group2_unucf

November 16, 2020

[1]:
```
# Zeroth Exercise
```

0.0.1 Exercise 1: Matrices and Vectors in Python

As mentioned, we will use Python for the coding exercise. We will use version 3.6. Additionally, for this and all following exercise we will use NumPy, one of the most fundamental python libraries, designed to efficiently create and operate on multi-dimensional arrays (, i.e., vectors, matrices and higher order tensors)

1.1 Create Matrices and Vectors in Python We first create a matrix

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 2 & 6 \\ 7 & 3 & 8 \end{pmatrix}$$

and two vectors

$$v = \begin{pmatrix} 3 \\ 1 \\ 2 \end{pmatrix}, \; w = \begin{pmatrix} 1 \\ 1 \\ 2 \end{pmatrix}$$

[2]:
```
import numpy as np

# Let us create a matrix A and vectors v and w:
A = np.array([[1, 2, 3],
              [4 ,2, 6],
              [7, 3, 8]])
v = np.array([3, 1, 2])
w = np.array([1, 1, 2])
print("Matrix A:")
print(A)
print("Vector v:")
print(v)
print("Vector w:")
print(w)
```

```
Matrix A:
[[1 2 3]
 [4 2 6]
 [7 3 8]]
Vector v:
[3 1 2]
Vector w:
[1 1 2]
```

1.2 Basic Operations with Vectors and Matrices *1.2.1* Implement the v + w and 2 · A. Print out your results and verify that they are correct. *1.2.2* Do the matrix vector product A · v. Print out your result and verify that the result is correct. *1.2.3* Do the matrix matrix product A · A. Print out your result and verify that the result is correct.

Loesung 1.2 Implementation

[3]:
```
elem_wise_sum = v + w
print(f"result of v + m is {elem_wise_sum}")
```

```
result of v + m is [4 2 4]
```

[4]:
```
elem_wise_mult = 2*A
print(f"result of 2*A is \n {elem_wise_mult}")
```

```
result of 2*A is
 [[ 2  4  6]
 [ 8  4 12]
 [14  6 16]]
```

[5]:
```
matrix_vector_prod = np.dot(A, v)
print(f"the result of A matrix multipled with v is {matrix_vector_prod}")
```

```
the result of A matrix multipled with v is [11 26 40]
```

[6]:
```
# Two matrices can be multiplied in the same fashion:
matrix_matrix_prod = A@A
print(matrix_matrix_prod)
```

```
[[ 30  15  39]
 [ 54  30  72]
 [ 75  44 103]]
```

Loesung 1.2 Verification

$$\mathrm{v} + \mathrm{w} = \begin{bmatrix} 3 & 1 & 2 \end{bmatrix} + \begin{bmatrix} 1 & 1 & 2 \end{bmatrix} = \begin{bmatrix} 4 & 2 & 4 \end{bmatrix}$$

$$2^{*}\mathrm{A} = 2 * \begin{bmatrix} 1 & 2 & 3 \\ 4 & 2 & 6 \\ 7 & 3 & 8 \end{bmatrix} = \begin{bmatrix} 2 & 4 & 6 \\ 8 & 4 & 12 \\ 14 & 6 & 16 \end{bmatrix}$$

$$\text{A * v} = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 2 & 6 \\ 7 & 3 & 8 \end{bmatrix} * \begin{bmatrix} 3 \\ 1 \\ 2 \end{bmatrix} = \begin{bmatrix} 3+2+6 \\ 12+2+12 \\ 21+3+16 \end{bmatrix} = \begin{bmatrix} 11 \\ 26 \\ 40 \end{bmatrix}$$

$$\text{A*A} = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 2 & 6 \\ 7 & 3 & 8 \end{bmatrix} * \begin{bmatrix} 1 & 2 & 3 \\ 4 & 2 & 6 \\ 7 & 3 & 8 \end{bmatrix} = \begin{bmatrix} 1+8+21 & 2+4+9 & 3+12+24 \\ 4+8+42 & 8+4+18 & 12+12+48 \\ 7+12+56 & 14+6+24 & 21+18+64 \end{bmatrix} = \begin{bmatrix} 30 & 15 & 39 \\ 54 & 30 & 72 \\ 75 & 44 & 103 \end{bmatrix}$$ so the results are verified to be correct.

0.0.2 Exercise 2: Sampling from a Gaussian

Sample 1000 samples from a Gaussian distribution with mean $\mu = -2$ and standard deviation $\sigma = 0.5$.

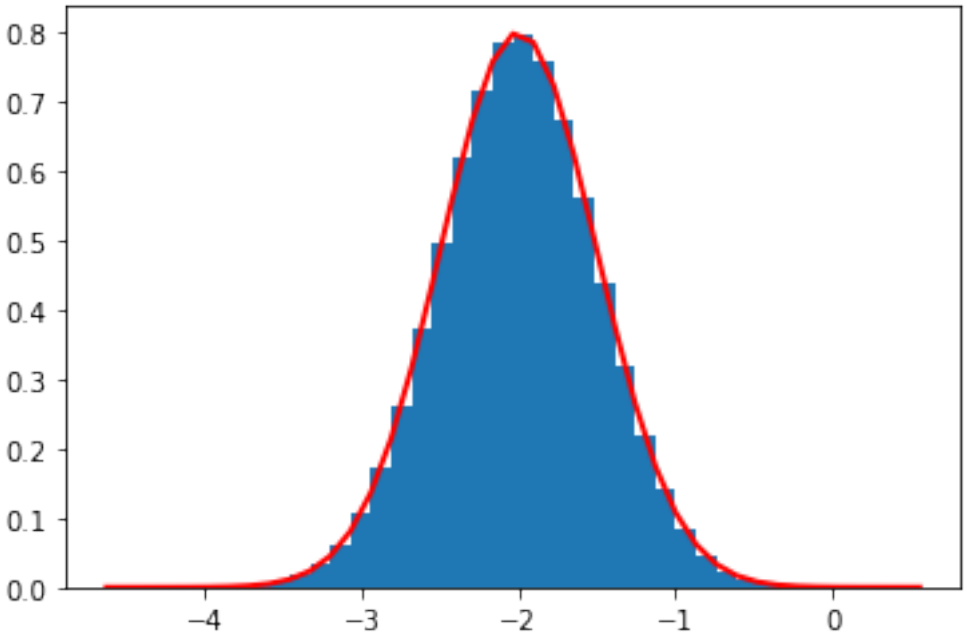
We also visualize the samples. For this we use matplotlib, a python package for plotting

[7]:
```
# for jupyter notebooks you need the following line of jupyter magic command␣
 ↪(that's literally the name)
# to display matplotlib figures inside the notebook. Make sure it comes before␣
 ↪the actual import!
%matplotlib inline
#actually import matplotlib
import matplotlib.pyplot as plt
from random import gauss
# In Machine Learning, it is often necessary to sample from a probability␣
 ↪distribution.
# To sample from a Gaussian distribution (https://en.wikipedia.org/wiki/
 ↪Normal_distribution):

mu, sigma = -2, 0.5 # mean and standard deviation
samples = [gauss(mu, sigma) for _ in range(10000000)]
# plot the histogram of the samples
count, bins, ignored = plt.hist(samples, 40, density=True)

# plot the density function
plt.plot(bins, 1/(sigma * np.sqrt(2 * np.pi)) * np.exp( - (bins - mu)**2 / (2 *␣
 ↪sigma**2) ),  linewidth=2, color='r')

# show the plot
plt.show()
```



0.0.3 Exercise 3: Solving a Linear Equation

You are given the linear equation

$$Ax = v.$$

We want to find the vector x. Assume that the matrix A and the vector v are given as in Exercise 1.1. Find the solution for x. Hint: Use the function 'linalg.solve' from numpy - this should always be preferred over direct matrix inversion.

[8]:
```
# To solve the linear equation system A x = v, you may use the np.linalg.solve␣
 ↪function. In other words we can
# use np.linalg.solve for inverting any matrix which does not stand alone.

x = np.linalg.solve(A, v) #solve for x here

print("Solution: x =", x)
print(f"Test: A@x = {A@x} is the same as v = {v}")
```

```
Solution: x = [-0.08333333  2.41666667 -0.58333333]
Test: A@x = [3. 1. 2.] is the same as v = [3 1 2]
```

0.1 Closing Remarks

NumPy provides easy and efficient ways to deal with vectors, matrices and tensors. Linear algebra functionality like the function introduced above, together with more advanced techniques such as broadcasting (https://numpy.org/doc/stable/user/basics.broadcasting.html), are the key

to efficient programming in Python. If you want to do any practical Machine Learning you need to familiarize yourself with those techniques. We will use and introduce more of this in the following exercises but also expect you to use it for your submission. So use them whenever possible and, most importantly, avoid unnecessary for loops.