Physics 77/88 - Fall 2024 - Homework 1

Unix and Python

Submit this notebook to bCourses to receive a credit for this assignment.
due: **Sept 11 2024**
Please upload both, the .ipynb file and the corresponding .pdf

Problem 1 (4P)

Imagine you have to import many **8 bit** color images (e. g. RGB) for training an ANN.

a) How many different colors can be saved theoreticaly in each of the images? Explain your answer. (2P)
b) Do the same for a **16 bit** image. (2P)

a) $2^8 = 256$ different colors can be represented because each on/off state of every bit can save a different color.
b) With similar reasoning to the 8 bit image, the 16 bit image can represent $2^{16} = 65536$ different colors.

Problem 2 (2P)

For many algorithms, you need to calculate the product P_{tot} of different **probabilities** P_i like eg.

$$P_{tot} = \Pi_i^I \, P_i$$

for large I.

a) Why could this be a problem? (1P)
b) How can you solve the problem? (1P)

a) Because the total probability gets multiplicatively smaller , for large I, the total probability would become too small to accurately represent by a given number of bits.
b) This problem can be solved by adding more bits to store the number (such as using a double) so that smaller quantities can be more accurately stored. Alternatively, by taking the logarithm of both sides, the product can be calculated instead as a sum. By then exponentiating, the total probability can be calculated without need for more bits.

Problem 3 (4P)

Write down the following numbers as binary and with base 3 (including derivation):

a) 21 (2P)

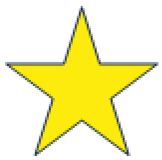
b) 27 (2P)

a) (i) Binary: $(1*2^4)+(0*2^3)+(1*2^2)+(0*2^1)+(1*2^0)=10101$

a) (ii) Base 3: $(2*3^2)+(1*3^1)+(0*3^0)=210$

b) (i) Binary: $(1*2^4)+(1*2^3)+(0*2^2)+(1*2^1)+(1*2^0)=11011$

b) (ii) Base 3: $(1*3^3)+(0*3^2)+(0*3^1)+(0*3^0)=1000$



Problem 4 - optional (4P)

Create an arbitrary list *L1* in Python. From *L1* create another list *L2* that lists only those properties of *L1* which are **dunder methods**.

In [1]:
```
L1 = []
L2 = []
L = dir(L1)
for i in L:
    if i[0:2] == '__' and i[-2:] == '__':
        L2 += [i]
print(L2)
```

```
['__add__', '__class__', '__class_getitem__', '__contains__', '__delattr__',
'__delitem__', '__dir__', '__doc__', '__eq__', '__format__', '__ge__', '__ge
tattribute__', '__getitem__', '__getstate__', '__gt__', '__hash__', '__iadd_
_', '__imul__', '__init__', '__init_subclass__', '__iter__', '__le__', '__le
n__', '__lt__', '__mul__', '__ne__', '__new__', '__reduce__', '__reduce_ex_
_', '__repr__', '__reversed__', '__rmul__', '__setattr__', '__setitem__', '_
_sizeof__', '__str__', '__subclasshook__']
```

Problem 5 (2P)

After a curve fit, a programm returns a vector *V* containing the errors, e. g.:

In [2]:
```
import numpy as np

V = np.array([0.3, 0.01, 0.2, 0.121, 0.11])
print(V)
```

```
[0.3   0.01  0.2   0.121 0.11 ]
```

How would you calculate the mean of the squared errors (MSE) most efficiently in Python?

In [3]:
```
MSE = np.mean(V**2)
print(MSE)
```

```
0.0313682
```