4 Python

Objective

One reason that there has been significant progress in deep learning is the availability of open source software tools (generally referred to as frameworks) for implementing deep networks. In this textbook we will interleave theoretical discussions with discussions of these frameworks. The interface most commonly used to access the frameworks is the Python programming language. In this chapter we provide a brief introduction to Python and a few of its most commonly used packages. This chapter is available as an interactive Jupyter Notebook at the website https://github.com/NNDesignDeepLearning/NNDesignDeepLearning/blob/master/05.PythonChapter/Code/ChapterNotebook/PythonChapter.ipynb

Theory and Examples

There are a number of open source software tools for developing deep neural networks. Many of these tools are called *frameworks*. A software framework provides a high-level interface for accessing a library of modular and highly optimized low-level functions. In the case of the deep learning frameworks, the low-level functions implement elementary neural network operations, like matrix multiplication, on CPUs or GPUs.

The two most popular frameworks are TensorFlow and PyTorch, which we will cover in later chapters of this text. Both of these frameworks have a Pythonic interface. In this chapter, to lay the groundwork for the framework chapters, we will first review some of the key Python concepts that are most important to the use of deep learning frameworks. In addition to the Python core, this chapter will also cover two important Python packages, NumPy and Pandas, which are very commonly used in deep learning workflows for loading and preparing data for training.

Of course it would be impossible to provide a complete coverage of Python, NumPy and Pandas in one chapter. Consider it a brief introduction to get you started. Please check out the Further Reading section for more detailed discussions.

Python

Python is a high-level, interpreted, general-purpose programming language created by Guido van Rossum and first released in 1991. It is currently the most popular language for machine learning, because of its balance of simplicity, flexibility and performance. Almost all of the deep learning frameworks are implemented with Python on the surface and C, C++ or CUDA (for GPU programming) under the hood. Python is open-source, with thousands of third-party modules (extensions, libraries). These include TensorFlow and PyTorch.

In this subsection, we will provide a brief introduction to the key features of Python that we will use for deep learning. To execute cells in the Jupyter Notebook, click into the cell and press `Shift+Enter`.

VARIABLES in Python are dynamically typed. If you enter an integer value for a variable, it will be typed as integer, but if you include a decimal point, it will be typed as floating point, as in the following

example.

```
a = 2
b = 3
c = a + b
print(c)
```

```
5
```

```
print(a/b)
```

```
0
```

```
d = 3.0
print(a/d)
```

```
0.6666666666666666
```

Lists: An important compound data type in Python is the *list*. A list can be written as a series of items, separated by commas, inside square brackets. They can be *indexed* (accessing individual elements by their position in the list) and *sliced* (selecting a subset of a list), as in the following examples.

```
a = [2, 4, 6, 8]
print(a[0])
```

```
2
```

Indexing in Python begins with 0.

```
print(a[-1])  # neg. index cnts from right
```

```
8
```

You can access the last element of a list with the index −1.

```
a[1:3] = [] # remove elements 1 and 2
print(a)
```

```
[2, 8]
```

You can add comments to code using the # symbol.

TUPLES: A tuple is similar to a list, but the tuple is *immutable*, which means that it cannot be changed after it is created. The list is mutable, which means that it is not write-protected, and therefore not as safe as a tuple. We will sometimes use tuples to pass inputs and targets into a training function.

```
a = 2, 4, 6, 8
print(a)
```

```
(2, 4, 6, 8)
```

A tuple is a list of items separated by a comma.

```
print(a[0])
```

```
2
```

```
a[0] = 9
```

```
raceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item
    ↪ assignment
```

This error occurs when you try to change a tuple.

DICTIONARIES are another useful data type in Python. A dictionary is a set of key-value pairs, where the key can be any immutable object (object that cannot be changed). A dictionary can be created by putting a comma-separated list of key:value pairs inside a set of braces. Here we create a dictionary that contains inputs and targets that could be used to train a neural network.

```
dict = {'input': [1, 3, 7], 'target':[3, 7, 15]}
print(dict['input'])
```

```
[1, 3, 7]
```

You can access a value by using the key.

```
print(dict.items())
```

```
[('input', [1, 3, 7]), ('target', [3, 7, 15])]
```

```
print(dict.keys())
```

```
['input', 'target']
```

IF STATEMENT: In addition to calculator functions, Python has the usual control flow tools that are found in other programming languages, such as the *if* statement. The first `if` can be followed by an `else`, or any number of `elif`'s, or it can be used alone. Unlike some other languages, Python uses indentation to group statements, so there is no `end` statement. Use the tab or spaces to create the indentation, but be careful to keep the exact spacing, and use a colon (:) at the end of the `if`, `elif` and `else` lines. If in interactive mode, leave a blank line at the end.

The standard Python convention is to use 4 spaces for indenting and not to use tabs.

```
x = -5
if x<0:
    y = -1
elif x>0:
    y = 1
else:
    y = 0
print(y)
```

```
-1
```

WHILE LOOP: Another control flow tool is the *while* loop, as shown in the following example.

```
a = [1, 2, 3, 4]
i, x = 0, 0  #Multiple assignment
while i<len(a):
    x = x + a[i]
    i = i + 1

print(x)
```

```
10
```

Notice that Python allows multiple variables to be assigned on the same line.

FOR LOOP: Python also has *for* loops that can be used to iterate over the items of any list.

```
x = 0
for num in a:
    x = x + num

print(x)
```

```
10
```

The index variable does not have to be an integer.

```
a = ['one', 'two', 'three', 'four']
for q in a:
    print(q)
```

```
one
two
three
four
```

For loops can also be used in *list comprehensions* to create lists.

```
a = [1, 2, 3, 4]
b = [x**2 for x in a]
print(b)
```

```
[1, 4, 9, 16]
```

LOGICAL OPERATORS: The logical operators in Python are `and`, `or` and `not`.

```
x, y = True, False
print(x and y)
print(x or y)
print(not x)
```

```
False
True
False
```

FUNCTIONS: The keyword `def` introduces the function definition. It is followed by the function name and parenthetical parameters. The body must be indented.

```
def f(y,q):
    return [y*y, 1/q]
```

The `return` line indicates which variables will be returned from the function.

```
zw = f(4.0,2.0)
print(zw)
```

```
[16.0, 0.5]
```

ITERATORS AND GENERATORS: Python *containers*, like `list`, `dict` and `tuple`, are *iterables*, which means that they can be iterated over. Technically, the `for` statement calls the `iter()` method with the container as the argument, which returns an *iterator* object. The iterator object has a method `__next__()` that accesses elements in the container one at a time. An iterator keeps an internal state, so it knows how to compute the next element.

The operation of iterating over objects in a container is important for training neural networks, since we generally need to bring data into the training process in an iterative way. The procedures for doing this will be critical, especially when data sets are too large to fit into memory at one time, or when processing is to be distributed across multiple CPUs and GPUs.

A *generator* is a type of iterator. It is written like a function, but with `return` replaced by `yield` . The `yield` statement pauses the function so that it can resume where it left off. Here is a generator that returns elements of a list two at a time.

```
def bytwo(x):
    n = len(x)
    for i in range(0, n-1, 2):
        yield x[i:i+2]
```

The range function generates arithmetic progressions from an initial value to a final value by some increment.

```
a=[1, 2, 3, 4]
zz = bytwo(a)
for qq in zz:
    print(qq)
```

```
[1, 2]
[3, 4]
```

MODULES: After an interactive Python session is closed, all definitions are lost. In order to save definitions, they can be written to a file and then later imported. This type of file is called a *module*. Definitions in a module can be imported into other modules or the main function. For example, assume the file `logic.py` contains the following.

```
def a(x,y):
    print(x and y)

def o(x,y):
    print(x or y)
```

We can call this module's functions with the following syntax.

```
import logic
print(logic.a(True,False))
print(logic.o(True,False))
```

```
False
True
```

Later we will be importing definitions from TensorFlow and PyTorch.

Classes are a way to combine data and functionality. Constructing a new class makes a new object and associated *methods* that can modify the object. For example, the following commands define a new object, which is a simple one-layer network with one input and one neuron. It has a simulation method, which defines how the network operates.

Methods are functions that are associated with a specific class of objects.

```
class simplenet:
    def __init__(self,weight,bias):
        self.w = weight
        self.b = bias

    def sim(self,p):
        return self.w*p + self.b
```

The `__init__` method is executed when an instance is created.

The `sim` function is a method for the simplenet class.

You can then create an instance of the class `simplenet` and simulate it.

```
net = simplenet(4.0,2.0)
print(net.sim(3.0))
```

```
14.0
```

NumPy

NumPy is the fundamental module for scientific computing in Python. NumPy functionality works conveniently with the deep learning frameworks we cover in this book, so let's consider some of its key components and concepts.

The key object in NumPy is the multidimensional array (tensor), which can be used to hold inputs and outputs of neural networks. An array can be constructed using the `array` function. The key attributes of an array are: `ndim` – the number of dimensions, `shape` – the sizes of each dimension, and `dtype` – the type of the elements in the array.

Python

```
import numpy as np
x = np.array([[1, 2, 3],[4, 5, 6]])
print(x)
print(x.ndim)
print(x.shape)
print(x.dtype)
```

```
[[1 2 3]
 [4 5 6]]
2
(2, 3)
int64
```

Array Operations: A number of operations can be performed between arrays: adding, subtracting, multiplying (dot and Hadamard) and dividing. To demonstrate, create some arrays of various sizes.

```
a = np.arange(6)
print(a)
```

```
[0 1 2 3 4 5]
```

The `arange` function returns evenly spaced values in a range. If we use only one argument, the initial value is zero, the increment is one and the argument is the total number of values.

```
b = a.reshape(2,3)
print(b)
c = np.arange(0,12,2).reshape(2,3)
print(c)
d = np.arange(0,24,4).reshape(3,2)
print(d)
```

```
[[0 1 2]
 [3 4 5]]
[[ 0  2  4]
 [ 6  8 10]]
[[ 0  4]
 [ 8 12]
 [16 20]]
```

The `reshape` method rearranges the elements in the array to have the specified numbers of elements in each dimension.

```
e = np.arange(3)
print(e)
```

```
[0 1 2]
```

The first dimension can be considered to be the number of rows, so `e` is a column vector, although displayed as a row here.

There are two types of array multiplication. The first is a Hadamard (or element-by-element) multiplication. The second is standard matrix multiplication.

```
print(b*c)
```

```
[[ 0  2  8]
 [18 32 50]]
```

The * represents Hadamard multiplication. The two arrays should be the same size.

```
print(np.matmul(c,d))
```

```
[[ 80 104]
 [224 320]]
```

For standard matrix muliplications (same as `dot` product for two dimensional arrays), columns of `c` must match rows of `d`.

We can also add arrays. Notice that in the second example the two arrays are not the same dimension.

```
print(b+c)
print(b+e)
```

```
[[ 0  3  6]
 [ 9 12 15]]
[[0 2 4]
 [3 5 7]]
```

When the arrays are not the same dimension, the smaller array is *broadcast* across the larger array so that they have compatible shapes.

Indexing and Slicing: As with lists, you can access individual elements of an array with *indexing* or subsets of an array with *slicing*.

```
print(b[1, 2])
print(b[0])
print(b[[0, 1],[1, 2]])
```

```
5
[0 1 2]
[1 5]
```

The basic slice syntax is `i:j:k` where `i` is the starting index, `j` is the stopping index, and `k` is the step.

```
print(a[0:5:2])
print(a[:5])
print(a[-4:])
```

```
[0 2 4]
[0 1 2 3 4]
[2 3 4 5]
```

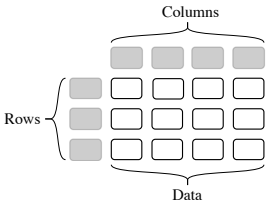
It is also possible to compute the sum or product of values of an array across various dimensions.

```
print(np.sum(b))
print(np.sum(b,axis=0))
print(np.prod(b,axis=1))
```

```
15
[3 5 7]
[ 0 60]
```

Pandas

Much of the effort (sometimes more than 90%) in any deep learning workflow is devoted to *data wrangling*. This involves loading, formatting and preprocessing the data before it is used to train the deep learning model. A Python module that is commonly used for data wrangling is *Pandas*. Pandas is an open source Python package (built on top of NumPy) that provides data structures as well as data analysis and manipulation tools. In this subsection we summarize some of the most useful features of Pandas.



The DataFrame: The main data structure of Pandas is *DataFrame*. DataFrame is a 2-dimensional structure with columns of potentially different types. It is similar to a spreadsheet.

Loading the Data: Pandas data frames are normally created by reading the data in from a file. At the time of this writing, Pandas can read in 14 different types of files:

1. Comma-separated values (CSV)
2. XLSX
3. ZIP
4. Plain Text (txt)
5. JSON
6. XML
7. HTML
8. Images
9. Hierarchical Data Format
10. PDF
11. DOCX
12. MP3
13. MP4
14. SQL

To illustrate how the process works, we will load a sample .csv file. (The file contains information about how patients performed on a particular medical test, as we will see later.) All of the Pandas methods for reading data files begin with the characters `read_`. For `.csv` files the command is `read_csv`.

```
import pandas as pd
sample_df = pd.read_csv('SampleDF.csv')
```

This is a small part of a larger file that was used as part of a Kaggle competition. The sample file is `SamplePulmonary.csv`, and can be downloaded from the textbook companion website `https://github.com/NNDesignDeepLearning/NNDesignDeepLearning`.

Getting Basic Info: Now that we have loaded a data frame, let's use some simple commmands to investigate its structure and contents. To find the shape of the DataFrame we can use `shape`.

```
print(sample_df.shape)
```

```
(100, 7)
```

The DataFrame has 100 rows and 7 columns. Each row corresponds to a patient's test result. We can find the names of the columns using `columns`.

Python

```
print(sample_df.columns)
```

```
Index(['Patient', 'Weeks', 'FVC', 'Percent', 'Age', 'Sex',
    ↪  'SmokingStatus'], dtype='object')
```

The first column is a patient id number, followed by the week in which the test was performed, the value of the test, the rating of the test as a percentage of an average patient in the same category, age, sex and smoking status (current, previous or never).

It is also possible to get a description of the DataFrame using the `describe()` method.

```
print(sample_df.describe())
```

```
            Weeks          FVC     Percent         Age
count  100.00000   100.000000  100.000000  100.000000
mean    35.89000  2759.820000   78.187965   67.110000
std     24.90868   925.766484   21.094723    6.738844
min      0.00000   969.000000   43.352279   49.000000
25%     16.50000  2118.000000   62.821569   64.000000
50%     32.00000  2597.500000   73.989508   68.000000
75%     49.00000  3267.000000   89.005946   72.000000
max    116.00000  5768.000000  153.145378   87.000000
```

This provides statistics on all of the numerical columns, including minimum and maximum values, mean and standard deviation, and quartiles.

We can also get some summary information using the `info` method.

```
sample_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100 entries, 0 to 99
Data columns (total 7 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   Patient        100 non-null    object
 1   Weeks          100 non-null    int64
 2   FVC            100 non-null    int64
 3   Percent        100 non-null    float64
 4   Age            100 non-null    int64
 5   Sex            100 non-null    object
 6   SmokingStatus  100 non-null    object
dtypes: float64(1), int64(3), object(3)
memory usage: 5.6+ KB
```

It provides information about each of the columns, including the `dtype` and the number of missing cells.

In addition to getting statistics on the DataFrame, we can also look at some parts of the DataFrame by using the `head()` and `tail()` methods. The `head()` method shows the first few rows of the DataFrame, and `tail()` shows the last few rows.

```
print(sample_df.head())
```

```
   Patient  Weeks   FVC    Percent  Age     Sex  \
0  ID00213     32  2972  81.828194   70    Male
1  ID00129      0  2253  59.622102   71    Male
2  ID00130     12  1648  68.116062   65  Female
3  ID00225     23   969  49.075715   77  Female
4  ID00082     33  2885  98.666211   49  Female

      SmokingStatus
0  Currently smokes
1      Never smoked
2      Never smoked
3      Never smoked
4  Currently smokes
```

Indexing and Slicing: Data can be extracted from a DataFrame in a number of ways. For example, you can work with a single column.

To select the column, provide the column name between square brackets – `[]`. The result is a Pandas *Series*.

```
fvc = sample_df['FVC']
fvc.describe()
```

```
count     100.000000
mean     2759.820000
std       925.766484
min       969.000000
25%      2118.000000
50%      2597.500000
75%      3267.000000
max      5768.000000
Name: FVC, dtype: float64
```

If you want to select multiple columns, you can enter a list of column names within the brackets. The returned object is a DataFrame:

```
twocol = sample_df[['Age', 'FVC']]
print(twocol.head())
```

```
   Age   FVC
0   70  2972
1   71  2253
2   65  1648
3   77   969
4   49  2885
```

It is also possible to select rows of a DataFrame. For example, you can consider only patients older than 75.

```
older = sample_df[sample_df['Age']>75]
print(older.shape)
```

```
(8, 7)
```

The condition inside the brackets – `sample_df['Age']>75` – checks which rows in the Age column have value greater than 75. The resulting shape shows that there are 8 patients older than 75.

If you want to access a select set of rows and columns, you need

to use the `loc/iloc` operators. If you are using column names, row labels or a condition expression, use the `loc` operator. When accessing rows and/or columns based on their position in the table, use the `iloc` operator. For example, if you want to access the Age and FVC value of patients over 75, you could use the following commands.

```
older_fvc = sample_df.loc[sample_df['Age']>75, ['Age', '
    ↪ FVC']]
print(older_fvc.head())
```

```
    Age   FVC
3    77   969
7    83  3171
16   87  2220
59   77  1550
82   77  1818
```

The part before the comma in the selection bracket represents the rows you want, and the part after the comma represents the columns you want.

Here is an example of using the `iloc` operator to access rows 3 through 5 and columns 2 to 3.

```
sample_subset = sample_df.iloc[2:5, 1:3]
print(sample_subset.head())
```

```
   Weeks   FVC
2     12  1648
3     23   969
4     33  2885
```

Reshaping: It is often useful to reshape a DataFrame in order to make the data compatible with a neural network input or target structure. There are several methods for reshaping DataFrames. We will touch on just a few popular ones.

The Pandas `pivot()` method takes three columns of the DataFrame as input arguments and then produces a new data frame that uses the three columns as index, columns and values. For example, we can take the `Patient` column to be the index, the `Weeks` column to be the columns and the `FVC` column to be the values, as in the

following.

```
pivoted = sample_df.pivot(index='Patient', values = 'FVC',
    ↪  columns='Weeks')
print(pivoted.iloc[:5, :10])
```

```
Weeks    0   2   4   5   6   7      9   10     12     13
Patient
ID000116 NaN NaN NaN NaN NaN NaN 3541.0 NaN    NaN 3410.0
ID000156 NaN NaN NaN NaN NaN NaN    NaN NaN    NaN    NaN
ID000206 NaN NaN NaN NaN NaN NaN    NaN NaN    NaN    NaN
ID000276 NaN NaN NaN NaN NaN NaN    NaN NaN 2472.0    NaN
ID000306 NaN NaN NaN NaN NaN NaN    NaN NaN    NaN    NaN
```

There is a column for every week that is indicated in the data set, and there is a row for every patient. If there is no FVC value for a certain patient in a given week in the original DataFrame, then the cell is filled with `NaN`.

Another reshaping method is `melt()`. It is used to change the DataFrame format from wide to long. One or more of the original columns are used as identifiers, and one or more remaining columns become one value column. For example, we can use `Patient` as the identifier, and `Age` as the value variable.

```
melted = sample_df.melt(id_vars='Patient',value_vars='Age')
print(melted.head())
```

```
                     Patient variable  value
0  ID00213637202257692916109      Age     70
1  ID00129637202219868188000      Age     71
2  ID00130637202220059448013      Age     65
3  ID00225637202259339837603      Age     77
4  ID00082637202201836229724      Age     49
```

Operations on DataFrames: Before applying data to neural network training, we often need to perform various preprocessing operations on the data. One way to do this would be to use `for` loops to iterate over the elements of the DataFrame. However, there are Pandas methods that are much more efficient. A very useful Pandas method is `apply()`. It can be used to apply a function along an axis of the DataFrame. There are two arguments to the `apply()`

method: the function you want to apply, and the axis you want to apply it to.

For example, consider the DataFrame `old_fvc` that we created earlier. If we want to find the average `'Age'` and `'FVC'` values, we can apply the NumPy `mean` function to the columns of the DataFrame.

```
print(older_fvc.apply(np.mean, axis=0))
```

```
Age      79.125
FVC    2284.500
dtype: float64
```

In addition to using built-in functions, you can also apply user-defined functions. In the example below we pass a Python `lambda` function to the `apply()` method. The first argument of `apply()` is the function to apply. This can be a built-in function, or a user-defined function. If the function is simple, it can be passed as a `lambda` function, which can be defined inline. A lambda expression consists of the *keyword* `lambda`, followed by the *bound variable* name, followed by a colon, and then completed by the *body* of the function. In the example below the bound variable is `x`, and the body is `x.max() - x.min()`. The `apply()` operation will compute the range of values (max minus min) in each column of the `old` DataFrame.

```
print(older_fvc.apply(lambda x: x.max() - x.min(), axis=0)
    ↪ )
```

```
Age      11
FVC    3081
dtype: int64
```

The `apply()` method can be used to apply any previously defined function. There are also built-in functions that can be used directly. For example, the `min()` method is applied below to find the minimum value in each column of the data frame.

Python

```
print(older_fvc.min(axis=0))
```

```
Age     76
FVC    969
dtype: int64
```

There are many more built-in DataFrame methods, including `max()`, `sum()`, `mean()`, `medium()`, `std()`, etc.

Epilogue

Most of the deep learning frameworks, like TensorFlow and PyTorch, can be accessed using Python. This chapter has provided a brief introduction to Python and two of its commonly used packages – NumPy and Pandas. This chapter is meant only as a quick introduction to Python, which will be used extensively in the next chapter on TensorFlow. For more in-depth coverage of this chapters' topics, see the Further Reading section on the next page.

Further Reading

[Python, 2019a] Software Foundation Python. Welcome to python.org. https://www.python.org/, 2019a. Accessed: 2019-06-24

This is the main web page for Python. From there you can download the latest releases and access documentation

[Python, 2019b] Software Foundation Python. The python tutorial. https://docs.python.org/3/tutorial/index.html, 2019b. Accessed: 2019-06-24

This is the official Python Tutorial. It is an excellent introduction to all of the features of Python.

[NumPy, 2019a] Developers NumPy. Numpy. https://www.numpy.org/, 2019a. Accessed: 2019-06-24

The main web page for NumPy. From here you can access downloads and documentation.

[NumPy, 2019b] Developers NumPy. Quickstart tutorial - numpy. https://www.numpy.org/devdocs/user/quickstart.html, 2019b. Accessed: 2019-06-24

The official NumPy tutorial. It covers the basic NumPy features and gives numerous examples.

[Pandas, 2021] Pandas. Pandas user guide. https://pandas.pydata.org/pandas-docs/stable/user_guide/index.html, 2021. Accessed: 2021-07-14

The official Pandas User Guide. It has a quick 10 minute introduction, as well as complete documentation of all features.

Labs

The labs that go with this chapter can be found at https://github.com/NNDesignDeepLearning/NNDesignDeepLearning/tree/master/05.PythonChapter/Code. Some of the labs will be in Jupyter Notebooks. You can open the labs and run them for free in Google Colab, if you have a gmail account. You can also download and run them on your personal computer.