In [2]:
```
from tensorflow.keras.preprocessing import text_dataset_from_directory
from tensorflow.strings import regex_replace
from tensorflow.keras.layers.experimental.preprocessing import TextVectorization
from tensorflow.keras.models import Sequential
from tensorflow.keras import Input
from tensorflow.keras.layers import Dense, LSTM, Embedding, Dropout

def prepareData(dir):
  data = text_dataset_from_directory(dir)
  return data.map(
    lambda text, label: (regex_replace(text, '<br />', ' '), label),
  )

# Assumes you're in the root level of the dataset directory.
# If you aren't, you'll need to change the relative paths here.
train_data = prepareData('./datasets/train')
test_data = prepareData('./datasets/test')

for text_batch, label_batch in train_data.take(1):
  print(text_batch.numpy()[0])
  print(label_batch.numpy()[0])
```

```
Found 25000 files belonging to 2 classes.
WARNING:tensorflow:From D:\apps\anaconda\envs\dlc\lib\site-packages\tensorflow\python\autograph
\pyct\static_analysis\liveness.py:83: Analyzer.lamba_check (from tensorflow.python.autograph.pyc
t.static_analysis.liveness) is deprecated and will be removed after 2023-09-23.
Instructions for updating:
Lambda fuctions will be no more assumed to be used in the statement where they are used, or at l
east in the same block. https://github.com/tensorflow/tensorflow/issues/56089
Found 25000 files belonging to 2 classes.
b"Sorry to say I have no idea what Hollywood is doing. Sure give us movies like Batman Begins. O
h, by the way Hollywood I think they may cover the story line in the movie Batman, but please do
n't entertain us what we would really want to see Batman and Superman together. I really hated t
his trailer because it left me wanting for more. I was looking around to see when it was coming
out. It was like a terrible practical joke. The graphics where good the story line seemed solid
and it had all the trappings of a great movie. Unfortunately it's not going to happen for now. T
o the producers, directors and all the actors great job but I hate you for doing this to me. You
left me wanting more."
1
```

In [3]:
```
model = Sequential()

# ----- 1. INPUT
# We need this to use the TextVectorization layer next.
model.add(Input(shape=(1,), dtype="string"))

# ----- 2. TEXT VECTORIZATION
# This layer processes the input string and turns it into a sequence of
# max_len integers, each of which maps to a certain token.
max_tokens = 1000
max_len = 100
vectorize_layer = TextVectorization(
  # Max vocab size. Any words outside of the max_tokens most common ones
  # will be treated the same way: as "out of vocabulary" (OOV) tokens.
  max_tokens=max_tokens,
  # Output integer indices, one per string token
  output_mode="int",
  # Always pad or truncate to exactly this many tokens
  output_sequence_length=max_len,
)

# Call adapt(), which fits the TextVectorization layer to our text dataset.
# This is when the max_tokens most common words (i.e. the vocabulary) are selected.
train_texts = train_data.map(lambda text, label: text)
vectorize_layer.adapt(train_texts)

model.add(vectorize_layer)
```

```
# ----- 3. EMBEDDING
# This layer turns each integer (representing a token) from the previous layer
# an embedding. Note that we're using max_tokens + 1 here, since there's an
# out-of-vocabulary (OOV) token that gets added to the vocab.
model.add(Embedding(max_tokens + 1, 128))

# ----- 4. RECURRENT LAYER
model.add(LSTM(64))

# ----- 5. DENSE HIDDEN LAYER
model.add(Dense(64, activation="relu"))

# ----- 6. OUTPUT
model.add(Dense(1, activation="sigmoid"))
```

In [4]:
```
model.compile(loss="binary_crossentropy", optimizer="adam", metrics=["accuracy"])
model.fit(train_data, epochs=10)

model.save_weights('rnn')

model.load_weights('rnn')
```

```
Epoch 1/10
782/782 [==============================] - 67s 69ms/step - loss: 0.5394 - accuracy: 0.7262
Epoch 2/10
782/782 [==============================] - 44s 57ms/step - loss: 0.4418 - accuracy: 0.7976
Epoch 3/10
782/782 [==============================] - 34s 43ms/step - loss: 0.4064 - accuracy: 0.8168
Epoch 4/10
782/782 [==============================] - 38s 48ms/step - loss: 0.3807 - accuracy: 0.8310
Epoch 5/10
782/782 [==============================] - 33s 42ms/step - loss: 0.3643 - accuracy: 0.8412
Epoch 6/10
782/782 [==============================] - 49s 63ms/step - loss: 0.3456 - accuracy: 0.8502
Epoch 7/10
782/782 [==============================] - 42s 53ms/step - loss: 0.3271 - accuracy: 0.8605
Epoch 8/10
782/782 [==============================] - 42s 54ms/step - loss: 0.3144 - accuracy: 0.8660
Epoch 9/10
782/782 [==============================] - 49s 62ms/step - loss: 0.3052 - accuracy: 0.8715
Epoch 10/10
782/782 [==============================] - 44s 55ms/step - loss: 0.2961 - accuracy: 0.8746
```

Out[4]: `<tensorflow.python.checkpoint.checkpoint.CheckpointLoadStatus at 0x1f3556d5fc8>`

In [5]:
```
model.evaluate(test_data)

# Should print a very high score like 0.98.
print(model.predict([
  "i loved it! highly recommend it to anyone and everyone looking for a great movie to watch."
]))

# Should print a very low score like 0.01.
print(model.predict([
  "this was awful! i hated it so much, nobody should watch this. the acting was terrible, the m
]))
```

```
782/782 [==============================] - 55s 67ms/step - loss: 0.5624 - accuracy: 0.7828
1/1 [==============================] - 1s 693ms/step
[[0.9934685]]
1/1 [==============================] - 0s 35ms/step
[[0.01894102]]
```

In [ ]: