Step-by-step understanding LSTM Autoencoder layers - Towards Data Science

Chitta Ranjan

Here we will break down an LSTM autoencoder network to understand them layer-by-layer. We will go over the input and output flow between the layers, and also, compare the LSTM Autoencoder with a regular LSTM network.

<<Download the free book, Understanding Deep Learning, to learn more>>

In my previous post, LSTM Autoencoder for Extreme Rare Event Classification [1], we learned how to build an LSTM autoencoder for a multivariate time-series data.

However, LSTMs in Deep Learning is a bit more involved. Understanding the LSTM intermediate layers and its settings is not straightforward. For example, usage of `return_sequences` argument, and `RepeatVector` and `TimeDistributed` layers can be confusing.

LSTM tutorials have well explained the structure and input/output of LSTM cells, e.g. [2, 3]. But despite its peculiarities, little is found that explains the mechanism of LSTM layers working together in a network.

Here we will break down an LSTM autoencoder network to understand them layer-by-layer. Additionally, the popularly used **seq2seq** networks are similar to LSTM Autoencoders. Hence, most of these explanations are applicable for seq2seq as well.

In this article, we will use a simple toy example to learn,

- Meaning of `return_sequences=True`, `RepeatVector()`, and `TimeDistributed()`.
- Understanding the input and output of each LSTM Network layer.
- Differences between a regular LSTM network and an LSTM Autoencoder.

Understanding Model Architecture

Importing our necessities first.

```
# lstm autoencoder to recreate a timeseries
import numpy as np
from keras.models import Sequential
from keras.layers import LSTM
from keras.layers import Dense
from keras.layers import RepeatVector
from keras.layers import TimeDistributed'''
A UDF to convert input data into 3-D
array as required for LSTM network.
'''

def temporalize(X, y, lookback):
    output_X = []
    output_y = []
    for i in range(len(X)-lookback-1):
        t = []
        for j in range(1,lookback+1):
            # Gather past records upto the lookback period
            t.append(X[[(i+j+1)], :])
        output_X.append(t)
        output_y.append(y[i+lookback+1])
    return output_X, output_y
```

Creating an example data

We will create a toy example of a multivariate time-series data.

```
# define input timeseries
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]])
```

Figure 1.1. Raw dataset.

As required for LSTM networks, we require to reshape an input data into *n_samples* x *timesteps* x *n_features*. In this example, the `n_features` is 2. We will make `timesteps = 3`. With this, the resultant `n_samples` is 5 (as the input data has 9 rows).

```
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

Figure 1.2. Data transformed to a 3D array for an LSTM network.

Understanding an LSTM Autoencoder Structure

In this section, we will build an LSTM Autoencoder network, and visualize its architecture and data flow. We will also look at a regular LSTM Network to compare and contrast its differences with an Autoencoder.

Defining an LSTM Autoencoder.

```
# 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=False))
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
_________________________________________________________________
```

Figure 2.1. Model Summary of LSTM Autoencoder.

```
# fit model
model.fit(X, X, epochs=300, batch_size=5, verbose=0)
# demonstrate reconstruction
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]]]
```

Figure 2.2. Input Reconstruction of LSTM Autoencoder.

The `model.summary()` provides a summary of the model architecture. For a better understanding, let’s visualize it in Figure 2.3 below.

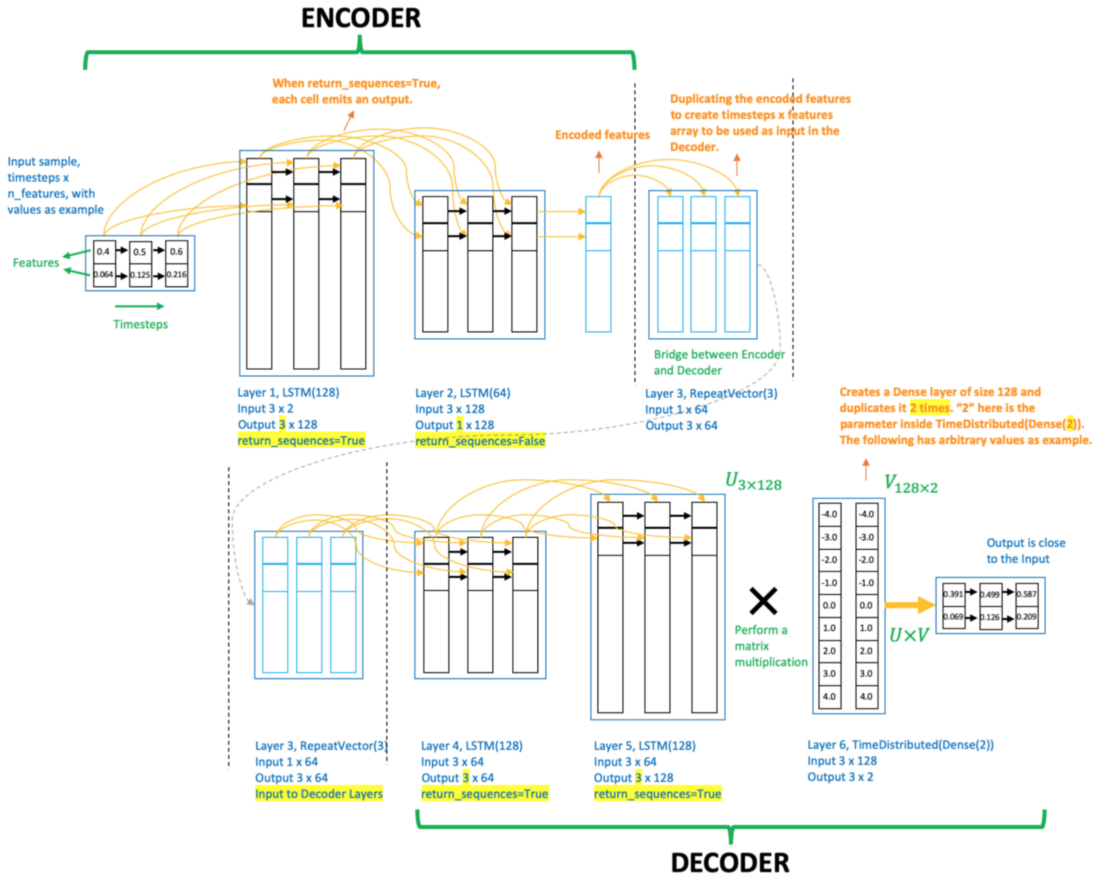

Figure 2.3. LSTM Autoencoder Flow Diagram.

The diagram illustrates the flow of data through the layers of an LSTM Autoencoder network for one sample of data. A sample of data is one instance from a dataset. In our example, one sample is a sub-array of size 3x2 in Figure 1.2.

From this diagram, we learn

- The LSTM network takes a 2D array as input.
- One layer of LSTM has as many cells as the timesteps.
- Setting the `return_sequences=True` makes each cell per timestep emit a signal.
- This becomes clearer in Figure 2.4 which shows the difference between `return_sequences` as `True` (Fig. 2.4a) vs `False` (Fig. 2.4b).

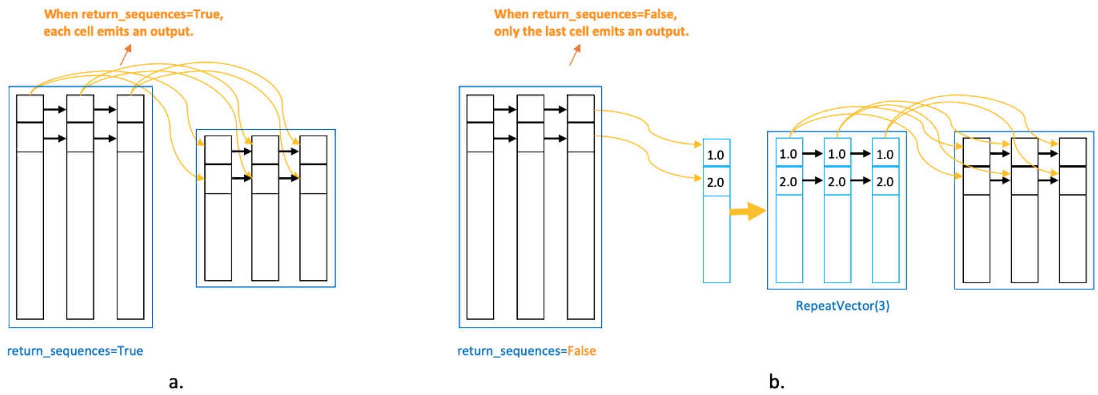

Figure 2.4. Difference between return_sequences as True and False.

- In Fig. 2.4a, signal from a timestep cell in one layer is received by the cell of the same timestep in the subsequent layer.
- In the encoder and decoder modules in an LSTM autoencoder, it is important to have direct connections between respective timestep cells in consecutive LSTM layers as in Fig 2.4a.
- In Fig. 2.4b, only the last timestep cell emits signals. The output is, therefore, **a vector**.
- As shown in Fig. 2.4b, if the subsequent layer is LSTM, we duplicate this vector using `RepeatVector(timesteps)` to get a 2D array for the next layer.
- No transformation is required if the subsequent layer is `Dense` (because a `Dense` layer expects a vector as input).

Coming back to the LSTM Autoencoder in Fig 2.3.

- The input data has 3 timesteps and 2 features.
- Layer 1, LSTM(128), reads the input data and outputs 128 features with 3 timesteps for each because `return_sequences=True`.
- Layer 2, LSTM(64), takes the 3x128 input from Layer 1 and reduces the feature size to 64. Since

`return_sequences=False`, it outputs a feature vector of size 1x64.
- The output of this layer is the **encoded feature vector** of the input data.
- This encoded feature vector can be extracted and used as a data compression, or features for any other supervised or unsupervised learning (in the next post we will see how to extract this).
- Layer 3, RepeatVector(3), replicates the feature vector 3 times.
- The RepeatVector layer acts as a bridge between the encoder and decoder modules.
- It prepares the 2D array input for the first LSTM layer in Decoder.
- The Decoder layer is designed to unfold the *encoding*.
- Therefore, the Decoder layers are stacked in the reverse order of the Encoder.
- Layer 4, LSTM (64), and Layer 5, LSTM (128), are the mirror images of Layer 2 and Layer 1, respectively.
- Layer 6, TimeDistributed(Dense(2)), is added in the end to get the output, where “2” is the number of features in the input data.
- The TimeDistributed layer creates a vector of length equal to the number of features outputted from the previous layer. In this network, Layer 5 outputs 128 features. Therefore, the TimeDistributed layer creates a 128 long vector and duplicates it 2 (= n_features) times.
- The output of Layer 5 is a 3x128 array that we denote as U and that of TimeDistributed in Layer 6 is 128x2 array denoted as V. A matrix multiplication between U and V yields a 3x2 output.
- The objective of fitting the network is to make this output close to the input. Note that this network itself ensured that the input and output dimensions match.

Comparing LSTM Autoencoder with a regular LSTM Network

The above understanding gets clearer when we compare it with a regular LSTM network built for reconstructing the inputs.

```
# 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
_________________________________________________________________
```

Figure 3.1. Model Summary of LSTM Autoencoder.

```
# fit model
model.fit(X, X, epochs=300, batch_size=5, verbose=0)
# demonstrate reconstruction
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]]]
```

Figure 3.2. Input Reconstruction of regular LSTM network.

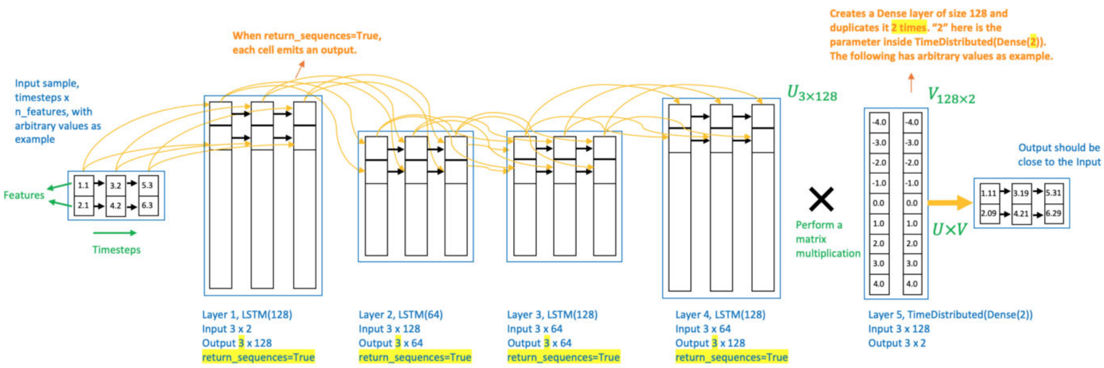

Figure 3.3. Regular LSTM Network flow diagram.

Differences between Regular LSTM network and LSTM Autoencoder

- We are using `return_sequences=True` in all the LSTM layers.
- That means, each layer is outputting a 2D array containing each timesteps.
- Thus, there is no one-dimensional encoded feature vector as output of any intermediate layer. Therefore, encoding a sample into a feature vector is not happening.
- **Absence of this encoding** vector differentiates the regular LSTM network for reconstruction from an LSTM Autoencoder.
- However, note that the number of parameters is the same in both, the Autoencoder (Fig. 2.1) and the Regular network (Fig. 3.1).
- This is because, the extra `RepeatVector` layer in the Autoencoder does not have any additional parameter.
- Most importantly, **the reconstruction accuracies of both Networks are similar**.

Food for thought

The rare-event classification using anomaly detection approach discussed in LSTM Autoencoder for rare-event classification [1] is training an LSTM Autoencoder to detect the rare events. The objective of the Autoencoder network in [1] is to reconstruct the input and classify the poorly reconstructed samples as a rare event.

Since, we can also build a regular LSTM network to reconstruct a time-series data as shown in Figure 3.3, **will that improve the results?**

The hypothesis behind this is,

> due to the absence of an encoding layer the accuracy of reconstruction can be better in some cases (because the dimension time-dimension is not reduced). Unless the encoded vector is required for any other analysis, trying a regular LSTM network is worth a try for a rare-event classification.

Github Repository

The complete code can be found here.

Conclusion

In this article, we

- worked with a toy example to understand an LSTM network layer-by-layer.
- understood the input and output flow from and between each layer.
- understood the meaning of `return_sequences` , `RepeatVector()` , and `TimeDistributed()`.
- compared and contrasted an LSTM Autoencoder with a regular LSTM network.