Tesi di Laurea in Ingegneria Biomedica Magistrale

ANALISI E SVILUPPO DI RETI NEURALI MULTI-INPUT PER LA PREDIZIONE DEL LIVELLO GLICEMICO IN PAZIENTI DIABETICI DI TIPO 1

SALVATORE PUZZO

Relatori

Patti Edoardo

Aliberti Alessandro

Santa Di Cataldo

Acquaviva Andrea

Politecnico di Torino

Ottobre 2019

Indice

Sommario

Il diabete mellito è una malattia che si vince solo quando la si conosce. La conoscenza passata permette di risolvere le complicazioni in atto, mentre la conoscenza del futuro permette di evitarle. In un paziente diabetico, infatti, predire il livello glicemico significa gestire con tempestività ed efficacia i possibili stati glicemici nocivi futuri, che possono portare nel peggiore dei casi a chetoacidosi e coma. E' proprio in questo scenario che inserisce il lavoro Tesi, che si pone come obiettivo quello di approfondire e adattare le tecniche di Machine Learning per predire il livello glicemico di pazienti affetti da diabete mellito di tipo 1, ottenuto grazie ai dispositivi di monitoraggio continuo. Una volta illustrate le basi della malattia e la teoria del neurone artificiale, saranno mostrati gli elementi che caratterizzano le reti Convoluzionali, note come CNN, e un particolare tipo di rete ricorrente, nota come LSTM. I risultati mostrano che quest'ultima tipologia, inserita in un innovativo costrutto chiamato Encoder-Decoder, raggiunge dei valori paragonabili allo stato dell'arte e si propone come una solida architettura per la costruzione di un modello che predice i valori glicemici sia nel breve che nel medio-lungo termine. La validazione avverrà secondo le più utilizzate e accettate metriche sia dal punto di vista analitico (RMSE, FIT, MAD, etc..) sia dal punto di vista clinico (Clarke Error Grid).

Riconoscimenti

Elenco delle figure

Elenco delle tabelle

Listings

Capitolo 1

Introduzione

1.1 Il diabete mellito

La Tesi si pone come obiettivo principale quello di approfondire un metodo tramite il quale predire il livello glicemico di pazienti affetti da diabete *mellito di tipo 1*, attraverso l'utilizzo di modelli basati su Reti Neurali. In questa prima parte, pertanto, verranno definiti i numerosi fattori a contorno, dall'origine della malattia fino alle più moderne tecniche di valutazione e monitoraggio del livello glicemico.

1.1.1 Definizione ed epidemiologia

Con *diabete mellito* si intende una serie di alterazioni nel metabolismo che sono caratterizzate da stati iperglicemici dovuti ad un parziale o totale deficit nella produzione di insulina [1], un ormone prodotto nel pancreas dalle cellule β presenti all'interno delle isole di Langerhans.

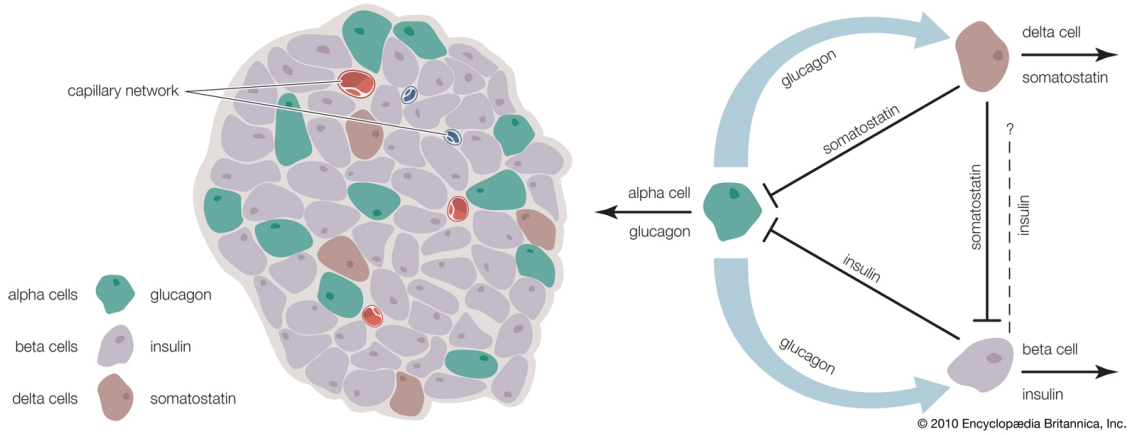


Figura 1.1: Le isole di Langerhans contengono, oltre alle cellule beta, anche altri tipi di cellule chiamate alfa e delta, che producono rispettivamente glucagone e somatostatina. Fonte: Encyclopaedia Britannica, Inc.

Solamente nel 2017, questa malattia ha causato 4 milioni di decessi, classificandosi come la quinta causa di mortalità a livello mondiale. In questo periodo storico, le persone affette da diabete sono circa 425 milioni[2] e il tasso di crescita non accenna a diminuire, anzi si stima che si arriverà a più di 625 milioni di diabetici nei

prossimi 30 anni, compresi tutti nella fascia di età tra i 20 e i 79 anni, rappresentando di fatto non solo una crisi di salute, ma, per certi aspetti, una vera e propria catastrofe[3] socio-sanitaria globale.

1.1.2 Classificazione e cause del diabete

Nonostante esistano diverse tipologie di diabete, il filo conduttore che lega le numerose sindromi cliniche legate a tale malattia è l'iperglicemia, cioè un stato di glicemia sanguigna a digiuno con valori superiori alla norma. Il diabete può essere classificato nelle sue 3 forme più diffuse [1] in:

1. diabete tipo 1 (**DM1**), presente nel 5/10% dei casi; è un tipo di diabete caratterizzato da totale deficit insulinico dovuto alla distruzione delle cellule β pancreatiche. La patogenesi di tale malattia, che causa la distruzione della massa β cellulare, ha caratteri fortemente genetici coesistenti con fattori legati sia al sistema immunitario, considerato il principio autoimmune del disturbo, sia a possibili infezioni virali [4]

2. diabete tipo 2 (**DM2**), rappresenta il tipo di diabete più diffuso, contemplando il 90/95% dei casi. Le fasi iniziali di tale disturbo sono caratterizzate generalmente da iper-attività compensatoria delle cellule β, che provoca l'insulino-resistenza ed infine iperglicemia [5]

3. diabete gestazionale, diagnosticato al secondo o terzo trimestre di gravidanza; è associato all'aumentata richiesta insulinica da parte del feto, che genera alterazioni metaboliche nella madre con effetti sul livello glicemico ematico. Nonostante solitamente il soggetto ritorni a stati normo-glicemici dopo la gravidanza, tale tipologia di diabete può comunque rappresentare un fattore di rischio per l'insorgenza di diabete di tipo 2 nel corso degli anni.

1.1.3 Diagnosi

Il range normo-glicemico è solitamente compreso tra i valori di 70 e 110 mg/dL (3,9 - 6,1 mmol/L).

E' possibile effettuare la diagnosi del diabete attraverso il controllo dei valori di glucosio ematico con [1]:

- **FTP** (Fasting Plasma Glucose), con glicemia $>$ 126 mg/dL, corrispondenti a 7 mmol/L, a digiuno nelle precedenti 8 ore

- **OGTT** (Oral Glucose Tolerance Test) con glicemia $>$ 200 mg/dL, corrispondenti a 11,1 mmol/L, alla distanza di 2 ore dalla somministrazione di 75g di glucosio

- glicemia casuale $>$ 200 mg/dL , o 11,1 mmol/L

- **Test HbA1C** o test sull'emoglobina glicata, con valore $>$ 6,5%

L'emoglobina glicata, la forma più diffusa di glico-emoglobina, rappresenta l'esito della condensazione delle molecole di glucosio con gli amino-gruppi liberi dell'emoglobina (*HbA1c*), presente nei globuli rossi.

I globuli rossi hanno nell'organismo un ciclo di vita di circa 120 giorni e l'emoglobina glicata ha una concentrazione proporzionale al livello di glucosio nel sangue. Il test HbA1C mostra infatti l'andamento glicemico degli ultimi 2-3 mesi, elevandosi come valida tecnica di valutazione del controllo glicemico a lungo raggio e prezioso strumento di monitoraggio di soggetti ad alto rischio di sviluppare il diabete e/o eventuali terapie in atto [1][6].

1.1.4 Complicanze del diabete

Le complicanze associate al diabete sono riconducibili alla costante presenza di uno stato iperglicemico e sono classificabili come acute e croniche, sia microvascolari che macrovascolari.

Gli esempi più diffusi di complicanze acute sono :

- **chetoacidosi diabetica**, caratterizzata da iperglicemia (>250 mg/dl), acidosi metabolica (pH < 7.3, HCO3- < 15 mmol/L) e iperchetonemia (>5 mmol/L). Quest'ultimi sono tutti fattori che generano un'alterazione metabolica grave, portando il paziente fino al coma [7]

- **ipoglicemia**, cioè l'abbassamento del livello glicemico del sangue al di sotto di 70 mg/dL. Negli stadi più gravi può portare al coma, anche più rapidamente rispetto alla chetoacidosi.

I casi più diffusi di complicanze croniche dovute al diabete sono:

- **nefropatia**, **neuropatia** e **retinopatia**, tra la complicanze microvascolari. Quest'ultima è considerata la più frequente e causa la diminuzione della vista che può condurre, nei casi più estremi, alla cecità

- **coronaropatia** e **arteriopatia periferica**, in cui l'infiammazione cronica e il danno sul tessuto endoteliale, favorisce la formazione graduale della placca aterosclerotica [8] che può portare effetti nefasti per il paziente, quali angina, dispnea o attacchi cardiaci.

Infine, una comune ripercussione di tali complicanze croniche sul corpo umano si manifesta come disfunzione strutturale di uno o più arti, tipicamente il piede, con conseguente formazione di ulcera, fenomeno noto come **piede diabetico**. Il peggioramento delle condizioni vascolari e ulcerose in stadi di infiammazione estremi rendono necessaria l'amputazione parziale o totale dell'arto interessato [9].

1.1.5 Monitoraggio del segnale glicemico

I dispositivi CGM (Continuous Glucose Monitoring) sono il risultato dell'avanzamento tecnologico sensoristico, unito al progressivo aumento della consapevolezza dell'importanza del monitoraggio costante dell'andamento glicemico. Tali dispositivi permettono di rilevare il valore del livello glicemico nel sangue, permettendo quindi di ottimizzare le dosi insuliniche da infondere al paziente e tenere sotto costante osservazione l'andamento glicemico, sia nel breve che nel lungo termine. Da un punto di vista chimico, il sensore, associato a tali dispositivi, misura concentrazione di glucosio nel fluido interstiziale, mezzo fondamentale per il passaggio degli

elementi nutritivi dalla circolazione dei vasi sanguigni alle cellule. La concentrazione di glucosio nel fluido interstiziale è simile a quella ematica quando la glicemia è stabile, mentre se quest'ultima è in rapida variazione, si genera un ritardo notevole compreso tra i 5 e i 10 minuti [10], mantenendo accettabili le misurazioni con un certo grado di tolleranza.

La svolta tecnologica che porterebbe un uso continuo di tali dispositivi deve comunque scontrarsi con diversi fattori: i pazienti e le famiglie, infatti, segnalano oltre al fattore economico, anche dolore da inserimento per via del sensore ad ago-cannula, problemi di precisione del valore rilevato e irritazione della pelle [11]. Con i dispositivi di ultima generazione è possibile, non solo conoscere il proprio trend glicemico passato e presente, ma, tramite un sistema di allarme e/o avvisi, correggere tempestivamente le alterazioni glicemiche in corso.

Tutte le tipologie di dispositivi CGM [12] in commercio si suddividono in due grandi famiglie:

- dispositivi CGM per il monitoraggio continuo del glucosio in tempo reale (**rtCGM**);
- dispositivi flash CGM (**FGM**).

Entrambe le tipologie hanno 3 elementi in comune: un sensore per rilevare il livello di glucosio interstiziale, solitamente un ago-cannula inserito nel sottocute; un trasmettitore che inoltra le letture effettuate dal sensore e un ricevitore, o monitor, che ne mostra il monitoraggio. Per quello che concerne le differenze, solamente il rtCGM riesce ad avvisare gli utenti di eventuali sbalzi glicemici, mentre il FGM riesce a identificare tali pattern solo dopo aver scansionato fisicamente il sensore con il lettore, limitandone l'effettivo uso solamente quando necessario. Il collegamento con la pompa insulinica è possibile solo con i dispositivi di lettura in tempo reale, mentre la calibrazione da parte dell'utente non è necessaria per i dispositivi flash, ma obbligatoria per quasi tutti i dispositivi rtCGM.

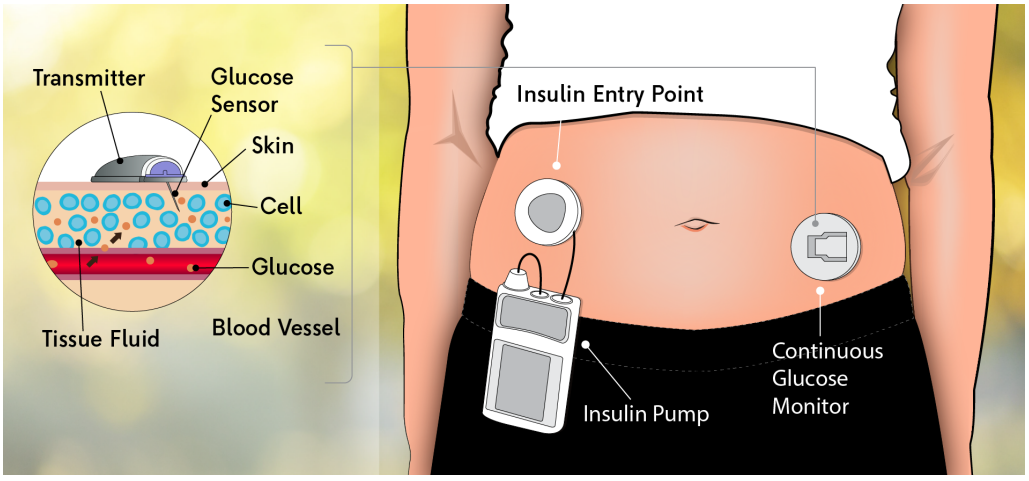


Figura 1.2: L'immagine rappresenta i tre elementi elementi che costituiscono un sistema CGM. Fonte: ontrackdiabetes.com

1.2 Gli standard per le metriche e la visualizzazione del segnale glicemico

La terapia diabetica a base insulinica è un argomento molto spigoloso, in quanto ogni paziente reagisce in diversi modi sia alle dosi insuliniche che nel metabolismo quotidiano. Con l'utilizzo del test dell'emoglobina glicata, è possibile conoscere un valore percentuale importante, cioè la concentrazione di emoglobina glicata nel sangue nelle ultime settimane e, in base a quel valore, si pongono le basi per effettuare un piano terapeutico adatto. Tuttavia ciò che non si evince è l'andamento giornaliero del paziente e se nell'arco della giornata è stato esposto ad eventi legati a ipoglicemia o iperglicemia. Negli ultimi 20 anni, l'avvento dei dispositivi CGM ha migliorato enormemente la consapevolezza del paziente del proprio stato glicemico e il naturale progresso tecnologico ha portato alla luce diverse perplessità su come interpretare la grande mole di dati a disposizione e sul corretto utilizzo dei dispositivi a monitoraggio continuo. Nel 2017, la Advanced Technologies & Treatments for Diabetes (ATTD), un ente che annualmente illustra lo stato di avanzamento tecnologico e scientifico sul panorama diabetico, ha indetto una conferenza internazionale [12] in cui esperti del settore, medici, ingegneri, ricercatori e perfino pazienti diabetici hanno cercato delle soluzioni a tali perplessità, fornendo una linea guida per l'utilizzazione, l'interpretazione e la comunicazione dei dati CGM nelle cure cliniche e nella ricerca.

1.2.1 Le linee guida degli esperti

Una delle tematiche più importanti su cui si è soffermata la conferenza è stata la consapevolezza della mancanza di un protocollo standardizzato della comunicazione dei dati, anche in combinazione con altre valutazioni cliniche, che possa generare un'adatta terapia per lo specifico paziente per aiutarlo nel controllo efficace del proprio livello glicemico.

Sono state quindi presentate 14 metriche di riferimento per una migliore interpretazione e comunicazione dei dati clinici, che saranno elencate come segue:

1. glucosio medio;
2. percentuale di tempo in ipoglicemia di livello 2 (< 54 mg/dL);
3. percentuale di tempo in ipoglicemia di livello 1 (> 54, < 70 mg/dL);
4. percentuale di tempo in regime fisiologico (> 70, < 180 mg/dL);
5. percentuale di tempo in iperglicemia di livello 1 (> 180, < 250 mg/dL);
6. percentuale di tempo in iperglicemia di livello 2 (> 250 mg/dL);
7. variabilità glicemica, sia primaria come CV (*Coefficent of Variation*) che secondaria come SD (*standard deviation*). L'andamento glicemico è stabile se CV $< 36\%$. In formule, la SD, o deviazione standard è data da:

$$SD = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N}(x_i - \overline{x})^2} \tag{1.1}$$

dove il valore medio su N campioni è

$$\bar{x} = \frac{1}{N} \sum_{i=1}^{N} x_i \tag{1.2}$$

mentre il CV, o Coefficiente di variazione, è il valore in % dato da:

$$CV = \frac{SD}{\bar{x}} * 100 \tag{1.3}$$

8. eA1c, valore stimato di emoglobina glicata, espresso dal valore di GMI [13];
9. dati glicemici segnalati in tre intervalli temporali (veglia, sonno, 24 ore), considerando che lo stato di veglia è calcolato dalle 6:00 fino a mezzanotte, mentre il sonno da mezzanotte alle 6:00;
10. sufficienza dei dati: minimo 2 settimane di dati da analizzare;
11. sufficienza dei dati utilizzabili: 70-80% delle possibili letture CGM in un periodo di 2 settimane;
12. episodi di iperglicemia o ipoglicemia, usando una definizione standard di episodio, inteso come numero di eventi iperglicemici o ipoglicemici consecutivi per una durata di almeno 15 minuti;
13. AUC (Area Under Curve), o area sotto la curva, parametro molto importante soprattutto per scopi di ricerca, che misura l'esposizione glicemica in base al periodo;
14. rischio di ipoglicemia e iperglicemia tramite gli indici raccomandati: LBGI per la bassa glicemia e HBGI per l'alta glicemia [14].

1.2.2 Lo standard di visualizzazione

E' stata aggiunta alle 14 metriche di riferimento anche un'ulteriore raccomandazione sullo standard di visualizzazione dei dati glicemici tramite l'utilizzo di un AGP [12], cioè un Ambulatory Glucose Profile.

L'AGP è un prezioso standard di visualizzazione che nasce nel 1987, tramite il lavoro congiunto di R.Mazze con i colleghi dell'A. Einstein College of Medicine, che sfruttava dati provenienti da automonitoraggio tramite pungi-dito [15]. Successivamente, tale strumento si è evoluto insieme ai dispositivi, conformandosi con i moderni CGM, sia per i *real time* che per i *flash*. Nel corso del tempo [16], tutte le aziende private che producono dispositivi per il monitoraggio del glucosio si sono completamente uniformate sull'utilizzo dell'AGP, fornendo un resoconto grafico di facile comprensione della tipica giornata del paziente.

L'AGP oggi ha raggiunto un alto grado di capacità informativa e descrittiva, classificandosi come uno standard necessario per tutti gli stakeholder.

La visualizzazione dall'AGP raccoglie le registrazioni di 14 giorni di glucosio in mg/dL, numero minimo di giorni per comprendere l'andamento glicemico nei successivi 30 giorni[17], rappresentandoli in un unico grafico della durata di 24 ore che corrispondono a 288 letture di un dispositivo CGM, con una frequenza di campionamento di 5 minuti.

Per gestire la variabilità giornaliera del glucosio, sono stati suddivisi i dati nelle distribuzioni percentili corrispondenti al 10°, 25°, 50°, 75° e 90° percentile. Nella fattispecie:

- il 50° percentile, o mediana, è la linea che mostra i valori glicemici all'interno di una giornata tipo e come si sviluppa solitamente l'andamento della curva glicemica;
- la distribuzione tra il 25° e il 75° percentile rappresenta il range interquartile, che copre il 50% dei dati e mostra l'andamento glicemico tipico;
- i valori che corrispondono al 10° e al 90° percentile indicano gli estremi di variabilità glicemica.

In seguito è presentato un esempio di *AGP report*, prelevato dall'articolo di riferimento [12] in cui, oltre allo standard di visualizzazione, sono presenti le note 14 metriche elencate poc'anzi.

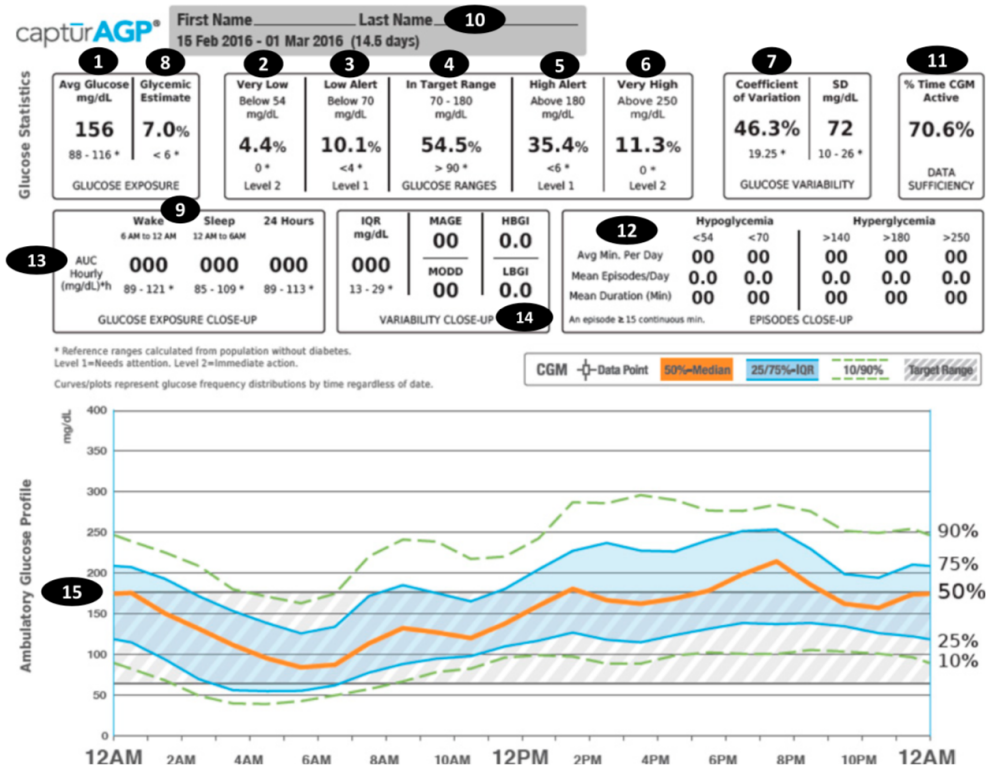


Figura 1.3: Esempio di AGP tratto da [12]. Sono visibili i numeri corrispondenti alle metriche di riferimento (1-14) presenti nella sezione 1.2.1 e allo standard di visualizzazione dell'AGP (15)

1.3 Struttura della tesi ed elenco capitoli

Il primo capitolo di questa Tesi funge da **Introduzione** al contesto, in questo caso al diabete, da cui si svilupperà tutto il lavoro successivo. Il capitolo 2, chiamato **Machine Learning e Reti Neurali**, illustrerà le basi su cui si fondano queste due note branche dell'Intelligenza Artificiale e sul loro impatto sulla società moderna. Nel capitolo 3, ci si soffermerà sull'analisi dello **Stato dell'arte dei modelli di reti neurali**, sottolineando i punti forti e le debolezze che si evincono dall'analisi delle prestazioni di altri lavori prelevati dagli articoli di riferimento. Il capitolo 4 , denominato **Dataset, AGP e Reti Neurali**, mostra tutte le fasi logiche e implementative per:

- generare uno script che riesca a ricreare il report AGP in versione *home made*;
- creare il set di dati estratti dall'OhioT1DM dataset [18] utile per l'allenamento e la validazione delle reti neurali;
- assemblare, compilare e valutare tali modelli predittivi.

Nel capitolo 5 si analizzeranno i risultati ottenuti dai modelli, che sanciranno il modello migliore rispetto alle feature selezionate. Infine, nel capitolo 6 saranno tratte le **Conclusioni** dell'intero lavoro di Tesi e i possibili sviluppi futuri.
Il linguaggio di programmazione utilizzato è Python, mentre l'editor testuale scelto per la stesura della Tesi è in formato LaTeX.

Capitolo 2

Machine Learning e Reti Neurali

2.1 Il Machine Learning

La capacità di imparare a svolgere compiti tramite esperienza fa parte della crescita dell'essere umano. Appena nati, non conosciamo nulla e non siamo in grado di provvedere a noi stessi, ma, accumulando esperienza, diventiamo ogni giorno più autonomi e in grado di svolgere attività sempre più complesse.

Il Machine Learning, unendo statistica e informatica, rende le macchine in grado di apprendere come svolgere un compito specifico senza essere stato appositamente programmato per svolgerlo, proprio come un uomo nella sua fase di apprendimento e crescita. Ad esempio, per un essere umano, saper distinguere un gatto da un cane deriva dalla consapevolezza di aver acquisito le caratteristiche dell'entità "cane" e dell'entità "gatto", acquisirne i tratti distintivi (taglia, conformazione della testa, occhi, etc..) e, di conseguenza, saperli riconoscere.

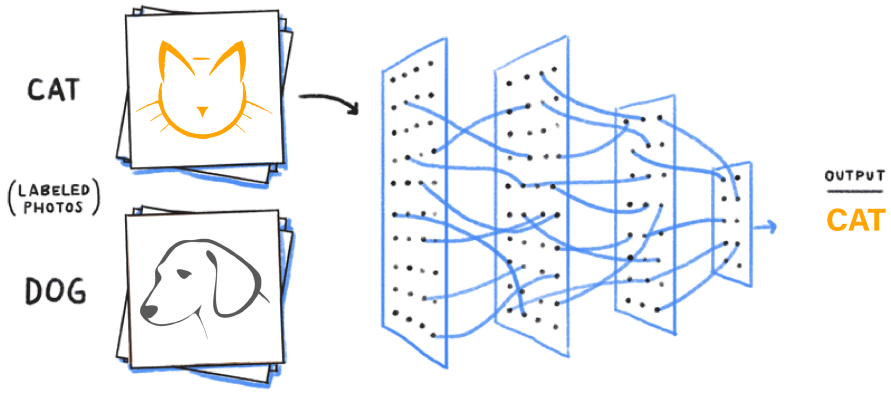


Figura 2.1: Esempio semplificato di un approccio di apprendimento automatico basato sul riconoscimento dei tratti di un gatto e di un cane attraverso una serie di filtri.

Un computer programmato per imparare acquisirà trend statistici all'interno dei dati che gli permettano di riconoscere un gatto e un cane. Estraendo conoscenza dai dati, potrebbe capire da solo che, ad esempio, i gatti hanno musi più corti e che i cani compaiono in una più larga varietà di taglie, e rappresentare queste caratteristiche in un dataset, cioè un insieme di dati, organizzati in elementi matematici di base come vettori o matrici.

In generale, ogni dataset è caratterizzato da una o più variabili che descrivono le proprietà osservabili di un oggetto, conosciute come *feature*. La lista di tutte le feature che descrivono l'oggetto è chiamata *feature vector*. I dati rappresentati dai campioni, per definizione, portano con se un carico di informazione, che però deve essere ottimizzata in modo da evitare che più feature diano la stessa informazione, fenomeno conosciuto come ridondanza.

Nella fase di apprendimento automatico, è cruciale che sia il computer ad identificare i trend all'interno delle feature e stabilire una funzione tramite cui gli input futuri verranno riconosciuti, mentre il programmatore dovrà solamente indirizzarlo su quale dataset scegliere o come applicare ciò che ha imparato, al fine di prendere decisioni future.

In questa fase ci saranno degli errori, ma più dati l'algoritmo di apprendimento riceve, più efficaci possono diventare le sue predizioni o classificazioni, quindi è di vitale importanza che il bagaglio di dati da cui il computer estrae conoscenza sia quanto più grande possibile per aumentare le capacità di generalizzazione, ma non eccessivamente grande da rendere inabile l'algoritmo ad applicare le "leggi" che lo governano.

Il Machine Learning, di fatto, rappresenta un cambiamento epocale nel paradigma informatico. La conoscenza, infatti, viene estratta dagli esempi e dall'esperienza, intesa come studio di dati tangibili, rappresentando il culmine di un processo induttivo.

Invece di scrivere un programma per un compito specifico, si collezionano numerosi esempi che specificano il corretto output per un dato input. Il programma che ne deriva avrà un aspetto diverso dal codice scritto a mano: si parla infatti di milioni di numeri e altrettanti parametri che la macchina è in grado di interpretare. Dal momento in cui la maggior parte del peso computazione e algoritmico viene auto-gestito dai computer, il costo dell'intero sistema diventa minore, perché una macchina costa meno di un programmatore esperto nel settore in cui è richiesto.
Riassumendo, i requisiti minimi per realizzare un modello di Machine Learning robusto sono:

- analisi dei dati a disposizione;
- preparazione ottimale del dataset a disposizione;
- scelta accurata di un training set e test set;
- scelta di algoritmi adatti ai task;
- processi iterativi basati sull'apprendimento automatico
- scalabilità del modello.

2.1.1 Il Machine Learning nella società moderna

Humans can typically create one or two good models a week; machine learning can create thousands of models a week
Thomas H. Davenport

Al giorno d'oggi, la maggior parte delle industrie lavorano con grandi quantità di dati, considerati un vero e proprio bene economico [19]. Tramite specifiche tecniche di Machine Learning mirate all'estrazione di conoscenza di tali dati, spesso in tempo reale, le società sono in grado di lavorare e prendere decisioni in modo più efficiente, ottenendo un vantaggio rispetto alla concorrenza. Sono innumerevoli i campi di applicazione del Machine Learning: agenzie governative, servizi finanziari, health care, pubblicità, marketing e trasporti, sono i settori di maggiore sviluppo.

Il Machine Learning nell'Healthcare

Tra tutti i settori appena citati, compreso l'Health Care, sono due gli elementi fondamentali che determinano il successo di un'applicazione basata sul Machine Learning : algoritmi intelligenti e dataset performanti. All'interno della sfera della salute e delle applicazioni correlate, i campi più importanti dove il Machine Learning ha avuto più impatto sono:

- diagnosi in imaging medico: ad esempio il caso del progetto Microsoft InnerEye®, in cui vengono sfruttate tecnologie di Machine Learning all'avanguardia per l'analisi automatica e quantitativa delle immagini radiologiche tridimensionali [20];

- chirurgia robotica: alcuni sistemi sfruttano l'apprendimento automatico per identificare distanze, movimenti o una parte del corpo specifica, ad esempio nell'identificare i follicoli piliferi nel caso di trapianto di capelli [21];

- Pancreas Artificiale, come nel caso del MiniMed 670G® [22], che combina monitoraggio e trattamento della glicemia tramite l'erogazione automatica di insulina basale;

- dispositivi di ultima generazione per la lettura del livello glicemico del sangue molto accurati, estremamente facili da utilizzare come il Freestyle Libre® [23] oppure combinati ad allarmi per glicemie, come nel caso di Medtronic Guardian3®[24], Senseonics Eversense®[25] o il Dexcom G6® [26].

2.1.2 Tipologie di apprendimento automatico nel Machine Learning

In questa sezione verrà spiegato il significato di apprendimento e le varie forme che può assumere.

"A computer program is said to learn from experience E with respect to some class of task T and performance measure P, if its perrformance at tasks in T as misureb by P, improves with experience"
Tom Mitchel

Esistono diverse tipologie di apprendimento:

- Supervisionato (*supervised learning*)
- Non Supervisionato (*unsupervised learning*)
- Rinforzato (*reinforcement learning*)

Apprendimento supervisionato

Con questo approccio, esiste una conoscenza pregressa dei dati, cioè i dati sono etichettati. L'apprendimento supervisionato ha come obiettivo quello di approssimare la funzione di mappatura tra input ed output, in modo tale che, nel caso in cui si abbiano nuovi dati di input, è possibile prevedere le variabili di output per quei dati. L'apprendimento supervisionato tratta fondamentalmente due tipi di problemi: regressione e classificazione [27].

Apprendimento non supervisionato

In questo caso [28] si usano dati che non sono etichettati, cioè senza una conoscenza pregressa. Si utilizzano solo i dati per ricercare pattern o strutture all'interno di essi. Le tecniche più utilizzate sono il clustering o la dimensionality reduction. Nel clustering, la classificazione avviene tramite un processo di raggruppamento di dati discreti in base alla similarità intra-classe e differenze inter-classe, mentre la dimensionality reduction rappresenta il corrispettivo per i dati continui.

Apprendimento rinforzato

Nell'apprendimento rinforzato [29] il sistema nella fase di allenamento riceve un feedback delle sue azioni. Questo tipo di learning interagisce con l'ambiente per imparare la serie di combinazioni e di azioni che danno i migliori risultati memorizzando la storia delle precedenti esperienze. In base alle azioni considerate virtuose o dannose dall'ambiente, l'algoritmo prenderà le decisioni future (di predizione o di classificazione) in funzione dei feedback che ha ricevuto durante la fase di apprendimento.

2.2 Reti Neurali

In questo paragrafo verranno trattate le basi biologiche e matematiche su cui si fondano le Reti Neurali, e verranno mostrate le potenzialità delle due tipologie principali di reti che sono state utilizzate nel lavoro di tesi: le LSTM (Long Short Term Memory) e le CNN (Convolutional Neural Network).

2.2.1 Dal neurone biologico al neurone artificiale

Le Reti Neurali rappresentano un punto di contatto tra diverse discipline come la Neurologia, la Psicologia e l'Intelligenza Artificiale [30]. Si cerca infatti di emulare, attraverso dispositivi software o hardware, i comportamenti delle trasmissioni sinaptiche che avvengono nel cervello durante la fase di apprendimento e trasmissione dell'informazione. Le Reti Neurali sono una branca del Machine Learning e proprio per questo hanno un approccio di apprendimento di tipo automatico che, come si evince dal nome stesso, si ispira all'elemento basilare tramite cui il cervello elabora le informazioni: il Neurone.

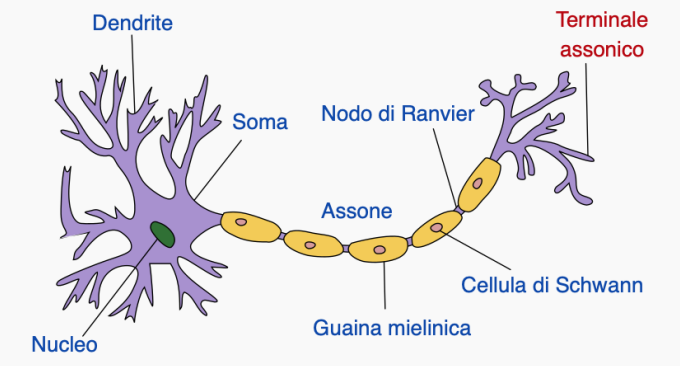


Figura 2.2: Struttura tipica di un neurone. Autore: Q.Jarosz

Un neurone biologico è caratterizzato da una semplice struttura costituita da 3 elementi essenziali:

- corpo cellulare, in cui è presente il nucleo che dirige tutte le attività del neurone;
- assone, singola fibra lunga che trasmette messaggi dal corpo cellulare ai dendriti di altri neuroni;
- dentriti, fibre che ricevono messaggi da altri neuroni e inoltrano tali messaggi al corpo cellulare.

I neuroni ricevono input elettrici dai dendriti, assorbono energia e la rilasciano tramite gli assoni, i canali di output, ad altri neuroni vicini, trasmettendo l'informazione sotto forma di impulso elettrico attraverso strutture specializzate chiamate sinapsi. Nel caso in cui il segnale favorisca o meno l'attivazione del neurone ricevente, la sinapsi può essere eccitatoria o inibitoria e solo dopo un certo periodo di tempo, chiamato tempo refrattario, il neurone può generare un ulteriore impulso, mostrando chiaramente la natura binaria di questo importante, quanto semplice, elemento biologico.

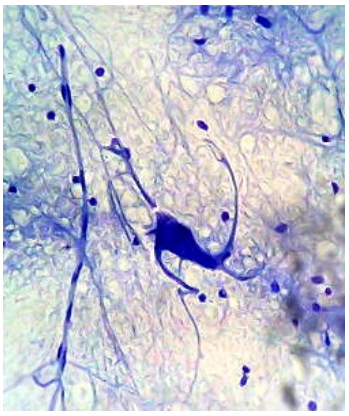


Figura 2.3: Neurone osservato con il microscopio ottico. Autore: F.Castets

Analogamente, una rete neurale artificiale è costituita dall'interconnessione di semplici unità di elaborazione, i cosiddetti neuroni artificiali, che hanno la propensione ad estrarre la conoscenza dai dati e a conservarla tramite dei pesi, chiamati pesi sinaptici. Le reti neurali artificiali si distinguono per due importanti caratteristiche:

- sono state concepite con l'abilità di poter approssimare qualsiasi funzione, sia essa lineare o non lineare. La linearità o la non linearità dipende unicamente dal processo di apprendimento e dalle funzioni di attivazione, che possono essere per l'appunto lineari o non lineari;
- aggiornamento dei pesi sinaptici durante l'allenamento. I campioni vengono somministrati alla rete e i pesi vengono modificati in modo da ridurre la distanza minima tra il target, l'obiettivo della rete, e l'uscita effettiva. In questo modo, la rete, a partire dagli esempi con cui si è allenata, costruisce una mappatura input-output del sistema.

2.2.2 Modello del neurone

Il neurone può essere descritto matematicamente da

$$y_k = \varphi\left(\sum_{j=0}^{p} x_j w_{kj+} + b_k\right) \tag{2.1}$$

oppure da

$$y_k = \varphi\left(\nu_k + b_k\right) \tag{2.2}$$

dove x_1,..., x_p sono i segnali di ingresso, b_k è il bias, w_{k1},..., w_{kp} sono i pesi sinaptici del neurone k, w_{k0} è il peso collegato con l'ingresso x_0=1, cioè la distorsione, $\varphi(\cdot)$ è la funzione di attivazione, ν_k è il potenziale d'azione e y_k è l'output del neurone k.

Il modello di un neurone è costituito da tre elementi di base:

1. pesi e bias: i collegamenti (sinapsi) vengono effettuati tramite i pesi, mentre i bias modificano l'effetto della funzione di attivazione. In particolare, il segnale di ingresso x_j, dove j sono le sinapsi connesse al neurone k, viene moltiplicato per il peso w_{kj}, in cui il primo pedice k si riferisce al neurone a cui si collegano

le sinapsi, mentre il secondo pedice si riferisce al neurone da cui provengono le sinapsi. Esiste anche un bias $b_k = w_{k0}$ (associato all'ingresso fisso x_0=1), il cui effetto aumenta (se positivo) o diminuisce (se negativo) l'ingresso netto della funzione di attivazione;

2. un operatore di somma, che effettua una somma pesata dei segnali di input per i pesi;

3. una funzione di attivazione per l'elaborazione dell'input del neurone e per la modulazione dell'output.

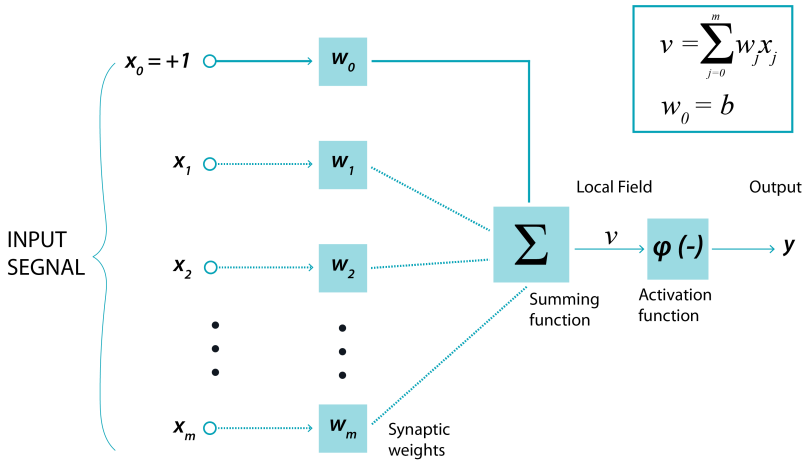


Figura 2.4: Esempio di Rete Neurale con Input, Output, pesi, bias, operatore di somma e funzione di attivazione.

2.2.3 Le funzioni di attivazione

La funzione di attivazione $\varphi(\nu)$ permette di stabilire l'uscita del neurone a cui è legata e contribuisce alla linearità e alla non-linearità del modello. Di seguito, saranno elencate le funzioni di attivazioni più utilizzate:

1. Funzione a soglia, descritta da

$$\varphi(\nu) = \begin{cases} 1, & \nu \geq 0 \\ 0, & \nu < 0 \end{cases} \tag{2.3}$$

2. Funzione lineare a tratti, in cui l'amplificazione è unitaria nella regione lineare. E' descritta da

$$\varphi(\nu) = \begin{cases} 1, & \text{se } \nu \geq +\frac{1}{2} \\ \nu, & \text{se } +\frac{1}{2} > \nu > -\frac{1}{2} \\ 0, & \text{se } \nu \leq -\frac{1}{2} \end{cases} \tag{2.4}$$

3. Funzione sigmoidale. Un esempio di funzione sigmoidale è la funzione logistica in cui è presente il parametro c, che influisce sulla pendenza. L'equazione ha valori di uscita compresi tra 0 e 1 ed è descritta da

$$\varphi(\nu) = \frac{1}{1 + e^{-c\nu}} \tag{2.5}$$

4. Funzione tangente iperbolica o tanh. E' una funzione simile a quella sigmoidale e si differisce da essa per essere antisimmetrica con valori compresi tra -1 e 1. L'equazione è descritta da

$$\varphi(\nu) = \frac{1 - e^{-2c\nu}}{1 + e^{-2c\nu}} \tag{2.6}$$

2.2.4 Architettura

L'architettura di una rete neurale artificiale standard è sempre influenzata dall'algoritmo di apprendimento utilizzato ed è composta da:

- unità di input, come vettore di numeri a lunghezza fissa definito dall'utente;
- unità nascoste (opzionale), che rappresentano calcoli intermedi appresi dalla rete tramite somme ponderate. Aggiungono non linearità;
- unità di output, come vettore di numeri a lunghezza fissa anche qui definito dall'utente.

Una rete neurale che ha l'architettura con due o più unità nascoste viene chiamata Deep e il meccanismo di apprendimento legato ad essa viene chiamato Deep Learning.

2.2.5 Convolutional Neural Networks

Ispirate dalla corteccia visiva dei mammiferi, le reti neurali convoluzionali [31], note anche come CNN (da Convolutional Neural Networks), sono reti neurali artificiali che rappresentano lo stato dell'arte per classificazione delle immagini ed il riconoscimento di schemi e strutture ricorrenti nei dati (*pattern recognition*). Negl'ultimi anni hanno subito uno sviluppo notevole [32], [33] e adesso sono comunemente impiegate per identificare oggetti, volti, individui, neoplasie e molti altri aspetti dell'elaborazione visiva.

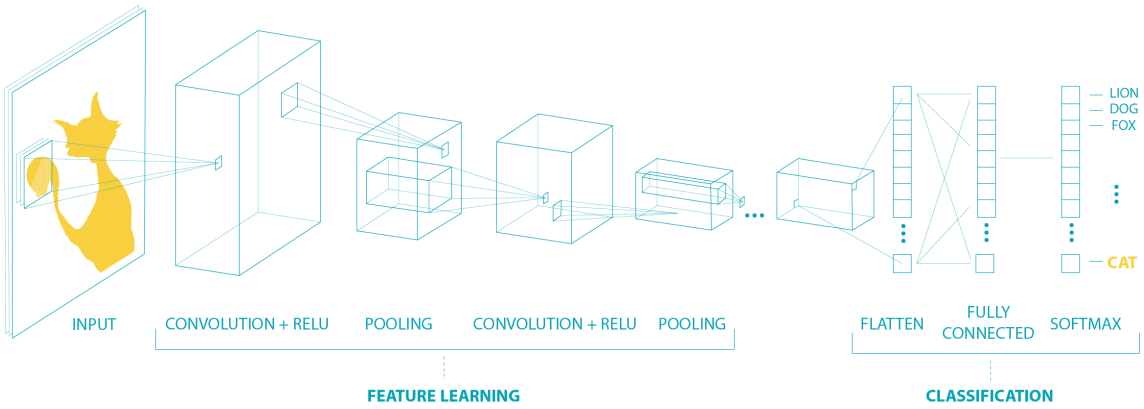


Figura 2.5: Esempio semplificato di Convolutional Neural Network.

Le reti neurali convoluzionali ricevono in input le immagini, intese come matrici di numeri, trasmettendo l'informazione attraverso uno o più strati (o *layer*), al fine di eseguire l'estrazione e la classificazione delle caratteristiche più importanti delle matrici di input. Il principale punto di forza delle CNN [34] è il basso numero di pesi richiesti per poter definire l'architettura della rete stessa.

Ciò che si ottiene limitando le connessioni tra i layer, è un campo ricettivo (*receptive field*) ridotto, per cui ogni neurone è collegato solo a pochi neuroni degli strati precedenti anziché a ciascuno. Inoltre, i pesi sono condivisi cosicché neuroni con la stessa caratteristica in diverse aree dell'immagine inneschino la stessa risposta. In altre parole, se la rete riesce ad imparare un particolare tipo di geometria in una sezione dell'immagine, sarà in grado di riconoscere tale geometria anche in altre parti dell'immagine.

Esistono diversi tipi di layer come :

- layer convoluzionale;
- layer non lineare;
- layer di pooling.

Layer Convoluzionale

Il layer Convoluzionale [35] prende il nome dall'operazione di convoluzione che viene effettuata tra la matrice di input (matrice 3D nel caso di immagini) e il filtro o kernel, che produce come risultato la *feature map*, o mappa delle caratteristiche, cioè la specifica feature che i nodi di una rete neurale tradizionale tentano di classificare. Tale layer può essere costituito da uno o più filtri a cascata.

Se si esegue una convoluzione, in pratica si fa scorrere il filtro sulla matrice di ingresso. In ogni posizione, viene eseguita una moltiplicazione puntuale tra la regione di input selezionata dal filtro (il *receptive field*) e il filtro stesso. Il risultato sarà posizionato sulla *feature map*.

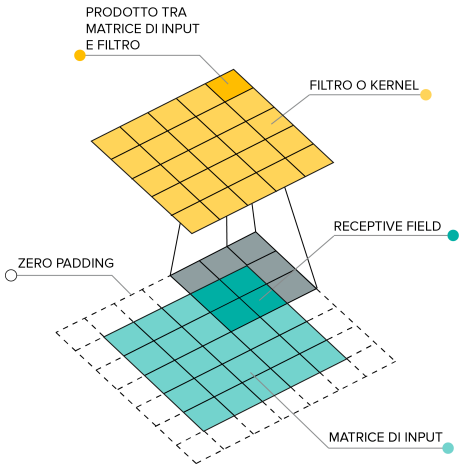


Figura 2.6: Esempio semplificato della convoluzione tra la matrice di input e il kernel

Per ogni prodotto puntuale tra il filtro e il *receptive field* della matrice di input il risultato è uno scalare, quindi è per questo motivo che per poter ottenere una matrice, bisogna far scorrere il filtro per l'intero input. Come è possibile osservare dalla figura 2.6, per poter mantenere la stessa dimensionalità della matrice di ingresso, è possibile effettuare l'operazione di *zero padding*, aggiungendo degli zeri ai bordi dell'immagine in base alle dimensioni del kernel.
Se si eseguono numerose convoluzioni sull'input, ogni operazione utilizzerà un filtro diverso. Ciò si traduce nell'ottenimento di diverse mappe delle caratteristiche sempre più astratte. Alla fine, mettendo insieme in cascata tutte le *feature map* si otterrà l'output finale del layer di convoluzione, con il livello di astrazione proporzionale alla profondità del layer stesso.

Layer Non Lineare

E' un layer di neuroni che applica funzioni di attivazioni di non linearità come:

- ReLU, dall'inglese Rectified Linear Unit, cioè unità lineare rettificata. E' una funzione di attivazione definita come la parte positiva del suo argomento (neurone). L'equazione è descritta da

$$f(z_j) = max(0, x_j) \tag{2.7}$$

- Softmax. Tale funzione comprime un vettore k-dimensionale in un altro vettore dalle stesse dimensioni ma con valori compresi tra 0 e 1, la cui somma è 1. L'equazione è descritta da

$$f(z_j) = \frac{e^{z_j}}{\sum_j e^{z_j}} \tag{2.8}$$

Layer di pooling

Il layer di pooling riduce la dimensione spaziale, al fine di diminuire la quantità di parametri della rete. Ciò si ottiene partizionando la matrice di input in un insieme di sottoregioni non sovrapposte e trovando il valore massimo dei pixel in ciascuna regione: si parla infatti di max-pooling [36].

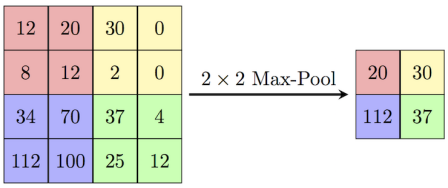


Figura 2.7: Esempio di riduzione della dimensione spaziale di una matrice tramite max-pooling

2.2.6 Long Short Term Memory

Le Long Short Term Memory [37], altrimenti note come LSTM, sono un particolare tipo di reti neurali ricorrenti, o RNN, cioè delle reti neurali cicliche adatte per le predizioni, poiché permettono all'informazione, acquisita tramite i pesi, di essere mantenuta nel tempo.

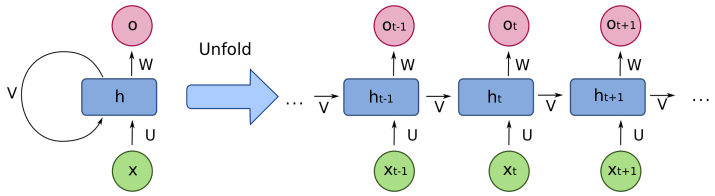


Figura 2.8: Una rete neurale ricorrente unfold, in cui si nota la ciclicità tra output ed input. Autore: F. Deloche, CC BY-SA 4.0

Rispetto alle reti neurali tradizionali, nelle reti ricorrenti i valori di uscita di uno strato di un livello superiore vengono utilizzati come ingresso ad uno strato di livello inferiore. Un esempio di RNN è mostrata in figura 2.9.

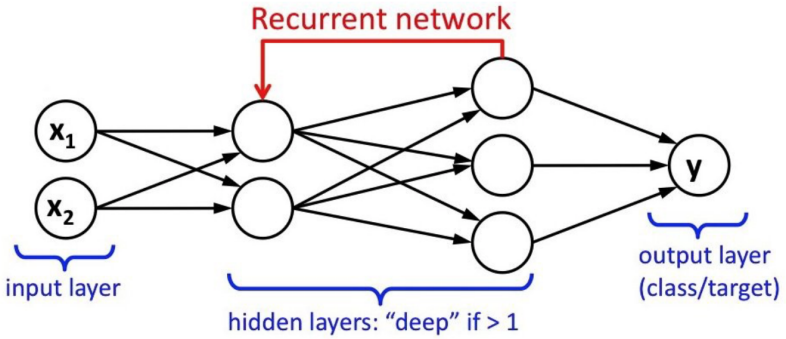


Figura 2.9: Tipica architetture di una rete neurale ricorrente. Autore: L. Araujo dos Santos

Il maggior limite di questa tipologia di rete, nel caso di algoritmo di apprendimento basato sul backpropagation [38], è tuttavia il " vanishing/exploding gradient" [39]: nella fase di apprendimento, i gradienti dell'errore vengono propagati indietro fino al layer iniziale passando attraverso le moltiplicazioni di matrice continue e dopo un certo numero di epoche, i pesi divergono all'infinito o convergono a zero. Nel caso del vanishing gradient, i pesi convergono a zero e quindi la rete, nel momento in cui effettuerà le predizioni, terrà in considerazione solo gli eventi più recenti, mentre nel caso di exploding gradient il valore dei pesi "esplode", tendendo all'infinito. Non si avrà dunque memoria passata e i gradienti saranno caratterizzati solo da valori indefiniti, i cosiddetti NaN.

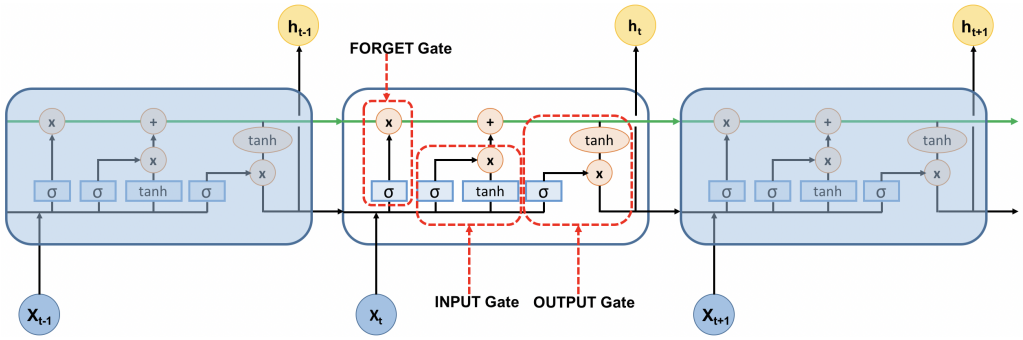


Figura 2.10: Schema di una rete LSTM con: INPUT, FORGET, OUTPUT gate, stato della cella; funzioni di attivazione sigmoide e tanh

Dal punto di vista strutturale (vedi 2.10), la rete LSTM standard consiste di 4 elementi fondamentali:

- *stato* della cella, è l'elemento che trasporta l'informazione che può essere modificata dai *gate*. In figura 2.10 è la linea verde orizzontale che passa all'interno della cella di memoria;

- *forget gate*, ponte in cui l'informazione passa attraverso la funzione sigmoide. I valori di uscita sono compresi tra 0 e 1. Se il valore è più vicino a 0 significa *dimentica l'informazione*, se è più vicino a 1 significa *mantieni l'informazione*;

- *input gate*, è l'elemento che aggiorna lo stato della cella. E' ancora presente la funzione sigmoide, ma in combinazione con la funzione tanh. La funzione tanh riduce i valori di ingresso tra -1 e 1. Quindi si moltiplicata l'output di tanh con l'output della funzione sigmoide, che deciderà quali informazioni è importante mantenere dall'output tanh;

- *output gate*, è l'elemento che decide come dovrebbe essere il prossimo stato nascosto, che ricorda le informazioni sugli input precedenti alle celle temporali successive. Innanzitutto, lo stato nascosto precedente e l'input corrente passano in una funzione sigmoide, il cui output, cioè lo stato di cella appena modificato, passa per la funzione tanh. Si moltiplica quindi tale output con l'output sigmoideo per decidere quali informazioni devono essere contenute nello stato nascosto. Il nuovo stato di cella e il nuovo stato nascosto vengono quindi riportati al passaggio temporale successivo.

A differenza delle RNN, le LSTM grazie all'utilizzo di "celle di memoria" e di funzioni gate, che sono presenti all'interno di esse e che possono essere considerati come dei neuroni, ricordano l'informazione per lunghi periodi, permettendo di risolvere di fatto il problema del "vanishing/exploding gradient".

2.2.7 Gli Encoder-Decoder

L'architettura nota come Encoder-Decoder (ENC-DEC), introdotta da Google nel 2014 in [40], è un tipo di architettura composta da due elementi chiave: l'*encoder* (o codificatore) e il *decoder* (o decodificatore). Entrambi questi sistemi sono collegati da uno spazio ridotto intermedio, comunemente noto come *bottleneck* (o collo di bottiglia). Analizzeremo questi tre elementi per capire le potenzialità di questa architettura e come si possono sfruttare per i fini della predizione di sequenze multi-step, conosciute come *sequence-to-sequence*, o *seq2seq* in breve.

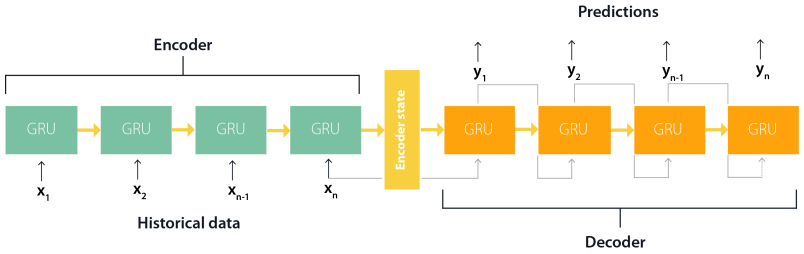


Figura 2.11: Esempio di architettura ENC-DEC con una serie temporale

Lo schema dell'ENC-DEC è presentato in figura 2.11. Il codificatore prende in input le sequenze, le analizza e le comprime in una rappresentazione interna a lunghezza fissa, in modo da riassumere in linguaggio codificato le caratteristiche più significative. A questo punto, il decodificatore interpreta la rappresentazione interna delle sequenze e la utilizza per generare la sequenza di output. In questo modo, si avranno due reti neurali distinte con distinti compiti:

- l'encoder si allenerà a comprendere e riassumere in maniera sempre più precisa le sequenze di input in uno spazio codificato intermedio;
- il decoder si concentrerà solo a generare sequenze di output sempre più accurate a partire dallo stesso stato codificato intermedio.

Una rete neurale standard, come quelle presentate nei paragrafi precedenti, affronta il duplice compito di imparare dai dati e effettuare un'inferenza, mentre con l'approccio ENC-DEC questi compiti vengono svolti ad hoc da due reti distinte e indipendenti tra loro.

Inoltre, sia l'encoder che il decoder sono reti neurali e perciò godono della capacità di generalizzazione in domini totalmente differenti. Si potrebbe facilmente sostituire la sequenza di numeri (tipo la sequenza di valori glicemici) con sequenze di parole o lettere, rendendo possibili traduzioni da una lingua ad un'altra, tipo dall'inglese all'italiano. Nel caso specifico della traduzione dalla lingua inglese a quella italiana, piuttosto che avere una rete neurale molto complessa in grado di capire l'inglese e parlare l'italiano, potremmo sfruttare due reti (encoder e decoder) in cui il codificatore capisce l'inglese ma non sa parlare l'italiano e il decodificatore

sa parlare l'italiano ma non capisce l'inglese. Lavorando insieme tramite lo spazio codificato intermedio, queste due reti specializzate hanno tutte le potenzialità per superare la singola rete complessa.

Nel capitolo 4 saranno implementate reti neurali LSTM e CNN a partire dallo stato dell'arte e, successivamente, saranno modificate con un approccio ENC-DEC per la predizione del livello glicemico per orizzonti di previsione di diversi minuti nel futuro.

Capitolo 3

Lo Stato dell'Arte per la predizione del glucosio con reti neurali

In questo capitolo, si ripercorreranno le tappe più importanti per la predizione del glucosio, fino ad arrivare ai migliori algoritmi di Machine Learning che attualmente sono stati già utilizzati per la predizione del glucosio con l'OhioT1DM dataset[18].

3.1 I primi approcci di Machine Learning con i modelli classici

L'interesse nella predizione del glucosio è ovviamente esploso nel momento in cui sono stati commercializzati i dispositivi CGM, ma anche prima dell'avvento di tali dispositivi si era manifestato un certo interesse in questo ambito sfruttando tra i 3 e i 10 campioni al giorno[41],[42] con una frequenza di campionamento dei dati dalle 3 alle 8 ore. L'approccio utilizzato è, quindi, inevitabilmente diverso da quello per utilizzato per i dati CGM. Quando si allena una rete neurale, si vuole ottenere un modello che effettui l'inferenza a un determinato orizzonte di previsione, cioè quanto lontano il modello riesce a prevedere il futuro valore della variabile. L'orizzonte di previsione è però legato alla frequenza di campionamento, che nel caso di misurazioni SMBG (Self-Monitoring Blood Glucose, automonitoraggio tramite il pungidito) risulta essere di qualche ora, mentre con i nuovi dati CGM, invece, varia da 1 minuto a 5-10-15 minuti.

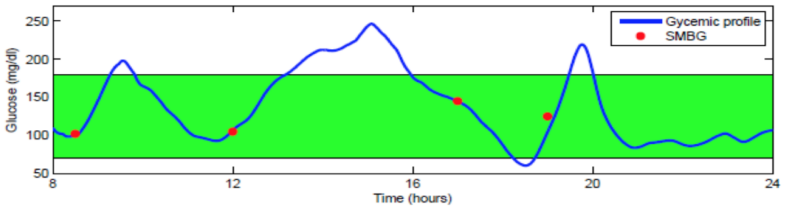


Figura 3.1: Comparazione tra l'andamento del livello di glucosio reale (linea blu) e le misurazioni ottenute tramite automonitoraggio con pungidito (punti rossi). La fascia verde rappresenta la fascia di valori considerati normali

Nel grafico 3.1, è possibile osservare l'esempio di un duplice andamento dei dati glicemici tramite un dispositivo CGM (linea blu) e quelli ottenuti tramite automonitoraggio(punti rossi). E' facile evidenziare come l'automonitoraggio non è stato in grado di rilevare i picchi iperglicimici post-prandiali e il calo glicemico pomeridiano. Quindi, dato che l'orizzonte di previsione deve essere almeno pari al periodo di campionamento e si riesce a ricavare meno informazioni sul profilo glicemico del paziente, le techiche basate su dati SMBG sono qualitatativamente e quantitativamente lontane dal livello attuale con i dispositivi CGM, anche se comunque hanno aperto la strada del Machine Learning nell'ambito della predizione del livello glicemico del sangue e del trattamento del diabete in generale. Con l'avvento dei dispositivi CGM, gli algoritmi più popolari, che sfruttano per la maggioranza solo le informazioni glicemiche, sono stati il modello autoregressivo (AR)[43] e autoregressivo a media mobile (ARMA)[44]. Inoltre, si è cercato di sfruttare le informazioni sui carboidrati ingeriti e sulla terapia insulinica con i modelli ARX e ARMAX[45], aggiungendo informazioni collaterali come agenti esogeni esterni da aggiungere al modello di riferimento basato sulle letture del glucosio. E' lecito aspettarsi che i modelli classici lineari (AR, ARMA, ARX, ARMAX etc..) non siano efficaci come i modelli neurali, per via delle capacità di quest'ultimi di apprendere ed utilizzare segnali eterogenei tra loro in modelli empirici non lineari. I prossimi paragrafi saranno utili per identificare quali sono state le tecniche più utilizzate nell'ultimo decennio per diversi tipi di dataset. In seguito, verranno analizzati articoli dedicati ad algoritmi di Machine Learning basati sull'Ohio T1DM dataset, lo stesso che verrà utilizzato per l'implementazione di reti neurali nel personale lavoro di Tesi.

3.2 Alcuni esempi di Reti Neurali per la predizione di glucosio

Saranno presentati in questa sezione alcuni articoli sulla predizione del glucosio con informazioni riguardanti: scopo, tipologia di rete neurale, dataset, canali di input, risultati e conclusioni.

3.2.1 Rete Neurale Artificiale

Nel 2010, uno dei primi lavori sulle reti neurali fu presentato da Perez Gandìa et al. [46]. La rete neurale artificiale proposta ha livelli nascosti a 10 e 5 neuroni, rispettivamente. La funzione di attivazione utilizzata per entrambi i layer è quella sigmoidale. La rete prende in input 20 minuti prima dell'attuale valore corrente per restituire in output la singola predizione glucosio nel tempo dell'orizzonte di previsione (PH) scelto.

Il modello è stato addestrato su due distinti set di dati composti unicamente da misurazioni glicemiche: uno costituito da 9 soggetti monitorati utilizzando il sensore CGM Guardian, con frequenza di campionamento di 5 minuti e l'altro formato da 6 soggetti monitorati con il sistema CGM di Navigator, con una frequenza di campionamento di 1 minuto. Per abbassare l'effetto del rumore presente nei dati campionati a 1 minuto, è stato deciso di utilizzare il filtro Kalmann.

I risultati sono suddivisi per dispositivo in termini di RMSE (mean ± Standard Deviation in mg/dL) e per orizzonti di previsione di 15, 30, 45 minuti:

	Dispositivo Guardian		
	RMSE (media ± SD mg/dl) a PH di		
	15 min	*30 min*	*45 min*
NN	9.74±2.71	17.45±5.44	25.08±8.73
ARM	9.26±2.97	20.27±7.27	30.30±11.89

Tabella 3.1: Risultati per dati CGM da dispositivo Guardian. NN sta per modello rete neurale mentre ARM sta per modello autoregressivo.

	Dispositivo FreeStyle Navigator		
	RMSE (media ± SD mg/dl) a PH di		
	15 min	*30 min*	*45 min*
NN	10.38±3.15	19.51±5.53	29.07±6.77
ARM	10.46±3.55	24.36±8.97	39.36±10.38

Tabella 3.2: Risultati per dati CGM da dispositivo FreeStyle Navigator. NN sta per modello rete neurale mentre ARM sta per modello autoregressivo.

Confrontando tale approccio con una tecnica precedentemente pubblicata dallo stesso gruppo di ricerca basata su un modello autoregressivo, si è giunto alla conclusione che tale modello neurale risulta essere una soluzione affidabile per la previsione delle concentrazioni di glucosio a partire dai soli dati CGM.

3.2.2 Le reti neurali di tipo jump e l'utilizzo di un modello fisiologico

In questo lavoro risalente al 2013, Zecchin et al.[47] utilizzano una rete neurale di tipo *jump* (o salto), cioè una rete neurale con input collegati non solo al primo layer nascosto ma anche al layer di output.

I canali di input della rete sono 4: due correlati al segnale CGM (serie temporale del glucosio e la sua derivata nel tempo tempo) e due relativi alle informazioni ottenute tramite un modello fisiologico di ingerimento dei carboidrati al tempo corrente e alla sua tendenza media negli ultimi 30 min. L'output è sempre un singolo neurone, che restituisce la predizione glucosio nel tempo dell'orizzonte di previsione (PH = 30 minuti).

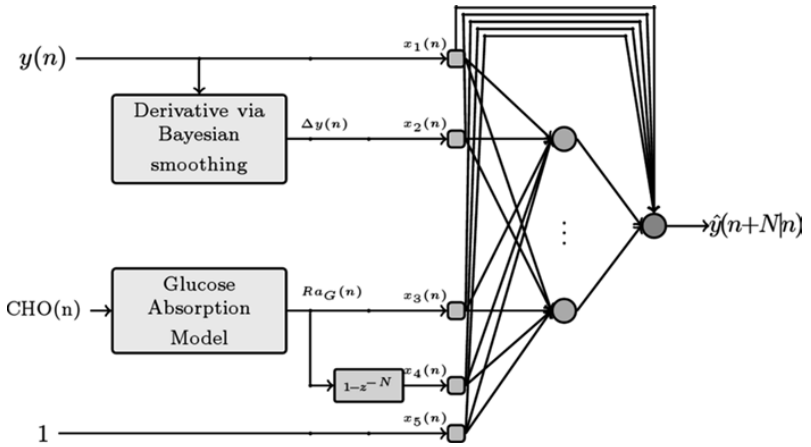


Figura 3.2: Immagine tratta da [47]. Viene visualizzato lo schema della rete Jump e i collegamenti dei neuroni di input sia con i neuroni appartenenti allo strato nascosto sia al neurone di ouput. E' possibile notare anche i 4 canali di ingresso alla rete, due relativi alla serie temporale di glucosio e due relativi al modello fisiologico di assorbimento del glucosio.

La concentrazione di glucosio è stata misurata dal sensore CGM Dexcom SEVEN PLUS® , che ha un periodo di campionamento di 5 minuti. La rete è stata sviluppata e testata su un dataset di 20 pazienti con diabete di tipo 1, monitorati per 2 o 3 giorni consecutivi in condizioni di vita quotidiana. Il dati utilizzati per allenare e testare sono stati normalizzati prima di essere introdotti nel modello neurale al fine di aumentarne la stabilità.

I risultati in tabella 3.3 sono presentati secondo 3 differenti metriche (RMSE, TG ed $ESOD_{norm}$) per un singolo orizzonte di previsione di 30 minuti. Il valore riportato rappresenta la media e la deviazione standard ottenuti testando il modello su 10 pazienti. Per ciò che concerne le metriche utilizzate, a parte l'RMSE, che è radice dell'errore quadratico medio, c'è la presenza del TG e dell'ESOD. In breve, più il TG è vicino al PH, migliore è la previsione, mentre l'$ESOD_{norm}$ quantifica approssimativamente il rischio di generare falsi avvisi di glicemie. Più vicino a 1, migliori sono le serie temporali predette.

	Media ± SD per PH = 30 minuti		
	RMSE	TG (minuti)	$ESOD_{norm}$
Rete Jump	16.6 ± 3.1	18.5 ± 3.4	9.6 ± 1.6

Tabella 3.3: Media ± deviazione standard provenienti da 10 soggetti di testing per un orizzonte di previsione di 30 minuti.

La complessità del modello fisiologico utilizzato unita alla semplicità della rete neurale proposta mostrano che le previsioni di concentrazione di glucosio utilizzando questo tipo di approccio sono possibili e paragonabili a quelle dello stato dell'arte, ma con un orizzonte di previsione limitato solamente a 30 minuti.

3.2.3 Le potenzialità di un approccio ibrido convoluzionale e ricorrente

L'articolo di Kezhi Li et al.[48], proposto nel 2018, risulta essere molto interessante per diversi aspetti. Innanzitutto, viene utilizzata una rete neurale ibrida (vedi 3.3), costituita in serie da una rete neurale convoluzionale CNN, una rete ricorrente LSTM e un layer di output, assemblate in modo che gli output della CNN siano gli input della rete LSTM. La predizione è poi affidata ad uno strato *fully connected*, cioè uno strato in cui i neuroni sono tutti completamente collegati ai neuroni del layer precedente.

Per l'allenamento e il testing del modello, vengono utilizzati due dataset di natura molto diversa: un dataset sintetico di 10 pazienti, ottenuto tramite il simulatore UVA/Padova T1D[49], e un dataset di 10 pazienti reali, ottenuto da uno studio clinico. I dati utilizzati in entrambi i dataset sono organizzati secondo 4 input: letture da dispositivi CGM con periodo di campionamento di 5 minuti, dati sui pasti che indicano il tempo e la quantità di carboidrati ingeriti, dati sull'insulina con ciascuna quantità di bolo e l'indice temporale associato ai 3 input precedenti.

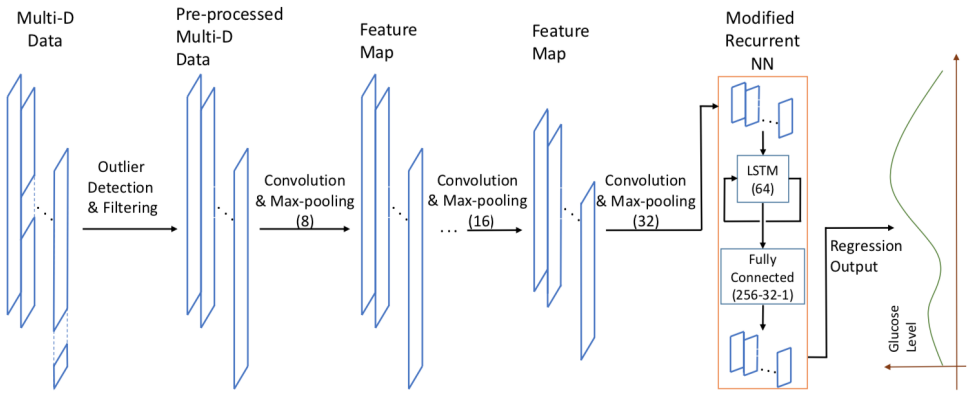


Figura 3.3: Immagine tratta da [48]. Viene visualizzato lo schema della rete ibrida CRNN. Si nota il flusso che parte dal filtraggio dati, passando per una rete CNN legata in serie ad una rete LSTM a sua volta collegata ad un fully connected per l'inferenza del glucosio

Per ciò che concerne il pre-processing, non viene effettuato alcun filtraggio per i dati sintetici mentre viene utilizzato un filtro con un kernel Gaussiano unidimensionale sulle letture del glucosio. L'analisi delle performance sono condotte sia utilizzando il dataset di dati *in-silico*, cioè sintetici, sia utilizzando i dati di pazienti reali, secondo le note metriche di RMSE e TG ad un orizzonte di previsione di 30 e 60 minuti.

I risultati di questo lavoro vengono valutati comparando l'approccio neurale con altri approcci della letteratura classici, tipo il SVR e l'ARX, con lo stesso metodo di pre-elaborazione dati e lo stesso dataset.

In seguito, vengono presentate le due tabelle. La prima valida per i dati sintetici, la seconda per i dati reali:

		Dataset con 10 pazienti virtuali	
		Media ± SD	
		RMSE (mg/dL)	TG(min)
PH = 30	CRNN	9.38 ± 0.71	29.0 ± 0.7
	SVR	12.48 ± 1.94	23.3 ± 1.6
	ARX	11.32 ± 1.34	20.5 ± 1.7
PH = 60	CRNN	18.87 ± 2.25	49.8 ± 2.9
	SVR	23.46 ± 3.33	32.6 ± 4.1
	ARX	25.73 ± 3.24	19.8 ± 2.7

Tabella 3.4: Risultati per dati da pazienti virtuali, comparando CRNN (Convolutional Recurrent Neural Network), SVR (Support Vector Regression) e ARX (modello autoregressivo con input esogeni) tramite alcune metriche come RMSE e TG

		Dataset con 10 pazienti reali	
		Media ± SD	
		RMSE (mg/dL)	TG(min)
PH = 30	CRNN	21.07 ± 2.35	19.3 ± 3.1
	SVR	22.00 ± 2.83	18.6 ± 2.8
	ARX	21.56 ± 2.53	12.0 ± 3.0
PH = 60	CRNN	33.27 ± 4.79	29.3 ± 9.4
	SVR	34.35 ± 4.55	28.4 ± 5.2
	ARX	36.97 ± 4.75	13.6 ± 3.7

Tabella 3.5: Risultati per dati da pazienti reali, comparando CRNN (Convolutional Recurrent Neural Network), SVR (Support Vector Regression) e ARX (modello autoregressivo con input esogeni) tramite alcune metriche come RMSE e TG

Le tabelle dimostrano ancora una volta che un approccio neurale, in questo caso ibrido, riesca a raggiungere delle performance sulle predizioni non indifferenti sia nel breve che nel lungo periodo. Utilizzando dati simulati inoltre si dimostra come l'integrità, anche se forzata, dei campioni simulati dati in input al modello sia un ingrediente fondamentale per la buona riuscita di un approccio di Machine Learning. In tutti i casi infatti sia per PH=30 che per PH=60 minuti, la CRNN offre dei buoni risultati sia in termini di RMSE che di TG. Sarebbe stato interessante provare la stessa efficacia del modello con più feature di ingresso, per dimostrare se aumenta la variabilità dei risultati, aumentando gli input di ingresso alla rete sia per il dataset reale che per quello sintetico. Infine, anche se i dati simulati sono generati da un simulatore approvato dalla FDA, sarebbe stato utile mostrare i risultati basati ad esempio su un approccio caratterizzato da:

- Allenamento su dati reali e Validazione su dati sintetici
- Allenamento su dati sintetici e Validazione su dati reali

3.2.4 L'importanza del filtraggio dei dati in un dataset ampio

Aliberti et al. - Maggio 2019 In questo lavoro, Aliberti et al. [50] studiano modelli di predizione addestrati sui segnali di glucosio provenienti da un dataset molto ampio ed eterogeneo di 451 pazienti [51]. Le soluzioni neurali trattate sono state le reti autoregressive non lineari (NAR) e su reti LSTM (Long Short Term Memory), confrontate sperimentalmente con altri tre approcci della letteratura, rispettivamente basati su reti neurali Feed-Forward (FNN), modelli autoregressivi (AR) e reti neuronali ricorrenti (RNN). La NAR presenta un layer di input di 30 elementi, un layer nascosto di 30 elementi e un layer di output, che rappresenta la predizione, di 1 elemento. La rete neurale ricorrente LSTM è costituita da 1 layer con 30 celle di memoria seguita da un layer di output, adibito alla predizione. Il canale di input in questo caso è dato solamente da letture di glucosio da dispositivi CGM. Interessante come la generalizzazione conferita da un dataset ampio ed eterogeneo unita ad un filtraggio ottimale, basato su *smoothing* tramite la regolarizzazione di Tikhonov, riesca a raggiungere delle prestazioni molto competitive per lo stato dell'arte. Gli orizzonti di previsione considerati comprendono sia il lungo che il breve termine e sono: 30, 45, 60 e 90 minuti nel futuro. Le fasi più importanti di tale studio sono presentate in Figura 3.4

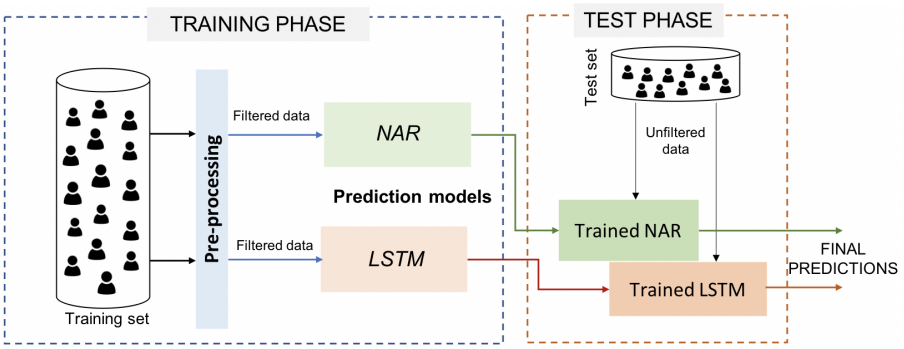


Figura 3.4: Immagine tratta da [50]. Sono presenti in questa immagine le fasi di training e di testing valide sia per la rete NAR che la rete LSTM.

Le metriche utilizzate per validare il modello (RMSE, R^2, Time Lag, MAD, FIT) sono inoltre corredate da informazioni di rilevanza come le Clark Error Grid (ESA), che forniscono un'interpretazione di importante validità clinica tra il segnale vero di testing e il segnale predetto di glucosio dal modello. Per uniformità di valutazione rispetto agli altri articoli di questo capitolo, verranno solamente riportati i valori di RMSE per i diversi orizzonte di previsione, riferendosi al dataset filtrato per la costruzione di modelli neurali NAR e LSTM.

	Training set filtrato			
	RMSE mg/dL per PH			
	30 min	*45 min*	*60 min*	*90 min*
NAR	18.2	25.31	33.12	47.64
LSTM	5.93	7.18	13.21	28.57

Tabella 3.6: Valori di RMSE riportati per le reti NAR e LSTM ai diversi orizzonti di previsione

Se la NAR ha ottenuto delle buone previsioni solo a breve termine (PH = 30 min), LSTM ha ottenuto prestazioni estremamente buone sia per l'inferenza del livello di glucosio a breve che a lungo termine (60 minuti e oltre), superando tutti gli altri metodi comparati in termini di performance e di validità clinica (ESA). Dato l'approccio vincente nel filtraggio dei dati e l'efficacia clinica di tale lavoro in un tema così delicato come la predizione del livello di glucosio del sangue, è stato deciso di utilizzare nel lavoro di implementazione eseguito nel capitolo 4 sia il filtraggio dei dati per la rimozione del rumore tramite Tikhonov, sia le stesse metriche di valutazione delle predizioni ottenute dal modello.

3.3 Reti neurali con OhioT1DM dataset

Questa sessione è dedicata all'elenco di alcuni articoli significativi che sono stati pubblicati utilizzando l'OhioT1DM dataset. Alcuni di questi lavori sono stati presentati IJCAI-ECAI 2018 [52] per una BGLPC(Blood Glucose Level Prediction Challenge), alla 27ª Conferenza congiunta internazionale sull'Intelligenza Artificiale. Sarà dimostrato come le reti ricorrenti LSTM e le reti CNN, in molte delle sue varianti, sono le più utilizzate per la predizione del glucosio nel sangue. Per ciò che concerne invece l'utilizzo delle feature presenti nel dataset, è stato dimostrato che un numero elevato di feature generano una profonda variabilità nelle predizioni, anziché migliorarle.

3.3.1 Un approccio standard con l'Ohio T1DM dataset

La rete neurale creata da Martinsson et al.[53] è basata sull'utilizzo di un'architettura neurale composta da un singolo layer di 128 celle di LSTM. I dati glicemici dei pazienti, ottenuti dal datset OhioT1DM, le cui caratteristiche principali saranno presentate nel Capitolo 4, sono stati suddivisi in 80% per l'allenamento e 20% per la fase di test e non hanno subito alcun tipo di pre-elaborazione. Gli orizzonti di previsione sono stati selezionai su 30 e 60 minuti, utilizzando esclusivamente i valori di glicemia nel sangue, a fronte delle numerose feature che vengono messe a disposizione all'interno del dataset. I risultati ottenuti da Martinsson et al. sono in linea con i risultati ottenuti da altri suoi colleghi, mostrando come l'utilizzo dell'LSTM sia comunque garanzia di predizioni accurate. Il lavoro svolto risulta essere abbastanza standard e computazionalmente economico e rappresenta sicuramente una base di partenza per un'analisi di stampo neurale.

	Ohio T1DM dataset 6 pazienti	
	RMSE ± SD mg/dL per PH	
	30 min	*60 min*
LSTM	20.1 ± 2.5	33.2 ± 3.1

Tabella 3.7: I risultati ottenuti da Martinsson et al sono presentati secondo valore medio e deviazione standard su 6 pazienti dell'OhioT1DM dataset

3.3.2 Le reti WaveNet

Il lavoro di Taiyu Zhu et al [54] si basa su un modello neurale già presente allo stato dell'arte chiamato WaveNet [55], un particolare tipo rete convoluzionale utilizzata

nell'elaborazione del segnale acustico, in particolare nella sintesi vocale multi-lingua. I tratti distintivi di questa rete neurale sono le *causal dilated convolutions*.

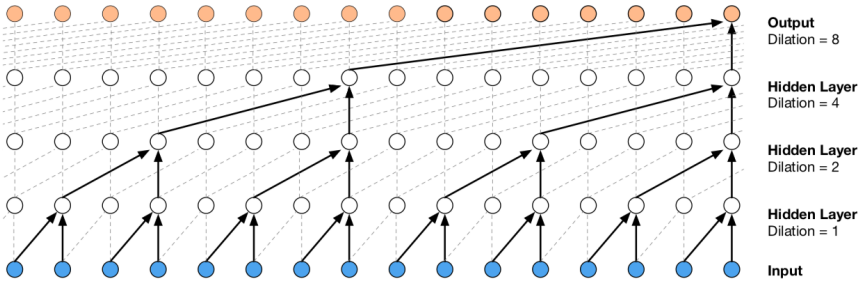


Figura 3.5: Immagine tratta da [55]. Si notano i campi recettivi dei neuroni che crescono esponenzialmente con l'aumentare della profondità della rete, cioè al crescere degli hidden layer, o strati nascosti.

In pratica, dato un neurone in una specifica posizione, quel neurone dipenderà solo da alcuni neuroni dello strato precedente, mentre il suo campo ricettivo cresce esponenzialmente con la profondità, che è importante per modellare le dipendenze temporali a lungo raggio. Per tale conformazione neurale, il lavoro svolto in questo articolo trasforma il problema di regressione del livello di glucosio in un problema di classificazione. Si sfruttano 4 canali di input dal dataset OhioT1DM: livelli di glucosio, eventi di insulina, quantità di carboidrati assunti e indici temporali. Per ciò che concerne la preparazione dei dati vengono sfruttate numerose tecniche come l'interpolazione lineare dei dati mancanti, la combinazione di dati appartenenti a pazienti diversi nella fase di allenamento e un filtro mediano per la rimozione del rumore presente dei dati CGM. Lo stile alternativo utilizzato per predire il valore di glucosio in 30 minuti nel futuro basato sulla classificazione anziché sulla regressione è sicuramente un aspetto considerevole di questo lavoro. Per prima cosa si calcola la variazione di glucosio tra il valore corrente e il valore a 30 minuti nel futuro, che corrispondono a 6 step avanti per via del campionamento a 5 minuti. Successivamente, si quantizza tale valore in 256 classi (o categorie) come target con una differenza di 1 mg / dL tra ciascuna classe, basandosi sul fatto dimostrato che il glucosio non subisce variazioni per più di ± 128 mg/dL in 30 minuti.

I risultati sono presentati in base alla radice dell'errore quadratico medio (RMSE) nella tabella 3.8 sottostante con il valore medio dei sei pazienti seguito da deviazione standard.

	Ohio T1DM dataset 6 pazienti
	RMSE $\pm$ SD mg/dL
WaveNet*	21.7267 $\pm$ 2.5237

Tabella 3.8: Vengono presentati i risultati WaveNet modificata da Taiyu Zu et al. I valori sono calcolati per un orizzonte di previsione di 30 minuti.*

Nonostante la creatività e l'ingegno con cui sono stati ottenuti i risultati (viene sfruttata una rete utilizzata allo stato dell'arte per la sintesi vocale), i valori di RMSE sono paragonabili ad altre tecniche, anche più semplici e più utilizzate nella predizione del glucosio. Da sottolineare che l'utilizzo delle WaveNet e delle

sue convoluzioni casuali-dilatate e la trasformazione del problema da regressione a classificazione, potrebbero essere elementi interessanti per la predizione del livello di glucosio nel sangue a lungo termine.

3.3.3 Comparazione tra i più noti modelli di Machine Learning

L'ultimo lavoro che sarà presentato in questo capitolo di stato dell'arte è quello a più ampio raggio sia nella comparazione di tecniche di Machine Learning sulla predizione del glucosio, sia sul diverso metodo per ottenere l'inferenza. In questo articolo Jinyu Xie et al [56], vengono implementati per fini comparativi i seguenti algoritmi di Machine Learning: autoregressione con modello di ingressi esogeni (ARX), regressione di Huber, regressione Ridge, regressione Elastic Net, regressione Lasso, Support Vector Regression con kernel lineare, Support Vector con kernel a base radiale, Random Forest, Gradient Boosting Trees, una rete neurale ricorrente LSTM e una rete di convoluzione temporale (TCN, Temporal Convolutional Network) [57], cioè una rete che sfrutta le dilatazioni temporali della WaveNet, già viste nel precedente articolo, ma con un approccio più complesso e *deeper*, cioè con molti più layer nascosti. Per ciò che riguarda il metodo per effettuare la predizione invece sono state elaborate due strategie principali per ottenere la previsione: metodo ricorsivo e metodo diretto. Per applicare il metodo ricorsivo, si addestra il modello a prevedere un solo step avanti, e si riutilizza il valore trovato per effettuare la previsione al passaggio successivo. Il metodo diretto considera il valore della y(t + k) a k step avanti, utilizzando solo i valori della funzione di input fino al tempo corrente t.

E' stato effettuato un intenso ed elaborato pre-processing sui dati provenienti dal dataset OhioT1DM che può essere riassunto in due fasi distinte: una fase di sincronizzazione e ricampionamento dei dati per l'allineamento ad una stessa linea temporale per paziente ed una fase di *imputation*, cioè quel processo statistico in cui si sostituiscono i dati mancanti nella serie temporale con dei valori sintetici. I canali di input utilizzati per tutti gli algoritmi sono 4: lettura di glucosio, quantità di insulina totale erogata, quantità di carboidrati durante i pasti e la frequenza cardiaca misurata.

Le reti neurali considerate hanno le seguenti caratteristiche architetturali:

- LSTM: 3 livelli nascosti con 50 celle per livello.

- TCN: vengono considerati 2 e 10 blocchi in serie di TCN, tutti aventi un fattore di dilatazione=2 e dimensione del kernel=4

	Ohio T1DM dataset 6 pazienti
	RMSE ± SD mg/dL
Regressione Lineare	19.62 ± 2.89
Elastic Net	20.15 ± 2.3
GradientBoostingTrees	20.73 ± 2.06
Huber	19.93 ± 3.38
Lasso	20.22 ± 2.27
Random Forest	21.05 ± 2.29
Ridge	19.62 ± 2.89
SVR-Kernel lineare	19.72 ± 3.31
SVR-Kernel radiale	19.53 ± 3.27
LSTM	20.72 ± 2.85
TCN-2 blocchi	21.32 ± 2.53
TCN-10 blocchi	21.20 ± 2.84

Tabella 3.9: Vengono presentati i risultati della comparazione di diverse tecniche di Machine Learning. I valori sono calcolati per un orizzonte di previsione di 30 minuti tramite metodo diretto.

I risultati mostrano una certa controtendenza rispetto a quello che ci si aspettava. Infatti l'algoritmo più performante, riportato sempre come valor medio dell'RMSE con deviazione standard, è proprio la regressione lineare, la più semplice tra le regressioni in quanto sfrutta solamente le relazioni lineari presenti all'interno della serie storica analizzata. E' stato riportato questo articolo, più che per i risultati empirici, per la rimarcata importanza della gestione dei dati appartenenti al dataset. Dalla Tabella 3.9, che mostra solamente i risultati delle performance col il metodo diretto (valore medio ± deviazione standard tra i 6 pazienti), si nota infatti un certo equilibrio tra i diversi algoritmi utilizzati, il che porta a sottolineare i due concetti cardine dell'affascinante mondo del Machine Learning:

1. nessun singolo algoritmo di apprendimento automatico è universalmente il migliore algoritmo. Bisogna provarlo. Concetto conosciuto come *No Free Lunch Theorem*

2. non esiste modello che funziona bene se i dati non sono "impacchettati" e somministrati alla rete nel modo appropriato. Concetto che stabilisce che le performance dipendono per quasi la totalità dal grado di qualità del dataset utilizzato.

Capitolo 4

Dataset, AGP e Reti Neurali

Il capitolo 4 rappresenta il cuore implementativo di tutto il lavoro di Tesi. Saranno mostrati i passi più importanti, i ragionamenti e le supposizioni che hanno definito i seguenti elementi:

- Dataset, dall'estrazione delle feature fino alla costruzione del dataset di partenza completo;
- Software open-source *home made* in grado di emulare le prestazioni di un AGP (Ambulatory Glucose Profile) commerciale con valenza di supporto clinico;
- Reti Neurali che sono state scelte come oggetto di studio, le cui prestazioni saranno riportate e commentate nel capitolo 5.

Il linguaggio di programmazione utilizzato per lo sviluppo e l'implementazione dei codici è Python, un linguaggio ad oggetti ad alto livello, molto usato nell'ambito del Machine Learning.

4.1 Ohio T1DM dataset

Il set di dati noto come OhioT1DM dataset è il risultato di un gentlemen's agreement siglato tra il Politecnico di Torino e l'Università dell'Ohio per l'utilizzo di tale dataset solo per scopo di ricerca. Il dataset è sviluppato da uno studio, mostrato nell'articolo [18], creato appositamente per l'evento IJCAI-ECAI 2018[52], cioè una sfida tra ricercatori per sviluppare algoritmi di previsione del livello di glucosio nel sangue basati sul Machine Learning. Il set di dati include dati raccolti da 6 pazienti anonimi con ID 559, 563, 570, 575, 588 e 591. Per ogni paziente, sono stati raccolti i seguenti dati in un unico file XML(vedi esempio in figura 4.1):

- livello di glucosio nel sangue da dispositivo CGM;
- livello di glucosio nel sangue da rilevazione manuale tramite pungi-dito, presentata come *finger stick*
- dosi di insulina basale, presentati come *basal* e *temp basal*, dove basal indica il periodo di infusione continua di insulina basale, mentre temp basal indica il periodo in cui tale infusione è sospesa

- dosi di insulina da bolo, presentati come *bolus*, ma in realtà presente nelle forme di bolo singolo, bolo *normal* e *double wave*.
- orari di pasti autoreportati con stime di carboidrati ingeriti
- orari di sonno, lavoro ed esercizio fisico
- eventi ipoglicemici
- eventi febbrili
- eventi stressanti
- frequenza cardiaca, presentata come *heart rate*
- risposta galvanica cutanea, presentata come *GSR, Galvanic Skin Response*
- temperatura cutanea in Fahrenheit
- temperatura dell'aria in Fahrenheit
- conteggio dei passi

4.1.1 Data Inspection

A parte i dati auto-riportati come sonno, lavoro e pasti, il resto dei dati hanno una frequenza di campionamento di 5 minuti e sono stati ottenuti dai dispositivi Medtronic Enlite®, per il monitoraggio del livello glicemico e insulinico, e da una banda fitness Basis Peak, che ha fornito dati fisiologici collaterali. Le informazioni sui pasti sono ottenute dalla pompa Bolus Wizard (BWZ), che viene utilizzata dai pazienti per calcolare il bolo da somministrare. I dati coprono un periodo di 8 settimane circa e sono stati rilasciati già divisi in training set e test set. I 6 pazienti, 2 maschi e 4 femmine, hanno tutti un'età compresa tra i 40 e i 60 anni.

Alla completezza ed eterogeneità dei dati forniti, corrispondono però alcuni aspetti negativi che obbligano ad effettuare un massiccio pre-processing:

1. esiste un altissimo grado di asincronia tra i dati, cioè non si trovano tutti sulla stessa linea temporale. Esempio: rilevamento valore di glucosio di 243 mg/dL alle 00:01 del 14 gennaio e rilevamento valore di frequenza cardiaca di 98 battiti/minuto alle 00:04 del 14 gennaio.

2. in tutte le feature, esistono dei buchi temporali sia di pochi minuti che di qualche ora, che creano problemi sia in fase di allenamento che di validazione.

I buchi temporali (gap) sono delle mancate letture da parte del dispositivo CGM e sono causati essenzialmente da:

- spegnimento del dispositivo per cambio batterie
- spegnimento del dispositivo per la calibrazione periodica
- possibili irritazioni cutanee o spostamenti volontari del paziente che portano ad un parziale distacco del sensore

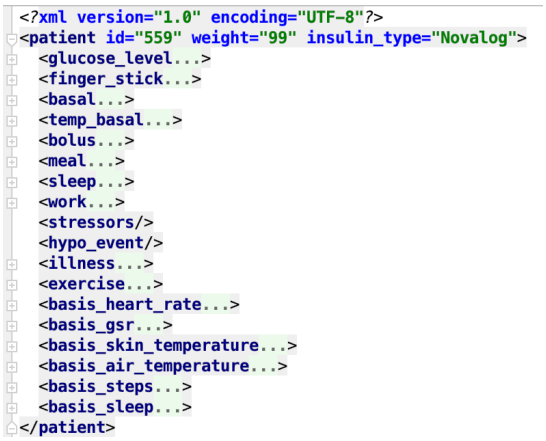


Figura 4.1: Esempio di file XML. Si nota l'ID del paziente, il peso e la tipologia di insulina nella seconda riga. In seguito, tutti i campi archiviati disposti per categoria.

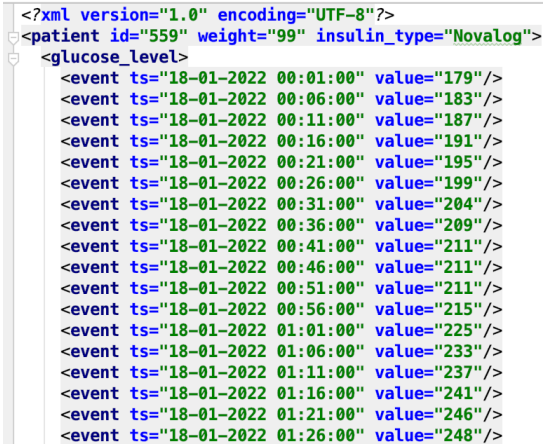


Figura 4.2: Particolare di file XML. Si nota che il campo glucosio selezionato ha due attributi: l'indice temporale e il valore corrente di glucosio

La Tabella 4.1 riporta il numero e le percentuali di dati mancanti per paziente. I dati mostrano come circa il 9% delle delle serie temporali glicemiche post-sincronizzazione per paziente è caratterizzato da gap temporali.

	559	*563*	*570*	*575*	*588*	*591*	**MEDIA**	**SD**
# gap	1230	969	627	1180	466	1902	**1062**	510
% gap	10,42	7,48	5,44	9,31	3,68	15,00	**8,56**	4,01

Tabella 4.1: Numeri e percentuali di gap temporali nei file di training, valutati dopo la creazione dei DataFrame Pandas, ottenuti tramite la funzione implementata nel paragrafo 4.1.2

Le percentuali si riferiscono ai soli file di training. Si noti come il paziente 591 sia il paziente con il numero più alto di gap temporali, pari a 1902 campioni, quasi il doppio della media tra i pazienti, che è di 1000 campioni circa, corrspondenti quasi a 4 giorni di dati glicemici.

4.1.2 Data Preparation

L'obiettivo della fase di preparazione dei dati per paziente è quello di creare una griglia, o Data-Frame, di tante serie temporali quante sono le feature del dataset e sincronizzarle secondo un unico filo conduttore temporale, in modo che sia possibile osservare come si comportano tutte le feature selezionate allo stesso timestamp, cioè allo stesso momento. A tal fine, sono state seguite 5 fasi:

1. Selezione delle feature
2. Ingegnerizzazione e Sincronizzazione dei dati
3. Analisi dei *missing value*, cioè dei valori mancanti nei buchi temporali
4. Analisi, confronto e scelta di tecniche tra interpolazione e *imputation*, che sarà tradotta con imputazione, anche se il significato è totalmente differente dal consueto significato in italiano

5. Controllo qualità finale

E' importante chiarire come le analisi non siano conseguenziali, ma di supporto per la fase più importante di ingegnerizzazione e sincronizzazione dei dati. Tutte le fasi saranno adesso analizzate nelle successive sottosezioni.

FASE 1 - Selezione delle feature

Questa fase è caratterizzata dalla cernita iniziale effettuata sulle feature candidate a far parte della fase successiva di ingegnerizzazione e sincronizzazione dei dati. Da ogni file XML(ne esistono due per paziente, uno di training e uno di testing), sono state inizialmente scartate tutte quelle feature che mediamente apportavano poco "peso", in quanto non presenti in tutti i pazienti e se presenti, si presentavano molto raramente.

Ecco le feature che sono state concettualmente scartate: eventi ipoglicemici, eventi febbrili, eventi stressanti, orari di lavoro e di esercizio fisico, finger stick.
In seguito vengono riportate invece, tutte le feature, intese ancora come campi dei file XML, che sono state "ammesse" alla fase successiva perché potenzialmente ritenute importanti per sviluppi futuri: letture CGM di glucosio, basal, temp basal, boli, frequenza cardiaca, GSR (risposta galvanica della pelle), temperatura della pelle e temperatura dell'aria, conteggio dei passi, conteggio dei carboidrati nei pasti e qualità/durata del sonno.

FASE 2 - Ingegnerizzazione e sincronizzazione dei dati

Questa è sicuramente la fase più corposa delle 5 elencate.
La figura 4.5 mostra una panoramica che porta dal semplice file XML alla griglia con tutte le feature sincronizzate in un DataFrame Pandas, una struttura di dati bidimensionale, dove i dati sono allineati in modo tabellare in righe e colonne.

L'uso della libreria Pandas è quasi obbligatorio per tutti coloro che si cimentano nella costruzione di dataset, poiché offre una serie di vantaggi e librerie molto comode se si parla di manipolazione e generazione di dati/serie temporali/data-frame in linguaggio Python.

Si mostreranno adesso nel dettaglio i passi più importanti di tale processo per ogni singolo paziente:

- **Estrazione dai file XML.**
  A partire dal file XML, si sfruttano funzioni tali da essere in grado di: leggere il file XML, scegliere il campo desiderato (mettendolo come input alla funzione), estrarre i dati temporali e i corrispettivi valori correnti e dare in output la serie temporale di Pandas della feature di interesse. Le feature ottenute sono di due tipologie: continue o semi-continue.

```
1 import xml.etree.ElementTree as ET
2
3 def load_ohio_series(xml_path, variate_name, attribute,
    time_attribue="ts"):
4     tree = ET.parse(xml_path)
5     root = tree.getroot()
6     for child in root:
7         if child.tag == variate_name:
8             dates = []
```

```
            values = []
            for event in child:
                if(not type(time_attribue)==type((1,))):
                    time_attribue=(time_attribue,)

                for (i,t) in enumerate(time_attribue):
                    ts = event.attrib[t]
                    date = pd.to_datetime(ts, format='%d-%m-%Y %H:%M:%S')
                    date = date.replace(second=0)
                    value = float(event.attrib[attribute])
                    if(i==len(time_attribue)-1 and len(time_attribue)>1):
                        #value= float('nan')
                        value = 0
                    if(len(dates)>0 and dates[-1]>=date):
                        del dates[-1]
                        del values[-1]
                    dates.append(date)
                    values.append(value)
            index = pd.DatetimeIndex(dates)
            series = pd.Series(values, index=index)
            return series
```

Listing 4.1: Funzione che estrae le informazioni dal file XML e le converte in Series di Pandas

Le feature denominate come continue sono le feature che hanno un rilevamento costante, a meno di gap temporali, da parte di dispositivi associati o al dispositivo CMG o alla fitness band con una propria frequenza di campionamento, mentre le feature semi continue, sono caratterizzate da eventi limitati ad un circoscritto periodo di tempo (vedi la feature sleep, cioè il sonno) o da eventi che si manifestano un numero limitato di volte durante l'arco della giornata (vedi Meal, Bolus, Steps cioè Cibo, Boli e Passi). Un'altra caratteristica delle variabili non continue è quella di avere degli zeri quando la variabile non si manifesta e il valore vero quando si manifesta. Quindi per esempio Meal(o Pasti) avrà nell'arco della giornata tutti zeri, tranne i valori corrispondenti ai carboidrati all'ora di inizio pasto autosegnalata dal paziente.

- **Passaggio nell'ambiente di lavoro del Pandas DataFrame.**
  Per lavorare con i DataFrame si associa ad ogni serie la classe DataFrame, ottenendo la possibilità di sfruttare un'ampia gamma di funzioni utili per la preparazione dei dati.

```
import pandas as pd

glucose_series_DF = pd.DataFrame(glucose_series)
```

Listing 4.2: Conversione in DataFrame pandas delle serie temporale del glucosio

In particolare **resample**, è immediatamente utilizzata per ricampionare tutte le serie temporali ad intervalli di 5 minuti, in modo da semplificare il lavoro successivo di interpolazione.

- **Interpolazione delle variabili continue e filling del variabili non continue.**

I segnali delle caselle in verde dell'immagine 4.5, cioè i segnali delle variabili continue, sono stati ricampionati ad intervalli di 5 minuti tramite la funzione resample con metodo pchip (discusso nella FASE 4 di questo capitolo), utilizzata in serie dopo **ffill** con limite = 1, una funzione che agisce secondo il *filling forward*, cioè se il valore manca in un punto ad un particolare timestamp, verrà interpolato con il valore più vicino prima di tale punto. Oltre a tale regola, lo stesso valore può essere utilizzato al massimo una sola volta nel processo(da qui il parametro del limite = 1), mentre i valori seguenti saranno etichettati semplicemente come mancanti e saranno soggetti all'interpolazione.

```
resampled_GS = glucose_series_DF.resample('300S',convention='
    start').ffill(limit=1).interpolate("pchip")
```

Listing 4.3: Ricampionamento e interpolazione della serie di glucosio

Diverso è il caso delle variabili non continue.

In esse, la stragrande maggioranza di dati mancanti sono i dati virtualmente nulli, cioè i dati in cui non esiste la misurazione, ma che, per poterla inserire nella stessa griglia temporale delle variabili continue, è stata posta uguale a 0.

```
resampled_bolus = bolus_series_DF.resample('300s', convention='
    start').asfreq().fillna(0)
```

Listing 4.4: Esempio di ricampionamento del bolo

E' un approccio *naive*, che sarà ulteriormente approfondito nella sezione 4.1.4 riguardanti gli *sparse data* .

- **Ciclo for effettuato per ogni giorno corrispondente a 288 timesteps**
  In tale ciclo avviene l'effettiva costruzione del dataset con le feature selezionate. E' un lungo processo dove si ripetono i seguenti punti:

  1. Selezione dei valori da ogni serie per il giorno selezionato

```
import pandas as pd

x_temp = resampled_GS[pd.Timestamp(timestamp_day):pd.
    Timestamp(timestamp_day_after)]
```

Listing 4.5: Esempio di selezione della giornata del livello di glucosio

  2. Aggiornamento delle feature basal e temp basal in una singola feature

```
def make_values_null(basal, temp_basal, time):
    """ basal gives continuus value of basal insulin
    erogated,
    while temp_basal gives when the erogation of basal
    insulin is off in a certain day time: f.e. '2022-01-17'
    This function replace with 0 the values when erogation
    is off in basal insulin series

    INPUT : basal= basal series
            temp_basal = temp_basal series
            time in string format 'yyyy-mm-dd' (year-month
    -day)
    Author: Salvatore Puzzo
    """

```

```
12
13      basal = basal[pd.Timestamp(time) : pd.Timestamp(time)+
    pd.Timedelta('1 day')]
14      temp_basal = temp_basal[pd.Timestamp(time) : pd.
    Timestamp(time)+pd.Timedelta('1 day')]
15      actual_time_stamp = basal.astype(str)
16      day = actual_time_stamp.index[0]
17      day_converted = day.strftime('%Y-%m-%d')
18
19      len_temp_basal = len (temp_basal)
20
21      # se in quel giorno non ci sono temp_basal, ritorna la
        serie basal
22      if len_temp_basal == 0:
23          basal = basal
24
25      # CASO A = se presente una sola misurazione in
    temp_basal metti 0 dal momento in cui inizia fino alla
    fine della giornata di basal
26      if len_temp_basal == 1:
27          basal.loc[temp_basal.index[0]:basal.
    last_valid_index()] = 0
28          basal = basal
29      # CASO B = se ci sono due misurazioni metti 0 dal
    momento in cui inizia fino alla seconda misurazione
30      if len_temp_basal == 2:
31          basal.loc[temp_basal.index[0]:temp_basal.index[1]]
        = 0
32          basal = basal
33      # CASO C = se ci sono tre misurazioni: CASO B + CASO A
34      if len_temp_basal == 3:
35          basal.loc[temp_basal.index[0]:temp_basal.index[1]]
        = 0
36          basal.loc[temp_basal.index[2]:basal.
    last_valid_index()] = 0
37          basal = basal
38      # CASO D = con 4 misurazioni: CASO B + CASO B
39      if len_temp_basal == 4:
40          basal.loc[temp_basal.index[0]:temp_basal.index[1]]
        = 0
41          basal.loc[temp_basal.index[2]:temp_basal.index[3]]
        = 0
42          basal = basal
43
44      # CASO E = con 5 misurazioni: CASO B + CASO B + CASO A
45      if len_temp_basal == 5:
46          basal.loc[temp_basal.index[0]:temp_basal.index[1]]
        = 0
47          basal.loc[temp_basal.index[2]:temp_basal.index[3]]
        = 0
48          basal.loc[temp_basal.index[4]:basal.
    last_valid_index()] = 0
49          basal = basal
50      # CASO F = con 6  misurazioni: CASO B + CASO B + CASO
    B
51      if len_temp_basal == 6:
52          basal.loc[temp_basal.index[0]:temp_basal.index[1]]
        = 0
```

```
53          basal.loc[temp_basal.index[2]:temp_basal.index[3]]
    = 0
54          basal.loc[temp_basal.index[4]:temp_basal.index[5]]
    = 0
55          basal = basal
56      return basal
57
```

Listing 4.6: Funzione che aggiorna l'insulina basale

3. Aggiornamento della feature bolus, aggiungendo informazioni anche sul bolo di tipo *normal* e *double wave*. La quantità di bolo, rifacendosi a [56] è stato convertito in unità di bolo al minuto, dividendo il bolo di tipo "normale" per 5 minuti (periodo di campionamento) e dividendo il bolo di tipo double wave (a lento rilascio) per la sua intera durata.
4. Conversione della temperature della pelle e dell'aria in gradi Celsius

```
1       def fahr_to_celsius(temp_fahr):
2           """Convert Fahrenheit to Celsius
3
4               Return Celsius conversion of input"""
5           temp_celsius = (temp_fahr - 32) * 5 / 9
6           return temp_celsius
7
```

Listing 4.7: Funzione che converte i gradi Fahrenheit in Celsius

5. Reindicizzazione di ogni feature ad una serie di riferimento con indici da mezzanotte a mezzanotte, con frequenza di campionamento di 5 minuti.

```
1 x_temp = x_temp.reindex(lista_temp)
```

Listing 4.8: Esempio di reindexing del glucosio alla serie nulla di riferimento

6. Nuovo riempimento con zeri delle variabili non continue

```
1 h_temp = h_temp.fillna(0)
```

Listing 4.9: Esempio di fillna della variabile rappresentante i carboidrati. In questo caso fillna trova e sostituisce i valori NaN semplicemente con 0

7. Riallineamento di ogni feature in un unico DataFrame di Pandas. Tramite il parametro *how = 'outer'* della funzione *join* è possibile allineare le colonne considerando gli indici esterni già presenti con un effetto a catena che permette la costruzione del dataset feature dopo feature.

```
1
2 prova_join_temp = x_temp.join(y_temp, how='outer').join(
    z_temp, how='outer').join(h_temp, how='outer').join(
    m_temp, how='outer').join(n_temp, how='outer').join(
    o_temp, how ='outer').join(pq_temp, how ='outer').join(
    s_temp, how ='outer').join(t_temp, how ='outer')
```

Listing 4.10: Allineamento delle feature tramite la funzione join che permette di creare il dataset finale grezzo

8. Sostituzione della feature Air Temp, cioè temperatura dell'aria, con Delta Temp, caratterizzata dalla differenza tra temperatura della pelle e temperatura dell'aria. Il motivo di tale sostituzione risiede nel fatto che è

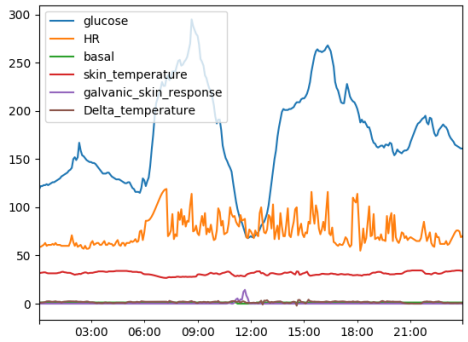


Figura 4.3: Andamento delle feature continue in una giornata tipo del dataset completo

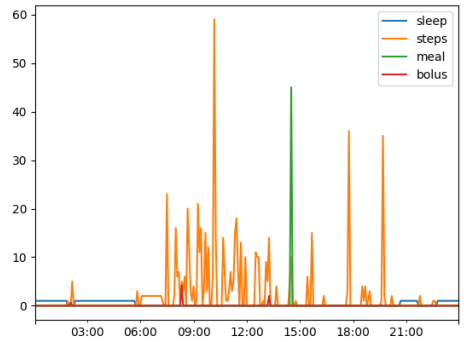


Figura 4.4: Andamento delle feature non continue in una giornata tipo dal dataset completo

dannoso per l'allenamento di una rete neurale somministrare dati molto simili tra loro, di conseguenza per non perdere anche le informazioni derivanti dalla conoscenza della temperatura dell'aria esterna, si è preferito creare una feature fittizia che abbia:

- valori con range molto limitato;
- valori effettivi molto piccoli simili a 1;
- valori che trasportano la conoscenza di una caratteristica.

Tutte queste caratteristiche sono comunemente ben accette se si parla di Machine Learning, poichè aiutano la macchina ad estrarre conoscenza più facilmente dai dati empirici.

```
1 dataset_grezzo['Delta_temperature'] = dataset_grezzo['
    skin_temperature'] - dataset_grezzo['air_temperature']
```

Listing 4.11: Generazione di una feature fittizia chiamata Delta Temp

In questo modo si otterrà un DataFrame completo per il singolo paziente in cui i dati sono perfettamente sincronizzati (riquadro verde nell'immagine 4.5 dove è possibile vedere le feature finali all'interno dei box corrispondenti).

Durante la fase di allenamento e testing delle reti neurali non sempre verranno utilizzate tutte le feature, ma è comunque importante possedere un dataset completo e pronto per qualsiasi ulteriore implementazione e/o ingegnerizzazione di nuove feature a partire da quelle appena create.
Esaminando le figure 4.3 e 4.4, si possono osservare gli andamenti delle grandezze continue (grafico a sinistra) e non continue (grafico a destra). In particolare, nella figura 4.3 sono presenti gli andamenti del: glucosio (blu), frequenza cardiaca (arancione), insulina basale (verde), temperatura della pelle (rosso), risposta galvanica della pelle (viola) e la differenza tra temperatura della pelle con la temperatura dell'aria (marrone). In figura 4.4, troviamo i valori di: sonno in valori binari(blu), numero di passi (arancio), quantità di carboidrati ingeriti (verde) e bolo di insulina (rosso).

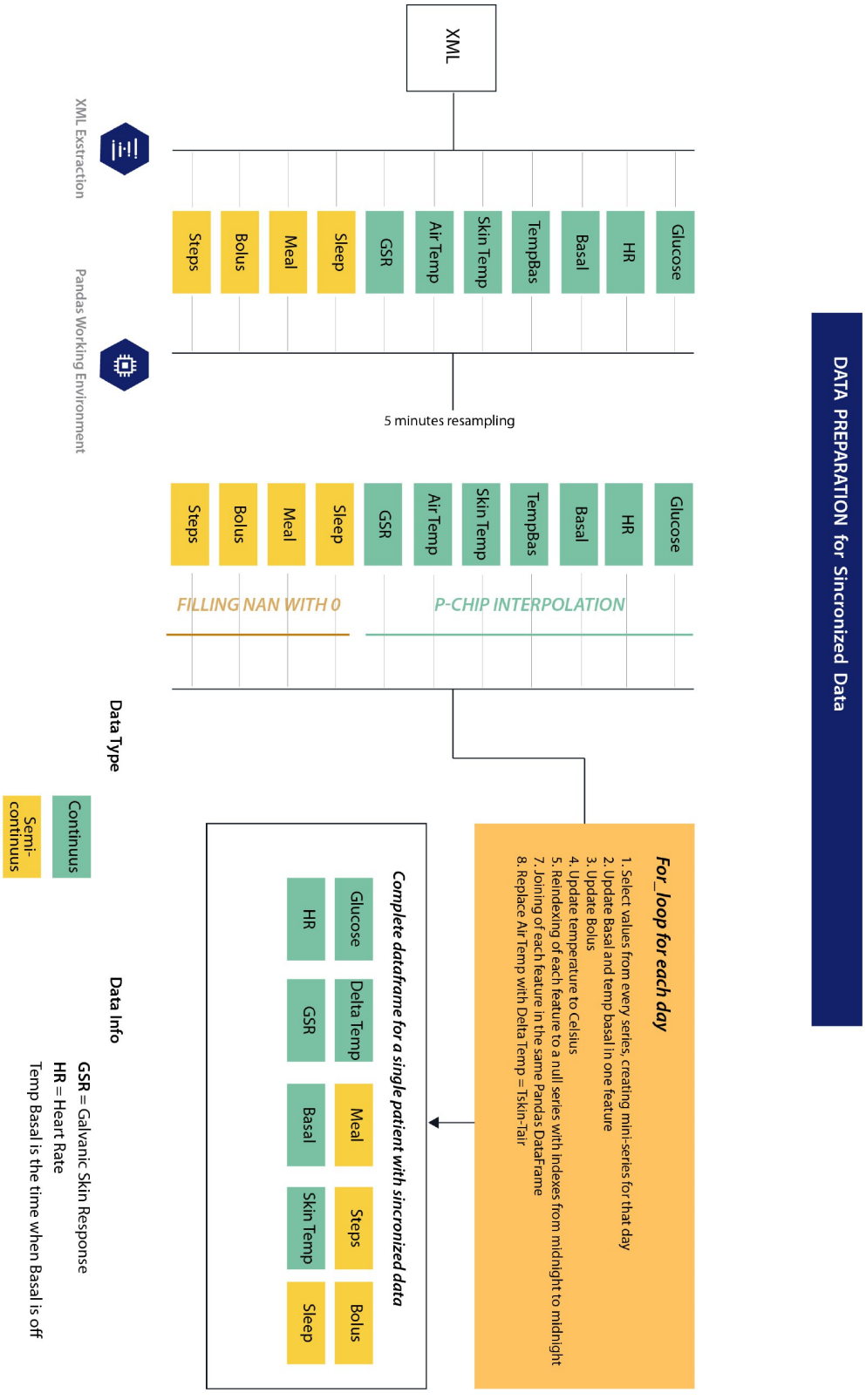


Figura 4.5

Il riassunto grafico delle azioni per poter raggiungere la sincronizzazione. Osservare il flusso di lavoro da sinistra a destra, ruotando di un angolo retto il foglio in senso antiorario.

FASE 3 - Analisi dei missing value

L'analisi dei valori mancanti in una serie temporale è di vitale importanza non solo nella fase di sviluppo, ma anche in fase di *debugging* e messa a punto del codice. Grazie alla libreria Pandas, è stato possibile creare un codice che identificasse i NaN, cioè le misurazioni mancanti, sia per singola feature sia per tutto il dataset. Viene presentato il codice in Python della funzione per l'analisi dei missing value, che prende in input l'intero DataFrame.

```
def count_missing_data (df):
    print('Nan per columns')
    per_columns = df.isnull().sum()
    print('Nan percentage_per_columns')
    percentage_per_columns = df.isnull().sum() / len(df) * 100
    print('Nan total')
    total = df.isnull().sum().sum()
    print('Nan percentage_on_total')
    percentage_on_total = total / ( df.shape[0]* (df.shape[1]-1))
    return per_columns, percentage_per_columns, total,
   percentage_on_total
```

Listing 4.12: Funzione per il conteggio dei valori mancanti

```
"""lista missing data_post_sincro: HOW-TO-READ.
        lista di missing data per colonna,
        lista di missing data per colonna percentuali,
        float64 : numero di missing data totali, missing data
   totali in percentuale
        """
glucose                    1230
sleep                         0
steps                         0
meal                          0
HR                            1
bolus                         0
basal                        96
skin_temperature              1
galvanic_skin_response        1
Delta_temperature             1
dtype: int64,

glucose                   10.415785
sleep                      0.000000
steps                      0.000000
meal                       0.000000
HR                         0.008468
bolus                      0.000000
basal                      0.812939
skin_temperature           0.008468
galvanic_skin_response     0.008468
Delta_temperature          0.008468
dtype: float64, 1330, 0.012513995916485544]
```

Listing 4.13: esempio di output alla console per il paziente 559 post sincronizzazione

FASE 4 - Interpolazione e Imputazione

Sia l'interpolazione che l'imputazione sono tra le tecniche più utilizzate per colmare i vuoti temporali delle mancate misurazioni all'interno delle serie storiche, nella fattispecie la serie storica del livello glicemico del sangue. In questa fase sono state analizzate e provate entrambe le tecniche per osservarne gli effetti che sanciranno la scelta del metodo di *filling* che sarà utilizzato in tutto il dataset.

La tecnica di imputazione è stata già utilizzata da Xie et al. [56] e rappresenta il punto di partenza per la scelta della tecnica di filling value, cioè di riempimento dei valori mancanti con dati virtuali. Si tratta di un filtro di smoothing, chiamato filtro di Kalmann, che richiede l'informazione futura per poter eseguire il filtraggio e il conseguente riempimento dei gap. E' una tecnica quindi che non permette il trattamento dei dati in real time. La tecnica di interpolazione, antagonista della precedente, è invece l'interpolazione *p-chip*, acronimo di Piecewise Cubic Hermite Interpolating Polynomial, che utilizza le funzioni di approssimazioni *spline* cubiche monotoniche per assegnare il valore ai nuovi punti. La scelta di questo tipo di interpolazione è basata su 3 principi basilari:

1. essendo una funzione cubica, porta con sé un approccio non lineare, caratteristico dei segnali biologici

2. conserva la monotonia dei dati analizzati

3. impedisce l'*overshoot*, quindi la generazione di dati al di fuori dell'intervallo in cui si manifestano e di fatto non fisiologicamente corretti

A livello implementativo, l'imputazione è stata eseguita tramite la funzione **na.kalman**, facente parte della libreria *ImputeTS*, con il metodo di input impostato su "StructTS", mentre il linguaggio utilizzato è R, uno dei più noti linguaggi utilizzati per il Machine Learning. Il codice di tale implementazione è riportato in seguito:

```
library(imputeTS)
patient_563 <- read.csv("/patient_563_DF_training_just_glucose")
patient_563_imputed <- na.kalman(patient_563, model ="StructTS")
```

Listing 4.14: Kalmann imputation in linguaggio R

La funzione per implementare l'interpolazione pchip fa parte invece della sottoclasse **DataFrame** della libreria Pandas, ed è definita come parametro negli input della funzione **interpolate**.

Esaminiamo adesso le figure 4.6, 4.7 e 4.8, in cui si vuole riportare il diverso effetto delle tecniche di interpolazione e imputazione su una stessa porzione temporale. In 4.6, si osserva il segnale filtrato tramite imputazione (in rosso) rispetto al segnale originale in blu; in 4.7, si osserva il segnale interpolato con la tecnica p-chip (in verde) rispetto al segnale originale (in blu); in 4.8, si osserva il segnale interpolato con la tecnica p-chip (in verde) rispetto al segnale filtrato con imputazione (in rosso);

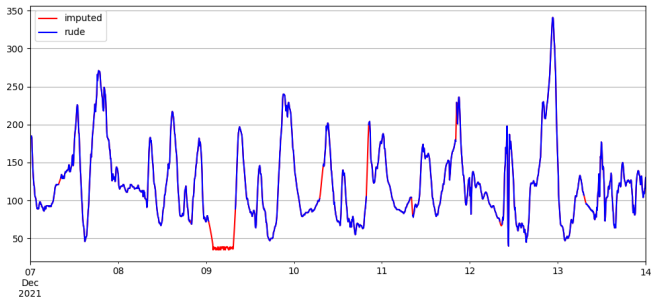


Figura 4.6: Effetto dell'imputazione tramite filtro kalmann in rosso su segnale CGM in blu

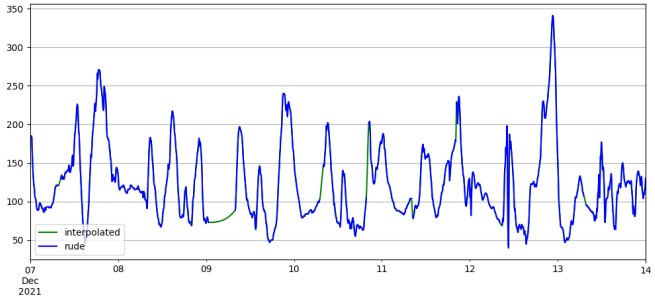


Figura 4.7: Effetto dell'interpolazione pchip in verde su segnale CGM in blu

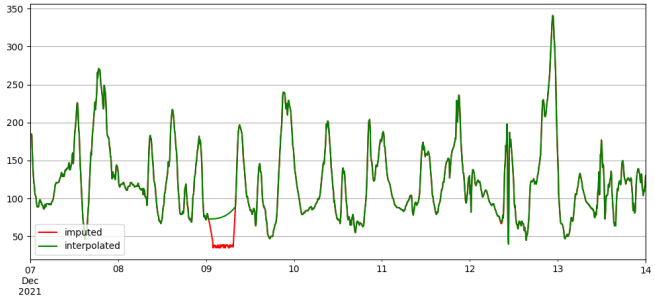


Figura 4.8: Differenza tra imputazione in rosso e interpolazione in verde. E' bene notare che i gap minori sono trattati allo stesso modo dalle due tecniche. Ciò si intuisce dal fatto che la parte verde nelle zone interessate sia completamente sovrapposta alla parte rosse, eccetto nel maxi gap tra il 9 e il 10 dicembre

E' stata scelta una porzione di serie temporale del paziente 575, caratterizzata da un gap temporale tra il 9 e il 10 dicembre, nella settimana tra il 7 e il 14 dicembre.

Se da una parte è possibile constatare in tutte le figure che le due tecniche si comportano alla stessa maniera nei gap più piccoli dell'ordine di qualche minuto, notiamo come in gap più corposi dell'ordine di qualche ora, l'effetto interpolante del filtro Kalmann provochi una generazione di dati che vanno oltre il valore fisiologico (nelle figure 4.6 e 4.8 la curva rossa è stata "tagliata" ai limiti fisiologici, anche se in realtà proseguiva in valori molto negativi) e di conseguenza potrebbero essere dannosi per l'allenamento e la validazione delle reti neurali.L'interpolazione, d'altro canto, ha un approccio intrinsecamente più prudente, generando dei valori che non si discostano dal range fisiologico e quindi più idonei per l'utilizzo all'interno dell'apprendimento automatico. L'interpolazione pchip per via delle caratteristiche analizzate e degli effetti sulla serie temporale mostrati è stata scelta come metodo primario di riempimento dei gap non solo del livello glicemico, ma anche di altre feature continue.

FASE 5 - Controllo finale della qualità delle feature

Questa rappresenta la fase finale per il controllo dei gap residui prima della costruzione del dataset finale. Per evitare ulteriori immissioni di dati fittizi provenienti da interpolazioni, si è scelto di riempire i buchi temporali rimanenti con dei valori fisiologici appartenenti al paziente negli stessi momenti ma in giorni precedenti o successivi. A tal fine è stata modificata la funzione **count missing data**, affinché possa essere utilizzata in combinazione con una nuova funzione, chiamata **ultimate filling**. Come è possibile leggere dal codice sottostante, tenendo in considerazione dell'output della funzione di conteggio di NaN modificata, si sfruttano le due semplici funzioni di *fillna* e *shift*, in modo da riempire di buchi temporali (fillna) con valori traslati nel tempo di un particolare valore (shift). In pratica si tratta un ciclo for in grado di colmare i gap presenti con un valore 1, 2, 3 giorni dopo o -1, -2, -3 giorni prima del valore effettivo, dove 1 giorno è uguale a 288 timesteps.

```
import pandas.DataFrame
def count_missing_data (df):
    per_columns = df.isnull().sum()
    a = list(per_columns)
    if all(v == 0 for v in a) == True:
        missing = False
    else:
        missing=True

    return per_columns, missing

def ultimate_filling(patient_test, missing):
    freqs = ['1d', '-1d', '2d', '-2d', '3d', '-3d', '4d', '-4d']
    for freq in freqs :
        if not missing:
            print ("No more NaN in your DataFrame")
            break
        if missing:
            patient_test = patient_test.fillna(patient_test.shift(
   freq=freq))
            per_columns, missing = count_missing_data(patient_test)

    return patient_test, per_columns
```

Listing 4.15: Doppia funzione per il controllo di qualità finale

La funzione darà in output il DataFrame (patient test) con i valori mancanti sostituiti con i valori target e un ulteriore report testuale (per columns) per un check finale sulla qualità della sostituzione avvenuta.

4.1.3 Filtraggio del segnale glicemico tramite regolarizzazione di Tikhonov

Rifacendosi all'articolo [50], l'implementazione del filtraggio di tipo smoothing del segnale glicemico è affidato alla regolarizzazione di Tikhonov, scelta dal gruppo di ricerca per il conveniente trade-off tra gli episodi di glicemia non rilevati e i falsi positivi che si potrebbero trovare in caso di omissione di tale filtro. In tutte le tipologie di sensori volti a rilevare un segnale biologico, esiste sempre una componente di rumore, che deve essere ridotta, prima di somministrare tale segnale alla rete neurale. L'obiettivo è quello di abbassare le componenti frequenziali del rumore (solitamente alte frequenze), ottenendo quindi il segnale filtrato. La regolarizzazione di Tikhonov, utilizzata in [50], produce il segnale $\hat{y}$, dato dalla seguente formula:

$$\hat{y} = U_d * w \tag{4.1}$$

dove $\hat{y}$ è il segnale filtrato, $\mathrm{U_d}$ denota l'operatore integrale con una matrice N x N e w rappresenta il vettore derivata prima N x 1 del segnale di input. Andando più nel dettaglio:

$$U_{\mathrm{d}} = \begin{bmatrix} 1 & 0 & 0\dots & 0 & 0 & 0 \\ 1 & 1 & 0\dots & 0 & 0 & 0 \\ 1 & 1 & 1\dots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & 1 & 1\dots & 1 & 0 & 0 \\ 1 & 1 & 1\dots & 1 & 1 & 0 \\ 1 & 1 & 1\dots & 1 & 1 & 1 \end{bmatrix}$$

Mentre per stimare w, si effettua la minimizzazione del funzionale f(w), dato dalla seguente formula:

$$f(w) = \|y - U_d * w\|^2 + \lambda_d^2 \|L_d * w\|^2 \tag{4.2}$$

dove y è il segnale originale glicemico, $\mathrm{L_d}$ è la matrice operatore derivata seconda e λ_d è un parametro di regolarizzazione. Definendo $\mathrm{L_d}$ come:

$$L_{\mathrm{d}} = \begin{bmatrix} 0 & 0 & 1\dots & 0 & 0 & 0 \\ 0 & 0 & 0\dots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0\dots & 1 & 0 & 0 \\ 0 & 0 & 0\dots & -2 & 1 & 0 \\ 0 & 0 & 0\dots & -1 & -2 & 1 \end{bmatrix}$$

In [50], vengono selezionati i valori di $\lambda_d = 3000$ e la soluzione analitica che permette di calcolare il segnale filtrato $\hat{y}$, che è pari a:

$$\hat{y} = (U_d^T * U_d + \lambda_d * L_d^T * L_d)^{-1} * U_d^T * y \tag{4.3}$$

L'operazione di filtraggio sarà effettuata ciclicamente solo sui dati di training dei pazienti dopo aver effettuato le operazioni di pre-processing basati sull'interpolazione, mentre per ciò che concerne la fase di testing, è stato deciso di mantenerli grezzi.

In figura 4.9, è possibile osservare l'effetto di filtraggio tramite regolarizzazione di Tikhonov di una porzione del file di training di un paziente dell'OhioT1DM dataset.

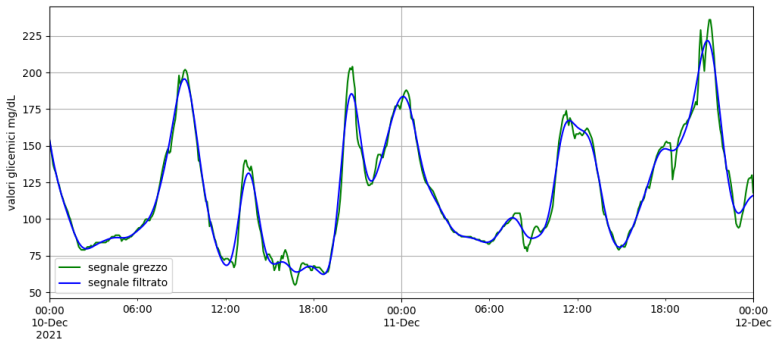


Figura 4.9: Andamento del segnale grezzo (verde) vs Segnale filtrato tratto (blu) dal paziente 575 tra il 10 e il 12 dicembre

4.1.4 Il problema degli sparse data

In questa sottosezione si prenderà in esame il problema degli sparse data. Per prima cosa, traduciamo il termine inglese *sparse*, traducibile come sparso, *aggettivo che rimanda ad un gruppo o ad una serie di gruppi che si dispongono volutamente o casualmente in modo disordinato, a distanze irregolari l'una dall'altra*[58]. Nell'ambito del Machine Learning e nella statistica in generale, una matrice è definita sparsa quando è composta principalmente da valori uguali a zero. Le matrici dense, invece, sono diametralmente opposte poiché sono caratterizzate da valori per lo più diversi da zero.

In aggiunta a tali definizioni, anche una grande quantità di valori mancanti, gli ormai noti *missing value*, rendono i dataset sparsi, manifestando una difficoltà maggiore nell'acquisire previsioni accurate. Come già anticipato nei capitoli precedenti, la scarsità di dati è ampiamente considerata come una delle cause principali della non soddisfacente precisione degli algoritmi di classificazione e regressione nel Machine Learning. Per calcolare la sparsità di un DataFrame dato che non esiste un metodo diretto, è stata implementata una semplice funzione in Python, sfruttando la libreria **numpy**, nota insieme a Pandas per la gestione e il processing di grandi quantità di dati in formato tabulare, matriciale e vettoriale.

```
1 import pandas as pd
2 import numpy as np
3
4 def calculate_sparsity(df):
5     numpy_dataset = df.to_numpy()
6     sparsity = 1.0 - np.count_nonzero(numpy_dataset) /
    numpy_dataset.size
7     return sparsity
```

Listing 4.16: Funzione per il calcolo della sparsity generale dell'intero DataFrame

Il grafico 4.10 mostra i dati ottenuti con il metodo proposto. Come si evince dai dati, è stato calcolato che circa un terzo dell'intero training set di ogni paziente ha valori pari a 0 e ciò non è salutare per una rete neurale, poichè viene utilizzata una grossa fetta di potenza compitazionale nell'imparare dati che non portano informazione. La seconda porzione di codice presentata è invece una funzione che calcola la sparsity per categoria, strumento che sarà poi analizzato nel capitolo 5.

```
import pandas as pd
import numpy as np

def calculate_sparsity_per_columns(df):
    numpy_dataset = df.to_numpy()
    singol_sparsity = []
    for column in range(0, numpy_dataset.shape[1]):
        feature=numpy_dataset[:,column]
        singol_sparsity.append(1.0 - np.count_nonzero(feature) /
   feature.size)
    return singol_sparsity
```

Listing 4.17: Funzione per il calcolo della sparsity per fearture dell'intero DataFrame per paziente

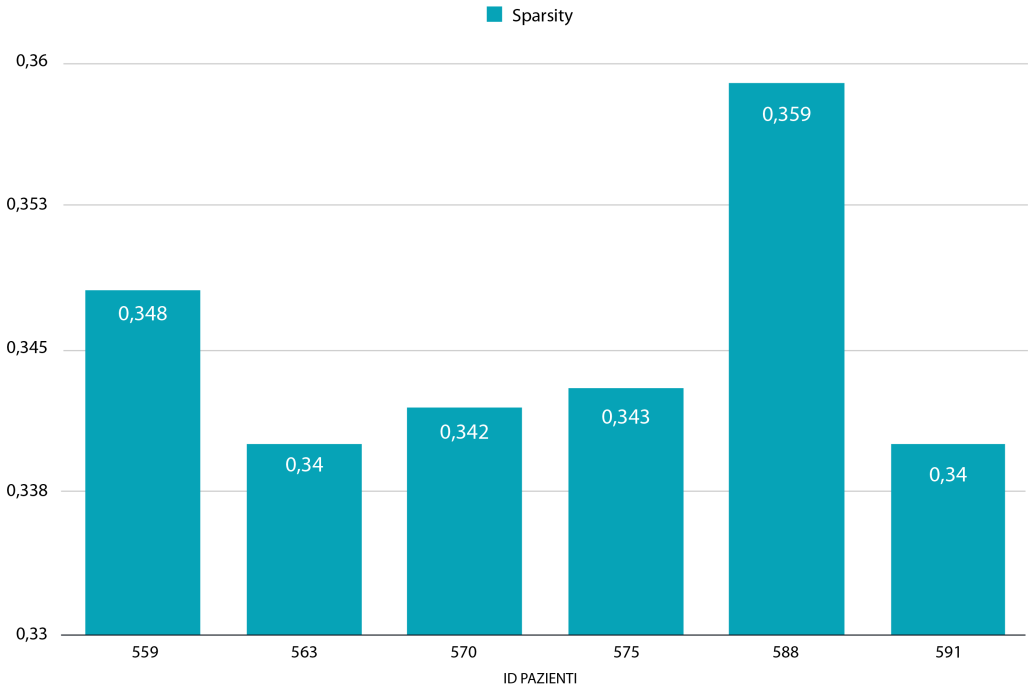


Figura 4.10: Grafico sulla sparsità generale divisa per ID pazienti nei soli dataset di training

Gestione della sparsità nei dataset

I due approcci più utilizzati per gestire gli *sparse data* sono l'interpolazione o la cancellazione delle feature, o porzioni di esse. Nel lavoro di Tesi, questi due semplici approcci sono a priori inutilizzabili, poiché le feature denominate non continue sono state appositamente create con l'idea che quando la grandezza in questione si manifesta, allora il dato campione acquisisce il valore della grandezza, mentre se non esiste la lettura per un determinato periodo, quel dato campione assume a priori

il valore 0. Dato che il dataset "non continuo" è stato costruito in tal modo, è necessario adottare delle tecniche che non modifichino la natura delle serie storiche, in modo da rendere più gestibile tale problema in caso di utilizzo di feature non continue. La soluzione scelta si basa sull'utilizzo del Layer DCAM, argomento della prossima sottosezione, mentre ulteriori approfondimenti riguardanti la distribuzione della sparsity all'interno del dataset saranno analizzati nel paragrafo 5.1.

4.1.5 Layer DCAM

In questa sottosezione viene presentato il Layer DCAM, acronimo che sta per Dual Chamber Absortion Model, che tradotto sta per Modello di Assorbimento a Due Compartimenti. La definizione di tale layer, di novella fattura, è stata trovata in letteratura [59] e consiste nel calcolare una funzione parametrica che, grazie alla definizione dei parametri k_1 e k_2, sia in grado di riprodurre l'andamento continuo di assorbimento, basato su un modello matematico a due compartimenti A e B come visualizzato in figura 4.11.

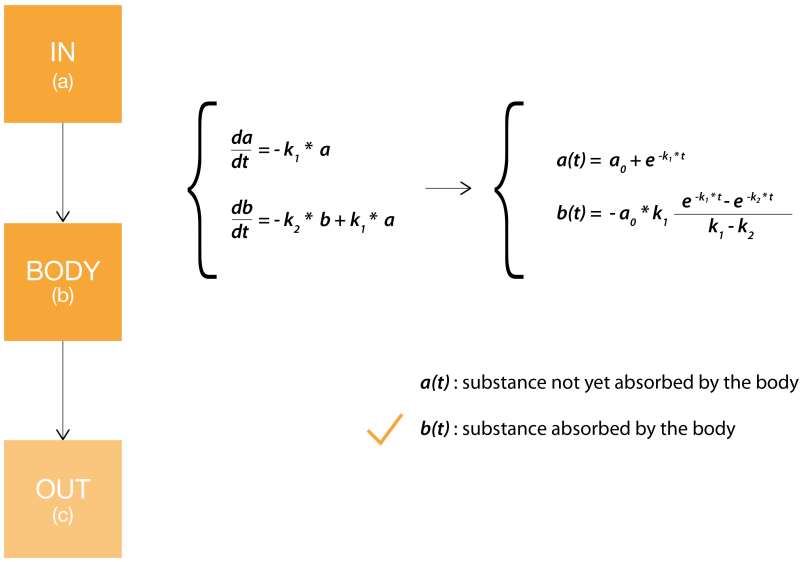


Figura 4.11: Immagine tratta da [59] che riassume il modello bicompartimentale e l'origine delle equazioni trattate.

La caratteristica più interessante di tale layer è che si pone all'inizio della catena neurale, agendo da "filtro" e trasformando la quantità di sostanza presente nella feature selezionate (ad esempio Pasti) secondo un modello di assorbimento e decadimento caratteristico per ogni feature processata, a prescindere dalla natura della sostanza stessa. Si riassumono in ordine gli step che hanno portato alla scelta di tale layer:

1. Gli *sparse data* sono un problema significativo che influisce sulle prestazioni della rete
2. Si cerca un metodo che cambi l'impostazione dei dati sparsi in un modo compatibile con le reti neurali
3. Si utilizza il layer DCAM, che agisce senza modifiche nel set di dati ponendosi all'inizio della catena neurale

Osservando con attenzione il codice implementativo del DCAM layer, quest'ultimo può essere definito secondo tre livelli:

- DCAM come Classe
- DCAM come Funzione
- DCAM come Layer

DCAM as a Class - DCAM come Classe

È una classe del linguaggio Python, con 4 funzioni definite al suo interno:

1. *__init__*, il cosiddetto costruttore in linguaggio ad oggetti. Quando viene creata un'istanza della classe DCAM, automaticamente viene richiamata anche tale funzione nativa che definisce sia l'oggetto DCAM stesso e sia l'elenco di feature selezionate come attributi;
2. *build* è la funzione che definisce i pesi che saranno allenabili dalla rete neurale in cui il layer è posto;
3. *dexp* definisce la funzione parametrica che implementa l'assorbimento e il decadimento in base al modello matematico semplificato di riferimento.
4. *call* funzione che permette di invocare la funzione dexp.

In seguito, sarà presentato il codice di questa sottosezione.

```
from keras.layers import Layer
class DCAM(Layer):
    """
    This layer exploit the Dual-Chamber Absorption model to make the sparse data continuus.
    """

    def __init__(self, sparse_feature= [x,y,z], **kwargs):
        self.sparse_feature = sparse_feature
        super(DCAM, self).__init__(**kwargs)

    def build(self, input_shape):
        n = len(self.sparse_feature)

        # Create a trainable weight variable for this layer.
        init=np.ones((input_shape[-1],2),dtype='float32')
        init[:,1]=0.5
        self.kernel = K.variable(init)
        self.trainable_weights=[self.kernel]
        super(DCAM, self).build(input_shape)  # Be sure to call this at the end

    def dexp(self,data):...

    def call(self, x):
        return self.dexp(x)
```

Listing 4.18: Funzioni all'interno della classe DCAM. Si noti che x y z sono gli indici delle feature scelte dall'utente su cui si vuole che il DCAM agisca

DCAM as a function - DCAM come Funzione

La funzione cuore di questo layer è la **dexp**, già introdotta nella definizione del DCAM come classe. Oltre alla definizione matematica della funzione (vedi l'equazione 4.4) nella prima parte, è importante sottolineare che il sistema creato in [59] si comporti come un sistema lineare, con la presenza di input, output, risposta all'impulso e operazioni basate sulla convoluzione.

La funzione di assorbimento/decadimento (dexp) della sostanza viene applicata sulle feature selezionate grazie all'uso di maschere, mentre le caratteristiche non sparse sono immuni a tale layer. In seguito è riportata la funzione *dexp* e il codice da cui è estratta:

$$f(x) = -x * k_1 \frac{e^{-k_1 t} - e^{-k_2 t}}{k1 - k2} \tag{4.4}$$

```
def dexp(self,data):
    calc = lambda x, t, k1,k2: x * -k1 * ((K.exp(-k1 * t) - K.exp(-
   k2 * t)) / (k1 - k2))

    l=data.shape[1]
    channels=data.shape[-1] #canali= feature
    data=K.temporal_padding(data,padding=(0,l-1))#zero padding
   causale
    data = K.expand_dims(data,axis=1)#aggiungi una dimensione
    imp = K.variable(K.zeros((l,channels)))
    i=K.expand_dims(K.constant(list(range(0,l))))
    imp=calc(1, i,K.expand_dims(self.kernel[:,0],axis=0),K.
   expand_dims(self.kernel[:,1],axis=0))

    mask=np.zeros((l,channels,channels))#crea una maschera per
   tirare fuori i 9 canali e separare le convoluzioni
    for i in range(0,channels):
        mask[:, i, i] = 1

    fsel_w=np.zeros((l,channels))#fsel crea la maschera per
   selezionare le feature che ho dichiarato
    fsel_w[:,self.sparse_feature]=1
    fsel_w=K.variable(fsel_w,name='fsel_w')

    fsel_b=np.zeros((l,channels))#b bias w pesi
    fsel_b[0,:]=1#delta di kronecker
    fsel_b[0,self.sparse_feature]=0#in questo caso, non metto a 1
   ma a 0, per impedire che venga modificato
    fsel_b = K.variable(fsel_b, name='fsel_b')

    imp=imp*fsel_w+fsel_b#per pesi + bias
    imp=K.expand_dims(imp)
    imp=K.repeat_elements(imp,channels,-1)
    imp=imp*mask#applica la maschera
    imp=K.expand_dims(imp,axis=0)
    c = K.conv2d(data, imp, padding="valid",strides=(1,1),
   data_format='channels_last')
    c=K.squeeze(c,axis=1)#toglie dimensione extra
    return c
```

Listing 4.19: Funzione dexp della classe DCAM

DCAM as a Layer - DCAM come Layer

L'ultimo livello di definizione del DCAM è il livello di definizione più alto, che contiene tutte le altre definizioni, agendo da front-end per il programmatore, in quanto si da la possibilità di dare in input la lista degli indici delle feature su cui si vuole applicare il sistema DCAM. E' consigliabile utilizzare tale layer all'inizio della catena neurale, subito dopo la definizione del modello sequenziale, proprio come nell'esempio riportato sotto.
L'input di DCAM è rappresentato da un oggetto lista tra [ ], dove il numero riportato, rappresenta l'indice della colonna del set di feature scelto per la fase di allenamento e validazione del modello neurale.

```
def build_VANILLA(id_patient,X,y,DCAM,cells,epochs,batch_size,
    n_steps_in=12,n_steps_out=12):
    # the dataset knows the number of features, e.g. 10
    n_features = y.shape[2]
    # define model
    verbose=2
    model = Sequential()
    model.add(DCAM([1,3]))
    model.add(LSTM(cells, activation='relu'))
    model.add(LSTM(cells, activation='relu'))
    model.add(LSTM(cells, activation='relu'))
    model.add(Dense(n_features))
    model.compile(optimizer='adam', loss='mse')

    history=model.fit(X, y, epochs=epochs, verbose=
    verbosebatch_size=batch_size, validation_split=0.2,
                callbacks=[update, EarlyStopping(patience=10),
    saveCheckpoint(filename)])
    return model,history,filename
```

Listing 4.20: Esempio di utilizzo del layer DCAM in un'architettura neurale

4.1.6 Creazione di nuove feature

L'utilizzo del layer DCAM potrebbe rappresentare una risposta al bisogno di abbassare l'incidenza della sparsity tramite la trasformazione di alcune feature(tipo i pasti), ma si è cercato comunque di intraprendere anche altre strade, ad esempio creando nuove feature. Sono state implementate, infatti, ulteriori possibili feature:

1. **InsulinDeliveryRate**, il totale apporto insulinico che il paziente sfrutta nell'arco della giornata. L'articolo [56] ha ideato questo approccio per poter unire in un'unica feature le unità insuliniche basali e di bolo. E' stato riprodotto il metodo presentato nell'articolo, tradotto nel seguente codice Python:

```
patient['InsulinDeliveryRate'] = patient['basal'] / 5 + patient
    ['bolus']
```

Listing 4.21: Creazione della feature IDR (InsulinDeliveryRate) con apporto totale di insulina sia basale che di bolo in un DataFrame Pandas

2. **MEST calories**, la trasformazione in calorie dei carboidrati forniti dai pasti e dai passi effettuati dal paziente. La parola MEST, neologismo coniato in questo lavoro, deriva infatti da ME (meal = pasti) e ST (steps = passi). E' una feature caratterizzata dalla differenza istante per istante dei valori di calorie riferite ai pasti con quelle bruciate con i passi. In tale feature si hanno dei valori positivi quando il paziente ingerirà carboidrati e valori negativi quando invece effettuerà dei passi, poiché si vuole invitare la rete a comprendere che i passi bruciano calorie, mentre i pasti le accumulano. Vengono solamente considerati i valori calorici dei dati di cui si dispongono, calcolati attraverso i seguenti valori di conversione trovati in letteratura [60], [61]:

    - calorie (pasti) = grammi (carboidrati ingeriti) * 4
    - calorie (passi) = numero di passi * 0.04

```
patient['MESTcalories'] = patient['meal'] * 4 - patient['steps']
    ] * 0.04
```

Listing 4.22: Creazione della feature MEST calories con apporto totale di calorie proveniente da pasti (+) e passi (-) in un DataFrame Pandas

4.2 Ambulatory Glucose Profile home made

Al fine di avere un ulteriore e potenziale strumento di convalida clinica oltre alla Clarke Error Grid (vedi sezione 5.3), è stato deciso di implementare un AGP (Ambulatory Glucose Profile) report sfruttando tutte le indicazioni fornite nell' International Consensus on Use of Continuous Glucose Monitoring [12], la cui valenza clinica e diagnostica è stata già discussa nella sezione 1.2. Il codice pensato per tutti i pazienti, analizzerà i dati provenienti dal training set e si organizzeranno i dati sfruttando le potenzialità dei *dict* di Python, cioè delle strutture gerarchicamente ordinate secondo *chiave:valore* in grado di interagire con una grande mole di dati e facilmente accessibili dall'utilizzatore/programmatore. Solitamente, gli AGP report sono automaticamente ottenibili dalle applicazioni mobile collegate ai vari dispositivi di lettura CGM, ma non possedendo alcun dispositivo commerciale CGM e di conseguenza alcuna mobile APP da cui attingere gli algoritmi per la generazione del report, è stato deciso di implementare una versione *open source*, molto simile sia nella sostanza che nella forma alla versione commerciale.

4.2.1 Script basato sulle linee guida

In base alle note 15 regole (14 metriche analitiche e 1 standard di visualizzazione) fornite in [12], è stato elaborato un codice molto corposo in grado di analizzare, estrapolare, elaborare ed infine visualizzare i dati richiesti per tutti i pazienti facenti parte dell'OhioT1DM dataset. Nella parte sottostante sarà presentato un esempio di output, caratterizzato dalla presentazione dei dati tramite i dizionari e liste di dizionari, cioè elementi o liste di elementi disposti secondo il costrutto *chiave*:*valore*. Lo schema in Figura 4.12 mostra graficamente la suddivisione dei dati delle metriche AGP.

L'output dello script *AGP_metrics* è presentato secondo due ramificazioni principali entrambe fondamentali sia per l'analisi generale del paziente e che per l'analisi specifica degli eventi di glicemia nei singoli giorni. Nel dettaglio, le due diramazioni presentano le seguenti caratteristiche:

- **per class analysis**:, cioè le analisi specifiche giornaliere.
  Per ogni tipologia di livello glicemico vHYPO, HYPO, TARGET, HYPER, vHYPER, vvHYPER che stanno per ipoglicemia grave, ipoglicemia, livello sicuro/target, iperglicemia, iperglicemia elevata, iperglicemia grave si analizzano il numero, la durata media e la somma totale degli episodi glicemici nel giorno specifico, che rappresenta la chiave del dizionario. Questo set di informazioni sono utili per disporre della memoria giornaliera degli eventi nel caso il medico o il paziente stesso sia interessato all'andamento delle singole classi glicemiche giorno dopo giorno.

- **general metrics**: metriche globali.
  Sono gli elementi principali che mostrano l'andamento clinico globale del paziente nei giorni di registrazione analizzati. In ordine sono presenti: valore medio di glucosio, coefficiente di variazione globale, la deviazione standard, numero e percentuale di eventi glicemici totali, indici eA1c (dal GMI in[13]), LGBI e HBGI, valore medio giornaliero della durata degli eventi glicemici, numero di episodi medi e durata media per episodio totali, Area Under Curve normalizzata negli orari di veglia, sonno e durante l'arco di 24 ore.

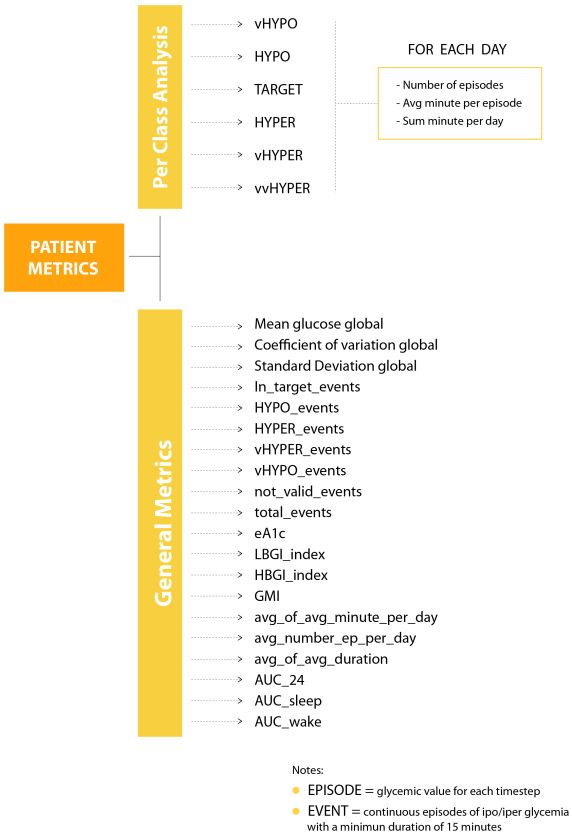


Figura 4.12: Schema per la suddivisione delle analisi generali e specifiche nell'AGP report

Nel codice sottostante, vengono riportate le informazioni estratte dal file di testing del paziente 575. I termini *events* ed *episode* hanno significato differente: *events* rappresenta il singolo valore glicemico rilevato ogni 5 minuti mentre *episode* richiama l'episodio glicemico, definito come evento continuo di iperglicemia o ipoglicemia con durata di almeno 15 minuti.

```
{'patient_metrics':
{'per class analysis':

[{'vHYPO':
{1.0: {'number of episodes': 0, 'avg_minute_per_episode': 0, '
    sum_minute_per_day': 0},
2.0: {'number of episodes': 1, 'avg_minute_per_episode': 30.0, '
    sum_minute_per_day': 30},
3.0: {'number of episodes': 0, 'avg_minute_per_episode': 0, '
    sum_minute_per_day': 0},
4.0: {'number of episodes': 1, 'avg_minute_per_episode': 25.0, '
    sum_minute_per_day': 25},
5.0: {'number of episodes': 1, 'avg_minute_per_episode': 30.0, '
    sum_minute_per_day': 30},
6.0: {'number of episodes': 0, 'avg_minute_per_episode': 0, '
    sum_minute_per_day': 0},
7.0: {'number of episodes': 0, 'avg_minute_per_episode': 0, '
    sum_minute_per_day': 0},
8.0: {'number of episodes': 2, 'avg_minute_per_episode': 35.0, '
    sum_minute_per_day': 70},
9.0: {'number of episodes': 0, 'avg_minute_per_episode': 0, '
    sum_minute_per_day': 0}}},

```

```
{'HYPO':
{1.0: {'number of episodes': 0, 'avg_minute_per_episode': 0, '
    sum_minute_per_day': 0},
2.0: {'number of episodes': 2, 'avg_minute_per_episode': 70.0, '
    sum_minute_per_day': 140},
3.0: {'number of episodes': 0, 'avg_minute_per_episode': 0, '
    sum_minute_per_day': 0},
4.0: {'number of episodes': 2, 'avg_minute_per_episode': 32.5, '
    sum_minute_per_day': 65},
5.0: {'number of episodes': 0, 'avg_minute_per_episode': 0, '
    sum_minute_per_day': 0},
6.0: {'number of episodes': 2, 'avg_minute_per_episode': 62.5, '
    sum_minute_per_day': 125},
7.0: {'number of episodes': 1, 'avg_minute_per_episode': 15.0, '
    sum_minute_per_day': 15},
8.0: {'number of episodes': 3, 'avg_minute_per_episode': 15.0, '
    sum_minute_per_day': 45},
9.0: {'number of episodes': 0, 'avg_minute_per_episode': 0, '
    sum_minute_per_day': 0}}},

{'TARGET':
{1.0: {'number of episodes': 1, 'avg_minute_per_episode': 265.0, '
    sum_minute_per_day': 265},
2.0: {'number of episodes': 4, 'avg_minute_per_episode': 222.5, '
    sum_minute_per_day': 890},
3.0: {'number of episodes': 2, 'avg_minute_per_episode': 260.0, '
    sum_minute_per_day': 520},
4.0: {'number of episodes': 5, 'avg_minute_per_episode': 152.0, '
    sum_minute_per_day': 760},
5.0: {'number of episodes': 5, 'avg_minute_per_episode': 98.0, '
    sum_minute_per_day': 490},
6.0: {'number of episodes': 5, 'avg_minute_per_episode': 114.0, '
    sum_minute_per_day': 570},
7.0: {'number of episodes': 5, 'avg_minute_per_episode': 73.0, '
    sum_minute_per_day': 365},
8.0: {'number of episodes': 5, 'avg_minute_per_episode': 204.0, '
    sum_minute_per_day': 1020},
9.0: {'number of episodes': 5, 'avg_minute_per_episode': 31.0, '
    sum_minute_per_day': 155}}},

{'HYPER':
{1.0: {'number of episodes': 3, 'avg_minute_per_episode': 165.0, '
    sum_minute_per_day': 495},
2.0: {'number of episodes': 4, 'avg_minute_per_episode': 41.25, '
    sum_minute_per_day': 165},
3.0: {'number of episodes': 3, 'avg_minute_per_episode': 33.33, '
    sum_minute_per_day': 100},
4.0: {'number of episodes': 5, 'avg_minute_per_episode': 25.0, '
    sum_minute_per_day': 125},
5.0: {'number of episodes': 6, 'avg_minute_per_episode': 39.17, '
    sum_minute_per_day': 235},
6.0: {'number of episodes': 5, 'avg_minute_per_episode': 47.0, '
    sum_minute_per_day': 235},
7.0: {'number of episodes': 5, 'avg_minute_per_episode': 53.0, '
    sum_minute_per_day': 265},
8.0: {'number of episodes': 4, 'avg_minute_per_episode': 48.75, '
    sum_minute_per_day': 195},
9.0: {'number of episodes': 7, 'avg_minute_per_episode': 32.86, '
```

```
    sum_minute_per_day': 230}}},

{'vHYPER':
{1.0: {'number of episodes': 6, 'avg_minute_per_episode': 80.83, '
    sum_minute_per_day': 485},
2.0: {'number of episodes': 2, 'avg_minute_per_episode': 75.0, '
    sum_minute_per_day': 150},
3.0: {'number of episodes': 3, 'avg_minute_per_episode': 105.0, '
    sum_minute_per_day': 315},
4.0: {'number of episodes': 5, 'avg_minute_per_episode': 63.0, '
    sum_minute_per_day': 315},
5.0: {'number of episodes': 4, 'avg_minute_per_episode': 123.75, '
    sum_minute_per_day': 495},
6.0: {'number of episodes': 4, 'avg_minute_per_episode': 93.75, '
    sum_minute_per_day': 375},
7.0: {'number of episodes': 4, 'avg_minute_per_episode': 148.75, '
    sum_minute_per_day': 595},
8.0: {'number of episodes': 1, 'avg_minute_per_episode': 15.0, '
    sum_minute_per_day': 15},
9.0: {'number of episodes': 10, 'avg_minute_per_episode': 40.0, '
    sum_minute_per_day': 400}}},

{'vvHYPER':
{1.0: {'number of episodes': 2, 'avg_minute_per_episode': 55.0, '
    sum_minute_per_day': 110},
2.0: {'number of episodes': 0, 'avg_minute_per_episode': 0, '
    sum_minute_per_day': 0},
3.0: {'number of episodes': 1, 'avg_minute_per_episode': 425.0, '
    sum_minute_per_day': 425},
4.0: {'number of episodes': 1, 'avg_minute_per_episode': 15.0, '
    sum_minute_per_day': 15},
5.0: {'number of episodes': 1, 'avg_minute_per_episode': 75.0, '
    sum_minute_per_day': 75},
6.0: {'number of episodes': 1, 'avg_minute_per_episode': 20.0, '
    sum_minute_per_day': 20},
7.0: {'number of episodes': 1, 'avg_minute_per_episode': 50.0, '
    sum_minute_per_day': 50},
8.0: {'number of episodes': 0, 'avg_minute_per_episode': 0, '
    sum_minute_per_day': 0},
9.0: {'number of episodes': 5, 'avg_minute_per_episode': 57.0, '
    sum_minute_per_day': 285}}}]},

'general metrics':

{'Mean glucose global': 154.21,
'Coefficient of variation global': 39.91,
'Standard Deviation global': 61.54,
'In_target_events':
 [{'number': 1510}, {'percentage on total events': 61.08}],
 'HYPO_events':
 [{'number': 100}, {'percentage on total events': 4.05}],
 'HYPER_events':
 [{'number': 631}, {'percentage on total events': 25.53}],
'vHYPER_events':
[{'number': 196}, {'percentage on total events': 7.93}],
 'vHYPO_events':
 [{'number': 35}, {'percentage on total events': 1.42}],
'not_valid_events':
```

```
[{'number': 0}],
'total_events':
[{'number': 2472}],
'eA1c': 6.99,
'LBGI_index': 1.41,
'HBGI_index': 6.79,
'GMI': 7%,
'avg_of_avg_minute_per_day':
[{'vHYPO_list_avg_minute_per_episode': 13.33}, {'
   HYPO_list_avg_minute_per_episode': 21.67}, {'
   HYPER_list_avg_minute_per_episode': 53.93}, {'
   vHYPER_list_avg_minute_per_episode': 82.78}, {'
   vvHYPER_list_avg_minute_per_episode': 77.44}],
'avg_number_ep_per_day':
[{'vHYPO_list_avg_number_ep_per_day': 0.55}, {'
   HYPO_list_avg_number_ep_per_day': 1.11}, {'
   HYPER_list_avg_number_ep_per_day': 4.67}, {'
   vHYPER_list_avg_number_ep_per_day': 1.33}, {'
   vvHYPER_list_avg_number_ep_per_day': 4.33}],
'avg_of_avg_duration':
[{'vHYPO_list_avg_duration': 17.22},
{'HYPO_list_avg_duration': 43.33},
{'HYPER_list_avg_duration': 227.22},
{'vHYPER_list_avg_duration': 108.88},
{'vvHYPER_list_avg_duration': 349.44}],
'AUC_24': 144,
'AUC_sleep': 115,
'AUC_wake': 154}}
```

Listing 4.23: Esempio di output per il paziente 575 per 9 giorni di registrazione

4.2.2 La visualizzazione dell'AGP

Rappresentare l'andamento glicemico giornaliero e globale è fondamentale per consentire al medico di effettuare la giusta diagnosi e pianificare l'adatta terapia al paziente. La visualizzazione dall'AGP, come spiegato nella sezione 1.2, raccoglie le registrazioni di 14 giorni di glucosio in mg/dL e li rappresenta in un unico grafico della durata di 24 ore (ovvero 288 timestep) suddiviso nelle distribuzioni percentili corrispondenti al 10°, 25°, 50°, 75° e 90° percentile. La maggiore distribuzione avverrà tra il 75° e il 25° percentile, mentre le escursioni massime, conosciute come scarti interquartili, di glicemia saranno comprese dal 10°(valori ipoglicemici) e dal 90°(valori iperglicemici) percentile. Il metodo presentato di organizzazione e visualizzazione dei dati in base alle indicazioni fornite dall'articolo [62] rende più facilmente leggibile e accessibile il file di output alla console, aiutando ad esempio il programmatore nella costruzione di una GUI (Graphical User Interface), che permetta al paziente di poter monitorare i propri dati glicemici, qualora disponibili, senza dover utilizzare software commerciali.

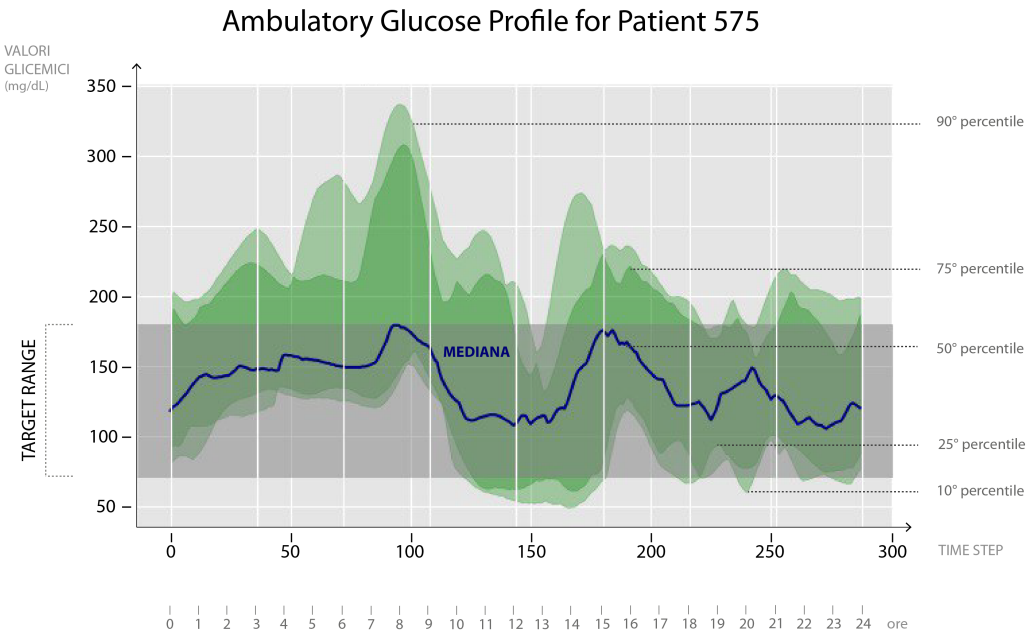


Figura 4.13: Esempio di andamento AGP. Si notino le denominazioni degli assi x e y e le distribuzioni percentili

4.3 Implementazione Reti Neurali

L'ultima sezione di questo capitolo è dedicata alla costruzione delle reti neurali e al *reshaping* dei dati da somministrare in input a quest'ultime.

4.3.1 Creazione delle batch

Come già mostrato nella fase di costruzione del dataset, i dati presenti nel DataFrame Pandas finale sono organizzati in formato tabulare bidimensionale secondo righe e colonne. Le colonne rappresentano le feature del dataset, mentre righe rappresentano la loro evoluzione con un periodo di campionamento di 5 minuti. Le reti neurali però hanno bisogno di un formato particolare per i dati di input, è quindi necessario effettuare un *reshaping*, un cambiamento di forma, del DataFrame. Si sfrutta la funzione **split sequences** in modo da costruire X e Y, cioè i dati già formattati in base al numero di input delle sequenze di ingresso e di uscita. In particolare, dal formato: [campioni, feature] si passa a [campioni, timesteps, feature]. Il formato con *Timesteps* è molto importante per le reti neurali, in quanto si preferisce somministrare i dati organizzati in spazi temporali ristretti chiamati *batch* e in 3 dimensioni, dove la terza dimensione è rappresentata dalla feature. La matrice **X** sarà caratterizzata da dati raggruppati secondo sequenze lunghe 30 campioni, corrispondenti a 150 minuti, per ogni feature selezionata e avrà invece **Y**, cioè la matrice target di output, costituita da sequenze lunghe 18 campioni, corrispondenti temporalmente a 90 minuti. In generale, nella fase di training la rete si allenerà a:

- studiare e memorizzare in X i pattern temporali di sequenze lunghe 30 campioni
- predire le successive sequenze temporali lunghe 18 campioni come in Y

Questi due valori non sono scelti a caso: 30 campioni di input sono stati scelti ispirandosi a [50], mentre i 18 campioni di output sono fondamentali se si vogliono raggiungere orizzonti di previsione di massimo 90 minuti (18 campioni x 5 minuti di campionamento = 90 minuti di batch).

```
def split_sequences(numpy_array, len_batch_input, len_batch_output):
    X, y = list(), list()
    for i in range(len(numpy_array)):
        end_ix = i + len_batch_input
        out_end_ix = end_ix + len_batch_output
        if out_end_ix > len(numpy_array):
            break
        seq_x, seq_y = numpy_array[i:end_ix, :], numpy_array[end_ix:out_end_ix, :]
        X.append(seq_x)
        y.append(seq_y)
    return array(X), array(y)
```

Listing 4.24: Funzione che crea le batch per ogni feature

La matrice di Y sarà ulteriormente ridotta ad una dimensione, in quanto l'intento è allenare la rete su un set multivariato di input, ma ottenere la predizione dell'unica feature di interesse, cioè il glucosio.

```
import numpy as np
patient_n = np.array(patient) #trasformo il pandas DataFrame in
    matrice di numpy
X,y = split_sequences(patient_n, 30, 18)

X.shape
>>>(12626, 30, 10)

y = y[:,:,0:1] #seleziono solo la feature glucosio, mantenendo y in
     3D
y.shape
>>>(12626, 18, 1)
```

Listing 4.25: Esempio applicativo per l'intero file di training del paziente 575 con 12626 campioni corrispondenti a circa 44 giorni di registrazione per la creazione delle matrici X e y

4.3.2 La libreria Keras

Le reti neurali hanno avuto uno sviluppo impressionante negli ultimi anni e la loro crescita è stata senza dubbio alimentata dalla pubblicazione online di librerie molto note nell'ambito di sviluppo come:

- Tensorflow, una potente libreria open source originariamente costruita da Google per uso interno dedicata all'apprendimento automatico.
- Keras, un'API creata per semplificare l'addestramento e la convalida della rete neurale, supportata a partire dal 2017 ufficialmente da Tensorflow stesso.

Nel lavoro di implementazione delle rete neurali saranno sfruttate le librerie di Keras, considerate adeguate per lo scopo della Tesi.

```
from keras.models import Sequential
from keras.layers import LSTM, Conv1D, MaxPooling1D, Flatten,
    ConvLSTM2D, Reshape
from keras.layers import Dense
from keras.layers import RepeatVector
from keras.layers import TimeDistributed, BatchNormalization
```

Listing 4.26: Librerie Keras importate in linguaggio Python

Le tipologie di reti neurali scelte, in cui non sono ancora riportati i parametri ottimali scelti, sono presentate in ordine:

1. **LSTM**, reti ricorrenti Long-Short-Memory-Term
2. Encoder LSTM, Decoder LSTM, note come **ENC-DEC LSTM**
3. Encoder CNN, Decoder LSTM, note come **ENC-DEC CNN-LSTM**

Oltre alle varie tipologie di reti neurali e ai layer che li contraddistinguono, vengono definiti ulteriori elementi comuni alle 4 tipologie appena presentate:

- Primo layer, se utilizzato, della catena è rappresentato dal DCAM, in cui vengono definite le eventuali feature che si vogliono filtrare e rendere continue.

- **Batch Normalization**, è dimostrato [63] che aumenta le performance della rete neurale, tramite la normalizzazione delle mini-Batch in fase di allenamento, consentendo ad esempio l'utilizzo di learning rate più alti, nel caso della Tesi si utilizza un learning rate di 0,001.

- Ottimizzatore **adam**[64], da ADAptive Moment estimation, il migliore tra gli ottimizzatori del processo di apprendimento delle reti neurali.

- Utilizzo dell'**Early Stopping**, con *patience* = 50. Questa caratteristica permette di interrompere la fase di allenamento prematuramente in caso di peggioramento della *loss* (vedi Figura 4.14), l'indice monitorante della qualità dell'allenamento, per 50 epoche consecutive.

- Funzione di attivazione, introdotta nella sezione 2.2.5, è la $ReLU$, funzione che estrae la parte positiva dell'argomento.

- Paramentro di loss scelto è **MSE** (Mean Squared Error), un parametro definito come la somma dei quadrati della differenza tra le variabili calcolate ed effettive divisa per il numero di elementi analizzati.

- **SaveCheckpoint** si è rivelata una funzione utile perché permette di salvare automaticamente i punti di controllo durante e alla fine dell'allenamento, in modo da utilizzare il modello addestrato all'ultimo salvataggio nel caso ci siano interruzioni o errori nella fase di allenamento.

- Il dataset nella fase di training è ulteriormente suddiviso tramite la *validation split* in 80% sub-training set e 20% sub-testing set. Vengono utilizzati i termini sub-training e sub-testing poichè il dataset è ottenuto già diviso in training e testing set.

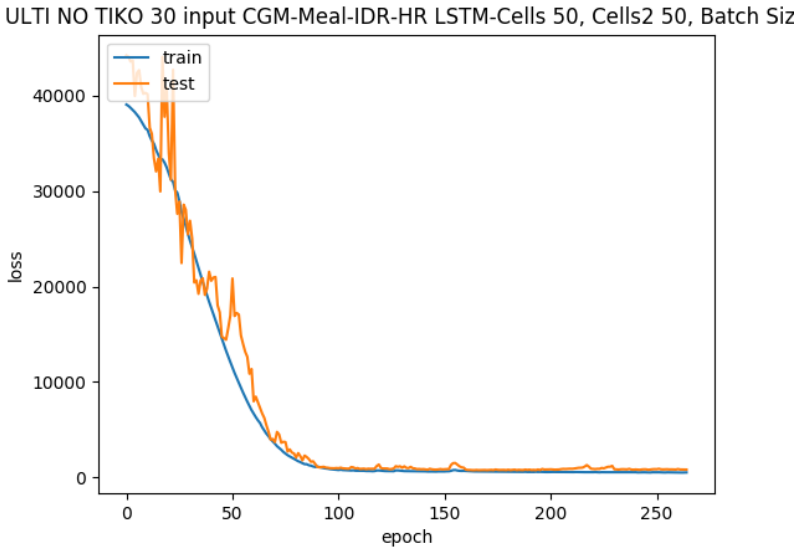


Figura 4.14: Esempio dell'andamento della loss in fase di allenamento del paziente 570. Si noti come nonostante le 500 epoche preimpostate, l'allenamento sia concluso alla 260esima epoca per via dell'Early Stopping

4.3.3 Implementazione di reti neurali ricorrenti

Tra le più note ed utilizzate nello stato dell'arte nella predizione di titoli finanziari, andamenti in borsa e soprattutto del livello glicemico del sangue, le LSTM rappresentano uno step importante e sicuramente una base da cui partire per effettuare l'inferenza glicemica. Sebbene sia stato provato nello stato dell'arte che le LSTM funzionano molto bene con dataset costituiti solamente da letture di glucosio, è obiettivo di tale Tesi, testare se quest'ultime abbiamo la stessa efficacia con l'aggiunta di altre feature collaterali al glucosio, tutte presenti all'interno del dataset costruito secondo i passi mostrati sempre in questo capitolo, ma nella sottosezione Data Preparation.

Sarà presentata per prima la funzione **build VANILLA**, caratterizzata dalla definizione del modello sequenziale, da layer LSTM e di un layer Dense destinato all'inferenza del valore futuro di glucosio.

```
def build_VANILLA(id_patient,X,y,DCAM,cells,epochs,batch_size,
    n_steps_in=12,n_steps_out=12):
     # the dataset knows the number of features, e.g. 10
     n_features = y.shape[2]
     # define model
     verbose=2
     model = Sequential()
     model.add(DCAM([1]))
     model.add(LSTM(cells, activation='relu'))
     model.add(LSTM(cells, activation='relu'))
     model.add(LSTM(cells, activation='relu'))
     model.add(Dense(n_features))
     model.compile(optimizer='adam', loss='mse')

     history=model.fit(X, y, epochs=epochs, verbose=
    verbosebatch_size=batch_size, validation_split=0.2,
                callbacks=[update, EarlyStopping(patience=10),
    saveCheckpoint(filename)])
     return model,history,filename
```

Listing 4.27: Funzione che assembla e allena un modello neurale LSTM

E' un'architettura *stacked*, dove i layer nascosti LSTM sono posti in serie, aventi le seguenti caratteristiche:

- *batch size* (grandezza della batch)
- layer DCAM, introdotto nella sezione 4.1.5, utile per eventuali ottimizzazioni delle feature non continue. In questo caso è riportato il numeri 1 come indice della feature dei pasti;
- 3 layer LSTM con lo stesso numero di celle;
- layer Dense per ottenere la predizione della feature glucosio;
- la funzione di attivazione utilizzata è la ReLU (vedi equazione 2.7).

4.3.4 Il costrutto degli Encoder-Decoder

La seconda tipologia di rete implementata è il costrutto ENC-DEC, cioè Encoder-Decoder. Come già anticipato nel capitolo 2, questo approccio è caratterizzato da

due reti neurali distinte, collegate da uno spazio latente intermedio. In seguito vengono presentati gli aspetti più importanti, il codice in Python e i parametri nella funzione **build LSTM**.

```
def build_LSTM(X,y,DCAM,cells,cells2,epochs,batch_size,n_steps_in
    =30,n_steps_out=18):
     # the dataset knows the number of features, e.g. 10
     n_features = y.shape[2]
     # define model
     verbose=2
     model = Sequential()
     model.add(DCAM([1]))
     model.add(LSTM(cells, activation='relu'))
     model.add(BatchNormalization())
     model.add(RepeatVector(n_steps_out))
     model.add(LSTM(cells2, activation='relu', return_sequences=True
    ))
     model.add(BatchNormalization())
     model.add(TimeDistributed(Dense(n_features)))
     model.compile(optimizer='adam', loss='mse')
     # fit model
     history=model.fit(X, y, epochs=epochs, verbose=verbose,
    batch_size=batch_size, validation_split=0.2,
                callbacks=[update, EarlyStopping(patience=10),
    saveCheckpoint(filename)])
     return model,history,filename
```

Listing 4.28: Funzione che assembla e allena un modello neurale ENC-DEC LSTM

A parte i parametri e i layer con cui si è già preso confidenza nel'architettura precedente, è possibile notare la funzione **RepeatVector**, ovvero il collo di bottiglia in cui viene depositata la rappresentazione interna dei valori del codificatore. Questa funzione agirà per più volte, una volta per ogni valore della sequenza di output. Questi vettori ripetuti rappresentano il layer di collegamento, che agisce da input alla successiva rete decodificante LSTM. Le caratteristiche più importanti sono:

- *batch size*, cioè grandezza della batch
- layer DCAM, se applicato;
- 1 layer LSTM sia nel codificatore che ne decodificatore;
- layer Dense per ottenere la predizione della feature glucosio;
- la funzione di attivazione utilizzata è la ReLU.

ENC-DEC CNN-LSTM

Simile al costrutto precedente, questa architettura presenta il codificatore CNN, destinato all'estrazione di caratteristiche dei segnali di ingresso, e il decodificatore LSTM, destinato esclusivamente all'inferenza glicemica. Essendo un'architettura ENC-DEC è ancora presente lo stato intermedio. La funzione **build CNN LSTM** mostra i layer caratteristici della rete CNN come Convoluzione 1D, Pooling, introdotti nei paragrafi 2.2.5 e 2.2.5. In particolare, si è scelto una tipologia di CNN classica e ispirata ad AlexNet [33], caratterizzata dall'utilizzo della funzione di attivazione ReLU e dalla combinazione di più layer convoluzionali seguiti dal pooling. La rete convoluzionale ha i seguenti parametri in ingresso:

- numero di batch
- layer DCAM, come nel caso precedente
- numero di filtri convoluzionali
- grandezza del kernel 3x3
- una grandezza di pooling = 2
- 1 layer LSTM nel decoder
- layer Dense per ottenere la predizione della feature glucosio
- funzione di attivazione ReLU

```
def build_CNN_LSTM(X,y,cells,DCAM, batch_size, epochs, n_filters,
    n_steps_out=18):
     # the dataset knows the number of features, e.g. 10
     n_features = y.shape[2]
     # define model
     verbose=2

     model = Sequential()
     model.add(DCAM([1]))
     model.add(Conv1D(filters=n_filters, kernel_size=3, activation='
    relu'))
     model.add(Conv1D(filters=n_filters, kernel_size=3, activation='
    relu'))
     model.add(MaxPooling1D(pool_size=2))
     model.add(Flatten())
     model.add(RepeatVector(n_steps_out))
     model.add(LSTM(cells, activation='relu', return_sequences=True)
    )
     model.add(BatchNormalization())
     model.add(TimeDistributed(Dense(n_features)))
     model.compile(optimizer='adam', loss='mse')
     # fit model
     history=model.fit(X, y, epochs=epochs, verbose=verbose,
    batch_size=batch_size,validation_split=0.2,
                callbacks=[update, EarlyStopping(patience=10),
    saveCheckpoint(filename)])
     return model,history,filename
```

Listing 4.29: Funzione che assembla e allena un modello neurale ENC-DEC CNN LSTM

Capitolo 5

Analisi e Risultati

In questo capitolo si analizzeranno i risultati relativi agli strumenti implementativi presentati nel capitolo precedente. In particolare, ci si soffermerà sui seguenti argomenti:

- Effetto della sparsità dei dati, confrontando il dataset di partenza e il dataset modificato con le nuove feature create
- Analisi sull'effetto del filtraggio dei dati tramite il tool AGP_homemade
- Presentazione delle metriche di valutazione delle predizioni da modelli neurali
- Analisi delle prestazioni dei modelli neurali sviluppati

5.1 Analisi sugli sparse data

La *sparsity*, o sparsità, è collegata a strutture composte principalmente da valori uguali a zero e dannose per il Machine Learning. Rifacendosi ai metodi esposti nel capitolo precedente, è stata calcolata nel dataset con le feature iniziali.

	ID pazienti					
	559	**563**	**570**	**575**	**588**	**591**
Glucosio	0%	0%	0%	0%	0%	0%
Sonno	66,20%	73,60%	70,90%	74,50%	69,40%	71,90%
Passi	77,40%	67,50%	76,20%	68,20%	79,70%	68,90%
Pasti	98,90%	99,30%	99,80%	99,20%	99,40%	99,20%
Freq.Cardiaca	0%	0%	0%	0%	0%	0%
Ins. Bolo	99,70%	99,50%	93,80%	99,20%	99,70%	99,50%
Ins. Basale	5,00%	0,10%	0,60%	0,50%	8,20%	7,90%
Temp. Pelle	0%	0%	0%	0%	0%	0%
GSR	0%	0%	0%	0%	0%	0%
Delta Temp.	0,60%	0,40%	0,30%	0,40%	2,60%	0,40%

Tabella 5.1: Tabella che riporta i dati di sparsity per feature all'interno dei file dell'intero training set diviso per paziente

Secondo i dati della tabella 5.1 e al contributo percentuale che esso hanno per ciascun paziente nel grafico 5.1, si possono ottenere informazioni utili non solo per

capire il grado di completezza del dataset, ma anche l'andamento generale dei parametri vitali prelevati dal paziente durante l'intero arco di registrazione per il Trial Clinico. In particolare, si nota che:

- Le feature Continue di: **Glucosio**, **Frequenza cardiaca**, **Temperatura della pelle**, **GSR**(Risposta Galvanica della pelle) sono giustamente con una percentuale nulla di sparsity, poiché tutti i valori fisiologici riportati sono non nulli. Discorso differente deve essere effettuato per il **Delta Temp**, cioè la differenza tra temperatura della pelle e quella ambientale, in quanto riporta una percentuale tra lo 0,30% e il 2,60% riferito al paziente 588. Ciò sta ad indicare che i pazienti raramente sono soggetti a sbalzi di temperatura notevoli, il che porta a pensare che in media sono sedentari o che lavorano in ambiente con una temperatura molto simile alla temperatura corporea.

- Per quanto riguarda il **Sonno**, la percentuale più alta di sparsity è legata al paziente 563 e 575, il che sottolinea come questi ultimi abbiano dormito meno rispetto agl'altri pazienti durante il Trial Clinico.

- Per i **Passi** effettuati, i valori più alti di sparsity sono determinati dai pazienti 559, 570 e 588, ciò significa che tali pazienti sono quelli che hanno camminato di meno e probabilmente abbiano eseguito un'attività fisica molto limitata, conducendo uno stile di vita sedentario.

- Tra tutti i pazienti che utilizzano il **Bolo Insulinico**, si segnala il paziente 570, che, con un 93,80% di sparsity, minore di 5-6 punti percentuali rispetto al resto dei pazienti, è quello che ha fatto un uso stimato maggiore e/o prolungato nel tempo di insulina di tipo bolo.

- In base ai dati di utilizzo di **Insulina Basale**, è possibile osservare come i pazienti 563, 570 e 575 siano quelli che hanno tenuto per più tempo attiva la somministrazione di quest'ultima. D'altro canto, i pazienti 559, 588 e 591 hanno interrotto la somministrazione in media il 7% del tempo totale.

- Per quanto riguarda il **Delta Temp**. l'unico valore rilevante è di 2,6% di sparsity del paziente 588, valore 5-6 volte più grande rispetto agl'altri pazienti. Una possibile interpretazione di questo valore è che questo paziente abbia uno stile di vita sedentario il che è confermato dal fatto che è anche il paziente che ha effettuato meno passi.

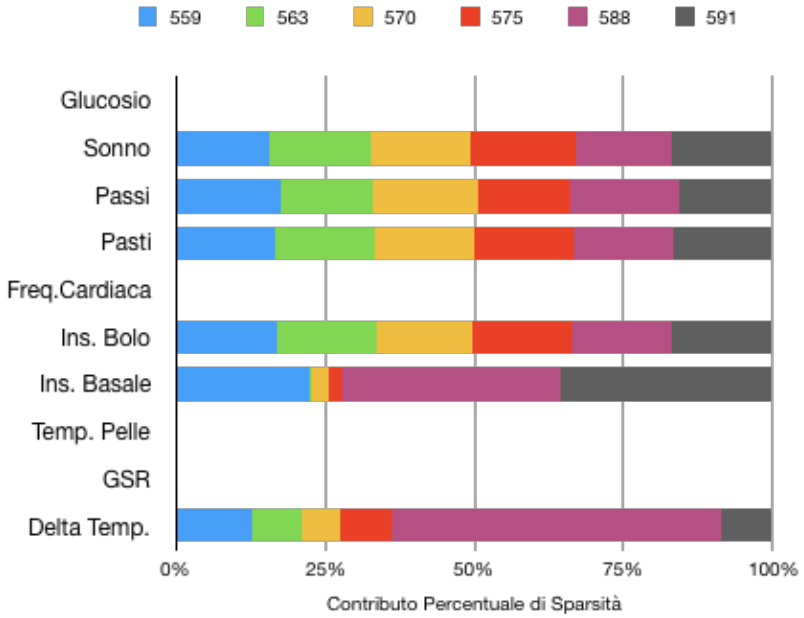


Figura 5.1: Grafico che descrive la distribuzione percentuale tra tutti i pazienti della sparsity per feature

Oltre alle feature già descritte, sono state implementate nel paragrafo 4.1.6 altre due feature collaterali, cioè la **IDR** (apporto di insulina totale) e la **MESTcalories**(conteggio calorie). Verrà analizzato il loro impatto sulla sparsity sia nel dataset che nelle singole feature. Nella tabella 5.2 sono riportati i valori ottenuti utilizzando le due nuove feature (IDR e MEST). Sono stati rimosse in seguito all'introduzione di tali feature:

- **Bolus** e **Basal**, poiché l'IDR, da Insulin Delivery Rate, normalizza e unisce i due contributi di insulina in un'unica feature.

- **Pasti** e **Passi**, perché trasformando la quantità di carboidrati ingeriti e i passi effettuati in calorie assorbite e bruciate, l'informazione derivante da quest'ultime diventa ridondante e quindi rimovibile.

	ID pazienti					
	559	**563**	**570**	**575**	**588**	**591**
Glucosio	0%	0%	0%	0%	0%	0%
Sonno	66,20%	73,60%	70,90%	74,50%	69,40%	71,90%
Freq.Cardiaca	0%	0%	0%	0%	0%	0%
Temp. Pelle	0%	0%	0%	0%	0%	0%
GSR	0%	0%	0%	0%	0%	0%
Delta Temp.	0,60%	0,40%	0,30%	0,40%	2,60%	0,40%
MEST calories	76,79%	67,24%	76,13%	67,86%	79,3%	68,69%
IDR	4,80%	0,006%	0,57%	0,46%	8,24%	7,91%

Tabella 5.2: Tabella che riporta i dati di sparsity per feature all'interno dei file dell'intero training set finale modificato con le nuove feature e diviso per paziente. IDR sta per Insulin Delivery rate, cioè l'apporto totale di insulina del paziente (bolo+basale)

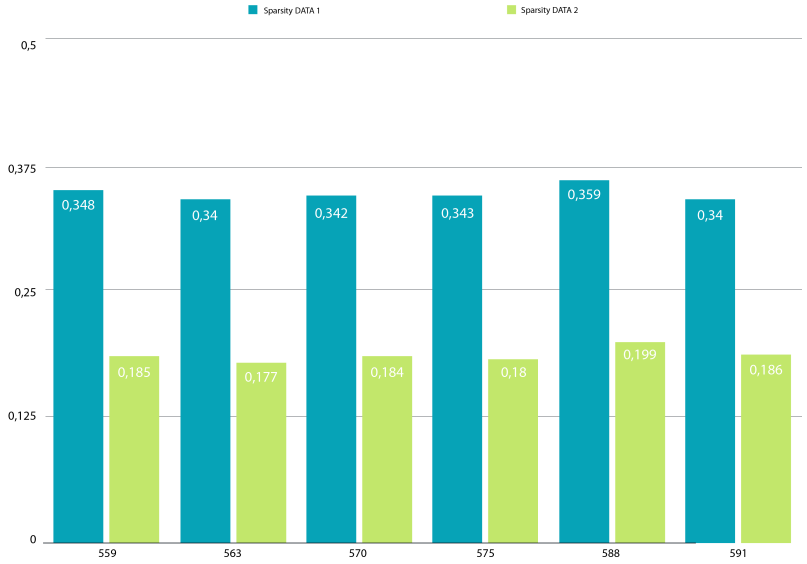


Figura 5.2: Si nota come la sparsity divisa per paziente si sia quasi dimezzata in tutti i casi del dataset 2 (in verde), cioè il dataset modificato con le nuove feature

E' chiaro anche visivamente (vedi grafico in Figura 5.2 e 5.3) che la sparsity è diminuta per via dei seguenti elementi chiave:

- Il numero di feature è calato da 10 a 8 senza modificare il carico di informazioni che aveva il dataset originale. Infatti, l'informazione insulinica è stata semplicemente aggregata in un unico "contenitore" (**IDR**), mentre l'informazione derivante dai pasti, in termini di quantità di carboidrati ingeriti, e dei passi, in termini di numeri di passi, è stata semplicemente convertita con un elemento comune ad entrambe, ovvero le calorie

- Le feature eliminate (Pasti, Passi, Bolo, Basal) portavano un peso di sparsity molto alto. Lo dimostra la nuova sparsity del dataset modificato (vedi il grafico 5.2), con valori diminuiti in media del 46,38%

- La feature **MEST calories** è caratterizzata da valori comunque alti di sparsity tra il 67% e il 79%, ma rappresenta un trade off accettabile in quanto le due feature che sostituisce, cioè Passi e Pasti, detenevano una sparsity media rispettivamente del 73% e del 99,3%.

- L'**IDR**, con una media di sparsity tra i pazienti del 3,7% abbassa notevolmente il livello di sparsity del insulina basale precedente ed abbatte quasi completamente l'incidenza del bolo insulinico, che nel dataset iniziale corrispondeva in media al 98,6%.

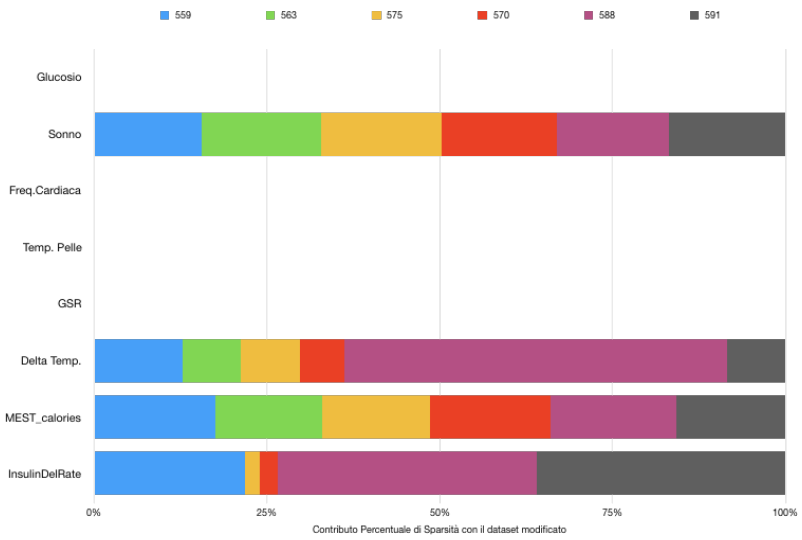


Figura 5.3: Grafico che descrive la distribuzione percentuale tra tutti i pazienti del dataset modificato della sparsity per feature

5.2 Analisi tramite l'Ambulatory Glucose Profile home made

In questa sezione si adatteranno le proprietà espresse dell'Ambulatory Glucose Profile.

Il codice pensato per tutti i pazienti, analizzerà i dati provenienti dal training set e si organizzeranno i dati sfruttando le potenzialità dei *dict* di Python, cioè delle strutture gerarchicamente ordinate secondo *chiave:valore* in grado di interagire con una grande mole di dati e facilmente accessibili dall'utilizzatore/programmatore.

L'Ambulatory Glucose Profile, noto come AGP, rappresenta uno standard per la valutazione e la visualizzazione dei dati glicemici del paziente ed è utilizzato in forma di report dal paziente diabetico, dai familiari e soprattuto dal medico per monitorare l'incidenza della malattia sul paziente e stabilire una cura personalizzata adeguata. Il report commerciale è strutturato secondo le 14 metriche presentate in sezione 1.2 e con lo standard di visualizzazione dei dati attraverso divisioni percentili.

E' uno strumento innovativo ed estremamente attuale che garantisce una visione chiara della serie glicemica nei 14 giorni di registrazione. Vista l'importanza di tale strumento, è stato deciso di programmare in Python uno script in grado di riprodurre esattamente le stesse metriche, riportando i dati glicemici su un grafico che definisce la "giornata tipo" del paziente diabetico, con le stesse suddivisioni percentili del software commerciale associato ai dispostivi di monitoraggio continuo. Nel lavoro di Tesi, è stato pensato di sfruttare l'AGP_homemade per valutare gli effetti del filtraggio sulla stessa serie glicemica di un paziente e per constatare se esistono delle differenze sostanziali riscontrabili all'interno del report.

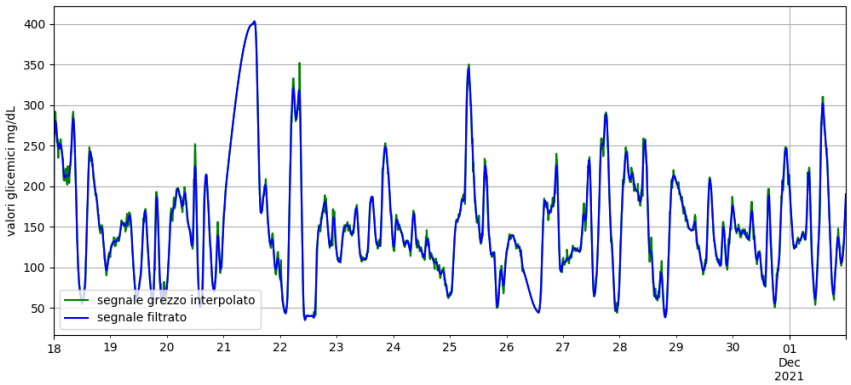


Figura 5.4: Confronto tra serie glicemiche: valori interpolati in verde vs valori filtrati in blu

Ad esempio, si consideri il paziente 575. Su di esso si vuole valutare l'effetto della regolarizzazione di Tikhonov, espressa nella sottosezione 4.1.3, sulla serie interpolata, in un periodo di 14 giorni e verrà effettuato un confronto con la serie glicemica di base. Nell'articolo di Aliberti et al [50], tale filtraggio sul segnale glicemico ha permesso di migliorare nettamente le performance, quindi l'obiettivo in questa sezione è verificare se la regolarizzazione di Tikhonov nella sua forma più semplice, ovvero come filtro di smoothing, provoca degli effetti negativi nelle metriche rispetto alla serie di base riscontrabili nel report dell'AGP_home made. Si vuole valutare infatti

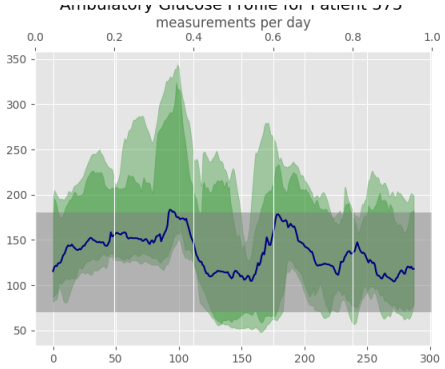


Figura 5.5: Report grafico AGP per il paziente 575 su dati non filtrati

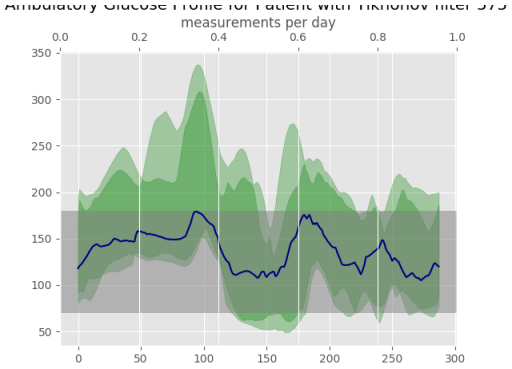


Figura 5.6: Report grafico AGP per il paziente 575 su dati filtrati

se l'effetto del filtraggio fa abbassare l'energia del segnale in termini di picchi glicemici. Le Figure 5.5 e 5.6 mostrano il report grafico AGP delle due serie temporale nei primi 14 giorni di registrazione dei dati di training del paziente 575, mentre il grafico in Figura 5.4 rappresenta l'andamento dei valori glicemici del paziente 575 rilevati tra 18 Novembre e il 2 Dicembre. Da quest'ultima immagine è possibile intravedere l'effetto regolatore del filtro del segnale in blu, che riduce l'ampiezza di alcuni picchi del segnale glicemico e di conseguenza provoca una possibilità concreta di perdita di informazioni su eventi di ipoglicemia e iperglicemia, onde per cui è importante investigare questo delicato aspetto tramite il tool.

Dai due grafici in Figura 5.6 e 5.6 si nota sin da subito una certa somiglianza, con la sola e attesa differenza sulla forma della mediana e dei valori percentili. La serie di Tikhonov, infatti, effettua uno filtraggio sui dati, che rende l'andamento glicemico più sinuoso rispetto ai dati non filtrati.
Se si prendono in considerazione le 14 metriche utili per la generazione del report, si hanno i seguenti valori:

1. glucosio medio:
    - Serie filtrata: 152.5 mg/dL
    - Serie base: 152.03 mg/dL
2. percentuale di tempo in ipoglicemia di livello 2 (< 54 mg/dL):
    - Serie filtrata: 3.67 %
    - Serie base: 3.92 %
3. percentuale di tempo in ipoglicemia di livello 1 (> 54, < 70 mg/dL):
    - Serie filtrata: 5.63 %
    - Serie base: 5.78 %
4. percentuale di tempo in regime fisiologico (> 70, < 180 mg/dL):
    - Serie filtrata: 63.24 %
    - Serie base: 62.95 %

5. percentuale di tempo in iperglicemia di livello 1 (> 180, < 250 mg/dL):
    - Serie filtrata: 18.85 %
    - Serie base: 18.82 %
6. percentuale di tempo in iperglicemia di livello 2 (> 250 mg/dL):
    - Serie filtrata: 8.61 %
    - Serie base: 8.53 %
7. variabilità glicemica:
    - Serie filtrata: CV = 45.12; SD = 68.81
    - Serie base: CV = 45.57; SD = 69.28
8. eA1c:
    - Serie filtrata: 6.96 %
    - Serie base: 6.95 %
9. dati glicemici segnalati in tre intervalli temporali (veglia, sonno, 24 ore)
10. Sono stati analizzati 14 giorni consecutivi;
11. sufficienza raggiunta dei dati utilizzabili in entrambe le serie. La serie base prima di essere interpolata possedeva, nei primi 14 giorni di registrazione, il 90,4% di dati glicemici veri, corrispondenti a 386 valori non validi su 4032 campioni totali;
12. numero di episodi medi al giorno di iperglicemia o ipoglicemia, usando una definizione standard di episodio, inteso come numero di eventi iperglicemici o ipoglicemici consecutivi per una durata di almeno 15 minuti. Le etichette vHYPO, HYPO, HYPER, vHYPER e vvHYPER corrispondono a ipoglicemia grave, ipoglicemia, iperglicemia, iperglicemia elevata e iperglicemia grave ed hanno i seguenti valori di episodi medi al giorno:
    - Serie filtrata: vHYPO = 0.5, HYPO = 1.43, HYPER = 4.07, vHYPER = 0.64, vvHYPER = 2.5
    - Serie base: vHYPO = 0.57, HYPO = 1.57, HYPER = 4.21, vHYPER = 0.71, vvHYPER = 3
13. AUC (Area Under Curve):
    - Serie filtrata: AUC su 24 ore: 138, AUC durante il sonno: 144, AUC durante la veglia: 135
    - Serie base:AUC su 24 ore: 137, AUC durante il sonno: 144, AUC durante la veglia: 134
14. rischio di ipoglicemia e iperglicemia tramite gli indici raccomandati: LBGI per la bassa glicemia e HBGI per l'alta glicemia:
    - Serie filtrata: LBGI: 2.10, HBGI: 6.73

- Serie base: LBGI: 2.18, HBGI: 6.72

Anche dal punto di vista empirico le metriche applicate su entrambe le serie non presentano differenze significative, con ciò si dimostra che l'applicazione del filtraggio di Tikhonov sui dati glicemici del paziente 575 non comporta una perdita della qualità delle metriche presentate nel report AGP e di conseguenza non genera dei rischi clinici per la valutazione dello stato di salute del paziente.

5.3 Le metriche di valutazione per le reti neurali

Ispirandosi all'articolo di Aliberti et al. [50], al fine di valutare l'accuratezza nella predizione del livello glicemico, sono state utilizzate una serie di metriche che permettono una visione analitica e clinica chiara delle inferenze dei modelli neurali.

5.3.1 Criteri per la valutazione analitica delle predizioni

Le metriche di valenza analitica sono utilizzate per valutare il grado di similarità tra il valore osservato e il valore predetto. Sono elencate in seguito:

- **RMSE**, il *Root Mean Squared Error* è sicuramente la metrica più utilizzata nel caso di valutazioni tra i valori osservati e i valori predetti da una rete neurale. Rappresenta infatti l'errore che tra la predizione e il valore effettivo, definito dalla formula:

$$RMSE = \sqrt{\frac{1}{n}\sum_{t=1}^{n} e_t^2} \tag{5.1}$$

  dove e_t è la differenza tra valore vero e valore predetto per ogni n-esimo campione

- **R^2**, rappresenta la correlazione tra il valore predetto e il valore osservato con un valore compreso tra 0 (non c'è correlazione) e 1 (massima correlazione). La funzione implementata si basa sulla formula matematica seguente:

$$R^2(y, \hat{y}) = 1 - \frac{\sum_{i=0}^{n-1}(y_i - \hat{y}_i)^2}{\sum_{i=0}^{n-1}(y_i - \hat{y})^2} \tag{5.2}$$

  dove n è il numero di campioni e la $\hat{y}$ al denominatore è uguale a :

$$\hat{y} = \frac{1}{n}\sum_{i=0}^{n-1} y_i \tag{5.3}$$

- **Time Lag**, viene calcolato come il massimo della funzione di cross-correlazione tra il segnale e sua predizione, utile per valutare l'allineamento tra i valori predetti e quelli reali

- **MAD**, è la *Mean Absolute Difference*, cioè la media degli errori assoluti tra i valori osservati e quelli predetti. Minore è tale valore, maggiore sarà l'accuratezza della previsione

$$MAD = \frac{1}{n}\sum_{i=0}^{n-1} |y_i - \hat{y}| \tag{5.4}$$

- **FIT**, è sempre un valore espresso in percentuale tra 0 e 100, dove una percentuale più vicina al 100 indica una migliore accuratezza della predizione. E' espresso dalla seguente formula:

$$FIT = \left(1 - \frac{\sum_{i=0}^{n-1}(y_i - \hat{y}_i)^2}{\sum_{i=0}^{n-1}(y_i - \hat{y})^2}\right) * 100 \tag{5.5}$$

5.3.2 Criteri per la valutazione clinica delle predizioni

La metrica di valenza clinica utilizzata è invece la **Clarke Error Grid**. Questo tipo di valutazione è importante perché quantifica l'impatto clinico che le previsioni possono generare, suddividendo i valori all'interno di macro aree definite come nel grafico in figura 5.7.

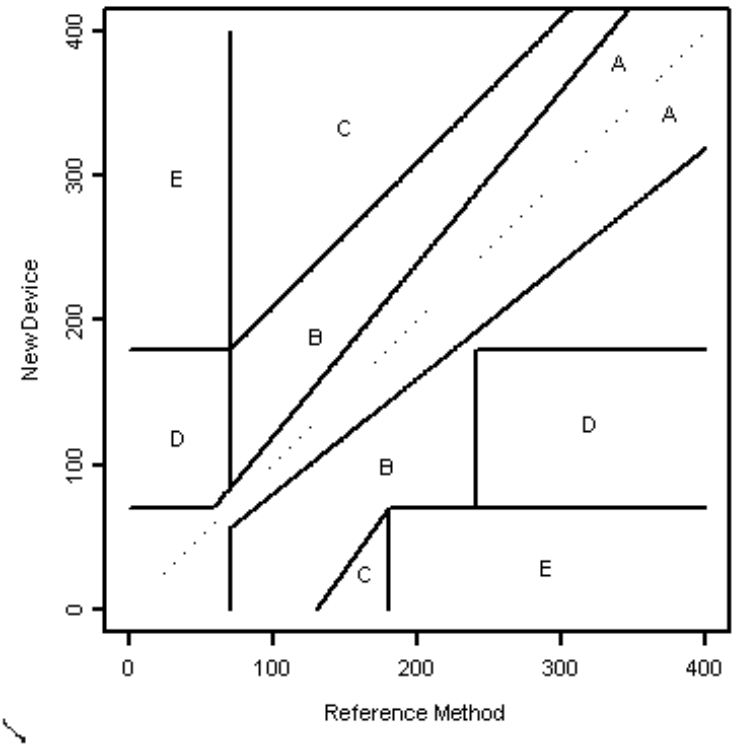


Figura 5.7: Schema rappresentativo della Clarke Error Grid con suddivisione in aree A, B, C, D, E

La aree identificate sono suddivise come segue:

- **zona A**, dove sono classificate le predizioni con una differenza inferiore al 20% rispetto al valore reale. Idealmente è la regione che dovrebbe essere più popolata per un'ottima valutazione della predizione
- **zona B**, dove sono classificati i valori con differenze superiori al 20% ma che portano a valutazioni ancora clinicamente accettabili delle predizioni
- **zona C**, dove risiedono i campioni predetti non considerati importanti nella predizione per una valutazione clinica
- **zona D**, in cui vengono inserite le errate predizioni che potrebbero indurre a errori nelle valutazioni cliniche con conseguenze pericolose per la salute del paziente, poiché non si identificano i casi di iperglicemia e ipoglicemia
- **zona E**, dove la predizioni sono così errate che si confondono casi ipoglicemici con iperglicemici e viceversa.

5.4 Analisi prestazioni delle reti neurali

Il principale scopo di questa Tesi è cercare di ottenere la conformazione migliore per garantire un modello, che sia in grado di generare delle predizioni quanto più accurate possibile. La predizione dal modello neurale è ottenuta sfruttando la funzione **predict**, importata dalle librerie Keras[65]. Tale funzione riceve in input la sequenza X da predire con il rispettivo target Y. L'output della funzione è la sequenza temporale predetta con la stessa lunghezza del segnale glicemico di input. Al fine di ottenere una valutazione basata su dati reali, sono stati prelevati dai dataset di testing di tutti i pazienti dei segmenti *puri*, cioè dei segmenti con letture glicemiche che non hanno subito alcun tipo di preprocessing al netto di gap temporali, cioè senza *NaN* all'interno.

Si è impostato il lavoro di analisi delle prestazioni seguendo le fasi:

1. Creare dei segmenti puri di testing
2. Determinare i modelli e le feature del dataset all'interno di scenari
3. Ricerca dei parametri dei modelli e del dataset migliori utilizzando il paziente 575 come riferimento
4. Applicazione parametri ottimali del paziente 575 e valutazione preliminare globale delle predizioni dei pazienti rimanenti.
5. In base all'analisi delle metriche, si sceglie lo scenario migliore, che include il dataset e il modello neurale che garantiscono le performance migliori
6. Il modello e il dataset scelti sono ottimizzati nuovamente per identificare, se esiste, il miglior assetto di parametri

5.4.1 Definizione dei segmenti di testing

La creazione dei segmenti di testing è il primo e fondamentale step per ottenere le predizioni del glucosio. A differenza di quanto esposto nella maggior parte degli articoli presentati nello capitolo 3, si è scelto di valutare la qualità delle inferenze generate dai modelli attraverso la scelta di specifici segmenti, anziché dell'intero dataset di testing messo a disposizione dall'OhioT1DM dataset [18]. Il motivo di tale scelta si basa sul concetto fondamentale che l'attendibilità della combinazione tra dataset e modello, deve essere valutata su dati contigui che non hanno subito alcun tipo di filtraggio e/o pre-processing.
Il segmento di testing viene definito come una raccolta di dati glicemici consecutivi con indici validi anche per altre feature del test set. Si crea un nuovo segmento tra valori indefiniti, presenti nel DataFrame come *NaN*, non consecutivi di glucosio con lunghezza minima corrispondente al numero di input del modello per ottenere la predizione. Dato che per il paziente 588, sono stati rilevati solamente due segmenti contigui, si è deciso di utilizzare per tutti i pazienti i primi due segmenti trovati.
Nell'elenco sottostante sono presentate tutte le lunghezze in campioni di ciascuno dei due segmenti per paziente che saranno utilizzati per generare le predizioni:

- **paziente 559**. Primo segmento: 206 , secondo segmento: 282

- **paziente 563**. Primo segmento: 864, secondo segmento: 1175
- **paziente 570**. Primo segmento: 80, secondo segmento: 275
- **paziente 575**. Primo segmento: 662, secondo segmento: 143
- **paziente 588**. Primo segmento: 402, secondo segmento: 1649
- **paziente 591**. Primo segmento: 388, secondo segmento: 1361

5.4.2 Gli scenari analizzati

La ricerca del modello ottimale per le predizioni utilizzando i dati dei pazienti dell'OhioT1DM inizia con la definizione della coppia di dataset/rete neurale a partire da una conformazione già esistente. In questo caso si utilizzerà la rete neurale presente in [56] con i seguenti elementi caratteristici:

- Dataset: 4 feature selezionate [Glucosio, Pasti, IDR, HR] corrispondenti rispettivamente ai valori di glucosio, della quantità di carboidrati ingeriti, all'apporto totale di insulina e alla frequenza cardiaca.
- Modello neurale: rete VANILLA caratterizzata da 3 layer in serie LSTM e con 50 neuroni per layer. La topologia della rete rispetta i parametri comuni presentati nel paragrafo 4.3.2

Oltre a questa tipologia di dataset e di modello, si è deciso di ampliare la gamma di combinazioni da analizzare considerando gli effetti che i seguenti fattori possano contribuire nella qualità della predizione: filtro di Tikhonov sulla serie temporale del glucosio; utilizzo della feature ideata *MEST_calories*; il layer DCAM e infine il costrutto neurale dell'Encoder-Decoder.
Sono stati creati 4 scenari, in ognuno dei quali vengono implementati i tre modelli neurali scelti (VANILLA, Encoder-Decoder LSTM, Encoder-Decoder CNN LSTM), aventi le seguenti caratteristiche:

1. 4 feature selezionate [Glucosio, Pasti, IDR, HR] senza aver applicato il filtro di Tikhonov sulla serie temporale del glucosio. Il resto delle feature è invariato. Questo scenario per semplicità è stato denominato **STANDARD NO TIKO**
2. 4 feature selezionate [Glucosio, Pasti, IDR, HR] con applicato il filtro di Tikhonov sulla serie temporale del glucosio. Il resto delle feature è invariato. Questo scenario invece è lo **STANDARD TIKO**
3. 4 feature selezionate [Glucosio, Pasti, IDR, HR] con applicato il filtro di Tikhonov sulla serie temporale del glucosio e avente come primo layer della catena neurale il DCAM, di cui si è già parlato nella sezione 4.1.5. Questo caso è stato denominato come **STANDARD TIKO DCAM**
4. 4 feature selezionate [Glucosio, MEST_calories, IDR, HR] con applicato il filtro di Tikhonov sulla serie temporale del glucosio. La feature Pasti è stata sostituita con la nuova feature MEST_calories creata nella sezione 4.1.6. Il nome dello scenario di questo ultimo caso si è ispirato al nome della nuova feature, cioè **MEST DATASET**

La ricerca dei parametri, che caratterizzano lo scenario migliore, tra i 4 precedentemente elencati, inizia con un'ottimizzazione preliminare con i seguenti valori, utilizzando solo il paziente 575 come modello su cui ricercare le performance ottimali nei vari modelli al variare di pochi parametri:

- **modello VANILLA**: 50 neuroni nel layer LSTM e ottimizzato solo su valori di BatchSize di 64 e 512 campioni.
- **modello ENC-DEC LSTM**: 50 neuroni nei due layer LSTM e ottimizzato solo su valori di BatchSize di 64 e 512 campioni.
- **modello ENC-DEC CNN LSTM**: si è deciso di effettuare una ricerca ristretta di parametri per il numero di filtri convoluzionali (16 e 64) dell'encoder, per il numero di celle (40, 80) all'interno layer LSTM del decoder e su valori di BatchSize di 64 e 512 campioni.

.

Per ogni modello corrispondente, sono stati valutati tutti i pazienti per tutti gli scenari possibili, estromettendo il paziente 575 dal computo metrico per non rendere inefficaci i risultati e contemporaneamente valutare la robustezza del modello per gli altri pazienti. L'esito dell'ottimizzazione dei parametri trovati sul singolo paziente, ha permesso di definire il seguente assetto:

- modello VANILLA: 50 neuroni nel layer LSTM e BatchSize di 512 campioni.
- modello ENC-DEC LSTM: 50 neuroni nei due layer LSTM e BatchSize di 512 campioni.
- modello ENC-DEC CNN LSTM: 16 filtri convoluzionali dell'encoder, 40 celle nel layer LSTM del decoder e BatchSize di 512 campioni.

Nella pagina seguente viene riportata la Tabella 5.3, che mostra i risultati riguardanti le performance delle predizioni secondo le metriche: RMSE, Time Lag, R^2, MAD, FIT e Clarke Error Grid(riportata solamente la zona A). I dati mostrati fanno riferimento al modello Encoder-Decoder LSTM dello scenario STANDARD NO TIKO. Si nota come i valori di RMSE in tutti gli orizzonti di previsione siano molto competitivi (23.48 ± 4.92 su un orizzonte di previsione di 30 minuti) con altri modelli neurali presentati nel capitolo 3, inoltre c'è ancora da considerare che è stata eseguita una metodologia di testing basata sulla media tra valori provenienti da due segmenti su cui non è stato effettuato alcun tipo di rimodellamento sui dati, mentre la maggior parte degli articoli analizzati sullo stesso tema esegue il medesimo pre-processing sia sul dataset di training che sull'intero dataset di testing.

Tabella 5.3: La tabella illustra i risultati ottenuti divisi per metriche e per orizzonti di previsione (PH) dall'ottimizzazione preliminare dello scenario STANDARD NO TIKO, il migliore tra gli scenari analizzati

Enc-Dec LSTM	***len_seg***	*206*	*282*	*864*	*1175*	*80*	*275*	*402*	*1649*	*388*	*1361*		
	PH\id_paz	**559**		**563**		**570**		**588**		**591**		**MEDIA**	**SD**
RMSE (mg/dL)	**6**	24,39	20,91	18,67	21,80	27,29	18,21	25,16	20,27	33,72	22,51	23,38	4,92
	9	29,49	25,32	27,68	29,77	29,44	24,57	32,57	27,67	38,52	28,39	29,45	4,17
	12	32,09	30,11	34,69	36,14	29,56	29,96	37,97	32,85	42,28	32,73	33,96	4,27
	18	37,35	38,50	43,09	47,03	38,08	38,92	45,73	40,70	48,55	37,87	41,99	4,24
Time Lag (steps)	**6**	0,00	4,00	3,00	4,00	5,00	1,00	5,00	5,00	6,00	6,00	3,67	2,00
	9	0,00	7,00	6,00	6,00	9,00	2,00	7,00	8,00	9,00	9,00	6,00	3,08
	12	0,00	9,00	8,00	8,00	12,00	5,00	10,00	11,00	12,00	12,00	8,33	3,84
	18	0,00	13,00	13,00	13,00	0,00	13,00	16,00	16,00	18,00	17,00	11,33	6,67
R2	**6**	0,89	0,72	0,81	0,78	-11,64	0,92	0,71	0,80	0,63	0,79	-0,60	4,14
	9	0,83	0,60	0,59	0,59	-57,67	0,85	0,52	0,63	0,53	0,66	-5,84	19,44
	12	0,80	0,43	0,35	0,40	-64,58	0,75	0,36	0,47	0,44	0,55	-6,73	21,69
	18	0,73	0,07	-0,01	-0,02	-140,64	0,54	0,07	0,20	0,27	0,40	-15,42	46,96
MAD	**6**	7,36	8,64	8,42	9,20	7,97	6,53	11,29	8,59	19,81	11,81	9,76	3,99
	9	9,69	11,57	12,36	12,71	8,76	8,92	15,08	11,55	22,72	15,05	12,60	4,30
	12	10,68	14,58	15,54	15,57	9,36	10,94	17,94	13,74	24,81	17,66	14,80	4,66
	18	13,37	19,07	19,70	20,58	12,60	14,76	22,32	17,10	28,64	20,92	18,68	5,00
FIT	**6**	66,33	47,18	56,65	53,28	-255,58	71,76	46,55	55,11	39,52	54,02	20,09	103,85
	9	59,36	36,52	35,71	36,12	-665,96	60,64	31,06	38,80	31,51	42,04	-37,36	235,99
	12	55,76	24,70	19,40	22,38	-709,79	50,46	19,77	27,44	25,36	33,24	-51,61	247,17
	18	47,58	3,72	-0,44	-0,86	-1090,12	32,12	3,38	10,31	14,51	22,71	-108,87	368,33
Zona A - CEG (%)	**6**	86,47	91,93	92,94	89,76	100,00	93,72	81,05	90,34	57,19	78,05	87,04	12,33
	9	84,96	86,55	77,68	80,59	100,00	87,89	67,06	80,74	51,76	68,76	79,69	13,69
	12	85,71	77,13	69,23	72,57	100,00	79,37	60,64	75,65	47,28	62,55	74,18	14,88
	18	78,95	54,71	57,38	57,79	100,00	72,20	47,23	67,07	43,45	54,26	64,31	17,56

La scelta dello scenario migliore

La stessa tipologia di tabella valida per il modello neurale Encoder-Decoder LSTM dello scenario STANDARD NO TIKO, è stata creata anche per gli altri due modelli (VANILLA e Encoder-Decoder CNN LSTM) e per i rimanenti 3 scenari(STANDARD TIKO, STANDARD TIKO DCAM e MEST DATASET). I risultati di tali scenari, divisi per ciascun modello analizzato, non sono stati riportati per non appesantire la grafica della Tesi, ma vengono comunque trascritti i valori medi delle metriche più importanti per capire quale scenario e quale modello garantisce le migliori prestazioni.
Nel caso specifico della valutazione per la scelta dello scenario migliore, sono state considerate le due metriche che hanno il maggior peso analitico e clinico nella valutazione di una predizione di una serie temporale glicemia: RMSE e Clarke Error Grid. Tale decisione risiede nella natura di tali metriche, infatti mentre l'RMSE quantifica l'errore nella predizione, la Clarke Error Grid valuta l'impatto che tale errore porterebbe alla salute del paziente.

Nella pagina seguente, in tabella 5.4, sono riportati i risultati della comparazione di ogni singolo modello neurale nei 4 scenari, analizzati secondo le medie delle metriche sui due segmenti di testing su tutti i pazienti (escluso il 575).

Da un'attenta analisi delle colonne VAN, LSTM e CNN, che corrispondono rispettivamente ai 3 modelli VANILLA, Encoder-Decoder LSTM e Encoder-Decoder CNN-LSTM, si può osservare come il secondo scenario, ovvero lo STANDARD NO TIKO, offra le migliori performance globali grazie ai risultati ottenuti dalle due architetture Encoder-Decoder, poiché hanno ottenuto i risultati più bassi di RMSE e quelli più alti nelle percentuali espresse della zona A della Clarke Error Grid.

Inoltre, all'interno di tale scenario è possibile identificare il modello neurale Encoder-Decoder LSTM (colonna LSTM della Tabella 5.4) come il modello le cui previsioni sono effettuate con più elevata accuratezza e rigore clinico per tutti gli orizzonti di previsione.

La fase di ottimizzazione preliminare ha definito il miglior scenario, il miglior dataset e il miglior modello neurale che, in base ai dati forniti, riesca a predire meglio la serie glicemica.
La fase successiva, invece, sarà caratterizzata da un'ottimizzazione più ampia della topologia del modello neurale vincitore, per esplorare se nuovi set di parametri possano eventualmente migliorarne le prestazioni globali.

Tabella 5.4: Tabella con i risultati del confronto tra i 4 diversi scenari. I risultati riportati sono valori medi delle metriche sui due segmenti di testing su tutti i pazienti (escluso il 575) e sono privi di deviazione standard

	Scenari	*STAND. TIKO*			*STAND. NO TIKO*			*STAND. TIKO DCAM*			*MEST DATASET*		
	PH\Mod	**VAN**	**LSTM**	**CNN**	**VAN**	**LSTM**	**CNN**	**VAN**	**LSTM**	**CNN**	**VAN**	**LSTM**	**CNN**
RMSE (mg/dL)	6	49,55	28,21	101,93	41,94	23,38	27,32	53,04	50,25	139,95	118,10	42,42	128,30
	9	54,55	35,15	148,48	41,40	29,45	31,52	53,15	64,12	191,52	150,65	48,04	147,02
	12	57,97	39,49	292,32	41,34	33,96	35,35	51,60	71,89	284,70	185,17	60,23	239,52
	18	74,74	46,18	633,14	43,05	41,99	38,49	54,64	79,63	449,83	262,61	61,83	968,47
Zona A-CEG(%)	6	59,01	81,98	51,13	61,74	87,04	82,20	49,40	69,83	32,31	52,84	72,56	31,03
	9	53,91	73,95	44,83	63,84	79,69	77,39	48,94	61,78	26,91	43,70	66,35	26,07
	12	50,65	68,17	41,28	64,86	74,18	73,29	51,72	57,52	23,70	40,86	61,74	20,84
	18	45,47	60,77	37,24	62,11	64,31	59,13	47,32	50,63	19,05	42,78	54,32	18,46

5.4.3 Ottimizzazione finale

Come dimostrato poc'anzi, lo scenario migliore tra quelli proposti è rappresentato dallo **STANDARD NO TIKO**, cioè quello in cui sono presenti le feature relative alle serie temporali del glucosio, dei pasti, dell'apporto totale di insulina e della frequenza cardiaca, senza effettuare la regolarizzazione di Tikhonov sulla serie temporale glicemica. Dallo scenario migliore è stato possibile distinguere anche il modello di rete neurale che garantisce una predizione significativa e paragonabile allo stato dell'arte, ovvero l'architettura Encoder-Decoder avente in entrambe le componenti un layer LSTM. Si è deciso, quindi, di impostare un'ulteriore e più ampia ottimizzazione, effettuata a partire sempre dal paziente 575, per valutare se esiste un settaggio che possa migliorare i risultati preliminari globali ottenuti precedentemente. La Tabella 5.5 racchiude tutte le combinazioni delle terna di valori per le variabili *CELL1*, *CELL2* e *BATCHSIZE*, che corrispondono rispettivamente al numero di celle dei layer LSTM sia nell'encoder (CELL1) che nel decoder (CELL2) e alla lunghezza della batch nella fase di allenamento.

CELL1	20	80	120	150	180
CELL2	20	80	120	150	180
BATCH SIZE	32	128	512		

Tabella 5.5: Griglia da cui verranno ottimizzate in base ai valori di metriche delle previsioni ottenute, la terna finale CELL1, CELL2 e BATCH SIZE

La ricerca della migliore terna è stata effettuata allenando 75 reti neurali diverse ottenute dalla combinazione delle variabili presenti nella griglia in Tabella 5.5.

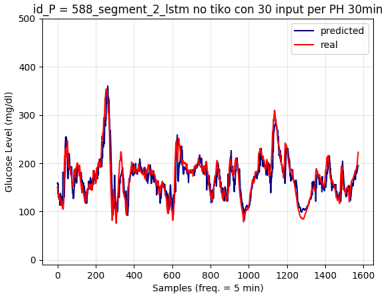
Tra i casi analizzati e valutati, solamente 6 reti neurali con le terne di parametri (20, 20, 512), (20, 80, 512), (80, 20, 512), (80, 80, 512), (150, 80, 32) e (120, 80, 32) sono riuscite a migliorare le metriche sul singolo paziente 575, ma nessuna di esse sorprendentemente è riuscita a migliorare i risultati globali sugli altri pazienti ottenuti dall'ottimizzazione preliminare precedente. Il motivo di tale risultato potrebbe essere legato a numerosi fattori, ma si stima che l'elemento che abbia determinato il fallimento dell'ottimizzazione sulle 75 reti neurali sia legato alla variabilità interpaziente, che impedisce a tali modelli di essere uniformemente valutati con parametri topologici comuni. Una possibile soluzione potrebbe ricercarsi nella modifica dei dati di training, in modo da unire ai dati di un singolo paziente anche altre letture glicemiche di pazienti diversi, ma non è stato ulteriormente investigato questo aspetto. D'altronde, ciò non deve sorprendere più di tanto poiché anche nell'articolo di Xie et al. [56], preso come riferimento per la definizione degli scenari, sono stati scelti 50 neuroni per il layer LSTM, sebbene con un'architettura neurale e una metodologia di testing totalmente diversa.

Di conseguenza, i parametri ottimali rimangono quelli scelti precedentemente e vengono nuovamente riportati:

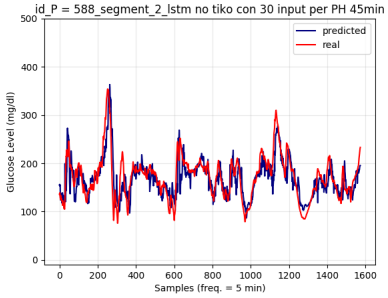
- DATASET: STANDARD NO TIKO con le feature di glucosio, pasti, insulina e frequenza cardiaca
- MODELLO: Encoder-Decoder LSTM
- Numero di celle nel layer LSTM dell'encoder: 50

- Numero di celle nel layer LSTM del decoder: 50
- Grandezza della batch nella fase di allenamento: 512

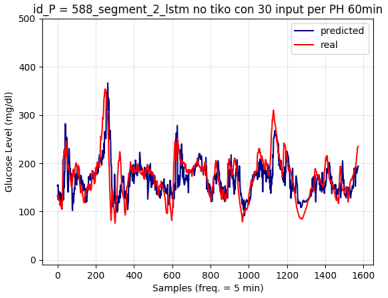
Vengono presentati alcuni esempi di grafici con i valori reali e predetti, seguiti dalle rispettive Clarke Error Grid. Le figure 5.8a, 5.8b, 5.8c e 5.8d rappresentano il grafico di comparazione tra il segmento 2 di testing del paziente 588 in rosso e la sequenza predetta in blu per i 4 orizzonti di previsione. Al di sotto, nello stesso ordine, vengono presentate le corrispondenti Clarke Error Grid, cioè dei grafici di dispersione (o scatter plot), dove vengono riportate due variabili (valore reale e valore predetto) nelle diverse zone di interesse A, B, C, D, E del diagramma cartesiano. Nel primo set di figure proposte del paziente 588, il cui secondo segmento è quello con il maggior numero di campioni, si nota la bontà della predizione, confermata dai valori di RMSE pari a 20 mg/dL, 27 mg/dL, 32 mg/dL e 40 mg/dL per una previsione a 30, 45, 60 e 90 minuti. Anche il secondo set di figure (5.9a, 5.9b, 5.9c, 5.9d) conferma i risultati precedenti, dove la concentrazione dei punti è quasi interamente nella zona centrale, ovvero la zona A, della Clarke Error Grid. In particolare le percentuali della zona A sono del 90 %, 81 %, 76 % e 67 % per gli stessi orizzonti di previsione valutati con l'RMSE.



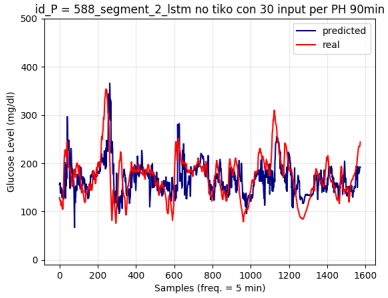
(a) Predizione per PH = 30



(b) Predizione per PH = 45

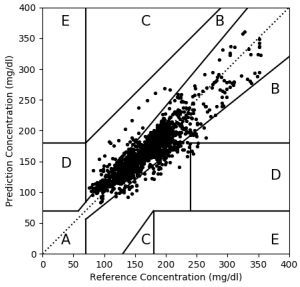


(c) Predizione per PH = 60

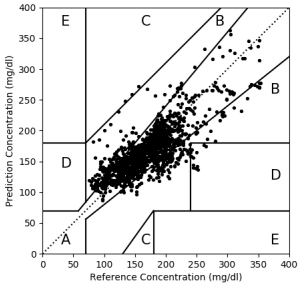


(d) Predizione per PH = 90

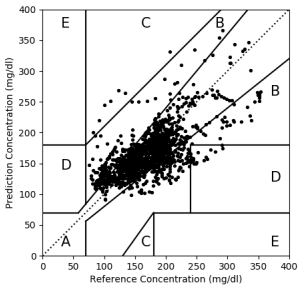
Figura 5.8: Set di immagini che riporta la predizione effettuata sul segmento 2 del paziente 588 lungo 1649 campioni per tutti gli orizzonti di previsione



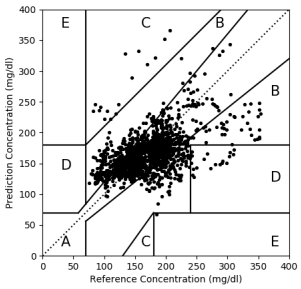
(a) Clarke Error Grid per PH = 30 minuti



(b) Clarke Error Grid per PH = 45 minuti



(c) Clarke Error Grid per PH = 60 minuti



(d) Clarke Error Grid per PH = 90 minuti

Figura 5.9: Set di immagini che riporta le Clarke Error Grid del segmento 2 del paziente 588 lungo 1649 campioni per tutti gli orizzonti di previsione

5.5 Tempi e grandezze computazionali

L'intero ciclo di allenamento delle reti neurali implementate ha richiesto il supporto computazionale dell'HPC[66] del Politecnico di Torino, le cui specifiche tecniche sono riportante in Tabella 5.6

Tabella 5.6: Elenco delle specifiche tecniche Casper nel progetto HPC

Architecture	Linux Shared-Memory Cluster
Node Interconnect	Infiniband DDR 20 Gb/s
Service Network	Ethernet 1 Gb/s
CPU Model	2x Opteron 6276/6376 (Bulldozer) 2.3 GHz 16 cores
Performance	4.360 TFLOPS
Power Consumption	7 kW
Computing Cores	512
Number of Nodes	16
Total RAM Memory	2.0 TB DDR3 REGISTERED ECC
OS	Centos 7.4 - OpenHPC 1.3.4
Scheduler	SLURM 17.11

Sono annotati i tempi computazionali di allenamento nelle fasi di:

- Ottimizzazione preliminare sul paziente 575: circa 5 ore
- Ottimizzazione finale sul paziente 575: circa 75 ore

- Tempo medio per allenare una singola rete neurale: circa 20 minuti

La fase di testing tramite la funzione **predict** della libreria Keras ha richiesto in media 10 secondi per segmento(di lunghezza variabile tra i pazienti), quindi 20 secondi per paziente e 120 secondi in totale per tutti i pazienti. Se si considera anche la fase di generazione delle metriche e salvataggio dei grafici inerenti alla predizione, la tempistica si alza di circa 10 secondi per paziente, arrivando quindi ad un periodo totale di 180 secondi, ovvero 3 minuti. A differenza della fase di allenamento e ottimizzazione per la ricerca dei parametri ottimali effettuata tramite il Cluster, messo a disposizione dal dipartimento DAUIN del Politecnico di Torino, la fase di testing è stata effettuata interamente su un MacBook Pro (Retina, 13-inch, Early 2015) con processore 2,7 GHz Intel Core i5, memoria 8 GB 1867 MHz DDR3 e Sistema Operativo macOS Mojave versione 10.14.6.

I modelli ottenuti dell'Encoder-Decoder LSTM dello scenario vincitore hanno un peso molto contenuto, infatti pesano in media 500 KB, probabilmente per il ridotto numero di celle presente nei layer LSTM. Questo aspetto è da tenere altamente in considerazione se si volesse trasferire il modello in un contesto applicativo mobile.

Capitolo 6

Conclusioni

6.1 Conclusioni

L'idea che sta alla base del lavoro di Tesi è quello di investigare e costruire, tramite tecniche di Machine Learning, dei modelli di reti neurali in grado di predire con una certa accuratezza il livello glicemico del sangue di pazienti affetti da diabete mellito di tipo 1. La ricerca di una predizione migliore può essere di vitale importanza per la salute del paziente diabetico, costretto ad affrontare numerose complicanze che possono diventare letali se non gestite tempestivamente. La Tesi si è sviluppata seguendo un triplice obiettivo: disporre di un dataset opportuno, elaborare algoritmi predittivi e fornire uno standard di visualizzazionedi dati glicemici.

Perciò, si è deciso di implementare tale progetto affrontando tutte le problematiche che ruotano attorno alla costruzione di un modello di reti neurali, elaborando un metodo che mette insieme le analisi, gli approcci e gli strumenti che permettono la costruzione di un dataset opportuno, dall'estrazione dei file XML dell'OhioT1DM dataset alla preparazione delle batch per l'allenamento e il testing delle reti stesse. Si è creato un tool capace di emulare e classificare i dati glicemici secondo standard clinici ufficiali e si è dimostrato che l'effetto di un particolare tipo di filtraggio sulla serie glicemica di un paziente non produce conseguenze nocive significative nelle metriche del report. Sono state implementate anche delle strade alternative per cercare di contrastare la presenza di una tipologia di dati dannosa (gli *sparse data*), che potenzialmente riduce le prestazioni del modello a base neurale, sfruttando sia le potenzialità del layer DCAM sia la creazione di feature fittizie, come la MEST_calories, ma entrambi gli approcci non hanno portato miglioramenti significativi delle prestazioni a livello globale.

I risultati presentati nel paragrafo 5.4.3 mostrano la miglior architettura in grado di predire la serie temporale glicemica con una buona accuratezza nei diversi orizzonti di previsione per l'intero set di pazienti provenienti dal dataset analizzato.

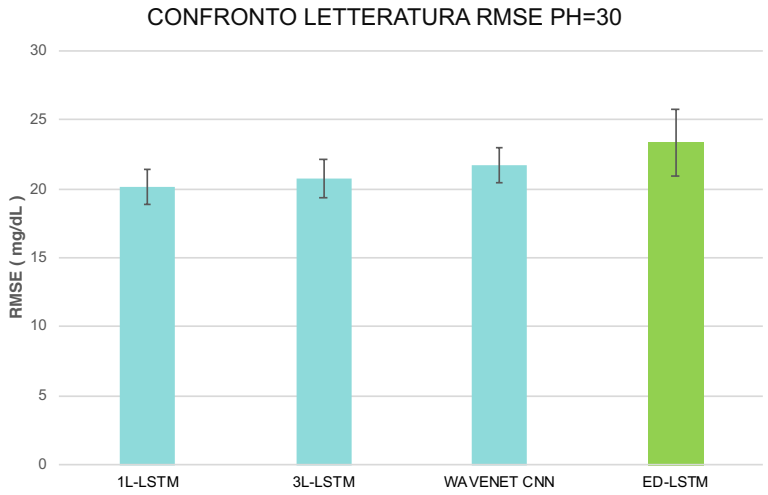


Figura 6.1: In questo grafico vengono comparati i valori di RMSE per un PH=30 min del modello con quelli di altri modelli allo stato dell'arte che utilizzano lo stesso dataset.

Se si paragonano le prestazioni del modello implementato con altri modelli presenti allo stato dell'arte (vedi grafico in Figura 6.1), si nota subito che, in termini di RMSE e per un orizzonte di previsione di 30 minuti, non esistono differenze statisticamente significative tra i modelli presentati nella sezione 3.3 e il modello risultante dall'ottimizzazione (colonna in verde). Da ciò emergono due concetti importanti:

- il modello implementato ha delle performance che lo allineano allo stato dell'arte
- la dimostrazione che è il dataset il vero effetto limitante in un sistema di Machine Learning, poiché tutti i lavori di riferimento, compreso quello di Tesi, non hanno ricevuto un supporto adeguato per il training.

6.2 Le problematiche da superare

La ricerca sulla predizione del livello glicemico ematico è iniziata ad essere importante con l'avvento di dispositivi CGM, approvati per la prima volta dalla US Food and Drug Administration (FDA) nel 1999 e in grado di misurare per ogni intervallo temporale il livello di glucosio nel sangue. Al di là dei metodi neurali e classici che sono stati implementati nelle ultime due decadi, l'elemento chiave che ancora risulta essere limitante è senza dubbio il set di dati che viene somministrato al modello.

Le sorti di un modello di apprendimento automatico sono molto dipendenti dalla quantità e dalla qualità del dataset che si utilizza per allenare e testare. L'OhioT1DM dataset presenta delle caratteristiche oggettive in netto contrasto tra loro che ne limitano le potenzialità. I dati raccolti infatti sono molto eterogenei tra loro e garantiscono una raccolta quasi completa dello stato del paziente affetto da diabete di tipo 1, ma d'altro canto coprono un arco temporale di registrazione di solamente due settimane, molto limitato se si è nell'ambito del Machine Learning. Un altro limite è sicuramente l'alto grado di asincronia dei dati fisiologici, unita alla presenza di gap temporali in cui non vengono effettuate registrazioni, comportando un massiccio lavoro preliminare sul dataset "danneggiato".

Le problematiche relative al dataset relativi a pazienti glicemici possono essere riassunte secondo questi due aspetti:

1. Quantità e qualità dei dati: numero limitato di pazienti e di periodi di registrazione dei segnali fisiologici, compreso il livello glicemico. Le serie presentano dei gap temporali che costringono una massiccia fase di preprocessing

2. Utilizzo dei dati: la maggior parte dei dispositivi CGM commerciali danno in dotazione anche mobile APP, da cui è possibile monitorare solamente il segnale glicemico. Quello che manca è un'integrazione sincronizzata multisensoriale di alta qualità, per garantire al soggetto un grado di monitoraggio globale del proprio stato di salute.

6.2.1 Generative Adversarial Networks

Alla mancanza di un numero sufficiente di dati, si potrebbe ovviare tramite dei simulatori di dati fisiologici dei pazienti, in grado di generare settimane o addirittura mesi di registrazioni sintetiche del livello glicemico del sangue. Tutto sta nel cercare il miglior metodo che possa generare dei dati che siano fisiologicamente e clinicamente accettabili. Se si volesse seguire un approccio legato all'Intelligenza Artificiale (AI) per la creazione di simulatori, le tipologie di reti neurali più promettenti sono le recenti reti neurali GAN [67], da Generative Adversarial Networks, cioè Reti Neurali Generative Avversarie.

Le reti GAN spiccano per le doti generative e sono caratterizzate da 2 componenti: il **generatore** e il **descriminatore**, due reti neurali a se stanti con compiti separati, che competono fin quando non si raggiunge un equilibrio (da qui il nome Reti Neurali Generative Avversarie). Il generatore riceve in input rumore casuale ed ha il compito di generare nuovi dati quanto più simili ai dati reali, mentre il discriminatore impara come distinguere i nuovi dati sintetici, ricevuti in input, dai dati reali. Quest'ultimo implementa una classificazione binaria (*real*,*fake*) e rilascia in output la probabilità se un dato è riconosciuto come fittizio (fake) o come un dato vero (real). Infine, quando il descriminatore produrrà una probabilità di 0,5, non avrà più capacità per distinguere la natura del dato, si raggiungerà l'equilibrio e le reti finiranno di competere. La natura generativa della GAN potrebbe rappresentare una chiave sia per costruire dei modelli predittivi, come quelli studiati in questo lavoro di Tesi, sia per ottenere simulatori di pazienti che generano registrazioni di segnale glicemico sintetico e che abbiano la sensibilità, tramite un opportuno training, di generare dati riferiti a più pazienti, mantenendo una certa variabilità inter-paziente e intra-paziente.

6.2.2 Internet of Things

L'integrazione di numerose sorgenti di dati in un unica piattaforma potrebbe garantire una costruzione di un sistema *user centered*, dove il paziente gestisce il suo stato di salute attraverso il monitoraggio dei propri segnali vitali. Una soluzione potrebbe essere adottare un sistema basato sull'IoT [68] (Internet of Things, o Internet delle Cose).

L'IoT rappresenta un innovativo e moderno sistema di comunicazione in cui ogni oggetto intelligente di uso comune può sincronizzarsi con gli altri oggetti tramite

Internet, dando possibilità all'uomo di relazionarsi ad essi e governarli da remoto. Nella sanità e, in particolar modo, nella Telemedicina, il fattore IoT è in forte crescita, poiché permette di formare un collegamento diretto con tra il paziente e il medico o i familiari, tramite il monitoraggio e la condivisione dei parametri vitali. Nel caso di pazienti affetti da diabete, tali segnali, oltre a quello glicemico, potrebbero essere rilevati da sensori indossabili (o *wearable*) collegati ad un server domestico e inoltrati via remoto al paziente stesso, ai caregiver e/o al medico, i quali possono monitorare condizioni di pericolo (tipo iperglicemia o ipoglicemia) o l'efficacia di una terapia insulinica, assicurando al paziente un supporto notevole sia emotivo che clinico.

Nell'esempio di un sistema di monitoraggio domestico basato sull'IoT, si potrebbe pensare ad un elemento centrale in grado di:

- comunicare wireless con i sensori, ricevendo le misurazioni real-time dei segnali glicemici e vitali, con la possibilità di effettuare sia elaborazione che memorizzazione dei dati ricevuti

- comunicare via remoto, inoltrando i dati su piattaforme online condivise dagli utenti interessati (paziente, caregiver, medico) e garantendo un costante monitoraggio dello stato clinico del paziente tramite sistemi di allarme e/o allerta

Il protocollo di comunicazione di tale sistema potrebbe essere l'MQTT [69]. Quest'ultimo è un protocollo di messaggistica appartenente alla tipologia "Publish/Subscribe", caratterizzato dalla presenza di un dispositivo che agisce da intermediario, chiamato *broker*, che riceve i dati dai sensori registrati, noti come *publisher*, e inoltra i messaggi ricevuti agli enti sottoscritti al servizio, chiamati *subscriber*.

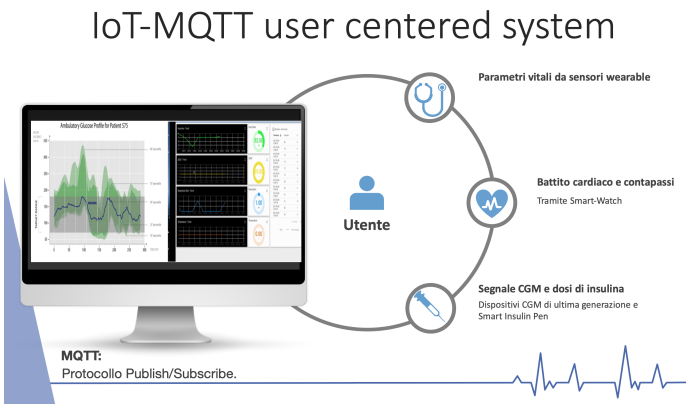


Figura 6.2: Esempio di sistema user centered basato su IoT con piattaforma WEB di monitoraggio

La potenzialità di questo protocollo si basa sul concetto di scalabilità del servizio unito ad un uso di banda molto limitato, il che lo rende perfettamente idoneo ad un sistema IoT, dove gli *smart object*, tipo i dispositivi CGM, erogatori di insulina e altra sensoristica, siano connessi ad un server Cloud o domestico tramite, ad esempio, una Raspberry Pi[70] utilizzata come *broker*, in grado di raccogliere i dati e inoltrarli ad una API Web, garantendo agli utenti registrati un feedback in tempo reale sulle misurazioni disponibili.

Bibliografia

[1] Diabetes Care. «Care in Diabetesd2018». In: *Diabetes Care* 41.1 (2018), S137–S143.

[2] *IDF Diabetes Atlas 8th Edition. 2017.* URL: `https://diabetesatlas.org`.

[3] Eberhard Ritz et al. «End-stage renal failure in type 2 diabetes: a medical catastrophe of worldwide dimensions». In: *American journal of kidney diseases* 34.5 (1999), pp. 795–808.

[4] Fabio Arturo Grieco et al. «Immunology in the clinic review series; focus on type 1 diabetes and viruses: how viral infections modulate beta cell function». In: *Clinical & Experimental Immunology* 168.1 (2012), pp. 24–29.

[5] Masato Kasuga. «Insulin resistance and pancreatic β cell failure». In: *The Journal of clinical investigation* 116.7 (2006), pp. 1756–1760.

[6] Kerry J Welsh, M Sue Kirkman e David B Sacks. «Role of glycated proteins in the diagnosis and management of diabetes: research gaps and future directions». In: *Diabetes care* 39.8 (2016), pp. 1299–1306.

[7] Jeannette Goguen e Jeremy Gilbert. «Hyperglycemic emergencies in adults». In: *Canadian journal of diabetes* 42 (2018), S109–S114.

[8] Michael J Fowler. «Microvascular and macrovascular complications of diabetes». In: *Clinical Diabetes* 29.3 (2011), pp. 116–122.

[9] Gruppo di Studio Intersocietario Piede Diabetico SID-AMD. URL: `https://www.siapav.it/pdf/Piede-Diabetico%5C%202005%5C%20%5C%5B2%5C%5D%5C.pdf`.

[10] Michael S Boyne et al. «Timing of changes in interstitial and venous blood glucose measured with a continuous subcutaneous glucose sensor». In: *Diabetes* 52.11 (2003), pp. 2790–2794.

[11] Katharine Garvey e Joseph I Wolfsdorf. «The impact of technology on current diabetes management». In: *Pediatric Clinics* 62.4 (2015), pp. 873–888.

[12] Thomas Danne et al. «International consensus on use of continuous glucose monitoring». In: *Diabetes care* 40.12 (2017), pp. 1631–1640.

[13] Richard M Bergenstal et al. «Glucose management indicator (GMI): a new term for estimating A1C from continuous glucose monitoring». In: *Diabetes care* 41.11 (2018), pp. 2275–2280.

[14] Boris P Kovatchev et al. «Assessment of risk for severe hypoglycemia among adults with IDDM: validation of the low blood glucose index.» In: *Diabetes care* 21.11 (1998), pp. 1870–1875.

[15] R S Mazze et al. «Ambulatory Glucose Profile: Representation of Verified Self-Monitored Blood Glucose Data». In: *Diabetes care* 10 (gen. 1987), pp. 111–7. DOI: 10.2337/diacare.10.1.111.

[16] *AGPrep*. URL: http://www.agpreport.org/agp/about.

[17] Timothy C Dunn et al. «Development of the Likelihood of Low Glucose (LLG) algorithm for evaluating risk of hypoglycemia: a new approach for using continuous glucose data to guide therapeutic decision making». In: *Journal of diabetes science and technology* 8.4 (2014), pp. 720–730.

[18] Cindy Marling e Razvan C Bunescu. «The OhioT1DM Dataset For Blood Glucose Level Prediction.» In: *KHD@ IJCAI*. 2018, pp. 60–63.

[19] Mi Jordan e TM Mitchell. «Machine learning: Trends, perspectives, and prospects». English. In: *Science* 349.6245 (2015), pp. 255–260. ISSN: 0036-8075.

[20] Microsoft. *Project InnerEye – Medical Imaging AI to Empower Clinicians*. URL: https://www.microsoft.com/en-us/research/project/medical-image-analysis/.

[21] Inc. Restoration Robotics. *THE ARTAS iX^{TM} ROBOTIC SYSTEM*. URL: https://artas.com/physicians/.

[22] Medtronic. *MiniMed 670G Insulin Pump System*. URL: http://medtronicdiabetes.com/products/minimed-670g-insulin-pump-system.

[23] Abbott. *FreeStyleLibrePro*. URL: https://provider.myfreestyle.com.

[24] Medtronic. *Guardian Sensor 3*. URL: https://pro.medtronic-diabete.it/guardian-sensor-3.

[25] Eversense. *Senseonics Eversense*. URL: https://www.eversensediabetes.com.

[26] Dexcom. *DexcomG6*. URL: https://www.dexcom.com/it-IT.

[27] G. F. Page. «INTRODUCTION TO MACHINE LEARNING , by Ethem Alpaydin, MIT Press, 2004, xxx + 415 pp., with index. 160 ref. distributed chapter by chapter. ISBN 0-262-01211-1. (Hardback 32.95)». eng. In: *Robotica* 24.1 (2006), pp. 143–143. ISSN: 0263-5747.

[28] Zoubin Ghahramani. «Unsupervised Learning». In: Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 72–112. ISBN: 978-3-540-28650-9. DOI: 10.1007/978-3-540-28650-9_5. URL: https://doi.org/10.1007/978-3-540-28650-9_5.

[29] Richard S. Sutton e Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 1998. URL: http://www.cs.ualberta.ca/~sutton/book/the-book.html.

[30] Nils J. Nilsson. «Artificial intelligence: A modern approach». eng. In: *Artificial Intelligence* 82.1-2 (1996), pp. 369–380. ISSN: 00043702.

[31] Yann LeCun et al. «Gradient-Based Learning Applied To Document Recognition». In: (1998).

[32] Andrew Zisserman Karen Simonyan. «Very Deep Convolutional Networks for Large-Scale Image Recognition». In: *arXiv:1409.1556* (2014).

[33] Geoffrey E. Hinton Alex Krizhevsk Ilya Sutskever. «ImageNet Classification with Deep Convolutional Neural Networks». In: (2012).

[34] Stanford CS class. *CS231n: Convolutional Neural Networks for Visual Recognition*. URL: http://cs231n.github.io/convolutional-networks/.

[35] Ian Goodfellow, Yoshua Bengio e Aaron Courville. *Deep Learning*. MIT Press, 2016.

[36] Zhou e Chellappa. «Computation of optical flow using a neural network». eng. In: *IEEE 1988 International Conference on Neural Networks*. IEEE, 1988, 71–78 vol.2.

[37] Sepp Hochreiter e Jürgen Schmidhuber. «Long Short-Term Memory». In: *Neural Comput.* 9.8 (nov. 1997), pp. 1735–1780. ISSN: 0899-7667. DOI: 10.1162/neco.1997.9.8.1735. URL: http://dx.doi.org/10.1162/neco.1997.9.8.1735.

[38] Henry J Kelley. «Gradient theory of optimal flight paths». In: *Ars Journal* 30.10 (1960), pp. 947–954.

[39] Razvan Pascanu, Tomas Mikolov e Yoshua Bengio. «On the difficulty of training Recurrent Neural Networks». In: (2012).

[40] Ilya Sutskever, Oriol Vinyals e Quoc V Le. «Sequence to sequence learning with neural networks». In: *Advances in neural information processing systems*. 2014, pp. 3104–3112.

[41] V. Tresp, T. Briegel e J. Moody. «Neural-network models for the blood glucose metabolism of a diabetic». In: *IEEE Transactions on Neural Networks* 10.5 (set. 1999), pp. 1204–1213. ISSN: 1045-9227. DOI: 10.1109/72.788659.

[42] W. Sandham et al. «Blood glucose prediction for diabetes therapy using a recurrent artificial neural network». In: *9th European Signal Processing Conference (EUSIPCO 1998)*. Set. 1998, pp. 1–4.

[43] Giovanni Sparacino et al. «Glucose concentration can be predicted ahead in time from continuous glucose monitoring sensor time-series». In: *IEEE Transactions on biomedical engineering* 54.5 (2007), pp. 931–937.

[44] Meriyan Eren-Oruklu et al. «Estimation of future glucose concentrations with subject-specific recursive linear models». In: *Diabetes technology & therapeutics* 11.4 (2009), pp. 243–253.

[45] Kamuran Turksoy et al. «Hypoglycemia Early Alarm Systems Based On Multivariable Models». eng. In: 52.35 (2013). ISSN: 0888-5885.

[46] C. Pérez-Gandía et al. «Artificial Neural Network Algorithm for Online Glucose Prediction from Continuous Glucose Monitoring». In: *Diabetes Technology & Therapeutics* 12.1 (2010). PMID: 20082589, pp. 81–88. DOI: 10.1089/dia.2009.0076. eprint: https://doi.org/10.1089/dia.2009.0076. URL: https://doi.org/10.1089/dia.2009.0076.

[47] C. Zecchin et al. «Jump neural network for online short-time prediction of blood glucose from continuous monitoring sensors and meal information». eng. In: *Computer Methods and Programs in Biomedicine* 113.1 (2014), pp. 144–152. ISSN: 0169-2607.

[48] Kezhi Li et al. «Convolutional Recurrent Neural Networks for Glucose Prediction». In: (2018).

[49] Chiara Dalla Man et al. «The UVA/PADOVA Type 1 Diabetes Simulator: New Features». eng. In: 8.1 (2014), pp. 26–34. ISSN: 1932-2968.

[50] Alessandro Aliberti et al. «A Multi-Patient Data-Driven Approach to Blood Glucose Prediction». eng. In: *IEEE Access* 7 (2019), pp. 69311–69325. ISSN: 2169-3536.

[51] URL: http://diabetes.jaeb.org.

[52] URL: https://www.ijcai-18.org.

[53] John Martinsson et al. «Automatic blood glucose prediction with confidence using recurrent neural networks». eng. In: 2018, pp. 64–68.

[54] Taiyu Zhu et al. «A Deep Learning Algorithm for Personalized Blood Glucose Prediction». In: *KHD@IJCAI*. 2018.

[55] Aäron van den Oord et al. «WaveNet: A Generative Model for Raw Audio». In: *SSW*. 2016.

[56] Jinyu Xie e Qian Wang. «Benchmark Machine Learning Approaches with Classical Time Series Approaches on the Blood Glucose Level Prediction Challenge.» In: *KHD@ IJCAI*. 2018, pp. 97–102.

[57] Shaojie Bai, J. Zico Kolter e Vladlen Koltun. «An Empirical Evaluation of Generic Convolutional and Recurrent Networks for Sequence Modeling». In: *arXiv:1803.01271* (2018).

[58] *definizione sparso*. URL: http://www.treccani.it/vocabolario/sparso/.

[59] URL: https://gitlab.com/AlesioRFM/dcam-neuron.

[60] URL: https://health.gov/dietaryguidelines/dga2010/DietaryGuidelines2010.pdf.

[61] URL: https://calculator.academy/steps-to-calories-calculator/#f1p1%7Cf2p0.

[62] Thomas Danne et al. «International Consensus on Use of Continuous Glucose Monitoring». eng. In: *Diabetes care* 40.12 (2017), p. 1631. ISSN: 01495992.

[63] Sergey Ioffe e Christian Szegedy. «Batch normalization: Accelerating deep network training by reducing internal covariate shift». In: *arXiv preprint arXiv:1502.03167* (2015).

[64] Diederik P. Kingma e Jimmy Ba. «Adam: A Method for Stochastic Optimization». In: *arXiv e-prints*, arXiv:1412.6980 (dic. 2014), arXiv:1412.6980. arXiv: 1412.6980 [cs.LG].

[65] Keras Documentation - Predict function. *Utils - Keras Documentation*. URL: https://keras.io/models/model/#predict.

[66] hpc@polito. *Project of Academic Computing within the Department of Control and Computer Engineering at the Politecnico di Torino*. URL: http://hpc.polito.it/.

[67] Ian Goodfellow et al. «Generative Adversarial Nets». In: *Advances in Neural Information Processing Systems 27*. A cura di Z. Ghahramani et al. Curran Associates, Inc., 2014, pp. 2672–2680. URL: http://papers.nips.cc/paper/5423-generative-adversarial-nets.pdf.

[68] Luigi Atzori, Antonio Iera e Giacomo Morabito. «The internet of things: A survey». In: *Computer networks* 54.15 (2010), pp. 2787–2805.

[69] Roger A Light et al. «Mosquitto: server and client implementation of the MQTT protocol.» In: *J. Open Source Software* 2.13 (2017), p. 265.

[70] Warren Gay. *Raspberry Pi Hardware Reference*. 1st. Berkely, CA, USA: Apress, 2014. ISBN: 1484208005, 9781484208007.