Deep learning and AI methods

Session 1: First things in Python

7:04

- Instructor: Anastassia Baxevani, Krzysztof Podgorski, Statistics, Lund University, LUS
- For more information visit the CANVAS class website.

This session attempts to give you a very concise introduction to Python programming by showing its features through hands-on approach on designing some simple mathematical models. It also show nice capacities of Google Markdown Notebook in organizing reports on computational studies. Thus you will learn how to handle files and your computer work in COLAB. This file should also work in Jupiter Notebook, which is a more advanced markup tool, if you have both Python and Tensor Flow on your private computer.

Project 1

In this small project we will learn some of Python programming tools.

Here how one performs basic arithmetic operations:

```
a1=1; a2=2; #parameters
u1=5; u2=4; #arguments
f=[a1*u1*u2,a2*u1/u2]
print(f)
```

```
[20, 2.5]
```

```
#Another run of the same function with different arguments but with the same parameters
a1=1; a2=2;
u1=0.5; u2=0.3;
f=[a1*u1*u2,a2*u1/u2]
print(f)
```

```
[0.15, 3.3333333333333335]
```

```
# Here we will write the same function with different parameters but the same arguments as th
a1=3; a2=1;
u1=5; u2=4;
f=[a1*u1*u2,a2*u1/u2]
print(f)
```

```
[60, 1.25]
```

Another function

```
b1=4; b2=1;
v1=-2; v2=3;
g=[1/(1+b1*v1**2),1/(1+b2*v2**2)]
print(g)
```

```
[0.058823529411764705, 0.1]
```

7:04

```
#Similarily as before we change the arguments but keep the same parameters
b1=4; b2=1;
v1=-2.8; v2=43;
g=[1/(1+b1*v1**2),1/(1+b2*v2**2)]
print(g)
```

```
[0.030902348578491966, 0.0005405405405405405]
```

```
#Here we change the function itself using the same arguments and parameters as in the first c
b1=4; b2=1;
v1=-2; v2=3;
g=[1/(b1/v1**3),1/(b2/v2**3)]
print(g)
```

```
[-2.0, 27.0]
```

Comment:

The above simple functions were easy to test with different argument but one would prefer to have an option to evaluate many arguments at the same time. This can be done with vectors and matrices as shown next.

Matrices in Python

```
ua=[[5,4],[1,2]] # 2 vectors to make a matrix
print(ua)
print(ua[0][1]) # prints out the first and second point in the ua matrix.
```

```
[[5, 4], [1, 2]]
4
```

```
#experimenting with basic python features
ua=[[5,4],[1,2]]

print(ua)
```

```
print(ua[0][:]) # Only first row

print(ua[:][1]) #last row

print(ua[0][0],ua[1][1]) #diagonal

print([ua[0][0],ua[1][1]]) #diagonal as a matrix
```

```
[[5, 4], [1, 2]]
[5, 4]
[1, 2]
5 2
[5, 2]
```

```
# Here we create a longer matrix to help to understand and instead I will use pandas to creat
import pandas as pd
ua=pd.DataFrame([[5,4],[1,2], [2,3], [4,1], [6,6], [9,23]])

print(ua)
print(" ")
print(ua[0][:]) # Only first column
print(" ")

print(ua[5:]) #last row
print(" ")

print(ua[0][0],ua[1][1]) #diagonal
print(" ")

print([ua[0][0],ua[1][1]]) #diagonal as a matrix
```

```
   0   1
0  5   4
1  1   2
2  2   3
3  4   1
4  6   6
5  9  23

0    5
1    1
2    2
3    4
4    6
5    9
Name: 0, dtype: int64

   0   1
5  9  23

5 2

[5, 2]
```

Comment:

Working with matrices and vectors in python is very primitive and does not have many options. One needs to upload some fancier libraries.

7:04

NumPy library in Python

Most of the power of a programming language is in its libraries. A library is a collection of files (called modules) that contains functions for use by other programs. May also contain data values (e.g., numerical constants) and other things. Library's contents are supposed to be related, but there's no way to enforce that. The Python standard library is an extensive suite of modules that comes with Python itself. Many additional libraries are available from PyPI (the Python Package Index).

A library is a collection of modules, but the terms are often used interchangeably, especially since many libraries only consist of a single module, so don't worry if you mix them.

A program must import a library module before using it. Use import to load a library module into a program's memory. Then refer to things from the module as *module_name.thing_name*.

Python uses '.' to mean "part of".

- *NumPy* is the fundamental package (library, module) for scientific computing with Python. It contains among other things:
  - a powerful N-dimensional array object
  - sophisticated functions
  - useful linear algebra, Fourier transform, and random number capabilities

```
#Probably the most standard library
import numpy

print('pi is', numpy.pi)
print('cos(pi) is', numpy.cos(numpy.pi))
```

```
pi is 3.141592653589793
cos(pi) is -1.0
```

```
print('Euler constant is', numpy.e)
print('log(e) is', numpy.log(numpy.e))
```

```
Euler constant is 2.718281828459045
log(e) is 1.0
```

Import specific items from a library module to shorten programs. Use *from ... import ...* to load only specific items from a library module. Then refer to them directly without library name as prefix.

```
from numpy import cos, pi
print('cos(pi) is', cos(pi))
```

```
     cos(pi) is -1.0
```

7:04

```
from numpy import log, e
print('Euler constant is', e)
print('log(e) is', log(e))
```

```
     Euler constant is 2.718281828459045
     log(e) is 1.0
```

```
# Here I did the same as above but formatted it with only 2 decimal places
from numpy import log, e
print('Euler constant is {:.2f}'.format(e))
print('log(e) is', log(e))
```

```
     Euler constant is 2.72
     log(e) is 1.0
```

Create an alias for a library module when importing it to shorten programs. Use *import ... as ... to* give a library a short alias while importing it. Then refer to items in the library using that shortened name.

```
import numpy as np

print('cos(pi) is', np.cos(np.pi))
```

```
     cos(pi) is -1.0
```

```
import numpy as n

print('cos(pi) is', n.cos(n.pi))


print('sin(pi) is', n.sin(n.pi)) # Experimental code
```

```
     cos(pi) is -1.0
     sin(pi) is 1.2246467991473532e-16
```

Some more complicate operations using mathematical functions:

```
c1=1; c2=2;
w1=5; w2=4;
h=np.exp(-c1*w1-c2*w2)
print(h)
```

```
2.2603294069810542e-06
```

7:04

```
c1=1; c2=2;
w1=-2; w2=0.5;
h=[n.exp(-c1*w1-c2*w2),-c1*w1-c2*w2]
print(h)
```

```
[2.718281828459045, 1.0]
```

Comment:

One has to be careful how the name of functions from the uploaded libraries are used. There are several ways to shorten the name of functions but one has to be careful not to overide the existing functions by own functions with similar names.

Defining a function

A function is a way to capture code that is commonly executed. Consider the following function that can be used to trim white space from a string capitalize the first letter.

```
def process_string(str):
    t = str.strip()
    return t[0].upper()+t[1:]
```

This function can now be called quite easily.

```
str = process_string("  hello  ")
print(str)
```

```
 Hello
```

```
#Copying a string
def double_string(str):
    tt = str.strip()
    return tt[0:]+tt[0:]
```

This function can now be called equally easily.

```
str = double_string("Hello")

print(str)
welcome = "Welcome to Lund " # Here we experiment with different text and print it      th t
print(double_string(welcome))
```

```
     HelloHello
     Welcome to LundWelcome to Lund
```

```
#The newly defined functions can be superposed

str = double_string(process_string("   hello   "))
print(str)
```

```
     HelloHello
```

Functions related to Topic 2 in Discussion 1

Defining numerical functions and their evaluations.

```
def f(u,a):
    a1=a[0];a2=a[1];u1=u[0];u2=u[1]
    return [a1*u1*u2,a2*u1/u2]


print(ua)

u=ua[0];a=ua[1]
print(f(u,a))
```

```
        0   1
     0  5   4
     1  1   2
     2  2   3
     3  4   1
     4  6   6
     5  9  23
     [20, 10.0]
```

```
def g(v,b):
    b1=b[0];b2=b[1];v1=v[0];v2=v[1]
    return [1/(1+b1*v1**2),1/(1+b2*v2**2)]


v=[-1,6];b=[0.5,1]
```

```
print(g(v,b))
```

```
[0.6666666666666666, 0.02702702702702703]
```

```
vb=[[5,4],[1,2]]
```

7:04

```
v=vb[0];b=vb[1]
print(g(v,b))
```

```
[0.038461538461538464, 0.030303030303030304]
```

```
vb=[[10,5],[2,4]] # Here we experiment with new variables in the function g
```

```
v=vb[1];b=vb[1]
print(g(v,b))
```

```
[0.1111111111111111, 0.015384615384615385]
```

Comment:

One can utilize functions in Python to shorten programming complex mathematical formulas.

TASK 1:

After reading the first and third topics for *Discussion 1*, explain where we have in the code below parameters, where are there their estimators.

```
mu = [2, -3] # mu is a parameter representing the true mean

Sigma = [[1, 0], [0, 10]]  # diagonal covariance which is also a parameter

X= np.random.multivariate_normal(mu, Sigma, 500) # Randomly generated dataset

print(X.shape) # dimensions

print([ sum(X[:,0])/500 , sum(X[:,0])/500]) # manual average of the sample values

#even better

barX=np.mean(X, axis=0)  # Xbar represents the mean of sample X and is an estimator.

print(barX)

print(np.cov(np.transpose(X))) # need it to be 2x2 covariance matrix # Here using n-1, if did
# would do S = 499/500 *np.cov(no.transpose(X)) # This would give a better MLE convergence.
```

```
S=np.cov(np.transpose(X))  # S here represents the covariance array of X and is an estimator.
```

```
(500, 2)
[1.9486557301799459, 1.9486557301799459]
[ 1.94865573 -3.22883338]
[[ 1.00878478 -0.051298  ]
 [-0.051298    9.17026387]]
```

7:04

Solution:

In the code above we estimated 500 samples using the paramaters Sigma and mu to create random vectors. Additionally we calculated our estimators barX(the mean of X- estimation of mu) and S (The covariance matrix - The estimator of Sigma). Here we use Numpy.cov to estimate the covariance matrix, and Numpy's mean function to estimate the mean of X(barX).

Another code to analyze and comment on.

Task 2

How can the log-likelihood be evaluated? Look at the code below and comment what is done in it.

```
from scipy.stats import multivariate_normal

ll=np.log(multivariate_normal.pdf(X, mean=mu, cov=Sigma)) # Here we are computing the real mu

print(ll.shape) # dimension

y = np.sum(ll)
print(y)

hatll=np.log(multivariate_normal.pdf(X, mean=barX, cov=S))  # Here we are estimating the log
print(hatll.shape)

#It is smaller than the estimater, the estimator will be a little off from the real values as

haty = np.sum(hatll)
print(haty)
```

```
(500,)
-1977.0428697601033
(500,)
-1974.0455288455182
```

```
diff = y/haty
print(diff) # This is the percent difference of the real data and one using the estimators.
```

```
1.0015183747643035
```

7:04

Solution:

In the code above, we calculated the log-liklihood by taking the log of the multivariate norr probability density function(pdf) of every X given the real parameters (ll) and estimated parameters(hatll). Then we summed ll's together to create the joint probability density function(y). Lastly we compared the difference between the real sum of errors vs the estimated sum of errors. Here we see the estimate is close to the real value with only a 0.17% difference seen here with the different parameter.

Task 3

Evaluation of the likelihood using a function. Please, try to write your own function that evaluates the above likelihood. Demonstrate how it can be used to evaluate the values discussed above.

```
# write function to evaluate liklihood. non-trivial way to do so.
# Need to evaluate it but recreating values or enter other values.
from scipy.stats import multivariate_normal
def llGaussian(X,mu,Sigma):
  log_like = np.log(multivariate_normal.pdf(X, mean = mu, cov = Sigma))
  return np.sum(log_like)

print(llGaussian(X,mu,Sigma)) #Here we estimate the value of the likelood using the known par
print(llGaussian(X,barX,S))
```

```
-1977.0428697601033
-1974.0455288455182
```

Solution

Above we programmed a function that computes the Gaussian log-likelihood based on the previous code and using a function to do so. Here we defined the function as llGaussian, and use 3 arguments. At the end we returned the sum of the Gaussian log liklihood. When using a function you have flexibility by entering different arguments. In the above code I use both the actual parameters and estimators to calculate the log-likelihood. .

Conclusions to Project 1

We have used the python libraries to help compute the Gaussian log-Likelihood. This method is used to find the maximum values of the likelihoods. This was done through using the multivariate Gaussion distribution probability density function to calculate the joint log likelihood of bo wn parameters, and the estimators. At the end we used function to compute the Gaussian log Likelihood.

7:04

Grader box:

In what follows the grader will put the values according the following check list:

- 1 Have all commands included in a raw notebook been evaluated? (0 or 0.5pt)
- 2 Have all commands been experimented with? (0 or 0.5pt)
- 3 Have all experiments been briefly commented? (0 or 0.5pt)
- 4 Have all tasks been attempted? (0, 0.5, or 1pt)
- 5 How many of the tasks have been completed? (0, 0.5, or 1pt)
- 6 How many of the tasks (completed or not) have been commented? (0, 0.5, or 1pt)
- 7 Have been the conclusions from performing the tasks clearly stated? (0, 0.5, or 1pt)
- 8 Have been the overall organization of the submitted Lab notebook been neat and easy to follow by the grader? (0, or 0.5pt)

1 Have all commands included in a raw notebook been evaluated? (0 or 0.5pt)

```
Gr1=0
```

Grader's comment (if desired):

N/A

2 Have all commands been experimented with? (0 or 0.5pt)

```
Gr2=0
```

Grader's comment (if desired):

N/A

3 Have all experiments been briefly commented? (0 or 0.5pt)

```
Gr3=0
```

Grader's comment (if desired):

N/A

4 Have all tasks been attempted? (0, 0.5, or 1pt)

```
Gr4=0
```

Grader's comment (if desired):

N/A

5 How many of the tasks have been completed? (0, 0.5, or 1pt)

```
Gr5=0
```

Grader's comment (if desired):

N/A

6 How many of the tasks (completed or not) have been commented? (0, 0.5, or 1pt)

```
Gr6=0
```

Grader's comment (if desired):

N/A

7 Have been the conclusions from performing the tasks clearly stated? (0, 0.5, or 1pt)

```
Gr7=0
```

7:04

Grader's comment (if desired):

N/A

8 Have been the overall organization of the submitted Lab notebook been neat and easy to follow by the grader? (0, or 0.5pt)

7:04

```
Gr8=0
```

Grader's comment (if desired):

N/A

Overall score

```
Gr1+Gr2+Gr3+Gr4+Gr5+Gr6+Gr7+Gr8
```

```
0
```

Score and grader's comment (if desired):

N/A

✓ 0s completed at 1:36 PM

7:04