How to Prepare Text Data for Deep Learning with Keras

by **Jason Brownlee** on October 2, 2017 in **Natural Language Processing**

You cannot feed raw text directly into deep learning models.

Text data must be encoded as numbers to be used as input or output for machine learning and deep learning models.

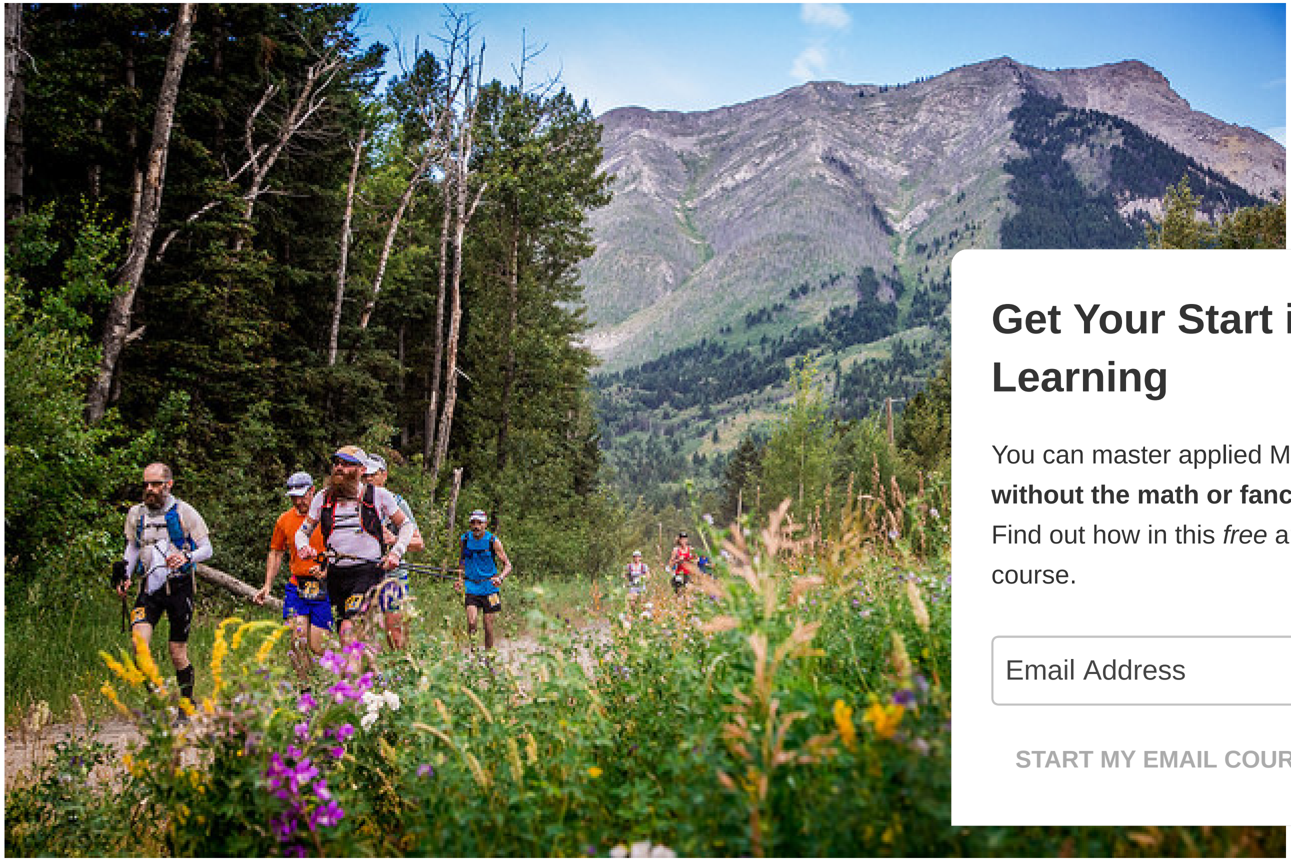
The Keras deep learning library provides some basic tools to help you prepare your text data.

In this tutorial, you will discover how you can use Keras to prepare your text data.

After completing this tutorial, you will know:

- About the convenience methods that you can use to quickly prepare text data.
- The Tokenizer API that can be fit on training data and used to encode training, validation, and test documents.
- The range of 4 different document encoding schemes offered by the Tokenizer API.

Let’s get started.



How to Prepare Text Data for Deep Learning with Keras
Photo by ActiveSteve, some rights reserved.

Tutorial Overview

This tutorial is divided into 4 parts; they are:

[redacted]

1. Split words with text_to_word_sequence.
2. Encoding with one_hot.
3. Hash Encoding with hashing_trick.
4. Tokenizer API

Split Words with text_to_word_sequence

A good first step when working with text is to split it into words.

Words are called tokens and the process of splitting text into tokens is called tokenization.

Keras provides the text_to_word_sequence() function that you can use to split text into a list of words.

By default, this function automatically does 3 things:

- Splits words by space (split=” “).
- Filters out punctuation (filters=’!”#$%&()*+,-./:;<=>?@[\\]^_`{|}~\t\n’).
- Converts text to lowercase (lower=True).

You can change any of these defaults by passing arguments to the function.

Below is an example of using the text_to_word_sequence() function to split a document (in this case

```
from keras.preprocessing.text import text_to_word_sequence
# define the document
text = 'The quick brown fox jumped over the lazy dog.'
# tokenize the document
result = text_to_word_sequence(text)
print(result)
```

Running the example creates an array containing all of the words in the document. The list of words is printed for review.

```
['the', 'quick', 'brown', 'fox', 'jumped', 'over', 'the', 'lazy', 'dog']
```

This is a good first step, but further pre-processing is required before you can work with the text.

Encoding with one_hot

It is popular to represent a document as a sequence of integer values, where each word in the document is represented as a unique integer.

Keras provides the one_hot() function that you can use to tokenize and integer encode a text document in one step. The name suggests that it will create a one-hot encoding of the document, which is not the case. Instead, the function is a wrapper for the hashing_trick() function described in the next section. The function returns an integer encoded version of the document. The use of a hash function means that there may be collisions and not all words will be assigned unique integer values.

Instead, the function is a wrapper for the hashing_trick() function described in the next section. The function returns an integer encoded version of the document. The use of a hash function means that there may be collisions and not all words will be assigned unique integer values.

As with the text_to_word_sequence() function in the previous section, the one_hot() function will ma split words based on white space.

In addition to the text, the vocabulary size (total words) must be specified. This could be the total nu nd to encode additional documents that contains additional words. The size of the vocabulary defines th Ideally, this should be larger than the vocabulary by some percentage (perhaps 25%) to minimize the on is used, although as we will see in the next section, alternate hash functions can be specified when c

We can use the text_to_word_sequence() function from the previous section to split the document in unique words in the document. The size of this set can be used to estimate the size of the vocabular

For example:

```
from keras.preprocessing.text import text_to_word_sequence
# define the document
text = 'The quick brown fox jumped over the lazy dog.'
# estimate the size of the vocabulary
words = set(text_to_word_sequence(text))
vocab_size = len(words)
print(vocab_size)
```

We can put this together with the one_hot() function and one hot encode the words in the document. The complete example is listed below.

The vocabulary size is increased by one-third to minimize collisions when hashing words.

```
from keras.preprocessing.text import one_hot
from keras.preprocessing.text import text_to_word_sequence
# define the document
text = 'The quick brown fox jumped over the lazy dog.'
# estimate the size of the vocabulary
words = set(text_to_word_sequence(text))
vocab_size = len(words)
print(vocab_size)
# integer encode the document
result = one_hot(text, round(vocab_size*1.3))
print(result)
```

Running the example first prints the size of the vocabulary as 8. The encoded document is then printed as an array of integer encoded words.

```
8
[5, 9, 8, 7, 9, 1, 5, 3, 8]
```

Hash Encoding with hashing_trick

A limitation of integer and count base encodings is that they must maintain a vocabulary of words an

An alternative to this approach is to use a one-way hash function to convert words to integers. This a ch is faster and requires less memory.

Keras provides the hashing_trick() function that tokenizes and then integer encodes the document, j flexibility, allowing you to specify the hash function as either 'hash' (the default) or other hash functio function.

Below is an example of integer encoding a document using the md5 hash function.

```
from keras.preprocessing.text import hashing_trick
from keras.preprocessing.text import text_to_word_sequence
# define the document
text = 'The quick brown fox jumped over the lazy dog.'
# estimate the size of the vocabulary
words = set(text_to_word_sequence(text))
vocab_size = len(words)
print(vocab_size)
# integer encode the document
result = hashing_trick(text, round(vocab_size*1.3), hash_function='md5')
print(result)
```

[redacted]

Running the example prints the size of the vocabulary and the integer encoded document.

We can see that the use of a different hash function results in consistent, but different integers for words as the one_hot() function in the previous section.

```
8
[6, 4, 1, 2, 7, 5, 6, 2, 6]
```

Tokenizer API

So far we have looked at one-off convenience methods for preparing text with Keras.

Keras provides a more sophisticated API for preparing text that can be fit and reused to prepare multiple text documents. This may be the preferred approach for large projects.

Keras provides the Tokenizer class for preparing text documents for deep learning. The Tokenizer m documents or integer encoded text documents.

For example:

```
from keras.preprocessing.text import Tokenizer
# define 5 documents
docs = ['Well done!',
        'Good work',
        'Great effort',
        'nice work',
        'Excellent!']
# create the tokenizer
t = Tokenizer()
# fit the tokenizer on the documents
t.fit_on_texts(docs)
```

Once fit, the Tokenizer provides 4 attributes that you can use to query what has been learned about your documents:

- **word_counts**: A dictionary of words and their counts.
- **word_docs**: An integer count of the total number of documents that were used to fit the Tokenizer.
- **word_index**: A dictionary of words and their uniquely assigned integers.
- **document_count**: A dictionary of words and how many documents each appeared in.

For example:

```
# summarize what was learned
print(t.word_counts)
print(t.document_count)
print(t.word_index)
print(t.word_docs)
```

Once the Tokenizer has been fit on training data, it can be used to encode documents in the train or test datasets.

The texts_to_matrix() function on the Tokenizer can be used to create one vector per document provided per input. The length of the vectors is the total size of the vocabulary.

This function provides a suite of standard bag-of-words model text encoding schemes that can be provided via a mode argument to the function.

The modes available include:

- ‘*binary*‘: Whether or not each word is present in the document. This is the default.
- ‘*count*‘: The count of each word in the document.
- ‘*tfidf*‘: The Text Frequency-Inverse DocumentFrequency (TF-IDF) scoring for each word in the d
- ‘*freq*‘: The frequency of each word as a ratio of words within each document.

We can put all of this together with a worked example.

```
from keras.preprocessing.text import Tokenizer
# define 5 documents
docs = ['Well done!',
        'Good work',
        'Great effort',
        'nice work',
        'Excellent!']
# create the tokenizer
t = Tokenizer()
# fit the tokenizer on the documents
t.fit_on_texts(docs)
# summarize what was learned
print(t.word_counts)
print(t.document_count)
print(t.word_index)
print(t.word_docs)
# integer encode documents
```

```
encoded_docs = t.texts_to_matrix(docs, mode='count')
print(encoded_docs)
```

[redacted]

Running the example fits the Tokenizer with 5 small documents. The details of the fit Tokenizer are printed. Then the 5 documents are encoded using a word count.

Each document is encoded as a 9-element vector with one position for each word and the chosen encoding scheme value for each word position. In this case, a simple word count mode is used.

```
OrderedDict([('well', 1), ('done', 1), ('good', 1), ('work', 2), ('great', 1), ('effort', 1), ('nice', 1), ('excellent', 1)])
5
{'work': 1, 'effort': 6, 'done': 3, 'great': 5, 'good': 4, 'excellent': 8, 'well': 2, 'nice': 7}
{'work': 2, 'effort': 1, 'done': 1, 'well': 1, 'good': 1, 'great': 1, 'excellent': 1, 'nice': 1}
[[ 0.  0.  1.  1.  0.  0.  0.  0.  0.]
 [ 0.  1.  0.  0.  1.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  1.  1.  0.  0.]
 [ 0.  1.  0.  0.  0.  0.  0.  1.  0.]
 [ 0.  0.  0.  0.  0.  0.  0.  0.  1.]]
```

Further Reading

This section provides more resources on the topic if you are looking go deeper.

- Text Preprocessing Keras API
- text_to_word_sequence Keras API
- one_hot Keras API
- hashing_trick Keras API
- Tokenizer Keras API

Summary

In this tutorial, you discovered how you can use the Keras API to prepare your text data for deep learning.

Specifically, you learned:

- About the convenience methods that you can use to quickly prepare text data.
- The Tokenizer API that can be fit on training data and used to encode training, validation, and te

- The range of 4 different document encoding schemes offered by the Tokenizer API.

Do you have any questions?
Ask your questions in the comments below and I will do my best to answer.

About Jason Brownlee

Dr. Jason Brownlee is a husband, proud father, academic researcher, author, professional developer and a machine learning practitioner. He is dedicated to helping developers get started and get good at applied machine learning. Learn more.
View all posts by Jason Brownlee →

‹ How to Prepare Text Data for Machine Learning with scikit-learn

How ras ›

6 Responses to *How to Prepare Text Data for Deep Learning with Keras*

Chiedu October 2, 2017 at 6:40 am #

Hi Jason,
Do you have any plans to cover word embeddings usine either word2vec or GloVe and how they work with keras?

Jason Brownlee October 2, 2017 at 9:40 am #

REPLY

Yes! I have many posts on word embeddings scheduled for the coming days/weeks.

[redacted]

Lalit Parihar October 6, 2017 at 6:59 pm #

REPLY

Hello Jason,
It seems the attributes mentioned for Tokenizer have been typed incorrectly, document_count and word_docs have been inter-changed.

Thanks,
Lalit

Jason Brownlee October 7, 2017 at 5:52 am #

REPLY

Thanks Lalit, in which part of the tutorial exactly?

Gopika Bhardwaj October 16, 2017 at 3:07 am #

How do we further apply a neural network on this data?

Jason Brownlee October 16, 2017 at 5:45 am #

Great question, I will have many tutorials about how to do that coming out on the blog in co

Leave a Reply

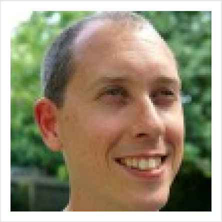
Name (required)

Email (will not be published) (required)

Website

SUBMIT COMMENT

Welcome to Machine Learning Mastery



Hi, I'm Dr. Jason Brownlee.
My goal is to make practitioners like YOU awesome at applied machine learning.

Read More

You're a Professional (and you need results)!

The field moves quickly…
How Long Can You Afford To Wait?

POPULAR

Time Series Prediction with LSTM Recurrent Neural Networks in Python with Keras
JULY 21, 2016

Your First Machine Learning Project in Python Step-By-Step
JUNE 10, 2016

Develop Your First Neural Network in Python With Keras Step-By-Step
MAY 24, 2016

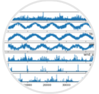
Sequence Classification with LSTM Recurrent Neural Networks in Python with Keras
JULY 26, 2016

How to Setup a Python Environment for Machine Learning and Deep Learning with Anaconda
MARCH 13, 2017

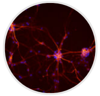
Time Series Forecasting with the Long Short-Term Memory Network in Python
APRIL 7, 2017

Multi-Class Classification Tutorial with the Keras Deep Learning Library
JUNE 2, 2016

Regression Tutorial with the Keras Deep Learning Library in Python
JUNE 9, 2016



Multivariate Time Series Forecasting with LSTMs in Keras
AUGUST 14, 2017



How to Implement the Backpropagation Algorithm From Scratch In Python
NOVEMBER 7, 2016

Privacy | Contact | About