SEBASTIAN RASCHKA

Introduction to Artificial Neural Networks and Deep Learning

A Practical Guide
with Applications in Python

Introduction to Artificial Neural Networks and Deep Learning

A Practical Guide with Applications in Python

Sebastian Raschka

Contents

Website . . . . . **i**

Appendix G - TensorFlow Basics **1**
TensorFlow in a Nutshell 1
Installation 3
Computation Graphs 3
Variables 5
Placeholder Variables 8
CPU and GPU 9

Website

Please visit the GitHub repository[1] to download code examples used in this book.

If you like the content, please consider supporting the work by buying a copy of the book on Leanpub[2].

I would appreciate hearing your opinion and feedback about the book! Also, if you have any questions about the contents, please don't hesitate to get in touch with me via mail@sebastianraschka.com or join the mailing list[3].

Happy learning!

Sebastian Raschka

[1] https://github.com/rasbt/deep-learning-book
[2] https://leanpub.com/ann-and-deeplearning
[3] https://groups.google.com/forum/#!forum/ann-and-dl-book

Appendix G - TensorFlow Basics

This appendix offers a brief overview of TensorFlow, an open-source library for numerical computation and deep learning. This section is intended for readers who want to gain a basic overview of this library before progressing through the hands-on sections that are concluding the main chapters.

The majority of *hands-on* sections in this book focus on TensorFlow and its Python API, assuming that you have TensorFlow >=0.12 installed if you are planning to execute the code sections shown in this book.

In addition to glancing over this appendix, I recommend the following resources from TensorFlow's official documentation for a more in-depth coverage on using TensorFlow:

- **Download and setup instructions**[4]
- **Python API documentation**[5]
- **Tutorials**[6]
- **TensorBoard, an optional tool for visualizing learning**[7]

TensorFlow in a Nutshell

At its core, TensorFlow is a library for efficient multidimensional array operations with a focus on deep learning. Developed by the Google Brain Team, TensorFlow was open-sourced on November 9th, 2015. And augmented by its convenient Python API layer, TensorFlow has gained much popularity and wide-spread adoption in industry as well as academia.

TensorFlow shares some similarities with NumPy, such as providing data structures and computations based on multidimensional arrays. What makes TensorFlow particularly suitable for deep learning, though, are its primitives for defining functions on tensors, the ability of parallelizing tensor operations, and convenience tools such as automatic differentiation.

---

[4] https://www.tensorflow.org/get_started/os_setup

[5] https://www.tensorflow.org/api_docs/python/

[6] https://www.tensorflow.org/tutorials/

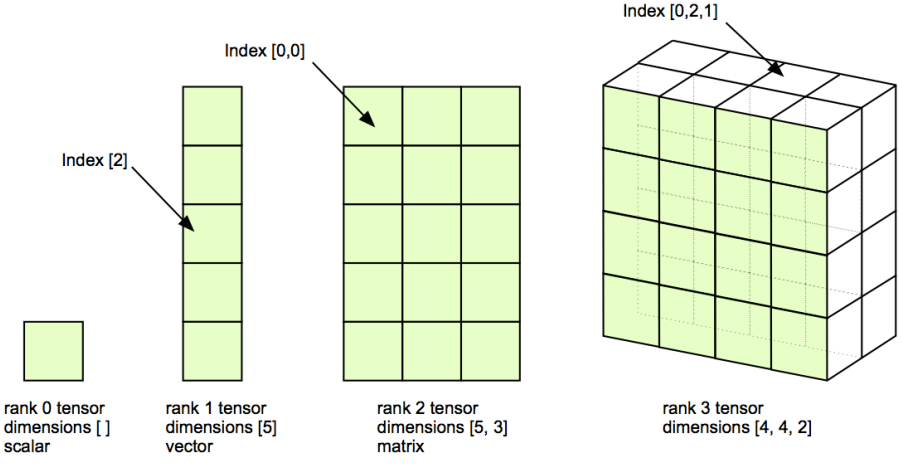
[7] https://www.tensorflow.org/how_tos/summaries_and_tensorboard/

While TensorFlow can be run entirely on a CPU or multiple CPUs, one of the core strength of this library is its support of GPUs (Graphical Processing Units) that are very efficient at performing highly parallelized numerical computations. In addition, TensorFlow also supports distributed systems as well as mobile computing platforms, including Android and Apple's iOS.

But what is a *tensor*? In simplifying terms, we can think of tensors as multidimensional arrays of numbers, as a generalization of scalars, vectors, and matrices.

1. Scalar: $\mathbb{R}$
2. Vector: $\mathbb{R}^n$
3. Matrix: $\mathbb{R}^n \times \mathbb{R}^m$
4. 3-Tensor: $\mathbb{R}^n \times \mathbb{R}^m \times \mathbb{R}^p$
5. ...

When we describe tensors, we refer to its "dimensions" as the *rank* (or *order*) of a tensor, which is not to be confused with the dimensions of a matrix. For instance, an $m \times n$ matrix, where m is the number of rows and n is the number of columns, would be a special case of a rank-2 tensor. A visual explanation of tensors and their ranks is given is the figure below.



Tensors

Installation

Code conventions in this book follow the Python 3.x syntax, and while the code examples should be backward compatible to Python 2.7, I highly recommend the use of Python >=3.5.

Once you have your Python Environment set up (Appendix - Python Setup), the most convenient ways for installing TensorFlow are via `pip` or `conda` – the latter only applies if you have the Anaconda/Miniconda Python distribution installed, which I prefer and recommend.

Since TensorFlow is under active development, I recommend you to consult the official "Download and Setup[8]" documentation for detailed installation instructions to install TensorFlow on you operating system, macOS, Linux, or Windows.

Computation Graphs

In contrast to other tools such as NumPy, the numerical computations in TensorFlow can be categorized into two steps: a construction step and an execution step. Consequently, the typical workflow in TensorFlow can be summarized as follows:

- Build a computational graph
- Start a new *session* to evaluate the graph
  - Initialize variables
  - Execute the operations in the compiled graph

Note that the computation graph has no numerical values before we initialize and evaluate it. To see how this looks like in practice, let us set up a new graph for computing the column sums of a matrix, which we define as a constant tensor (`reduce_sum` is the TensorFlow equivalent of NumPy's `sum` function).

In [1]:

[8] https://www.tensorflow.org/get_started/os_setup

```
import tensorflow as tf

g = tf.Graph()

with g.as_default() as g:
    tf_x = tf.constant([[1., 2.],
                        [3., 4.],
                        [5., 6.]], dtype=tf.float32)
    col_sum = tf.reduce_sum(tf_x, axis=0)

print('tf_x:\n', tf_x)
print('\ncol_sum:\n', col_sum)
```

Out [1]:

```
tf_x:
 Tensor("Const:0", shape=(3, 2), dtype=float32)

col_sum:
 Tensor("Sum:0", shape=(2,), dtype=float32)
```

As we can see from the output above, the operations in the graph are represented as `Tensor` objects that require an explicit evaluation before the `tf_x` matrix is populated with numerical values and its column sum gets computed.

Now, we pass the graph that we created earlier to a new, active *session*, where the graph gets compiled and evaluated:

In [2]:

```
with tf.Session(graph=g) as sess:
    mat, csum = sess.run([tf_x, col_sum])

print('mat:\n', mat)
print('\ncsum:\n', csum)
```

Out [2]:

```
mat:
 [[ 1.  2.]
 [ 3.  4.]
 [ 5.  6.]]

csum:
 [  9.  12.]
```

Note that if we are only interested in the result of a particular operation, we don't need to `run` its dependencies – TensorFlow will automatically take care of that. For instance, we can directly fetch the numerical values of `col_sum_times_2` in the active session without explicitly passing `col_sum` to `sess.run(...)` as the following example illustrates:

In [3]:

```
g = tf.Graph()

with g.as_default() as g:
    tf_x = tf.constant([[1., 2.],
                        [3., 4.],
                        [5., 6.]], dtype=tf.float32)
    col_sum = tf.reduce_sum(tf_x, axis=0)
    col_sum_times_2 = col_sum * 2


with tf.Session(graph=g) as sess:
    csum_2 = sess.run(col_sum_times_2)

print('csum_2:\n', csum_2)
```

Out [3]:

```
csum_2:
 [array([ 18.,  24.], dtype=float32)]
```

Variables

Variables are constructs in TensorFlow that allows us to store and update parameters of our models during training. To define a "variable" tensor, we use TensorFlow's `Variable()`

constructor, which looks similar to the use of `constant` that we used to create a matrix previously. However, to execute a computational graph that contains variables, we must initialize all variables in the active session first (using `tf.global_variables_initializer()`), as illustrated in the example below.

In [4]:

```
g = tf.Graph()

with g.as_default() as g:
    tf_x = tf.Variable([[1., 2.],
                        [3., 4.],
                        [5., 6.]], dtype=tf.float32)
    x = tf.constant(1., dtype=tf.float32)

    # add a constant to the matrix:
    tf_x = tf_x + x

with tf.Session(graph=g) as sess:
    sess.run(tf.global_variables_initializer())
    result = sess.run(tf_x)
```

print(result)

```
print(result)
```

Out [4]:

```
[[ 2.  3.]
 [ 4.  5.]
 [ 6.  7.]]
```

Now, let us do an experiment and evaluate the same graph twice:

In [5]:

```
with tf.Session(graph=g) as sess:
    sess.run(tf.global_variables_initializer())
    result = sess.run(tf_x)
    result = sess.run(tf_x)
```

Out [5]:

```
[[ 2.  3.]
 [ 4.  5.]
 [ 6.  7.]]
```

As we can see, the result of running the computation twice did not affect the numerical values fetched from the graph. To update or to assign new values to a variable, we use TensorFlow's `assign` operation. The function syntax of `assign` is `assign(ref, val, ...)`, where '`ref`' is updated by assigning '`value`' to it:

In [6]:

```
g = tf.Graph()

with g.as_default() as g:
    tf_x = tf.Variable([[1., 2.],
                        [3., 4.],
                        [5., 6.]], dtype=tf.float32)
    x = tf.constant(1., dtype=tf.float32)

    update_tf_x = tf.assign(tf_x, tf_x + x)


with tf.Session(graph=g) as sess:
    sess.run(tf.global_variables_initializer())
    result = sess.run(update_tf_x)
    result = sess.run(update_tf_x)

print(result)
```

Out [6]:

```
[[ 3.  4.]
 [ 5.  6.]
 [ 7.  8.]]
```

As we can see, the contents of the variable `tf_x` were successfully updated twice now; in the active session we

- initialized the variable `tf_x`
- added a constant scalar `1.` to `tf_x` matrix via `assign`
- added a constant scalar `1.` to the previously updated `tf_x` matrix via `assign`

Although the example above is kept simple for illustrative purposes, variables are an important concept in TensorFlow, and we will see throughout the chapters, they are not only useful for updating model parameters but also for saving and loading variables for reuse.

Placeholder Variables

Another important concept in TensorFlow is the use of placeholder variables, which allow us to feed the computational graph with numerical values in an active session at runtime.

In the following example, we will define a computational graph that performs a simple matrix multiplication operation. First, we define a placeholder variable that can hold 3x2-dimensional matrices. And after initializing the placeholder variable in the active session, we will use a dictionary, `feed_dict` we feed a NumPy array to the graph, which then evaluates the matrix multiplication operation.

In [7]:

```
import numpy as np

g = tf.Graph()

with g.as_default() as g:
    tf_x = tf.placeholder(dtype=tf.float32,
                          shape=(3, 2))

    output = tf.matmul(tf_x, tf.transpose(tf_x))


```

```
with tf.Session(graph=g) as sess:
    sess.run(tf.global_variables_initializer())
    np_ary = np.array([[3., 4.],
                       [5., 6.],
                       [7., 8.]])
    feed_dict = {tf_x: np_ary}
    print(sess.run(output,
                   feed_dict=feed_dict))
```

Out [7]:

```
[[  25.   39.   53.]
 [  39.   61.   83.]
 [  53.   83.  113.]]
```

Throughout the main chapters, we will make heavy use of placeholder variables, which allow us to pass our datasets to various learning algorithms in the computational graphs.

CPU and GPU

Please note that all code examples in this book, and all TensorFlow operations in general, can be executed on a CPU. If you have a GPU version of TensorFlow installed, TensorFlow will automatically execute those operations that have GPU support on GPUs and use your machine's CPU, otherwise. However, if you wish to define your computing device manually, for instance, if you have the GPU version installed but want to use the main CPU for prototyping, we can run an active section on a specific device using the `with` context as follows

```
with tf.Session() as sess:
    with tf.device("/gpu:1"):
```

where

- "/cpu:0": The CPU of your machine.
- "/gpu:0": The GPU of your machine, if you have one.
- "/gpu:1": The second GPU of your machine, etc.
- etc.

You can get a list of all available devices on your machine via

```
from tensorflow.python.client import device_lib

device_lib.list_local_devices()
```

For more information on using GPUs in TensorFlow, please refer to the GPU documentation at https://www.tensorflow.org/how_tos/using_gpu/.