti inanimati come computer, macchine, robot, di prendere decisioni, di compiere lavori complessi ed assumere comportamenti tipici di un essere umano mostrando quindi 'intelligenza'.

Una possibile strada per ottenere macchine dotate di intelligenza artificiale è appunto il ML, dove si cerca di risolvere un problema senza programmare esplicitamente le operazioni da svolgere ma cercando di far apprendere al sistema la soluzione del problema, mostrandogli un gran numero di esempi. Se per ottenere la soluzione di un problema si fa uso delle cosiddette reti neurali, che molto spesso hanno un elevato numero di strati, si sta invece lavorando nel campo del DL.

Negli ultimi anni si è riportata molta attenzione sulle architetture di *Deep Learning.* Nonostante i primi lavori sulle reti neurali siano stati introdotti nel 1943 (3) solo nel l'ultimo decennio sono state utilizzate concretamente; come ad esempio nella *computer vision,* nel riconoscimento automatico della lingua parlata, nell'elaborazione del linguaggio naturale, nel riconoscimento audio, nella bioinformatica, ecc.

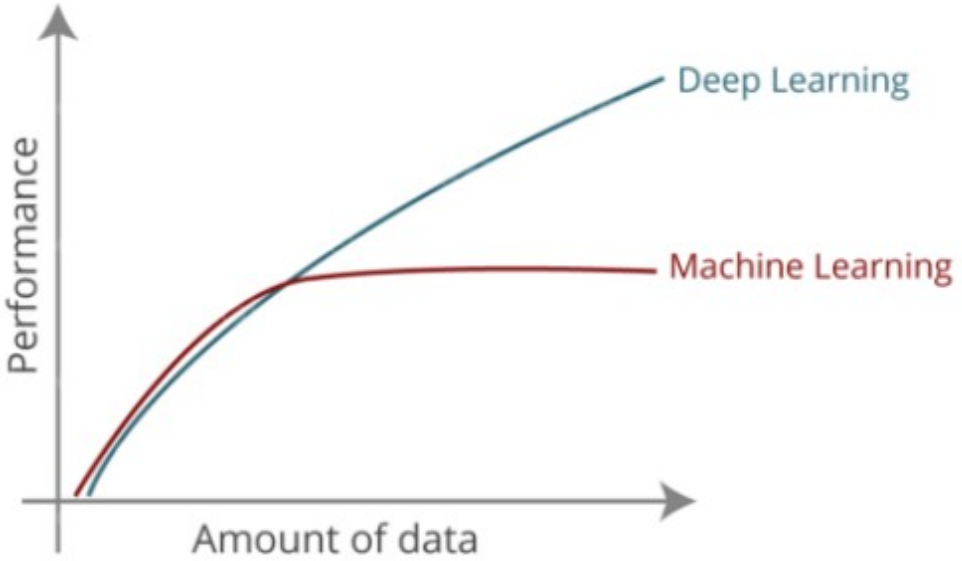

Illustrazione 2: Confronto tra ML e DL rispetto ai dati utilizzati.

Questa esplosione dell'utilizzo del *deep learning* è stata resa possibile principalmente grazie a due fattori:

- L'aumento esponenziale dei dati a disposizione dovuto alla digitalizzazione della società negli ultimi anni che permette al DL di superare in termini di performance il ML.
- L'aumento esponenziale della potenza di calcolo. Per gestire una così grande mole di dati, la velocità computazionale assume un ruolo determinante se si vuole esplorare in modo esaustivo le varie possibilità di modellazione di una rete neurale. L'avvento delle GPU ad esempio ha dato la possibilità alla comunità scientifica di allenare reti anche molto grandi in tempi relativamente brevi.

Si può definire quindi un sistema che utilizza il *Deep Learning*, come un sistema che:

- Elabora dati grezzi, estrapolando a livelli gerarchici multipli le caratteristiche importanti dei dati (Apprendimento non supervisionato). Le caratteristiche di più alto livello sui dati vengono derivate da quelle di più basso livello.
- Usa vari livelli di unità non lineari per svolgere il compito di estrazione di caratteristiche (*features)* rilevanti dai dati. Ciascun livello utilizza l'uscita del livello precedente come ingresso.

2.1.1 Neural Network

Nel campo del *Deep Learning* una rete neurale è un modello computazionale composto da neuroni artificiali, ispirato vagamente alla struttura di un neurone biologico. Per capire come funziona un neurone si introdurrà innanzitutto il concetto di classificatore lineare.
Supponendo di avere un database di addestramento composto da N immagini $x_i \in \mathbb{R}^D, i = 1, \ldots, N$ e che queste immagini debbano essere classificate in K classi diverse, allora il *dataset di training* è formato dalle coppie: (x_i, y_i) *dove* $y_i \in \{1, \ldots, K\}$ L'obbiettivo del classificatore è quello di definire una funzione $f : \mathbb{R}^D \rightarrow \mathbb{R}^K$ che mappa ogni immagine x_i ad un punteggio che indica quanto è probabile che l'immagine appartenga alla classe K.
Modellando un esempio del mondo reale, si consideri il dataset CIFAR-10 (4) che è composto da 60000 immagini RGB 32x32 appartenenti a 10 classi diverse.
Ogni immagine formata da 32x32x3 pixel può essere vista come un vettore colonna $x_i \in \mathbb{R}^{3072}$, si definisce dunque la funzione lineare f che produce il punteggio assegnato all'immagine x_i di appartenere alla classe considerata:

$$f(x_i, W, b) = Wx_i + b$$

dove :

- $W \in \mathbb{R}^{10 x 3072}$ è la matrice dei pesi.
- $b \in \mathbb{R}^{10}$ è il vettore dei scostamenti *(bias).*

Si può dedurre che l'obbiettivo sarà far imparare al classificatore, attraverso il *dataset* di addestramento, i parametri necessari per classificare una nuova immagine data in ingresso restituendo in uscita un punteggio che deve risultare più alto per la classe corretta mentre più basso per le altre classi.

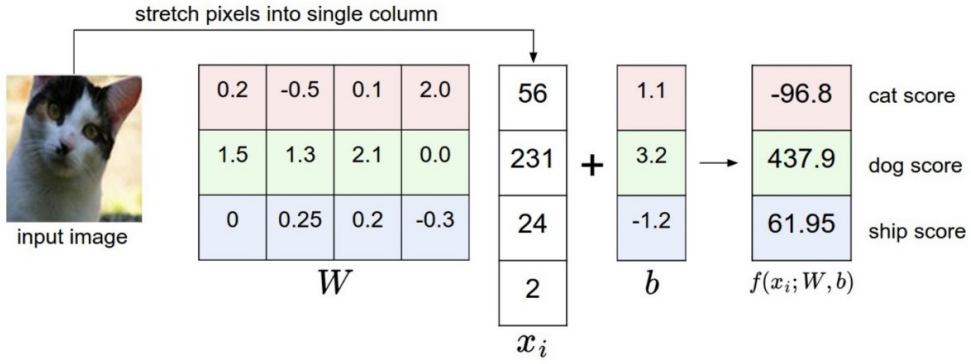

Illustrazione 3: Esempio funzionamento classificatore lineare.

Se ora a questo classificatore viene aggiunta una non linearità come ad esempio una *sigmoide σ,* abbiamo ottenuto quello che viene chiamato classificatore *logistic regression.*

$$f(x_i, W, b) = \sigma(Wx_i + b) \quad \text{dove} \quad \sigma(x) = f(x) = \frac{1}{1+e^{-x}} \tag{2.1}$$

Questo classificatore dovrà essere allenato a ottimizzare una funzione di costo chiamata *binary cross-entropy.*

$$L(W, b) = \frac{-1}{m} \sum_{i=1}^{m} \left(y_i \log(f_{x_i}) + (1 - y_i) \log(1 - f_{x_i}) \right) \tag{2.2}$$

dove m è il numero di esempi nel database di *training* e $f_{x_i} = f(x_i, W, b)$.

Un classificatore di questo tipo si presta bene per compiti di classificazione binaria mentre una sua generalizzazione multi-classe è data dal classificatore *softmax.*

La funzione :

$$softmax_j(z) = \frac{e^{z_j}}{\sum_k e^{z_k}} \tag{2.3}$$

prende in ingresso un vettore di punteggi z e non fa altro che portare questi punteggi in un intervallo che va da zero a uno e inoltre la somma dei punteggi restituiti è uguale a uno.

In questo caso durante il *training* invece verrà ottimizzata la funzione di costo chiamata *cross-entropy:*

$$L = -\sum_i y_i^{true} \log(y_i^{true}) \tag{2.4}$$

dove y_i^{true} è l'output reale e y_i^{pred} è l'output predetto dal classificatore.

Si può ora introdurre quello che nel mondo del *deep learning* è conosciuto come neurone.

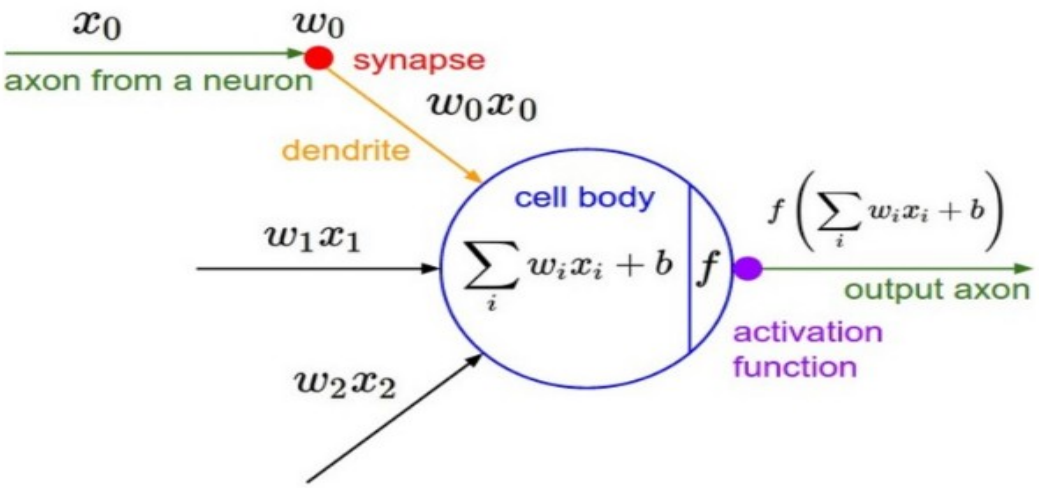

Illustrazione 4: Funzionamento neurone.

Ogni neurone può ricevere più di un ingresso per volta, ad ognuno di questi ingressi x_i viene moltiplicato un vettore di pesi corrispondente w_i e aggiunto un bias *b*. Al risultato di questa operazione viene poi applicata una non linearità (Es. Sigmoide (2.1)).

Si può facilmente intuire che un neurone a cui viene applicata la funzione di costo *cross-entropy* (2.4) non è altro che un classificatore *softmax* binario.

Spostando ora l'attenzione sulle funzioni di attivazione, ovvero le cosiddette non linearità, verranno elencate di seguito alcune delle funzioni di attivazione più utilizzate. La funzione *sigmoide* descritta dalla formula (2.1) non è l'unica possibilità:

- Relu $f(x)=\begin{cases}0, se\ x<0\\ x, se\ x\geq 0\end{cases}$ (2.5)

- LeakyRelu $f(x)=\begin{cases}\alpha x, se\ x<0\\ x, se\ x\geq 0\end{cases}$ (2.6)

- Tanh $f(x)=\frac{e^x-e^{-x}}{e^x+e^{-x}}$ (2.7)

L'architettura formata da un insieme di neuroni connessi tra loro in modo da creare un grafo diretto aciclico è soprannominata rete neurale.

Il primo strato di neuroni della rete è detto strato di ingresso mentre l'ultimo strato della rete è detto strato di uscita, tutti gli strati che si trovano tra lo strato di ingresso e lo strato di uscita sono detti strati nascosti.

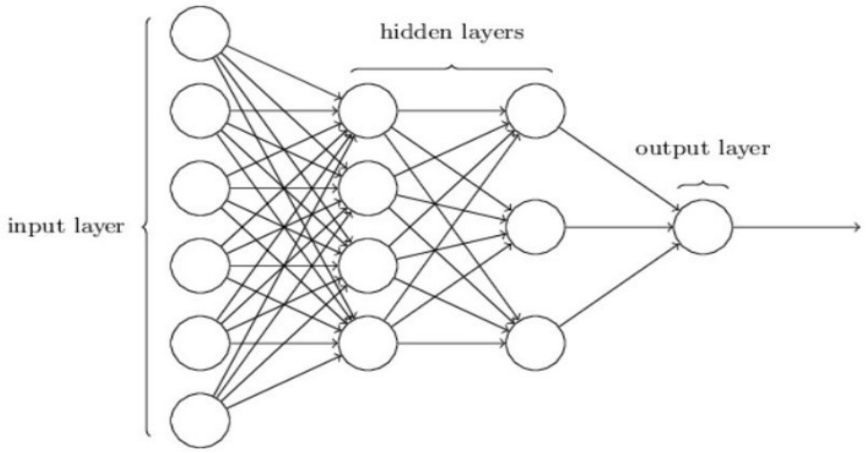

Illustrazione 5: Esempio architettura di una rete neurale.

Viene mostrato un classico esempio di rete *fully connected* nell'illustrazione 5. L'uscita di uno strato rappresenta l'ingresso dello strato successivo e non può mai essere data in ingresso a uno degli strati precedenti. Questo meccanismo permette agli ultimi strati della rete di estrarre *features* di più alto livello rispetto a quelle che possono essere estratte dagli strati iniziali della rete.

L'aggiornamento dei pesi dei singoli neuroni in questa architettura è un metodo matematico chiamato *backpropagation (5)* e si riferisce alla propagazione all'indietro del gradiente, ovvero propagare dall'uscita all'ingresso errore commesso dalla rete.

La potenza di queste reti è data dal fatto che hanno la certezza di poter approssimare qualsiasi funzione matematica con lo svantaggio che all'aumentare della complessità della rete aumenta anche il numero di pesi che vanno imparati necessitando dunque di una elevata potenza computazionale per poterli gestire. Questo rappresenta uno dei principali limiti delle reti *fully connected.*

2.1.2 Convolutional Neural Networks

Le *Convolutional Neural Networks (CNN)* nascono per risolvere alcuni problemi che insorgono con l'utilizzo delle reti *fully connected.* Come visto in precedenza le reti *fully connected* hanno un enorme numero di parametri da imparare all'aumentare della complessità della rete. Ciò può creare, soprattutto per le reti molto profonde, problemi di gestione dovuti alla grande quantità di tempo necessaria per il *training* e dovuti necessità di avere molta RAM a disposizione per controllare l'enorme volume di dati generato.

Inoltre, le reti *fully connected* non riescono a catturare in pieno una particolare caratteristica presente in alcune tipologie di dati ovvero la coerenza spaziale.

Prendendo come esempio, un immagine di un bambino, è molto probabile che i pixel della parte superiore che rappresentano il viso, siano correlati tra loro, e abbiano quindi caratteristiche simili come il colore, mentre è molto meno probabile che siano correlati ai pixel della parte inferiore che rappresentano le gambe e i piedi. Questa proprietà è detta coerenza spaziale ed è presente in immagini, video, audio, dati testuali, e talvolta nelle serie temporali.

Uno studio effettuato negli anni ‘60 da Hubel e Wiesel (6) ha messo in evidenza come la corteccia visiva nel cervello contiene neuroni che individualmente rispondono solo a una piccola regione del campo visivo, da questa scoperta nel 1982 Kunihiko Fukushima cercando di riprodurre questa caratteristica ha introdotto le reti convolutive (7), che sono state poi allenate tramite *backpropagation* (8) da Yann LeCun dopo qualche anno.

Le CNN sono molto simili alle reti *fully connected* infatti:

- Sono composte da neuroni con pesi e *bias* appresi durante il *training.*
- Ogni neurone riceve ingressi multipli, che vengono moltiplicati con i rispettivi pesi interni *W* e successivamente, se richiesto, viene applicata una funzione di attivazione.
- L’intera rete convolutiva esprime una funzione matematica differenziabile.

La differenza principale con la struttura del neurone classico risiede nel fatto che ogni neurone convolutivo è dotato di un filtro utilizzato per compiere una operazione di cross-correlazione sulla matrice d’ingresso che produce il corrispondente output.

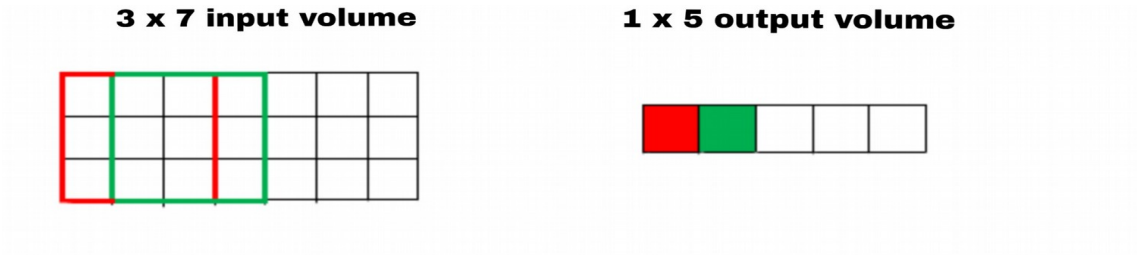

Illustrazione 6: Funzionamente Convolutional Neural Network.

Come mostrato nell’illustrazione 6 per costruire la matrice di uscita viene fatto scorrere sulla matrice di ingresso, lungo un asse, il filtro convolutivo.
Formalmente l’uscita y viene calcolata come segue:

$$y_{a,b}=f\left(\sum_{i=0}^{s_f-1}\sum_{j=0}^{n-1} w_{i,j}^{b}\, x_{a+i,j}+b^{b}\right) \qquad (2.8)$$

Dove:

- x è la matrice di input di dimensione m x n.
- f è la funzione di attivazione.
- $w_{i,j}^{b}$ è il peso nella posizione i,j del b-esimo filtro.
- b^{b} è il bias del b-esimo filtro.

- $s_f \cdot n$ è la dimensione dei filtri utilizzati.

Siccome questa convoluzione scorre lungo un singolo asse è chiamata convoluzione monodimensionale (1D) ed è quella che si usa per serie numeriche o testi, mentre siccome le immagini hanno una dimensione in più dovuta ai canali RGB, per gestire immagini e video è utilizzata la convoluzione 2D o 3D.

In questo caso il numero di parametri da imparare è ottenuto moltiplicando la dimensione dei filtri per il numero totale dei filtri e non dipende dalle dimensioni dell'input. Differente invece era la situazione delle reti *fully connected* dove il numero di parametri da imparare è legato alla dimensione dei dati in ingresso diventando facilmente enorme.

2.1.3 Long Short Term Memory

Le reti ricorrenti sono particolari reti neurali che cercano di riprodurre alcune caratteristiche dell'apprendimento umano. L'uomo non ricomincia a pensare da zero ad ogni istante durante lo svolgimento di un compito ma memorizza passo dopo passo le informazioni necessarie per continuare le sue azioni, per esempio durante la lettura di un testo, il cervello comprende il contesto di ogni parola basandosi su ciò che è stato imparato dalle parole lette in precedenza, quindi non butta via tutte le informazioni acquisite quando legge una nuova parola, ma memorizza ciò che è necessario per proseguire la lettura.

Le tradizionali reti neurali viste fino ad ora pur cercando di riprodurre alcuni comportamenti del cervello umano non sono in grado di memorizzare nel tempo.

Le reti ricorrenti nascono esattamente per sopperire a questa mancanza e sono particolari reti neurali che contengono al loro interno dei cicli per permettere all'informazione di persistere nel tempo.

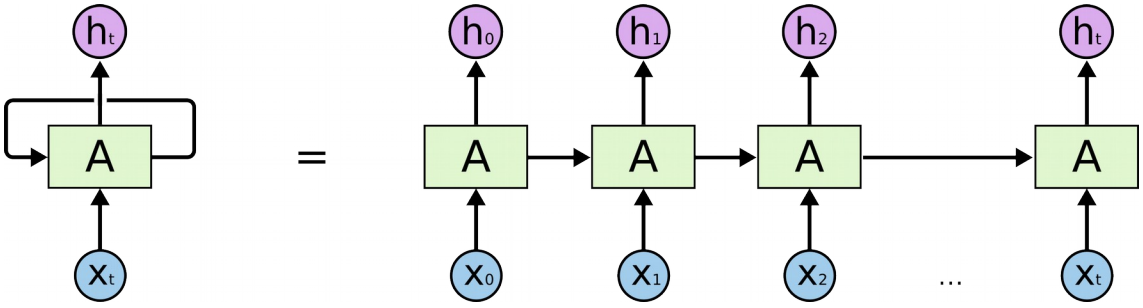

Illustrazione 7: Rete neurale ricorrenti con cicli al suo interno.

Una rete ricorrente può essere pensata come una copia ripetuta della stessa rete, che passa le informazioni dal primo all'ultimo step. In figura 7, x_t rappresenta l'ingresso della rete e si può notare come questa struttura si adatta bene a dati in ingreso come sequenze o liste.

Infatti negli ultimi anni le reti ricorrenti sono state utilizzate in vari problemi come: riconoscimento del parlato, traduzioni, riconoscimento del contesto nei video e nelle serie temporali.

Le *Long Short Term Memory,* LSTM, introdotte nel 1997 da Hochreiter e Schmidhuber (9), sono delle speciali reti ricorrenti capaci di memorizzare informazione a lungo termine.

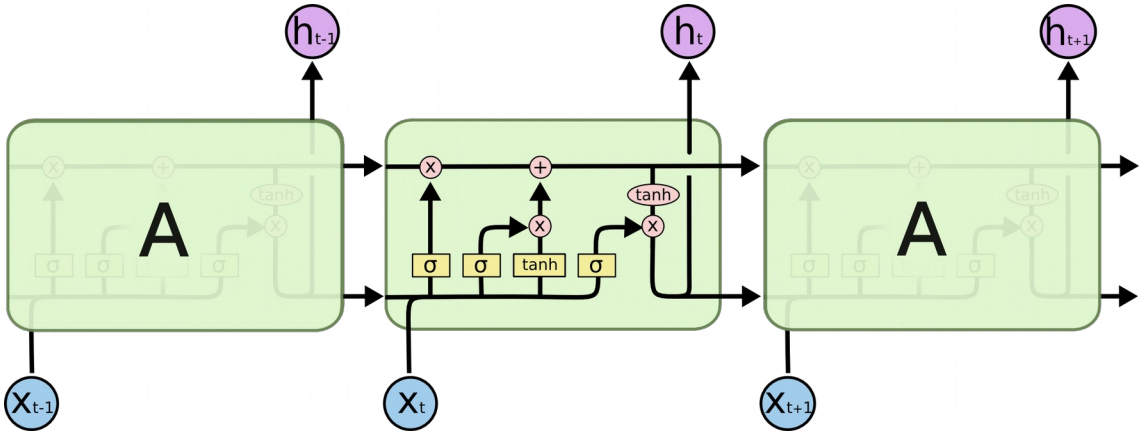

Illustrazione 8: Struttura interna di una cella LSTM.

Il primo passo all'interno di una cella LSTM è decidere quanta informazione decidiamo di dimenticare dallo stato della cella precedente h_{t-1}.
Questa operazione è affidata al primo *layer* all'interno alla cella LSTM chiamato *forget gate layer ft.*

$$f_t = \sigma(W_f[h_{t-1}, x_t] + b_f) \tag{2.9}$$

Il prossimo passo è quello di decidere cosa delle nuove informazioni in ingresso x_i andiamo a memorizzare nello stato della cella corrente h_t , questa operazione viene effettuata in due parti.
La prima parte è affidata a quello che è chiamato *input gate layer* che tramite una funzione sigmoide decide quale valori saranno aggiornati all'interno della cella.

$$i_t = \sigma(W_i[h_{t_{-i}}, x_t] + b_i) \tag{2.10}$$

Successivamente tramite un *layer* che utilizza la tangente iperbolica come funzione di attivazione, si crea un vettore di valori candidati per l'aggiornamento della cella.

$$\widetilde{C}_t = \tanh(W_c[h_{t-1}, x_t] + b_c) \tag{2.11}$$

Ora finalmente si può andare ad aggiornare il nuovo stato della cella, moltiplicando l'uscita del *forget gate* f_i con lo stato precedente della cella ovvero C_{t-1} e poi sommando il prodotto tra i_t e $\widetilde{C}_t$, formalmente:

$$C_t = f_t * C_{t_1} + i_t * \widetilde{C}_t \tag{2.12}$$

Rimane ora da determinare quale sarà l'output della cella LSTM. L'uscita si baserà sullo stato della cella ma sarà solo una versione filtrata di esso, quindi innanzitutto si decide tramite un ulteriore sigmoide quali parti dello stato della cella saranno utilizzate come output.

$$o_t = \sigma(W_o[h_{t-1}, x_t] + b_o) \tag{2.13}$$

L'ultimo passo è quello di calcolare la tangente iperbolica dello stato della cella corrente e moltiplicarlo per l'uscita del *output gate layer* o_t in modo da restituire in uscita solo l'informazione opportunamente filtrata contenuta nel nuovo stato.

$$h_t = o_t * \tanh(C_t) \tag{2.14}$$

2.1.4 Generative Adversarial Networks

Le *Generative Adversarial Networks, GAN,* sono una delle più recenti e innovative architetture nel mondo del *deep learning,* composte da due reti neurali che competono una contro l'altra.

Sono state introdotte nel 2014 da Ian Goodfellow (10) e sono state definite da Yann LeCun, il direttore del dipartimento di ricerca sull'intelligenza artificiale di Facebook, "The most interesting idea in the last 10 years in ML". Il potenziale della GAN è veramente alto siccome ciò che fa, è imparare a replicare una distribuzione che modella i dati in ingresso, per poter generare nuovi dati sintetici con le stesse caratteristiche di quelli reali campionando da essa. La GAN può essere utilizzata in svariati domini, come ad esempio: immagini, musica, video, sequenze numeriche ecc.

All'interno della GAN una prima rete, chiamata generatore, genera nuove istanze di dati mentre la seconda rete, il discriminatore, valuta l'autenticità dei dati che riceve decidendo se essi appartengono alla distribuzione dei dati reali provenienti dal *dataset* di *training* oppure no.

L'esempio classico che si fa per capire il funzionamento di una GAN è quello di voler generare immagini di numeri scritti a mano, come quelli che si possono trovare nel MNIST (11), un database creato per esperimenti di questo tipo nel l'ambito *deep learning*. Il generatore prende in ingresso del rumore casuale e da esso deve generare un'immagine, questa immagine viene impacchettata insieme ad altre immagini generate (etichettate con il valore 0) e a immagini appartenenti ai dati reali (etichettate con il valore 1) e data in pasto al discriminatore.

Il discriminatore prende queste immagini reali e generate e ritorna per ognuna di esse la probabilità che l'immagine in ingresso sia reale o meno, ovvero restituisce un numero compreso tra 0 e 1.

Il discriminatore epoca dopo epoca, quindi impara a discriminare verificando tramite le label (valore di *ground-truth)* assegnate ad ogni immagine, se la probabilità che ha predetto sia corretta o meno, mentre il generatore riceve un *feedback* di quanto le immagine che genera siano realistiche, dal numero di volte che riesce a ingannare il discriminatore.

Ancora, si può immaginare il generatore come un contraffattore di banconote, e il discriminatore come un poliziotto addestrato a riconoscere le banconote reali da quelle false. Il generatore col tempo si specializzerà nel tentare di ingannare il discriminatore, mentre il discriminatore diventerà sempre più bravo a riconoscere le banconote false.

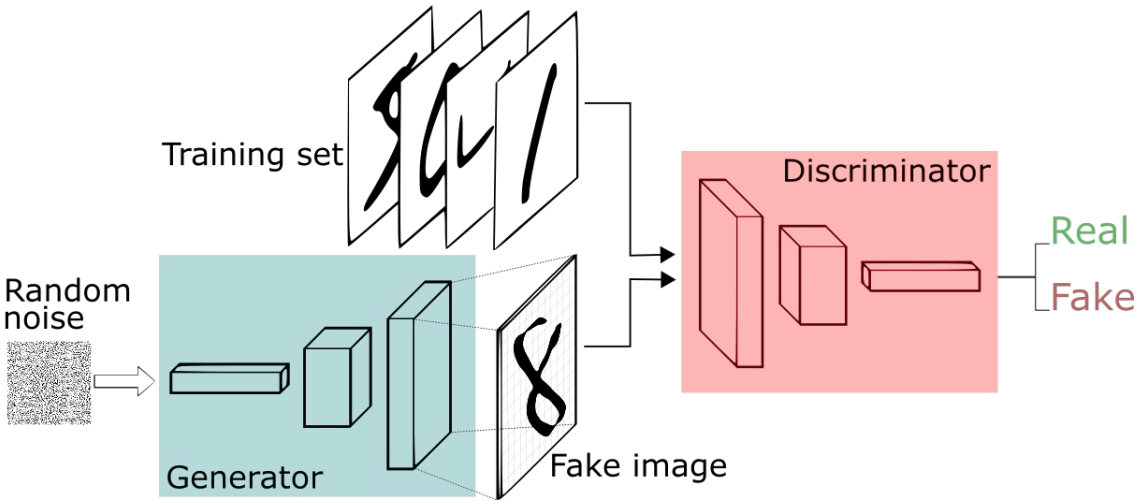

Illustrazione 9: Esempio Architettura GAN che genera immagini di numeri scritti a mano.

Le due reti cercano di ottimizzare differenti e opposte funzioni di costo in un gioco minimax:

$$\min_G \max_D V(D,G) = E_{P_{Data}(x)}[\log D(x)] + E_{P_z(z)}[\log(1 - D(G(z)))] \qquad (2.15)$$

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) viene indicato l'elemento campionato dalla distribuzione appresa dal generatore dato l'input z e con D(x) si indica la probabilità di appartenenza al *dataset* originario dell'elemento x, che il Discriminatore stima per il campione x. Il rumore z è utilizzato per ottenere variabilità nell'output generato.

2.1.5 Modello Sequence-to-Sequence

Il modello Sequence-to-Sequence è impiegato in numerosi sistemi utilizzati ogni giorno, l'applicazione principale che può venire in mente è sicuramente quella di *Google Translate*, ma non solo, è un modello usato per il riconoscimento del linguaggio parlato (esempio l'assistente vocale di Google), oppure è utilizzato per quello che viene chiamato *video-captioning* cioè descrivere un video in modo testuale.

Viene introdotto da Google nel 2014 (12) ed è un modello che ha come obbiettivo quello di mappare un ingresso di lunghezza fissata, in un output che non deve necessariamente avere la stessa lunghezza dell'input.

Per esempio nei problemi di traduzione l'input difficilmente ha la stessa dimensione dell'output, se si traduce la frase "Cosa fai oggi ?" in cinese l'ingresso di 3 parole si trasforma in un output di 7 simboli "今天你在做什麼" e chiaramente non può essere usata una rete ricorrente per mappare ogni frase in italiano in una frase in cinese.

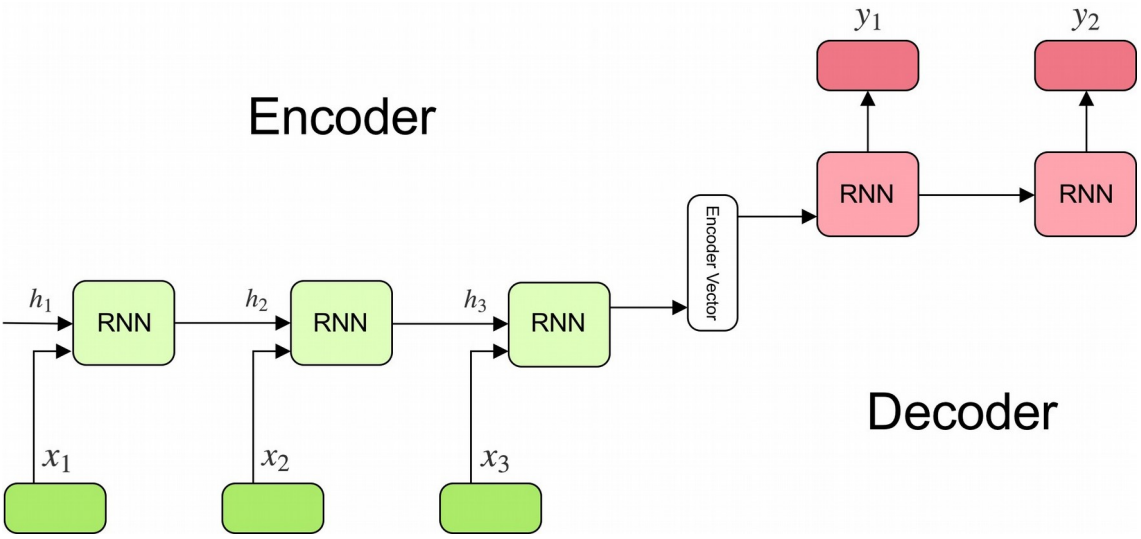

Illustrazione 10: Esempio modello Sequence-to-Sequence.

Come mostrato in figura 10 il modello *Sequence-to-Sequence* è composto da un *encoder*, che comprime le informazioni di input in un vettore di *encoding*, questo vettore è poi utilizzato dal *decoder* per generare la nuova sequenza desiderata.

L'*encoder* è formato da una sequenza di unità ricorrenti, come ad esempio LSTM, dove, ognuna di esse accetta un singolo elemento della sequenza di input (*time-step)*, ricavando informazioni da questo elemento e propagando l'informazione allo *step* successivo. Per esempio, se l'ingresso è una frase in lingua italiana, ogni cella riceverà una singola parola seguendo l'ordine imposto dalla frase.

L'*encoder* incapsula tutta l'informazione estrapolata dall'ingresso in un vettore che è utilizzato dal *decoder* per fare una predizione accurata.

Il vettore di *encoding* è utilizzato come ingresso della prima cella ricorrente del *decoder.* Il *decoder* è formato anch'esso da una sequenza di unità ricorrenti che producono l'*output step by step.*

Ognuno degli stati interni delle celle ricorrenti del *decoder,* viene inizializzato con lo stato della cella precedente mentre la prima cella è inizializzata con lo stato interno dell'ultima cella dell'*encoder*. Ogni cella del *decoder* riceve in ingresso l'uscita della cella precedente. Quindi se il *decoder* deve restituire una frase tradotta, ogni parola o simbolo è l'output di una singola unità. La potenza del modello risiede proprio nel fatto che non c'è correlazione tra la lunghezza dell'ingresso e quella dell'uscita.

2.1.6 Attention

Il meccanismo dell'*attention* nasce per aiutare le reti neurali a memorizzare lunghe frasi nel campo della *neural machine translation (13).* È un meccanismo che viene utilizzato soprattutto in simbiosi con i modelli S*equence-to-Sequence* e ha come idea di dare una differente importanza agli elementi della sequenza data in ingresso anziché ritenerli tutti di ugual valore, questo fa si che il modello si focalizzi di più sui dati ritenuti rilevanti.

L'obbiettivo dell'*attention* quindi è mettere in evidenza i concetti fondamentali dai dati in ingresso, e non di focalizzarsi su tutta l'informazione ricevuta, ciò non solo aiuta il modello ad imparare meglio, ma permette anche di capire quali sono le *features* necessarie e quali invece sono poco importanti per il compito richiesto (*features selection).*

Anche nel campo della visione artificiale viene utilizzata l'*attention,* permettendo ai modelli utilizzati di concentrarsi di più su una particolare regione dell'immagine piuttosto che un'altra.

Esistono vari metodi per calcolare l'*attention,* di seguito verrà descritto come opera il meccanismo *attention* utilizzato durante il progetto di tesi.

Si introduce una matrice di attenzione *A* che cattura la similarità che c'è tra un elemento di una sequenza ingresso e i suoi elementi vicini.

Si consideri $e_{t,t'} \in A$ l'elemento che cattura la somiglianza tra la rappresentazione dello stato nascosto h_t , nel modello S*equence-to-Sequence,* riferita all'ingresso x_t al tempo t e la rappresentazione dello stato nascosto $h_{t'}$ riferita all'ingresso $x_{t'}$ al tempo *t'*.

Quanto finora descritto, viene implementato in modo simile a una cella LSTM:

$$g_{t,t'} = \tanh\left(W_g h_t + W_{g'} h_{t'} + b_t\right) \tag{2.16}$$

$$e_{t,t'} = \sigma\left(W_a g_{t,t'} + b_a\right) \tag{2.17}$$

Dove σ è la funzione di attivazione sigmoide (2.1), W_g e $W_{g'}$ sono i pesi delle matrici corrispondenti agli stati nascosti h_t e $h_{t'}$ calcolati dal modello Sequence-to-Sequence agli istanti t e t', W_a è la matrice dei pesi appresa dal *layer* di *attention*, b_t e b_a sono i vettori di bias.

Ora si applica un'operazione di *softmax (2.3)* per portare tutti i coefficienti appartenenti alla matrice *A* tra 0 e 1 e si ottengo i valori dei coefficienti di *attention*.

$$a_{t,t'} = softmax\left(e_{t,t'}\right) \tag{2.18}$$

Come visto in precedenza il *decoder* del modello S*equence-to-Sequence* prende in input un vettore che racchiude tutte le informazioni utili che l'*encoder* è riuscito ad estrapolare dai dati.

Quindi l'ultimo passo che compie il meccanismo dell'*attention* è fare un prodotto elemento per elemento tra gli stati nascosti della sequenza e i coefficienti di *attention* ottenendo il vettore l_t che verrà poi passato come ingresso al *decoder*:

$$l_t = \sum_{t'=1}^{n} a_{t,t'} h_{t'} \tag{2.19}$$

2.2 Mercato azionario

In questa sezione verrà spiegato l'ambiente in cui, il modello realizzato durante il progetto di tesi andrà a lavorare, verrà quindi fatta un'introduzione su cosa sono i mercati azionari, come vengono analizzati, come è possibile per gli investitori ricavare guadagno, verrà introdotta la teoria dei mercati efficienti, e saranno presentati i titoli azionari utilizzati.

Azioni, società quotate in borsa, indici al ribasso, sono termini di cui spesso si sente parlare ma di cui molti non conoscono il vero significato.

Il mercato azionario è la piazza telematica in cui avviene un enorme numero di contrattazioni che hanno come oggetto azioni, ossia delle quote di capitale sociale. Fanno parte di questo luogo non fisico i compratori e i venditori di azioni.

Rinunciando a quote della società, i venditori di azioni ricevono benefici finanziari per permettere all'impresa di fare nuovi investimenti, raccogliendo capitali e favorendo l'ingresso di nuovi soci, evitando quindi prestiti bancari, ed è questa la principale motivazione per cui esistono le azioni. Gli investitori acquistando azioni ottengono la possibilità di introiti se vengono emessi fondi per i proprietari di esse, hanno la possibilità di rivendere a loro volta queste azioni e hanno il diritto di partecipare all'amministrazione dell'impresa.

Occorre però distinguere il mercato primario dal mercato secondario. Il mercato primario è riservato agli strumenti di nuova emissione, ovvero sono oggetto di

transazioni sul mercato primario quegli strumenti che derivano da un aumento di capitale o da un'offerta pubblica iniziale, cioè l'offerta fatta da chi vuole acquistare azioni di una società che intende quotarsi in borsa. Si va dall'emissione di obbligazioni per le società private fino a titoli di stato. Le azioni acquistate sul mercato primario sono quelle che permettono alle imprese di ottenere un guadagno.

Il mercato secondario invece è oggetto di negoziazione di titoli già presenti sul mercato e gli scambi avvengono solo tra investitori e quindi non portano guadagno all'impresa.

In questo caso gli investitori guadagnano se le azioni hanno una grande richiesta e quindi il loro valore sale. Se il loro valore sale di molto l'azienda può immettere nuove azioni sul mercato primario ad un prezzo maggiore ma ne l'investitore ne l'azienda può avere la certezza che questo succeda.

Riassumendo, se un investitore compra un nuovo titolo direttamente da una azienda che ha deciso di vendere quote della società parliamo di mercato primario mentre se un investitore compra titoli già presenti nel mercato azionario da un altro investitore parliamo di mercato secondario.

2.2.1 Mercato al rialzo

É probabilmente la forma di mercato più conosciuta, dove un investitore compra dei titoli nella speranza che essi possano aumentare il loro valore nel tempo per poi rivenderli ad un prezzo più alto.

Detto anche *bull market*, il processo che porta al guadagno è abbastanza semplice, ad esempio: un investitore compra azioni di una società che pensa sia in crescita, il prezzo attuale delle azioni è di 5€ l'una e l'investitore ne compra 100 ad un prezzo totale di 500€, dopo un mese il loro valore delle azioni è salito ad un prezzo di 7€ l'una, l'investitore a questo punto vende le sue azioni al prezzo totale di 700€.

Se al contrario, dopo un mese le azioni hanno un valore di 3€ l'una, ora ciò che l'investitore può ricavare è solo 300€.

2.2.2 Mercato al ribasso

Detto anche *bear market*, consiste in un approccio differente al mercato da parte degli investitori, che applicano una vendita allo scoperto delle azioni, e quindi sperano che il valore delle azioni cali nel tempo.

Il processo è leggermente più complicato rispetto al mercato al rialzo, l'investitore che pensa che il prezzo di un'azione possa calare nel tempo chiede in prestito azioni a un intermediario, per esempio una banca, per poi rivenderle istantaneamente.

Quando il prezzo cala, l'investitore ricompra lo stesso numero di azioni vendute inizialmente ma ad un prezzo minore, per saldare il proprio debito con l'intermediario generando profitto.

Ad esempio un investitore prende in prestito 100 azioni, tramite intermediario, di una società al valore di 5€ e le rivende immediatamente ottenendo 500€, dopo un mese il valore delle azione cala a 3€, l'investitore ora ricompra le 100 azioni per un prezzo totale di 300€ per restituirle all'intermediario generando un profitto di 200€.

Se invece dopo un mese il valore delle azione sale a 7€ l'una, allora l'investitore dovrà spendere 700€ per saldare il suo debito, perdendo quindi capitale.

2.2.3 Teoria dei mercati efficienti

Formulata da Eugene F.Fama (14) è una teoria che afferma che un mercato è detto efficiente quando i prezzi riflettono completamente tutte le informazioni disponibili, ipotizzando tre tipi di efficienza: in forma debole, in forma semi-forte e in forma forte.

Un mercato è definito efficiente in forma debole se il prezzo di un titolo incorpora in sé tutte le informazioni date dalla serie storica dei prezzi. Un mercato è detto invece efficiente in forma semi-forte se il prezzo del titolo incorpora tutta l'informazione data dalla serie storica dei prezzi e dalle informazioni pubbliche disponibili in ogni istante. Un mercato è detto efficiente in forma forte se il prezzo del titolo incorpora in se le informazione della serie storica, l'informazione disponibile pubblicamente ad ogni istante della serie, e le informazioni private. Questo tipo di formulazione non esclude la presenza di anomalie nei ritorni, purché esse siano normalmente distribuite.

2.2.4 Teoria del Random Walk

É una teoria in stretta correlazione con la teoria dei mercati efficienti che afferma che i prezzi delle azioni variano in modo completamente casuale e senza una vera correlazione con quello che è avvenuto nel passato.

Il modello di *random walk* più utilizzato è quello gaussiano, dove ogni valore *p(t)* di una serie continua rappresentante un prezzo di una azione, è legato a quello precedente come segue:

$$p(t)=p(t-1)+x(t) \tag{2.20}$$

Dove *p(t-1)* è il valore del prezzo all'instante precedente e *x(t)* è il valore assunto dalla variabile aleatoria $X \sim N(\mu,\sigma)$.

La densità di probabilità della variabile aleatoria è una gaussiana con media 0 quindi:

$$f(x,\sigma^2)=\frac{1}{\sigma\sqrt{2\pi}}e^{-\frac{(x)^2}{2\sigma^2}} \quad (2.21)$$

Questo sottolinea l'impossibilità di prevedere una serie finanziaria analizzando tutta la sua storia precedente. Secondo questa teoria quindi un investitore di successo non usa tecniche avanzate di analisi ma il fatto che il suo investimento vada a buon fine dipende solo dal caso. Mentre considerando gli investitori di maggior successo, data l'esistenza di milioni di investitori è probabile che alcuni di essi possano effettuare una serie di scelte 'fortunate' in modo tale da trarre un alto profitto.

2.2.5 Indici azionari

Per descrivere l'andamento dei mercati azionari, o una parte di essi, si usano gli indici. Gli indici azionari sono la sintesi del valore del paniere di titoli azionari che rappresentano. Un paniere di titoli azionari è semplicemente un insieme di strumenti finanziari opportunamente scelti per rappresentare l'andamento di una sezione di mercato. I titoli che sono stati utilizzati per allenare il modello *GAN* proposto, sono i seguenti: FTSE MIB, FTSE 100, S&P 500, CAC 40, SP&T/SX 60, SMI20, DAX30, IBEX35, STOXX 50 e TOPIX.

I dati utilizzati sono stati messi a disposizione da Axyon Ai e verranno presentati brevemente di seguito.

FTSE MIB

Financial Times Stock Exchange Milano Indice di Borsa, FTSE MIB, è il più significativo indice azionario della borsa italiana, racchiude di norma le azioni delle 40 società italiane più importante, anche se queste hanno sede all'estero tra cui per esempio: ENEL, ENI, UNICREDIT, FERRARI ecc.

Illustrazione 11: Andamento FTSE MIB.

FTSE 100

Financial Times Stock Exchange, FTSE 100, è l'indice azionario inglese che rappresenta le 100 società più capitalizzate, quotate al London Stock Exchange tra cui: BARCLAYS, EASYJET, ADMIRAL, VODAFONE ecc.

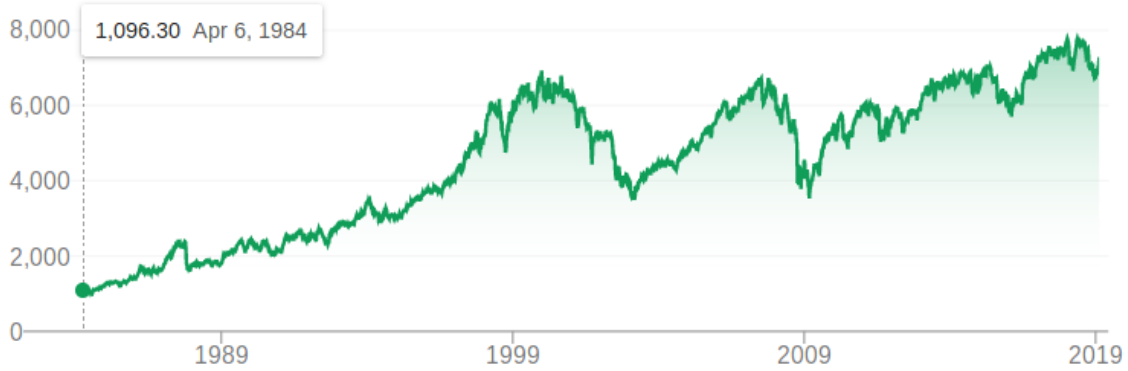

Illustrazione 12: Andamento FTSE 100.

S&P 500

Standard & Poor's 500, S&P 500, nasce nel 1957 ed l'indice che segue l'andamento di un paniere azionario formato dalle 500 aziende statunitensi a maggiore capitalizzazione tra cui: AMAZON, COCA-COLA, INTEL, PAYPAL ecc.

Illustrazione 13: Andamento S&P 500.

CAC 40

Cotation Assistèe en Continu, CAC 40, è il principale indice della borsa francese e prende i 40 valori più significativi tra le 100 maggiori capitalizzazioni della borsa di Parigi, tra le più importanti: RENAULT, PEGEOUT, L'OREAL ecc.

Illustrazione 14: Andamento CAC 40.

S&P/TSX 60

Standard & Poor's Toronto Stack Exchange, S&P/TSX 60 è un'unità di *Standard & Poor's,* ed è l'indice azionario delle 60 aziende più quotate in Canada tra cui: BLACKBERRY, THOMSON REUTERS, METRO ecc.

Illustrazione 15: Andamento S&P/STX 60.

SMI 20

Swiss Market Index, SMI 20, è il più importante indice svizzero che, introdotto nel 1988, comprende le 20 società svizzere con maggiore capitalizzazione tra cui: ADECCO, NESTLE, ZURICH ecc.

Illustrazione 16: Andamento SMI 20.

DAX 30

Deutsche Aktie index 30, DAX 30, è il segmento della borsa di Francoforte contenente i 30 titoli a maggiore capitalizzazione tedeschi. Tra cui: BMW, BAYER, ADIDAS, SIEMENS, SAP ecc.

Illustrazione 17: Andamento DAX 30.

IBEX 35

Ìndice Bursàtil Espanol, IBEX 35, è il paniere dei 35 titoli azionari più importanti di Spagna, inaugurato nel 1992 comprende importanti società tra cui: BBVA, REPSOL, BANCO SANTADER ecc.

Illustrazione 18: Andamento IBEX 35.

STOXX 50

L'EURO STOXX 50 è un indice azionario di titoli dell'euro-zona introdotto nel 1998 contenente società di rilievo europee come: ADIDAS, ENI, ING, BMW ecc.

Illustrazione 19: Andamento STOXX 50.

TOPIX

Tokyo Stock Price Index, TOPIX, è un indice giapponese che tiene traccia di tutte le compagnie nazionali tra cui: CANON, PANASONIC, JR CENTRAL, NISSAN ecc.

Illustrazione 20: Andamento TOPIX.

3. GAN per l’analisi del mercato azionario

In questo capitolo, l’obbiettivo sarà introdurre in maniera esaustiva il problema che è stato affrontato durante il progetto di tesi, si passerà poi ad esporre lo studio di fattibilità per dimostrare che il *deep learning* e in particolare le GAN sono utilizzabili per analizzare il mercato azionario, scomponendo l’analisi sostanzialmente in due parti; la prima parte dove verrà analizzato il discriminatore, e la seconda parte dove verrà preso in esame il generatore.

Successivamente si introdurrà la metrica che è stata scelta ed utilizzata per valutare quanto il risultato prodotto dalla GAN sia valido, infine si descriveranno le principali problematiche affrontate e le rispettive soluzioni applicate.

3.1 Definizione del problema

L’obbiettivo di questo progetto di tesi è generare scenari futuri multipli il più verosimili possibile per ogni indici azionario presentato in ingresso. In particolare si vuole progettare uno strumento in grado di generare i 5 prezzi futuri di un indice (una settimana), avendo in ingresso: la serie storica dei prezzi dell’indice nei 120 giorni precedenti (6 mesi), e 29 indicatori tecnici calcolati giorno per giorno. Inoltre si vuole modellare uno strumento totalmente agnostico, ovvero non specializzato su un singolo indice ma in grado di generalizzare e ricevere in ingresso un qualunque indice azionario senza dover utilizzare un’ulteriore modello parallelo.

Avendo a disposizione un database con le sequenze storiche reali a partire dal 3 gennaio del 2000 fino al 29 dicembre del 2017 degli indici descritti nella sezione 2.2.5, il problema non viene definito strettamente di tipo *supervised*, ovvero dato un certo ingresso vogliamo mapparlo nell’uscita desiderata, perché quello che si vuole ottenere è un paniere di scenari futuri probabili e non solamente quello che poi si andrà a verificare realmente, con lo scopo ultimo di capirne la distribuzione e permetterne un’analisi dettagliata.

Le GAN contrariamente sono pensate per risolvere problemi di tipo *unsupervised*, ovvero non si ha un preciso *output* per mappare i dati in ingresso ma si vuole imparare la distribuzione di quest’ultimi, da cui poi la GAN potrà campionare nuovi esempi che saranno differenti da quelli forniti ma avranno stesse proprietà.

Ciò che si è pensato, è stato di condizionare la GAN con i 29 indicatori tecnici dei giorni precedenti per generare il futuro, ovvero si vuole far in modo che la GAN non generi futuri campionati in modo casuale dalla distribuzione che impara ma che possa condizionare il proprio campionamento da ciò che riceve in ingresso. Possiamo parlare quindi *Context-Conditional GAN, CCGAN (15).*

3.2 Studio di fattibilità

Nella prima parte del progetto si è preso in considerazione generatore e discriminatore singolarmente andando a studiare se fossero effettivamente in grado di adempiere ai propri compiti.

Questa sezione verrà scomposta in due parti: la prima che prende in esame il discriminatore e la seconda che analizza il generatore.

3.2.1 Discriminatore

Sono molti i dubbi sull'effettiva possibilità di distinguere una serie reale da una generata, infatti se la teoria del R*andom Walk* descritta nella sezione 2.2.4 fosse vera sarebbe inutile fornire in ingresso i 120 giorni precedenti per generare i possibili scenari futuri, siccome tutta l'informazione utile per prevedere il prezzo successivo sarebbe racchiusa nell'ultimo valore della serie.

Inoltre, se il discriminatore fosse in grado di distinguere le sequenze generate, quanti valori dovrebbero essere aggiunti alla sequenza reale per permettere al discriminatore di essere capace di discriminare?

È facilmente intuibile che una sequenza di 121 valori, dove 120 sono reali e uno solo sintetico sarebbe molto difficile da riconoscere mentre una sequenza composta interamente da dati sintetici potrebbe essere un compito più semplice.

E ancora, allenare il discriminatore su indici azionari diversi ovvero su dati che hanno diverse distribuzioni pur avendo alcune caratteristiche comuni è possibile o il discriminatore non sarebbe in grado di generalizzare?

Per rispondere a queste domande è stato creato un esperimento su misura dove si è allenato un classificatore lineare, che prende in ingresso solo i prezzi delle azioni e non gli indicatori tecnici, a distinguere serie reali da serie generate da un generatore *Conditional Random Walk*, che per costruire uno scenario futuro calcola la media μ e deviazione standard σ della sequenza fornita in ingresso e campiona da una gaussiana centrata su μ e con deviazione standard σ i successivi elementi della sequenza.

Per semplicità è stata presa in considerazione una serie storica di lunghezza pari a 100 ed è stata fatta un esplorazione completa partendo con la generazione di un solo prezzo sui 100 totali, fino a generare 99 prezzi futuri partendo dal primo singolo valore reale.

Quello che emerge, come è possibile vedere in figura 21, è che all'aumentare della lunghezza della sequenza generata, aumenta l'*accuracy* del classificatore.

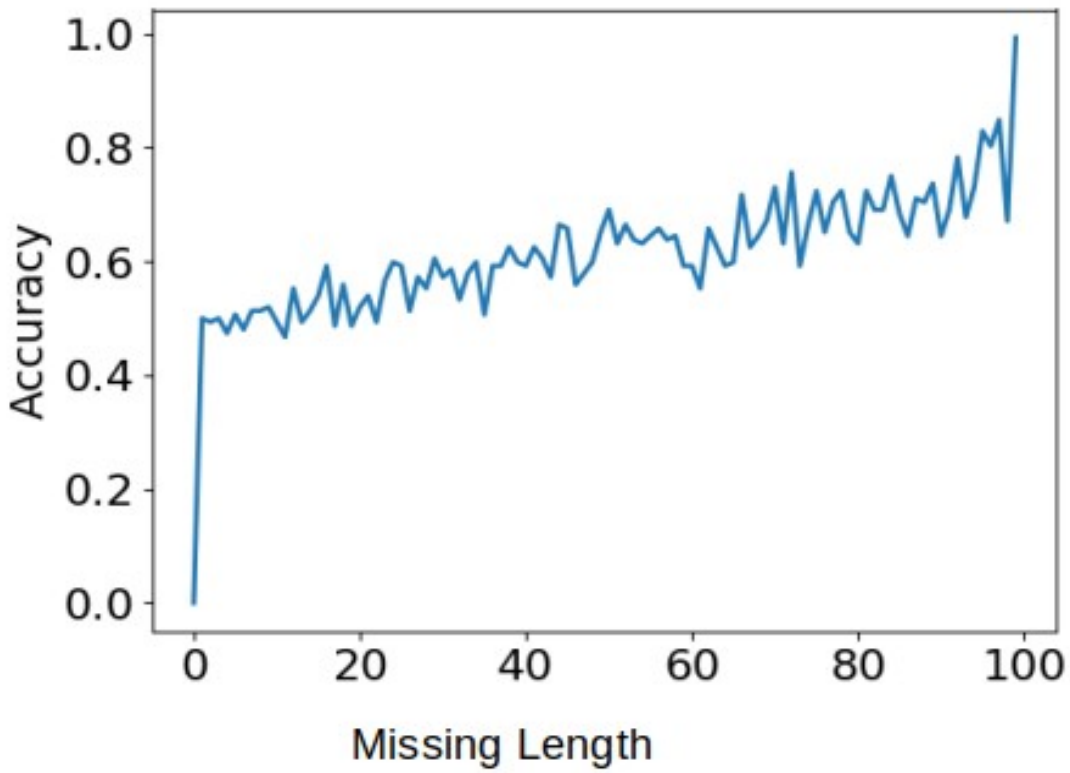

Illustrazione 21: Risultato del primo test, sull'asse y l'accuracy del classificatore e su l'asse x il numero di valori sintetici all'interno della sequenza analizzata.

Si può notare che con una sequenza composta per 50% da valori generati e per l'altro 50% da valori reali, il classificatore è in grado di discriminare con una *accuracy* maggiore del 60%, e ciò è una controprova della teoria del *Random Walk* siccome siamo in grado di riconoscere uno scenario futuro non solo dal suo valore precedente ma grazie a tutta la sequenza fornita.

Questo esperimento è stato effettuato allenando il classificatore su un singolo indice, il FTSE MIB.

Rimane da dimostrare se è possibile allenare un classificatore a distinguere sequenze reali da quelle generate allenandolo su più indici contemporaneamente, rimane da capire quale sia il numero minimo di elementi sintetici da aggiungere alla serie necessario al classificatore per essere in grado di discriminare e rimane da capire se utilizzare gli indicatori tecnici possa portare beneficio alle *performance* del discriminatore.

Per cui è stato elaborato un secondo esperimento (questa volta è stata presa in considerazione una sequenza di lunghezza 60), dove sono stati presi in considerazione anche gli indicatori tecnici, precedentemente non utilizzati, utilizzando lo stesso tipo di generatore usato per l'esperimento precedente ovvero *Conditional Random Walk* con l'unica differenza che questa volta le sequenze generate non provenivano

esclusivamente dall'indice FTSE MIB ma da diversi indici, ovvero: FTSE MIB, FTSE 100, S&P 500 e DAX 30.

Come classificatore per il secondo esperimento è stata utilizzata la rete neurale impiegata poi anche nel modello GAN proposto e che verrà quindi illustrata nel dettaglio in sezione 4.3. Quello che interessa per l'esperimento è che sia in grado di ricevere in ingresso sia i prezzi della sequenza che gli indicatori tecnici.

I risultati mostrati in figura 22 e 23 sono stati ottenuti allenando il Discriminatore prima sul singolo indice (FTSE MIB) e poi allenandolo contemporaneamente su più indici testandolo però su un singolo indice (sempre FTSE MIB) in modo da poter confrontare i risultati.

Con lo stesso criterio del primo esperimento è stata fatta un'esplorazione completa partendo col generare un solo valore della sequenza fino a generare l'intera sequenza a partire dal primo valore.

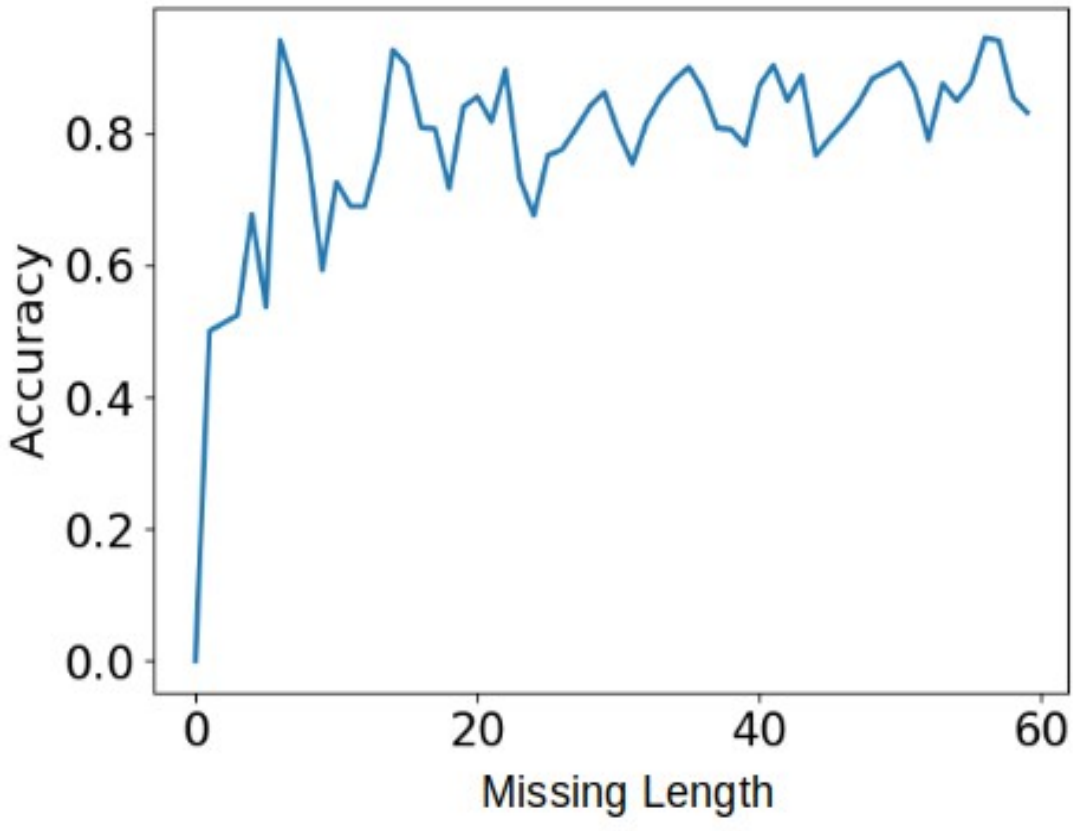

Illustrazione 22: Risultati secondo esperimento con training e test solo sul FTSE MIB.

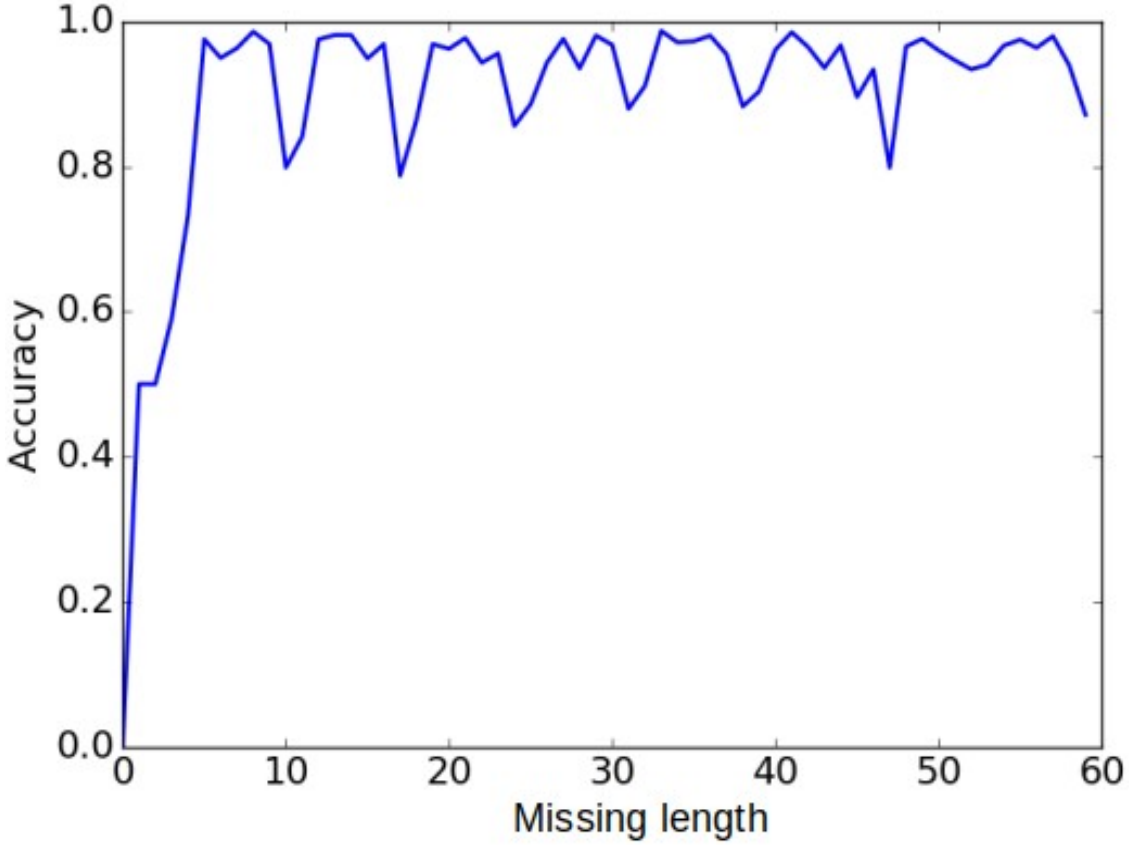

Illustrazione 23: Risultati del secondo esperimento con training su FTSE MIB, FTSE 100, S&P 500 e DAX30 e test su FTSE MIB.

Dalla figura 22 si può subito notare come le prestazioni siano più alte rispetto al primo esperimento, ciò è dovuto all'utilizzo degli indicatori tecnici (che d'ora in poi verranno chiamati contesto) e dimostra che il contesto è utile al discriminatore per aumentare la sua *accuracy.*

Dal confronto tra figura 22 e figura 23 si evince inoltre che il discriminatore riesce a catturare informazioni generali per aumentare le sue performance se viene allenato su più indici contemporaneamente confermando non solo che ci sia correlazione tra valori della stessa sequenza, ma anche che c'è qualche tipo di correlazione tra sequenze di diversi indici. Tutto ciò è dimostrato dal notevole aumento di prestazioni messo in evidenza in figura 23, allenando il discriminatore in modo agnostico, rispetto a figura 22 dove il discriminatore è allenato esclusivamente sul FTSE MIB. Infine si nota che il discriminatore inizia ad avere buone performance, quando i valori futuri generati sono maggiori o uguali a 5 ed è per questo motivo che si è scelto di far generare alla GAN, proposta e analizzata nel capitolo 4, i 5 prezzi futuri, in modo che il discriminatore abbia a disposizione abbastanza informazione per discriminare ed il generatore non debba generare sequenze troppo lunghe che renderebbe difficile il compito di 'ingannare' il discriminatore nella sua valutazione.

3.2.2 Generatore

Lo studio di fattibilità che è stato effettuato sul generatore è servito per comprendere se una rete neurale generativa, fosse in grado di generare scenari futuri per più indici contemporaneamente oppure se si ci dovesse limitare a creare un generatore per ogni singolo indice. É stato quindi modellato un esperimento che dimostrasse l'effettiva possibilità di creare un generatore agnostico, (in grado di generalizzare su più indici) prendendo in considerazione il generatore che è stato utilizzato per il modello GAN proposto e quindi spiegato in dettaglio successivamente in sezione 4.2 .

L'esperimento è stato scomposto in 3 parti:

- Allenare il generatore su un singolo indice per generare scenari futuri solo di quell'indice (Mono).
- Allenare il generatore su più indici contemporaneamente e generare scenari futuri di uno a piacere tra gli indici utilizzati per l'addestramento dando indicazione al generatore di che indice si sta trattando volta per volta tramite vettore *one-hot* (Agnostico *one-hot*).
- Allenare il generatore su più indici e generare scenari futuri di uno a piacere tra gli indici utilizzati per addestrare il generatore senza dare nessuna indicazione su che indice si sta trattando (Agnostico).

Come metrica per questo esperimento è stato utilizzato il *Mean Squared Error,* ovvero l'errore quadratico medio tra gli elementi reali e quelli generati o più formalmente:

$$MSE = \frac{1}{n}\sum_{i=i}^{n}(Y_i - \hat{Y}_i)^2 \tag{3.1}$$

dove n è il numero di valori generati per la serie, Y_i è il valore reale i-esimo e $\hat{Y}_i$ è il valore generato i-esimo.

Il test è stato effettuato per i seguenti indici: FTSE MIB, FTSE 100, S&P 500 e DAX30.

(MSE)	*Mono*	*Agnostico one-hot*	*Agnostico*
FTSE MIB	0,62	0,50	**0,44**
FTSE 100	1,34	**0,48**	0,51
S&P 500	3,48	**0,55**	**0,55**
DAX 30	1,12	0,60	**0,50**

Tabella 1: Risultati studio fattibilità sul generatore.

I risultati di questo esperimento, esposti in tabella 1, mostrano non solo la possibilità di poter utilizzare un generatore agnostico, ma confermano anche quanto dimostrato durante gli esperimenti sul discriminatore ovvero che c'è una correlazione tra indici diversi.

Infatti i risultati sono nettamente migliori, quando il generatore è allenato su più indici contemporaneamente mettendo in evidenza che seppur appartenendo a distribuzioni diverse gli indici hanno qualche caratteristica in comune che permette alla rete di migliorare le sue performance avendo a disposizione una maggiore quantità di dati. Questo risultato si collega a un proprietà che spesso viene sfruttata per le immagini; immaginando che si voglia creare una rete neurale che debba generare immagini di gatti e che si voglia creare poi una seconda rete neurale per generare immagini di gatti e di cani, è auspicabile che la seconda rete possa funzionare meglio perché, pur essendo gatti e cani animali diversi, possiamo ritrovare in essi alcune caratteristiche comuni come il pelo, il naso, quattro zampe ecc. che facilitano il secondo modello ad imparare meglio la distribuzione dei dati.

Si è stato inoltre tentato, di aiutare la rete a riconoscere gli indici diversi tramite un vettore *one-hot*, il vettore *one-hot* non è altro che una sequenza di zeri lunga quanto il numero totale di indici utilizzati, dove ogni zero è associato per posizione ad uno degli indici, quando viene dato in ingresso l'indice *i* viene inserito un uno in posizione *i* all'interno del vettore *one-hot* e dato in ingresso alla rete.

In tabella 1, si nota come il vettore *one-hot* non porti miglioramenti al generatore è quindi stato deciso di escluderlo.

3.3 Metrica

Scegliere una metrica è una fase molto delicata durante un progetto di *machine learning* dove bisogna rappresentare numericamente il risultato per capire e confrontare modelli diversi, è molto importante quindi riuscire a racchiudere in questa metrica ciò che ci si prefissa come obbiettivo.

Nei problemi di classificazione la scelta della metrica spesso ricade sull'*accuracy*, ovvero il numero di predizioni corrette fratto il numero totale di predizioni effettuate.

Si può mettere in evidenza la criticità di scegliere la giusta metrica con un semplice esempio. Supponendo di avere un dataset con 49900 immagini di gatti e solo 100 immagini di cani, e che l'obbiettivo del classificatore è individuare le immagini dei cani.

Se il classificatore dicesse per tutte le immagini che non c'è un cane, avrebbe un *accuracy* del 99,8% ma non si avvicinerebbe neanche minimamente all'obbiettivo per esso prefissato. Nei problemi di classificazione vengono quindi usate anche metriche come *precision* e *recall (16)*, per ottenere un risultato completo sull'efficacia del modello.

Focalizzando l'attenzione sul problema di generazione di serie temporali, potrebbe erroneamente essere scelto il *Mean Squared Error (3.1)* ovvero la media quadratica delle differenze tra le serie reali e quelle generate.

Questa scelta implicherebbe però che l'obbiettivo finale del modello proposto sia generare scenari futuri più vicini possibile a quello reale mentre quello che si vuole ottenere è un numero elevato numero di scenari futuri vero-simili, non necessariamente vicini a quello reale.

Si è scelto quindi di utilizzare una formulazione leggermente diversa della *likelihood* come metrica. Per ogni sequenza data in ingresso saranno generati 500 scenari futuri per i successivi 5 giorni. Considerando, per ora, i prezzi delle 500 predizioni per il giorno 1, questi vengono utilizzati per creare una distribuzione di probabilità (per semplicità è stato scelto di modellare una gaussiana), successivamente viene calcolata la densità di probabilità del valore reale della sequenza al giorno 1 rispetto alla distribuzione appena calcolata.

Più formalmente:

$$Likelihood(y) = \ln \sum_{i=1}^{N} K((y - x_i); h) \tag{3.2}$$

dove:

- N è il numero di scenari futuri generativa.
- y è il valore reale della serie al giorno 1.
- x_i : i = 1,…,N sono i prezzi generati.
- K(x;h) è il *kernel* gaussiano $K(x;h) = \exp(\frac{-x^2}{2h^2})$ dove h è la larghezza di banda della gaussiana.

Questo procedimento viene ripetuto per tutti e 5 i giorni predetti ottenendo 5 diversi valori di *likelihood*, per ottenere un singolo risultato numerico viene poi calcolata semplicemente la media di essi.

Una metrica secondaria utilizzata è l'errore finale percentuale (FE%) che prende in considerazione solamente il valore finale della serie generata e ne calcola la distanza con il valore finale della serie reale in termini percentuali, ovvero:

$$FE\,\% = 100 \frac{|y_{true} - y_{fake}|}{y_{true}}\,\% \tag{3.3}$$

Dove y_{true} é l'ultimo valore della sequenza reale e y_{fake} è l'ultimo valore della sequenza generata.

3.4 Problematiche riscontrate

In questa sezione verranno esposte le problematiche principali che sono state affrontate durante lo sviluppo del progetto ovvero: *Overfitting* e *Mode-Collapse.* Verranno anche illustrate le rispettive modifiche apportate per garantire la risoluzione delle problematiche riscontrate.

3.4.1 Overfitting

Il *Machine Learning* di tipo *supervised*, come accennato in precedenza, approssima una funzione obbiettivo *f* che mappa le variabili di ingresso X nelle variabili di uscita Y ovvero *Y = f(X).* La funzione *f* viene imparata durante l'addestramento del modello, utilizzando il *training* set e deve essere in grado di generalizzare sui nuovi dati.

L'*overfitting*, è quella particolare condizione dove il modello impara dettagli non importanti dai dati di addestramento, ad esempio del rumore, influendo negativamente sulle performance quando vengono presentati nuovi esempi. L'*overfitting* è molto frequente su modelli non lineari in grado quindi di approssimare meglio la funzione obbiettivo *f*.

Nel caso GAN, l'*overfitting* si verifica quando una delle due reti memorizza completamente il *dataset* di *training* per riuscire a sovrastare la rete avversaria.

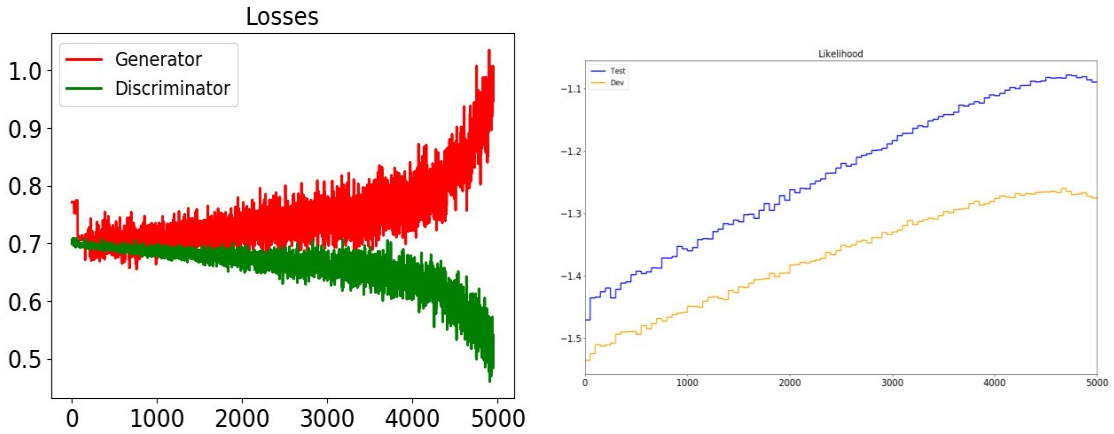

Illustrazione 24: A sinistra l'andamento delle funzioni di costo del generatore e del discriminatore durante il training. A destra l'andamento della likelihood calcolata durante il training sul test set e sul validation set.

Come si nota nell'illustrazione 24, le funzioni di *loss* del generatore e del discriminatore rimangono molto vicine nella fase iniziale del *training*, si allontanano leggermente dopo circa 1000 epoche per poi allontanarsi definitivamente dopo circa 4000 epoche, ed è proprio in questo momento che si verifica l'*overfitting.*

Il discriminatore in particolare ha imparato alla perfezione il dataset di *training*, raggiungendo un *accuracy* del 100%, non lasciando quindi nessuna possibilità al generatore di migliorarsi e produrre sequenze in grado di essere etichettate come vere ricevendo *feedback* positivi dalla funzione di *loss*, questo comporta un incremento del valore della *loss* del generatore portando l'intera architettura a una condizione di stallo.

Per evitare il problema del *overfitting* si è deciso quindi di utilizzare la tecnica dell'*early stopping (17)*, ovvero il *training* del modello sarà bloccato quando si nota l'insorgere del *overfitting* e non dopo un numero prefissato di epoche. Solitamente per fermare il *training* con la tecnica del *early stopping* si ci basa esclusivamente sull'andamento delle funzioni di *loss*, si è deciso però di monitorare anche l'andamento della *likelihood* e fermare l'addestramento quando quest'ultima non mostra ulteriori segni di miglioramento.

Nel caso mostrato in figura 23 è ragionevole fermare il training dopo 4000 epoche, si può notare infatti come la *likelihood* sul *validation set* e sul *test set* non migliora ulteriormente e allo stesso tempo le funzioni di *loss* iniziano ad allontanarsi definitivamente.

Va prestata particolare attenzione però a non fermarsi troppo presto, cadendo nel problema esattamente opposto, l'*underfitting.*

L'*underfitting* è quella particolare condizione dove il modello non riesce a trovare una funzione obiettivo *f* efficace né per il *training* set né per i dati nuovi (*test set)* ed spesso dovuto a una scarsa disponibilità di dati o ad una complessità della rete non adeguata per il problema proposto.

Ritornando all'esempio di figura 23, se il *training* fosse stato interrotto dopo solo 1000 epoche, avremmo avuto una condizione di *underfitting*, dove il valore della *likelihood* sarebbe stato molto inferiore rispetto a quello ricavabile dai dati ingresso.

È bene quindi trovare il giusto compromesso per evitare questo tipo di problematiche ed è per questo motivo che si è monitorato anche l'andamento della *likelihood* durante il *training* oltre che l'andamento delle funzioni di costo.

3.4.2 Mode Collapse

Il *Mode Collapse* è un problema che spesso può affliggere sia le GAN, sia i modelli S*equence-to-Sequence* e che si è dovuto affrontare durante lo sviluppo del progetto di tesi.

In particolare si parla di *Mode Collapse* quando a prescindere dall'ingresso che viene mostrato alla rete, essa produce sempre un'uscita identica o comunque con una scarsa variabilità.

Prendendo come esempio le GAN quello che succede è che avendo un *dataset* con una distribuzione che presenta diversi picchi, il generatore impara solamente uno di questi e osservando che il discriminatore non riesce a distinguere la sequenza generata, continua a produrre in output la stessa cosa pur non essendo la soluzione migliore, un esempio grafico di *Mode Collapse* è mostrato nell'illustrazione 25, dove in rosso sono mostrate le sequenze generate e in verde la sequenza reale.

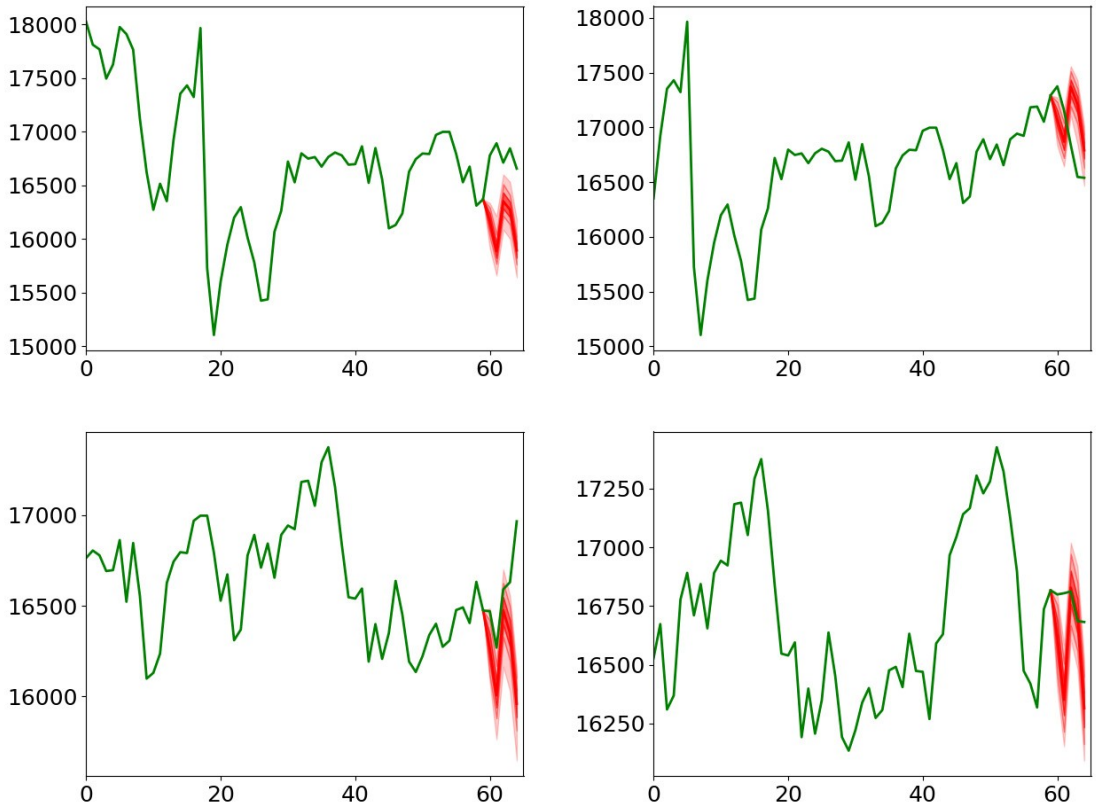

Illustrazione 25: Esempio di Mode Collapse.

Come primo tentativo per risolvere questo problema si è provato ad abbassare di molto la complessità della rete per tentare di evitare che il generatore fosse in grado di raggiungere un punto sub-ottimale e prevalere sul discriminatore producendo sequenze senza valore. Questo approccio effettivamente non ricadeva nel problema del *Mode Collapse* ma abbassava la qualità delle sequenze generate a livello di *likelihood* rientrando di nuovo in uno dei problemi affrontati nella sezione 3.4.1 ovvero l'*underfitting.*

Successivamente si è provato a sbilanciare il modo di allenare la GAN, per evitare che il generatore imparasse una soluzione sub-ottimale, ovvero si è tentato ad allenare di più il discriminatore, sospendendo il *training* del generatore per un determinato intervallo di epoche. Questa soluzione pur portando un leggero miglioramento, non risolve il problema del *Mode Collapse* e inoltre complicato capire quale sia il giusto numero di epoche in cui il *training* del generatore debba essere sospeso.

BATCH NORMALIZATION

Come mostrato dallo stesso GoodFellow, padre della tecnologia GAN, esistono varie opportunità per evitare il *Mode Collapse* (18), tra queste molto valida è l'opzione *Batch Normalization (19).*

È estremamente consigliato normalizzare i dati che vengono forniti in ingresso a una rete neurale, ciò permette di imparare da essi con maggiore velocità riducendo la varianza interna dell'input, è molto probabile infatti che ci siano *features* che vanno da 0 a 1 e magari altre che vanno da 0 a 100.

Per normalizzazione si intende il processo che calcola media e deviazione standard di un set di *features*, e che per ogni elemento del *dataset* ne sottrae la media e lo divide per la deviazione standard.

L'idea che sta alla base della *Batch Normalization* è molto semplice, se la normalizzazione applicata ai dati in ingresso migliora le prestazioni, essa dovrebbe migliorare le prestazioni anche se applicata agli strati nascosti di una rete neurale.

Nel caso preso in esame risulta ineffetti molto utile. Come spiegato nella sezione 5.1.3 l'input viene normalizzato indice per indice, ma allenando la rete contemporaneamente su diversi indici potremmo avere distribuzioni molto diverse in ingresso per ogni *batch.*

Applicando la *Batch Normalization* in alcuni strati nascosti del generatore come mostrato nella sezione 4.2 non solo si ottiene un miglioramento delle prestazioni ma viene risolto anche il problema del *Mode Collapse.*

4. Modello

In questo capitolo verrà analizzata l'architettura realizzata per cercare una soluzione al problema esposto in sezione 3.1, verranno mostrati nel dettaglio i singoli elementi e come essi cooperano infine verrà analizzato l'ottimizzatore scelto per il modello.

4.1 Architettura globale

Come illustrato precedentemente, l'architettura GAN è composta da due reti neurali concorrenti. In particolare, in questo progetto è stata utilizzata una architettura *Context-conditional GAN* dove il generatore oltre al vettore di *noise*, solitamente dato in ingresso alle GAN, riceve in ingresso anche la sequenza dei 120 prezzi precedenti dell'indice azionario preso in esame e la matrice di contesto, che per ogni prezzo giornaliero rappresenta i suoi indicatori tecnici finanziari. Il generatore produce quindi i 5 prezzi successivi e questi vengono concatenati coi 120 dati in ingresso producendo una nuova sequenza.

Il discriminatore riceve in ingresso un vettore di 125 prezzi, non sapendo se questo è stato campionato dal *dataset* originale oppure è stato generato dal generatore, e produce in uscita la probabilità che la sequenza sia reale. Si può apprezzare meglio la struttura del modello tramite lo schema a blocchi rappresentato nell'illustrazione 26.

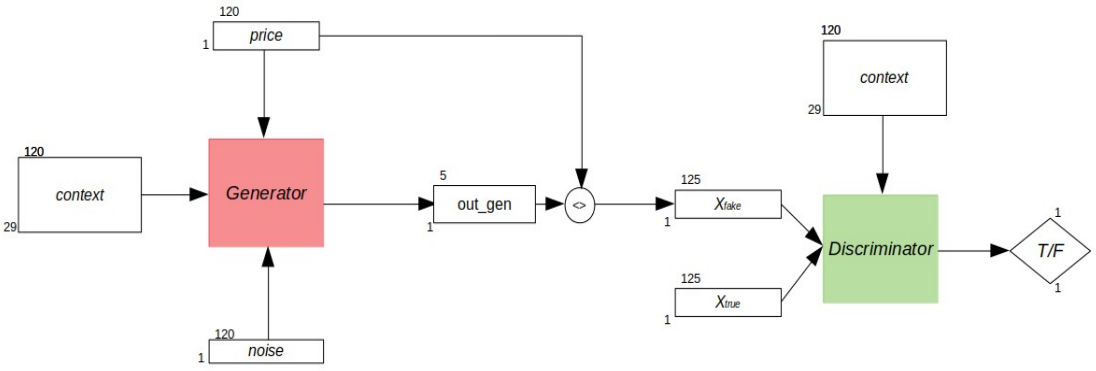

Illustrazione 26: Rappresentazione grafica del modello proposto.

Dove:

- Il *Generator* è la rete neurale descritta in sezione 4.2 .
- Il *Discriminator* è la rete neurale descritta in sezione 4.3.
- La matrice *Context* (29,120) è la matrice dei 29 indicatori tecnici per ognuno dei 120 giorni in ingresso e viene fornita in input sia al discriminatore che al generatore.
- Il vettore *Price* (1,120) è il vettore contenente i 120 prezzi della sequenza appartenenti all'indice analizzato.
- Il vettore di *Noise* (1,120) è il vettore rumore dato in ingresso al generatore, campionato da una gaussiana di media $\mu = 0$ e deviazione standard $\sigma = 1$, con dimensioni pari al doppio delle dimensioni dello spazio latente scelto per le celle ricorrenti all'interno del generatore.
- Il vettore *Out_gen* (1,5) è il vettore che contiene i 5 prezzi futuri generati dal generatore.
- Il vettore X_{fake} (1,125) è il vettore risultante dalla concatenazione tra il vettore P*rice* dato in ingresso e il vettore *Out_gen.*
- Il vettore X_{true} (1,125) è il vettore contenente la sequenza reale composta dal vettore *Price* e i 5 valori reali successivi.
- Il valore *T/F* (1,1) è la probabilità, secondo il discriminatore, che ha la sequenza ricevuta in input, di appartenere al *dataset* originale.

4.2 Generatore

Il generatore è la rete neurale della GAN adibita al compito di creare la sequenza dei prezzi futuri. Per questa rete si è deciso di utilizzare il modello S*equence-to-Sequence* con l'utilizzo dell'*attention,* spiegati rispettivamente in sezione 2.1.5 e 2.1.6.

Essendo un modello S*equence-to-Sequence* l'analisi del generatore si può scomporre in due parti: *Encoder e Decoder.*

L’*Encoder* è utilizzato per condensare le informazioni importanti ricevute in ingresso in un singolo vettore. Il *Decoder* utilizza queste informazioni compresse per generare i prezzi successivi della sequenza.

4.2.1 Encoder

L’*Encoder* è l’elemento che compone la prima parte del generatore, ed è composto da una sequenza di celle LSTM, ognuna di queste celle riceve in ingresso i dati rispettivi a un singolo giorno, quindi il prezzo dell’indice azionario e gli indicatori tecnici, per un totale di 120 giorni. Al termine della sequenza si ottiene un vettore di *encoding* che racchiude tutte le informazioni ricavate, riproducendo dunque la struttura classica dell’*encoder* in un modello S*equence-to-Sequence.*

Per far si che alcuni giorni, dove si sono verificate variazioni di prezzo più alte o dove gli indicatori tecnici mettono in evidenza caratteristiche rilevanti, abbiano più importanza rispetto ad altri dove il prezzo è rimasto pressoché invariato, è stato introdotto il meccanismo dell’*attention.*

Ciò che avviene nella pratica, è che per ogni giorno, viene calcolato un coefficiente di *attention,* (i dettagli sono descritti in sezione 2.1.6) più alto è il coefficiente di *attention* più è importante il giorno ad esso associato. Tutte queste informazioni vengono poi compresse nel vettore di *encoding.*

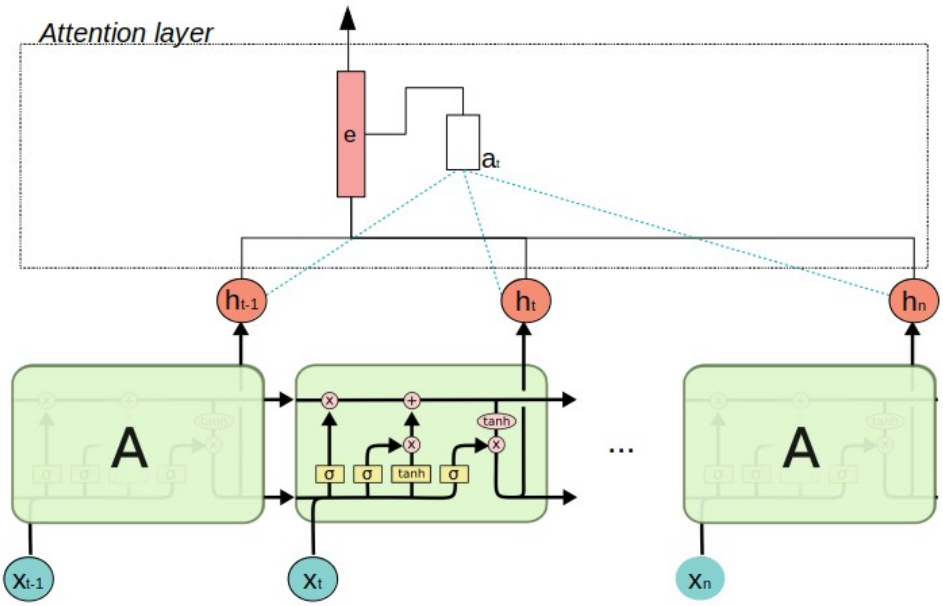

Illustrazione 27: Rappresentazione grafica del encoder all'interno del generatore.

In figura 27 è mostrato graficamente il modello dell'*encoder,* dove:

- x_i con $i = 1, \ldots, n$ è l'ingresso della cella LSTM i-esima, contenente il prezzo dell'indice per il giorno i e i rispettivi indicatori tecnici.

- h_i con $i = 1, \ldots, n$ è lo stato nascosto della cella LSTM corrispondente al i-esimo ingresso.

- a_i con $i = 1, \ldots, n$ è il coefficiente di *attention* calcolato dallo stato nascosto i-esimo corrispondente al i-esimo ingresso.

- e è il vettore di *encoding* che verrà fornito al *Decoder,* ottenuto moltiplicando i coefficienti di *attention* con i rispettivi stati nascosti delle celle LSTM.

4.2.2 Decoder

Il *Decoder* è l'elemento che compone la seconda parte del generatore. È costruito con una seconda sequenza di celle LSTM, esattamente come prevede il modello *sequence-to-sequence(12).*

La prima cella del *decoder* è inizializzata con lo stato interno dell'ultima cella dell'*encoder* su cui però viene sommato il rumore per ottenere variabilità nel risultato e riceve in ingresso il vettore di *encoding* prodotto dal generatore.

All'uscita di ogni cella ricorrente viene applicata la *batch normalization,* vista in sezione 3.4.2, e viene fornita ingresso a una rete neurale (2.1.1) composta da un singolo neurone che fornisce in uscita il prezzo futuro della sequenza. Questa operazione viene ripetuta fino a generare i 5 prezzi successivi dell'indice, inizializzando ogni cella del *decoder*, con lo stato della cella precedente e fornendo in ingresso a ogni cella l'uscita della cella precedente, a cui è stata applicata la *batch normalization.*

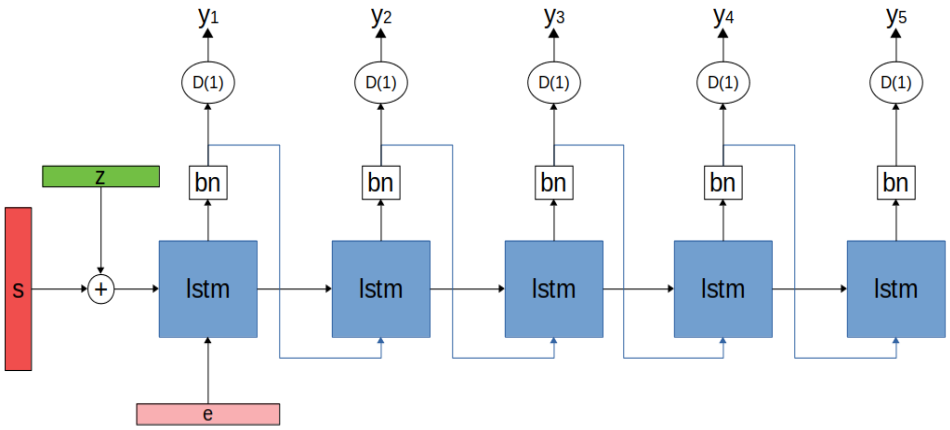

Illustrazione 28: Architettura del decoder all'interno del generatore.

In figura 28 è illustrato graficamente il funzionamento del *decoder,* dove:

- *s* sta a indicare lo stato dell'ultima cella dell'*encoder* utilizzato per inizializzare lo stato della prima cella del *decoder.*

- *e* è il vettore di *encoding* generato dell'*encoder.*

- *Bn* è il la *batch normalization* applicata all'uscita di ogni cella LSTM.

- *D(1)* rappresenta la rete neurale con un neurone che genera il prezzo futuro.

- y_i con $i = 1, \dots, 5$ sono i 5 prezzi generati.

- *z* è il vettore di *noise* con media $\mu = 0$ e deviazione standard $\sigma = 1$ sommato allo stato finale dell'ultima cella dell'*encoder.*

4.3 Discriminatore

Il discriminatore è la rete neurale della GAN adibita a riconoscere se le sequenze di prezzi ricevute in input provengono dal *dataset* originale o se esse sono state generate. Se il discriminatore diventa abbastanza efficiente, solo le sequenze vero-simili verranno contrassegnate come originali garantendo la produzione di scenari di alta qualità.

Il discriminatore utilizzato presenta due sequenze di strati convolutivi (2.1.2). Ogni operazione di convoluzione è seguita da una operazione di *LeakyReLu* (2.6) come consigliato da Chintala (20). La prima sequenza di convoluzioni prende in ingresso gli indicatori tecnici per i primi 120 giorni, la seconda invece prende l'intera sequenza di 125 prezzi a cui viene concatenata l'elaborazione degli indicatori tecnici prodotta dalla prima sequenza di convoluzioni. L'ultimo strato della rete è formato da un *fully connected* (2.1.1) che produce la probabilità che la sequenza sia vera.

In figura 29 è mostrata la rappresentazione grafica dell'architettura del discriminatore, dove:

- *Additional info* (29,120) rappresenta la matrice degli indicatori tecnici per i primi 120 giorni.

- *Input sequence* (1,125) rappresenta l'intera sequenza di 125 prezzi azionari.

- *Conv(x,y,z)* rappresenta un *layer* convolutivo dove: *x* è il numero di filtri, *y* è la dimensione del *kernel* dei filtri, e *z* è lo *stride* (scostamento del filtro tra una convoluzione e quella successiva).

- *Output* (1,1) è la probabilità che l'*Input sequence* sia reale.

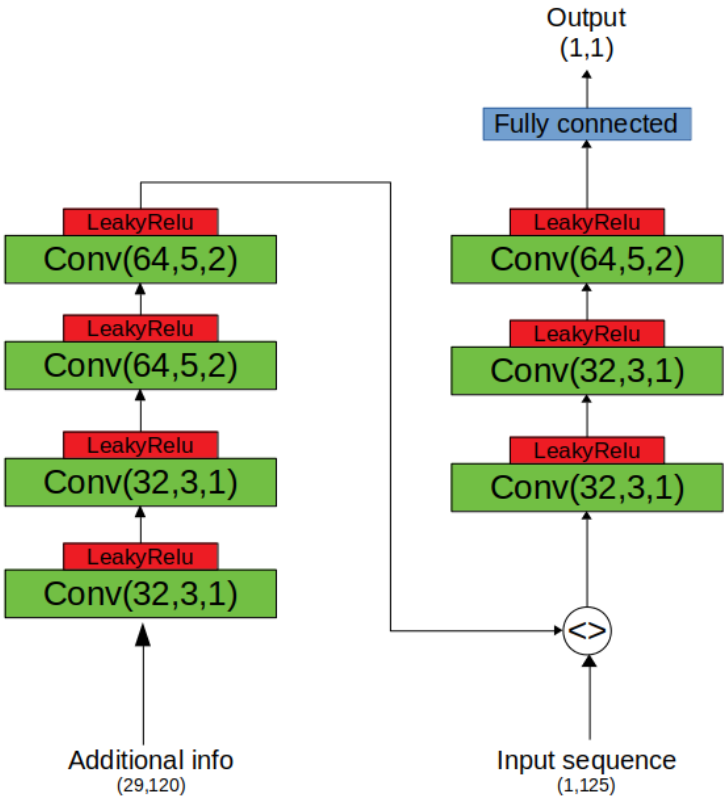

Illustrazione 29: Schema a blocchi dell'architettura del discriminatore.

4.4 Loss

La funzione di *loss*, anche chiamata funzione di costo, è una delle componenti fondamentali delle reti neurali, essa è utilizzata per calcolare la compatibilità tra il valore generato dalla rete e il valore reale che si cerca di ottenere. La funzione di *loss* assume un valore più basso al crescere della precisione e robustezza del modello, mentre se la rete comincia a produrre risultati errati la *loss* penalizza la rete aumentando il suo valore, dando cosi una chiara indicazione al modello su come modificare i propri pesi interni.

Come descritto in sezione 2.1.4, siccome la GAN è composta da due reti neurali che competono tra loro, saranno presenti nel modello due funzioni di *loss* opposte, ovvero al calare del valore assunto da una aumenta il valore assunto dall'altra e viceversa, questo tipo funzioni di *loss* è detto *adversarial*.

La *loss* per il discriminatore non è stata modificata rispetto a quella classica delle GAN, mentre per il generatore è stata combinata una parte *surpevised (Mean Squared Error (3.1))* a quella *adversarial* per generare non solo serie realistiche che possano trarre in inganno il discriminatore ma che possano avvicinarsi maggiormente al valore reale.

$$\max_D V(D,G) = E_{P_{Data}(x)}[\log D(x)] + E_{P_z(z)}[\log(1-D(G(z)))] \quad (4.1)$$

$$\max_G V(D,G) = \alpha E_{P_z(z)}[\log(1-D(G(z)))] - (1-\alpha) E_{P_z(z)} \|o - G(z)\|_2 \quad (4.2)$$

Dove $\alpha \in [0,1]$ rappresenta la percentuale di contributo *adversiarial* che viene usata dalla funzione di *loss* del generatore.

4.5 Ottimizzatore

Quando inizia l'addestramento di una rete neurale, i suoi parametri vengono inizializzati automaticamente in modo casuale. Ovviamente, questo comporterà prestazioni molto basse da parte della rete, ma aggiornando questi parametri in modo che il valore della funzione di *loss* decresca epoca dopo epoca, le prestazioni del modello saranno sempre più elevate.

Esistono molti metodi con cui questi parametri vengono aggiornati, quello che si è deciso di utilizzare è lo *Stochastic Gradient Descent (SGD)* più *Momentum(21).*
Questo metodo avvicina a piccoli passi i parametri della rete alla soluzione ottima, il *Momentum* invece non è altro che una regolarizzazione che permette al modello di convergere più velocemente. Formalmente:

$$V_t = \beta V_{t_1} + \alpha \nabla_\theta L(\theta - \beta V_{t-1}, X, y) \quad (4.3)$$

$$\theta = \theta - V_t \quad (4.4)$$

Dove :

- L è la funzione di *loss.*
- θ sono i parametri della rete.
- X è l'ingresso della rete.
- y è l'uscita della rete.
- α è il *learning rate.*
- β è il coefficiente di *Momentum.*

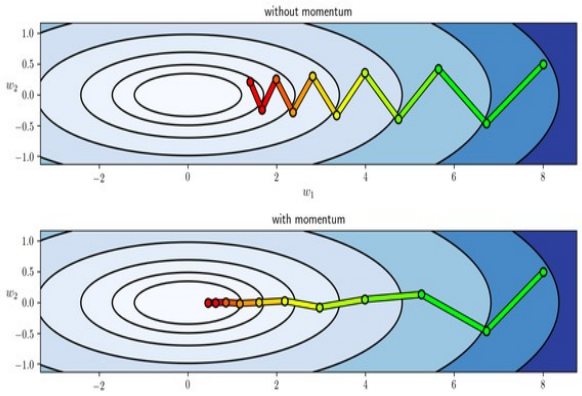

Illustrazione 30: Spiegazione grafica di come funziona SGD con e senza momentum.

5. Esperimenti e risultati

In questo capitolo si analizzerà innanzitutto la struttura del *dataset* utilizzato e le elaborazioni che sono state effettuate sui dati. Verranno poi presi in esame gli iperparametri del sistema con una descrizione di essi e i rispettivi valori utilizzati per il processo di *training* del modello.

Per poter confrontare i risultati verranno poi descritte le *baseline* utilizzate e infine saranno esposti i risultati qualitativi e quantitativi ottenuti.

5.1 Dataset

Nei progetti di *Machine Learning* uno degli elementi più importanti per ottenere buoni risultati è il *Dataset.*

Per *Dataset* si intende l'insieme dei dati che si ha disposizione, su cui si andrà a modellare la soluzione. È buona norma dividere i dati che si hanno a disposizione in tre *Dataset* indipendenti:

- *Training Set:* è il *dataset* utilizzato per allenare il modello, che viene utilizzato quindi per modificare i pesi e i *biases* interni alla rete neurale. È l'unico *dataset* che viene mostrato alla rete per imparare a eseguire il compito assegnato.
- *Validation Set:* questo *dataset* viene mostrato alla rete per valutare la qualità dell'informazione appresa, viene utilizzato per modificare gli iperparametri del modello ma non vengono apprese informazioni dai dati di questo *dataset* quindi influenza solo indirettamente l'architettura.
- *Test Set:* questo *dataset* è utilizzato puramente per confrontare le prestazioni tra diverse architetture e quindi per ottenere i risultati finali, non influenza in nessun modo l'architettura del modello.

Nel caso in esame grazie all'azienda ospitante si aveva a disposizione la sequenza dei prezzi degli indici azionari descritti in sezione 2.2.5 e dei relativi indicatori tecnici, a partire dal 3 gennaio del 2000 fino al 29 dicembre del 2017 per un totale di 4696 prezzi per ogni indice.

Nelle prossime sezioni sarà descritto come è stato suddiviso il *Dataset* e come sono stati elaborati i dati prima di essere dati in pasto alla rete siccome l'architettura descritta in sezione 4.1 accetta come ingresso una sequenza di prezzi di lunghezza 120.

5.1.1 Variazione percentuale

Come si sente spesso parlare in finanza, è abitudine comune analizzare gli indici azionari tramite le loro variazioni percentuali giornaliere.

La prima operazione effettuata sui dati è stata quindi di trasformare i prezzi in variazioni percentuali rispetto al prezzo del giorno precedente, ovvero:

$$\begin{Bmatrix} 0, & se\ t=0 \\ \frac{p(t)-p(t-1)}{p(t-1)*100}, & se\ t\neq 0 \end{Bmatrix} \tag{5.1}$$

È una semplificazione efficiente trattare tutti i prezzi come variazioni percentuali siccome rende i dati più gestibili per il modello considerando che indici diversi possono avere prezzi anche molto differenti.
È sempre possibile ritornare al valore originale del prezzo tenendo traccia del primo valore del prezzo della sequenza, i successivi saranno semplicemente ottenuti moltiplicando il prezzo iniziale per le successive variazioni percentuali.

5.1.2 Split

Una volta calcolate le variazioni percentuali la seconda operazione effettuata è stata quella di dividere i dati in: *Training set, Validation set e Test set* come descritto in sezione 5.1 .

Si è scelto di assegnare l'80% dei dati al *dataset* di *training,* il 10% al *validation* e il restante 10% al *test set.*

Ognuno degli indici sarà suddiviso quindi:

	Numero giorni
Training set	3756
Validation set	470
Test set	469

Tabella 2: Dimensioni dei dataset di training, validation e test di un singolo indice considerando il numero di giorni presi in considerazione.

Come detto in precedenza il modello però è stato allenato utilizzando contemporaneamente tutti i 10 indici analizzati in sezione 2.2.5, i *dataset* quindi avranno le seguenti dimensioni:

	Numero giorni
Training set	37560
Validation set	4700
Test set	4690

Tabella 3: Dimensioni dei dataset complessivi di training, validation e test considerando il numero di giorni presi in considerazione.

Illustrazione 31: Rappresentazione grafica della suddivisione in training, validation e test sul FTSE MIB.

5.1.3 Normalizzazione

Un'altra importante elaborazione che è stata fatta sui dati è la normalizzazione. La normalizzazione è un operazione che aiuta i modelli di *Machine Learning* ad apprendere dai dati semplificandoli ulteriormente, ma conservando il loro valore informativo.

Per normalizzare un *dataset* si calcola la media μ e la rispettiva deviazione standard σ e si sostituisce ogni elemento *x* del *dataset* come segue:

$$x_n = \frac{x - \mu}{\sigma} \tag{5.2}$$

Questa operazione porta il *dataset* ad avere media $\mu_n = 0$ e $\sigma_n = 1$, facendo in modo che la scala non incida più in nessun modo per una *features* normalizzata. Quello che è stato fatto nel caso in esame è stato di normalizzare il prezzo delle azioni rispetto al *dataset* di *training,* ovvero calcolare μ e σ sul *dataset* di *training* e normalizzare tutti e tre i *dataset* di *training, validation* e *test* con questi coefficienti. Questa scelta è stata dettata dal fatto che solitamente quello che si conosce di un indice azionario è la media e la varianza del passato (quindi del *training set)* e non c'è modo di sapere la media e la varianza del futuro (ovvero *validation* e *test set*) fino al verificarsi di esso.

Anche la normalizzazione è un'operazione invertibile che permette quindi di tornare al valore originale di *x* semplicemente moltiplicando per la varianza σ e poi sommando la media μ.

5.1.4 Overlap

L'ultima operazione effettuata sui dati è stata quella di creare delle sequenze di prezzi di lunghezza pari a 125, dove i primi 120 prezzi saranno utilizzati come ingresso e gli ultimi 5 come valore di *ground-truth.*

Seppur creare sequenze consecutive di 125 prezzi sia una buona soluzione oltre che la più intuitiva, il numero totale di sequenze disponibili per il dataset di *traning* sarebbe solamente di 300, mentre per il *validation set* addirittura solo 37.

Quello che si è scelto di fare quindi di creare sequenze sovrapposte per il 95% della loro lunghezza ottenendo un numero molto più elevato di serie su cui addestrare il modello. Questa operazione può essere vista come un'operazione di *data augmentation* ed è stato dimostrato in più ambiti come questo aumenti le prestazioni(22). Si è scelto inoltre di tenere separati i *dataset* di *Test* di ogni indice per permettere un confronto indice per indice. Il numero di sequenze finali ottenuto per i tre *datasets* sono i seguenti:

	Numero sequenze
Training set	5190
Validation set	500
Test set (per indice)	50

Tabella 4: Numero di sequenze contenute nel Training set, Validation set e Test set.

5.1.5 Indicatori tecnici

In questa sezione verranno descritti gli indicatori tecnici che come introdotto in sezione 4.1 l'architettura proposta prende in ingresso per ognuno dei prezzi della sequenza. Anche questi indicatori sono stati forniti da Axyon AI e sono gli stessi che vengono utilizzati con successo nei loro prodotti. Ovvero:

1. *Moving Average Convergence Divergence* (5,10) questo indicatore mostra la differenza in valore assoluto tra la media mobile esponenziale (EMA, è una media mobile che da un peso maggiore agli ultimi valori della serie) del prezzo del titolo a 5 giorni con l'EMA a 10 giorni. Questo indicatore indica se il movimento verso l'alto o verso il basso del prezzo si stia rafforzando o indebolendo.
$$MACD = |5\,days\ EMA - 10\,days\ EMA| \tag{5.3}$$

2. *Moving Average Convergence Divergence* (10,20) è l'indicatore finanziario descritto al punto 1 ma calcolato con l'EMA a 10 giorni e l'EMA a 20 giorni.
$$MACD = |10\,days\ EMA - 20\,days\ EMA| \tag{5.4}$$

3. *Moving Average Convergence Divergence* (20,40) è l'indicatore finanziario descritto al punto 1 ma calcolato con l'EMA a 20 giorni e l'EMA a 40 giorni.
$$MACD = |20\,days\ EMA - 40\,days\ EMA| \tag{5.5}$$

4. *Moving Average Convergence Divergence* (40,60) è l'indicatore finanziario descritto al punto 1 ma calcolato con l'EMA a 40 giorni e l'EMA a 60 giorni.
$$MACD = |40\,days\ EMA - 60\,days\ EMA| \tag{5.6}$$

5. *Moving Average Convergence Divergence* (60,90) è l'indicatore finanziario descritto al punto 1 ma calcolato con l'EMA a 10 giorni e l'EMA a 20 giorni.
$$MACD = |60\,days\ EMA - 90\,days\ EMA| \tag{5.7}$$

6. *Chande Momentum Oscillator* (5) questo indicatore calcola la differenza tra la somma dei guadagni a 5 giorni e la somma delle perdite a 5 giorni e viene diviso poi per la somma dei movimenti totali del prezzo nello stesso periodo.

$$\frac{\sum^{5} Gain - \sum^{5} loss}{\sum^{5} Gain + \sum^{5} loss} * 100 \qquad (5.8)$$

7. *Chande Momentum Oscillator* (10) è l'indicatore finanziario descritto al punto 6 ma calcolato su una finestra temporale di 10 giorni.

$$\frac{\sum^{10} Gain - \sum^{10} loss}{\sum^{10} Gain + \sum^{10} loss} * 100 \qquad (5.9)$$

8. *Chande Momentum Oscillator* (20) è l'indicatore finanziario descritto al punto 6 ma calcolato su una finestra temporale di 20 giorni.

$$\frac{\sum^{20} Gain - \sum^{20} loss}{\sum^{20} Gain + \sum^{20} loss} * 100 \qquad (5.10)$$

9. *Chande Momentum Oscillator* (40) è l'indicatore finanziario descritto al punto 6 ma calcolato su una finestra temporale di 40 giorni.

$$\frac{\sum^{40} Gain - \sum^{40} loss}{\sum^{40} Gain + \sum^{40} loss} * 100 \qquad (5.11)$$

10. *Chande Momentum Oscillator* (60) è l'indicatore finanziario descritto al punto 6 ma calcolato su una finestra temporale di 60 giorni.

$$\frac{\sum^{60} Gain - \sum^{60} loss}{\sum^{60} Gain + \sum^{60} loss} * 100 \qquad (5.12)$$

11. *Chande Momentum Oscillator* (90) è l'indicatore finanziario descritto al punto 6 ma calcolato su una finestra temporale di 90 giorni.

$$\frac{\sum^{90} Gain - \sum^{90} loss}{\sum^{90} Gain + \sum^{90} loss} * 100 \qquad (5.13)$$

12. *Rate of Change* (2,5) è la variazione percentuale tra il prezzo dell'indice azionario preso in esame al giorno 2 e il prezzo al giorno 5.

$$\frac{p(5)-p(2)}{p(2)} \tag{5.14}$$

13. *Rate of Change* (2,10) è la variazione percentuale tra il prezzo dell'indice azionario preso in esame al giorno 2 e il prezzo al giorno 10.

$$\frac{p(10)-p(2)}{p(2)} \tag{5.15}$$

14. *Rate of Change* (2,20) è la variazione percentuale tra il prezzo dell'indice azionario preso in esame al giorno 2 e il prezzo al giorno 20.

$$\frac{p(20)-p(2)}{p(2)} \tag{5.16}$$

15. *Rate of Change* (2,40) è la variazione percentuale tra il prezzo dell'indice azionario preso in esame al giorno 2 e il prezzo al giorno 40.

$$\frac{p(40)-p(2)}{p(2)} \tag{5.17}$$

16. *Rate of Change* (2,60) è la variazione percentuale tra il prezzo dell'indice azionario preso in esame al giorno 2 e il prezzo al giorno 60.

$$\frac{p(60)-p(2)}{p(2)} \tag{5.18}$$

17. *Rate of Change* (2,90) è la variazione percentuale tra il prezzo dell'indice azionario preso in esame al giorno 2 e il prezzo al giorno 90.

$$\frac{p(90)-p(2)}{p(2)} \tag{5.19}$$

18. *Relative Strength Index* (5) è un indicatore finanziario che misura la grandezza dei recenti cambiamenti di prezzo nell'arco degli ultimi 5 giorni, per valutare l'eccesso di acquisto o l'eccesso di vendita nel prezzo dell'indice. Ed è calcolato come segue:

$$RSI(5)=100-\left[\frac{100}{1+\frac{AVG(gain)_5}{AVG(loss)_5}}\right] \tag{5.20}$$

19. *Relative Strength Index* (10) è l'indicatore tecnico descritto al punto 18 ma calcolato su una finestra temporale di 10 giorni.

$$RSI(10) = 100 - \left\lceil \frac{100}{1 + \frac{AVG(gain)_{10}}{AVG(loss)_{10}}} \right\rceil \tag{5.21}$$

20. *Relative Strength Index* (20) è l'indicatore tecnico descritto al punto 18 ma calcolato su una finestra temporale di 20 giorni.

$$RSI(20) = 100 - \left\lceil \frac{100}{1 + \frac{AVG(gain)_{20}}{AVG(loss)_{20}}} \right\rceil \tag{5.22}$$

21. *Relative Strength Index* (40) è l'indicatore tecnico descritto al punto 18 ma calcolato su una finestra temporale di 40 giorni.

$$RSI(40) = 100 - \left\lceil \frac{100}{1 + \frac{AVG(gain)_{40}}{AVG(loss)_{40}}} \right\rceil \tag{5.23}$$

22. *Relative Strength Index* (60) è l'indicatore tecnico descritto al punto 18 ma calcolato su una finestra temporale di 60 giorni.

$$RSI(60) = 100 - \left\lceil \frac{100}{1 + \frac{AVG(gain)_{60}}{AVG(loss)_{60}}} \right\rceil \tag{5.24}$$

23. *Relative Strength Index* (90) è l'indicatore tecnico descritto al punto 18 ma calcolato su una finestra temporale di 90 giorni.

$$RSI(90) = 100 - \left\lceil \frac{100}{1 + \frac{AVG(gain)_{90}}{AVG(loss)_{90}}} \right\rceil \tag{5.25}$$

24. *Weighted Moving Average* (5) è un indicatore tecnico che calcola una media pesata del passato, dando più importanza ai prezzi recenti rispetto a quelli più lontani ed è calcolato su una finestra temporale di 5 giorni:

$$WMA(5) = (x_1 * (\frac{1}{5})) + (x_2 * (\frac{2}{5})) + (x_3 * (\frac{3}{5})) + (x_4 * (\frac{4}{5})) + (x_5 * (\frac{5}{5}))$$

25. *Weighted Moving Average* (10) è lo stesso indicatore tecnico del punto 24 ma calcolato su una finestra temporale di 10 giorni.

26. *Weighted Moving Average* (20) è lo stesso indicatore tecnico del punto 24 ma calcolato su una finestra temporale di 20 giorni.

27. Weighted Moving Average (40) è lo stesso indicatore tecnico del punto 24 ma calcolato su una finestra temporale di 40 giorni.

28. Weighted Moving Average (60) è lo stesso indicatore tecnico del punto 24 ma calcolato su una finestra temporale di 60 giorni.

29. Weighted Moving Average (90) è lo stesso indicatore tecnico del punto 24 ma calcolato su una finestra temporale di 90 giorni.

Come descritto, questi indicatori tecnici, forniscono al generatore informazioni più dettagliate riguardo alla storia passata di un singolo prezzo, per far si che possa generare sequenze del futuro il più vero-simili possibile.

5.2 Metodo di Addestramento

In questa sezione sarà descritto come è stato allenato il modello proposto nel capitolo 4. La presenza di due reti neurali all'interno dell'architettura GAN spesso può causare l'insorgere di problemi, tra quelli più comuni di cui le GAN soffrono, troviamo:

- *Non-convergence*, ovvero i parametri del modello non acquisiscono mai stabilità, oscillando, epoca dopo epoca, senza raggiungere un punto di convergenza.

- *Mode-Collapse,* problema descritto e affrontato in sezione 3.4.2.

- *Diminished Gradient,* ovvero il discriminatore sovrasta il generatore impedendo a quest'ultimo di ricevere gradiente utile per migliorarsi.

- *Unbalancing overfitting,* ovvero una delle due reti impara a memoria il *dataset* di *training* portando il modello a non migliorarsi più.

- Alta sensibilità agli iperparametri del sistema.

L'obbiettivo quindi durante il *training* della GAN è fare in modo che le due reti neurali conservino un equilibrio tra loro e che nessuna delle due possa prevalere sull'altra.

Il metodo con cui è stato allenato il modello prende spunto dai suggerimenti forniti in (20) e (18), all'inizio del *training* si è scelto di allenare il discriminatore più frequentemente rispetto al generatore, per far si che potesse essere in grado di riconoscere meglio le sequenze reali, sfruttando il fatto che intuitivamente durante le prime epoche il generatore crea sequenze abbastanza riconoscibili. È stato quindi imposto un intervallo decrescente di 25 epoche durante il quale il generatore non viene allenato. Quando questo intervallo raggiunge dimensione pari a 1, per ogni epoca vengono allenate entrambe le reti.

Analizzando un'epoca dove vengono allenati sia il discriminatore che il generatore, e considerando un *mini-batch* di dimensione *n*, il primo passo è quello di allenare il discriminatore, a cui viene dato in ingresso $\frac{n}{2}$ sequenze di 125 prezzi prese in modo casuale dal *dataset* e $\frac{n}{2}$ sequenze prese sempre in modo casuale, dal *dataset* ma composte da 120 valori reali e da 5 valori generati dal generatore. Gli esempi originali sono contrassegnati con una *label* di valore 1 mentre quelli sintetici con una *label* di valore 0.

Il secondo passo è quello di allenare il generatore congelando i pesi del discriminatore. Il generatore, riceve in ingresso *n* serie prese a caso dal *dataset* e ne genera la parte finale, queste sequenze sintetiche vengono contrassegnate con *label* uguale a 1, per cercare di far modificare i pesi del generatore in modo che l'output del discriminatore sia 1. La *loss adversarial* viene quindi propagata attraverso il discriminatore, senza modificarne i pesi e riversata sul generatore. La *loss supervised* invece viene calcolata semplicemente calcolando la distanza che c'è tra le sequenze originali e quelle generate.

5.2.1 Iperparametri di training

In questa sezione verranno descritti tutti gli iperparametri del sistema. Gli iperparametri del sistema, sono quei parametri impostati dal progettista della rete sulla base di scelte progettuali volte al miglioramento delle *performance* sul *validation set.*

Gli iperparametri globali utilizzati per il progetto in esame sono:

- *Epochs = 5000* è il numero di epoche, ovvero il numero di volte per cui si ripete il procedimento descritto in sezione 5.2 .

- *Batch size = 32* è la dimensione del *mini-batch* descritto in sezione 5.2 , ovvero del numero di sequenze mostrato al discriminatore e al generatore per ogni epoca.

- *Evaluate likelihood loss = 'True'* è un *flag* che indica se verranno analizzati gli andamenti della *likelihood* e delle funzioni di *loss* durante il *training* o meno.

- *Evaluate likelihood Samples = 100* è il numero di sequenze generate che viene utilizzato per calcolare la *likelihood* durante il *training,* come descritto in sezione 3.3 .

- *Evaluate likelihood loss Interval = 50* è l'intervallo di epoche che intercorre tra una valutazione della *likelihood* e delle funzioni di *loss* e quella successiva.

- *Given Length = 120* è la lunghezza della sequenza reale data in ingresso al modello.

- *Sequence Length = 5* è la lunghezza dello scenario futuro generato dalla GAN.

- *Features Length = 31* è il numero di *features* che la rete prende in ingresso per ogni giorno della sequenza e comprende i 29 indicatori tecnici descritti in sezione 5.1.5 , la data di inizio della sequenza e il prezzo dell'indice per quel determinato giorno.

- *Overlap Percentage = 95%* è la percentuale di sovrapposizione con cui vengono create le sequenze, come descritto in sezione 5.1.4.

- *Split* = (80%,10%,10%) è la divisione effettuata sui dati per creare *training set, validation set* e *test set.*

- *Method = 'rate_split_norm_overlap'* è l'ordine con cui vengono effettuate le operazioni descritte in sezione 5.1 .

- *Task = 'rate'* permette di selezionare la modalità di uso del prezzo dell'indice ovvero variazione percentuale o prezzo reale. *Rate* sta per variazione percentuale.

- *Trai*n *equally = 'False'* è un flag che indica se l'addestramento di generatore e discriminatore avviene in modo bilanciato.

- *Generator interval = 25* intervallo di epoche per cui il generatore non viene allenato in caso di *training* sbilanciato.

- *Decay generator interval = 'True'* indica se l'intervallo di sbilanciamento del *training* decresce epoca dopo epoca fino ad arrivare ad un addestramento bilanciato o meno.

Mentre gli iperparametri utilizzati per il generatore sono:

- *Only adversarial loss = 'False'* è un flag che indica se per il generatore viene usata una *loss* completamente *adversarial* oppure se viene utilizzato anche il contributo di una *loss supervised.*

- *Adversarial Weight = 0.5* coefficiente che indica il contributo che da la *loss adversarial* all'addestramento del generatore rispetto al contributo dato dalla *loss supervised.*

- *Generator States = 60* è la dimensione dello spazio latente delle celle LSTM che compongono l'*encoder* del generatore.

- *Generator Noise Handling = 'add'* indica il metodo con cui viene aggiunto il rumore allo stato finale della sequenza dell'*encoder* che viene utilizzato per inizializzare la prima cella LSTM del *decoder. 'Add'* somma il rumore allo stato, altre opzioni possibili sono '*mul*' per moltiplicare il rumore allo stato e '*concat*' per concatenare il rumore allo stato.

- *Noise mean = 0* è il valore della media della gaussiana utilizzata per campionare il rumore fornito in ingresso al generatore.

- *Noise std = 1* è il valore della deviazione standard della gaussiana utilizzata per campionare il rumore fornito in ingresso al generatore.

- *Generator agnostic = 'True'* è un *flag* che indica se usare il generatore in modalità agnostica o avere un generatore diverso per ognuno degli indici dati in ingresso.

- *Optimizer = 'SGD'* metodo di ottimizzazione per l'aggiornamento dei parametri del generatore, descritto in sezione 4.5.

- *Learning Rate = 0.0002* parametro che indica la velocità dell'ottimizzazione dei pesi del generatore, come mostrato nella formula (4.3) in sezione 4.5.

- *Momentum = 0.9* è il coefficiente di *momentum* descritto in sezione 4.5.

Infine gli iperparametri utilizzati per il discriminatore sono:

- *Discriminator Filters = [32, 32, 64]* è il numero di filtri utilizzato per ognuno dei *layer* del discriminatore che prende in ingresso i prezzi della sequenza, dove il primo *layer* contiene 32 filtri, il secondo 32, e il terzo 64.

- *Discriminatore Kernel Size = [3, 3, 5]* è la dimensione dei filtri utilizzati per ognuno dei *layer* del discriminatore che prende in ingresso i prezzi della sequenza. Dove il primo e il secondo *layer* contengono filtri di dimensione 3 e il terzo contiene filtri di dimensione 5.

- *Discriminatore Strides = [1, 1, 2]* è la dimensione dello *stride* utilizzato, ovvero di quanti valori un filtro scorre ad ogni step. Dove i filtri del primo e del secondo *layer* hanno *stride pari* 1 mentre quelli del terzo *layer* hanno *strides* pari a 2.

- *Discriminator additional info = 'True'* è un *flag* che indica se il discriminatore debba usare o meno, gli indicatori tecnici per discriminare le sequenze in ingresso.

- *Discriminator Additional info Filters = [32, 32, 64, 64]* è il numero di filtri contenuti in ognuno dei *layer* che analizzano gli indicatori tecnici della sequenza.

- Discriminator Additional info *Kernel Size = [3, 3, 5, 5]* è la dimensione dei filtri utilizzati per ognuno dei *layer* del discriminatore che analizzano gli indicatori tecnici della sequenza.

- Discriminator Additional info *Strides = [1, 1, 2, 2]* è la dimensione dello *stride* utilizzato dai filtri per ognuno dei *layer* del discriminatore che analizzano gli indicatori tecnici della sequenza.

- *Alpha = 0.2* è il coefficiente α utilizzato per la LeakyReLu all'interno del discriminatore, mostrato in formula (2.6).

- *Optimizer = 'SGD'* metodo di ottimizzazione per l'aggiornamento dei parametri del discriminatore, descritto in sezione 4.5.

- *Learning Rate = 0.0002* parametro che indica la velocità dell'ottimizzazione dei pesi del modello, come mostrato nella formula (4.3) in sezione 4.5.

- Momentum = 0.9 è il coefficiente di *momentum* descritto in sezione 4.5.

5.3 Baseline

Nel mondo del *Machine Learning* per valutare i risultati ottenuti, quello che di norma viene fatto è un confronto tra diverse b*aseline.*

Una *Baseline* è un sistema o un modello, volto a risolvere lo stesso problema di quello preso in esame ma utilizzante un diverso approccio. Il risultato prodotto da una *baseline* deve essere confrontabile con il risultato ottenuto dal modello proposto.

Una *baseline* valida quindi è un modello o euristica in grado di prendere in ingresso una sequenza di 120 valori, appartenente ad uno degli indici azionari introdotti in sezione 2.2.5 e fornire in uscita un numero arbitrario di scenari futuri possibili di lunghezza pari a 5.

Come *baseline* si è deciso di utilizzare il modello *random walk* e un modello GAN con una diversa architettura interna, in possesso dall'azienda ospitante prima dell'inizio del progetto di tesi. Entrambi questi modelli verranno illustrati nelle sezioni seguenti. Inoltre si è scelto di utilizzare come *baseline* il modello proposto nel capitolo 4, allenato però su un singolo indice (GAN 1) e su 5 indici azionari (GAN 5).

Per confrontare i risultati è necessario analizzare gli indici singolarmente siccome non tutte le *baseline* sono in grado di trattare il problema in modo agnostico.

5.3.1 Modello Random Walk

Il modello *Random Walk* genera i valori futuri della sequenza basandosi su una euristica che dice semplicemente che i 120 valori in ingresso siano provenienti da una distribuzione di tipo gaussiano e che quindi i 5 valori futuri appartengano ancora a questa distribuzione.

Dopo aver ricevuto in ingresso una sequenza opportunamente modellata come illustrato nell'intera sezione 5.1 , questo modello ne calcola la media μ e la deviazione standard σ per creare una distribuzione gaussiana. I 5 valori successivi vengono generati quindi campionando da questa distribuzione gaussiana di media μ e deviazione standard σ.

5.3.2 Sequence-to-Dense GAN

Questo modello sfrutta anch'esso la tecnologia GAN ed è stato implementato dall'azienda ospitante (23) precedentemente all'inizio di questo progetto di tesi.

Per poter ottenere risultati confrontabili a quelli del modello proposto nel capitolo 4 è stata applicata la stessa modellazione dei dati illustrata in sezione 5.1 . Questo modello è

in grado di generare sequenze prendendo in considerazione solo un singolo indice per volta e quindi per ottenere sequenze per *n* indici si dovranno utilizzare *n* diversi modelli. La sostanziale differenza a livello architetturale con il modello descritto nel capitolo 4 risiede nel generatore. In questo caso l'*encoder* del generatore, è una semplice serie di celle LSTM (esempio in Illustrazione 8) che per ogni *step* prende in ingresso il prezzo giornaliero dell'indice e suoi indicatori tecnici e produce un vettore di *encoding*, ricollegandosi almeno in questo fase alla struttura classica del modello *sequence-to-sequence (12).*

Per creare variabilità nell'output viene aggiunto al vettore di *encoding* del rumore campionato da una distribuzione gaussiana con media $\mu = 0$ e deviazione standard $\sigma = 1$. Questo vettore di *encoding*, come mostrato nell'illustrazione 32, viene poi dato in pasto a una rete neurale *fully-connected* avente 3 *layers;* il primo *layer (input layer)* che prende in ingresso il vettore di *encoding*, il secondo *layer (hidden layer)* contenente 64 neuroni e il terzo *layer (output layer)* che produce 5 valori d'uscita che saranno i prezzi generati.

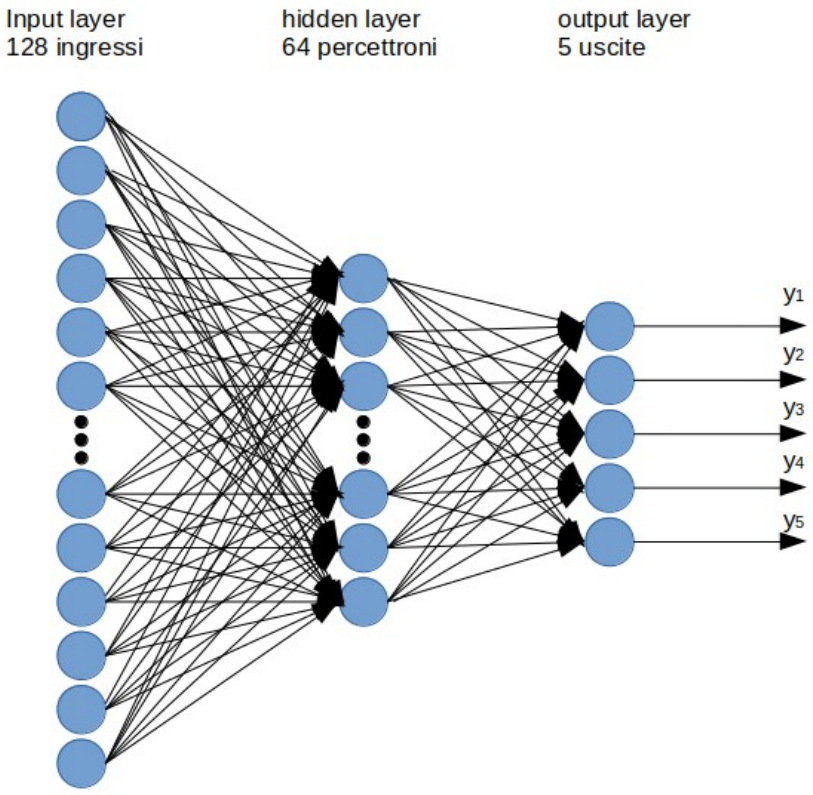

Illustrazione 32: Modello Decoder del generatore nel modello GAN sequence-to-dense.

Per il discriminatore invece non ci sono particolari differenze, ovvero è una sequenza di reti convolutive che prende in ingresso una sequenza di 125 prezzi e deve determinare se questa appartiene ai dati reali oppure se è stata generata dal generatore.

5.4 Risultati Quantitativi

In un progetto i risultati quantitativi sono i termini strettamente numerici prodotti in uscita. Nell'ambito *Machine Learning* è abbastanza immediato capire che i risultati quantitativi sono dati dai valori delle metriche prese in considerazione calcolate sul database di *test*. In questo progetto i risultati quantitativi sono dati dalla analisi delle metriche descritte in sezione 3.3 ovvero la *likelihood* e l'errore percentuale finale (FE %).

Le metriche sono state calcolate sui database di *test* dei seguenti indici: *FTSE MIB, FTSE 100, S&P 500, DAX30, TOPIX*.

Per l'analisi si sono considerate le *baseline* descritte nella sezione precedente ovvero: il modello *Random Walk* (RW), la GAN Sequence-to-Dense (GAN SD), il modello GAN del capitolo 4 allenato prima su un singolo indice (GAN 1) poi su tutti e 5 gli indici citati sopra (GAN 5), mentre il modello totalmente agnostico proposto è stato allenando su tutti e 10 gli indici descritti in sezione 2.2.5 ed avrà la sigla GAN AG.

I risultati sulla *likelihood* sono mostrati in tabella 5 e mettono in evidenza come il modello proposto riesca ad ottenere *performance* migliori rispetto a tutte le *baseline* analizzate.

Likelihood	FTSE MIB	FTSE 100	S&P 500	DAX 30	TOPIX
RW	-1.263	-1.538	-1.038	-1.110	-1.175
GAN SD	-1.287	-1.222	-1.160	-1.281	-1.267
GAN 1	-1.095	-1.026	-0.994	-1.024	-1.064
GAN 5	-1.091	-1.025	-0.992	-1.015	-1.039
GAN AG	**-1.087**	**-1.022**	**-0.989**	**-1.012**	**-1.037**

Tabella 5: Risultati quantitativi in termini di likelihood calcolati sulle diverse baseline proposte.

Un secondo risultato quantitativo dato dall'analisi dell'errore percentuale finale, ovvero la metrica secondaria utilizzata per valutare il progetto, calcolato sulle *baseline* illustrate, è mostrato in tabella 6.

In questo caso le performance del modello proposto sono leggermente peggiori ma riesce comunque ad avere la meglio rispetto alle altre *baseline.*

I buoni risultati ottenuti dalla GAN 5 mettono in evidenza la correlazione che c'è tra alcuni indici, infatti nella realtà è altamente improbabile che, se ad esempio l'indice FTSE MIB abbia un forte andamento verso il basso, allora il DAX 30 possa avere avere un forte andamento verso l'alto. Da ciò si deduce che allenare la rete con indici che abbiano in qualche modo una correlazione tra loro, possa far aumentare le *performance* del modello, confermando i risultati degli studi di fattibilità descritti nel capitolo 3.

FE%	FTSE MIB	FTSE 100	S&P 500	DAX 30	TOPIX
RW	2.792	1.738	**1.260**	2.410	2.060
GAN SD	2.616	1.673	1.376	2.270	2.490
GAN 1	3.060	2.060	1.535	2.579	3.395
GAN 5	2.473	**1.582**	1.638	**2.104**	1.987
GAN AG	**2.429**	1.667	1.610	2.179	**1.964**

Tabella 6: Risultati quantitativi in termini di FE% calcolati sulle diverse baseline proposte.

Complessivamente, in termini quantitativi, i risultati si possono dunque ritenere molto soddisfacenti.

5.5 Risultati Qualitativi

I risultati qualitativi sono tutti i risultati non numerici che possono essere estrapolati da un progetto e permettono di valutare in maniera visuale la qualità di un approccio rispetto al suo obbiettivo. Nel caso specifico di questo progetto come risultato qualitativo si andrà a confrontare visivamente le serie prodotte dalle *baseline, Random Walk* (illustrazione 33) e S*equence-to-Dense GAN* (illustrazione 34) rispetto a quelle prodotte dal modello proposto (Illustrazione 35), successivamente verrà mostrato come milgiorano le sequenze generate dalla GAN agnostica durante l'addestramento (Illustrazione 36) e infine verranno mostrati i coefficienti di *attention* che la rete assegna a una sequenza presentata ingresso (illustrazione 37).

Tutti i risultati mostrati in questa sezione si riferiscono al *dataset* di *test* dell'indice FTSE MIB.

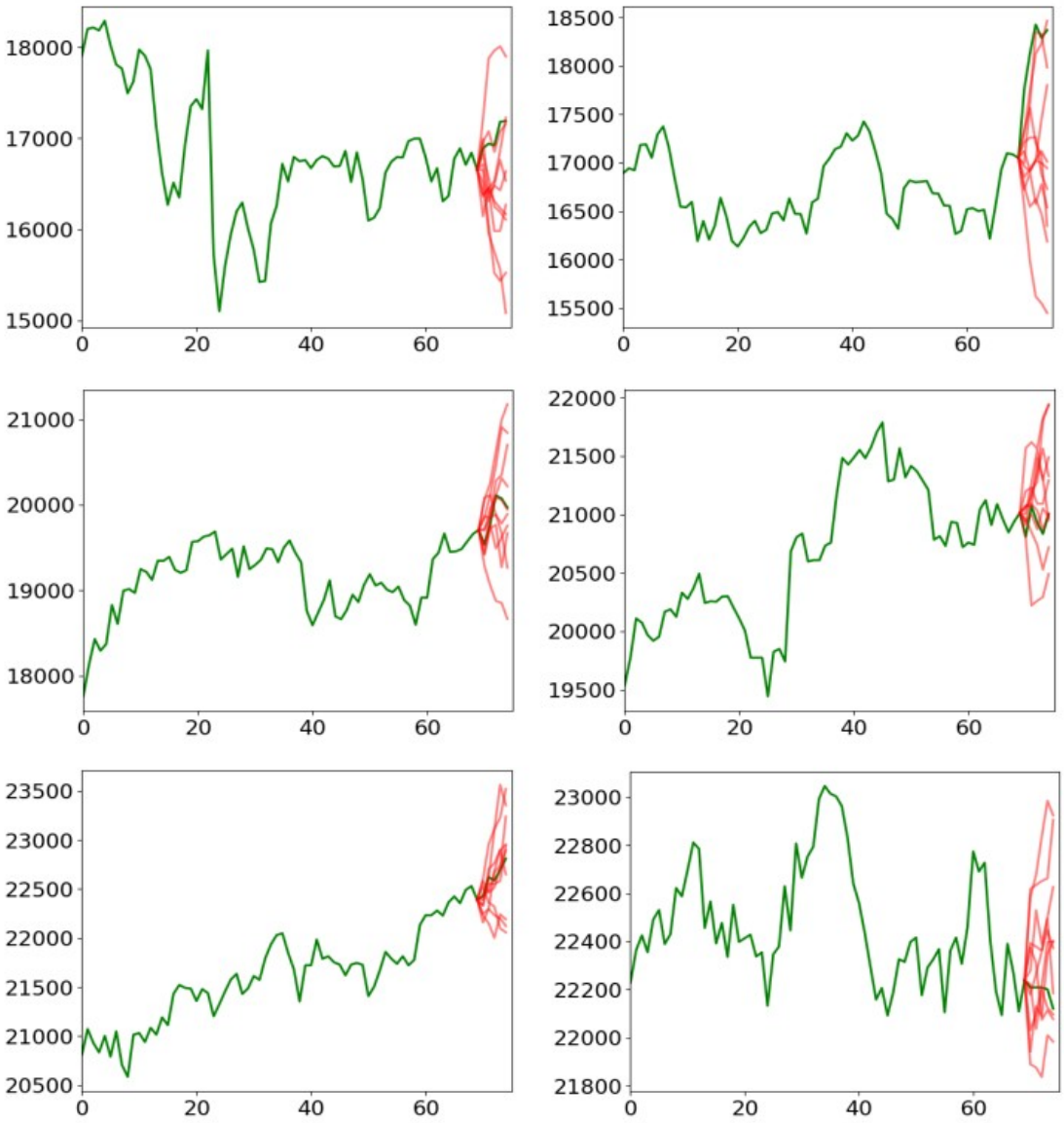

Illustrazione 33: Sequenze generate con il modello Random Walk, in diversi possibili scenari dell'indice FTSE MIB. In verde è rappresentata la sequenza reale mentre in rosso sono rappresentati 10 scenari futuri generati.

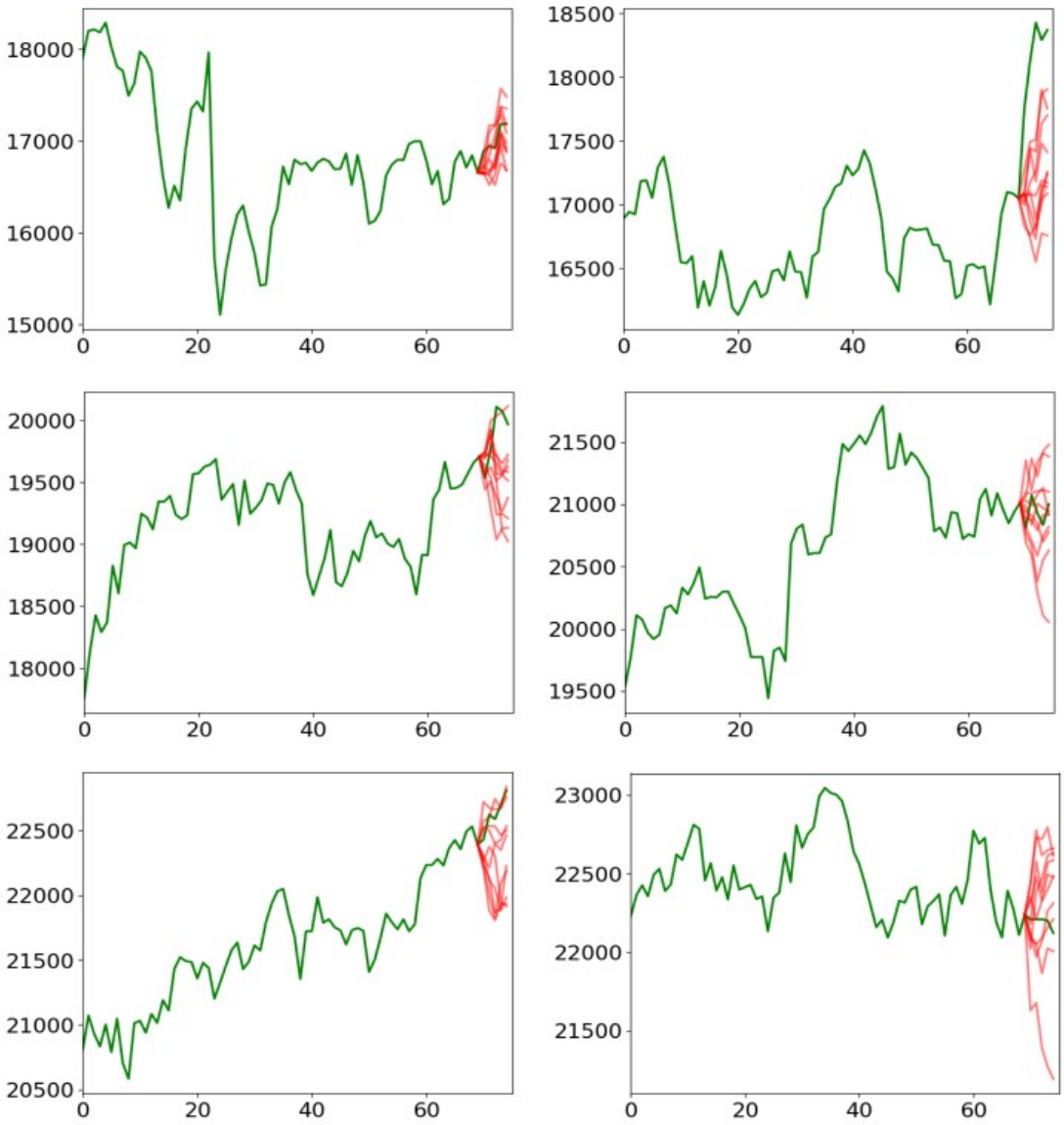

Illustrazione 34: Sequenze generate con il modello GAN Sequence-to-Dense, in diversi possibili scenari dell'indice FTSE MIB. In verde è rappresentata la sequenza reale mentre in rosso sono rappresentati 10 scenari futuri generati.

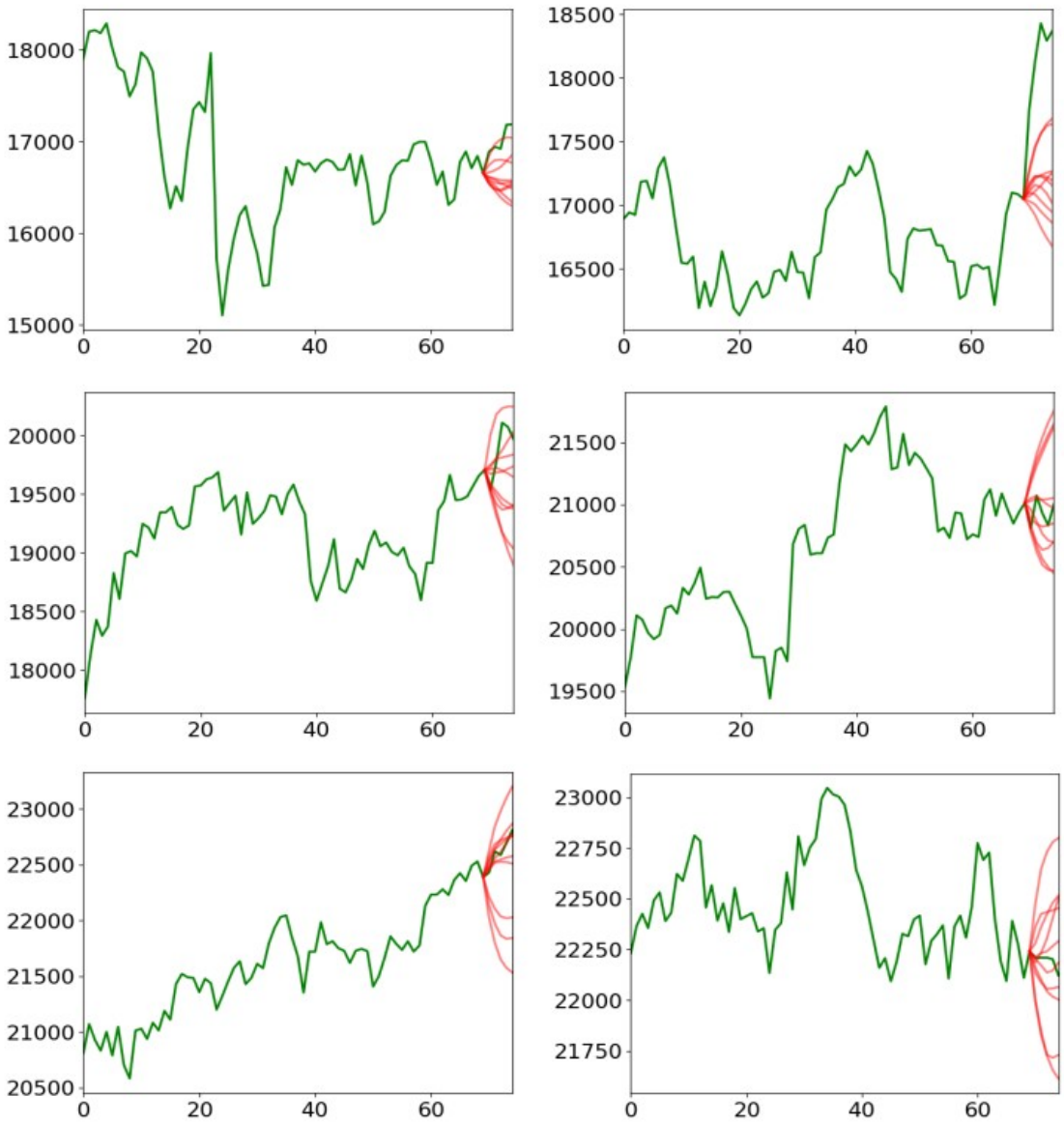

Illustrazione 35: Sequenze generate con il modello GAN agnostico, in diversi possibili scenari dell'indice FTSE MIB. In verde è rappresentata la sequenza reale mentre in rosso sono rappresentati 10 scenari futuri generati.

Illustrazione 36: Rappresentazione grafica del miglioramento della qualità delle sequenze generate, sul database di training dell'indice FTSE MIB, durante il training del modello GAN agnostico. Il primo grafico in alto a sinistra mostra le sequenze generate all'inizializzazione del modello, mentre l'ultimo grafico in basso a destra mostra le sequenze generate al termine dell'addestramento della rete.

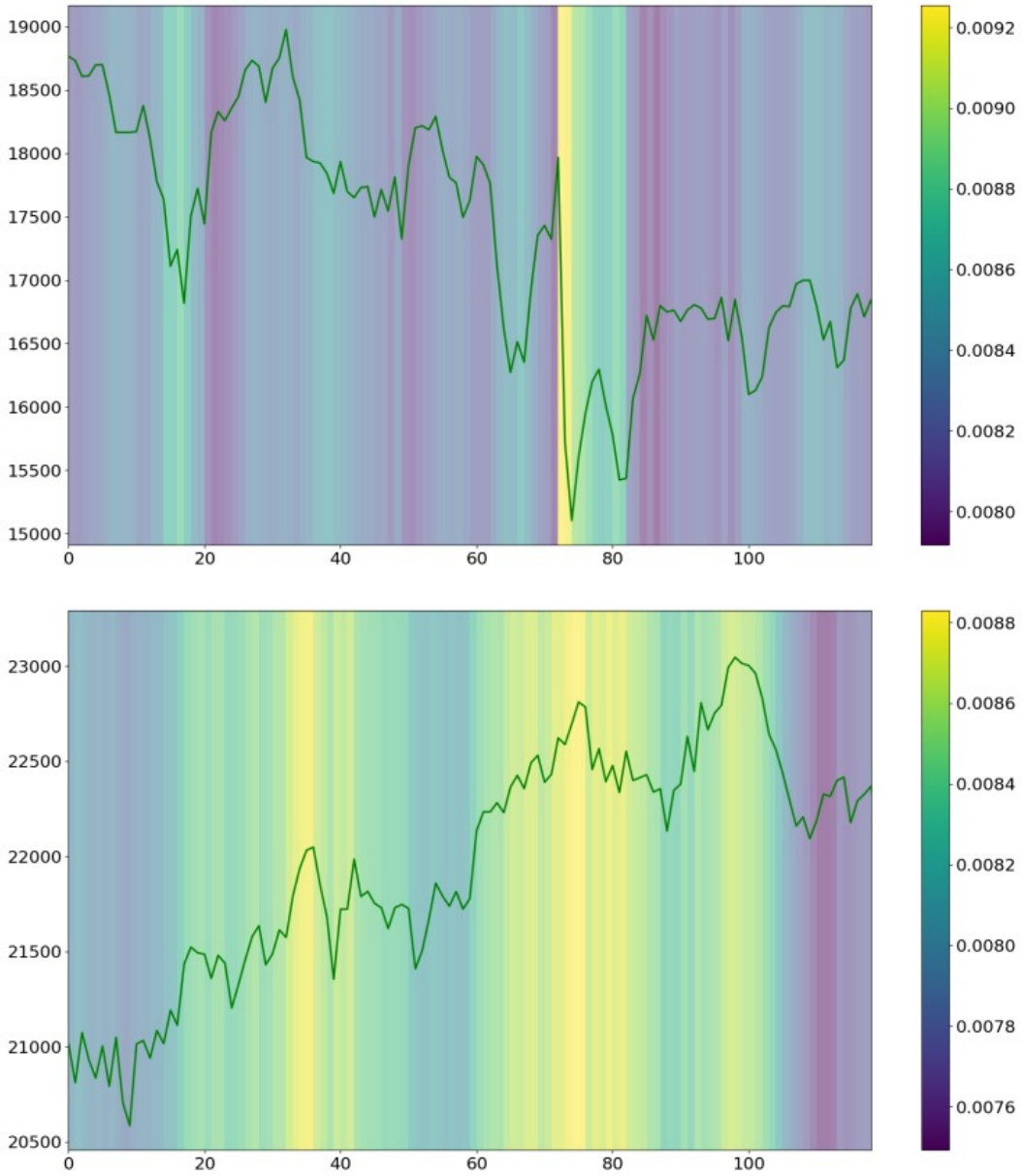

Illustrazione 37: Rappresentazione grafica del vettore di attention. I due diagrammi fanno riferimento al vettore di attention calcolato per due sequenze (di 120 valori), presenti nel database di test dell'indice FTSE MIB. In verde è rappresentata la sequenza dei valori allo scorrere del tempo, mentre il colore dello sfondo del grafico rappresenta il coefficiente di attention ad ogni istante temporale. Come mostrano le color bar sulla destra più questo coefficiente è alto più lo sfondo assume una tonalità chiara verso il giallo, viceversa meno il coefficiente di attention è alto, più lo sfondo assume una tonalità scura verso il viola.

6. Conclusioni

Lo scopo della tesi era produrre uno strumento, possibilmente utilizzando la tecnologia GAN, che andasse a potenziare e permettesse l'industrializzazione di un modello provvisorio già sviluppato dall'azienda ospitante precedentemente l'inizio di questo progetto di tesi, per affrontare la previsione dell'andamento delle serie finanziare.

Si è cercato di raggiungere questo obbiettivo, sicuramente ambizioso, attraverso diverse fasi. Partendo da uno studio accurato della letteratura allo stato dell'arte riguardo le ultime tecniche di *deep learning* e in particolare della tecnologia GAN, insieme ad uno studio introduttivo di quello che è stato il campo di applicazione del progetto ovvero la finanza e in particolare il mercato azionario.

Sono stati successivamente eseguiti diversi test approfonditi, per andare a studiare il problema ed estrarre elementi importanti per indirizzare il progetto verso una soluzione e capire quale correlazione avessero tra loro i dati utilizzati, giungendo alla conclusione, che era possibile ottenere migliori risultati prendendo in considerazione più indici azioni contemporaneamente siccome quest'ultimi pur essendo indici diversi, hanno elementi in comune che permettono di allenare con maggiore efficienza quella che poi è stata chiamata GAN agnostica, ovvero una GAN in grado di essere allenata contemporaneamente su un qualunque tipo di indice azionario.

Si è passati poi, allo studio e alla ricerca del miglior modello da utilizzare per risolvere questo tipo di problema, applicando alcune delle più recenti tecnologie nel campo del *deep learning* come il modello *sequence-to-sequence* e l'*attention.* Di pari passo con lo studio e la ricerca è avvenuta anche l'implementazione a livello *software*, affrontando diversi problemi e permettendo quindi di acquisire familiarità con i più utilizzati *frameworks* in campo *machine learning.*

Oltre alla ricerca di un modello valido, si è ricercato anche quale fosse la migliore metrica per valutare il modello e si giunti alla decisione di utilizzare la *likelihood* per catturare al meglio la qualità degli scenari generati rispetto a quello reale.

Una volta ottenuto un modello soddisfacente e una sua corrispondente implementazione funzionante, la fase successiva è stata quella di confrontare i risultati con quelli di alcune *baseline* scelte.

Il modello presentato sicuramente non permette di avere una previsione affidabile al 100% del futuro di una serie finanziaria, ma il risultato ottenuto si può ritenere più che positivo siccome fornisce forti indicazioni su quelli che potrebbero essere gli scenari futuri di una serie finanziaria ed essendo più affidabile di tutte le *baseline* proposte.

Il modello allo stato attuale riceve in ingresso una sequenza di 120 prezzi e per ogni prezzo i rispettivi indicatori tecnici. Il generatore calcola il vettore di *attention* e genera un numero arbitrario di scenari futuri. Questi scenari futuri vengono mostrati al discriminatore che scarta quelli che ritiene generati e infine viene fornito in uscita un grafico ad intervalli di confidenza per gli scenari futuri, come mostrato nell'illustrazione 38.

Il risultato ottenuto permette di studiare le informazioni statistiche come la volatilità, permette di testare strategie di investimento ed eseguire un analisi dei rischi.

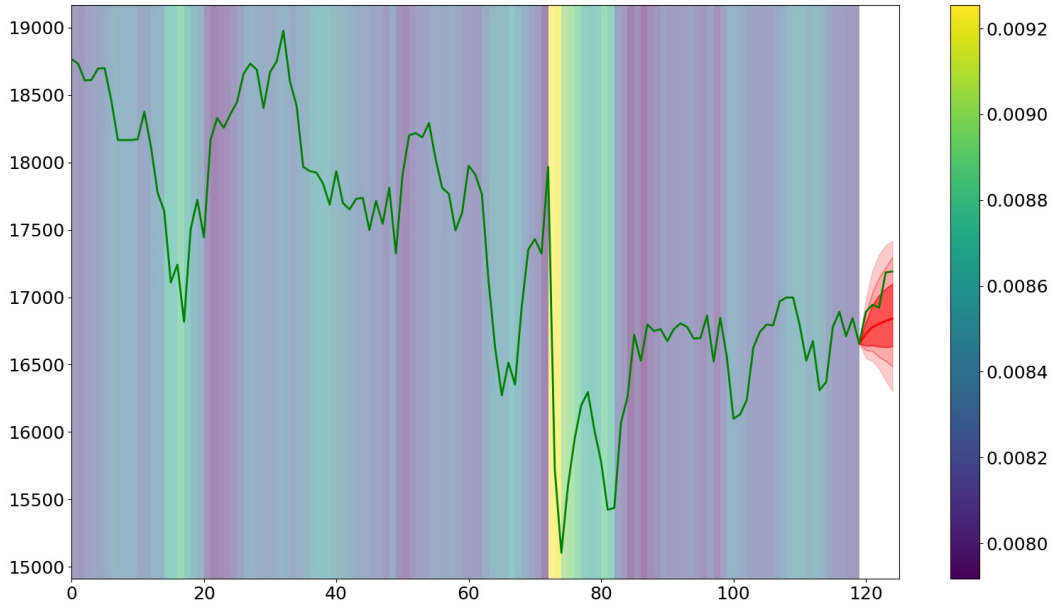

Illustrazione 38: Risultato finale che mostra gli intervalli di confidenza del generatore per le serie che sono passate positivamente attraverso l'esame del discriminatore. In verde è rappresentata la sequenza reale lungo i 125 giorni. Il ventaglio di scenari futuri che parte dal giorno 120 ha diverse gradazioni di colore: la linea centrale in rosso è la mediana e l'intervallo con la tonalità di rosso più accesa rappresenta i valori dal sedicesimo percentile fino al ottantaquattresimo percentile. Il colore di background rappresenta il vettore di attention calcolato per la sequenza.

6.1 Sviluppi futuri

Pur avendo raggiunto un risultato soddisfacente, si pensa che il modello proposto possa avere ancora margini di miglioramento apportando alcune modifiche. Verranno ora descritte le principali idee di sviluppo:

- Cambiare il metodo con cui il modello viene allenato. Attualmente il modello viene addestrato pescando casualmente dal *dataset* serie finanziarie, che possono provenire da indici diversi ma anche da instanti temporali diversi. É ragionevole pensare che dando in ingresso alla rete sequenze di diversi indici appartenenti però allo stesso istante temporale le prestazioni possano migliorare considerando la forte correlazione che gli indici hanno tra di loro.
- Con la stessa idea del punto precedente, potrebbe essere utile per migliorare le performance generare contemporaneamente gli scenari futuri di diversi indici azionari. Questo perché, se per esempio viene generato un andamento al rialzo per un indice europeo al tempo t e per lo stesso istante temporale t viene generato uno scenario al ribasso per un altro indice europeo, pur essendo magari entrambi gli scenari probabili se considerati singolarmente, diventano non compatibili se presi in considerazione contemporaneamente. Quindi con la collaborazione tra generatore e discriminatore potrebbe essere efficace prendere in ingresso n sequenze appartenenti ad n indici e generare, n scenari uno per ogni indice, compatibili tra loro.
- Valutare eventuali modifiche al modello del generatore ispirandosi ad un modello in stile *variational* (24) e successivamente verificare se questo porti o meno un miglioramento delle prestazioni.
- Considerare l’opportunità di utilizzare la GAN proposta per generare valori a diverse frequenze, ovvero provare a cambiare l’orizzonte di generazione. Dal generare il prezzo futuro di giorno in giorno si potrebbe provare a generare i prezzi futuri di ora in ora o addirittura di minuto in minuto, andando a studiare se tutte le considerazioni fatte in questo progetto di tesi rimangono valide anche per alte frequenze.

Ringraziamenti

Grazie a mia mamma, la donna con le spalle più forti che conosco che con tanti sacrifici ha permesso che tutto questo fosse possibile.
Grazie al mio papà che con tanti insegnamenti mi ha dato un sogno in cui credere, e che pur non essendo più tra noi mi ha sempre guidato nel mio percorso.
Grazie a mia nonna Lucia che mi ha insegnato cos'è l'amore, la pazienza ed il valore delle piccole cose.
Grazie ai miei nonni Antonio e Mario, i miei maestri di vita, i miei angeli, di cui custodisco gelosamente meravigliosi ricordi.
Grazie alla mia ragazza Lorena per tutti i momenti belli, per sopportare i miei mille sbalzi d'umore e per avermi sempre sostenuto anche negli attimi più cupi di questo percorso e grazie alla sua famiglia che mi ha sempre accolto come un figlio e con grandi mangiate.
Grazie a mio fratello Federico e alla mia interminabile lista di cugini, Dayana, Francesca, Elisabet, Giuseppe, Antonio, Giorgia, Felice, Davide, Christian, Marianna, Barbara, Mario, Gerardo, Lorenzo, Sara, Nadia e il piccolo Liam, con cui sono cresciuto e ho condiviso tanti giochi e risate.
Grazie ai miei amici Andrea e Pietro compagni di tanti avventure, follie e idee visionarie.
Grazie ai miei amici Marco, Pasquale e Aldo con cui ho passato le più belle serate e con cui è sempre bello ridere e parlare.
Grazie al mio amico Singh con cui ho condiviso questo percorso e che ha mi ha sempre dato consigli utili e inutili.
Grazie a i miei zii Rosaria, Enzo A., Cristina, Gennaro, Enzo L., Alessandro, Nicoletta, Cinzia, Franco, Marco, Pina e Lina perché con ognuno di voi ho passato bei momenti e ho sempre ricevuto tanti insegnamenti.
Grazie al professor Calderara, per avermi insegnato una materia meravigliosa come il Machine Learning e avermi indirizzato durante il percorso di tesi.
Grazie a Jacopo, Daniele e ai ragazzi del team di Axyon AI per avermi affiancato, avermi dedicato tempo prezioso e permesso di realizzare un bellissimo progetto di tesi.

Grazie a tutti voi non solo per aver riempito la pagina più bella della mia tesi ma per aver riempito tutte le pagine della mia vita.

Bibliografia

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

(2) François Chollet. Keras. 2015.

(3) «A logical calculus of the ideas immanent in nervous activity.» Walter Pitts , Warren McCulloch.

(4) Krizhevsky, Alex, Vinod Nair, and Geoffrey Hinton. "The CIFAR-10 dataset."online: http://www.cs.toronto.edu/kriz/cifar.html (2014).

(5) P.J. Werbos. Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences. Harvard University, 1975.

(6) D.H. Hubel, T.N. Wiesel, “Receptive fields, binocular interaction and functional acrhitetture in the cat’s visual cortex” Neurophisiology Laboratory, Harvard Medical School, 1961.

(7) 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.

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

(9) Sepp Hochreiter, Jurgen Schmidhuber, “Long Short-Term Memory”, 1997.

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

(11) Yann LeCun “The MNIST database of handwritten digits” http://yann.lecun.com/exdb/mnist/

(12) Ilya Sutskever, Oriol Vinyals, Quoc V.le “Sequence to Sequence Learning with Neural Networks” 2014 arXiv:1409.3215v3.

(13) Dzmitry Bahdanau, KyungHyun Cho, Yoshua Bengio, “Neural Machine Translation by jointly learning to align and traslate” 2016.

(14) 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.

(15) Emily Denton, Sam Gross e Rob Fergus. «Semi-Supervised Learning with Context-Conditional Generative Adversarial Networks» (2016).

(16) Provost F.,Fawcett T. & Kohavi R. (1998). *The case against accuracy estimation for comparing induction algorithms*. Proceeding of the 15th International Conference on Machine Learning (pp. 445-453).

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

(18) Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford,Xi Chen, «Improved Techniques for Training GANs» (2016) arXiv:1606.03498.

(19) "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift" Sergey Ioffe, Christian Szegedy (2015)

(20) Soumith Chintala et al. How to Train a GAN? Tips and Tricks to Make GANs Work. 2016.

(21) "On the Momentum term in Gradient Descent Learning Algorithms" Ning Qian pp.145–151, 1999.

(22) Alex Hernandez-Garcıa & Peter Konig "Do deep nets really need weight decay and dropout?" 2018.

(23) "FinGAN - Generating Stock Price Sequences to Simulate Future Market Scenarios" M.Casolari, J.Credi, D.Grassi, S.Calderara. 2018

(24) "Auto-Encoding Variational Bayes" Diederik P. Kingma, Max Welling, (2014) Arxiv: 1312.6114