Chapter 26

Project: Sequence Classification of Movie Reviews

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. In this project you will discover how you can develop LSTM recurrent neural network models for sequence classification problems in Python using the Keras deep learning library. After completing this project you will know:

- How to develop an LSTM model for a sequence classification problem.
- How to reduce overfitting in your LSTM models through the use of dropout.
- How to combine LSTM models with Convolutional Neural Networks that excel at learning spatial relationships.

Let's get started.

26.1 Simple LSTM for Sequence Classification

The problem that we will use to demonstrate sequence learning in this tutorial is the IMDB movie review sentiment classification problem, introduced in Section 22.1. 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)
```

Listing 26.1: Import Classes and Functions and Seed Random Number Generator.

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(nb_words=top_words)
```

Listing 26.2: Load and Split the Dataset.

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)
```

Listing 26.3: Left-Pad Sequences to all be the Same 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), nb_epoch=3, batch_size=64)
```

Listing 26.4: Define and Fit LSTM Model for the IMDB Dataset.

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))
```

Listing 26.5: Evaluate the Fit Model.

For completeness, here 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(nb_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, nb_epoch=3, batch_size=64)
# Final evaluation of the model
scores = model.evaluate(X_test, y_test, verbose=0)
print("Accuracy: %.2f%%" % (scores[1]*100))
```

Listing 26.6: Simple LSTM Model for the IMDB Dataset.

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%
```

Listing 26.7: Output from Running the Simple LSTM Model.

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. Now, let's look at some extensions of this simple model that you may also want to bring to your own problems.

26.2 LSTM For Sequence Classification With Dropout

Recurrent Neural networks like LSTM generally have the problem of overfitting. Dropout can be applied between layers using the Dropout Keras layer. We can do this easily by adding new Dropout layers between the Embedding and LSTM layers and the LSTM and Dense output layers. We can also add dropout to the input on the Embedded layer by using the dropout parameter. For example:

```
model = Sequential()
model.add(Embedding(top_words, embedding_vecor_length, input_length=max_review_length,
    dropout=0.2))
model.add(Dropout(0.2))
model.add(LSTM(100))
model.add(Dropout(0.2))
model.add(Dense(1, activation='sigmoid'))
```

Listing 26.8: Define LSTM Model with Dropout Layers.

The full code listing example above with the addition of Dropout layers is as follows:

```
# LSTM with Dropout 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 import Dropout
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(nb_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,
    dropout=0.2))
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=['accuracy'])
print(model.summary())
model.fit(X_train, y_train, nb_epoch=3, batch_size=64)
# Final evaluation of the model
scores = model.evaluate(X_test, y_test, verbose=0)
print("Accuracy: %.2f%%" % (scores[1]*100))
```

Listing 26.9: LSTM Model with Dropout Layers for the IMDB Dataset.

Running this example provides the following output.

```
Epoch 1/3
16750/16750 [==============================] - 108s - loss: 0.5802 - acc: 0.6898
Epoch 2/3
16750/16750 [==============================] - 108s - loss: 0.4112 - acc: 0.8232
Epoch 3/3
16750/16750 [==============================] - 108s - loss: 0.3825 - acc: 0.8365
Accuracy: 85.56%
```

Listing 26.10: Output from Running the LSTM Model with Dropout Layers.

We can see dropout having the desired impact on training with a slightly slower trend in convergence and in this case a lower final accuracy. The model could probably use a few more epochs of training and may achieve a higher skill (try it an see). Alternately, dropout can be applied to the input and recurrent connections of the memory units with the LSTM precisely and separately. Keras provides this capability with parameters on the LSTM layer, the `dropout_W` for configuring the input dropout and `dropout_U` for configuring the recurrent dropout. For example, we can modify the first example to add dropout to the input and recurrent connections as follows:

```
model = Sequential()
model.add(Embedding(top_words, embedding_vecor_length, input_length=max_review_length,
    dropout=0.2))
model.add(LSTM(100, dropout_W=0.2, dropout_U=0.2))
model.add(Dense(1, activation='sigmoid'))
```

Listing 26.11: Define LSTM Model with Dropout on Gates.

The full code listing with more precise LSTM dropout is listed below for completeness.

```
# LSTM with dropout 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(nb_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,
    dropout=0.2))
model.add(LSTM(100, dropout_W=0.2, dropout_U=0.2))
model.add(Dense(1, activation='sigmoid'))
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
print(model.summary())
model.fit(X_train, y_train, nb_epoch=3, batch_size=64)
# Final evaluation of the model
scores = model.evaluate(X_test, y_test, verbose=0)
print("Accuracy: %.2f%%" % (scores[1]*100))
```

Listing 26.12: LSTM Model with Dropout on Gates for the IMDB Dataset.

Running this example provides the following output.

```
Epoch 1/3
16750/16750 [==============================] - 112s - loss: 0.6623 - acc: 0.5935
Epoch 2/3
16750/16750 [==============================] - 113s - loss: 0.5159 - acc: 0.7484
Epoch 3/3
16750/16750 [==============================] - 113s - loss: 0.4502 - acc: 0.7981
Accuracy: 82.82%
```

Listing 26.13: Output from Running the LSTM Model with Dropout on the Gates.

We can see that the LSTM specific dropout has a more pronounced effect on the convergence of the network than the layer-wise dropout. As above, the number of epochs was kept constant and could be increased to see if the skill of the model can be further lifted. Dropout is a powerful technique for combating overfitting in your LSTM models and it is a good idea to try both methods, but you may bet better results with the gate-specific dropout provided in Keras.

26.3 LSTM and CNN For Sequence Classification

Convolutional neural networks excel at learning the spatial structure in input data. The IMDB review data does have a one-dimensional spatial structure in the sequence of words in reviews and the CNN may be able to pick out invariant features for good and bad sentiment. This learned spatial features may then be learned as sequences by an LSTM layer. We can easily add a one-dimensional CNN and max pooling layers after the Embedding layer which then feed the consolidated features to the LSTM. We can use a smallish set of 32 features with a small filter length of 3. The pooling layer can use the standard length of 2 to halve the feature map size. For example, we would create the model as follows:

```
model = Sequential()
model.add(Embedding(top_words, embedding_vecor_length, input_length=max_review_length))
model.add(Convolution1D(nb_filter=32, filter_length=3, border_mode='same',
    activation='relu'))
model.add(MaxPooling1D(pool_length=2))
model.add(LSTM(100))
model.add(Dense(1, activation='sigmoid'))
```

Listing 26.14: Define LSTM and CNN Model.

The full code listing with a CNN and LSTM layers is listed below for completeness.

```
# LSTM and CNN 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.convolutional import Convolution1D
from keras.layers.convolutional import MaxPooling1D
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
```