EN 001203 Artificial Neural Networks

E002: Numerical Programming and Optimization

Faculty of Engineering, Khon Kaen University

Submission: https://autolab.en.kku.ac.th

==

* Submit an answer to a question with a file with `txt` extension. E.g., an answer for Q1 should be submitted in a text file "`Q1.txt`"

* Submit a program to a (programming) problem with a file with a proper extension. E.g., a python program for P2 should be named "`P2.py`"

* All answers and programs must be packaged together in a tar file.

* Each question or problem is worth 60 points. There are 5 problems, thus 300 points are counted as a total score for this exercise.

* Floating-point numbers are graded using tolerance 0.001.

==

"If we are facing in the right direction,

all we have to do is keep on walking."

-- Joseph Goldstein

P1. Dot product. Write function `sop` with interface `sop(w, u)`, where `w` and `u` are supposed to be Numpy arrays. The function returns a sum of products of both arguments. That is, the sop function returns $\sum_{i=0}^{d} w_i \cdot x_i$, when $w_0, \ldots, w_d$ are elements of `w` and $x_0, \ldots, x_d$ are elements of `x`.

[Hint: it is easier with Numpy dot.]

Given the function is called as follows

```
import numpy as np
from P1 import sop

if __name__ == "__main__":

    x = np.array([1, 2, 3, 4])
    w = np.array([-1, 1, 0, 2])
    print("w dot x =", sop(w, x))
```

output example will look like:

```
w dot x = 9
```

You may use a starter code below.

Starter code:

```
"""
P1. Online averaging.
Student name: Tenaciti Kurios
"""

import numpy as np

def sop(w, x):
    # WRITE YOUR CODE HERE
    return 0


if __name__ == "__main__":
    # Test your code sufficiently!
    d = np.array([1, 2, 3, 4])
    v = np.array([-1, 0, 1, 2])

    print("dot product =", sop(d, v))
```

P2. Loss and gradient. Given a loss function $g(u) = \frac{1}{1+\exp(-8u)} - \frac{1}{1+\exp(-u)}$, write a loss function with interface `loss(u)` and its gradient function with interface `grad(u)`.

[Hint: you need a chain rule of derivative; recall the quotient rule of derivative.]

Given the functions are called as follows

```
import numpy as np

from P2 import loss, grad

if __name__ == "__main__":
    us = np.linspace(-1, 0, num=5)
    print(us)
    print(loss(us))
    print(grad(us))
```

output example may look like:

```
[-1.   -0.75 -0.5  -0.25  0.  ]
[-0.26860607 -0.31834868 -0.35955446 -0.31862058  0.    ]
[-0.19393003 -0.19816292 -0.09370206  0.5938146   1.75  ]
```

P3. Minimization with Gradient Descend. Given the problem formulation:

$$min_u \, g(u)$$

`where` $g(u) = \frac{1}{1+\exp(-8\,u)} - \frac{1}{1+\mathrm{ex}\ (-u)}$, write a function with interface `gmin(u0, lr, N)` to find the minimizer u^* using gradient descend algorithm. The function takes in values of an initial value of the control variable as `u0`, step size `lr` and a number of iterations `N`. The function returns a value of the minimizer found in the process.

[Hint: test it out, check the optimizing progress, play around and test it with various sets of arguments; students are encouraged to use visualization to gain understanding as much as possible.]

Given the functions are called as follows

```
from P3 import gmin

if __name__ == "__main__":
    ux = gmin(u0=0, lr=0.2, N=10)
    print(ux)
```

output example may look like:

```
-0.430047280075
```

P4. Maximization with Gradient Descend. Given the problem formulation:

$$max_u \, g(u)$$

`where` $g(u) = 2 + 3\,u - u^2$, write a function with interface `gmax (u0, lr, N)` to find the maximizer u^* using gradient descend algorithm. The function takes in values of an initial value of the control variable as `u0`, step size `lr` and a number of iterations `N`. The function returns a value of the maximizer found in the process.

[Hint: it is MAXIMIZATION! students are encouraged to use visualization to gain understanding as much as possible.]

Given the functions are called as follows

```
from P4 import gmax

if __name__ == "__main__":
    ux = gmax(u0=0, lr=0.2, N=10)
    print(ux)
```

output example may look like:

```
1.4909300736
```

P5. Two-Dimensional Problem with Gradient Descend. Given the problem formulation:

$$min_{\vec{u}}\, g(\vec{u})$$

where $\vec{u} = [u_1, u_2]$ and $g(\vec{u}) = u_1^2 + 1.5u_1u_2 - 16u_1 + 2u_2^2 - 35u_2$, write a function with interface `g2min(u0, lr, N)` to find the minimizer using gradient descend algorithm. The function takes in values of an initial value of the control variable `u0` as a numpy array of shape (2,1), step size `lr` and a number of iterations `N`. The function returns a value of the minimizer found in the process.

[Hint: always check the progress with losses over iteration steps.]

Given the functions are called as follows

```
import numpy as np
from P5 import g2min

if __name__ == "__main__":
    u0= np.array([[0],[0]])
    ux = g2min(u0, lr=0.1, N=50)
    print("u*=", ux)
```

output example may look like:

```
u*= [[ 2.00301865]
 [ 7.99838447]]
```