LSTM

September 14, 2020

Why LSTM? (LSTM : Long Short Time Memory) RNN's can't take care of long term dependencies(later output depends on earlier input), LSTM takes care of it.

LSTM is an advanced RNN. (RNN : Recurrent Neural Network which preserves the sequancial information about data.)

[8]:
```
# LSTM for sequence classification in the IMDB dataset

import numpy
from keras.datasets import imdb
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
from keras.layers.embeddings import Embedding
from keras.preprocessing import sequence

# fix random seed for reproducibility
numpy.random.seed(7)
```

Objective : Data is about IMDB reviews sentiment classification whether the review is positive or negative.

[7]:
```
#Refer: https://keras.io/datasets/#imdb-movie-reviews-sentiment-classification
# the data has pre distribution of train and test data
# load the dataset but only keep the top n words, zero the rest

top_words = 5000
(X_train, y_train), (X_test, y_test) = imdb.load_data(nb_words=top_words)
```

```
WARNING:tensorflow:The `nb_words` argument in `load_data` has been renamed
`num_words`.
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-
datasets/imdb.npz
17465344/17464789 [==============================] - 24s 1us/step
```

[10]:
```
print(X_train[1])
print(type(X_train[1]))
print(len(X_train[1]))
```

```
[1, 194, 1153, 194, 2, 78, 228, 5, 6, 1463, 4369, 2, 134, 26, 4, 715, 8, 118,
1634, 14, 394, 20, 13, 119, 954, 189, 102, 5, 207, 110, 3103, 21, 14, 69, 188,
8, 30, 23, 7, 4, 249, 126, 93, 4, 114, 9, 2300, 1523, 5, 647, 4, 116, 9, 35, 2,
4, 229, 9, 340, 1322, 4, 118, 9, 4, 130, 4901, 19, 4, 1002, 5, 89, 29, 952, 46,
37, 4, 455, 9, 45, 43, 38, 1543, 1905, 398, 4, 1649, 26, 2, 5, 163, 11, 3215, 2,
4, 1153, 9, 194, 775, 7, 2, 2, 349, 2637, 148, 605, 2, 2, 15, 123, 125, 68, 2,
2, 15, 349, 165, 4362, 98, 5, 4, 228, 9, 43, 2, 1157, 15, 299, 120, 5, 120, 174,
11, 220, 175, 136, 50, 9, 4373, 228, 2, 5, 2, 656, 245, 2350, 5, 4, 2, 131, 152,
491, 18, 2, 32, 2, 1212, 14, 9, 6, 371, 78, 22, 625, 64, 1382, 9, 8, 168, 145,
23, 4, 1690, 15, 16, 4, 1355, 5, 28, 6, 52, 154, 462, 33, 89, 78, 285, 16, 145,
95]
<class 'list'>
189
```

[11]:
```python
# truncate and/or pad input sequences
max_review_length = 600
X_train = sequence.pad_sequences(X_train, maxlen=max_review_length)
X_test = sequence.pad_sequences(X_test, maxlen=max_review_length)

print(X_train.shape)
print(X_train[1])
```

```
(25000, 600)
[   0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
```

```
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    1  194 1153  194    2   78  228    5    6
 1463 4369    2  134   26    4  715    8  118 1634   14  394   20   13
  119  954  189  102    5  207  110 3103   21   14   69  188    8   30
   23    7    4  249  126   93    4  114    9 2300 1523    5  647    4
  116    9   35    2    4  229    9  340 1322    4  118    9    4  130
 4901   19    4 1002    5   89   29  952   46   37    4  455    9   45
   43   38 1543 1905  398    4 1649   26    2    5  163   11 3215    2
    4 1153    9  194  775    7    2    2  349 2637  148  605    2    2
   15  123  125   68    2    2   15  349  165 4362   98    5    4  228
    9   43    2 1157   15  299  120    5  120  174   11  220  175  136
   50    9 4373  228    2    5    2  656  245 2350    5    4    2  131
  152  491   18    2   32    2 1212   14    9    6  371   78   22  625
   64 1382    9    8  168  145   23    4 1690   15   16    4 1355    5
   28    6   52  154  462   33   89   78  285   16  145   95]
```

Why padding into the data ?

To speed up the LSTM, it should be provided with a batch of data at a time. To create similarity in the batches of the data we feed up the padded data to the LSTM.

[12]:
```
# create the model (after creation data will be feeded for training it.)

embedding_vecor_length = 32
model = Sequential()
model.add(Embedding(top_words+1, embedding_vecor_length,␣
 ↪input_length=max_review_length))
model.add(LSTM(100))
model.add(Dense(1, activation='sigmoid'))
model.compile(loss='binary_crossentropy', optimizer='adam',␣
 ↪metrics=['accuracy'])
print(model.summary())
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding (Embedding)        (None, 600, 32)           160032
_________________________________________________________________
lstm (LSTM)                  (None, 100)               53200
_________________________________________________________________
dense (Dense)                (None, 1)                 101
=================================================================
Total params: 213,333
Trainable params: 213,333
Non-trainable params: 0
```

```
_________________________________________________________________
None
```

[ ]:
```
#not fully trained yet

model.fit(X_train, y_train, epochs=10, batch_size=64)
# Final evaluation of the model
scores = model.evaluate(X_test, y_test, verbose=0)
print("Accuracy: %.2f%%" % (scores[1]*100))
```

```
Epoch 1/10
391/391 [==============================] - 1523s 4s/step - loss: 0.4831 -
accuracy: 0.7558
Epoch 2/10
373/391 [===========================>..] - ETA: 1:57 - loss: 0.2775 - accuracy:
0.8888
```

Referances : https://machinelearningmastery.com/sequence-classification-lstm-recurrent-neural-networks-python-keras/

https://datascience.stackexchange.com/questions/10615/number-of-parameters-in-an-lstm-model