UNIVERSITÀ DELLA CALABRIA
DIPARTIMENTO DI MATEMATICA E INFORMATICA

PROGETTO

DEEP LEARNING

*GRUPPO **brAIns***

Marco Bellizzi 223966
Domenico Spagnolo 227805
Giuseppe Oliva 223956

Sommario

TASK 1.1: Next value prediction **3**
1. Introduzione 3
2. Data understanding e preparation 3
3. Modeling 5
4. Evaluation 7
5. Conclusioni 7

TASK 1.2: Analysis of the effects of changes of window size and window shiftt **8**

TASK 2: Anomaly detection **10**
1. Introduzione 10
2. Data understanding e preparation 12
3. Modeling 15
4. Evaluation 17
5. Conclusioni 19

TASK 1.1: Next value prediction

1. Introduzione

L'obiettivo del task 1.1 è quello di predire i valori successivi di una sequenza di dati. Sono fornite 3 serie temporali (X, Y, Z) registrate ogni 10 secondi. Data una sequenza di 5 minuti ogni minuto, l'obiettivo è quello di predire il valore successivo di ogni sequenza per ciascuna serie.

2. Data understanding e preparation

Dopo aver caricato il dataset di test (test.csv) e di training (train.csv) forniti in input, si è proceduto a normalizzare il training set. Attraverso questo procedimento abbiamo ricondotto una variabile aleatoria distribuita secondo una media μ e varianza σ^2, ad una variabile aleatoria con distribuzione "standard", ossia di media 0 e varianza pari a 1. Il procedimento prevede di sottrarre alla variabile aleatoria la sua media e dividere il tutto per la deviazione standard.

```
test = pd.read_csv(os.path.join('DeMaCS_Project_Data/task1_next_value_prediction', 'test.csv'), sep = ',')
train = pd.read_csv(os.path.join('DeMaCS_Project_Data/task1_next_value_prediction', 'train.csv'), sep = ',')

train_mean = train.mean()
train_std = train.std()

train = (train - train_mean) / train_std
train
```

	x	y	z
0	-0.318759	1.317360	-0.591473
1	-0.688905	1.621573	0.454343
2	-0.552642	1.340891	-0.500452
3	-1.292934	0.799694	-0.463301
4	-0.835336	1.243408	-0.164231
...	...	...	...
144906	0.866929	-1.291140	0.025242
144907	0.866929	-1.291140	0.025242
144908	0.866929	-1.289460	0.023385
144909	0.866929	-1.291140	0.025242
144910	0.866929	-1.289460	0.025242

Successivamente abbiamo estratto le sequenze. A tal proposito è stata definita la funzione `extract_seq()` che riceve come parametri:

- `data_set`: il dataset da cui estrarre le serie temporali;
- `max_len`: lunghezza delle serie temporali;
- `step`: ogni quanto avviene l'estrazione della singola serie temporale;
- `axis`: specifica dell'asse (x, y, z).

Questa funzione estrae sotto-sequenze monodimensionali e colleziona il relativo valore successivo, che rappresenta il valore che vogliamo predire. L'estrazione è stata fatta su ogni singolo asse con `max_len=30` e `step`=6 (window size = 30 e window shift = 6) in entrambi i dataset forniti.

```
def extract_seq(data_set, max_len, step, axis):
  print("extracting sequences")
  sequences = []
  values = []
  for i in range(0, len(data_set) - max_len - 1, step):
    print("\r" + str(int((i+1) / len(data_set) * 100)) + " %", end="")
    sequences.append(data_set[axis][i: i + max_len])
    values.append(data_set[axis][i + max_len])
  sequences = np.array(sequences)
  sequences = sequences.reshape((sequences.shape[0], sequences.shape[1], 1))
  values = np.array(values)
  values = values.reshape((values.shape[0], 1))
  print("\r100 %")
  return sequences, values
```

```
sequences_train_x, values_train_x = extract_seq(train, 30, 6, 'x')
sequences_train_y, values_train_y = extract_seq(train, 30, 6, 'y')
sequences_train_z, values_train_z = extract_seq(train, 30, 6, 'z')

sequences_test_x, values_test_x = extract_seq(test, 30, 6, 'x')
sequences_test_y, values_test_y = extract_seq(test, 30, 6, 'y')
sequences_test_z, values_test_z = extract_seq(test, 30, 6, 'z')
```

3. Modeling

In questa fase abbiamo scelto il tipo di reti neurali più adatte al nostro scopo. Nel nostro caso la rete neurale è composta da un layer LSTM (Long short-term memory) con la stessa forma di input delle sequenze estratte e un layer Dense con un solo nodo senza funzione di attivazione allo scopo di effettuare la predizione di un solo valore. Una LSTM è una rete neurale sequenziale, che consente alle informazioni di persistere. È in grado di gestire il problema del vanishing gradient. Come funzione di loss abbiamo usato la **Mean Absolute Error (MAE)**, che è uguale al rapporto tra la differenza in valore assoluto del valore reale e il valore predetto e il numero di osservazioni. In generale, una funzione di loss misura il grado di accuratezza con cui un certo modello statistico descrive un set di dati di un certo fenomeno. Come ottimizzatore è stato scelto Adam, un metodo di discesa del gradiente stocastico utilizza Momentum e Adaptive Learning Rates (RMSprop) per convergere più velocemente.

$$\text{MAE} = \frac{|(y_i - y_p)|}{n}$$

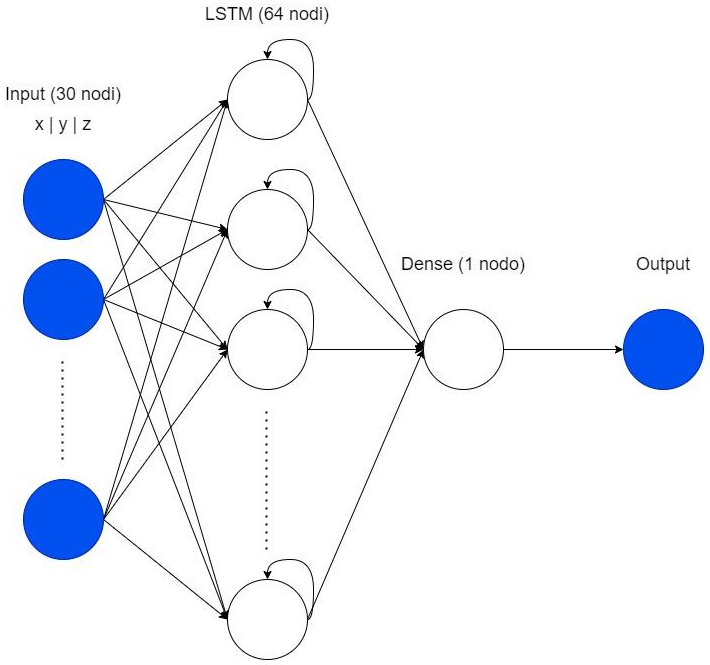
y_i = actual value
y_p = predicted value
n = number of observations/rows

Infine, abbiamo usato le sequenze di training per addestrare il modello e quelle di test per valutarlo.

```
model_x = Sequential()
model_x.add(layers.LSTM(64, input_shape=(sequences_train_x.shape[1], sequences_train_x.shape[2])))
model_x.add(layers.Dense(1))
model_x.compile(loss='mean_absolute_error', optimizer = 'adam')
history_x = model_x.fit(sequences_train_x, values_train_x, epochs=5)
```

```
model_y = Sequential()
model_y.add(layers.LSTM(64, input_shape=(sequences_train_x.shape[1], sequences_train_x.shape[2])))
model_y.add(layers.Dense(1))
model_y.compile(loss='mean_absolute_error', optimizer = 'adam')
history_y = model_y.fit(sequences_train_y, values_train_y, epochs=5)
```

```
model_z = Sequential()
model_z.add(layers.LSTM(64, input_shape=(sequences_train_x.shape[1], sequences_train_x.shape[2])))
model_z.add(layers.Dense(1))
model_z.compile(loss='mean_absolute_error', optimizer = 'adam')
history_z = model_z.fit(sequences_train_z, values_train_z, epochs=5)
```


LSTM (64 nodi)
Input (30 nodi)
x | y | z
Dense (1 nodo)
Output

4. Evaluation

In questa fase abbiamo valutato la qualità della reta neurale. La valutazione è stata fatta calcolando la mean absolute error delle sequenze estratte dal test set. Abbiamo provveduto a normalizzare i dati relativi alla sequenza usando la media e la deviazione standard usati per il training set. Abbiamo poi effettuato la predizione utilizzando la funzione `predict()`, il quale risultato è stato poi denormalizzato.

A tal proposito è stata definita la funzione `get_error()` che riceve come parametri:

- `sequences`: le sequenze sulle quali effettuare la predizione.
- `values`: valori reali da predire;
- `axis`: asse di riferimento (x, y, z);
- `model`: modello di riferimento (model_x, model_y, model_z).

```
def get_error(sequences, values, axis, model):
  print("getting error")
  error = 0
  for i in range(sequences.shape[0]):
    print("\r" + str(int((i+1) / sequences.shape[0] * 100)) + " %", end="")
    prediction = model.predict((sequences[[i]] - train_mean[axis]) / train_std[axis])
    prediction = (prediction * train_std[axis]) + train_mean[axis]
    error += abs(prediction[0][0] - values[i][0])
  print("\r100 %")
  return error / sequences.shape[0]
```

```
# x=81.06; y=85.26; z=79.94
print("errore sull' asse x = " + str(get_error(sequences_test_x, values_test_x, 'x', model_x)))
print("errore sull' asse y = " + str(get_error(sequences_test_y, values_test_y, 'y', model_y)))
print("errore sull' asse z = " + str(get_error(sequences_test_z, values_test_z, 'z', model_z)))
```

```
getting error
100 %
errore sull' asse x = 75.72323637960645
getting error
100 %
errore sull' asse y = 81.05911578277306
getting error
100 %
errore sull' asse z = 73.73348487640465
```

5. Conclusioni

In conclusione, ci possiamo ritenere soddisfatti dei risultati ottenuti poiché gli errori ottenuti sui tre assi (x=75.72; y=81.06; z=73.73) sono al di sotto delle soglie di errore massime (x=81.06; y=85.26; z=79.94).

TASK 1.2: Analysis of the effects of changes of window size and window shiftt

L'obiettivo del task 1.2 è di analizzare come varia l'errore variando i parametri di window size e di window shift. A tal proposito abbiamo riutilizzato le funzioni di estrazione delle sequenze e del calcolo dell'errore create in precedenza, cambiando i valori di window size e window shift per l'estrazione delle sequenze. Per vedere come varia l'andamento dell'errore sono stati testati diversi parametri per window size (3, 6, 12) e window shift (15, 30, 60) mentre per quanto riguarda il modello di rete neurale abbiamo mantenuto il precedente.

```
labels = []
errors = []

for step_ in [3, 6, 12]:
  for maxlen_ in [15, 30, 60]:
    print("\nstep " + str(step_) + " - maxlen " + str(maxlen_) + "\n")

    sequences_train, values_train = extract_seq(train, maxlen_, step_, 'x')
    sequences_test, values_test = extract_seq(test, maxlen_, step_, 'x')

    model = Sequential()
    model.add(layers.LSTM(64, input_shape=(sequences_train.shape[1], sequences_train.shape[2])))
    model.add(layers.Dense(1))
    model.compile(loss='mean_absolute_error', optimizer = 'adam')
    model.fit(sequences_train, values_train, epochs=5)

    labels.append(str(step_) + " " + str(maxlen_))
    errors.append(get_error(sequences_test, values_test, 'x', model))
```

In seguito, si è proceduto a visualizzare graficamente l'andamento dell'errore:

```
plt.plot(labels, errors, 'r', label='Error')
plt.title('Variation of errors')
plt.xlabel('Parameter (step - maxlen)')
plt.ylabel('Error')
plt.legend()
plt.show()
```

Con il seguente risultato:

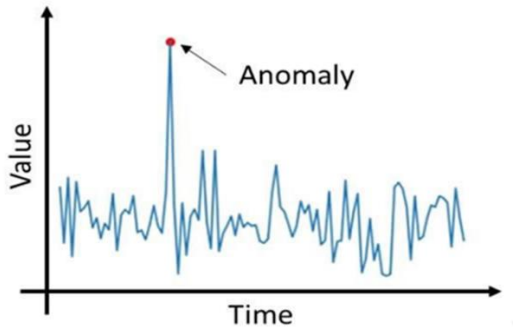
Dove si evince che le migliori scelte per window shift e window size sono rispettivamente 12 e 30. Con questi parametri, infatti, l'errore è il più basso.

TASK 2: Anomaly detection

1. Introduzione

L'obiettivo del task 2 è di identificare periodi off in pazienti affetti da Parkinson. Il Parkinson è una malattia cerebrale che causa una lenta perdita di controllo dei movimenti. Può causare tremori, contratture muscolari, movimenti rallentati e problemi di equilibrio. In molti soggetti, può anche causare problemi cognitivi o demenza (la memoria e la capacità di apprendimento peggiorano col tempo). La malattia di Parkinson è causata da danni al cervello che possono impattare sul controllo del movimento e dell'equilibrio. Generalmente, il sintomo più comune è il tremore (di una parte del corpo che il soggetto non riesce a controllare). Il manifestarsi di questi sintomi è considerato un periodo off.

Il processo di **Anomaly Detection** consiste nel riconoscimento di condizioni o circostanze inattese all'interno di un dataset, cioè un qualcosa che differisce dalla norma.



Nel nostro caso, l'anomaly detection consiste nell'identificare i periodi di off nei soggetti affetti da Parkinson. L'analisi di un dataset reale può fornire importanti indicazioni circa i soggetti analizzati. Il dataset fornito è stato estratto da un dispositivo indossabile, dotato di accelerometro, in grado di raccogliere diverse informazioni dei pazienti. In particolare, il dataset di training riporta circa 944 mila record mentre il dataset di test riporta circa 723 mila record.

I dataset riportano 7 attrubuti:

- **patient**: codice identificativo del paziente;
- **timestamp**: timestamp della rilevazione;
- **tsDate**: data e ora della rilevazione;
- **x, y, z**: variabili dell'accelerometro;
- **heartRate**: battito cardiaco.

Per identificare le anomalie abbiamo usato un autoencoder che prova a replicare sequenze consecutive di osservazioni. **Ad ogni osservazione è associata una sequenza delle sue 30 osservazioni successive**. La sequenza è considerata anomala se ha un alto errore di ricostruzione; Dunque, l'autoencoder non è riuscito a predirla bene. L'errore di una sequenza è stato calcolato come la somma dei valori assoluti delle differenze tra i valori reali e i valori predetti di ogni osservazione nella sequenza. Successivamente per ogni paziente è stata individuata una soglia di errore in funzione della lista di errori di ogni osservazione, e abbiamo considerato anomala una osservazione che ha un volare di errore più alto rispetto alla relativa soglia. Infine, abbiamo riportato le osservazioni anomale di ogni paziente in un file csv.

2. Data understanding e preparation

A differenza del Task 1 in cui i dati forniti sono già stati preventivamente campionati a 10 secondi, in questo caso i dati del test set hanno un intervallo temporale di 10 secondi mentre quelli del training hanno un intervallo temporale di 1 secondo. Inoltre, i dati del training set contengono dei valori nulli sull'attributo heart rate (-1). Abbiamo dunque provveduto a ricampionare i dati del training set a 10 secondi, aggregando 10 record alla volta in uno, calcolando la media dei valori, escludendo dal calcolo i record che contengono valori nulli dell'heart rate. Abbiamo poi rimosso sia dal training set che dal test set le colonne **patient**, **tsDate** e **timestamp** non utili al fine dell'addestramento del modello. Infine, abbiamo normalizzato il training set.

Nello specifico si è proceduto a:

- Campionare a 10 secondi il training set

```
train_ad_sampled = []
cont = 0
cont_not_null = 0
x = 0
y = 0
z = 0
heart = 0

for line in train_ad:
    cont += 1
    values = line.split(",")

    if not cont == 1 and not int(values[4]) == -1:
        cont_not_null += 1
        x += int(values[1])
        y += int(values[2])
        z += int(values[3])
        heart += int(values[4])

    if cont % 10 == 0:
        if cont_not_null > 0:
            train_ad_sampled.append([str(x / cont_not_null), str(y / cont_not_null), str(z / cont_not_null), str(heart / cont_not_null)])
        x = 0
        y = 0
        z = 0
        heart = 0
        cont_not_null = 0
```

- Normalizzare il training set

```
train_ad_sampled_mean = train_ad_sampled.mean()
train_ad_sampled_std = train_ad_sampled.std()

train_ad_sampled = (train_ad_sampled - train_ad_sampled_mean) / train_ad_sampled_std
train_ad_sampled
```

- Importare il test set

```
test_ad = pd.read_csv(os.path.join('DeMaCS_Project_Data/task2_anomaly_detection', 'ad_test.csv'), sep = ',')
test_ad
```

- Fare lo split del test set raggruppando per **patient**

```
df_list = [d for _, d in test_ad.groupby(['patient'])]
# update indexes
for d in df_list:
    d.index = pd.RangeIndex(len(d.index))
```

- Fare una copia dei dataset del test set

```
# copy the datasets
df_copy = df_list
```

- Rimuovere **patient**, **timestamp**, **tsDate** e normalizzare

```
test_ad_norm_list = []
for d in df_list:
  test_ad_norm = d.drop(['patient', 'timestamp', 'tsDate'], 1)
  test_ad_norm = (test_ad_norm - train_ad_sampled_mean) / train_ad_sampled_std
  test_ad_norm_list.append(test_ad_norm)
```

A differenza del task 1, in cui l'input è costituito da sequenze contenenti solo i valori di un asse, in questo caso abbiamo creato una funzione che ricevuto come parametri: un dataset, la window size e la window shift estrae le sequenze. Le sequenze estratte hanno 2 dimensioni: la prima è la lunghezza delle sotto-sequenze (30) e la seconda è il numero di attributi che vogliamo aggiungere (4). Abbiamo utilizzato questa funzione per estrarre le sequenze dal training e dai test sets di ogni paziente.

```
def extract_seq_ad(data_set, maxlen, step):
  print("extracting sequences")
  sequences = []
  #for i in range(0, 1000, step):
  for i in range(0, len(data_set) - maxlen, step):
    print("\r" + str(int((i+1) / len(data_set) * 100)) + " %", end="")
    sequence = []
    for j in range(0, maxlen):
      entry = []
      entry.append(data_set['x'][i+j])
      entry.append(data_set['y'][i+j])
      entry.append(data_set['z'][i+j])
      entry.append(data_set['heartRate'][i+j])
      sequence.append(entry)

    sequences.append(sequence)

  sequences = np.array(sequences)
  print("\r100 %")
  return sequences
```

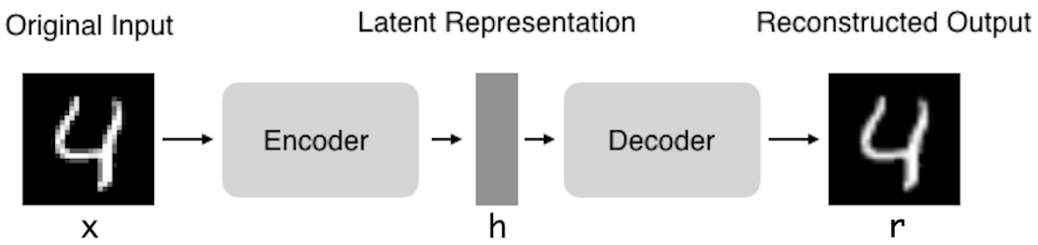
```
sequences_ad = extract_seq_ad(train_ad_sampled, 30, 1)

sequences_ad_test = []
for d in test_ad_norm_list:
    sequences_ad_test.append(extract_seq_ad(d, 30, 1))
```

3. Modeling

In questa fase abbiamo scelto la rete neurale più appropriata per il task. Nel nostro caso è stato scelto un Autoencoder, un tipo speciale di rete neurale che viene addestrata a copiare il suo input in output. Ad esempio, data una sequenza in input, un codificatore automatico prima la codifica in una rappresentazione latente di dimensioni inferiori, quindi, decodifica la rappresentazione latente in una nuova sequenza ricostruita a partire dall'input. Un autoencoder impara a comprimere i dati riducendo al minimo l'errore di ricostruzione.

Questa tipologia di rete neurale è composta da due parti:

- Encoder: la parte della rete che comprime l'input in uno spazio di variabili latenti e che può essere rappresentato dalla funzione di codifica h=f(x).
- Decoder: la parte che si occupa di ricostruire l'input sulla base delle informazioni precedentemente raccolte. È rappresentato dalla funzione di decodifica r=g(h).

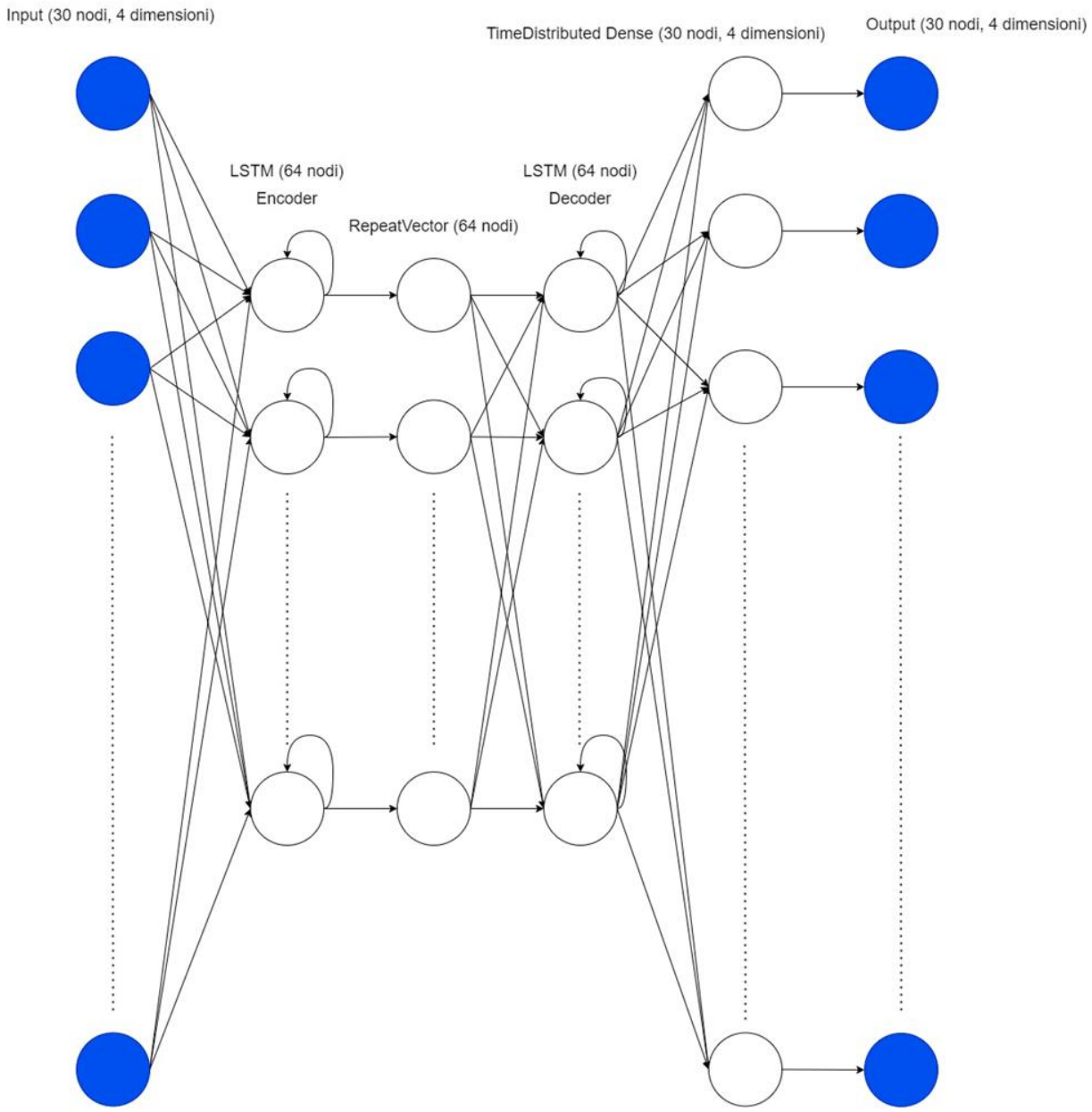


L'autoencoder nel suo complesso può quindi essere descritto dalla funzione d(f(x)) = r dove r è quanto più simile all'input originale x.

Il nostro modello è un autoencoder composto da:

- Un layer LSTM con la stessa forma di input delle sequenze;
- Un layer RepeatVector che replica l'output del layer precedente, per passare correttamente l'output al layer successivo;
- Un altro layer LSTM;
- Un layer TimeDistributed Dense senza funzione di attivazione per fare predizioni dell'input originale.

```
model_ad = Sequential()
model_ad.add(layers.LSTM(64, input_shape=(sequences_ad.shape[1], sequences_ad.shape[2])))
model_ad.add(layers.RepeatVector(sequences_ad.shape[1]))
model_ad.add(layers.LSTM(64, return_sequences=True))
model_ad.add(layers.TimeDistributed(layers.Dense(sequences_ad.shape[2])))
model_ad.compile(loss='mean_absolute_error', optimizer = 'adam')

history = model_ad.fit(sequences_ad, sequences_ad, epochs=10)
```


Input (30 nodi, 4 dimensioni)
LSTM (64 nodi)
Encoder
RepeatVector (64 nodi)
LSTM (64 nodi)
Decoder
TimeDistributed Dense (30 nodi, 4 dimensioni)
Output (30 nodi, 4 dimensioni)

4. Evaluation

In questa fase abbiamo calcolato l'errore di ricostruzione di ogni sequenza. L'errore di una sequenza è stato calcolato come la somma dei valori assoluti delle differenze tra i valori reali e i valori predetti di ogni osservazione nella sequenza. La funzione get_errors() riceve in input le sequenze e ne calcola gli errori. Per ogni paziente è stata individuata una soglia di errore in funzione della lista di errori di ogni osservazione, e abbiamo considerato anomala una osservazione che ha un volare di errore più alto rispetto alla relativa soglia.

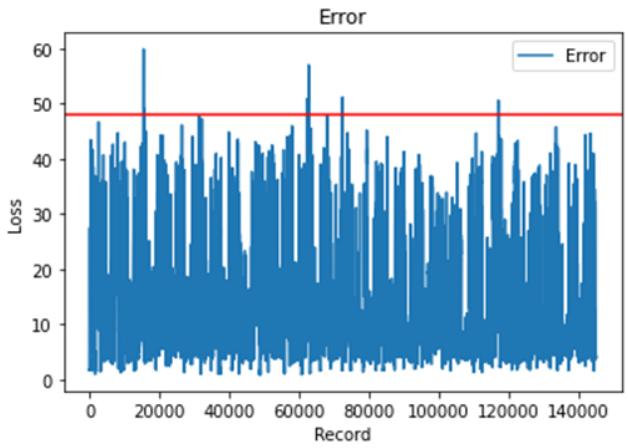
```
def get_errors(sequences):
    print("getting errors")
    errors = []
    for i in range(sequences.shape[0]):
    #for i in range(100):
      print("\r" + str(int((i+1) / sequences.shape[0] * 100)) + " %", end="")
      error = 0
      prediction = model_ad.predict(sequences[[i]])
      for j in range(sequences.shape[1]):
        for k in range(sequences.shape[2]):
          error += abs(sequences[[i]][0][j][k] - prediction[0][j][k])
      errors.append(error)
    print("\r100 %")
    return errors
```

```
errors = []
for seq in sequences_ad_test:
  errors.append(get_errors(seq))
```

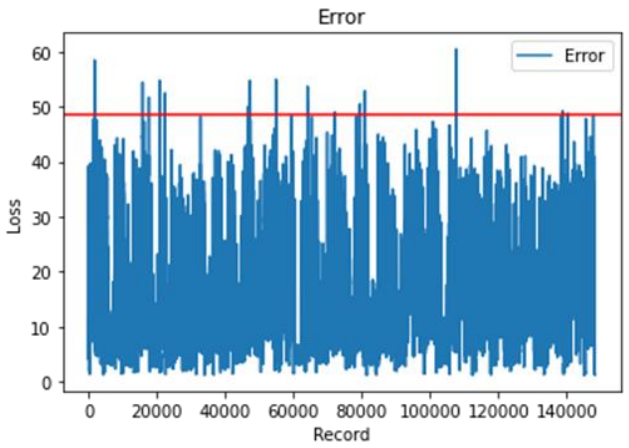
La soglia di ogni paziente è stata calcolata all'80% dell'intervallo tra il valore di errore minimo e massimo nel seguente modo:

```
tr = []
for error in errors:
  _min = error[0]
  _max = error[0]
  for i in range(len(error)):
    if error[i] < _min:
      _min = error[i]
    if error[i] > _max:
      _max = error[i]
  t = ((_max - _min) / 10 * 8) + _min
  tr.append(t)
  plt.plot(range(len(error)), error,  label='Error')
  plt.axhline(t, color='r')
  plt.title('Error')
  plt.xlabel('Record')
  plt.ylabel('Loss')
  plt.legend()
  plt.show()
```

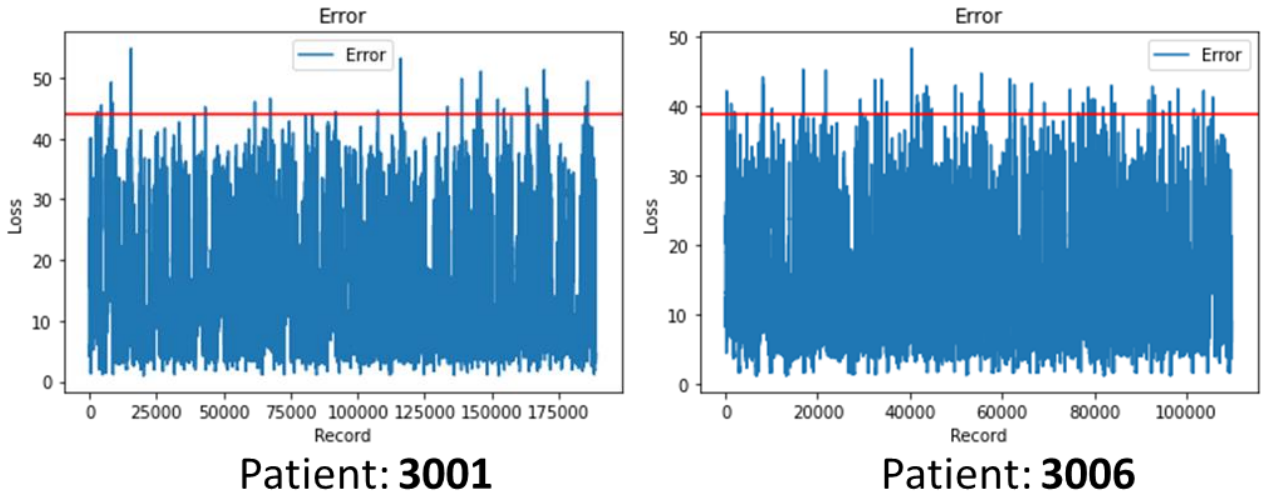
Successivamente abbiamo creato dei grafici in cui per ciascun paziente analizziamo l'andamento degli errori di ricostruzione delle osservazioni e la relativa soglia con i seguenti risultati:

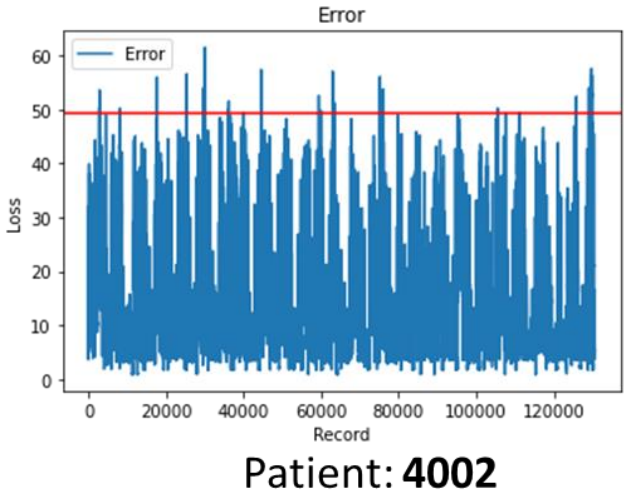


Patient: **1004**



Patient: **1006**





Le osservazioni con un valore di errore di ricostruzione superiore alla relativa soglia (linea rossa) rappresentano le anomalie individuate dall'autoencoder e quindi i periodi di off per ogni paziente. Infine, abbiamo creato un file csv in cui sono riportati i periodi di off per ciascun paziente.

```
off_period = open("off_period.csv", "w")
off_period.write("patient, tsDate, off\n")
for pat in range(len(errors)):
    for i in range(len(errors[pat])):
      if errors[pat][i] > tr[pat]:
        off_period.write(str(df_copy[pat].iloc[i]['patient']) + "," + str(df_copy[pat].iloc[i]['tsDate']) + ", True\n")
off_period.close()
```

5. Conclusioni

Anche per il task 2, i risultati ottenuti sono buoni in quanto l'autoencoder è stato in grado di riprodurre in output le sequenze fornite in input così da calcolare l'errore di ricostruzione. Strumento base per l'identificazione delle anomalie presenti in input e quindi i periodi di off di ciascun paziente.