CHAPTER 1

What Is Natural Language Processing?

Deep learning and machine learning continues to proliferate throughout various industries, and has revolutionized the topic that I wish to discuss in this book: natural language processing (NLP). NLP is a subfield of computer science that is focused on allowing computers to understand language in a "natural" way, as humans do. Typically, this would refer to tasks such as understanding the sentiment of text, speech recognition, and generating responses to questions.

NLP has become a rapidly evolving field, and one whose applications have represented a large portion of artificial intelligence (AI) breakthroughs. Some examples of implementations using deep learning are chatbots that handle customer service requests, auto-spellcheck on cell phones, and AI assistants, such as Cortana and Siri, on smartphones. For those who have experience in machine learning and deep learning, natural language processing is one of the most exciting areas for individuals to apply their skills. To provide context for broader discussions, however, let's discuss the development of natural language processing as a field.

T. Beysolow II, *Applied Natural Language Processing with Python*,
https://doi.org/10.1007/978-1-4842-3733-5_1

The History of Natural Language Processing

Natural language processing can be classified as a subset of the broader field of speech and language processing. Because of this, NLP shares similarities with parallel disciplines such as computational linguistics, which is concerned with modeling language using rule-based models. NLP's inception can be traced back to the development of computer science in the 1940s, moving forward along with advances in linguistics that led to the construction of formal language theory. Briefly, formal language theory models language on increasingly complex structures and rules to these structures. For example, the alphabet is the simplest structure, in that it is a collection of letters that can form strings called *words*. A formal language is one that has a regular, context-free, and formal grammar. In addition to the development of computer sciences as a whole, artificial intelligence's advancements also played a role in our continuing understanding of NLP.

In some sense, the single-layer perceptron (SLP) is considered to be the inception of machine learning/AI. Figure 1-1 shows a photo of this model.

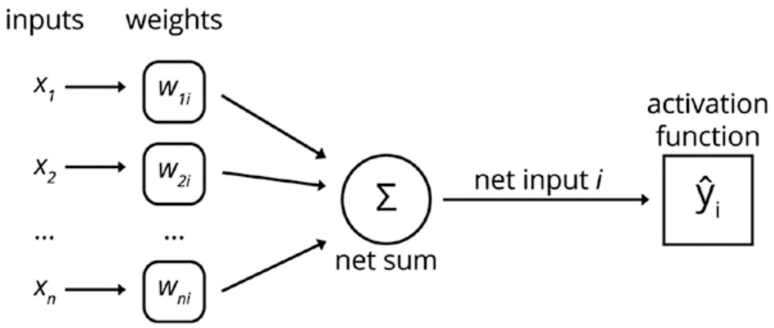

Figure 1-1. *Single-layer perceptron*

The SLP was designed by neurophysiologist Warren McCulloch and logician Walter Pitt. It is the foundation of more advanced neural network models that are heavily utilized today, such as multilayer perceptrons.

The SLP model is seen to be in part due to Alan Turing's research in the late 1930s on computation, which inspired other scientists and researchers to develop different concepts, such as formal language theory.

Moving forward to the second half of the twentieth century, NLP starts to bifurcate into two distinct groups of thought: (1) those who support a symbolic approach to language modelling, and (2) those who support a stochastic approach. The former group was populated largely by linguists who used simple algorithms to solve NLP problems, often utilizing pattern recognition. The latter group was primarily composed of statisticians and electrical engineers. Among the many approaches that were popular with the second group was Bayesian statistics. As the twentieth century progressed, NLP broadened as a field, including natural language understanding (NLU) to the problem space (allowing computers to react accurately to commands). For example, if someone spoke to a chatbot and asked it to "find food near me," the chatbot would use NLU to translate this sentence into tangible actions to yield a desirable outcome.

Skip closer to the present day, and we find that NLP has experienced a surge of interest alongside machine learning's explosion in usage over the past 20 years. Part of this is due to the fact that large repositories of labeled data sets have become more available, in addition to an increase in computing power. This increase in computing power is largely attributed to the development of GPUs; nonetheless, it has proven vital to AI's development as a field. Accordingly, demand for materials to instruct data scientists and engineers on how to utilize various AI algorithms has increased, in part the reason for this book.

Now that you are aware of the history of NLP as it relates to the present day, I will give a brief overview of what you should expect to learn. The focus, however, is primarily to discuss how deep learning has impacted NLP, and how to utilize deep learning and machine learning techniques to solve NLP problems.

A Review of Machine Learning and Deep Learning

You will be refreshed on important machine learning concepts, particularly deep learning models such as *multilayer perceptrons* (MLPs), *recurrent neural networks* (RNNs), and *long short-term memory* (LSTM) networks. You will be shown in-depth models utilizing toy examples before you tackle any specific NLP problems.

NLP, Machine Learning, and Deep Learning Packages with Python

Equally important as understanding NLP theory is the ability to apply it in a practical context. This book utilizes the Python programming language, as well as packages written in Python. Python has become the lingua franca for data scientists, and support of NLP, machine learning, and deep learning libraries is plentiful. I refer to many of these packages when solving the example problems and discussing general concepts.

It is assumed that all readers of this book have a general understanding of Python, such that you have the ability to write software in this language. If you are not familiar with this language, but you are familiar with others, the concepts in this book will be portable with respect to the methodology used to solve problems, given the same data sets. Be that as it may, this book is not intended to instruct users on Python. Now, let's discuss some of the technologies that are most important to understanding deep learning.

TensorFlow

One of the groundbreaking releases in open source software, in addition to machine learning at large, has undoubtedly been Google's TensorFlow. It is an open source library for deep learning that is a successor to Theano, a similar machine learning library. Both utilize data flow graphs for computational processes. Specifically, we can think of computations as dependent on specific individual operations. TensorFlow functionally operates by the user first defining a graph/model, which is then operated by a TensorFlow session that the user also creates.

The reasoning behind using a data flow graph rather than another computational format computation is multifaceted, however one of the more simple benefits is the ability to port models from one language to another. Figure 1-2 illustrates a data flow graph.

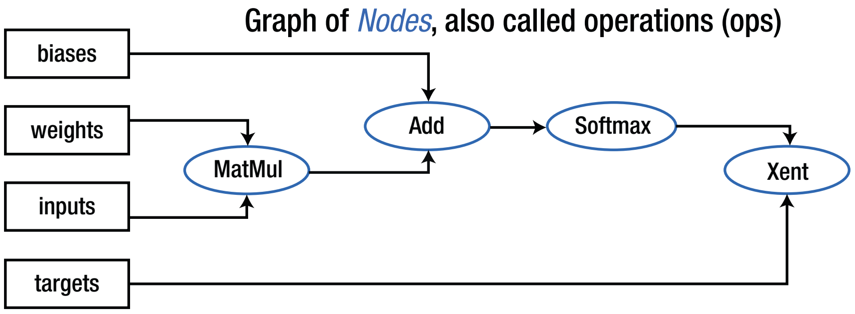

Figure 1-2. *Data flow graph diagram*

For example, you may be working on a project where Java is the language that is most optimal for production software due to latency reasons (high-frequency trading, for example); however, you would like to utilize a neural network to make predictions in your production system. Rather than dealing with the time-consuming task of setting up a training framework in Java for TensorFlow graphs, something could be written in Python relatively quickly, and then the graph/model could be restored by loading the weights in the production system by utilizing Java. TensorFlow code is similar to Theano code, as follows.

```
#Creating weights and biases dictionaries
weights = {'input': tf.Variable(tf.random_normal([state_
size+1, state_size])),
        'output': tf.Variable(tf.random_normal([state_size,
        n_classes]))}
    biases = {'input': tf.Variable(tf.random_normal([1, state_
    size])),
        'output': tf.Variable(tf.random_normal([1, n_classes]))}

    #Defining placeholders and variables
    X = tf.placeholder(tf.float32, [batch_size, bprop_len])
    Y = tf.placeholder(tf.int32, [batch_size, bprop_len])
    init_state = tf.placeholder(tf.float32, [batch_size, state_
    size])
    input_series = tf.unstack(X, axis=1)
    labels = tf.unstack(Y, axis=1)
    current_state = init_state
    hidden_states = []

    #Passing values from one hidden state to the next
    for input in input_series: #Evaluating each input within
    the series of inputs
        input = tf.reshape(input, [batch_size, 1]) #Reshaping
        input into MxN tensor
        input_state = tf.concat([input, current_state], axis=1)
        #Concatenating input and current state tensors
        _hidden_state = tf.tanh(tf.add(tf.matmul(input_
        state, weights['input']), biases['input'])) #Tanh
        transformation
        hidden_states.append(_hidden_state) #Appending the next
        state
        current_state = _hidden_state #Updating the current state
```

TensorFlow is not always the easiest library to use, however, as there often serious gaps between documentation for toy examples vs. real-world examples that reasonably walk the reader through the complexity of implementing a deep learning model.

In some ways, TensorFlow can be thought of as a language inside of Python, in that there are syntactical nuances that readers must become aware of before they can write applications seamlessly (if ever). These concerns, in some sense, were answered by Keras.

Keras

Due to the slow development process of applications in TensorFlow, Theano, and similar deep learning frameworks, Keras was developed for prototyping applications, but it is also utilized in production engineering for various problems. It is a wrapper for TensorFlow, Theano, MXNet, and DeepLearning4j. Unlike these frameworks, defining a computational graph is relatively easy, as shown in the following Keras demo code.

```
def create_model():
    model = Sequential()
    model.add(ConvLSTM2D(filters=40, kernel_size=(3, 3),
                       input_shape=(None, 40, 40, 1),
                       padding='same', return_sequences=True))
    model.add(BatchNormalization())

    model.add(ConvLSTM2D(filters=40, kernel_size=(3, 3),
                       padding='same', return_sequences=True))
    model.add(BatchNormalization())


    model.add(ConvLSTM2D(filters=40, kernel_size=(3, 3),
                       padding='same', return_sequences=True))
    model.add(BatchNormalization())

    model.add(ConvLSTM2D(filters=40, kernel_size=(3, 3),
                       padding='same', return_sequences=True))
    model.add(BatchNormalization())

    model.add(Conv3D(filters=1, kernel_size=(3, 3, 3),
                   activation='sigmoid',
                   padding='same', data_format='channels_last'))
    model.compile(loss='binary_crossentropy', optimizer='adadelta')
    return model
```

Although having the added benefit of ease of use and speed with respect to implementing solutions, Keras has relative drawbacks when compared to TensorFlow. The broadest explanation is that Keras users have considerably less control over their computational graph than TensorFlow users. You work within the confines of a sandbox when using Keras. TensorFlow is better at natively supporting more complex operations, and providing access to the most cutting-edge implementations of various algorithms.

Theano

Although it is not covered in this book, it is important in the progression of deep learning to discuss Theano. The library is similar to TensorFlow in that it provides developers with various computational functions (add, matrix multiplication, subtract, etc.) that are embedded in tensors when building deep learning and machine learning models. For example, the following is a sample Theano code.

```
(code redacted please see github)
X, Y = T.fmatrix(), T.vector(dtype=theano.config.floatX)
    weights = init_weights(weight_shape)
    biases = init_biases(bias_shape)
    predicted_y = T.argmax(model(X, weights, biases), axis=1)

    cost = T.mean(T.nnet.categorical_crossentropy(predicted_y, Y))
    gradient = T.grad(cost=cost, wrt=weights)
    update = [[weights, weights - gradient * 0.05]]
    train = theano.function(inputs=[X, Y], outputs=cost,
updates=update, allow_input_downcast=True)
    predict = theano.function(inputs=[X], outputs=predicted_y,
allow_input_downcast=True)

    for i in range(0, 10):
        print(predict(test_x_data[i:i+1]))

if __name__ == '__main__':

    model_predict()
```

When looking at the functions defined in this sample, notice that `T` is the variable defined for a tensor, an important concept that you should be familiar with. Tensors can be thought of as objects that are similar to vectors; however, they are distinct in that they are often represented by arrays of numbers, or functions, which are governed by specific transformation rules unique unto themselves. Tensors can specifically be a single point or a collection of points in space-time (any function/model that combines x, y, and z axes plus a dimension of time), or they may be a defined over a continuum, which is a *tensor field*. Theano and TensorFlow use tensors to perform most of the mathematical operations as data is passed through a computational graph, colloquially known as a *model*.

It is generally suggested that if you do not know Theano, you should focus on mastering TensorFlow and Keras. Those that are familiar with the Theano framework, however, may feel free to rewrite the existing TensorFlow code in Theano.

Applications of Deep Learning to NLP

This section discusses the applications of deep learning to NLP.

Introduction to NLP Techniques and Document Classification

In Chapter 3, we walk through some introductory techniques, such as word tokenization, cleaning text data, term frequency, inverse document frequency, and more. We will apply these techniques during the course of our data preprocessing as we prepare our data sets for some of the algorithms reviewed in Chapter 2. Specifically, we focus on classification tasks and review the relative benefits of different feature extraction techniques when applied to document classification tasks.

Topic Modeling

In Chapter 4, we discuss more advanced uses of deep learning, machine learning, and NLP. We start with topic modeling and how to perform it via latent Dirichlet allocation, as well as non-negative matrix factorization. Topic modeling is simply the process of extracting topics from documents. You can use these topics for exploratory purposes via data visualization or as a preprocessing step when labeling data.

Word Embeddings

Word embeddings are a collection of models/techniques for mapping words (or phrases) to vector space, such that they appear in a high-dimensional field. From this, you can determine the degree of similarity, or dissimilarity, between one word (or phrase, or document) and another. When we project the word vectors into a high-dimensional space, we can envision that it appears as something like what's shown in Figure 1-3.

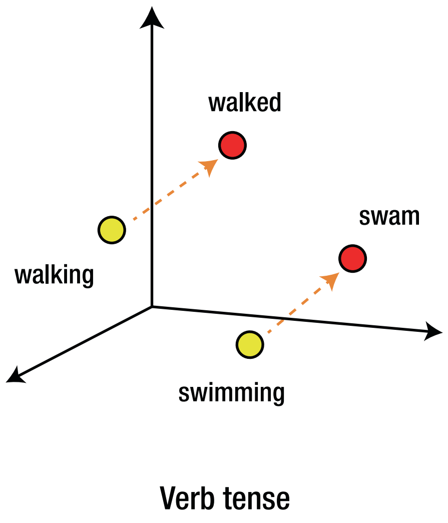

Figure 1-3. *Visualization of word embeddings*

Ultimately, how you utilize word embeddings is up to your own interpretation. They can be modified for applications such as spell check, but can also be used for sentiment analysis, specifically when assessing larger entities, such as sentences or documents in respect to one another. We focus simply on how to train the algorithms and how to prepare data to train the embeddings themselves.

Language Modeling Tasks Involving RNNs

In Chapter 5, we end the book by tackling some of the more advanced NLP applications, which is after you have been familiarized with preprocessing text data from various format and training different algorithms. Specifically, we focus on training RNNs to perform tasks such as name entity recognition, answering questions, language generation, and translating phrases from one language to another.

Summary

The purpose of this book is to familiarize you with the field of natural language processing and then progress to examples in which you can apply this knowledge. This book covers machine learning where necessary, although it is assumed that you have already used machine learning models in a practical setting prior.

While this book is not intended to be exhaustive nor overly academic, it is my intention to sufficiently cover the material such that readers are able to process more advanced texts more easily than prior to reading it. For those who are more interested in the tangible applications of NLP as the field stands today, it is the vast majority of what is discussed and shown in examples. Without further ado, let's begin our review of machine learning, specifically as it relates to the models used in this book.