Sequence Classification with LSTM using Keras

Sequence classification is a predictive modeling problem where you have some sequence of inputs over space or time and the task is to predict a category for the sequence.

Problem Description:

Our problem is the IMDB movie review sentiment classification problem. Each movie review is a variable sequence of words and the sentiment of each movie review must be classified.

The Large Movie Review Dataset (often referred to as the IMDB dataset) contains 25,000 highly-polar movie reviews (good or bad) for training and the same amount again for testing. The problem is to determine whether a given movie review has a positive or negative sentiment.

Word Embedding Technique:

We will map each movie review into a real vector domain, a popular technique when working with text called word embedding. This is a technique where words are encoded as real-valued vectors in a high dimensional space, where the similarity between words in terms of meaning translates to closeness in the vector space.

Keras provides a convenient way to convert positive integer representations of words into a word embedding by an Embedding layer.

We will map each word onto a 32 length real valued vector. We will also limit the total number of words that we are interested in modeling to the 5000 most frequent words, and zero out the rest. Finally, the sequence length (number of words) in each review varies, so we will constrain each review to be 500 words, truncating long reviews and pad the shorter reviews with zero values.

Now that we have defined our problem and how the data will be prepared and modeled, we are ready to develop an LSTM model to classify the sentiment of movie reviews.

Simple LSTM for Sequence Classification

We start by importing the classes and functions required for this model and initializing the random number generator to a constant value to ensure we can easily reproduce the results

In [1]:
```
import numpy
from keras.datasets import imdb
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import LSTM
from tensorflow.keras.layers import Dropout
from tensorflow.keras.layers import Embedding
from keras.preprocessing import sequence
# fix random seed for reproducibility
numpy.random.seed(7)
```

```
Using TensorFlow backend.
```

We need to load the IMDB dataset. We are constraining the dataset to the top 5,000 words. We also split the dataset into train (50%) and test (50%) sets

In [2]:
```
# 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(num_words=top_words)
```

we need to truncate and pad the input sequences so that they are all the same length for modeling because same length vectors is required to perform the computation in Keras.

In [3]:
```
# truncate and pad input sequences
max_review_length = 500
X_train = sequence.pad_sequences(X_train, maxlen=max_review_length)
X_test = sequence.pad_sequences(X_test, maxlen=max_review_length)
```

define, compile and fit our LSTM model.

The first layer is the Embedded layer that uses 32 length vectors to represent each word. The next layer is the LSTM layer with 100 memory units (smart neurons). Finally, because this is a classification problem we use a Dense output layer with a single neuron and a sigmoid activation function to make 0 or 1 predictions for the two classes (good and bad) in the problem.

since this a binary classification problem, the loss function used is log loss(binary_crossentropy in Keras). The efficient ADAM optimization algorithm is used.

In [4]:
```
# create the model
embedding_vecor_length = 32
model = Sequential()
model.add(Embedding(top_words, embedding_vecor_length, input_length=max_revie
model.add(Dropout(0.2))
model.add(LSTM(100))
model.add(Dropout(0.2))
model.add(Dense(1, activation='sigmoid'))
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accurac
print(model.summary())
model.fit(X_train, y_train, epochs=3, batch_size=64)
```

```
Model: "sequential"
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding (Embedding)        (None, 500, 32)           160000
_________________________________________________________________
dropout (Dropout)            (None, 500, 32)           0
_________________________________________________________________
lstm (LSTM)                  (None, 100)               53200
_________________________________________________________________
dropout_1 (Dropout)          (None, 100)               0
_________________________________________________________________
dense (Dense)                (None, 1)                 101
=================================================================
Total params: 213,301
Trainable params: 213,301
Non-trainable params: 0
_________________________________________________________________
None
Epoch 1/3
391/391 [==============================] - 522s 1s/step - loss: 0.5740 - accur
acy: 0.7287
Epoch 2/3
391/391 [==============================] - 525s 1s/step - loss: 0.6041 - accur
acy: 0.6666
Epoch 3/3
391/391 [==============================] - 556s 1s/step - loss: 0.4377 - accur
```

Out[4]: <tensorflow.python.keras.callbacks.History at 0xe02ec7b3a0>

Once fit, we estimate the performance of the model on unseen reviews.

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

```
Accuracy: 85.38%
```