Deep Learning for Natural Language Processing

7-Day Crash-Course

Jason Brownlee

MACHINE LEARNING MASTERY

Disclaimer

Deep Learning for Natural Language Processing Crash Course

Edition: v1.8

Find the latest version of this guide online at: http://MachineLearningMastery.com

Contents

Before We Get Started...

We are awash with text, from books, papers, blogs, tweets, news, and increasingly text from spoken utterances. Working with text is hard as it requires drawing upon knowledge from diverse domains such as linguistics, machine learning, statistical methods, and these days, deep learning. Deep learning methods are starting to out-compete the classical and statistical methods on some challenging natural language processing problems with singular and simpler models. In this crash course, you will discover how you can get started and confidently develop deep learning for natural language processing problems using Python in 7 days. Let's get started.

Who Is This Crash-Course For?

Before we get started, let's make sure you are in the right place. The list below provides some general guidelines as to who this course was designed for. Don't panic if you don't match these points exactly, you might just need to brush up in one area or another to keep up.

You need to know:

- You need to know your way around basic Python, NumPy and Keras for deep learning.

You do NOT need to know:

- You do not need to be a math wiz!
- You do not need to be a deep learning expert!
- You do not need to be a linguist!

This crash course will take you from a developer that knows a little machine learning to a developer who can bring deep learning methods to your own natural language processing project. This crash course assumes you have a working Python 2 or 3 SciPy environment with at least NumPy, Pandas, scikit-learn and Keras 2 installed. If you need help with your environment, you can follow the step-by-step tutorial here:

- How to Setup a Python Environment for Machine Learning and Deep Learning. https://goo.gl/QwffqZ

Crash-Course Overview

This crash course is broken down into 7 lessons. You could complete one lesson per day (recommended) or complete all of the lessons in one day (hardcore). It really depends on the time you have available and your level of enthusiasm. Below are 7 lessons that will get you started and productive with deep learning for natural language processing in Python:

- **Lesson 01**: Deep Learning and Natural Language.
- **Lesson 02**: Cleaning Text Data.
- **Lesson 03**: Bag-of-Words Model.
- **Lesson 04**: Word Embedding Representation.
- **Lesson 05**: Learned Embedding.
- **Lesson 06**: Classifying Text.
- **Lesson 07**: Movie Review Sentiment Analysis.

Each lesson could take you 60 seconds or up to 30 minutes. Take your time and complete the lessons at your own pace. Ask questions and even put results online and share your results. The lessons expect you to go off and find out how to do things. I will give you hints, but part of the point of each lesson is to force you to learn where to go to look for help on and about the deep learning, natural language processing and the best-of-breed tools in Python (hint, I have all of the answers directly on this blog, use the search box). I do provide more help in the form of links to related material because I want you to build up some confidence and inertia. Post your results online, I'll cheer you on!

Hang in there, don't give up!

Lesson 01: Deep Learning and Natural Language

In this lesson, you will discover a concise definition for natural language, deep learning and the promise of deep learning for working with text data.

Natural Language Processing

Natural Language Processing, or NLP for short, is broadly defined as the automatic manipulation of natural language, like speech and text, by software. The study of natural language processing has been around for more than 50 years and grew out of the field of linguistics with the rise of computers. The problem of understanding text is not solved, and may never be, is primarily because language is messy. There are few rules. And yet we can easily understand each other most of the time.

Deep Learning

Deep Learning is a subfield of machine learning concerned with algorithms inspired by the structure and function of the brain called artificial neural networks. A property of deep learning is that the performance of these type of model improves by training them with more examples by increasing their depth or representational capacity. In addition to scalability, another often cited benefit of deep learning models is their ability to perform automatic feature extraction from raw data, also called feature learning.

Promise of Deep Learning for NLP

Deep learning methods are popular for natural language, primarily because they are delivering on their promise.

Some of the first large demonstrations of the power of deep learning were in natural language processing, specifically speech recognition. More recently in machine translation.

The 3 key promises of deep learning for natural language processing are as follows:

- **The Promise of Feature Learning**. That is, that deep learning methods can learn the features from natural language required by the model, rather than requiring that the features be specified and extracted by an expert.

- **The Promise of Continued Improvement**. That is, that the performance of deep learning in natural language processing is based on real results and that the improvements appear to be continuing and perhaps speeding up.
- **The Promise of End-to-End Models**. That is, that large end-to-end deep learning models can be fit on natural language problems offering a more general and better-performing approach.

Natural language processing is not *solved*, but deep learning is required to get you to the state-of-the-art on many challenging problems in the field.

Your Task

For this lesson you must research and list 10 impressive applications of deep learning methods in the field of natural language processing. Bonus points if you can link to a research paper that demonstrates the example. Post your answer online. I would love to see what you discover.

More Information

- What Is Natural Language Processing?
  https://goo.gl/ufQxqj
- What is Deep Learning?
  https://goo.gl/efPgLn
- Promise of Deep Learning for Natural Language Processing
  https://goo.gl/bULeuz
- 7 Applications of Deep Learning for Natural Language Processing
  https://goo.gl/dthkdu

In the next lesson, you will discover how to clean text data so that it is ready for modeling.

Lesson 02: Cleaning Text Data

In this lesson, you will discover how you can load and clean text data so that it is ready for modeling using both manually and with the NLTK Python library.

Text is Messy

You cannot go straight from raw text to fitting a machine learning or deep learning model. You must clean your text first, which means splitting it into words and normalizing issues such as:

- Upper and lower case characters.
- Punctuation within and around words.
- Numbers such as amounts and dates.
- Spelling mistakes and regional variations.
- Unicode characters
- and much more...

Manual Tokenization

Generally, we refer to the process of turning raw text into something we can model as *tokenization*, where we are left with a list of words or *tokens*. We can manually develop Python code to clean text, and often this is a good approach given that each text dataset must be tokenized in a unique way. For example, the snippet of code below will load a text file, split tokens by whitespace and convert each token to lowercase.

```
filename = '...'
file = open(filename, 'rt')
text = file.read()
file.close()
# split into words by white space
words = text.split()
# convert to lowercase
words = [word.lower() for word in words]
```

Listing 1: Example of manual tokenization.

You can imagine how this snippet could be extended to handle and normalize Unicode characters, remove punctuation and so on.

NLTK Tokenization

Many of the best practices for tokenizing raw text have been captured and made available in a Python library called the Natural Language Toolkit or NLTK for short. You can install this library using `pip` by typing the following on the command line:

```
sudo pip install -U nltk
```

Listing 2: Install the NLTK library.

After it is installed, you must also install the datasets used by the library, either via a Python script:

```
import nltk
nltk.download()
```

Listing 3: Python script to install NLTK data.

Or via a command line:

```
python -m nltk.downloader all
```

Listing 4: Command line to install NLTK data.

Once installed, you can use the API to tokenize text. For example, the snippet below will load and tokenize an ASCII text file.

```
# load data
filename = '...'
file = open(filename, 'rt')
text = file.read()
file.close()
# split into words
from nltk.tokenize import word_tokenize
tokens = word_tokenize(text)
```

Listing 5: Example of tokenization with NLTK.

There are many tools available in this library and you can further refine the clean tokens using your own manual methods, such as removing punctuation, removing stop words, stemming and much more.

Your Task

Your task is to locate a free classical book on the Project Gutenberg website, download the ASCII version of the book and tokenize the text and save the result to a new file. Bonus points for exploring both manual and NLTK approaches. Post your code online. I would love to see what book you choose and how you chose to tokenize it.

More Information

- Project Gutenberg.
  http://www.gutenberg.org/

- nltk.tokenize package API.
  http://www.nltk.org/api/nltk.tokenize.html
- How to Clean Text for Machine Learning with Python.
  https://machinelearningmastery.com/clean-text-machine-learning-python/

In the next lesson, you will discover the bag-of-words model.

Lesson 03: Bag-of-Words Model

In this lesson, you will discover the bag-of-words model and how to encode text using this model so that you can train a model using the scikit-learn and Keras Python libraries.

Bag-of-Words

The bag-of-words model is a way of representing text data when modeling text with machine learning algorithms. The approach is very simple and flexible, and can be used in a myriad of ways for extracting features from documents. A bag-of-words is a representation of text that describes the occurrence of words within a document.

A vocabulary is chosen, where perhaps some infrequently used words are discarded. A given document of text is then represented using a vector with one position for each word in the vocabulary and a score for each known word that appears (or not) in the document. It is called a *bag* of words, because any information about the order or structure of words in the document is discarded. The model is only concerned with whether known words occur in the document, not where in the document.

Bag-of-Words with scikit-learn

The scikit-learn Python library for machine learning provides tools for encoding documents for a bag-of-words model. An instance of the encoder can be created, trained on a corpus of text documents and then used again and again to encode training, test, validation and any new data that needs to be encoded for your model.

There is an encoder to score words based on their count called `CountVectorizer`, one for using a hash function of each word to reduce the vector length called `HashingVectorizer`, and a one that uses a score based on word occurrence in the document and the inverse occurrence across all documents called `TfidfVectorizer`. The snippet below shows how to train the `TfidfVectorizer` bag-of-words encoder and use it to encode multiple small text documents.

```
from sklearn.feature_extraction.text import TfidfVectorizer
# list of text documents
text = ["The quick brown fox jumped over the lazy dog.",
    "The dog.",
    "The fox"]
# create the transform
vectorizer = TfidfVectorizer()
# tokenize and build vocab
vectorizer.fit(text)
# summarize
```

```
print(vectorizer.vocabulary_)
print(vectorizer.idf_)
# encode document
vector = vectorizer.transform([text[0]])
# summarize encoded vector
print(vector.shape)
print(vector.toarray())
```

Listing 6: Encode text as a bag-of-words using scikit-learn.

Bag-of-Words with Keras

The Keras Python library for deep learning also provides tools for encoding text using the bag-of-words model in the `Tokenizer` class. As above, the encoder must be trained on source documents and then can be used to encode training data, test data and any other data in the future. The API also has the benefit of performing basic tokenization prior to encoding the words. The snippet below demonstrates how to train and encode some small text documents using the Keras API and the `count` type scoring of words.

```
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)
```

Listing 7: Encode text as a bag-of-words using Keras.

Your Task

Your task in this lesson is to experiment with the scikit-learn and Keras methods for encoding small contrived text documents for the bag-of-words model. Bonus points if you use a small standard text dataset of documents to practice on and perform data cleaning as part of the preparation. Post your code online. I would love to see what APIs you explore and demonstrate.

More Information

- A Gentle Introduction to the Bag-of-Words Model.
  https://goo.gl/s9a14Q
- How to Prepare Text Data for Machine Learning with scikit-learn.
  https://goo.gl/LeCjPW
- How to Prepare Text Data for Deep Learning with Keras.
  https://goo.gl/RjR7Wz

In the next lesson, you will discover word embeddings.

Lesson 04: Word Embedding Representation

In this lesson, you will discover the word embedding distributed representation and how to develop a word embedding using the Gensim Python library.

Word Embeddings

Word embeddings are a type of word representation that allows words with similar meaning to have a similar representation. They are a distributed representation for text that is perhaps one of the key breakthroughs for the impressive performance of deep learning methods on challenging natural language processing problems. Word embedding methods learn a real-valued vector representation for a predefined fixed sized vocabulary from a corpus of text.

Train Word Embeddings

You can train a word embedding distributed representation using the Gensim Python library for topic modeling. Gensim offers an implementation of the Word2Vec algorithm, developed at Google for the fast training of word embedding representations from text documents, You can install Gensim using `pip` by typing the following on your command line:

```
pip install -U gensim
```

Listing 8: Install the Gensim library.

The snippet below shows how to define a few contrived sentences and train a word embedding representation in Gensim.

```
from gensim.models import Word2Vec
# define training data
sentences = [['this', 'is', 'the', 'first', 'sentence', 'for', 'word2vec'],
     ['this', 'is', 'the', 'second', 'sentence'],
     ['yet', 'another', 'sentence'],
     ['one', 'more', 'sentence'],
     ['and', 'the', 'final', 'sentence']]
# train model
model = Word2Vec(sentences, min_count=1)
# summarize the loaded model
print(model)
# summarize vocabulary
words = list(model.wv.vocab)
```

```
print(words)
# access vector for one word
print(model['sentence'])
```

Listing 9: Example of fitting a word embedding model.

Use Embeddings

Once trained, the embedding can be saved to file to be used as part of another model, such as the front-end of a deep learning model. You can also plot a projection of the distributed representation of words to get an idea of how the model believes words are related. A common projection technique that you can use is the Principal Component Analysis or PCA, available in scikit-learn. The snippet below shows how to train a word embedding model and then plot a two-dimensional projection of all words in the vocabulary.

```
from gensim.models import Word2Vec
from sklearn.decomposition import PCA
from matplotlib import pyplot
# define training data
sentences = [['this', 'is', 'the', 'first', 'sentence', 'for', 'word2vec'],
     ['this', 'is', 'the', 'second', 'sentence'],
     ['yet', 'another', 'sentence'],
     ['one', 'more', 'sentence'],
     ['and', 'the', 'final', 'sentence']]
# train model
model = Word2Vec(sentences, min_count=1)
# fit a 2D PCA model to the vectors
X = model[model.wv.vocab]
pca = PCA(n_components=2)
result = pca.fit_transform(X)
# create a scatter plot of the projection
pyplot.scatter(result[:, 0], result[:, 1])
words = list(model.wv.vocab)
for i, word in enumerate(words):
  pyplot.annotate(word, xy=(result[i, 0], result[i, 1]))
pyplot.show()
```

Listing 10: Example of fitting and plotting a word embedding model.

Your Task

Your task in this lesson is to train a word embedding using Gensim on a text document, such as a book from Project Gutenberg. Bonus points if you can generate a plot of common words. Post your code online. I would love to see what book you choose and any details of the embedding that you learn.

More Information

- What Are Word Embeddings for Text?
  https://machinelearningmastery.com/what-are-word-embeddings/

- How to Develop Word Embeddings in Python with Gensim.
  https://machinelearningmastery.com/develop-word-embeddings-python-gensim/
- Project Gutenberg.
  http://www.gutenberg.org/

In the next lesson, you will discover how a word embedding can be learned as part of a deep learning model.

Lesson 05: Learned Embedding

In this lesson, you will discover how to learn a word embedding distributed representation for words as part of fitting a deep learning model

Embedding Layer

Keras offers an `Embedding` layer that can be used for neural networks on text data. It requires that the input data be integer encoded so that each word is represented by a unique integer. This data preparation step can be performed using the `Tokenizer` API also provided with Keras. The `Embedding` layer is initialized with random weights and will learn an embedding for all of the words in the training dataset. You must specify the `input_dim` which is the size of the vocabulary, the `output_dim` which is the size of the vector space of the embedding, and optionally the `input_length` which is the number of words in input sequences.

```
layer = Embedding(input_dim, output_dim, input_length=??)
```

Listing 11: Example of defining an `Embedding` layer.

Or, more concretely, a vocabulary of 200 words, a distributed representation of 32 dimensions and an input length of 50 words.

```
layer = Embedding(200, 32, input_length=50)
```

Listing 12: Concrete example of defining an `Embedding` layer.

Embedding with Model

The `Embedding` layer can be used as the front-end of a deep learning model to provide a rich distributed representation of words, and importantly this representation can be learned as part of training the deep learning model. For example, the snippet below will define and compile and neural network with an embedding input layer and a dense output layer for a document classification problem. When the model is trained on examples of padded documents and their associated output label both the network weights and the distributed representation will be tuned to the specific data.

```
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Flatten
from keras.layers.embeddings import Embedding
# define problem
vocab_size = 100
```

```
max_length = 32
# define the model
model = Sequential()
model.add(Embedding(vocab_size, 8, input_length=max_length))
model.add(Flatten())
model.add(Dense(1, activation='sigmoid'))
# compile the model
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
# summarize the model
model.summary()
```

Listing 13: Example of neural network with word embedding input.

It is also possible to initialize the `Embedding` layer with pre-trained weights, such as those prepared by Gensim and to configure the layer to not be trainable. This approach can be useful if a very large corpus of text is available to pre-train the word embedding.

Your Task

Your task in this lesson is to design a small document classification problem with 10 documents of one sentence each and associated labels of positive and negative outcomes and to train a network with word embedding on these data. Note that each sentence will need to be padded to the same maximum length prior to training the model using the Keras `pad_sequences()` function. Bonus points if you load a pre-trained word embedding prepared using Gensim. Post your code online. I would love to see what sentences you contrive and the skill of your model.

More Information

- Data Preparation for Variable Length Input Sequences.
  https://goo.gl/Xw2Ndw
- How to Use Word Embedding Layers for Deep Learning with Keras.
  https://goo.gl/qCGtCa

In the next lesson, you will discover how to develop deep learning models for classifying text.

Lesson 06: Classifying Text

In this lesson, you will discover the standard deep learning model for classifying text used on problems such as sentiment analysis of text.

Document Classification

Text classification describes a general class of problems such as predicting the sentiment of tweets and movie reviews, as well as classifying email as spam or not. It is an important area of natural language processing and a great place to get started using deep learning techniques on text data. Deep learning methods are proving very good at text classification, achieving state-of-the-art results on a suite of standard academic benchmark problems.

Embeddings + CNN

The modus operandi for text classification involves the use of a word embedding for representing words and a Convolutional Neural Network or CNN for learning how to discriminate documents on classification problems. The architecture is comprised of three key pieces:

- **Word Embedding Model**: A distributed representation of words where different words that have a similar meaning (based on their usage) also have a similar representation.
- **Convolutional Model**: A feature extraction model that learns to extract salient features from documents represented using a word embedding.
- **Fully-Connected Model**: The interpretation of extracted features in terms of a predictive output.

This type of model can be defined in the Keras Python deep learning library. The snippet below shows an example of a deep learning model for classifying text documents as one of two classes.

```
# define problem
vocab_size = 100
max_length = 200
# define model
model = Sequential()
model.add(Embedding(vocab_size, 100, input_length=max_length))
model.add(Conv1D(32, 8, activation='relu'))
model.add(MaxPooling1D(2))
model.add(Flatten())
```

```
model.add(Dense(10, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
model.summary()
```

Listing 14: Example of an Embedding + CNN model.

Your Task

Your task in this lesson is to research the use of the Embeddings + CNN combination of deep learning methods for text classification and report on examples or best practices for configuring this model, such as the number of layers, kernel size, vocabulary size and so on. Bonus points if you can find and describe the variation that supports n-gram or multiple groups of words as input by varying the kernel size. Post your findings online. I would love to see what you discover.

More Information

- Best Practices for Document Classification with Deep Learning.
  https://goo.gl/RSCK5h

In the next lesson, you will discover how to work through a sentiment analysis prediction problem.

Lesson 07: Movie Review Sentiment Analysis

In this lesson, you will discover how to prepare text data, develop and evaluate a deep learning model to predict the sentiment of movie reviews.

I want you to tie together everything you have learned in this crash course and work through a real-world problem end-to-end.

Movie Review Dataset

The Movie Review Dataset is a collection of movie reviews retrieved from the imdb.com website in the early 2000s by Bo Pang and Lillian Lee. The reviews were collected and made available as part of their research on natural language processing. You can download the dataset from here:

- Movie Review Polarity Dataset (`review_polarity.tar.gz`, 3MB).
  https://raw.githubusercontent.com/jbrownlee/Datasets/master/review_polarity.tar.gz

From this dataset you will develop a sentiment analysis deep learning model to predict whether a given movie review is positive or negative.

Your Task

Your task in this lesson is to develop and evaluate a deep learning model on the movie review dataset:

1. Download and inspect the dataset.
2. Clean and tokenize the text and save the results to a new file.
3. Split the clean data into train and test datasets.
4. Develop an Embedding + CNN model on the training dataset.
5. Evaluate the model on the test dataset.

Bonus points if you can demonstrate your model by making a prediction on a new movie review, contrived or real. Extra bonus points if you can compare your model to a neural bag-of-words model. Post your code and model skill online. I would love to see what you can come up with. Simpler models are preferred, but also try going really deep and see what happens.

More Information

- How to Prepare Movie Review Data for Sentiment Analysis.
  https://goo.gl/oyUe7D
- How to Develop a Deep Learning Bag-of-Words Model for Predicting Movie Review Sentiment.
  https://goo.gl/9D5JES
- How to Develop a Word Embedding Model for Predicting Movie Review Sentiment.
  https://goo.gl/HDjqrH

Final Word Before You Go...

You made it. Well done! Take a moment and look back at how far you have come. You discovered:

- What natural language processing is and the promise and impact that deep learning is having on the field.
- How to clean and tokenize raw text data manually and use NLTK to make it ready for modeling.
- How to encode text using the bag-of-words model with the scikit-learn and Keras libraries.
- How to train a word embedding distributed representation of words using the Gensim library.
- How to learn a word embedding distributed representation as a part of fitting a deep learning model.
- How to use word embeddings with convolutional neural networks for text classification problems.
- How to work through a real-world sentiment analysis problem end-to-end using deep learning methods.

Don't make light of this, you have come a long way in a short amount of time. This is just the beginning of your journey with deep learning for natural language processing. Keep practicing and developing your skills.

How Did You Go With The Crash-Course?

Did you enjoy this crash-course?
Do you have any questions or sticking points?

Let me know, send me an email at: **jason@MachineLearningMastery.com**

Take the Next Step

Looking for more help with Deep Learning for Natural Language Processing?

Grab my new book:
Deep Learning for Natural Language Processing
https://machinelearningmastery.com/deep-learning-for-nlp/