Da ECG a LLM

Leonardo Bigelli[1]

[1]l.bigelli2@campus.uniurb.it

14 luglio 2025

Riassunto

Questo progetto descrive l'implementazione di un modello di Deep Learning sviluppato per convertire una serie temporale di un elettrocardiogramma, in una serie di TOKENs. Questo cambiamento di dominio, permetterà in futuro di addestrare da zero un generico LLM su dati, non testuali, inerenti ad ECG.

1 Introduzione e obiettivi

I modelli LLMs stanno riscontrando un notevo successo, soprattutto in ambito testuale. In letteratura esistono diversi studi in materia. Una delle maggiori utilità è senza dubbio la generazione di testo a partire da una singola parola o frase.
Alcuni autori hanno sviluppato un sistema permettendo a un modello LLM di essere addestrato da zero su dati non testuali, inerenti ad acceleromentro, e dimostrando che è possibile utilizzare questi modelli per scopi di interpolazione [2].
Questo progetto nasce dall'idea che i dati in ambito medico sono spesso pochi per modelli di Deep Learning, perciò sarebbe interessante valutare la generazione di dati sintetici, tramite l'utilizzo di LLMs, a partire dai pochi valori di input che abbiamo a nostra disposizione. Per poter addestrare un Large Language Model da zero, o fine-tuning, è necessario però che i valori dei nostri dati siano sotto forma di *TOKENs*, ovvero un insieme finito di valori numeri interi che un LLM può comprendere. Il seguente progetto illustra una struttura basata su Autoencoder convuluzionari, capace di poter codificare una serie temporale di un ECG, in un insieme finito di Tokens.

2 Setup

In questa sezione verranno delucidate le tecniche implementate per poter implementare il progetto in questione.

2.1 Dataset

Per la sperimentazione è stato utilizzato il dataset MIT-BIH Arrhythmia Database, usufruibile attraverso il modulo *wfdb* di Python. Questo dataset è uno dei più utilizzati nel dominio della cardiologia computazionale e fornisce elettrocardiogrammi (ECG) annotate manualmente da esperti, provenienti da pazienti con aritmie documentate. I dati sono campionati a 360 Hz e hanno una durata media di circa 30 minuti, per un totale di circa 650.000 campioni per traccia.
Ogni file contiene due derivazioni ECG simultanee (tipicamente lead II e lead V1), ma nel nostro progetto è stata utilizzata una sola traccia per semplicità. Il segnale è fornito come sequenza di valori digitali normalizzati e viene caricato tramite il pacchetto 'wfdb', che consente di accedere direttamente ai tracciati '.dat' e alle annotazioni '.hea'.
In fase di preprocessing, il segnale continuo viene segmentato in finestre di 250 campioni ciascuna (õ.694 secondi), che vengono successivamente suddivise in sotto-blocchi più piccoli, di *'n'* campioni, come input all'autoencoder. La figura 1 mostra un generico segnale preso in considerazione. La scelta di questa struttura gerarchica permette di mantenere la sequenzialità temporale e consente l'analisi a livello microstrutturale del battito cardiaco, facilitando la successiva conversione in TOKENs.
Questa modalità di accesso e segmentazione rende il dataset MIT-BIH particolarmente adatto per esplorare tecniche di compressione, quantizzazione e tokenizzazione, offrendo segnali di alta qualità.

2.2 Framework utilizzato

Per lo sviluppo della rete neurale, è stato impiegato il framework PyTorch. Mentre per l'addestramento della rete stessa, è stata sfruttata l'applicazione Google Colab.

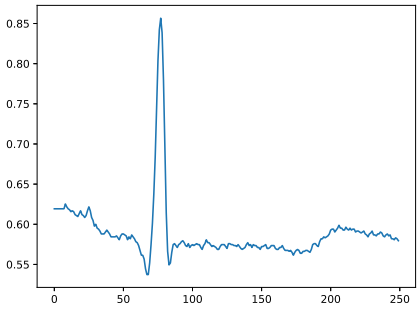

Figura 1: Sequenza di 250 campioni del primo canale di un ECG, normalizzati tra 0 e 1.

3 Metodologia

In questa sezione verranno spiegati i componenti utilizzati con relative motivazione. Segue una delucidazione sull'addestramento della rete finale.

3.1 Autoencoders

Il modello di rete principale è un autoencoder convoluzionale. Questa rete ha lo scopo di prendere un successione di campioni dell' ECG e di codificarla in un unico elemento. La dimensione dell'input (e output) dell'autoencoder dipende dalla lunghezza della dimensione del blocco di campioni consecutivi.
Per poter codificare n elementi in un singolo, lo spazio latente dell'autoencoder è stato definito con un singolo elemento. Quest'utlimo restituirà un solo valore ma di tipo **Float**.
L'architettura dell'autoencoder è descritta nella seguente tabella 1, essa ha lo scopo di ridurre solamente la dimensionalità degli elementi in input, fino al raggiungimento di una singola dimensione. Successivamente è stata sviluppata una versione con due layers custom, che si occupano della quantizzazione subito dopo dello spazio latente (con relativa decodifica).

3.2 Funzione di Quantizzazione

Considerando che la parte dell'autoencoder predisposta al processo di tokenizzazione (Encoder), restituisce valori non interi è necessario applicare una funzione che possa realizzare i token veri e propri che poi potranno essere utilizzati con modelli LLMs. Il processo seguito è chiamato *Quantizzazione* e consente di codificare segnali continui in rappresentazioni discrete. In questo progetto è stata utilizzata una reinterpretazione della funzione di quantizzazione citata e impiegata da Ansari et al.[1]

La funzione predisposta alla codifica da elementi dello spazio latente a TOKENs, si basa sulla suddivisione di un tensore in B bins su un intervallo definito su $[min\_val, max\_val]$, mappando ogni valore reale all'indice intero del bin corrispondente.
Segue la funzione sopra descritta:

```
bin_width = (max_val - min_val) /
            num_bins
q = ((x - min_val) /
     bin_width).floor()
     .clamp(0, num_bins - 1).to(torch.
         int64)
```

Dove, max_val e min_val sono valori scelti albritrarialmente che dovrebbero rappresentare gli estremi dei possibili valori dello spazio latente. In questo caso ± 5.

Per quanto riguarda la funzione inversa, dequantizzazione, ha lo scopo di mappare ciascun indice intero del bin nel centro del suo intervallo, ricostruendo così una versione continua approssimata del dato originario. Sebbene tale ricostruzione non sia perfettamente invertibile, essa è sufficiente per alimentare moduli decoder o ricostruttivi all'interno di autoencoder, garantendo coerenza. Segue la funzione di de-quantizzazione:

```
    bin_width = (max_val - min_val) /
                num_bins
    centers = min_val + bin_width *
              (q.float() + 0.5)
```

4 Risultati sperimentali

Nella seguente sezione, vengono illustrati gli esperiementi effettuati. La prima parte è incentrata sullo studio del numero di campioni da codificare in token. Questo rappresenta la parte più delicata dell'intero processo. Successivamente, la funzione di quantizzazione verrà valutata per poi raggiungere il risultato finale, ovvero l'applicazione della riduzione di dimensionalità con relativa funzione di quantizzazione.

4.1 Riduzione della dimesionalità

Per valutare l'efficenza del tokenizzatore in questione, la serie di partenza rappresentante l'ECG, lunga 250 campioni, è stata suddivisa in blocchi di dimensione variabile. Rispettivamente di dimensione 250, 125, 50, 25, 10, 5 e 2 campioni. Prima della suddivisione in blocchi, l'intera serie temporale è stata normalizzata in un intervallo di [0, 1]. Si noti come è

Tabella 1: Struttura del modello `DimReduction` con dimensioni dei tensori per input $(B, 1, 5)$

Blocco	Layer	Output shape
Encoder	Conv1D(1→8, k=3, p=1) + BN + ReLU	(B, 8, 5)
	Conv1D(8→16, k=3, p=1) + BN + ReLU	(B, 16, 5)
	Conv1D(16→32, k=3, p=1) + BN + ReLU	(B, 32, 5)
	AdaptiveAvgPool1D(1)	(B, 32, 1)
Decoder	ConvTranspose1D(32→16, k=3, p=1) + BN + ReLU	(B, 16, 1)
	Upsample(3)	(B, 16, 3)
	ConvTranspose1D(16→8, k=3, p=1) + BN + ReLU	(B, 8, 3)
	Upsample(5)	(B, 8, 5)
	ConvTranspose1D(8→1, k=3, p=1) + Sigmoid	(B, 1, 5)

stato eseguito un esperimento codificando anche l'intera finestra in un singolo token. La tabella 2 mostra i risultati ottenuti espressi tramite la **metrica Mean Absolute Error**. Come si può notare l'errore non sembra così elevato ed è inversamente proporzionale alla dimensione del blocco. Fondamentale è ricordare che il segnale è espresso con valori compresi tra 0 e 1. Blocchi più piccoli prevedono l'utilizzo di più token per codificare una sequenza da 250, quindi è necessario un *trade-of* per definire la dimensioni dei blocchi più ottimale.

La Figura 2 mostra l'errore di ricostruzione per quanto riguarda il processo di riduzione della dimensionalità da n a 1, senza l'applicazione della funzione di quantizzazione. Come si può notare, la codifica da 250 sample a 1 è la peggiore, in quanto la serie ricostruita è totalmente diversa dall'originale. Analizzando bene le varie figure, il migliore rapporto tra dimensione del blocco da tokenizzare e MSE, si riferisce ad una dimensione di 10 campioni consecutivi con relativo MSE di *0.000356*.

Tabella 2: Mean Square Errors comparati con le diverse dimensioni dei blocchi

dim_block	MSE
250	0.550570
125	0.254592
50	0.082695
25	0.011730
10	0.000356
5	0.000086
2	0.000009

4.2 Quantizzazione

Per ottenere i token finale è necessario applicare la funzione di quantizzazione sul risultato definito dallo spazio latente. I parametri da dover impostare per usufruire della seguente funzione sono il numero di bin, il valore minimo e il valore massimo dell'intervallo. Il primo parametro è stato scelto in maniera arbitraria di 256 e lasciato invariato per l'esperimento. Ovviamente più questo numero sarà piccolo e maggiore sarà l'errore di ricostruzione.
Per quanto riguarda i valori di minino e massimo, la soluzione ottimale sarebbe quella di utilizzare i valori limite reali dello spazio latente, cosa che a lato pratico non è possibile sapere. La figura 3 mostra uno segnale definito dallo spazio latente confrontato con la sua ricostruzione. Come si può notare l'errore è minimo, quasi assente, dovuto all'impostazione dei limiti.

Successivamente sono stati impostati in maniera arbitraria i valori dell'intervallo a [-5, +5]. La figura 4 mostra l'errore di ricostruzione della funzione di quantizzazione relativa a *n° bins = 256*, *min = -5* e *max = +5*.

4.3 Processo di Tokenizzazione completo

Al modello di autoencoer sono stati aggiunti due layers custom non addestrabili per implementare la funzione di quantizzazione e de-quatizzazione in maniera automatica.
Dopo aver addestrato la rete, l'errore MSE in fase di test ottenuto è di 0.000544. La figura 5 mostra sempre lo stesso segnale ECG usato in fase di test, ma processato con l'autoencoder finale.

5 Conclusioni

Il seguente progetto mira a risolvere un problema sostanziale nell'ambito medico, ovvero la carenza di dati. Una delle possibili soluzioni sarebbe quella di impiegare i vari LLM per la generazione di segnali sin-

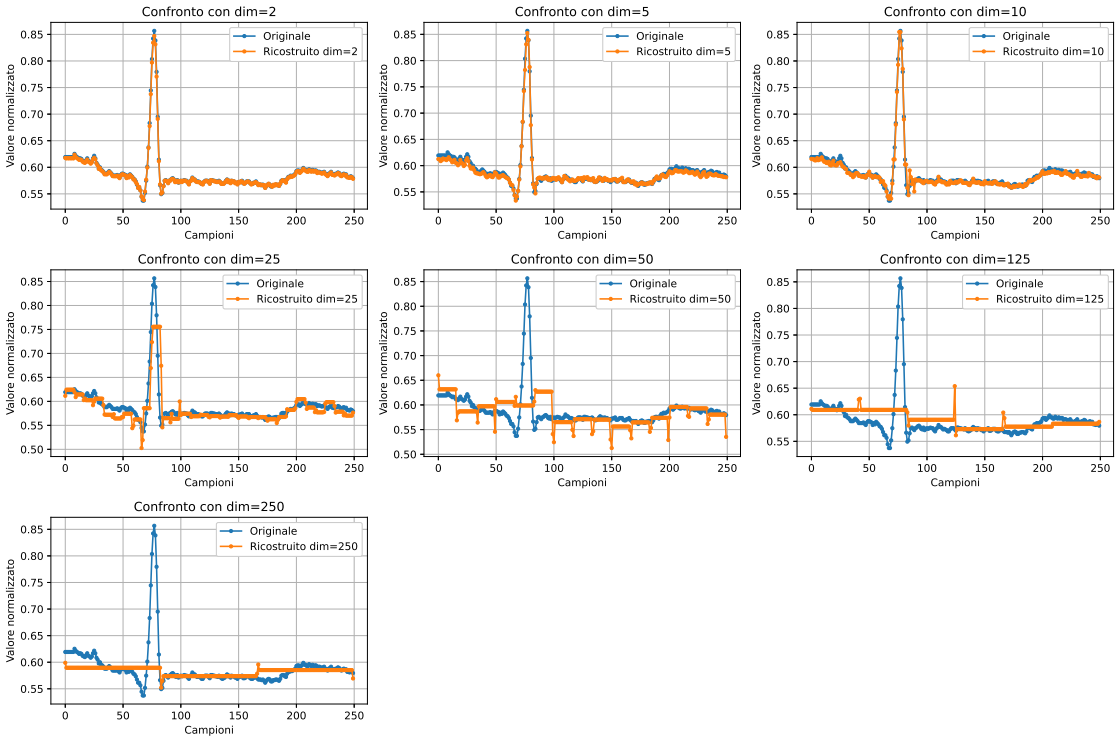

Figura 2: Serie di partenza confrontata con il risultato dell'autoencoder addestrato con diverse dimensioni di input. Rispettivamente di 2, 5, 10, 25, 50, 125 e 250 campioni.

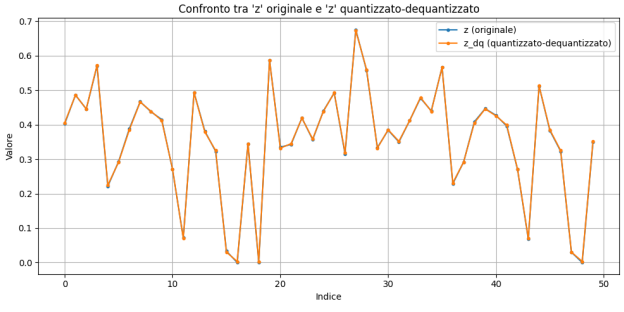

Figura 3: Funzione di quantizzazione applicata (con relativa de-quantizzazione) allo spazio latente, con estremi equivalenti a quelli dello spazio latente stesso. Il grafico mostra solamente una parte della lunghezza totale del segnale.

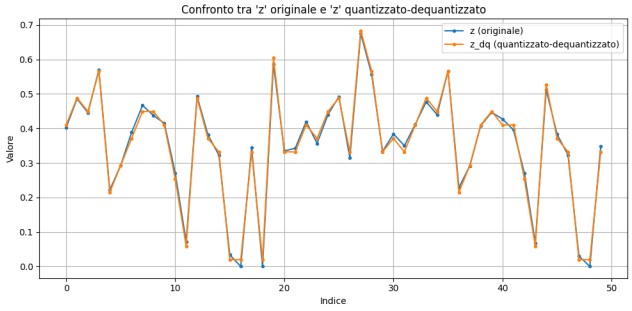

Figura 4: Funzione di quantizzazione applicata (con relativa de-quantizzazione) allo spazio latente, con estremi di [-5, +5]. Il grafico mostra solamente una parte della lunghezza totale del segnale.

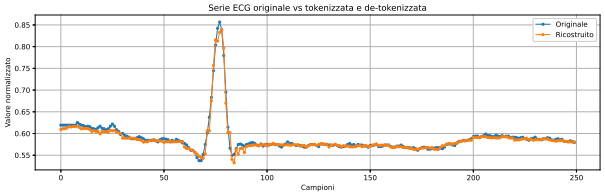

Figura 5: Serie di partenza confrontata con il risultato dell'applicazione dell'autoencoder con i due layers custum per gestire la quantizzazione.

tetici. Per far fronte a ciò, è necessario un cambio di dominio, per codificare n campioni consecutivi in un unico token, per poi essere usato per l'addestramento di un LLM generico.
Il progetto ha lo scopo di ideare un processo di tokenizzazione di serie inerenti al primo canale del ECG, tramite lo sviluppo di un architettura ad autoencoder con layer convoluzionali.
Dai vari test effettuati si può evincere che il processo di tokenizzazione è stato un successo, considerando l'errore di ricostruzione ottenuto molto basso.

5.1 Possibili sviluppi futuri

Questo progetto si limita a realizzare solamente il tokenizzatore, per maggior completezza sarebbe necessario addestrare da zero un modello LLM con lo scopo di poter generare sinteticamente segnali ECG a partire da altri. Per l'addestramento però è necessaria una grande potenza di calcolo, nonché di tempo. Motivo per cui non è stata studiata e approfondita anche questa parte nel seguente progetto.

Riferimenti bibliografici

[1] Abdul Fatir Ansari et al. "Chronos: Learning the language of time series". In: *arXiv preprint arXiv:2403.07815* (2024).

[2] Bigelli Leonardo et al. "Can LLMs Learn the Language of Inertial Sensors in Human Activity Recognition?" In: *Printing on 10th International Conference on Machine Learning Technologies* (2025).