Introduction to DL for Master Students

Prof. Alexander Binder

Week 01: python basics, numpy indexing + Inner product/Skalarprodukt

Task 0: Get a virtual environment for python running

Suggestion: use jupyter-lab which one can install with pip (in a python virtual environment!), conda or miniconda (in conda environments).

To create such environments:

---

pip:

```
#to create the venv in <yourpath>
python3 -m venv <yourpath>

#to activate the venv in <yourpath>
source <yourpath>/bin/activate

#install packages AFTER activation of the environment
pip3 install matplotlib,scikit-image,numpy,jupyter-lab

#run code here
python3 whateverscript.py

# to leave the virtualenv
deactivate
```

Tasks for Python coding

E1.1

(Listen, for/while loops, mehrere Ausgaben)

Implementiere eine Funktion, welche eine Liste mit float werten und einen Floatwert u als Eingaben akzeptiert, und zwei Kopien der Liste ausgibt: einmal die Listenwerte mit dem Wert u addiert, und einmal Listenwerte mit dem Wert u subtrahiert.

Drucke das Ergebnis mit print aus.

E1.2

(Listen, for/while loops)

Implementiere eine Funktion, welche zwei Listen mit strings als Eingaben akzeptiert und eine Liste ausgibt, welche alle Kombinationen von Verkettungen von strings in der Form $f(l_0, l_1) = \{u + v, u \in l_0, v \in l_1\}$ enthaelt. D.h. jede Verkettung von strings, bei der der linke string aus der ersten liste stammt, und der rechte string aus der zweiten Liste.

Drucke das Ergebnis mit print aus.

E1.3

Initialisiere ein dictionary mit Paaren, welche aus einem zufaelligen string und zufaelligen einem Float-wert bestehen.
Schreibe eine Funktion, welche ein dictionary als Eingabe nimmt, und jeden value-wert eines key-value paares quadriert, falls dieser Wert eine float-Zahl ist, und dann das resultierende dictionary zurueckgibt.

Drucke das Ergebnis mit print aus.

Task 2: Numpy Indexing

Do the tasks in numpytemplate_v2.py

Task 3: some python recap

Implement selection sort without sorting libraries or sorting functions – to refresh your python coding skills.

Implement it as a function which takes a list or a numpy array as input. `def selsort(arr):` and returns a sorted version of it.

The idea of selection sort is:

- at step 0 find the max element over all N elements, swap it on the top-position
- at step 1 find the max element over the lower N-1 elements, swap it on the 2nd position
- at step 2 find the max element over the lower N-2 elements, swap it on the 3rd position
- iterate this

You can implement it using two nested for-loops.
Alternative: Implement insertion sort `https://en.wikipedia.org/wiki/Insertion_sort`

Task 4: Inner products

Norms

Compute the euclidean vector norm for vectors

$$[1, 0, 2],\ [3, 4],\ [-7, 2, -4, \sqrt{12}]$$

Angles

Compute the inner product between these vectors and their angle in degrees:

$$[3, -2, 2], [1, 2, -1.5]$$
$$[2, 0, 5, 1], [-1, 2, 0, 1]$$

Plotting

plot the vectors using matplotlib

$$[3, -2, 2], [1, 2, -1.5]$$

orthogonal vectors

Find a vector orthogonal to $(2, 5, -3)$.

Think: What properties has the set of all vectors orthogonal to that vector?

Task 5 (a bit harder): Some non-trivial python coding task

Learning goal: use python builtin operator interfaces (here `__lt__`, `__repr__` in PyTorch one often needs for Datasets `__iter__`, `__getitem__`, `__len__`).

Coding goal is: implement a data structure which tuple of two elements with a non-standard notion for the relation $<$.

Why do I ask you to implement a custom class which allows for $a < b$ comparisons instead of a custom sort? Reason is that by implementing $a < b$ you can use any sorting libraries which rely on $<$ / `__lt__` to sort.
You can use the file **sortedtest_students.py**.

The data structure which you will use contains two elements in an ordered way: (i, str).
The zero-th element of the tuple i is an integer, the first element str is a string of length 5.
We have for two instances a, b of that 2-element-tuple-like data structure that $a < b$ holds under these conditions:

$$(a < b) = True \text{ if } \begin{cases} & a[0] < b[0] \\ \text{or} & a[0] = b[0] \text{ and } a[1] \overset{!!}{>} b[1] \end{cases}$$

Looks artificial ?

Consider an example, where you determine elibility for bonuses for some sinister club members (the hotel in John Wick?), based on

- their miles, their rank, their birthday (here 1950 > 1990, old folks first), their social credit scoring (its a sinister club, so $A > B$ if A has a lower social credit score), their spending in the last year.

Then you may have a relation $>$ which is based on a sequence of criteria with tie-breaks.
Tasks:

- use the sorted() routine of python to sort. This requires an implementation of the interface `def __lt__(self,other):`. This function compares/computes the relation (self<other) and returns True or False.
- implement the interface `def __lt__(self,other):` for your custom tuple object according to the above rule for $<$
- for a nice `print(yourweirdtuple)` output, implement `def __repr__(self):`.

```
class weirdlysortedtuple():
  def __init__(self):
    #inits it as an empty tuple
    self.data = ()

  def __init__(self,el1,el2):
    pass
    #TODO set the tuple in self.data using el1 and el2

  #https://docs.python.org/3/reference/datamodel.html#object.__str__
  def __repr__(self):
    return 'nothing'
    #TODO change to return the __repr__() call of self.data

  def __lt__(self,other):
    pass
    #TODO
    return False
```

Hints: you can use

```
import random,string
    charlist = random.SystemRandom().choices(string.ascii_lowercase, k=strlen)
    charlist = random.choices(string.ascii_lowercase, k=strlen)
```

to create random strings of a certain length.

The first way is suitable for Cryptography (non-reproducibility!),
the second way uses the seed in random (reproducibility) and thus can be made reproducible. You can try out both to see the difference when a seed is provided.