# Report 2 di tesi:

## Tutorial sulla costruzione e sull’addestramento di un sistema di predizione basato su LSTM

### Motivazione:

Al fine di analizzare il dataset fornito sulle affluenze dei taxi, potrebbe essere interessante costruire un modello basato su **LSTM (Long short-term memory)** per effettuare delle predizioni sui dati. La LSTM è una rete neurale artificiale ricorrente che prevede l’utilizzo di connessioni di feedback, particolarmente utilizzata ed efficace per l’individuazione di flussi di traffico anomalo.

### Problema:

Il task proposto è un tutorial per iniziare a capire il funzionamento dei sistemi di predizione e delle reti neurali.

Il tutorial scelto riguardava in particolare un dataset relativo al traffico di passeggeri su voli di linea dal 1949 al 1960.

Il dataset dei passeggeri è strutturato in un file .csv composto da due colonne:

* **Month:** rappresenta il mese dell’anno in cui è stata fatta la rilevazione
* **AirPassengers:** numero dei passeggeri rilevato

Dopo aver caricato il file .csv veniva richiesto di indicizzare il dataframe tramite il valore della data e quindi dividere il dataset in due set, uno di train e uno di test.

Il set di **train** serve per addestrare la rete neurale mentre il set di **test** servirà per testare i risultati ottenuti.

Questo può essere fatto o utilizzando la libreria *sklearn.model\_selection* attraverso il metodo *train\_test\_split,* fornendo la percentuale di dati che si vogliono impiegare per il test e successivamente il resto verrà usato per il train, oppure può essere fatto semplicemente selezionando dei valori all’interno del dataset.

Nel caso trattato da questo tutorial ci interessano i dati degli ultimi 12 mesi per cercare di predire l’andamento dei passeggeri nell’ultimo anno e per questo motivo confluiranno nel set di train tutti i dati mensili tranne gli ultimi 12, che faranno parte del set di test.

Viene poi fatto un piccolo passaggio di **preprocessing** andando a scalare i dati attraverso *MinMaxScaler* che riporta il valore dei dati tra un valore minimo e un valore massimo specificabili (nel nostro caso tra 0 e 1).

Attraverso il modulo di *TimeSeriesGenerator* si può trasformare una serie di dati temporali in batch di dimensione specificabile (nel nostro caso 6) per il test e la validazione.

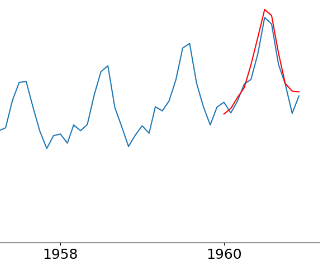
Viene poi usato un modello di tipo *Sequential* che è indicato per quando abbiamo uno stack di livelli in cui ogni livello ha un tensore di input e uno di output. Un **tensore** è un oggetto molto generale, rappresentato solitamente come una matrice a più dimensioni, definito a partire da uno spazio vettoriale **V**.

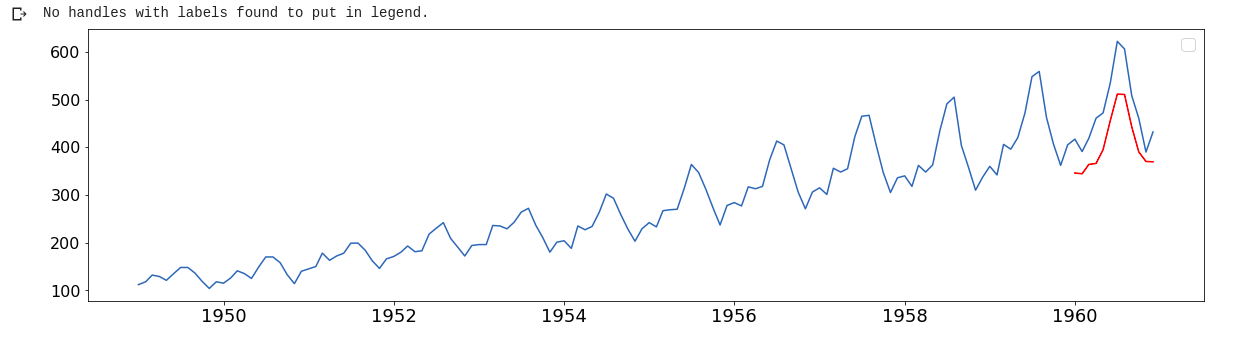
Al modello *Sequential* vengono quindi aggiunti 3 layer, il primo di classe **LSTM** per trattare le time-series, il secondo di classe **DROPOUT** per prevenire il problema dell’overfitting scalando tutti gli input che non sono a 0 a 1/(1 – *rate*) ( *rate* è specificabile dal programmatore ) e infine un layer di classe **DENSE**.

Il modello viene poi compilato e addestrato.

Per l’addestramento ci sono due interfacce disponibili:

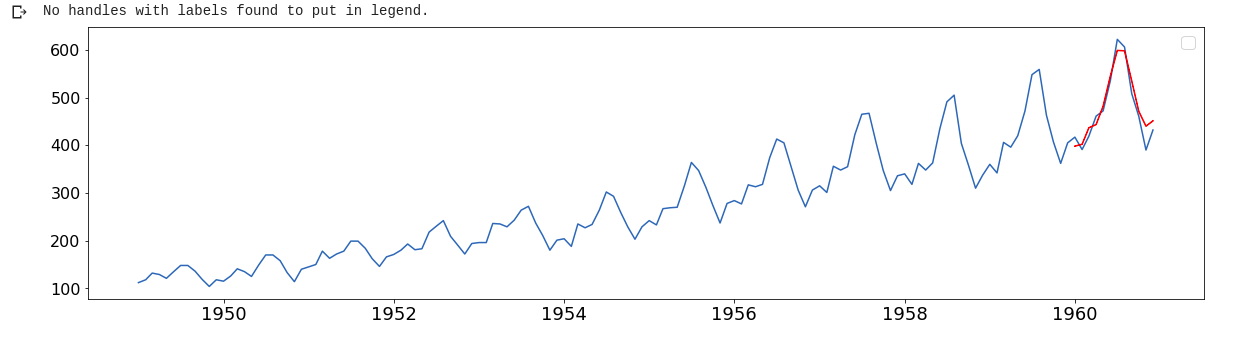
* *fit\_generator():* che prende in ingresso un oggetto di tipo Generator (e quindi anche un *TimeSeriesGenerato*r come abbiamo). Il metodo risulta essere però deprecato.
* -*fit():* chedalla nuova versione di Tensorflow implementa tutte le funzionalità che lo differenziavano dall’altro metodo, ma che in generale ha un’efficienza leggermente minore rispetto a *.fit\_generator().*

  
Fig Figura 1: Plot dei risultati usando .fit\_generator()

  
Fi Figura 2: Plot dei risultati usando .fit()

L’effetto che sembra mitigarsi aumentando le epoche di addestramento del modello.

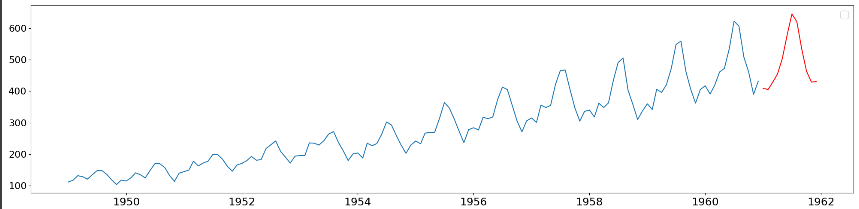
Il concetto di **epoca** è vicino a quello di iterazione e si riferisce ad un ciclo di addestramento su un set di dati fornito alla rete neurale. Aumentando il numero di epoche attraverso le quali si addestra la rete si spera di ottenere una maggiore generalizzazione dell’output. Più epoche sono necessarie quando si divide il dataset in set più piccoli per fornirli separatamente alla rete.

  
Figura 3: Plot dei risultati usando .fit() e addestrando il modello per 120 epoche (**batch\_size: 6)**

Dopo di questo viene creata una lista dove vengono salvati i risultati della predizione che andranno poi plottati come visto nelle figure 1,2 e 3 preparando opportunamente i dati, ovvero invertendo lo scaling effettuato inizialmente attraverso *MinMaxScaler.*

Infine veniva richiesto di plottare i risultati ottenuti confrontandoli con i dati reali.

Veniva poi richiesto di provare a simulare i dati per l’anno seguente, senza avere quindi il confronto con i dati reali.

  
Figu Figura 4: Simulazione completa senza confronto con dati reali. (**epoch: 120, batch\_size: 6)**

### Risultati:

In generale i risultati, osservabili nelle figure 1,2,3 sono abbastanza soddisfacenti in quanto si avvicinano molto ai dati reali ottenibili dal dataset fornito mentre quelli della figura 4, dove si fa una simulazione completa, risultano essere plausibili.

Provando ad eseguire il task con entrambi gli ambienti di programmazione precedentemente analizzati è venuta fuori, in questo caso, la superiorità di Google Colab. Infatti quest’ultimo rispetto a PyCharm è risultato essere molto più veloce e soprattutto pronto all’uso, senza la necessità di risolvere problemi di dipendenze che hanno portato via non poco tempo usando l’altro ambiente di sviluppo.

Fi 

Figura 5: Dati di PyCharm

F 

Figura 6: Dati di Google Colab

Dalle figura 5 e 6 si può notare infatti che mediamente il tempo di esecuzione per step/epoca con Google Colab è nell’ordine delle decine di millisecondi mentre con PyCharm è risultato essere nell’ordine di secondi.

Rimane comunque il fatto che il *loss* , calcolato sull’errore quadratico medio, risulta essere quasi identico in ogni iterazione.

Sono state fatte delle rilevazioni provando a variare il valore dei parametri caratteristici:

* **Batch\_size:** rappresenta la dimensione dei set in cui viene diviso il dataset prima di venir processato dalla rete neurale
* **Epoch:** numero di epoche per cui viene addestrata la rete

Immagine che contiene screenshot

Descrizione generata automaticamente

Figura 7: **epoch: 90 , batch\_size: 128 (default)**

Immagine che contiene screenshot

Descrizione generata automaticamente

Figura 8: **epoch: 120 , batch\_size: 128 (default)**

Immagine che contiene screenshot

Descrizione generata automaticamente

Figura 9: **epoch: 120 , batch\_size: 16**

Immagine che contiene screenshot

Descrizione generata automaticamente

Figura 0: **epoch 90 , batch\_size: 16**

### Conclusioni:

Il task si è rivelato interessante in quanto ha permesso di iniziare a prendere confidenza con i parametri tipici delle reti neurali e in particolare dei modelli basati su serie temporali.

Le rilevazioni effettuate si sono dimostrate abbastanza precise dopo aver fatto un po’ di tuning dei parametri. Infatti le figure 7,8,9 e 10 sono state inserite per dimostrare l’importanza dei due parametri più caratteristici di questa tipologia di reti neurali. Tutte i risultati ottenuti per le figure 7,8,9 e 10 sono ottenuti usando l’ambiente di Google Colab.