Deep Learning Project

Bartucci Simone, Fazio Francesco
A.A. 2021/2022

Contents

1 Introduzione **2**
1.1 Task 1 . . . 2
1.2 Task 2 . . . 2

2 Comprensione e preparazione dati **3**
2.1 Task 1 . . . 3
2.2 Task 2 . . . 4

3 Modello **5**
3.1 Task 1 . . . 5
3.2 Task 2 . . . 9

4 Valutazioni modello **10**
4.1 Task 1 . . . 10
4.2 Task 2 . . . 12

5 Analisi dati e risultati **13**
5.1 Task 1 . . . 13
5.2 Task 2 . . . 17

6 Conclusioni **23**
6.1 Task 1 . . . 23
6.2 Task 2 . . . 23

1 Introduzione

1.1 Task 1

Il progetto realizzato è costituito da due tracce distinte. Concentrandoci sulla prima traccia, per essa sono stati forniti due dataset, uno di train ed uno di test. All'interno di questi file, sono presenti diverse colonne in cui vengono riportate delle sequenze numeriche. Il primo obiettivo da raggiungere è stato quello di realizzare una rete neurale in grado di prevedere i valori presenti nel dataset di test.
Nella traccia di tale esercizio viene riportato che il tempo di acquisizione per la raccolta dei dati è di 10 secondi. Ci si pone come obiettivo quello di utilizzare delle sequenze di 5 minuti, distanziate tra loro di un minuto, per prevedere i valori seguenti. All'interno del codice queste due informazioni sono state tradotte come:

- lookback = 30 $\Longleftrightarrow$ sequenze temporali di 5 minuti
- window_shift = 6 $\Longleftrightarrow$ distanziamento tra le sequenze temporali

Per fare ciò, dopo aver elaborato opportunamente i dati, sono state realizzate delle serie temporali mediante i dati di train e di test, ossia dei tensori tridimensionali strutturati come segue:

$$dim(Time\_series) = (samples, lookback, number\ of\ features)$$

In parallelo, oltre a questi tensori, sono stati generati anche dei vettori di dimensione pari a "samples", che associano ad ogni sequenza della serie temporale il valore successivo corrispondente, ossia quello da predirre. Questo tipo di apprendimento è noto come supervised learning, in cui si addestra un modello su dati di input e di output noti, in modo che possa prevedere gli output futuri. Quindi i dati di input considerati sono i tensori precedentemente definiti e i dati di output sono i loro vettori associati.
La qualità delle previsioni realizzate è stata valutata mediante l'errore medio assoluto (MAE).
Una volta fatto questo, ci si è concentrati sulla struttura delle sequenze, studiando il modo in cui lookback e window_shift variano la qualità dell'apprendimento del modello.

1.2 Task 2

Per quanto riguarda la seconda task, anche in tale caso sono stati forniti due dataset, uno di train ed uno di test. L'obiettivo finale tuttavia è differente, ossia determinare la presenza di anomalie all'interno dei dati di test.
I due dataset infatti conservano informazioni su una serie di pazienti sani o malati di Parkinson.
Il Parkinson è una malattia neurodegenerativa, ad evoluzione lenta ma progressiva, che coinvolge, principalmente, alcune funzioni quali il controllo dei movimenti e dell'equilibrio.
I principali sintomi motori del Parkinson sono il tremore a riposo, la rigidità, la bradicinesia e, in una fase più avanzata, l'instabilità posturale .
Il tremore non è presente in tutti i pazienti. All'esordio della malattia, spesso i sintomi non vengono riconosciuti immediatamente, perché si manifestano in modo subdolo, incostante e la progressione della malattia è tipicamente lenta.
In tale contesto il deep learning diventa uno strumento attraverso il quale è possibile determinare le anomalie presenti all'interno di un paziente e poter verificare l'eventuale presenza della malattia o avere un'idea qualitativa del suo stato di avanzamento.
E' stato definito un approccio basato sulle serie temporali, finalizzato a rilevare le eventuali anomalie.
Sono stati presi in considerazione due dataset, uno contenente dati inerenti a pazienti sani e uno conentenente dati inerenti a pazienti con Parkinson. I due dataset contengono differenti informazioni. In particolare:

- Paziente considerato
- Accelerometro che legge i valori dati dai tre assi (x, y, z)

- Battito cardiaco
- Data e tempo di campionamento

Il dataset di train è costituito da volontari sani, che quindi non presentano disturbi legati al Parkinson. Il dataset di test invece è costituito da pazienti malati di Parkinson. In particolare, ogni paziente malato è sottoposto a cure mediche per alleviare i sintomi del Parkinson. Tuttavia, tali cure non sono efficaci in ogni momento e per tale ragione, vengono a generarsi degli intervalli temporali denominati "OFF Periods" nei quali i sintomi del Parkinson ritornano. Queste sono le anomalie che saranno ricercate all'interno dei dataset: periodi in cui i sintomi del Parkinson sono nuovamente presenti.
In base a quanto detto, il criterio scelto per la realizzazione del modello è stato il seguente:
Sono stati considerati i pazienti sani per allenare il modello e renderlo in grado di elaborare i valori legati ad una persona sana. Fatto questo, il modello è stato applicato sui dati di test, inerenti ai pazienti malati. In tal modo, i valori che il modello non è in grado di ricostruire con un'opportuna accuratezza sono stati classificati come anomalie legate al Parkinson.

2 Comprensione e preparazione dati

2.1 Task 1

I dati presenti nel train e nel test sono raccolti in tre colonne "x, y, z" e vengono campionati ogni dieci secondi. La traccia dell'esercizio richiede che, date sequenze temporali di 5 minuti ogni minuto, si debba prevedere il valore successivo di ogni sequenza.
Innanzitutto è stata effettuata una normalizzazione dei dati al fine di riscalarli nell'intervallo $[0, 1]$.
Nella cella in cui viene effettuata la normalizzazione, sui dati di train viene effettuato uno split, eliminando una piccola porzione dei dati finali, i quali verranno utilizzati non per allenare il modello ma per la validazione di quest'ultimo e verificarne la qualità. In questo modo è possibile studiare l'eventuale presenza di "overfitting" o "underfitting".
Dopo aver normalizzato il dataset, il passo successivo è stato elaborare un generatore di dati che restituisca una serie temporale di dimensione (samples,lookback,number of features) ed il relativo vettore di valori predetti con dimensione "samples", come spiegato precedentemente nell'introduzione della task 1.

```
def generator(data, lookback, min_index, max_index,window_shift=window_shift):
  dim_samples = ((len(data)- lookback + window_shift-1 )//window_shift, lookback,1)
  dim_target = ((len(data)- lookback + window_shift-1 )//window_shift,)
  samples = np.zeros(dim_samples)
  target = np.zeros(dim_target)
  i=0
  j=0
  while i<(len(data) - lookback + window_shift-1)//window_shift:
    for k in range(0, lookback,1):
      samples[i,k,0] =data[i*window_shift+k]
    target [i]=data[lookback + i*window_shift]
    i +=1
    j+= lookback
  return samples, target
```

All'interno di tale funzione, vengono realizzate le serie temporali ed il relativo vettore. Si noti che si è scelto di lavorare con gli array, di conseguenza è stato necessario determinare a priori la loro dimensione. Questa è stata ricavata matematicamente in modo empirico, analizzando diversi casi e determinando la formula che lega la dimensione dell'array a lookback e window_shift.
Per verificare la veridicità di tale relazione riportiamo un esempio. Consideriamo una sequenza di

numeri da 1 a 15, lookback = 3, window_shift = 2. Applicando a tale casistica la funzione generator otteniamo la seguente serie temporale:

```
[[ 1.  2.  3.]
 [ 3.  4.  5.]
 [ 5.  6.  7.]
 [ 7.  8.  9.]
 [ 9. 10. 11.]
 [11. 12. 13.]]
```

a cui viene associato il vettore:

```
[ 4.  6.  8. 10. 12. 14.]
```

come possiamo notare, le sequenze contengono 3 elementi ciascuna, come richiesto dal lookback e sono distanziate tra loro di un valore pari a 2, ossia il valore del window_shift. Le condizioni richieste quindi vengono perfettamente rispettate ed una volta arrivati all'ultima sequenza è possibile, il processo si arresta. Questo dimostra che anche la dimensione dell'array è stata definita in maniera corretta.

2.2 Task 2

Prima di essere utilizzati all'interno del modello, sia per il train che per il test, i dati sono stati elaborati e modificati opportunamente.
Per prima cosa ci si è concentrati sui dati di test e per ragioni di visualizzazione, le colonne del test sono state riordinate per essere ridisposte come quelle di train.
Successivamente sono state definite due liste, la prima contenente i pazienti analizzati, la seconda gli indici che si riferiscono alla fine della raccolta dati per ogni paziente. Ciò è stato fatto per dividere i dati di test, andando a considerare singolarmente i pazienti.
Successivamente, ci si è concentrati sui dati di train. Innanzitutto sono state eliminate tutte le righe contenenti 'HeartRate=-1', poichè nella colonna "Heart Rate" vi sono dati mancati al posto dei quali è stato riportato "-1". Questi valori sono stati eliminati, per far si che non andassero ad alterare le prestazioni del modello finale.
Ci si è poi concentrati sui tempi di campionamento dei due dataset. E' possibile osservare che, mentre i dati di test sono acquisiti ogni 10 secondi, i dati di train sono acquisiti ogni secondo. Per eliminare tale discrepanza, il train viene rielaborato, effettuando una media su intervalli di campionamento di 10 secondi. Il risultato finale è un dataset riscalato di un fattore 10 rispetto a quello iniziale, i cui valori sono distanziati di 10 secondi tra loro.
E' stata poi realizzata una normalizzazione dei dati per riscalarli nell'intervallo $[0, 1]$. In particolare, la normalizzazione è stata tale per cui le colonne "HeartRate", "x", "y" e "z" di train fossero comprese nell'intervallo $[0, 1]$. Per mantenere la stessa metrica, gli stessi parametri di normalizzazione di train, sono stati applicati ai dati di test.
Fatto questo, è stato possibile costruire le serie temporali utilizzate nel modello. Per generare la serie temporale la scelta di lookback e window_shift è stata arbitraria, in particolare sono stati considerati:

- lookback = 60
- window_shift = 1

La scelta di tale window_shift ci ha permesso di semplificare anche la funzione destinata alla generazione della serie temporale:

```
# Tramite il generator    possibile definire i tensori necessarari nel modello
def generator(data, paziente, lookback):
    samples = []

    for i in range(len(data['heartRate']) - lookback):
```

```
        if(paziente[i] == paziente[i+lookback]):
            samples.append(data[i : (i + lookback)])
    return np.stack(samples)
```

Si sottolinea la presenza di un "if" all'interno del generator, finalizzato a considerare sequenze che contengono dati di un solo paziente alla volta. In questo modo ogni seguenza contiene dati inerenti ad un solo paziente.
Si noti che nel generator della task 1 si sceglie di lavorare con array, di conseguenza è stato necessario determinare a priori la dimensione di questi. Nella task 2 invece il tutto viene semplificato utilizzando le liste, che ci permettono di non doverne esplicitare la dimensione a priori.
Le serie temporali presentano in tal caso una dimensione tale per cui

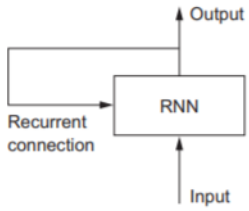
$$dim(Time\_series) = (samples, 60, 4)$$

il "number of features" in tal caso è pari a 4. Questo dipende dal fatto che sono state considerate le quattro colonne di "HeartRate","x","y" e "z" contemporaneamente, mentre nella task 1 avevamo "number of features=1" poiché le colonne erano considerate separatamente in tre simulazioni differenti.

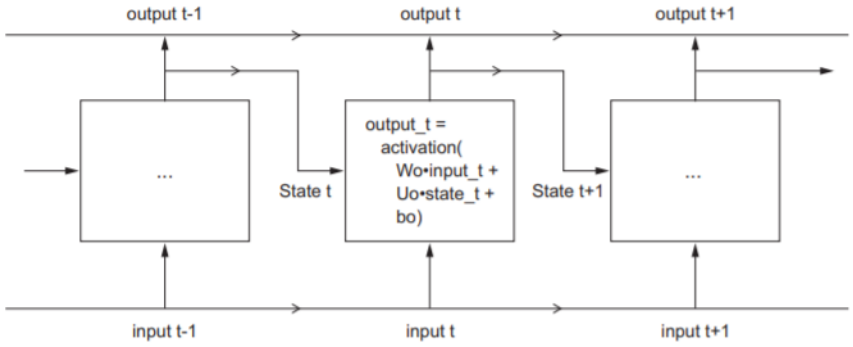
3 Modello

3.1 Task 1

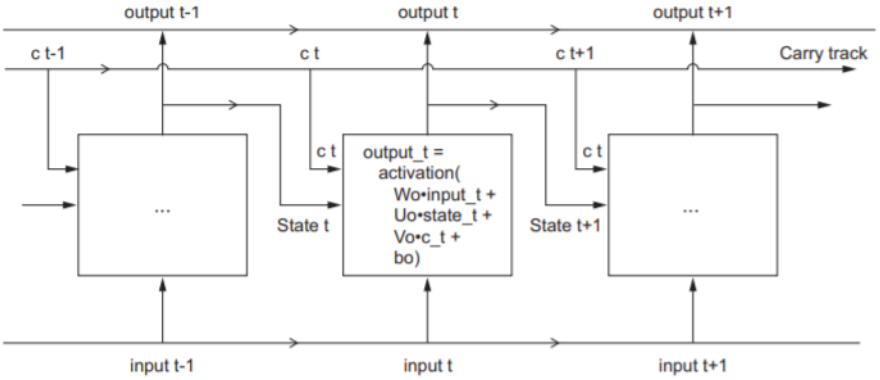
Una rete neurale RNN (recurrent neural network) è una categoria di rete neurale artificiale che include neuroni collegati tra loro in un loop, in particolare contiene connessioni all'indietro o auto-connessioni, invece di avere solo connessioni in avanti, come in una rete neurale feed-forward (FFNN). Una rete RNN è un tipo di rete neurale che presenta un loop interno. La schematizzazione più semplice che si può fare di una RNN è la seguente:



Esistono diversi modelli RNN, le più semplici sono le "SimpleRNN" le quali tuttavia non vengono usate frequentemente poiché soggette a diversi problemi. Sebbene teoricamente la SimpleRNN dovrebbe essere in grado di conservare le informazioni sugli input visti molti passaggi temporali prima, in pratica tali dipendenze sono impossibili da apprendere a lungo termine, ciò è dovuto al "vanishing gradient problem".
Per tale ragione vengono introdotte le unità LSTM (Long Short-Term Memory), che sono una variante delle SimpleRNN ma in grado di trasportare le informazioni in molti step temporali. Per comprendere ciò è possibile prendere in considerazione una cella si una SimpleRNN:



E' possibile ora aggiungere a tale modello una variabile in grado di trasportare i dati attraverso le fasi temporali. Facendo ciò, tale informazione avrà il seguente impatto sulle celle: sarà combinata con le connessioni in input e con le connessioni ricorrenti e influenzerà le variabili che verranno inviate allo step successivo. Concettualmente questo aumento di complessità del modello ha lo scopo di modulare l'output e lo stato successivo mediante i dati precedenti.
Schematizzando, si ottengono delle celle così strutturate:



Questo è il modo in cui è possibile interpretare un layer tipo LSTM.
Un altro modello di RNN molto diffuso è quello GRU (Gated recurrent unit), che lavora utilizzando lo stesso principio delle LSTM, ma che risulta essere meno complesso e quindi più veloce nella fase di esecuzione del modello. Proprio per tale considerazione all'interno del modello realizzato è stato scelto di utilizzare dei layer di tipo GRU.
Per tutte e tre le colonne "x,y,z" è stato mantenuto lo stesso modello, quello che differenzia i tre casi è il numero di epoche. Infatti per le colonne "x" e "y" sono state considerate 20 epoche, per quanto riguarda "z" sono state considerate 10 epoche. Per ottimizzare i parametri e trovare il modello più adatto è stata presa in considerazione la MAE sui dati di test. Tale scelta è stata determinata da alcuni valori di riferimento della MAE:

- x: 81.06
- y: 85.26
- z: 79.94

L'obiettivo finale è stato quindi quello di realizzare dei modelli che ci permettessero di ottenere valori di MAE confrontabili a questi o anche inferiori.

```
model = Sequential()
```

```
model.add((layers.GRU(16, input_shape = (seq_len, num_feat),
return_sequences = True, activation = 'relu')))
model.add((layers.GRU(8, input_shape = (seq_len, num_feat ),
return_sequences = False)))
model.add(layers.Dense(32))
model.add(layers.Dense(16))
model.add(layers.Dense(8))
model.add(layers.Dense(1))
model.compile(optimizer='adam', loss='mse',metrics=['mae'])
model.summary()
Model: "sequential"
```

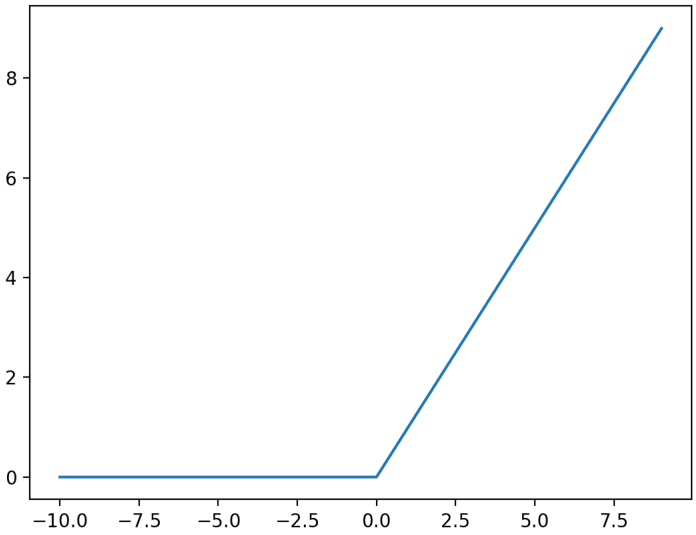
Layer (**type**)	Output Shape	Param #
gru (GRU)	(None, 30, 16)	912
gru_1 (GRU)	(None, 8)	624
dense (Dense)	(None, 32)	288
dense_1 (Dense)	(None, 16)	528
dense_2 (Dense)	(None, 8)	136
dense_3 (Dense)	(None, 1)	9

```
Total params: 2,497
Trainable params: 2,497
Non-trainable params: 0
```

Questo appena riportato è il modello utilizzato per tutte le sequenze temporali della task 1. La struttura comprende 2 layer GRU e 4 layer Dense.
Sequential è la classe che ci permette di aggiungere via via Layer dopo Layer dall' input all'output della sequenza.
Concentrandoci sui due layer GRU, la differenza più importante tra i due è il tipo di output. Mentre il secondo ha in output un tensore bidimensionale, il primo ha in output un tensore tridimensionale. Tale distinzione è stata fatta per via del fatto che l'output di una RNN può essere di due tipi, come appena evidenziato, l'input di una RNN invece deve essere necessariamente un tensore tridimensionale. Questa considerazione nel codice si esprime tramite la funzione "return_sequences = True".
Un'altra differenza molto importante è che il primo layer presenta una funzione di attivazione di tipo "relu" che è una delle funzioni di attivazioni più comunemente utilizzata. Brevemente, una funzione di attivazione viene utilizzata per mappare l'ingresso all'uscita. Questa funzione di attivazione aiuta una rete neurale ad apprendere relazioni e schemi complessi nei dati. La funzione di attivazione scelta "relu" (rectified linear unit) è definita come:

$$f(x) = max(0, x)$$

dove x è l'input del neurone. Graficamente ha la seguente forma:



Successivamente sono stati aggiunti 4 layer "Dense" tra i quali l'unica differenza è il numero di neuroni considerati. Nei layer Dense vengono creati dei neuroni, ognuno dei quali è connesso ad ogni output del layer precedente.
E' stato poi possibile compilare il modello, scegliendo come optimazer "adam".
Un optimazer è un algoritmo di ottimizzazione che permette d'individuare, attraverso una serie d'iterazioni, quei valori dei weight tali per cui la loss function risulti avere il valore minimo.
Adam è un algoritmo di ottimizzazione che può essere utilizzato al posto della classica procedura "Stochastic gradient descent" per aggiornare iterativamente i pesi di rete in base ai dati di addestramento in modo dinamico, al contrario dello Stochastic gradient descent in cui tale processo dinamico non avviene.
Come parte dell'algoritmo di ottimizzazione, l'errore del modello deve essere stimato ripetutamente. Ciò richiede la scelta di una funzione di errore, convenzionalmente chiamata "loss function", che può essere utilizzata per stimare la perdita del modello, in modo che i pesi possano essere aggiornati per ridurre la perdita alla valutazione successiva.
Per il calcolo della loss è stato scelto "loss = mse" dove mse è l'errore quadratico medio, definito come:

$$L(y, \hat{y}) = \frac{1}{N}\sum_{i=0}^{N}(y_i - \hat{y}_i)^2$$

con $\hat{y}$ valore predetto.
Infine la metrica di valutazione dell'errore scelta è "metrics = ['mae']" ossia l'errore assoluto medio, definito come:

$$MAE = \frac{|y_i - \hat{y}_i|}{N}$$

Una volta determinato il modello da utilizzare si è passato al "fit".

```
history=model.fit(x_train, y_train, epochs=num_epoch,
                  validation_data=(x_val, y_val),verbose =1, batch_size=128)
```

In esso vengono forniti in input i dati di train, quindi la time series e il suo relativo vettore dei valori predetti. Oltre a questo vengono fornite il numero di epoche (20 per "x" e "y" e 10 per "z"). Fatto questo, per la validazione del modello, è stato dato in input anche il validation set per osservare l'eventuale presenza di "overfitting" o "underfitting".

3.2 Task 2

Il modello sviluppato ed utilizzato nella seconda task è un autoencoder. L'aspetto più importante che definisce un autoencoder è che i livelli di input contengono esattamente la stessa quantità di informazioni del livello di output. Il motivo per cui il livello di input e il livello di output hanno lo stesso numero di unità è che un autoencoder mira a replicare i dati di input. In output infatti abbiamo una copia dei dati dopo averli analizzati e ricostruiti senza supervisione. Quindi a differenza della task 1, in tale caso è stato sviluppato un apprendimento noto come "unsupervised learning". I dati che si muovono attraverso un autoencoder non vengono mappati direttamente dall'input all'output, quindi la rete non si limita a copiare i dati di input.
Ci sono tre componenti in un autoencoder: una parte di codifica (input) , un componente che gestisce i dati (o collo di bottiglia) e una parte decoder (output). La rete viene quindi addestrata sui dati codificati e produce una ricreazione di quei dati. In tal modo la rete apprende le caratteristiche più importanti dei dati di input. Dopo aver addestrato il modello, questo è in grado di sintetizzare dati simili, con l'aggiunta o la sottrazione di determinate caratteristiche di destinazione. Da tale tipo di descrizione è possibile intuire che gli autoencoder siano degli ottimi modelli per la ricerca di anomalie all'interno di serie temporali.
Il modello realizzato è il seguente:

```
# Definiamo il modello
model = Sequential()
model.add(layers.GRU(64, input_shape=(x_train.shape[1], x_train.shape[2])))
model.add(layers.Dense(64))
model.add(layers.Dense(64))
model.add(layers.Dense(64))
model.add(layers.Dense(64))
model.add(layers.RepeatVector(x_train.shape[1]))
model.add(layers.Dense(64))
model.add(layers.Dense(64))
model.add(layers.Dense(64))
model.add(layers.Dense(64))
model.add(layers.GRU(64, return_sequences=True))
model.add(layers.TimeDistributed(Dense(x_train.shape[2])))
model.compile(optimizer=keras.optimizers.Adam(learning_rate=0.001), loss="mae")
model.summary()
```

Layer (**type**)	Output Shape	Param #
gru (GRU)	(None, 64)	13440
dense (Dense)	(None, 64)	4160
dense_1 (Dense)	(None, 64)	4160
dense_2 (Dense)	(None, 64)	4160
dense_3 (Dense)	(None, 64)	4160
repeat_vector (RepeatVector)	(None, 60, 64)	0
dense_4 (Dense)	(None, 60, 64)	4160
dense_5 (Dense)	(None, 60, 64)	4160

```
_________________________________________________________________
dense_6 (Dense)              (None, 60, 64)            4160
_________________________________________________________________
dense_7 (Dense)              (None, 60, 64)            4160
_________________________________________________________________
gru_1 (GRU)                  (None, 60, 64)            24960
_________________________________________________________________
time_distributed (TimeDistri (None, 60, 4)             260
=================================================================
Total params: 71,940
Trainable params: 71,940
Non-trainable params: 0
_________________________________________________________________
```

Nell' "encoder" abbiamo la presenza di un layer GRU e due layer Dense. Successivamente, viene aggiunto un layer "RepeatVector(x_train.shape[1])". Questo layer ha in input dei tensori bidimensionali di dimensione (samples, number of features) che vengono a generarsi per via della presenza di un layer GRU e restituisce un tensore tridimensionale di dimensione (samples,lookback, number of features). Questo viene fatto per riottenere un tensore tridimensionale. Successivamente abbiamo il "decoder" in cui vengono ripercorsi in maniera inversa tutti i passaggi fatti dall'encoder. Si noti che il layer GRU in tale caso presenta la parte di codice "return_sequence = true" poiché si vuole mantenere un tensore tridimensionale. Infine viene applicato un layer "TimeDistributed" al fine di riottenere un output della stessa dimensione di quello iniziale.
Successivamente è stato possibile passare alla compilazione del modello, è stato scelto lo stesso optimizer e la stessa loss della task 1, l'unica differenza è nel learning rate dell'optimizer, pari a 0.001. Il learning rate è un iperparametro che controlla quanto modificare il modello in risposta all'errore stimato ogni volta che i pesi del modello vengono aggiornati. Variando il learning rate è possibile variare il tasso di apprendimento del modello, ottimizzandolo e rendendolo stabile. La scelta del learning rate è impegnativa in quanto un valore troppo piccolo può comportare un lungo processo di allenamento, mentre un valore troppo grande può comportare l'apprendimento di una serie di pesi non ottimale troppo velocemente o un processo di training instabile.

4 Valutazioni modello

La valutazione del modello in entrambe le task si è incentrata sullo studio della loss function in funzione della validation loss. Si è infatti cercato di ottimizzare il modello analizzando la relazione tra le due, trovando i migliori parametri che minimizzassero entrambe e che evitassero lo svilupparsi di overfitting o underfitting.

4.1 Task 1

In seguito vengono riportati gli andamenti di Training Loss e Validation Loss per le tre sequenze temporali in funzione delle epoche.

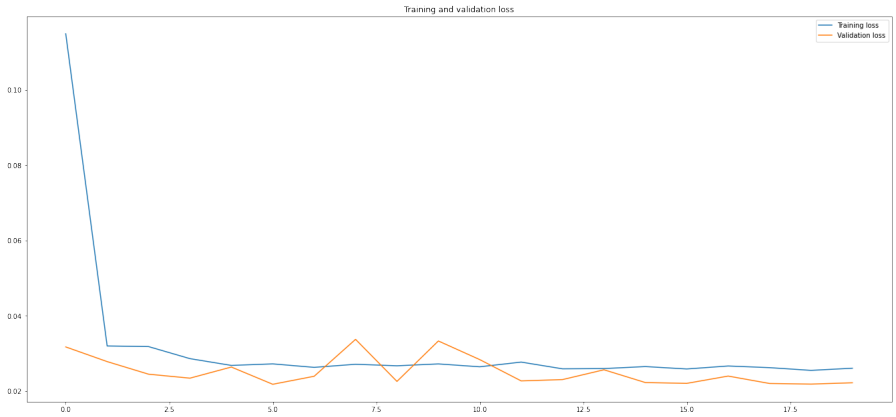


Figure 4.1: Training Loss e Validation Loss per x

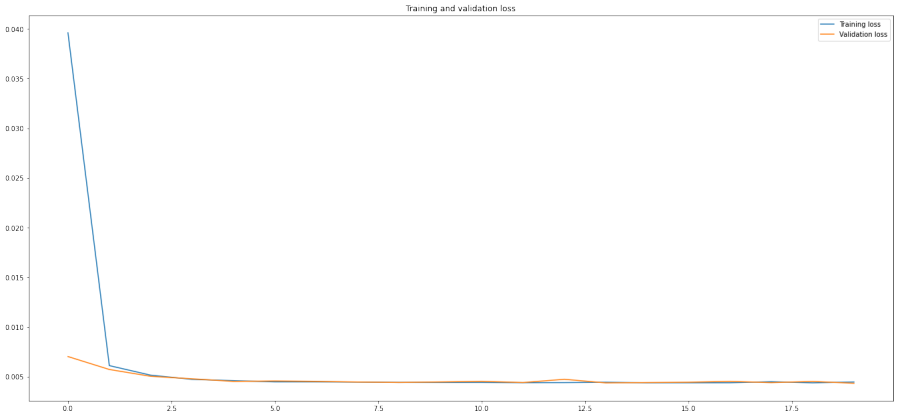


Figure 4.2: Training Loss e Validation Loss per y

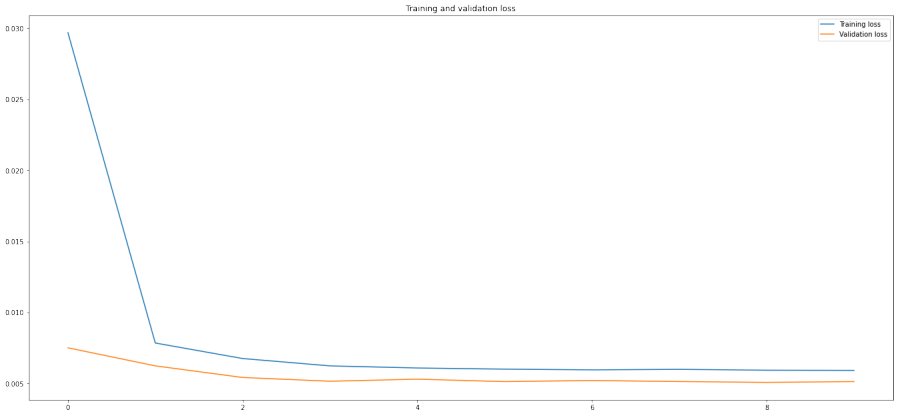


Figure 4.3: Training Loss e Validation Loss per z

In tutti e tre i grafici possiamo notare che l'ottimizzazione dei parametri ha fatto si che training loss e validation loss abbiano lo stesso andamento decrescente, privo di overfitting o underfitting.

Questo implica che il modello apprende e determina i pesi in maniera corretta. Infatti nel caso in cui la training loss fosse stata molto più bassa della validation loss saremmo stati nel caso di overfitting, in cui il modello si adatta troppo ai dati di training, perdendo di generalità e non essendo in grado di adattarsi ad ulteriori dataset, risultando quindi poco efficace. Questo in genere è un problema che si presenta quando all'interno del modello sono presenti troppi parametri o quando la rete è troppo profonda. Se invece la training loss fosse stata troppo alta rispetto alla validation loss, ci saremmo trovati nella situazione diametralmente opposta, quella dell'underfitting. L'underfitting è un problema di apprendimento che si verifica quando la classificazione si basa su pochi parametri o quando il modello utilizzato è troppo poco profondo.
In tutti e tre i grafici è possibile notare che il modello raggiunge dopo pochissime epoche la configurazione ottimale dei pesi e nelle epoche successive, tale configurazione rimane quasi immutata. Questa considerazione è supportata dal fatto che la training loss decresce rapidamente in tutti e tre i modelli dopo una sola epoca, per poi continuare a variare, ma di poco rispetto alla prima decrescita.
Dopo queste considerazioni sui grafici, dal punto di vista del Gradient Descent, è possibile affermare che il modello realizzato raggiunge un minimo locale o assoluto della training loss in funzione dei pesi dopo pochissime epoche.

4.2 Task 2

In seguito vengono riportati gli andamenti di Training Loss e Validation Loss in funzione delle epoche.

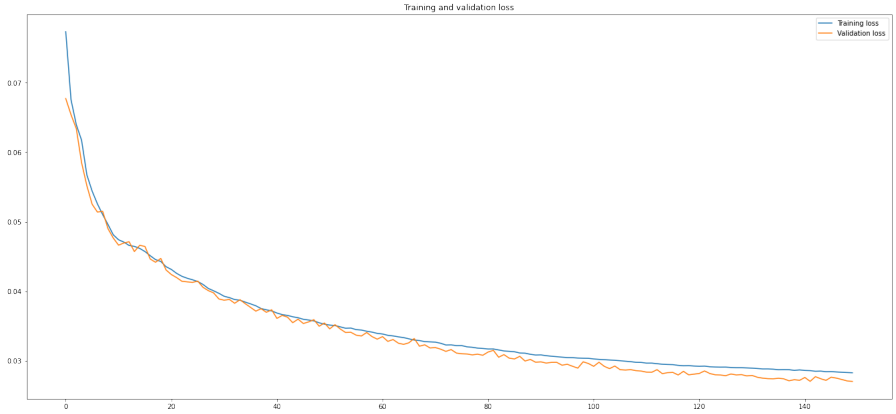


Figure 4.4: Training Loss e Validation Loss

Come possiamo notare tale grafico è molto differente da quelli considerati precedentemente per la task 1.
Innanzitutto ci concentriamo sul fatto che il numero di epoche è nettamente superiore rispetto a prima, sono state considerate 150 epoche. Tale scelta è giustificata dal fatto che, al contrario di prima non abbiamo una convergenza immediata, ma la training loss e validation loss continuano a decrescere, rendendo necessario un incremento delle epoche. Il processo viene arrestato nel momento in cui la pendenza delle due curve di training e validation tende ad essere nulla e di conseguenza l'apprendimento risulta essere molto più lento ed inefficace poiché si è prossimi a un minimo locale.
Osservando l' andamento delle due curve è possibile affermare che il modello utilizzato si adatta correttamente ai dati non mostrando evidenti fenomeni di overfitting e underfitting.

5 Analisi dati e risultati

5.1 Task 1

Una volta realizzato il modello ed effettuato il train, è stato possibile applicarlo alle serie temporali ottenute dai dati di test per prevedere i valori successivi di ogni sequenza. L'analisi dati della prima task può essere suddivisa in tre punti:

- Studio della MAE del test per ottenere valori confrontabili con quelli precedentemente citati;
- Analisi della MAE di una delle serie temporali del test al variare del lookback;
- Analisi della MAE di una delle serie temporali del test al variare del window_shift.

Una volta effettuate le previsioni dei valori successivi di ogni sequenza mediante la seguente riga di codice:

```
previsioni = model.predict(x_test)
```

è stata valutata la MAE del test:

```
differenza_test = np.absolute(previsioni -y_test)*massimo_train
TEST_MAE = sum(differenza_test)/len(differenza_test)
print('TEST_MAE=',TEST_MAE)
```

si noti che prima di calcolare la MAE, i dati di test e le previsioni sono stati riscalati, ossia riportati ai valori precedenti alla normalizzazione. Poiché nella prima riga stiamo già calcolando una differenza, è stato sufficiente moltiplicare per il massimo_train senza dover sommare nuovamente minimo_train, essendo la distanza tra due punti invariante per traslazioni.
Il risultato ottenuto per le tre sequenze è il seguente:

- Test_MAE_x = 81.00
- Test_MAE_y = 83.38
- Test_MAE_z = 79.86

Tutti e tre i valori risultano essere di poco inferiori a quelli di riferimento precedentemente riportati. Di conseguenza in tutti e tre i casi, il modello performa leggermente meglio rispetto a quello di riferimento. Riportiamo per conferma i grafici che mostrano l'andamento dei valori predetti rispetto a quelli effettivi. Anche in tale caso i dati sono stati riportati ai valori precedenti alla normalizzazione. Per riuscire a visualizzare in maniera chiara i grafici, non sono stati stampati tutti i valori predetti ma soltanto i primi 10000 dati. I valori predetti sono rappresentati in arancione tramite dei simboli '+' mentre i valori effettivi sono rappresentati in blu tramite dei simboli 'o'.

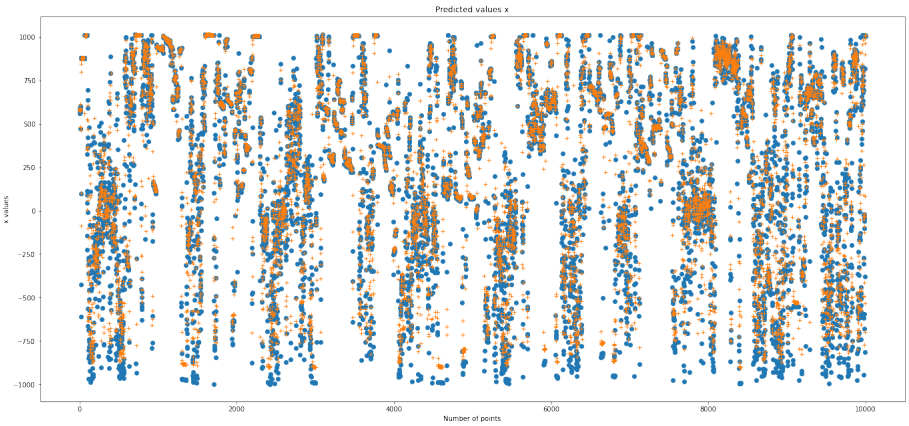


Figure 5.1: Valori predetti e valori effettivi per "x"

Figure 5.2: Valori predetti e valori effettivi per "y"

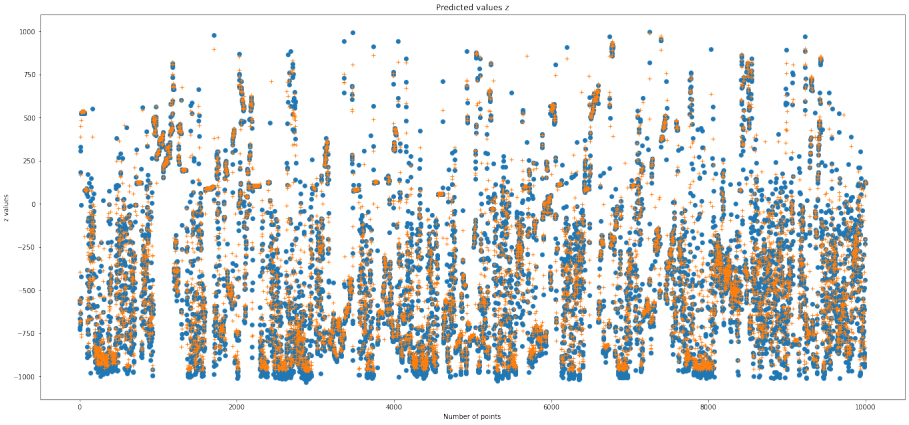


Figure 5.3: Valori predetti e valori effettivi per "z"

da tali grafici è possibile notare che in tutti e tre i casi, i valori predetti si avvicinano molto a quelli effettivi del test. E' lecito affermare che il modello è in grado di apprendere le caratteristiche più

importanti di tali sequenze temporali ed è in grado di prevederne i valori in avanti nel tempo, commettendo un errore medio che si assesta intorno al 5 % (tale stima dell'errore è stata fatta considerando il range a cui appartengono i dati [-1000,1000] e considerando che gli errori medi, quindi la MAE, siano dell'ordine del centinaio).
E' stato poi possibile passare al secondo punto della prima task. Per tale analisi, considerando la "z" e mantenendo fisso il valore del window_shift = 6 è stato determinato il valore della MAE per una serie di valori di lookback. La scelta di mantenere fisso il valore di window_shift è stata effettuata per poter fare un'analisi più chiara. Variando un solo parametro alla volta infatti, si può comprendere in maniera certa se questo sia migliorativo o peggiorativo nel modello, variando più parametri insieme invece non è possibile affermare quali di questi migliorino o peggiorino la previsione.
Riportiamo in tabella i valori della MAE ottenuti in funzione del lookback:

lookback	**TEST MAE**
1	89.92
2	84.08
5	82.91
10	89.46
15	85.99
20	81.51
25	88.58
30	79.86
60	88.56
90	84.53
120	83.03
150	81.39
180	90.13
210	80.55
240	83.78
270	81.00
300	78.01
330	81.65
360	96.30

Table 5.1: MAE in funzione del lookback

Da tale tabella è poi possibile ricavare il seguente grafico:

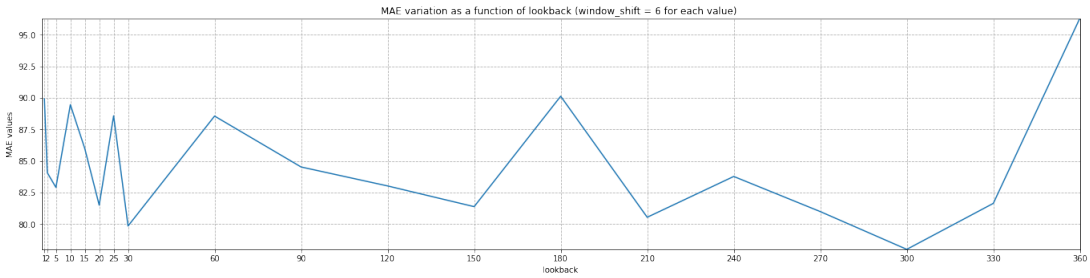


Figure 5.4: MAE in funzione del lookback

in cui, al contrario di quello che intuitivamente ci si aspetta, la MAE non diminuisce all'aumentare del lookback.
E' stato ottenuto il valore di MAE più basso per un lookback pari a 300, ma aumentando tale valore,

la MAE torna ad aumentare.
Intuitivamente si può supporre che considerando sequenze più lunghe il modello ha più dati come riferimento per poter prevedere il successivo migliorando la qualità della previsione, tuttavia questo non avviene. Non è quindi possibile individuare un trend di crescita o decrescita della MAE.
Tale risultato risulta però essere in linea con la letteratura, in particolare è stato preso come riferimento l'articolo [1], in cui sono state effettuate analisi di serie temporali di stock price, al variare del lookback e, analogamente a quanto ritrovato nello studio da noi fatto, anche in tale caso non è possibile definire un trend preciso di salita o discesa al variare del lookback.
L'analisi appena fatta è stata poi ripetuta, seguendo gli stessi criteri, fissando il lookback= 30 e facendo variare il window_shift. Riportiamo in tabella i risultati:

window_shift	**TEST MAE**
1	82.19
2	79.54
3	85.49
4	81.75
5	83.09
6	79.86
7	83.31
8	78.30
9	89.30
10	79.56
11	80.63
12	84.30

Table 5.2: MAE in funzione del window_shift

E il relativo grafico è:

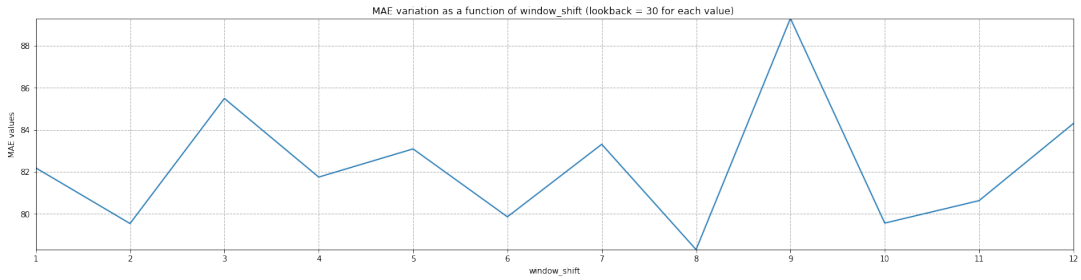


Figure 5.5: MAE in funzione del window_shift

Anche in tale caso non è possibile definire un trend di salita o di discesa in funzione del window_shift. E' quindi possibile estendere le considerazioni fatte sul lookback anche al window_shift.

Per far si che ogni misura sia riproducibile, riottenendo i valori riportati nelle tabelle precedenti, è stato necessario aggiungere una parte di codice che permetta di fissare i pesi iniziali che vengono definiti all'interno del modello.

```
os.environ['PYTHONHASHSEED'] = '0'
os.environ['CUDA_VISIBLE_DEVICES'] = ''
```

[1] "Analysis of look back period for stock price prediction with RNN variants: A case study on banking sector of NEPSE" Arjun Singh Sauda, Subarna Shakya, 2020.

```
np.random.seed(37)
rn.seed(1254)
tf.random.set_seed(89)
```

I pesi infatti, in generale vengono inizializzati in maniera randomica e il fitting del modello parte da questi per minimizzare la loss e la validation loss. Per tale ragione, considerando dei valori randomici il modello potrebbe convergere verso minimi locali differenti ogni volta che viene eseguito ottenendo quindi valori diversi.

5.2 Task 2

Per la seconda task ci si è concentrati sulla ricerca di anomalie all'interno dei dati "x","y","z" e "HeartRate".
Si è partiti applicando il modello ai dati di train e ricavandone la MAE come:

```
# Calcoliamo la MAE di train
x_train_pred = model.predict(x_train)
#Viene calcolata su ogni step temporale,
#come differenza tra i dati ricostruiti e quelli reali
train_mae_loss = (x_train_pred - x_train).reshape(-1,lookback*4)
train_mae_loss = np.mean(np.abs(train_mae_loss), axis=1)
train_mae_loss.shape
```

tale operazione è stata ripetuta per ogni serie numerica sia per il train che per il test. Abbiamo la presenza di un reshape nel calcolo della MAE, in tal modo al rigo di codice successivo viene calcolata la media su ogni istante temporale. In questo modo, ad ogni rigo del dataset viene associata una MAE univoca e non 4 differenti, legate a "x","y","z" e "HeartRate".
Ovviamente, avendo diviso ogni dataset di test in serie temporali più piccole, considerando singolarmente i pazienti, la MAE è stata calcolata per ognuna di esse. Il dataset di test è costituito da 5 pazienti, quindi sono state calcolate 5 MAE.
E' stato poi determinato un valore di soglia in funzione del massimo della MAE di train. Il suddetto valore risulta fondamentale per determinare le anomalie. Queste infatti vengono definite come tutti gli elementi del test la cui MAE risulta essere superiore alla soglia.
Viene poi generata una lista nella quale sono inseriti gli indici del dataset di test identificati come anomalia.
Fatto ciò, è possibile ricavare un dataset con 7 colonne, le quali presentano in ogni rigo gli indici legati alle anomalie ed i relativi dati. Consideriamo ora i valori di MAE del train.

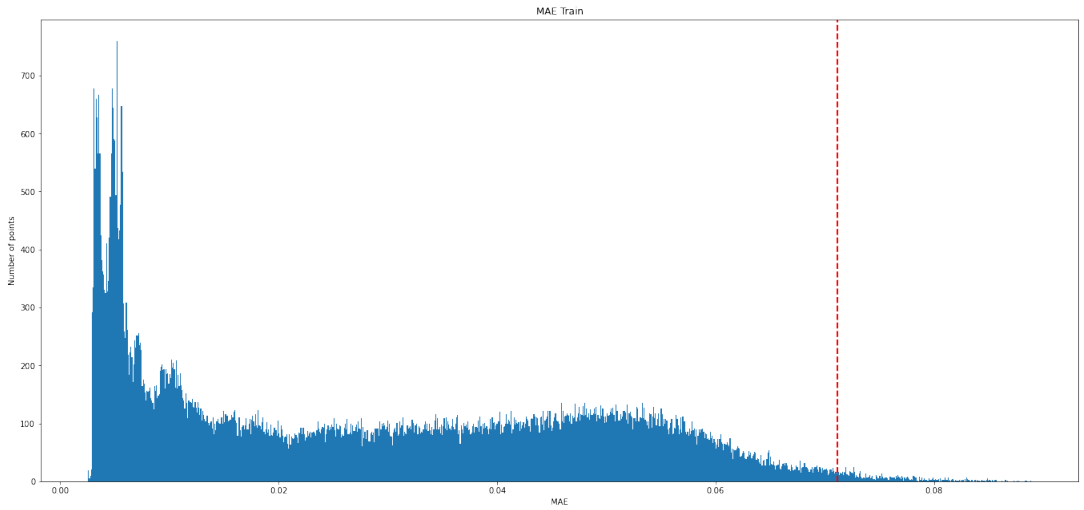


Figure 5.6: Istogramma della MAE dei dati di train e threshold; MAE (blu); threshold (rosso);

La scelta del threshold in questo tipo di analisi non segue un criterio fisso, ma dipende principalmente dal caso analizzato nel dettaglio e dai dati considerati. La nostra scelta è basata sulla MAE relazionata alla distribuzione dell'errore nell' istogramma. Si è ipotizzato che i valori associati alle anomalie non possano essere ricostruiti in maniera precisa e che quindi siano caratterizzati da un valore di MAE più alto. E' ovvio che i migliori modelli per tale tipo di ricerca, siano quelli che minimizzano la MAE del train. Considerando i valori privi di anomalie come riferimento, è possibile scegliere il threshold. Per omogeneità è stato scelto lo stesso criterio per ogni soglia (il codice però è stato scritto in modo da poter scegliere una soglia differente per ogni datasset di ogni paziente), pari all' 80% del massimo della MAE del train e non al 100%. Il modello, non essendo in grado di ricostruire perfettamente i dati privi di anomalie, presenta esso stesso un'imprecisione intrinseca, la quale viene compensata considerando un intervallo di confidenza del 20%. Si è scelto il 20% poiché al di sotto di tale valore sono presenti la quasi totalità dei valori di train e il numero di dati al di sopra di tale threshold, è trascurabile. Diminuendo ulteriormente il threshold, il numero di elementi ricostruiti omessi non risulta essere più trascurabile.
Prendiamo come esempio il paziente 3006. Ne riportiamo innanzitutto la MAE:

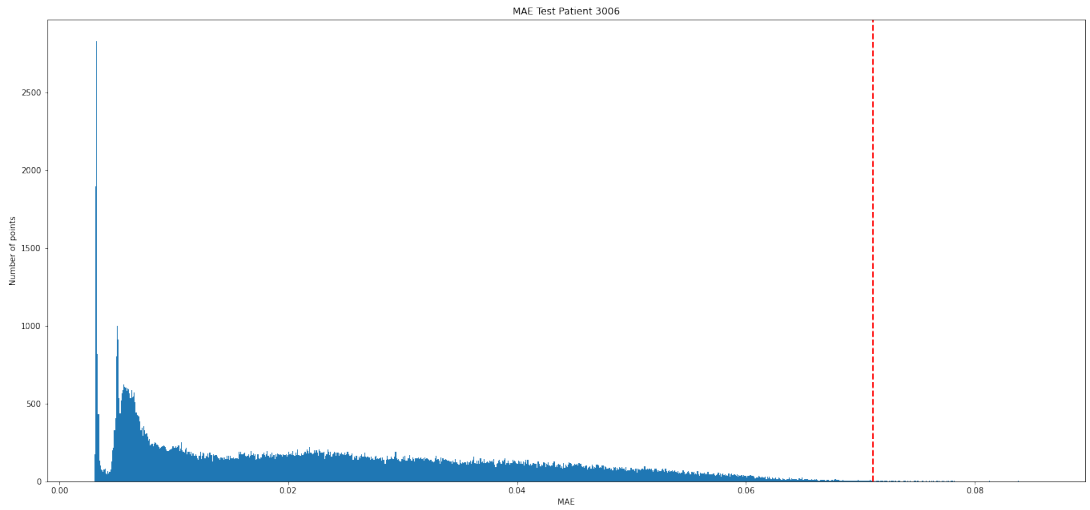


Figure 5.7: Istogramma della MAE del paziente 3006 e threshold; MAE (blu); threshold (rosso);

Il valore di Threshold considerato è circa pari a 0.0711.E' possibile notare che è sufficientemente grande da comprendere la quasi totalità dei valori ricostruiti alla sua sinistra e rendendo ragionevole l'assunzione fatta sull'intervallo di confidenza pari al 20%.
Come spiegato in precedenza, è stato poi possibile determinare gli indici dei valori che presentano una MAE superiore a quella di soglia mediante le seguenti righe di codice:

```
threshold = np.max(train_mae_loss)*0.8
anomalies = test_mae_loss > threshold

anomalous_data_indices = []

for data in range (0, len(test_mae_loss)-1):
  if(anomalies[data]==True ):
    anomalous_data_indices.append(data)

anomalies =  test_x.iloc[anomalous_data_indices_0]
```

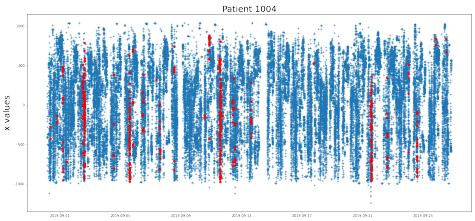
In "anomalies" consideriamo un vettore con dimensione pari a test_mae_loss, in cui in ogni riga sono presenti "True" se la colonna è associata ad un'anomalia, "False" se non lo è. Nel "for" vengono salvati in una lista gli indici associati ai valori "True". Infine in "anomalies" riportiamo le righe di test anomale ed i relativi valori.
In seguito a tale passaggio sono stati realizzati dei grafici contenenti i dati di test e le relative anomalie. Tali grafici sono riportati nelle pagine seguenti.
La trattazione appena fatta è stata ripetuta per tutti gli altri pazienti, procedendo nello stesso modo. Per tale ragione ne è stato analizzato esplicitamente solo uno mentre degli altri riportiamo soltanto i risultati.
La parte finale del codice è dedicata a visualizzare i dataset delle anomalie. Per far si che questi dati siano confrontabili con quelli inizialmente forniti, innanzitutto sono stati riscalati e riportati agli ordini di grandezza iniziali. In un secondo momento si è andato a lavorare sugli indici dei dataset, i quali partono tutti da zero per ogni paziente. Per far si che le anomalie corrispondano ai valori del dataset iniziale gli indici sono stati traslati opportunamente. Tali risultati sono stati salvati in un file e, per come sono stati rielaborati, possono essere confrontati direttamente con i dati di test iniziali senza ulteriori modifiche.
Riportiamo innanzitutto una tabella riassuntiva che comprende il numero di anomalie rilevate per ogni

paziente:

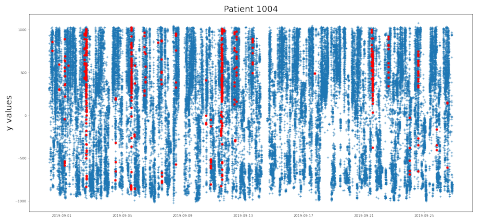
	Paziente 1004	**Paziente 1006**	**Paziente 3001**	**Paziente 3006**	**Paziente 4002**
Anomalie	556	1069	492	175	2639

Table 5.3: Numero di anomalie per ogni paziente

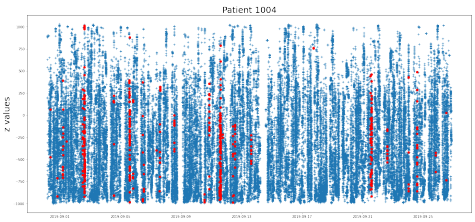
Da tale tabella è subito possibile identificare il paziente che risponde meglio alle cure e quello che risponde peggio. Infatti a numero di anomalie maggiore, corrisponde maggior tempo in cui le cure non sono efficaci. Il paziente con meno anomalie, in seguito alle analisi effettuate, risulta essere il 3006, mentre il paziente con maggior numero di anomalie risulta essere il 4002.
E' infine possibile rappresentare tutti i grafici dei dati di test con le relative anomalie al fine di osservare l'eventuale presenza di OFF periods.



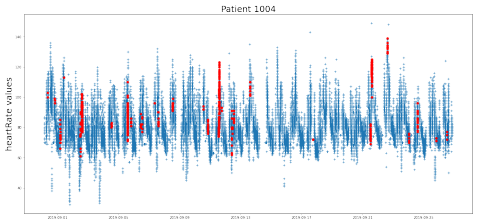
(a) Plot di "x" con le relative anomalie



(b) Plot di "y" con le relative anomalie

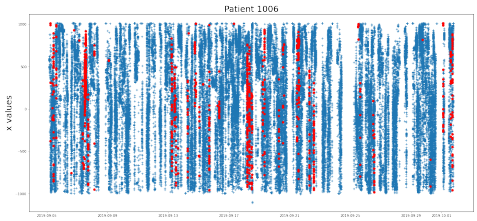


(c) Plot di "z" con le relative anomalie

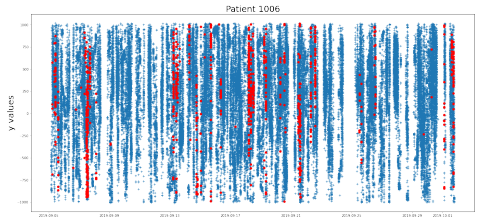


(d) Plot di "HeartRate" con le relative anomalie

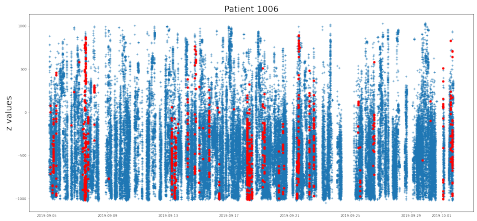
Figure 5.8: Paziente 1004: Dati di train (blu, rappresentati da simboli '+'); threshold (rosso, rappresentati da simboli 'o');



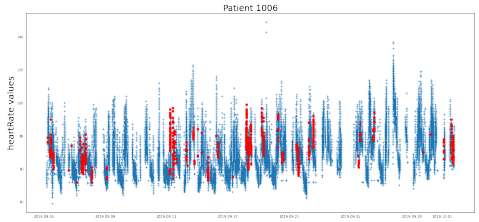
(a) Plot di "x" con le relative anomalie



(b) Plot di "y" con le relative anomalie

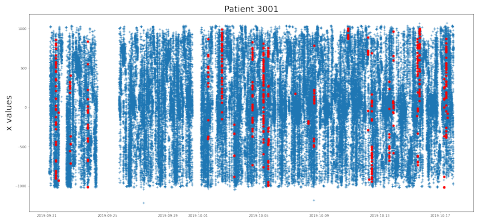


(c) Plot di "z" con le relative anomalie

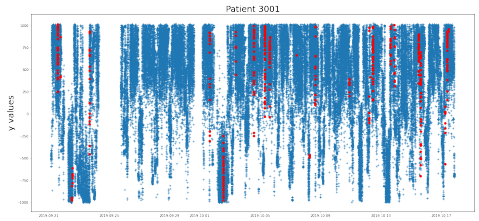


(d) Plot di "HeartRate" con le relative anomalie

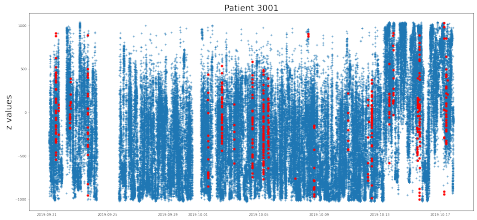
Figure 5.9: Paziente 1006: Dati di train (blu, rappresentati da simboli '+'); threshold (rosso, rappresentati da simboli 'o');



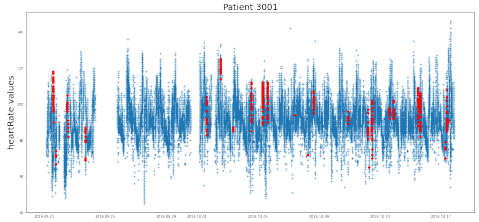
(a) Plot di "x" con le relative anomalie



(b) Plot di "y" con le relative anomalie

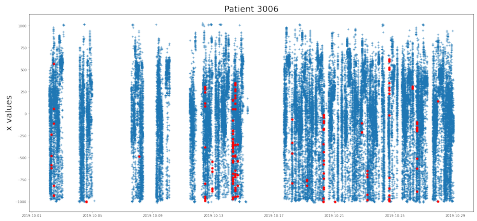


(c) Plot di "z" con le relative anomalie

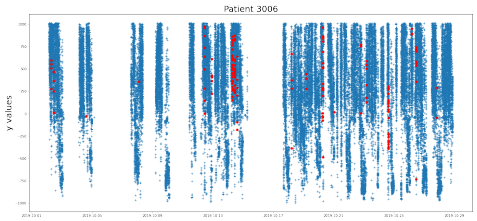


(d) Plot di "HeartRate" con le relative anomalie

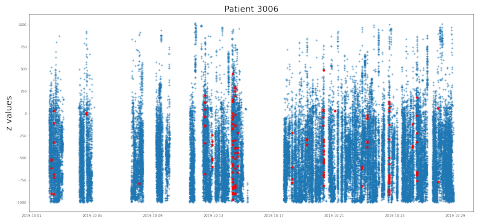
Figure 5.10: Paziente 3001: Dati di train (blu, rappresentati da simboli '+'); threshold (rosso, rappresentati da simboli 'o');



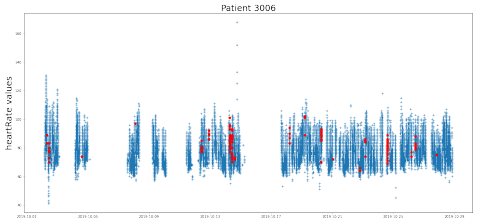
(a) Plot di "x" con le relative anomalie



(b) Plot di "y" con le relative anomalie

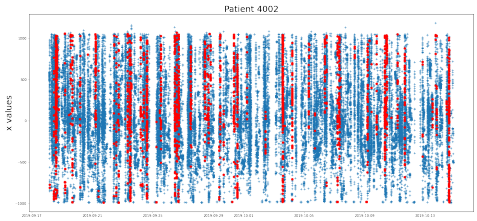


(c) Plot di "z" con le relative anomalie

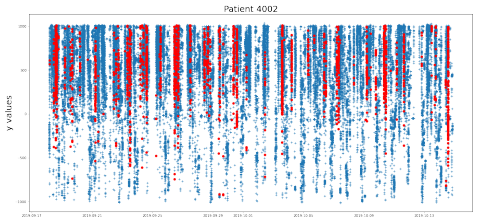


(d) Plot di "HeartRate" con le relative anomalie

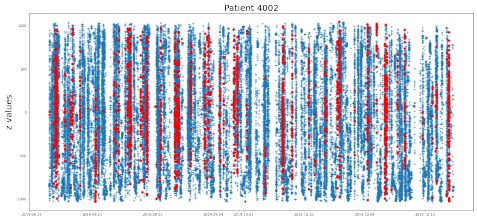
Figure 5.11: Paziente 3006: Dati di train (blu, rappresentati da simboli '+'); threshold (rosso, rappresentati da simboli 'o');



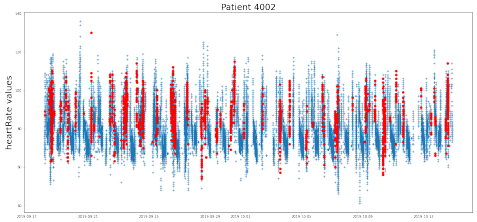
(a) Plot di "x" con le relative anomalie



(b) Plot di "y" con le relative anomalie



(c) Plot di "z" con le relative anomalie



(d) Plot di "HeartRate" con le relative anomalie

Figure 5.12: Paziente 4002: Dati di train (blu, rappresentati da simboli '+'); threshold (rosso, rappresentati da simboli 'o');

Effettuando l'analisi dati di tali grafici è possibile fare delle considerazioni generali che riguardano tutti i pazienti. Essendo la scelta del threshold arbitraria, non è possibile determinare quanti di questi possano essere considerati effettivamente OFF periods. In tale tipo di analisi infatti, avendo questo grado di libertà in più, è sempre legittimo chiedersi se un'anomalia sia associabile ad un OFF period o a un valore di threshold troppo basso. Nonostante la scelta della soglia sia stata fatta dopo un'analisi accurata, non è possibile affermare con sicurezza che sia la scelta più opportuna.

6 Conclusioni

6.1 Task 1

Ricapitolando, innanzitutto i dati di test e di train sono stati opportunamente elaborati, normalizzandoli, realizzando delle time series con window_shift= 6 e lookback=30 e i relativi vettori inerenti al dato successivo da predire. Successivamente è stato realizzato e allenato un modello che presenta 2 RNN e 4 layer Dense. Quest'ultimo, applicato ai dati di train, ci ha permesso di effettuare le previsioni di ogni sequenza.
E' possibile osservare che il modello è in grado di apprendere le principali caratteristiche di ogni serie temporale di train dopo pochissime epoche, trovando già alla seconda epoca, i pesi ottimali. Analizzando anche i grafici di training loss e validation loss si osserva che al termine di ogni fit, non sono presenti underfitting e overfitting. Come conseguenza diretta di ciò, si può dedurre che numero di livelli e il numero di parametri nel modello siano stati scelti correttamente, poichè diminuendo la profondità del modello o diminuendo il numero di neuroni ci si sarebbe potuti imbattere in fenomeni di underfitting e nella configurazione opposta, si sarebbe potuto rischiare overfitting.
Determinando infine le MAE per ogni colonna del dataset di test, si è osservato che queste siano leggermente inferiori a quelle di riferimento fornite. Quest'ultima evidenza è quella che permette di affermare che la trattazione effettuata possa essere ritenuta soddisfacente.
Come secondo punto, è stato valutato il modo in cui lookback e window_shift modificano le prestazioni del modello e quindi la MAE di test.
Per una comprensione più chiara di come questi due parametri influenzino i risultati, si è scelto di fissare uno dei due e far variare l'altro. In tal modo, una simulazione migliore o peggiore può essere attribuita ad un unico parametro.
I risultati in entrambi i casi hanno mostrato che non è possibile identificare dei trend di salita o di discesa. Tale affermazione è anche supportata da un recente articolo in cui si ottengono le stesse conclusioni. Quindi non si è in grado di affermare con certezza quali possano essere i valori migliori per window_shift e lookback se non effettuando delle prove.

6.2 Task 2

Come nella prima task, anche in questo caso i dati di train e test sono stati opportunamente trattati. Dopo aver eliminato le righe associate ad "HeartRate" pari a "-1", i dati di train sono stati rielaborati al fine di ottenere intervalli di campionamento pari a 10 secondi. Questo è stato fatto effettuando delle medie ogni 10 secondi.
Gli obiettivi principali della seguente task non si incentravano sul prevedere il valore successivo di una sequenza, bensì sul determinare delle anomalie associate agli OFF periods nei dati di test.
Per far ciò è stato realizzato un modello utilizzando un autoencoder contenente layer GRU e Dense. La scelta di utilizzare un autoencoder dipende dalla tipologia di dati considerati. Infatti, i dati di train sono assunti privi di anomalie, quindi utilizzando quest'ultimi è possibile, in linea di principio, realizzare un modello in grado di ricostruire correttamente solo dati non anomali.
In seguito a tale considerazione il modello è stato utilizzato per ricostruire i dati di train, per poter scegliere il valore di threshold. L'ultima operazione effettuata è stata quella di applicarlo sui dati di test, salvando in una lista tutti i valori che ricostruiti presentano una MAE superiore al valore di threshold.
In tale contesto è stato finalmente possibile determinare gli OFF periods.
La trattazione per tale problema e le considerazioni da fare sono più elaborate rispetto alla prima task. In particolar modo per l'analisi dati finale uno dei punti cruciali per determinare gli OFF periods è comprendere quali siano i valori di soglia ottimali. Tale scelta quindi, nonostante richieda uno studio dei dati di train e di test, non è univoca. La diretta conseguenza di ciò è che non è possibile affermare se un'anomalia, una volta determinata, sia effettivamente associabile a un OFF period, a una cattiva ricostruzione del modello o a una scelta inopportuna del Threshold.

Nonostante tali considerazioni, è comunque possibile osservare dai grafici che in ogni paziente le anomalie si addensano in periodi ben definiti e non sono diffuse in maniera omogenea.
Questo permette di supporre che il modello sia comunque in grado di comprendere periodi ben definiti in cui le cure sui pazienti non risultino essere particolarmente efficaci.