PROGRAMMING IN PYTHON I

Unit 08: Fast numerical computations and storing Python objects as files

Michael Widrich
Institute for Machine Learning

JOHANNES KEPLER
UNIVERSITY LINZ

FAST NUMERICAL COMPUTATIONS IN PYTHON

Motivation

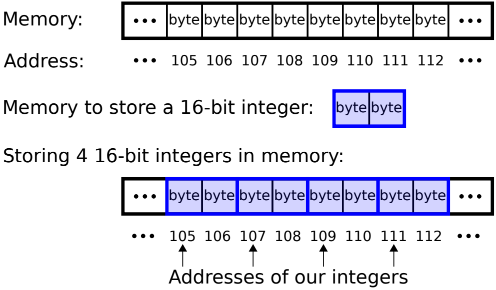
- We already learned that Python is an interpreted language
  - Very convenient to use
  - Slow, since optimization of the code is difficult at runtime
- We can use modules in Python that allow us to write fast code in Python
  - By providing optimized functions (e.g. `NumPy`, ...)
  - By providing tools for optimizing Python-like code (e.g. `Numba`, `PyTorch`, `Tensorflow`, ...)

NumPy

- NumPy is the go-to module for numerical computations in Python
- Provides a large range of functionalities for performing scientific computations and handling array data
  - These functions are typically highly optimized and implemented in C
- NumPy mainly deals with (multidimensional) array data based on the `numpy.ndarray` object
- Documentation/Tutorials:
  `https://numpy.org/doc/stable/index.html`

Arrays in NumPy (1)

- We already heard about the simple linear array
  - Elements are stored as one block with contiguous addresses in memory
  - Elements are fast to access since we can quickly compute their addresses



Arrays in NumPy (2)

- In Python, an element in a list is like a Python variable that is a reference to an object
  - Datatypes of objects are flexible
  - → Operations on elements are slower/clumsy (need to determine type of object before usage)
- In Numpy, an element in an array is (usually) a bit-pattern that directly represents the stored value
  - The array holds the information about the datatype (encoding/decoding scheme for bits) used in array
  - Datatype of elements in array is fixed (but we can create new arrays with a different datatype)
  - All elements in an array have the same datatype
  - → Operations on elements can be optimized better and are faster

Multidimensional arrays

- In Python lists, we already saw the concept of nested lists
  - Can be used to create 2-D or n-D arrays
  - Slow since we have to access the sub-lists to access our elements
- We can store n-D arrays as fast 1-D arrays
  - Done by NumPy in the background
  - Store n-D array in a flat manner
  - Row-major order: Consecutive elements of a row reside next to each other
  - Column-major order: Consecutive elements of a column reside next to each other

Multidimensional arrays: example (1)

- We want to store a 2-D array with 3 rows and 5 columns
  - 5 elements per row, 3 per column, 15 in total

0	1	2	3	4
5	6	7	8	9
10	11	12	13	14

- We can create a 1-D array with 15 elements
- We can say that
  - the first 5 elements belong to the row in the first column
  - the next 5 elements belong to the row in the second column
  - the last 5 elements belong to the row in the third column
  - = row-major order

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

Multidimensional arrays: example (2)

- We agreed on row-major order
- Now we want to access the element in the 4th column $c = 3$ and the 3rd row $r = 2$ (indices starting at 0 with $n_r = 5$ elements per row)

0	1	2	3	4
5	6	7	8	9
10	11	12	13	14

- We can compute the index in the 1-D array via $n_r \cdot r + c = 5 \cdot 2 + 3 = 13$

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

Indexing in NumPy

- Accessing arrays in Numpy is similar to accessing lists in Python
  - Index via integers:
    `my_array[my_index]`
  - Slicing is possible and fast (since elements are consecutively stored in memory):
    `my_array[:my_index]`
- Numpy offers many more fancy indexing options
  - Indexing multi-dimensional arrays directly:
    `my_array[my_row_index, my_col_index]`
    `my_array[2, 4, 8, 5]`
  - Indexing using lists of indices, boolean index masks, ...

STORING PYTHON OBJECTS TO FILES

Pickle and dill: multi-purpose storage

- pickle module
  - Allows us to store and load many types of Python objects
  - Stores data in binary files
  - Not compressed (unless we compress the file using compression modules)
  - Can handle many different Python objects
- dill module
  - Same interface as pickle
  - Extends functionality of pickle
  - Can store more types of Python objects
  - Often used as `import dill as pickle`

H5py: storing large data

- hdf5
  - Container-format that allows for storing large data
- h5py module
  - Uses hdf5 format to store Python objects in binary files
  - Container-access via dictionary-like interface
  - Array-access via NumPy-like indexing and slicing
  - Supports different compression algorithms
  - Accessing arrays can be done chunk-wise (e.g. if array does not fit in RAM at once)
  - Types of store-able objects are limited