
UNIVERSITÀ DI PISA

*DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE*

*RELAZIONE PER IL CONSEGUIMENTO DELLA*

*LAUREA IN INGEGNERIA INFORMATICA*

# Sviluppo di un modello LSTM per prevedere il numero di posti letto occupati in terapia intensiva da pazienti COVID-19

**Relatori:**

Prof. Beatrice Lazzerini

Dr. Francesco Pistolesi

**Candidato:**

Gabriele Benanti

Sessione di Laurea del 24/09/2021

# Abstract

Questa tesi tratta di come andare ad utilizzare algoritmi di intelligenza artificiale per andare a fare previsioni sulla pandemia COVID-19. In particolare, andremo a studiare come sono correlati il numero di vaccinazioni e il rapporto test-positivi/test-eseguiti al numero di posti occupati in terapia intensiva da pazienti COVID-19, con l'obiettivo di andare a compiere delle previsioni sul numero di posti letto occupati in terapia intensiva da pazienti COVID-19. Andremo ad usare l'algoritmo LSTM (Long Short-Term Memory), una rete neurale di tipo ricorrente (RNN). Le reti neurali rientrano nell'ambito del deep learning. Vedremo come costruire un modello sempre più preciso andando a scegliere come dare in ingresso alla rete neurale un determinato set di variabili di ingresso, la quale andrà a trovare un pattern che lega i vari dati in ingresso, producendo in uscita delle previsioni il più possibile accurate. Inoltre, vedremo come scegliere i parametri ottimi interni alla rete neurale. In conclusione, osserveremo come sia la struttura interna della rete neurale sia la dimensione del dataset dato in ingresso alla rete neurale siano due fattori significativi per ottenere un modello il più possibile accurato nel prevedere il numero di posti letto occupati in terapia intensiva.

# Sommario

# 1. Introduzione

La pandemia di COVID-19, iniziata nel dicembre del 2019 in Cina, ha avuto ripercussioni significative nelle nostre vite: gli Stati globali hanno dovuto affrontare una crisi economico-sanitaria inimmaginabile, dovendo nel minor tempo possibile riorganizzare i propri assetti di assistenza sanitaria. Come sappiamo la maggior parte degli Stati non è stata in grado di reggere l'onda d'urto generata dalla pandemia, con conseguenze disastrose sia dal punto di vista sanitario che economico.

Soffermandoci sul piano sanitario abbiamo potuto constatare come il numero di posti letto in terapia intensiva sia stato un fattore significativo per avere un numero minore di decessi causati da COVID-19. Infatti, la Germania a fronte di un valore di 33,9 posti letto di terapia intensiva ogni 100.000 abitanti (dati aggiornati al 2017) ha avuto 92.146 decessi per COVID-19 (al 30/08/2021), mentre l'Italia con un valore di 8,6 posti letto di terapia intensiva ogni 100.000 abitanti (dati aggiornati al 2020) ha avuto 129.000 decessi per COVID-19(al 30/08/2021).

Soffermandoci su questi dati si può capire quanto sia importante avere un adeguato numero di posti letto in terapia intensiva, considerando anche l'elevato costo di ogni singolo posto letto. Infatti, il costo medio giornaliero di degenza di un malato COVID-19 in un reparto di terapia intensiva è di 1.425 euro (al 22 febbraio 2021, dati Ministero della Salute).

L'obiettivo che si prefigge questo lavoro di tesi è di andare a prevedere il numero di posti letto occupati in terapia da pazienti affetti da COVID-19 grazie all'uso di algoritmi di intelligenza artificiale. In particolare, andremo ad utilizzare un modello a rete neurale, nella fattispecie una rete neurale ricorrente (RNN). Tra le RNN abbiamo l'algoritmo LSTM (Long Short-Term Memory), uno degli algoritmi più adatti a fare previsioni di serie temporali. L'uso di questo algoritmo ci permetterà di creare un modello che lega il numero di posti letto occupati in terapia intensiva da pazienti COVID-19 con il numero totale di vaccini somministrati, suddivisi per tipologia di vaccino (Pfizer/BioNTech, Moderna, Oxford/AstraZeneca, Johnson&Johnson), per andare a predire il numero di posti letto occupati in terapia intensiva da pazienti COVID-19. Inoltre, osserveremo come l'aggiunta del rapporto tra test-positivi/test-effettuati come terzo parametro in ingresso alla rete influenzerà le previsioni del nostro modello predittivo.

Il nostro scopo finale non sarà tanto andare a vedere se un modello LSTM sia affidabile o meno nel fare previsioni su serie temporali ma bensì andare a studiare quanto sia accurato al variare dei dati di ingresso e della sua configurazione interna.

# 2. Background

Nella relazione di tesi andremo ad affrontare argomenti di carattere scientifico che spaziano dalla matematica all'informatica. Nel caso dello studio delle reti neurali osserveremo come i due campi si vadano ad intrecciare molto spesso.

## 2.1 Intelligenza Artificiale

L'intelligenza artificiale (AI) è un termine che spesso ritroviamo in vari ambiti, dalle scienze pure all'economia. Lo scopo dell'intelligenza artificiale è quello di andare a creare strumenti che ci permettono di andare a studiare fenomeni molto complessi che richiedono di analizzare molti dati, in modo da riuscire a capire anche quelli più inafferrabili. Attraverso algoritmi di intelligenza artificiale possiamo andare a creare metodi di analisi predittiva.

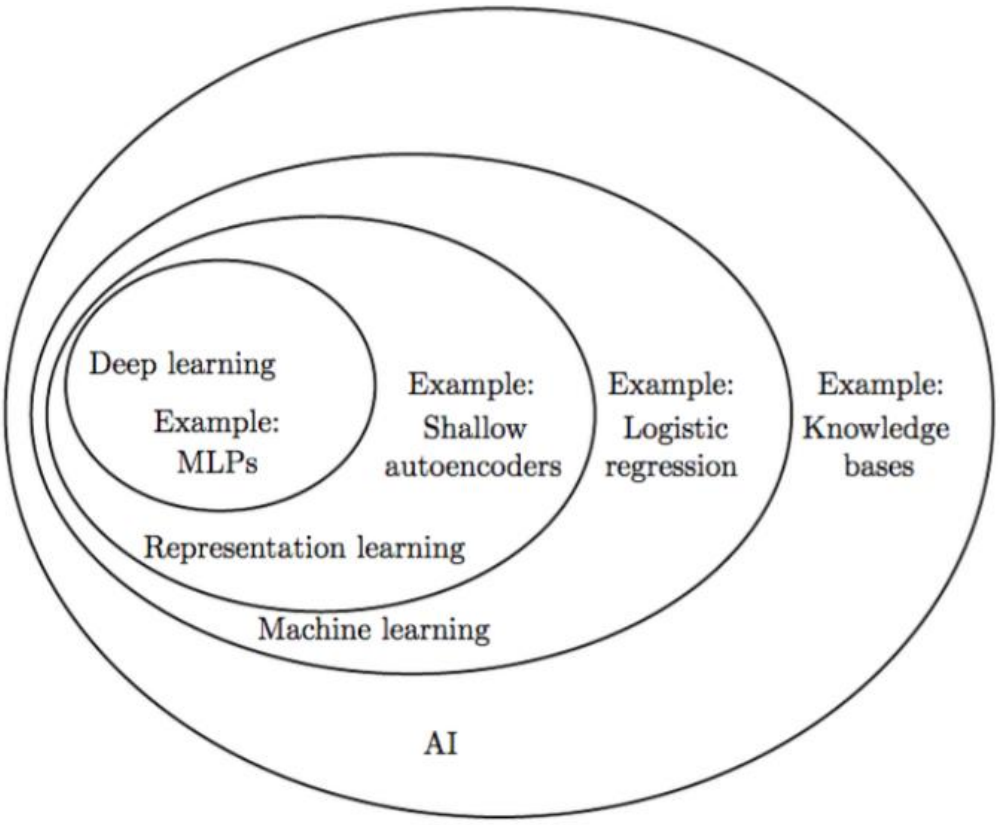



Figura 1: diagramma di Venn che mostra la correlazione tre AI, Deep Learning et cetera.

## 2.2 Machine Learning

Il Machine Learning (ML) è uno strumento che ci permette di trovare dei pattern nei dati. Gli algoritmi di Machine Learning si concentrano nell'ottimizzazione e nel trovare delle regolarità nei dati. Tutto ciò si ottiene grazie all'uso di elementi matematici e statistici. Uno dei grandi difetti nell'uso di algoritmi di Machine Learning è l'overfitting dei dati, ovvero quel fenomeno che si manifesta quando l'algoritmo basa in gran parte le proprie elaborazioni su dati di esempio o dati storici. Ciò si traduce nell'avere previsioni su dati nuovi con un errore molto elevato. Per minimizzare questo problema molti scienziati cercano di utilizzare metodi ibridi di Machine Learning. Inoltre, è fondamentale il modo in cui i dati da dare in ingresso a questi algoritmi vengono rappresentati. Ciò influisce significamenti su come l'algoritmo elaborerà i dati.

## 2.3 Reti Neurali

La rete neurale è un modello computazionale che si basa sul machine learning e prende ispirazione dalle reti neurali studiate in biologia. Ogni rete neurale è formata da dei nodi (paragonabili ai neuroni del cervello umano) suddivisi su più livelli. Ogni nodo della rete è collegato con altri nodi, i quali a loro volta sono collegati ad altri nodi su livelli più interni o esterni alla rete. Ogni nodo riceve informazioni da altri nodi attraverso dei collegamenti (paragonabili alle sinapsi). L'informazione ricevuta dal nodo viene moltiplicata per il peso definito dal costruttore della rete. Il peso viene usato per andare a regolare l'importanza del calcolo eseguito da un determinato nodo. La funzione non lineare (chiamata anche funzione di attivazione) viene aggiunta al risultato dell'elaborazione. Tipicamente la funzione di attivazione è una funzione *tan(h),* il cui scopo è quella di andare a regolare in che misura il risultato del nodo debba essere posto in uscita dal nodo stesso, uscita che può essere quella finale della rete o passata ad un altro nodo della rete.

## 2.4 Reti Neurali Ricorrenti

Le reti neurali ricorrenti (RNN) sono una tipologia di rete neurale che presenta la caratteristica di avere memoria, il che le rende molto simili a come funziona il cervello umano, e per questo sono

utilizzate in gran misura. Nelle reti neurali tradizionali i dati sono processati indipendentemente, mentre nel caso delle RNN i dati presentati in ingresso in forma di sequenza temporale vengono collegati tra di loro in iterazioni diverse dell'algoritmo grazie alla presenza di uno strato della rete neurale che funge da memoria interna.

## 2.5 Long-term dependencies

Con il termine Long-term dependencies si indica il problema che si manifesta nelle RNN quando la rete deve fare delle previsioni su dati che richiedono di essere collegati ad un contesto temporale (come, ad esempio, nel nostro caso i dati delle vaccinazioni e dei ricoveri devono essere collegati all'andamento della pandemia). In una rete RNN classica è possibile gestire la necessità di contestualizzare e legare i dati nel tempo, ma dipende da quanti dati devono essere salvati in memoria. Il vantaggio di usare LSTM al posto di RNN sta nel modo in cui la rete memorizza i dati: LSTM ci permette di memorizzare informazioni per un periodo più lungo rispetto a RNN.

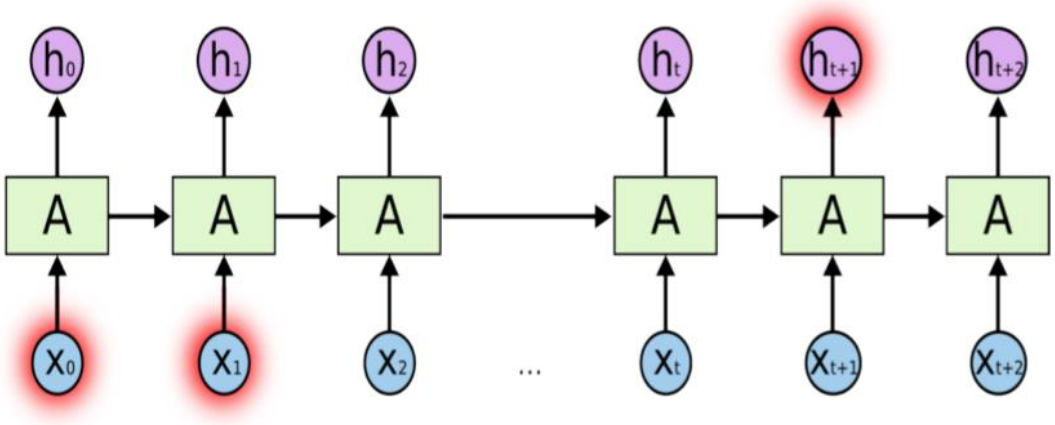
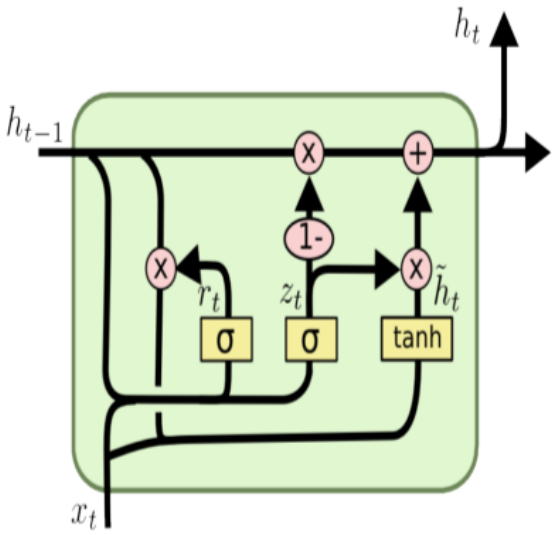



Figura 2: Long-term dependencies

## 2.6 Long short-term memory (LSTM)

LSTM è un tipo di RNN che presenta l'abilità di considerare le long-term dependencies. Il modello LSTM è stato sviluppato da due scienziati, Schmidhuber e Hochreiter nel 1997. Il grande pregio di LSTM è di avere una memoria che ci permette di andare a memorizzare informazioni per un periodo di tempo più lungo, e che quindi ci permette di risolvere il problema delle long-term dependencies. LSTM ha una struttura interna a catena e opera internamente usando gates e layer come le altre reti RNN. La struttura del LSTM è costruita in modo tale che una cella di stato operi

attraverso tutto LSTM, il cui valore viene modificato dall'azione dei gate che decidono se un aggiungere o meno i valori alla cella di stato. Esistono altri elementi all'interno della rete che prendono il nome di celle gate che hanno la funzione di andare a prendere i dati dall'output di precedenti LSTM o da altri layer interni alla rete e salvarli al loro interno. Tutto ciò rende LSTM reti con memoria.




$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$

$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$

$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

Figura 3: algoritmo long short-term memory

## 2.7 Data Mining

Con Data Mining si intende il processo di estrazioni di dati significativi e nascosti da un largo set di dati. Lo scopo di tutto è di andare a trovare tendenze nascoste e valori significativi non osservabili direttamente.

## 2.8 Regression Analysis

In matematica con il termine regression analysis si intende l'insieme degli strumenti che ci permette di andare a trovare correlazioni tra variabili dipendenti o indipendenti. Abbiamo diversi tipi di regressioni, la forma più semplice è la regressione lineare. Nella regressione lineare abbiamo une relazione lineare tra le variabili, corrispondente a una forma del tipo $y = \alpha + \beta x$ . Per rappresentare relazioni più complesse, che i modelli lineari non coprono, usiamo un modello regressivo del tipo:

$$y_i = \alpha + \beta x_i + \epsilon_i$$

Dove α e β rappresentano i parametri, $x_i$ la variabile indipendente e $\varepsilon i$ l'errore.

Nel caso in cui i modelli regressivi lineari non siano adatti ai nostri scopi, possiamo andare ad usare modelli regressivi del tipo:

$$y_i = \beta_0 + \beta_1 x_i + \beta_2 x_i^2 + \epsilon_i$$

Dove$\beta_0, \beta_1, \beta_2$ rappresentano i parametri. L'errore viene calcolato come la differenza tra il valore reale e il valore previsto.

## 2.9 Errore quadratico medio

L'errore quadratico medio (MSE) è un metodo che ci permette di valutare l'errore nei modelli predittivi.

$$MSE = \frac{1}{n}\sum_{n=1}^{n}\left(Y_i - \hat{Y}_i\right)^2$$

$\hat{Y}_i$ è i-esimo valore predetto, $\hat{Y}$ è i-esimo valore reale.

Andremo ad utilizzare MSE per scegliere i parametri ottimi per il nostro modello.

## 2.10 Radice dell'errore quadratico medio

La radice dell'errore quadratico medio (RMSE) è un metodo con cui noi andiamo a valutare di quanto i valori previsti si discostano dai valori reali. In particolare, RMSE indica quanto i residui (errori di predizione) si discostano dai valori reali.

$$RMSE = \sqrt{\frac{\sum_{n=1}^{n}\left(Y_i - \hat{Y}_i\right)^2}{n}}$$

Useremo RMSE per andare ad analizzare l'accuratezza delle nostre previsioni.

# 3. Sviluppo del modello

Il modello LSTM che andremo a sviluppare è implementato con Keras, una API (**application programming interface)** scritta in Python, che a sua volta si basa su librerie di più basso livello, come TensorFlow e Theano. TensorFlow è un framework open source che usa approcci di machine learning per il calcolo numerico. In ingresso alla rete verrà dato un dataset che contiene dati legati alla pandemia COVID-19.

## 3.1 Dataset

I dati sono stati raccolti a metà luglio 2021 dal sito https://ourworldindata.org/. Il dataset contiene dati relativi all'intervallo che va dal 27/12/2020 al 04/07/2021. Avendo scelto di andare a verificare se esiste una correlazione tra i dati delle vaccinazioni e i dati dei ricoveri in terapia intensiva e, essendo iniziate le vaccinazioni sul territorio italiano il 27/12/2020, sono stati esclusi i dati dei ricoveri antecedenti a quella data. I dati sono stati scaricati nel formato .csv. I dati sono relativi solo all'Italia.

| Entity | Day | Pfizer/BioNTech | Moderna | Oxford/AstraZeneca | Johnson&Johnson | Daily ICU occupancy |
|---|---|---|---|---|---|---|
| Italy | 2020-12-27 | 7221 | 0 | 0 | 0 | 2580 |
| Italy | 2020-12-28 | 8654 | 0 | 0 | 0 | 2565 |
| Italy | 2020-12-29 | 9671 | 0 | 0 | 0 | 2549 |
| Italy | 2020-12-30 | 14406 | 0 | 0 | 0 | 2528 |
| Italy | 2020-12-31 | 39875 | 0 | 0 | 0 | 2555 |
| Italy | 2021-01-01 | 50945 | 0 | 0 | 0 | 2553 |
| Italy | 2021-01-02 | 89476 | 0 | 0 | 0 | 2569 |
| Italy | 2021-01-03 | 124751 | 0 | 0 | 0 | 2583 |
| Italy | 2021-01-04 | 193692 | 0 | 0 | 1 | 2579 |
| Italy | 2021-01-05 | 273451 | 0 | 0 | 1 | 2569 |
| Italy | 2021-01-06 | 338472 | 1 | 0 | 1 | 2571 |

Tabella 1: esempio del contenuto del dataset.

Successivamente andremo ad aggiungere un ulteriore parametro al dataset: il rapporto test positivi/test eseguiti. Andremo a studiare l'effetto dell'aggiunta di questo parametro al nostro modello.

| Entity | Day | Pfizer/BioNTech | Moderna | Oxford/AstraZeneca | Johnson&Johnson | Daily ICU occupancy | short_term_positivity_ratio |
|---|---|---|---|---|---|---|---|
| Italy | 2020-12-27 | 7221 | 0 | 0 | 0 | 2580 | 0.103 |
| Italy | 2020-12-28 | 8654 | 0 | 0 | 0 | 2565 | 0.103 |
| Italy | 2020-12-29 | 9671 | 0 | 0 | 0 | 2549 | 0.10400000000000001 |
| Italy | 2020-12-30 | 14406 | 0 | 0 | 0 | 2528 | 0.10800000000000001 |
| Italy | 2020-12-31 | 39875 | 0 | 0 | 0 | 2555 | 0.11600000000000002 |
| Italy | 2021-01-01 | 50945 | 0 | 0 | 0 | 2553 | 0.11899999999999998 |
| Italy | 2021-01-02 | 89476 | 0 | 0 | 0 | 2569 | 0.122 |
| Italy | 2021-01-03 | 124751 | 0 | 0 | 0 | 2583 | 0.122 |
| Italy | 2021-01-04 | 193692 | 0 | 0 | 1 | 2579 | 0.124 |
| Italy | 2021-01-05 | 273451 | 0 | 0 | 1 | 2569 | 0.127 |
| Italy | 2021-01-06 | 338472 | 1 | 0 | 1 | 2571 | 0.131 |

Tabella 2: esempio del contenuto del dataset finale.

Il dataset è stato diviso in due parti: dataset di training e dataset di test. Il primo verrà utilizzato per andare ad allenare la rete, mentre il secondo verrà utilizzato per andare a fare le previsioni e a studiare l'accuratezza raggiunta dal modello. Il dataset originale è stato suddiviso seguendo questa modalità:

```
data_test=dataset.iloc[lambda x: x.index % 4 == 0,[3,4,5,6,11,12]].values
training_set=dataset.iloc[lambda x: x.index % 4 != 0,[3,4,5,6,11,12]].values
```

Viene aggiunto un record nel dataset di test ogni tre record contigui aggiungi al dataset di train. I due dataset sono quindi complementari.

### 3.1.1 Descrizione dei parametri del dataset

- **Pfizer/BioNtech**: numero totale di dosi iniettate. La vaccinazione si ritiene completa se un individuo ha ricevuto due dosi; perciò, il numero di dosi iniettate può risultare maggiore del numero delle persone nella popolazione.
- **Moderna**: numero totale di dosi iniettate. La vaccinazione si ritiene completa se un individuo ha ricevuto due dosi; perciò, il numero di dosi iniettate può risultare maggiore del numero delle persone nella popolazione.
- **Oxford/AstraZeneca**: numero totale di dosi iniettate. La vaccinazione si ritiene completa se un individuo ha ricevuto due dosi; perciò, il numero di dosi iniettate può risultare maggiore del numero delle persone nella popolazione.
- **Johnson&Johnson**: numero totale di dosi iniettate. La vaccinazione è completa dopo una dose.
- **Daily ICU occupancy:** numero di posti letto occupati in terapia intensiva giornalmente da pazienti COVID-19.
- **Short_term_positivity_ratio**: rapporto tra test positivi e test eseguiti, su base giornaliera.

## 3.2 Normalizzazione dei dati

Le reti LSTM migliorano la loro accuratezza se i dati presentati in ingresso vengono normalizzati. I dati del dataset sono stati normalizzati in un intervallo tra 0 e 1 utilizzando la funzione *MinMaxScaler* inclusa nella libreria **sklearn.preprocessing.**

## 3.3 Struttura LSTM

Le performance di un modello LSTM dipendono esclusivamente da come sono impostati i parametri interni alla rete (iperparametri) e da quanto viene allenato.

### 3.3.1 Input

Ogni record del dataset di training viene visto come un vettore 1x6, dove 6 indica il numero di parametri (features) dati in ingresso alla rete. L'insieme dei record del training dataset, ognuno con i relativi features, forma una matrice bidimensionale. Successivamente la matrice bidimensionale andrà trasformata in un vettore tridimensionale, poiché LSTM prende in ingresso solo vettori 3D. Useremo la libreria **numpy** per fare queste operazioni. I vettori 3D avranno questa forma: ( S x T x F ), dove S = numero di samples , T = dimensione del TimeSteps, F = numero di features.

- **Samples:** singolo campione di dati dato in ingresso alla rete neurale.
- **TimeStemps:** indica da quanti record del dataset è formato un sample.
- **Features:** singolo dato in un record (i.e. una colonna del dataset).

### 3.3.2 Hidden Layer

Quando progettiamo un modello LSTM dobbiamo tenere in considerazione quanti hidden layers deve contenere la nostra rete. L'hidden layers sono i livelli interni della rete , i quali determinano l'evoluzione della rete. Non esiste una tecnica rigorosa per andare a determinare il numero di hidden layer; perciò, andremo a fare delle prove sperimentali. In genere in modelli non troppo complessi il numero di hidden layer varia tra 1 e 5, ma ciò dipende fortemente dal campo in cui viene applicato LSTM.

### 3.3.3 Dense Layer

Il dense layer è un layer NN densamente collegato ad altri layer del livello successivo, in cui ogni cella del dense layer è collegata a quella del layer successivo. Tipicamente viene usato come layer di output della rete.

### 3.3.4 Numero di layer

Andremo a studiare come, al variare del numero dei layer, cambierà l'accuratezza del nostro modello. In particolare, testeremo due configurazioni:

- *hidden layer -> dense layer.*
- *hidden layer -> hidden layer -> dense layer.*

## 3.4 Esperimenti

In questa sezione andremo a studiare come l'aggiunta del rapporto tra test positivi/test effettuati come feature influisca sull'accuratezza delle previsioni della rete LSTM. Inoltre, sceglieremo i valori ottimi degli iperparametri della nostra rete LSTM.

### 3.4.1 Aggiunta del rapporto test positivi/test eseguiti.

Quando andiamo a scegliere i parametri da dare in ingresso alla nostra rete neurale dobbiamo preoccuparci di andare a scegliere dei dati che ci permettano di andare ad evidenziare maggiormente i legami e i meccanismi che esistono all'interno dei fenomeni studiati. Nel nostro caso, l'aggiunta del rapporto tra test positivi/test eseguiti ci permette di andare a legare il numero delle vaccinazioni (divise per tipologia di produttore) e il numero di posti letto occupati in terapia intensiva da pazienti COVID-19 con l'andamento giornaliero della pandemia. Per eseguire i test la rete è stata configurata nella seguente maniera:

- 1 hidden layer, units per layer = 32, dropout = 0.1, batch = 4, epoch = 100.

Test senza rapporto test positivi/test eseguiti come feature nel dataset.

- `MSE train: 0.0003 MSE test: 0.0044  RMSE test: 234.22`

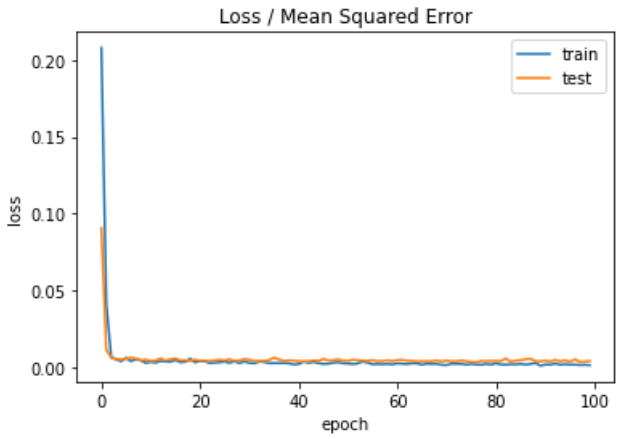



Figura 4.1: MSE del dataset di training e del dataset di test

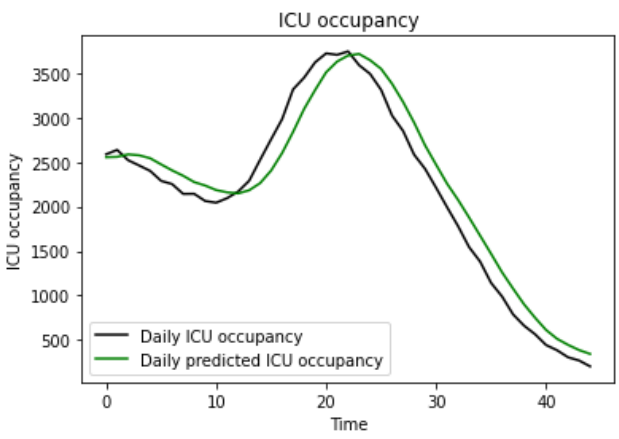



Figura 4.2: Previsioni dal 07/01/2021-03/07/2021. Un valore ogni 4 giorni.

Test con rapporto test positivi/test eseguiti come feature nel dataset.

❖ `MSE train: 0.0003 MSE test: 0.0039  RMSE test: 218.61`

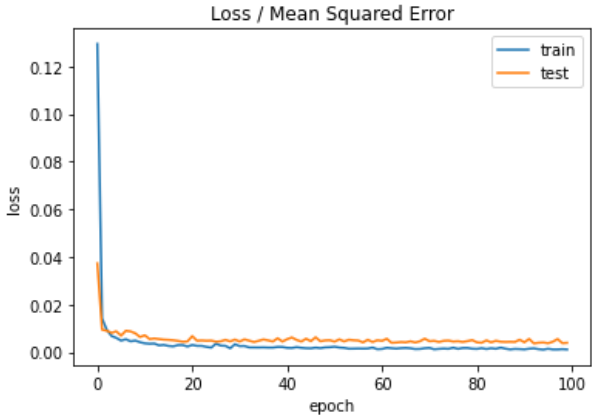



Figura 5.1: MSE del dataset di training e del dataset di test

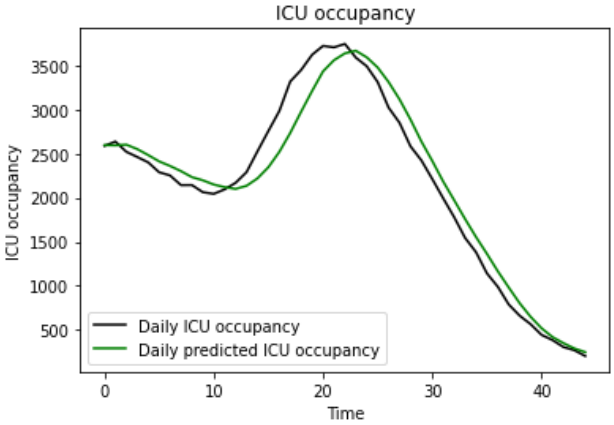



Figura 5.2: Previsioni dal 07/01/2021-03/07/2021. Un valore ogni 4 giorni.

Possiamo concludere che l'accuratezza del modello risulta migliorata nel caso in cui andiamo ad aggiungere il rapporto test positivi/test eseguiti. A parità di MSE calcolato sul dataset di train, MSE calcolato sul dataset di test risulta minore nel secondo caso. Inoltre, RMSE risulta minore nel secondo caso. Ciò si traduce in un modello più preciso nel fare previsioni.

### 3.4.2. Scelta degli iperparametri ottimi

Durante la costruzione del modello LSTM dobbiamo preoccuparci di andare a settare i valori degli iperparametri in modo tale da creare un modello il più accurato possibile nel fare le previsioni. Eseguiremo una serie di test empirici per scegliere i valori a noi più adatti. Andremo ad analizzare singolarmente ogni iperparametro, andando di volta in volta a settare la nostra rete LSTM con dei valori di default per andare poi a variare il valore dell'iperparametro preso in considerazione in quel momento, con l'obiettivo di individuare il valore che ci garantisce delle prestazioni soddisfacenti. Il valore dell'iperparametro che garantisce il più piccolo MSE, calcolato sia sul dataset di test, sia sul dataset di training, sarà quello migliore. Minore è il valore di MSE, più preciso sarà il nostro modello a fare predizioni. MSE viene calcolato tra il valore reale di posti letto occupati in terapia intensiva e il valore previsto dal modello dei posti letto occupati.

### 3.4.2 Dimensione del Batch

L'iperparametro batch indica il numero di samples che la rete deve elaborare per andare ad aggiornare i propri parametri interni. La dimensione del batch potrà essere corrispondente al più alla dimensione del dataset di train. Per fare le prove abbiamo usato la seguente configurazione:

- 1 hidden layer, units per layer = 32, dropout = 0.1, epoch = 100.

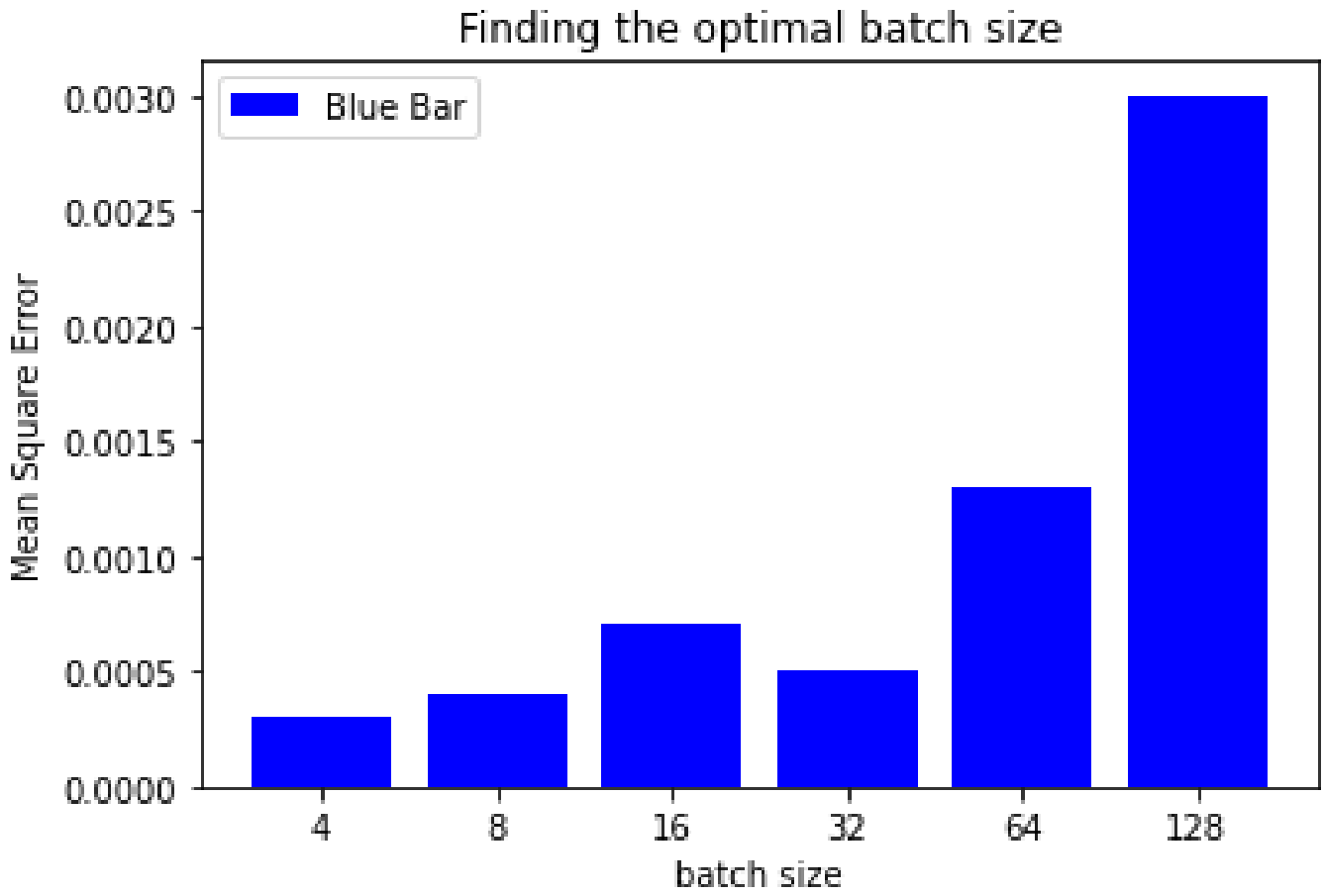



Figura 6: grafico che mostra MSE per ogni valore scelto di batch

Possiamo osservare che il valore MSE più piccolo si ottiene per batch = 4. In questo caso MSE vale 0.0003 .

### 3.4.3 Dimensione del dropout

Il dropout è una tecnica che ci permette di andare a diminuire l'overfitting, andando a scegliere casualmente alcune celle da disattivare in ogni layer. Il numero di celle disattivate dipenderà dal valore del dropout scelto. I test sono stati eseguiti con questa configurazione:

- 1 hidden layer, units per layer = 32, batch = 4, epoch = 100.

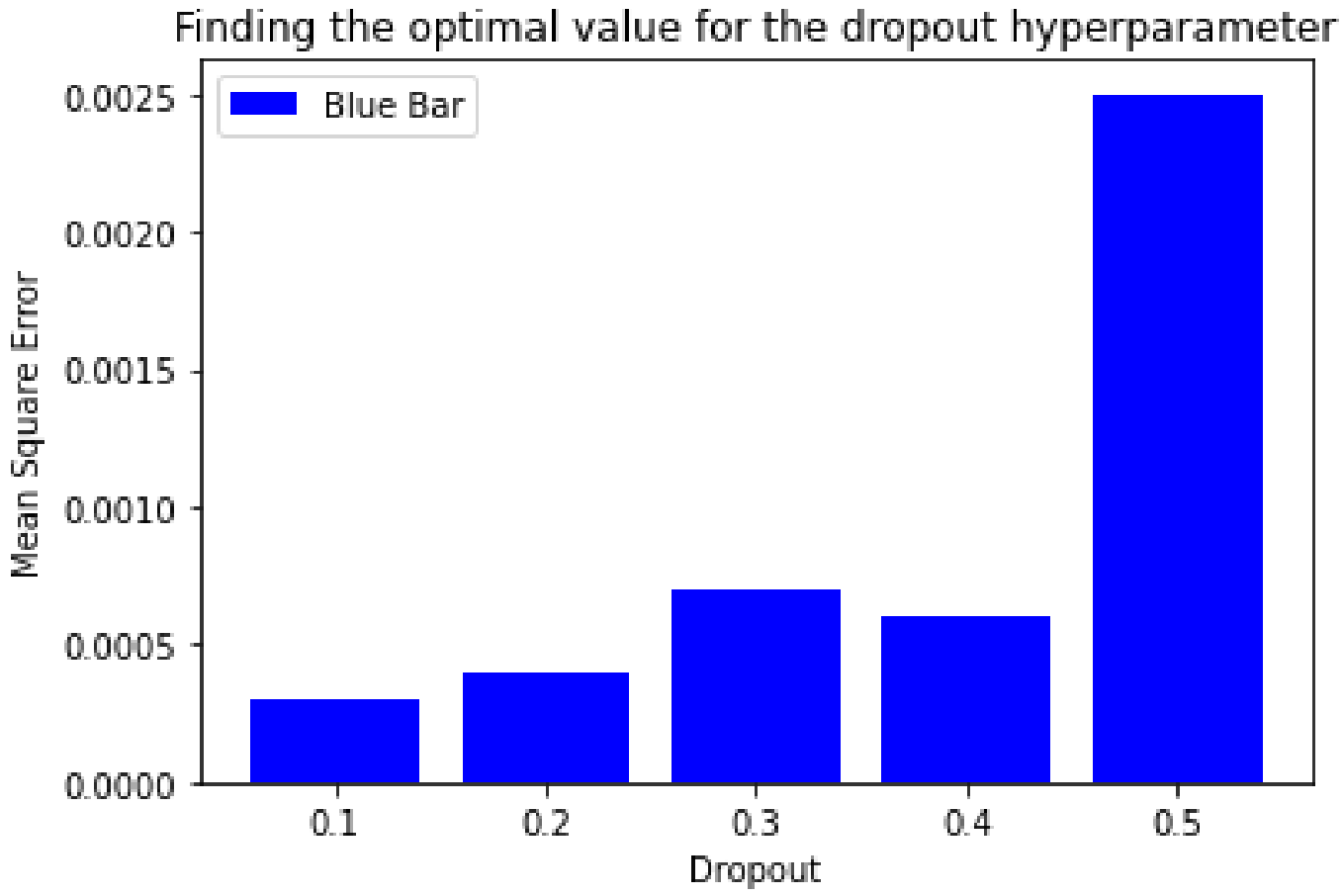



Figura 7: grafico che mostra i valori di MSE per ogni prova empirica.

Possiamo osservare che nel nostro caso dropout = 0.1 garantisce il minor MSE. In questo caso MSE = 0.0003.

### 3.4.4 Dimensione di epoch

Epoch indica quante volte il nostro dataset di training è stato passato in ingresso alla rete ed elaborato da essa; perciò, epoch = 1 corrisponde ad aver elaborato una sola volta l'intero dataset. La configurazione usata per eseguire i test empirici è la seguente:

- 1 hidden layer, units per layer = 32, dropout = 0.1, batch = 4.

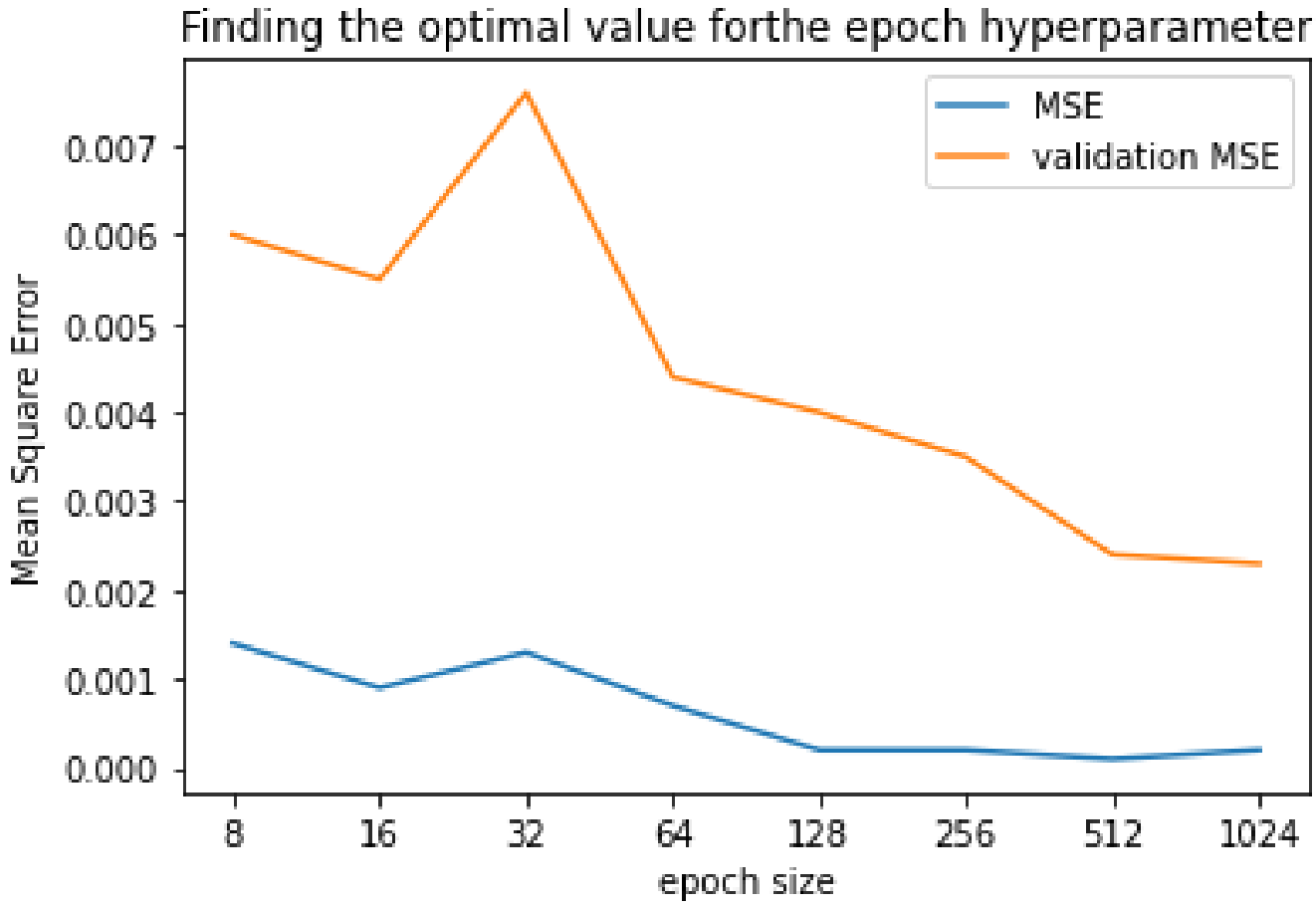



Figura 8: grafico che mostra MSE del training set (blu) e MSE del validation set (arancione) calcolati al variare della dimensione di epoch.

Nel nostro caso MSE del training set risulta minore con epoch = 512. Nonostante si abbiano valori molto simili del MSE del training set a partire da epoch = 128, possiamo notare che per epoch = 512 abbiamo un MSE del set di validazione molto simile al valore ottenuto con epoch = 1024. Con quest'ultimo valore però il tempo di esecuzione dell'algoritmo raddoppia rispetto a 512. Perciò sceglieremo epoch = 512 perché con prestazioni analoghe ci permette un'esecuzione più veloce.

### 3.4.5 Numero di celle LSTM per layer

Ogni layer della rete neurale è formato da un numero di celle. Vogliamo definire il numero ottimale di celle per un singolo hidden layer. Il numero di celle per il dense layer, che funge da layer di output, è impostato a 1. Per eseguire i test è stata usata la seguente configurazione.

- 1 hidden layer, dropout = 0.1, batch = 4, epoch = 512.

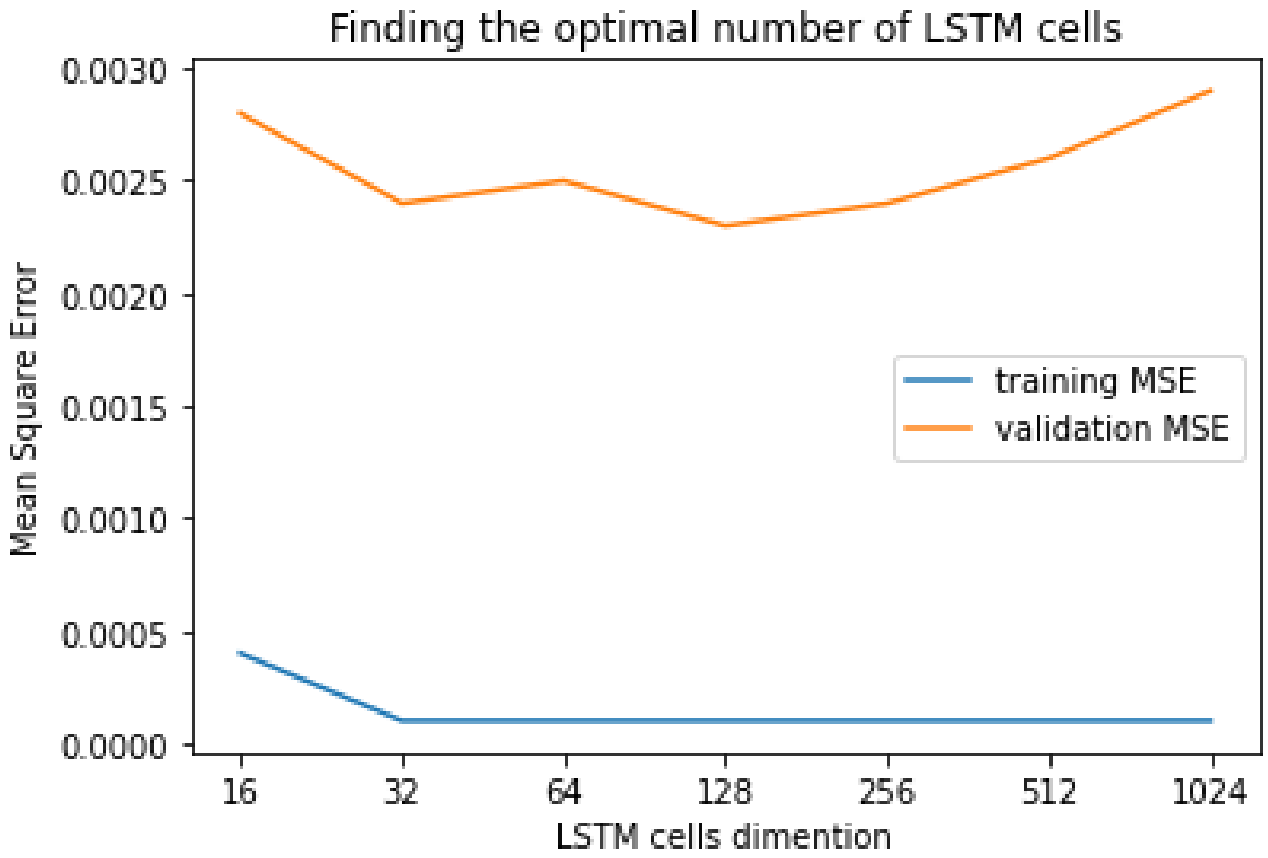



Figura 9: grafico che mostra MSE del training set (blu) e MSE del validation set (arancione) calcolati al variare del numero di celle.

Osserviamo che a partire da un numero maggiore o uguale di celle pari a 32 abbiamo un MSE calcolato sul training set costante. Scegliamo 128 come numero di celle per il singolo hidden layer della rete LSTM poiché presenta MSE più basso sul dataset di validazione. Inoltre, considerando l'esiguo numero di record nel dataset, non è necessario andare a creare una rete troppo grande.

### 3.4.6 Numero di hidden layer

Il nostro obiettivo rimane quello di andare a creare un modello che ci permette di ottenere delle previsioni il più accurate possibili. Ciò si traduce nell'andare ad avvicinare il valore di MSE, calcolato sia sul validation set sia sul training set, a zero. Questo risultato è possibile ottenerlo andando ad aggiungere uno o più hidden layer al modello LSTM. Noi testeremo il nostro modello con due hidden layer, e lo confronteremo con quello a singolo hidden layer. Per eseguire i test è stata usata la seguente configurazione.

- 2 hidden layer, dropout = 0.1, batch = 4, epoch = 512.

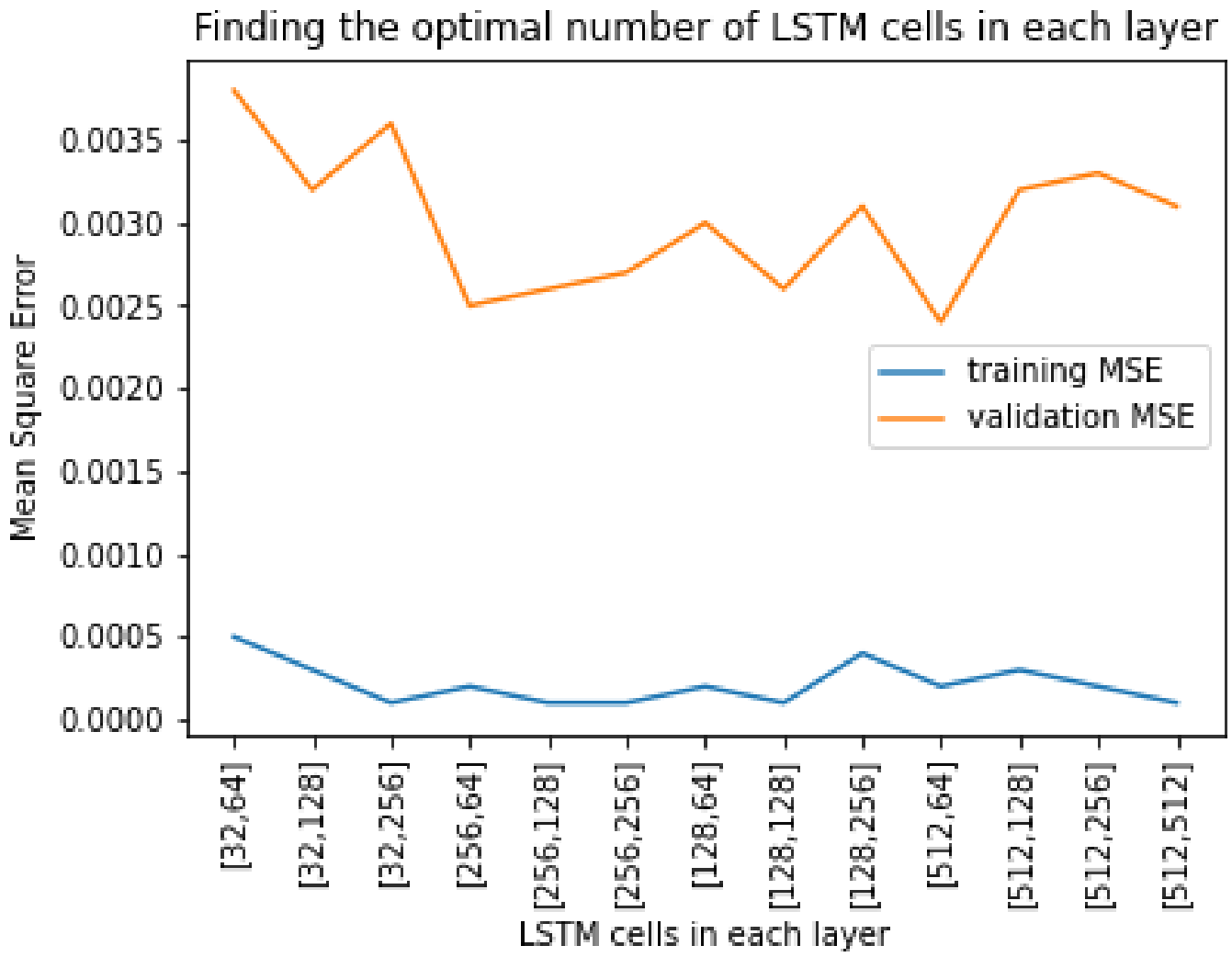



Figura 10: MSE di una rete LSTM con due hidden layer.

Possiamo osservare che le configurazioni [256,64] e [512,64] hanno valori di MSE più piccoli, quindi presenteranno prestazioni molto simili. Confrontando i valori di MSE ottenuti dalle configurazioni con due hidden layer con la configurazione a singolo hidden layer possiamo concludere che la configurazione con un solo hidden layer offre delle prestazioni migliori. Non è quindi necessario aggiungere un ulteriore hidden layer alla nostra rete LSTM.

### 3.4.7. Dimensione della finestra (timesteps)

Abbiamo testato attraverso delle prove la dimensione della finestra (timesteps). Dato il numero ridotto di record nel dataset iniziale, abbiamo testato valori molto piccoli. Avere una finestra piccola presenta il problema di non andare a considerare le relazioni di lungo termine tra i dati. Per eseguire i test è stata usata la seguente configurazione.

- 1 hidden layer, dropout = 0.1, batch = 4, epoch = 512, units = 128.

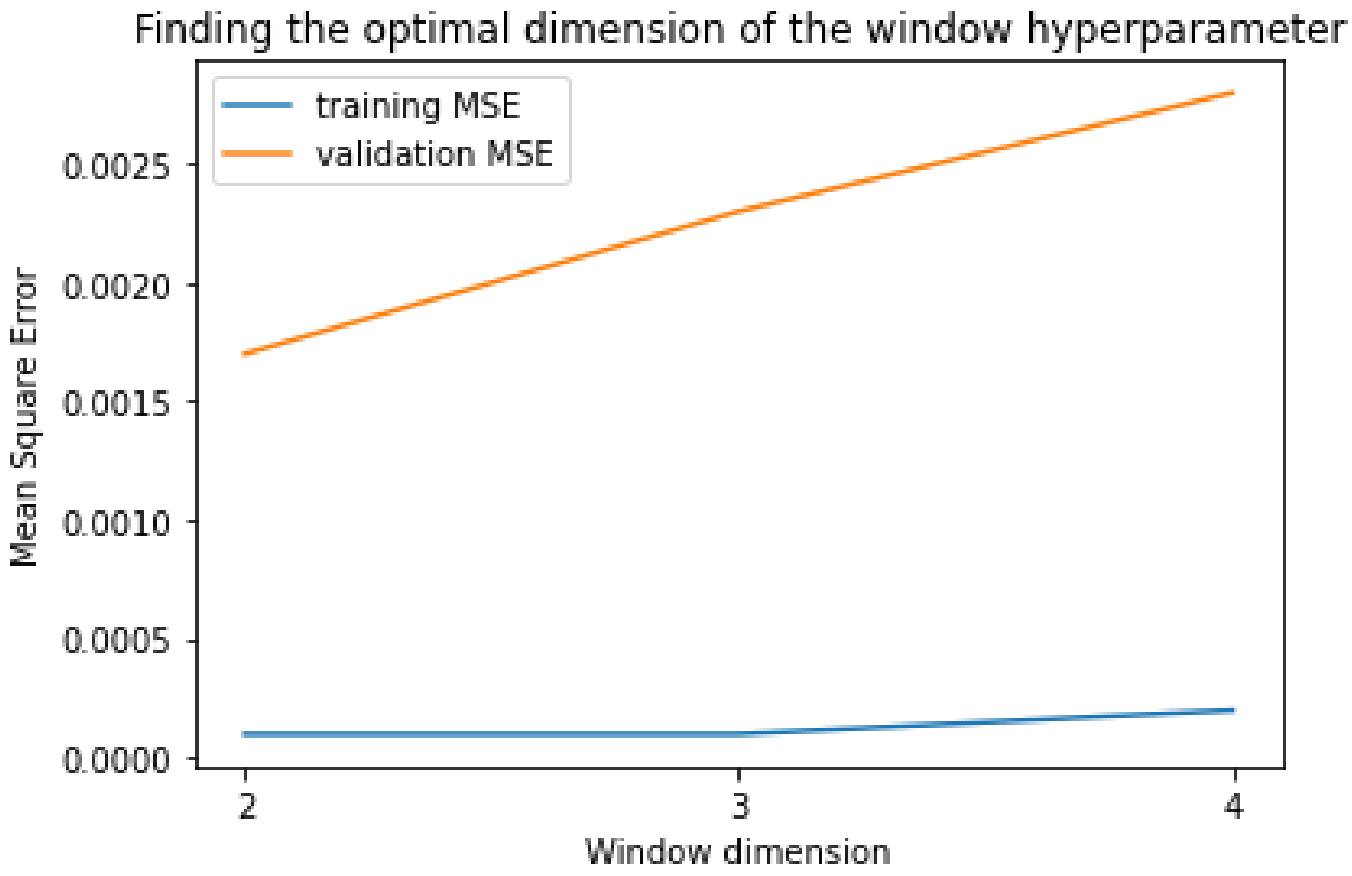



Figura 11: MSE calcolato al variare della dimensione della finestra.

La configurazione del modello LSTM che offre MSE più piccolo si ottiene con window = 2.

### 3.4.8. Decay

Con il termine Decay intendiamo il parametro che regola il learning rate dell'algoritmo di ottimizzazione. Nel nostro modello abbiamo deciso di utilizzare Adam come algoritmo di ottimizzazione, essendo quello più largamente usato. Per eseguire i test è stata usato un modello LSTM così configurato.

- 1 hidden layer, dropout = 0.1, batch = 4, epoch = 512, units = 128.

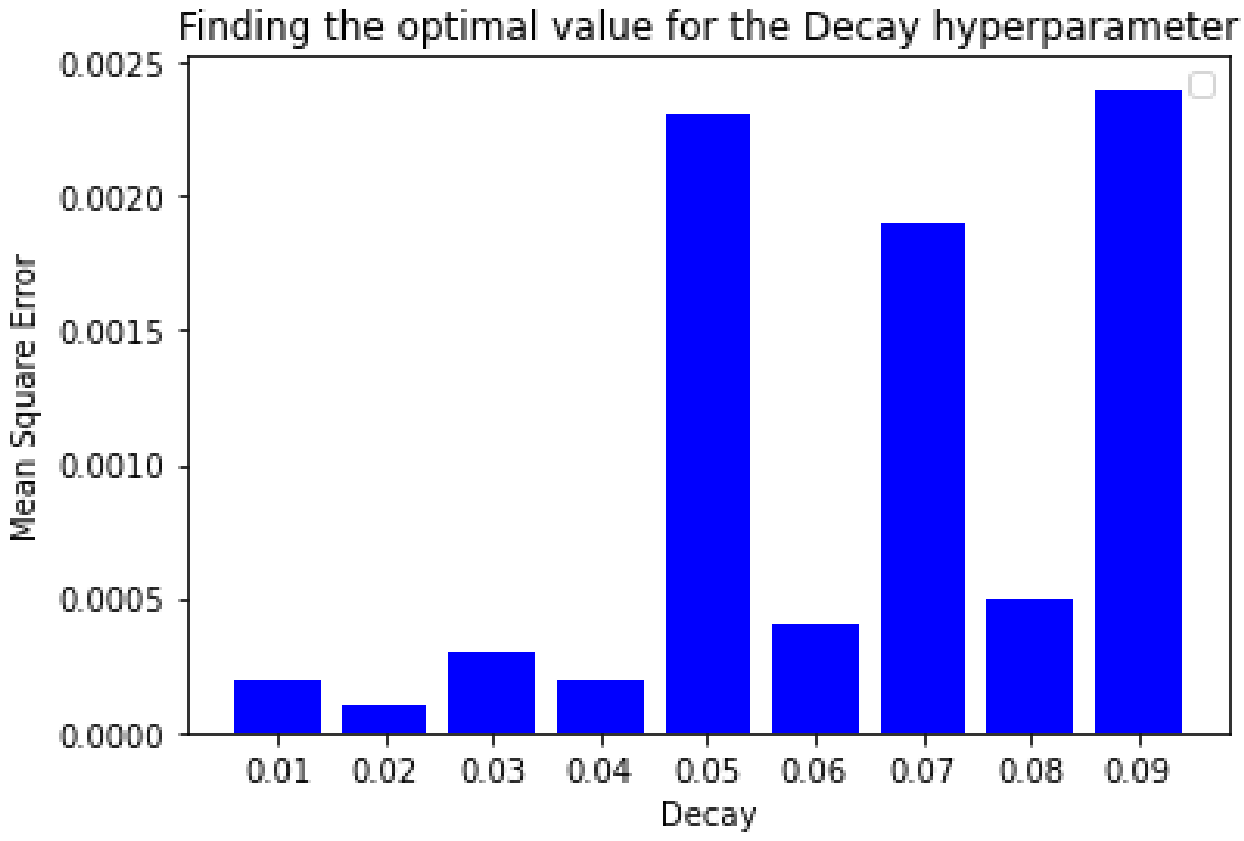



Figura 11: MSE ottenuto al variare del valore di Decay.

Nel nostro caso la configurazione che ha ottenuto un MSE minore è stata con Decay = 0.02.

## 4. Analisi dei risultati

Abbiamo ottenuto i valori ottimi degli iperparametri per il nostro modello andando a studiarli singolarmente. Abbiamo testato gli iperparametri uno per volta, andando a variare il valore di un singolo iperparametro per una data configurazione, e abbiamo confrontato i valori di MSE ottenuti dalle varie prove empiriche. I valori scelti sono quelli che ci hanno garantito MSE più piccoli nelle varie prove empiriche. Andando a mettere insieme i valori ottimi degli iperparametri ottenuti nelle varie prove otteniamo un modello LSTM con delle buone prestazioni.

- `Test Score: 111.68 RMSE`

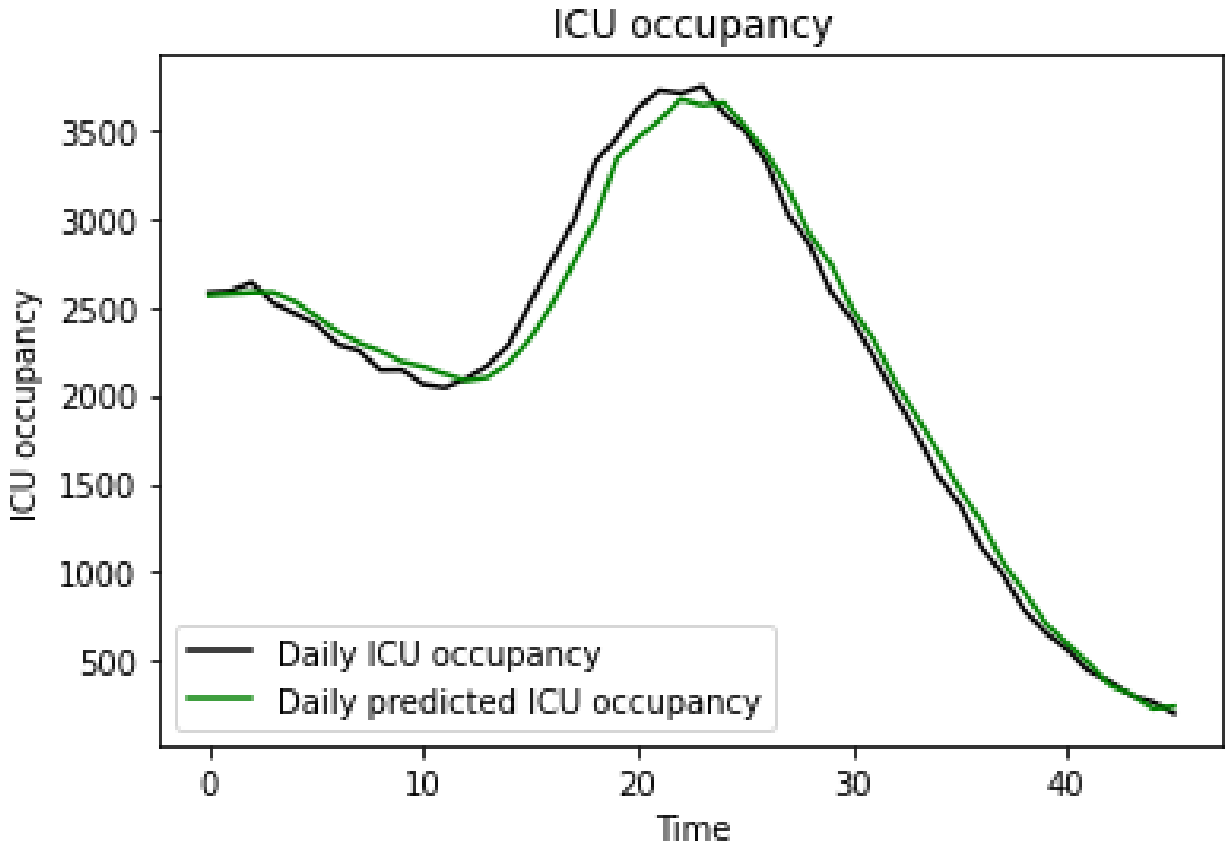



Figura 12: Grafico che illustra le previsioni eseguite dal 04/01/2021 al 03/07/2021. Si effettua una previsione ogni quattro giorni.

- `MSE training set: 0.0001 MSE validation set: 0.0010`


Figura 13: MSE modello finale

Dalla figura 12 possiamo osservare come il grafico delle predizioni (verde) si sovrappone quasi completamente al grafico dei valori reali (nero). Si può notare che il modello sovrastima il numero di posti letto occupati in terapia intensiva nel periodo che va indicativamente da fine gennaio 2021 a metà febbraio 2021. Successivamente il modello sottostima le previsioni che vanno da fine febbraio a inizio aprile. Infine, possiamo notare che la precisione del modello nella parte decrescente della curva (periodo che va indicativamente da metà aprile a inizio luglio) sia migliore rispetto alle parti precedenti. Complessivamente registriamo un RMSE di 111.68, il miglior valore ottenuto fino a questo punto.

Osservando la figura 13 possiamo notare che il modello non presenta problemi di overfitting, visto che la curva del MSE calcolato sul test set (arancione) si avvicina sempre di più alla curva del MSE calcolato sul training set (blu), mantenendo un andamento costante.

Inoltre, un fattore che sicuramente ha influenza sul modello è la dimensione del dataset di training. Nel nostro caso abbiamo un dataset di training formato da 142 record; andando a impostare la dimensione della finestra a 2 otteniamo 140 samples. Su questi samples è stata allenata la rete LSTM. Con un numero così piccolo di samples è difficile ottenere un modello particolarmente preciso nel fare previsioni.

## 5. Codice in Python

Riporto di seguito il codice utilizzato per costruire la rete LSTM finale ed effettuare i test sulle varie configurazioni.

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
 #Loading the Dataset
dataset = pd.read_csv('/content/drive/MyDrive/Colab Notebooks/italy/merged_final
.csv')
real_ICU_access=dataset.iloc[lambda x: x.index % 4 == 0, 11].values
real_ICU_access=real_ICU_access[2:]
data_test=dataset.iloc[lambda x: x.index % 4 == 0,[3,4,5,6,11,12]].values
training_set=dataset.iloc[lambda x: x.index % 4 != 0,[3,4,5,6,11,12]].values
 #Features Scaling
from sklearn.preprocessing import MinMaxScaler
sc = MinMaxScaler(feature_range = (0, 1))
training_set_scaled = sc.fit_transform(training_set)
 #Creating Data with Timesteps
X_train = []
y_train = []
for i in range(2, len(training_set)):
    X_train.append(training_set_scaled[i-2:i,:])
    y_train.append(training_set_scaled[i, 4])
X_train, y_train = np.array(X_train), np.array(y_train)
 #Creating Data with Timesteps
inputs = sc.transform(data_test)
X_test = []
Y_test=[]
for i in range(2,len(data_test)):
    X_test.append(inputs[i-2:i,:])
    Y_test.append(inputs[i,4])
X_test,Y_test = np.array(X_test),np.array(Y_test)
```

```
#Building the LSTM
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
from keras.layers import Dropout
from tensorflow import keras

regressor = Sequential()

regressor.add(LSTM(units = 128, return_sequences = False, input_shape = (X_tra
in.shape[1], X_train.shape[2])))

regressor.add(Dropout(0.1))

regressor.add(Dense(units = 1,activation='linear'))
regressor.summary()
opt = keras.optimizers.Adam(learning_rate=0.02)
regressor.compile(optimizer = opt , loss = 'mean_squared_error')

history=regressor.fit(X_train, y_train,validation_data=(X_test,Y_test), epochs
 = 512, batch_size = 4)


#Make prediction
predicted_ICU_accesses = regressor.predict(X_test)
result=predicted_ICU_accesses.flatten()
matrix_result=np.array([])
for i in range(len(result)):
  row=np.array([])
  for j in range(6):
    row=np.append(row,result[i])
  matrix_result=np.append(matrix_result,row)
matrix_result=np.reshape(matrix_result,(-1,6))
predicted_ICU_accesses = sc.inverse_transform(matrix_result)
print(predicted_ICU_accesses[:,4].shape)
#Plotting the results
plt.plot(real_ICU_access, color = 'black', label = 'Daily ICU occupancy')
plt.plot(predicted_ICU_accesses[:,4], color = 'green', label = 'Daily predicte
d ICU occupancy')
plt.title('ICU occupancy')
plt.xlabel('Time')
plt.ylabel('ICU occupancy')
plt.legend()
plt.show()
```

```
#RMSE
import numpy
import matplotlib.pyplot as plt
import math
from sklearn.metrics import mean_squared_error
testScore = math.sqrt(mean_squared_error(real_ICU_access, predicted_ICU_
accesses[:,4]))
print('Test Score: %.2f RMSE' % (testScore))


#Evaluate model
from matplotlib import pyplot
train_mse = regressor.evaluate(X_train, y_train, verbose=0)
test_mse= regressor.evaluate(X_test,Y_test,verbose=0)
print('Train: %.4f Test: %.4f'%(train_mse,test_mse))
pyplot.title('Loss / Mean Squared Error')
pyplot.plot(history.history['loss'], label='train')
pyplot.plot(history.history['val_loss'], label='test')
pyplot.xlabel('epoch')
pyplot.ylabel('loss')
pyplot.legend()
pyplot.show()
```

# 6. Conclusioni

Questo lavoro di tesi ci ha permesso di andare a studiare come implementare un modello LSTM per fare previsioni su serie temporali. Le conclusioni a cui siamo giunti sono che l'accuratezza di un modello LSTM migliora se si amplia il numero di input (features) dati in ingresso alla rete neurale. In particolare, si è potuto constatare come sia importante mettere in relazione dati che ci permettono di andare a contestualizzare e approfondire maggiormente il fenomeno da noi studiato. Ciò, infatti, si traduce nell'ottenere un modello abbastanza accurato nelle previsioni. In conclusione, possiamo affermare che un modello LSTM, sviluppato seguendo gli stessi passi illustrati in questo lavoro di tesi, avrà sicuramente delle buone prestazioni.

## BIBLIOGRAFIA:


- https://medium.com/@FhelDimaano/what-is-machine-learning-885aa35db58b
- https://towardsdatascience.com/introduction-to-machine-learning-for-beginners-eed6024fdb08
- https://towardsdatascience.com/machine-learning-an-introduction-23b84d51e6d0
- https://machinelearningmastery.com/diagnose-overfitting-underfitting-lstm-models/
- https://machinelearningmastery.com/time-series-prediction-lstm-recurrent-neural-networks-python-keras/
- https://machinelearningmastery.com/display-deep-learning-model-training-history-in-keras/
- https://machinelearningmastery.com/reshape-input-data-long-short-term-memory-networks-keras/
- https://machinelearningmastery.com/how-to-choose-loss-functions-when-training-deep-learning-neural-networks/
- *41° "Instant Report COVID-19" di ALTEMS- Alta scuola di economia e management dei sistemi sanitari*
- https://en.wikipedia.org/wiki/Mean_squared_error
- https://www.oecd.org/coronavirus/en/data-insights/intensive-care-beds-capacity