**Sequence Classification with LSTM Recurrent Neural Networks in Python with Keras**

<http://machinelearningmastery.com/sequence-classification-lstm-recurrent-neural-networks-python-keras/>

by [**Jason Brownlee**](http://machinelearningmastery.com/author/jasonb/) on July 26, 2016 in [**Long Short-Term Memory Networks**](http://machinelearningmastery.com/category/lstm/)

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 post, 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 reading this post 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.

* **Update Oct/2016**: Updated examples for Keras 1.1.0 andTensorFlow 0.10.0.
* **Update Mar/2017**: Updated example for Keras 2.0.2, TensorFlow 1.0.1 and Theano 0.9.0.



Sequence Classification with LSTM Recurrent Neural Networks in Python with Keras  
Photo by [photophilde](https://www.flickr.com/photos/photophilde/4875287879/), some rights reserved.

## Problem Description

The problem that we will use to demonstrate sequence learning in this tutorial is the [IMDB movie review sentiment classification problem](http://ai.stanford.edu/~amaas/data/sentiment/). 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](http://ai.stanford.edu/~amaas/papers/wvSent_acl2011.pdf) 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.



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | 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.



|  |  |
| --- | --- |
| 1  2  3 | # 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.



|  |  |
| --- | --- |
| 1  2  3  4 | # 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.



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | # 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.



|  |  |
| --- | --- |
| 1  2  3 | # Final evaluation of the model  scores = model.evaluate(X\_test, y\_test, verbose=0)  print("Accuracy: %.2f%%" % (scores[1]\*100)) |

For completeness, here is the full code listing for this LSTM network on the IMDB dataset.



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29 | # 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)) |

Running this example produces the following output.

Note, if you are using a TensorFlow backend, you may see some warning messages related to “PoolAllocator”, that you can ignore for now.



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | 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.

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

## 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. For example:



|  |  |
| --- | --- |
| 1  2  3  4  5  6 | model = Sequential()  model.add(Embedding(top\_words, embedding\_vecor\_length, input\_length=max\_review\_length))  model.add(Dropout(0.2))  model.add(LSTM(100))  model.add(Dropout(0.2))  model.add(Dense(1, activation='sigmoid')) |

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



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32 | # 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(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(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, 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)) |

Running this example provides the following output.



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | 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% |

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** for configuring the input dropout and **recurrent\_dropout** for configuring the recurrent dropout. For example, we can modify the first example to add dropout to the input and recurrent connections as follows:



|  |  |
| --- | --- |
| 1  2  3  4 | model = Sequential()  model.add(Embedding(top\_words, embedding\_vecor\_length, input\_length=max\_review\_length))  model.add(LSTM(100, dropout=0.2, recurrent\_dropout=0.2))  model.add(Dense(1, activation='sigmoid')) |

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



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29 | # 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(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, dropout=0.2, recurrent\_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, 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)) |

Running this example provides the following output.



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | 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% |

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.

## LSTM and Convolutional Neural Network 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:



|  |  |
| --- | --- |
| 1  2  3  4  5  6 | model = Sequential()  model.add(Embedding(top\_words, embedding\_vecor\_length, input\_length=max\_review\_length))  model.add(Conv1D(filters=32, kernel\_size=3, padding='same', activation='relu'))  model.add(MaxPooling1D(pool\_size=2))  model.add(LSTM(100))  model.add(Dense(1, activation='sigmoid')) |

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



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33 | # 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 Conv1D  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  (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(Conv1D(filters=32, kernel\_size=3, padding='same', activation='relu'))  model.add(MaxPooling1D(pool\_size=2))  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)) |

Running this example provides the following output.



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | Epoch 1/3  16750/16750 [==============================] - 58s - loss: 0.5186 - acc: 0.7263  Epoch 2/3  16750/16750 [==============================] - 58s - loss: 0.2946 - acc: 0.8825  Epoch 3/3  16750/16750 [==============================] - 58s - loss: 0.2291 - acc: 0.9126  Accuracy: 86.36% |

We can see that we achieve similar results to the first example although with less weights and faster training time.

I would expect that even better results could be achieved if this example was further extended to use dropout.

## Resources

Below are some resources if you are interested in diving deeper into sequence prediction or this specific example.

* [Theano tutorial for LSTMs applied to the IMDB dataset](http://deeplearning.net/tutorial/lstm.html)
* Keras code example for using an [LSTM and CNN](https://github.com/fchollet/keras/blob/master/examples/imdb_lstm.py) with [LSTM on the IMDB dataset](https://github.com/fchollet/keras/blob/master/examples/imdb_cnn_lstm.py).
* [Supervised Sequence Labelling with Recurrent Neural Networks](http://www.amazon.com/dp/3642247962?tag=inspiredalgor-20), 2012 book by Alex Graves ([and PDF preprint](https://www.cs.toronto.edu/~graves/preprint.pdf)).

## Summary

In this post you discovered how to develop LSTM network models for sequence classification predictive modeling problems.

Specifically, you learned:

* How to develop a simple single layer LSTM model for the IMDB movie review sentiment classification problem.
* How to extend your LSTM model with layer-wise and LSTM-specific dropout to reduce overfitting.
* How to combine the spatial structure learning properties of a Convolutional Neural Network with the sequence learning of an LSTM.

Do you have any questions about sequence classification with LSTMs or about this post? Ask your questions in the comments and I will do my best to answer.