Sequence Classification with LSTM Recurrent Neural Networks in Python with 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.

What makes this problem difficult is that the sequences can vary in length, be comprised of a very large vocabulary of input symbols and may require the model to learn the long-term context or dependencies between symbols in the input sequence.

We will discover how you can develop LSTM recurrent neural network models for sequence classification problems in Python using the Keras deep learning library.

We will look at the following concepts:

- How to develop an LSTM model for a sequence classification problem.

Problem Description

The problem that we will use to demonstrate sequence learning in this tutorial 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.

The data was collected by Stanford researchers and was used in a 2011 paper where a split of 50-50 of the data was used for training and test. An accuracy of 88.89% was achieved.

Keras provides access to the IMDB dataset built-in. The **imdb.load_data()** function allows you to load the dataset in a format that is ready for use in neural network and deep learning models.

The words have been replaced by integers that indicate the ordered frequency of each word in the dataset. The sentences in each review are therefore comprised of a sequence of integers.

Word Embedding

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 can quickly develop a small LSTM for the IMDB problem and achieve good accuracy.

Let’s start off 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.

```
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)
```

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.

```
# 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)
```

Next, we need to truncate and pad the input sequences so that they are all the same length for modeling. The model will learn the zero values carry no information so indeed the sequences are not the same length in terms of content, but same length vectors is required to perform the computation in Keras.

```
# 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)
```

We can now 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.

Because it is a binary classification problem, log loss is used as the loss function (**binary_crossentropy** in Keras). The efficient ADAM optimization algorithm is used. The model is fit for only 2 epochs because it quickly overfits the problem. A large batch size of 64 reviews is used to space out weight updates.

```
# 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())
model.fit(X_train, y_train, validation_data=(X_test,
y_test), epochs=3, batch_size=64)
```

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

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

Below is the full code listing for this LSTM network on the IMDB dataset.

```
# 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)
# 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)
# 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)
# 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())
model.fit(X_train, y_train, epochs=3, batch_size=64)
# Final evaluation of the model
scores = model.evaluate(X_test, y_test, verbose=0)
print("Accuracy: %.2f%%" % (scores[1]*100))
```

Your results may vary given the stochastic nature of the algorithm or evaluation procedure, or differences in numerical precision. Consider running the example a few times and compare the average outcome.

Running this example produces the following output.

```
Epoch 1/3
16750/16750 [==============================] - 107s -
loss: 0.5570 - acc: 0.7149
Epoch 2/3
16750/16750 [==============================] - 107s -
loss: 0.3530 - acc: 0.8577
Epoch 3/3
16750/16750 [==============================] - 107s -
loss: 0.2559 - acc: 0.9019
Accuracy: 86.79%
```

You can see that this simple LSTM with little tuning achieves near state-of-the-art results on the IMDB problem. Importantly, this is a template that you can use to apply LSTM networks to your own sequence classification problems.