Comprensione dettagliata dei livelli Lencm Autoencoder

Qui analizzeremo una rete di autoencoder LSTM per capirli strato per strato. Esamineremo il flusso di input e output tra i layer e inoltre confronteremo il codificatore automatico LSTM con una normale rete LSTM.

Chitta Ranjan
4 giu · 7 minuti di lettura

Nel mio post precedente, LSTM Autoencoder per la classificazione di eventi rari estremi [1], abbiamo imparato come costruire un autoencoder LSTM per una serie temporale multivariata.

Tuttavia, gli LSTM nel Deep Learning sono un po 'più coinvolti. Comprendere i livelli intermedi LSTM e le sue impostazioni non è semplice. Ad esempio, l'uso `return_sequences` dell'argomento `RepeatVector` e dei `TimeDistributed` livelli può essere fonte di confusione.

I tutorial LSTM hanno ben spiegato la struttura e l'input / output delle celle LSTM, ad esempio [2 , 3]. Ma nonostante le sue peculiarità, si scopre poco che spiega il meccanismo degli strati LSTM che lavorano insieme in una rete.

Qui analizzeremo una rete di autoencoder LSTM per capirli strato per strato. Inoltre, le reti **seq2seq** comunemente usate sono simili agli autotoders LSTM. Quindi, la maggior parte di queste spiegazioni sono applicabili anche a seq2seq.

In questo articolo, useremo un semplice esempio di giocattolo per imparare,

- Significato `return_sequences=True` , `RepeatVector()` e `TimeDistributed()` .
- Comprensione dell'input e dell'output di ciascun livello di rete LSTM.

- Differenze tra una normale rete LSTM e un codificatore automatico LSTM.

Comprensione dell'architettura del modello

Importare prima le nostre necessità.

```
# lstm autoencoder per ricreare
un'importazione in serie  numpy  come  np
da  keras.models  import Sequential
da  keras.layers  import LSTM
da  keras.layers  import Dense
da  keras.layers  import RepeatVector
da  keras.layers  import TimeDistributed

'' '
A UDF per convertire i dati di input
nell'array 3-D come richiesto per la rete LSTM.
'' '

def temporalize (X, y, lookback):
    output_X = []
    output_y = []
    per i inrange (len (X) -lookback-1):
        t = []
        per j in range (1, lookback + 1):
            # Raccogliere passato record rinnovassero il periodo di
lookback
             t.append (X [[(i + j + 1) ]
        ,: ]) output_X.append (t)
        output_y.append (y [i + lookback + 1])
    restituisce output_X, output_y
```

Creazione di un esempio di dati

Creeremo un esempio giocattolo di dati di serie temporali multivariate.

```
# definisci timeseries di input timeseries
 = np.array ([[0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9],
                      [0.1 ** 3, 0.2 ** 3, 0.3 ** 3, 0.4 ** 3, 0,5
** 3, 0,6 ** 3, 0,7 ** 3, 0,8 ** 3, 0,9 ** 3]]).

Transpose () timesteps = timeseries.shape [0]
n_features = timeseries.shape [1]

timeseries
```

```
array([[0.1  , 0.001],
       [0.2  , 0.008],
       [0.3  , 0.027],
       [0.4  , 0.064],
       [0.5  , 0.125],
       [0.6  , 0.216],
       [0.7  , 0.343],
       [0.8  , 0.512],
       [0.9  , 0.729]])
```

Figura 1.1 Set di dati non elaborati.

Come richiesto per le reti LSTM, è necessario rimodellare i dati di input in *n_samples* x *timesteps* x *n_features* . In questo esempio, `n_features` è 2. Faremo `timesteps = 3` . Con questo, il risultante `n_samples` è 5 (poiché i dati di input hanno 9 righe).

```
timesteps = 3
X, y = temporalize (X = timeseries, y = np.zeros (len (timeseries)),
lookback = timesteps)

n_features = 2
X = np.array (X)
X = X.reshape (X.shape [ 0], timesteps, n_features)

X
```

```
array([[[0.3  , 0.027],
        [0.4  , 0.064],
        [0.5  , 0.125]],

       [[0.4  , 0.064],
        [0.5  , 0.125],
        [0.6  , 0.216]],

       [[0.5  , 0.125],
        [0.6  , 0.216],
        [0.7  , 0.343]],

       [[0.6  , 0.216],
        [0.7  , 0.343],
        [0.8  , 0.512]],

       [[0.7  , 0.343],
        [0.8  , 0.512],
        [0.9  , 0.729]]])
```

Input data converted into 3D array of size n_samples x timesteps x n_features

Figura 1.2 Dati trasformati in un array 3D per una rete LSTM.

Comprensione di una struttura del codificatore automatico LSTM

In questa sezione, creeremo una rete di codificatore automatico LSTM e visualizzeremo la sua architettura e il flusso di dati. Vedremo anche una normale rete LSTM per confrontare e contrastare le sue differenze con un Autoencoder.

Definizione di un codificatore automatico LSTM.

```
# define model
 model = Sequential ()
model.add (LSTM (128, activation = 'relu', input_shape = (timesteps,
n_features), return_sequences = True ))
model.add (LSTM (64, activation = 'relu', return_sequences = Falso
))
model.add (RepeatVector (timesteps))
model.add (LSTM (64, Activation = 'relu', return_sequences = True ))
model.add (LSTM (128, Activation '' Relu ', return_sequences = True
) )
model.add (TimeDistributed (Dense (n_features)))
model.compile (optimizer = 'adam', loss = 'mse')
model.summary ()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_1 (LSTM)                (None, 3, 128)            67072
_________________________________________________________________
lstm_2 (LSTM)                (None, 64)                49408
_________________________________________________________________
repeat_vector_1 (RepeatVecto (None, 3, 64)             0
_________________________________________________________________
lstm_3 (LSTM)                (None, 3, 64)             33024
_________________________________________________________________
lstm_4 (LSTM)                (None, 3, 128)            98816
_________________________________________________________________
time_distributed_1 (TimeDist (None, 3, 2)              258
=================================================================
Total params: 248,578
Trainable params: 248,578
Non-trainable params: 0
_________________________________________________________________
```

Figura 2.1 Riepilogo del modello di LSTM Autoencoder.

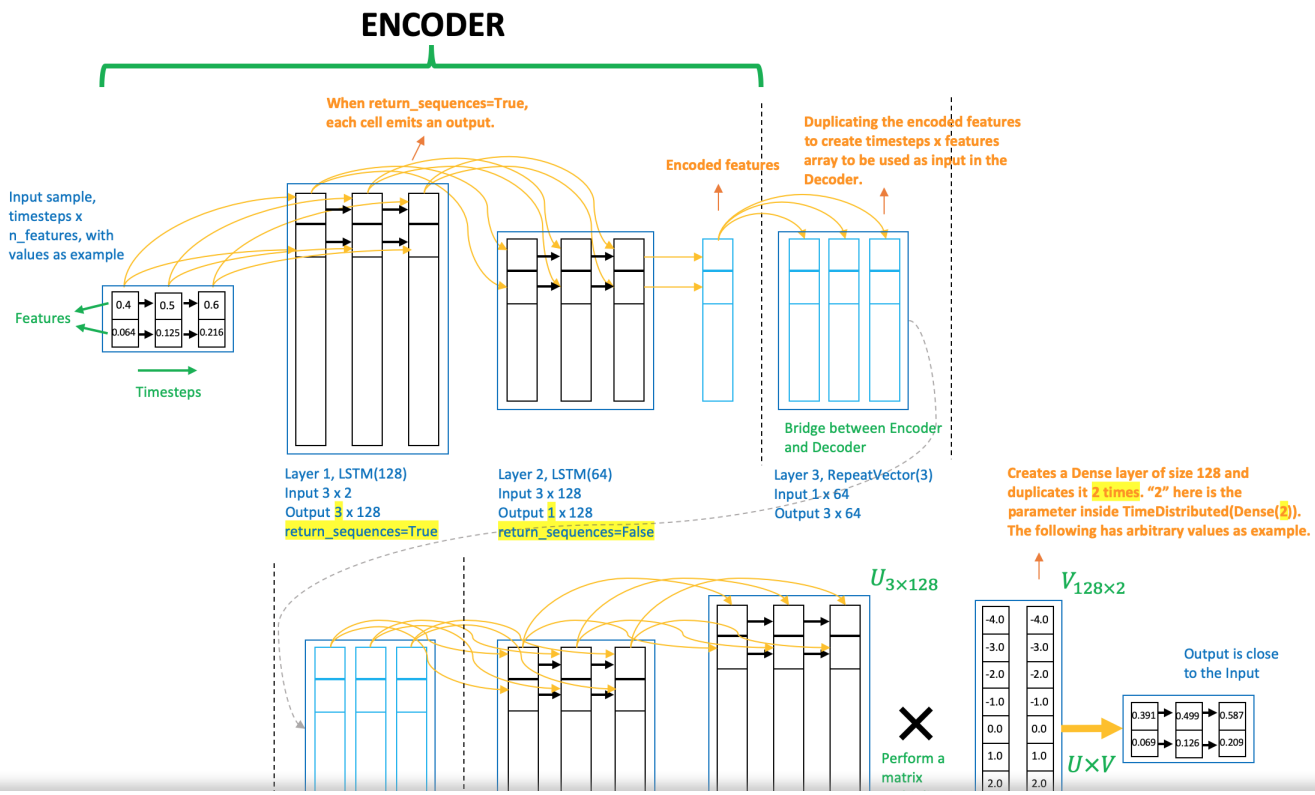
```
# fit model
 model.fit (X, X, epoche = 300, batch_size = 5, verbose = 0)
# dimostrazione della ricostruzione
 yhat = model.predict (X, verbose = 0)
print ('--- Predicted ---')
print (np.round (yhat, 3))
print ('--- Actual ---')
print (np.round (X, 3))
```

```
---Predicted---
[[[0.323 0.041]
  [0.423 0.069]
  [0.494 0.121]]

 [[0.391 0.069]
  [0.499 0.126]
  [0.587 0.209]]

 [[0.491 0.119]
```

```
   [0.596 0.216]
   [0.699 0.344]]

  [[0.596 0.203]
   [0.693 0.34 ]
   [0.808 0.513]]

  [[0.701 0.347]
   [0.798 0.509]
   [0.892 0.723]]]
---Actual---
[[[0.3   0.027]
  [0.4   0.064]
  [0.5   0.125]]

 [[0.4   0.064]
  [0.5   0.125]
  [0.6   0.216]]

 [[0.5   0.125]
  [0.6   0.216]
  [0.7   0.343]]

 [[0.6   0.216]
  [0.7   0.343]
  [0.8   0.512]]

 [[0.7   0.343]
  [0.8   0.512]
  [0.9   0.729]]]
```

Figura 2.2 Ricostruzione dell'ingresso del codificatore automatico LSTM.

Il `model.summary()` fornisce una sintesi del modello di architettura. Per una migliore comprensione, visualizziamolo nella Figura 2.3 di seguito.



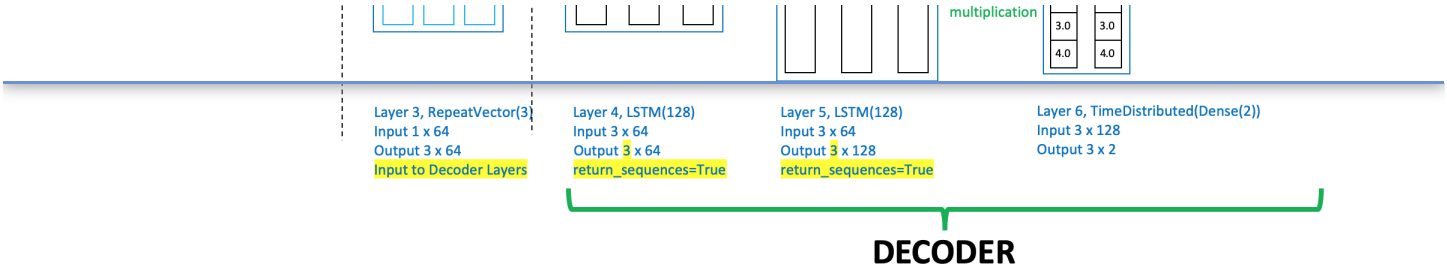


Figura 2.3 Diagramma di flusso dell'encoder automatico LSTM.

Il diagramma illustra il flusso di dati attraverso i livelli di una rete di codificatore automatico LSTM per un campione di dati. Un campione di dati è un'istanza di un set di dati. Nel nostro esempio, un campione è un sotto-array di dimensioni 3x2 nella Figura 1.2.

Da questo diagramma apprendiamo

- La rete LSTM accetta un array 2D come input.
- Uno strato di LSTM ha tante celle quanti sono i timestep.
- L'impostazione `return_sequences=True` rende ogni cella per timestep emette un segnale.
- Ciò diventa più chiaro nella Figura 2.4, che mostra la differenza tra `return_sequences` as `True` (Fig. 2.4a) vs `False` (Fig. 2.4b).

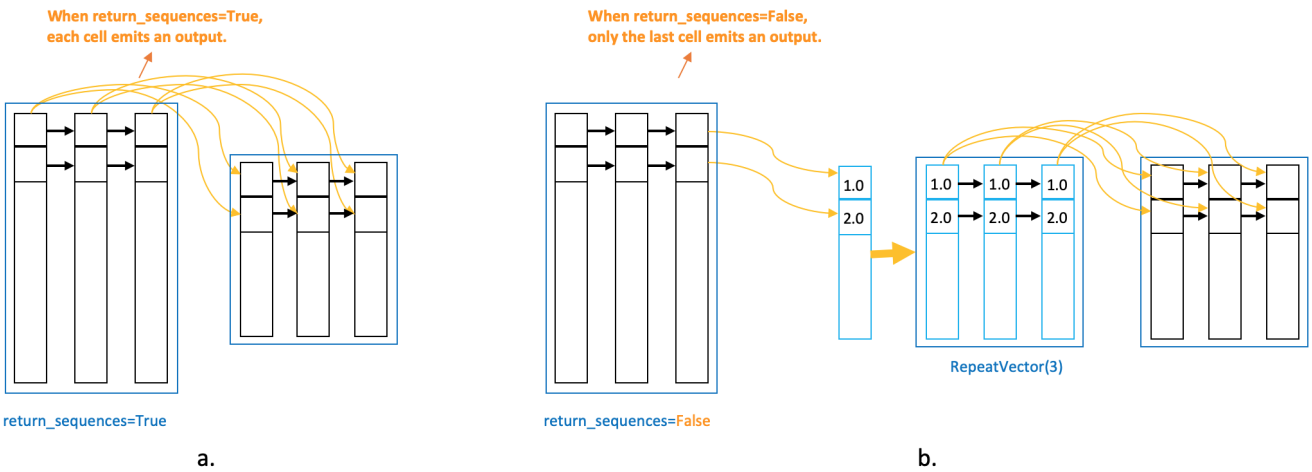


Figura 2.4 Differenza tra return_sequences come True e False.

- In Fig. 2.4a, il segnale da una cella timestep in uno strato è ricevuto dalla cella dello stesso timestep nello strato successivo.
- Nei moduli encoder e decoder in un autoencoder LSTM, è importante avere

connessioni dirette tra le rispettive celle timestep in strati LSTM consecutivi come in Fig 2.4a.

- In Fig. 2.4b, solo l'ultima cella timestep emette segnali. L'output è quindi **un vettore** .
- Come mostrato in Fig. 2.4b, se il livello successivo è LSTM, dupliciamo questo vettore usando `RepeatVector(timesteps)` per ottenere un array 2D per il livello successivo.
- Non è richiesta alcuna trasformazione se il livello successivo è `Dense` (poiché un `Dense` livello prevede un vettore come input).

Ritornare all'encoder LSTM in Fig 2.3.

- I dati di input hanno 3 timestep e 2 funzioni.
- Livello 1, LSTM (128), legge i dati di input e genera 128 funzionalità con 3 timestep per ciascuno perché `return_sequences=True` .
- Il livello 2, LSTM (64), accetta l'input 3x128 dal livello 1 e riduce la dimensione della funzione a 64. Da allora `return_sequences=False` , genera un vettore della dimensione 1x64.
- L'output di questo layer è il **vettore** di **feature codificato** dei dati di input.
- Questo vettore di funzionalità codificato può essere estratto e utilizzato come compressione di dati o funzionalità per qualsiasi altro apprendimento supervisionato o non supervisionato (nel prossimo post vedremo come estrarlo).
- Livello 3, RepeatVector (3), replica il vettore della funzione 3 volte.
- Il livello RepeatVector funge da ponte tra i moduli encoder e decoder.
- Prepara l'input di array 2D per il primo livello LSTM in Decoder.
- Il livello Decoder è progettato per spiegare la *codifica*.
- Pertanto, i livelli del decodificatore sono impilati nell'ordine inverso dell'encoder.
- Livello 4, LSTM (64) e Livello 5, LSTM (128), sono rispettivamente le immagini speculari del Livello 2 e del Livello 1.
- Il livello 6, TimeDistributed (Dense (2)), viene aggiunto alla fine per ottenere

l'output, dove "2" è il numero di funzionalità nei dati di input.

- Il layer TimeDistributed crea un vettore di lunghezza pari al numero di feature emesse dal layer precedente. In questa rete, il livello 5 fornisce 128 funzionalità. Pertanto, il livello TimeDistributed crea un vettore lungo 128 e lo duplica 2 (= n_features) volte.

- L'output di Layer 5 è un array 3x128 che denotiamo come U e quello di TimeDistributed in Layer 6 è un array 128x2 indicato come V. Una moltiplicazione di matrice tra U e V produce un output 3x2.

- L'obiettivo del montaggio della rete è di rendere questo output vicino all'ingresso. Si noti che questa stessa rete ha garantito che le dimensioni di input e output corrispondessero.

Confronto del codificatore automatico LSTM con una normale rete LSTM

La comprensione di cui sopra diventa più chiara quando la confrontiamo con una normale rete LSTM creata per ricostruire gli input.

```
# define model
 model = Sequential ()
model.add (LSTM (128, activation = 'relu', input_shape = (timesteps,
n_features), return_sequences = True ))
model.add (LSTM (64, activation = 'relu', return_sequences = True ))
model.add (LSTM (64, Activation = 'relu', return_sequences = True ))
model.add (LSTM (128, Activation = 'relu', return_sequences = True
))
model.add (TimeDistributed (Dense ( n_features)))
model.compile (optimizer = 'adam', loss = 'mse')
model.summary ()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_1 (LSTM)                (None, 3, 128)            67072
_________________________________________________________________
lstm_2 (LSTM)                (None, 3, 64)             49408
_________________________________________________________________
lstm_3 (LSTM)                (None, 3, 64)             33024
_________________________________________________________________
lstm_4 (LSTM)                (None, 3, 128)            98816
_________________________________________________________________
time_distributed_1 (TimeDist (None, 3, 2)              258
=================================================================
Total params: 248,578
Trainable params: 248,578
Non-trainable params: 0
_________________________________________________________________
```

Figura 3.1 Riepilogo del modello di LSTM Autoencoder.

```
# fit model
 model.fit (X, X, epoche = 300, batch_size = 5, verbose = 0)
# dimostrazione della ricostruzione
 yhat = model.predict (X, verbose = 0)
print ('--- Predicted ---')
print (np.round (yhat, 3))
print ('--- Actual ---')
print (np.round (X, 3))
```

```
---Predicted---
[[[0.306 0.026]
  [0.399 0.064]
  [0.5   0.124]]

 [[0.393 0.064]
  [0.502 0.124]
  [0.599 0.215]]

 [[0.502 0.126]
  [0.6   0.214]
  [0.7   0.343]]

 [[0.596 0.219]
  [0.699 0.344]
  [0.798 0.51 ]]

 [[0.703 0.34 ]
  [0.8   0.512]
  [0.899 0.727]]]
---Actual---
[[[0.3   0.027]
  [0.4   0.064]
  [0.5   0.125]]

 [[0.4   0.064]
  [0.5   0.125]
  [0.6   0.216]]

 [[0.5   0.125]
  [0.6   0.216]
  [0.7   0.343]]

 [[0.6   0.216]
  [0.7   0.343]
  [0.8   0.512]]

 [[0.7   0.343]
  [0.8   0.512]
  [0.9   0.729]]]
```

Figura 3.2 Ricostruzione dell'input della normale rete LSTM.

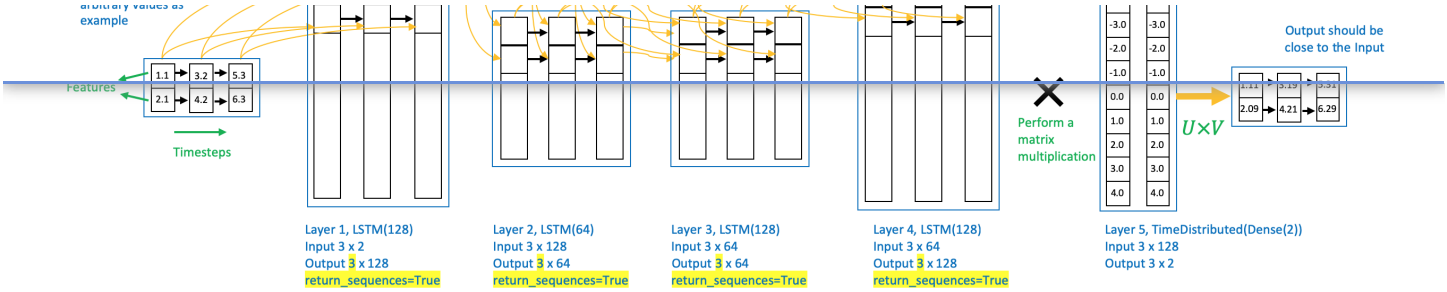


Figura 3.3 Diagramma di flusso della rete LSTM regolare.

Differenze tra la rete LSTM normale e il codificatore automatico LSTM

- Stiamo usando `return_sequences=True` in tutti i livelli LSTM.
- Ciò significa che ogni livello genera un array 2D contenente ciascun timestep.
- Pertanto, non esiste un vettore di feature codificato unidimensionale come output di alcun livello intermedio. Pertanto, la codifica di un campione in un vettore di funzionalità non sta avvenendo.
- **L'assenza di questo** vettore di **codifica** differenzia la normale rete LSTM per la ricostruzione da un codificatore automatico LSTM.
- Tuttavia, si noti che il numero di parametri è lo stesso in entrambi, Autoencoder (Fig. 2.1) e Rete normale (Fig. 3.1).
- Questo perché, il `RepeatVector` livello aggiuntivo in Autoencoder non ha alcun parametro aggiuntivo.
- Soprattutto, **le precisioni di ricostruzione di entrambe le reti sono simili** .

Cibo per la mente

La classificazione degli eventi rari che utilizza l'approccio di rilevazione delle anomalie discussa in LSTM Autoencoder per la classificazione degli eventi rari [1] sta addestrando un LSTM Autoencoder per rilevare gli eventi rari. L'obiettivo della rete Autoencoder in [1] è ricostruire l'input e classificare i campioni mal ricostruiti come un evento raro.

Da allora, possiamo anche costruire una normale rete LSTM per ricostruire i dati di una serie temporale come mostrato nella Figura 3.3, **ciò migliorerà i risultati?**

L'ipotesi alla base è che

> *a causa dell'assenza di uno strato di codifica, in alcuni casi l'accuratezza della ricostruzione può essere migliore (poiché la dimensione tempo-dimensione non viene ridotta). A meno che il vettore codificato non sia necessario per qualsiasi altra analisi, vale la pena provare una normale rete LSTM per una classificazione di eventi rari.*

Github Repository

Il codice completo è disponibile qui .

Conclusione

In questo articolo, noi

- ha lavorato con un esempio giocattolo per comprendere una rete LSTM strato per strato.
- compreso il flusso di input e output da e tra ciascun layer.
- compreso il significato di `return_sequences` , `RepeatVector()` e `TimeDistributed()` .
- ha confrontato e confrontato un autoencoder LSTM con una normale rete LSTM.

. . .

Nel prossimo articolo impareremo come ottimizzare una rete: ***come decidere di aggiungere un nuovo livello e le sue dimensioni*** ?

Riferimenti

1. Auto-codificatore LSTM per la classificazione di eventi rari estremi in telecamere
2. Un'introduzione delicata agli autoencoder LSTM
3. Comprensione di LSTM e dei suoi diagrammi

Apprendimento automatico Apprendimento profondo LSTM Codificatore automatico

Verso la scienza dei dati

A proposito di AiutoLegale