In [13]:

```
# Credits: https://machinelearningmastery.com/sequence-classification-lstm-recurrent-neural-networks-python-keras/
# 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)
```

In [14]:

```
#Refer: https://keras.io/datasets/#imdb-movie-reviews-sentiment-classification

# 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`.
<string>:6: VisibleDeprecationWarning: Creating an ndarray from ragged nested sequences (which is a list-or-tuple of lists-or-tuples-or ndarrays with different lengths
or shapes) is deprecated. If you meant to do this, you must specify 'dtype=object' when creating the ndarray
/usr/local/lib/python3.7/dist-packages/keras/datasets/imdb.py:155: VisibleDeprecationWarning: Creating an ndarray from ragged nested sequences (which is a list-or-tuple
of lists-or-tuples-or ndarrays with different lengths or shapes) is deprecated. If you meant to do this, you must specify 'dtype=object' when creating the ndarray
  x_train, y_train = np.array(xs[:idx]), np.array(labels[:idx])
/usr/local/lib/python3.7/dist-packages/keras/datasets/imdb.py:156: VisibleDeprecationWarning: Creating an ndarray from ragged nested sequences (which is a list-or-tuple
of lists-or-tuples-or ndarrays with different lengths or shapes) is deprecated. If you meant to do this, you must specify 'dtype=object' when creating the ndarray
  x_test, y_test = np.array(xs[idx:]), np.array(labels[idx:])
```

In [15]:

```
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
```

In [16]:

```
X_train.shape
```

Out[16]:

```
(25000,)
```

In [18]:

```
max(numpy.max(X_test))
```

Out[18]:

```
4998
```

In [19]:

```
# 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]
```

In [20]:

```
# create the model
embedding_vecor_length = 32
model = Sequential()
model.add(Embedding(top_words, 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())
#Refer: https://datascience.stackexchange.com/questions/10615/number-of-parameters-in-an-lstm-model
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding (Embedding)        (None, 600, 32)           160000
_________________________________________________________________
lstm (LSTM)                  (None, 100)               53200
_________________________________________________________________
dense (Dense)                (None, 1)                 101
=================================================================
Total params: 213,301
Trainable params: 213,301
Non-trainable params: 0
_________________________________________________________________
None
```

In [21]:

```
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 [==============================] - 32s 29ms/step - loss: 0.5784 - accuracy: 0.6638
Epoch 2/10
391/391 [==============================] - 11s 29ms/step - loss: 0.3153 - accuracy: 0.8718
Epoch 3/10
365/391 [===========================>..] - ETA: 0s - loss: 0.3356 - accuracy: 0.8614
```

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
<ipython-input-21-afd3f80307e0> in <module>()
----> 1 model.fit(X_train, y_train, epochs=10, batch_size=64)
      2 # Final evaluation of the model
      3 scores = model.evaluate(X_test, y_test, verbose=0)
      4 print("Accuracy: %.2f%%" % (scores[1]*100))

/usr/local/lib/python3.7/dist-packages/keras/engine/training.py in fit(self, x, y, batch_size, epochs, verbose, callbacks, validation_split, validation_data, shuffle, c
lass_weight, sample_weight, initial_epoch, steps_per_epoch, validation_steps, validation_batch_size, validation_freq, max_queue_size, workers, use_multiprocessing)
   1156                 _r=1):
   1157               callbacks.on_train_batch_begin(step)
-> 1158               tmp_logs = self.train_function(iterator)
   1159               if data_handler.should_sync:
   1160                 context.async_wait()

/usr/local/lib/python3.7/dist-packages/tensorflow/python/eager/def_function.py in __call__(self, *args, **kwds)
    887
    888       with OptionalXlaContext(self._jit_compile):
--> 889         result = self._call(*args, **kwds)
    890
    891       new_tracing_count = self.experimental_get_tracing_count()

/usr/local/lib/python3.7/dist-packages/tensorflow/python/eager/def_function.py in _call(self, *args, **kwds)
    915       # In this case we have created variables on the first call, so we run the
    916       # defunned version which is guaranteed to never create variables.
--> 917       return self._stateless_fn(*args, **kwds)  # pylint: disable=not-callable
    918     elif self._stateful_fn is not None:
    919       # Release the lock early so that multiple threads can perform the call

/usr/local/lib/python3.7/dist-packages/tensorflow/python/eager/function.py in __call__(self, *args, **kwargs)
   3022        filtered_flat_args) = self._maybe_define_function(args, kwargs)
   3023     return graph_function._call_flat(
-> 3024         filtered_flat_args, captured_inputs=graph_function.captured_inputs)  # pylint: disable=protected-access
   3025
   3026   @property

/usr/local/lib/python3.7/dist-packages/tensorflow/python/eager/function.py in _call_flat(self, args, captured_inputs, cancellation_manager)
   1959       # No tape is watching; skip to running the function.
   1960       return self._build_call_outputs(self._inference_function.call(
-> 1961           ctx, args, cancellation_manager=cancellation_manager))
   1962     forward_backward = self._select_forward_and_backward_functions(
   1963         args,

/usr/local/lib/python3.7/dist-packages/tensorflow/python/eager/function.py in call(self, ctx, args, cancellation_manager)
    594               inputs=args,
    595               attrs=attrs,
--> 596               ctx=ctx)
    597         else:
    598           outputs = execute.execute_with_cancellation(

/usr/local/lib/python3.7/dist-packages/tensorflow/python/eager/execute.py in quick_execute(op_name, num_outputs, inputs, attrs, ctx, name)
     58     ctx.ensure_initialized()
     59     tensors = pywrap_tfe.TFE_Py_Execute(ctx._handle, device_name, op_name,
---> 60                                         inputs, attrs, num_outputs)
     61   except core._NotOkStatusException as e:
     62     if name is not None:

KeyboardInterrupt:
```

In [ ]: