

13/15 points

Exercise1_TeamC

April 30, 2018

1 Exercise Sheet 1: Neurons

```
In [3]: import numpy as np
        from scipy import constants
        from matplotlib import pyplot as plt
```

1.1 Task 1: The logistic function (1 point - programming)

1/1

Write a function “logistic(h,a=1)” resembling the logistic function and a second function “signtrafunc” the more discrete transfer function equivalent in form of a function of the sign of the input. Also write a function “dlogistic(h,a=1)” resembling the derivative of the logistic function.

For defining the sign-based transfer function the inline definition of functions using the syntax `trafunc=lambda x: f(x)` might be handy, please write the logistic function, however, as a classical function definition. The `a=1` in the function definition sets the default to `a=1` so the function needs only 1 argument by default.

```
In [4]: import math

def logistic(h,a=1):
    returnVal = 1 / (1 + np.exp(-a * h))
    return returnVal

def dlogistic(h,a=1):
    return (a * np.exp(-a * h)) / (1 + np.exp(-a * h))**2

signtrafunc=lambda x: (np.sign(x) + 1)/2

# x = np.arange(-6.0, 6.0, 0.1)
# plt.plot(x, logistic(x))
# plt.plot(x, dlogistic(x))
# plt.plot(x, signtrafunc(x))
# plt.show()
```

1.2 Task 2: Building an artificial neuron (2 points - programming)

1/2

Define a class “neuron” resembling a simple perceptron. See <https://docs.python.org/3/tutorial/classes.html> for help on classes.

It should have the attributes “neuron.w” for the weights, “neuron.b” for the bias, the methods “neuron.trafunc(self,x)” and “neuron.dtrafunc(self,x)” for the transfer function and the method “neuron.out(self,x)” to calculate the output for input x.

Build a constructor method “_init_(self,...)”, where self refers to the object itself, which assigns the weight vector x , the bias b , the transfer function “trafunc” and it’s derivative “dtrafunc” to the neuron.

In [38]: `class neuron:`

```
def __init__(self, weight, bias):
    self.w = weight
    self.b = bias
```

```
def trafunc(self, h, logisticBool):
    if(logisticBool):
        return logistic(h)
    else:
        return signtrafunc(h)

def dtrafunc(self, h, logisticBool):
    if(logisticBool):
        return dlogistic(h)
    else:
        return 0
```

```
# I think the derivation of the sign function is basically 0
# , or undefined/infinite on the 0 mark
```

```
# return the "h" for the f(h) <- transfer function
def out(self, x):
    return(np.dot(self.w, x) - self.b)
```

DanielMiklody 30.04.2018 16:01:36
Notiz

methods can be assigned dynamically in the constructor in Python (see sample solution). This makes the neuron class more flexible.

-0.5

DanielMiklody 30.04.2018 16:04:34
Notiz

The out function was supposed to give the output of the neuron for a certain input (so $f(wx-b)$ involving the trafunc $f(h)$)

-0.5

1.3 Task 3: Use an artificial neuron (3 points - thinking and programming)

2.5/3

Use instances of the neuron class above to build the equivalents to logical “or” and “and”-functions and test them for 2-dimensional input vectors x resembling all possibilities of combinations ([0,0] [1,0], [0,1], [1,1]).

First, use the sign-based transfer function to solve the task and then apply the same weights w and bias b on a neuron with the logistic transfer function.

What’s the difference?

Hint: if you haven’t succeeded with Task 2, you can solve Task 3 analytically and write down the needed weights and results.

```
In [39]: # AND Gate with Sign function
print("AND Gate with Sign function")
neuron1 = neuron([20, 20], 25)
x = np.array([[0,0], [1,0], [0,1], [1,1]]) #input 2d vectors
print(neuron1.trafunc(neuron1.out(x.T), False))
```

```

# OR Gate with Sign function
print("\nOR Gate with Sign function")
neuron2 = neuron([30, 30], 10)
print(neuron2.trafunc(neuron2.out(x.T), False))

# AND Gate with Logistic function
print("\nAND Gate with Logistic function")
resultAnd = neuron1.trafunc(neuron1.out(x.T), True)
resultAnd[resultAnd > 0.5] = 1.0
resultAnd[resultAnd <= 0.5] = 0.0
print(resultAnd)

# OR Gate with Sign function
print("\nOR Gate with Logistic function")
resultOr = neuron2.trafunc(neuron2.out(x.T), True)
resultOr[resultOr > 0.5] = 1.0
resultOr[resultOr <= 0.5] = 0.0
print(resultOr)

```

DanielMiklody
Notiz

30.04.2018 17:30:27
it was meant to interpret the output without threshold...



-0.5

AND Gate with Sign function
[0. 0. 0. 1.]

OR Gate with Sign function
[0. 1. 1. 1.]

AND Gate with Logistic function
[0. 0. 0. 1.]

OR Gate with Logistic function
[0. 1. 1. 1.]

1.4 Task 4: The XOR function (100 extra points - thinking and maybe programming)

+1

Build an XOR function with a neuron of the class you have created.

In [24]: print("doesn't make sense, because how can I make the sum of the weight * input to be
doesn't make sense, because how can I make the sum of the weight * input to be less than the b.

1.5 Task 5: The current density (4 points - analytic derivation)

3.5/4

Analytically derive the current density $\vec{J} = -\sigma \nabla \phi$ of a dipole field at origin step-by-step from the potential equation $\phi(\vec{r}) = \frac{1}{4\pi\epsilon_0} \frac{\vec{p} \cdot \vec{r}}{|\vec{r}|^3}$ in Cartesian coordinates.

To this extent you will need to build the gradient $\nabla = \begin{pmatrix} \frac{\delta}{\delta x} \\ \frac{\delta}{\delta y} \\ \frac{\delta}{\delta z} \end{pmatrix}!$

You can take the dipolar moment \vec{p} to be a constant vector $\vec{p} = \begin{pmatrix} p_x \\ p_y \\ p_z \end{pmatrix}$.

The vector $\vec{r} = \begin{pmatrix} x \\ y \\ z \end{pmatrix}$ is the coordinates in cartesian coordinates and the norm is the euclidean norm $|\vec{r}| = \sqrt{x^2 + y^2 + z^2}$.

σ is the specific conductivity of the material (a constant).

Hint: You can hand in the solution in your format of choice: Write directly below (Markdown takes latex within \$\$, jsut double-click on the cell below), Pdf from Latex or Word, scan or photo of a hand-written derivation, etc....

1.5.1 Solution:

$$\vec{J} = -\sigma \nabla \phi =$$

```
In [21]: from IPython.display import Image
         Image(filename='task5p1.jpeg')
```

```
Out[21]:
```

task 5

$$\vec{p} = \begin{pmatrix} p_x \\ p_y \\ p_z \end{pmatrix} \text{ constant}$$

$$\vec{r} = \begin{pmatrix} x \\ y \\ z \end{pmatrix}$$

$$|\vec{r}| = \sqrt{x^2 + y^2 + z^2}$$

$$\vec{J} = \sigma \begin{pmatrix} \frac{d\phi}{dx}(\phi(\vec{r})) \\ \frac{d\phi}{dy}(\phi(\vec{r})) \\ \frac{d\phi}{dz}(\phi(\vec{r})) \end{pmatrix}$$

$$\phi(\vec{r}) = \left(\frac{1}{4\pi\epsilon_0} \right) \left(\frac{\left(\begin{pmatrix} p_x \\ p_y \\ p_z \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ z \end{pmatrix} \right)}{(x^2 + y^2 + z^2)^{\frac{3}{2}}} \right)$$

$$= \left(\frac{1}{4\pi\epsilon_0} \right) \left(\frac{x \cdot p_x + y \cdot p_y + z \cdot p_z}{(x^2 + y^2 + z^2)^{\frac{3}{2}}} \right)$$

$$\frac{d\phi}{dx} = 0 + \left(\frac{1}{4\pi\epsilon_0} \right) \left(V' \right)$$

$$V' = \frac{(x^2 + y^2 + z^2)^{\frac{3}{2}}(p_x) - (x p_x + y p_y + z p_z) \left(\frac{3}{2} \right) (x^2 + y^2 + z^2)^{\frac{1}{2}} (2x)}{(x^2 + y^2 + z^2)^5}$$

$$V' = \frac{(p_x)(x^2 + y^2 + z^2)^{\frac{3}{2}}}{(x^2 + y^2 + z^2)^5} - \frac{(3x)(x p_x + y p_y + z p_z)(x^2 + y^2 + z^2)^{\frac{1}{2}}}{(x^2 + y^2 + z^2)^5}$$

$$= \frac{p_x}{(x^2 + y^2 + z^2)^{\frac{5}{2}}} - \frac{(3x)(x p_x + y p_y + z p_z)}{(x^2 + y^2 + z^2)^{\frac{5}{2}}}$$

In [22]: `from IPython.display import Image`
`Image(filename='task5p2.jpeg')`

Out [22]:

$$\begin{aligned}
 \frac{d\phi}{dx} &= \frac{(p_x)(x^2+y^2+z^2)^{\frac{3}{2}}}{(x^2+y^2+z^2)^3} - \frac{(3x)(x \cdot p_x + y \cdot p_y + z \cdot p_z)(x^2+y^2+z^2)^{\frac{1}{2}}}{(x^2+y^2+z^2)^3} \\
 &= \frac{p_x}{(\sqrt{x^2+y^2+z^2})^3} - \frac{3x \left(\frac{x}{2}\right) \left(\frac{p_x}{p_0}\right) (x^2+y^2+z^2)^{\frac{1}{2}}}{(x^2+y^2+z^2)^3} \quad \left| \begin{array}{l} \vec{p} = \begin{pmatrix} p_x \\ p_y \\ p_z \end{pmatrix} \\ \vec{r} = \begin{pmatrix} x \\ y \\ z \end{pmatrix} \\ |\vec{r}| = \sqrt{x^2+y^2+z^2} \end{array} \right. \\
 &= \frac{p_x}{|\vec{r}-\vec{r}_0|^3} - \frac{3x(\vec{r}-\vec{r}_0)(\vec{p})}{(x^2+y^2+z^2)^{\frac{5}{2}}} \\
 &= \frac{p_x}{|\vec{r}-\vec{r}_0|^3} - \frac{3x(\vec{r}-\vec{r}_0)(\vec{p})}{(x^2+y^2+z^2)^{\frac{5}{2}}} \\
 \frac{d\phi}{dx} &= \frac{p_x}{|\vec{r}-\vec{r}_0|^3} - 3 \cdot \frac{(\vec{r}-\vec{r}_0)(\vec{p})}{|\vec{r}-\vec{r}_0|^5} \cdot x \\
 \frac{d\phi}{d\left(\frac{x}{2}\right)} &= \frac{\left(\frac{p_x}{2}\right)}{|\vec{r}-\vec{r}_0|^3} - 3 \cdot \frac{(\vec{r}-\vec{r}_0)(\vec{p})}{|\vec{r}-\vec{r}_0|^5} \cdot \left(\frac{x}{2}\right) \\
 \vec{E} &= -\frac{1}{4\pi\epsilon_0} \nabla \phi = \frac{\vec{p}}{|\vec{r}-\vec{r}_0|^3} - 3 \cdot \frac{(\vec{r}-\vec{r}_0)(\vec{p})}{|\vec{r}-\vec{r}_0|^5} (\vec{r}-\vec{r}_0)
 \end{aligned}$$

-0.5

J=sigma E

DanielMiklody

30.04.2018 16:08:08

Notiz

missing the final transformation to J from the derivative

1.6 Task 6: Electrical Field strength & potential of a dipole (5 points - programming) 4/5

Write a python function resembling the electrical field potential $\phi(\vec{r}, \vec{r}_0)$ and another one for the field strength \vec{E} of a dipole in infinite charge free space at point \vec{r}_0 in Cartesian coordinates. The function should take the position of the observed point \vec{r} , the position of the dipole \vec{r}_0 and the dipole moment \vec{p} as inputs and return the scalar potential. \vec{r}_0 should be set to the origin by default.

$$\phi(\vec{r}) = \frac{1}{4\pi\epsilon_0} \frac{\vec{p} \cdot (\vec{r} - \vec{r}_0)}{|\vec{r} - \vec{r}_0|^3}$$

$$\vec{E} = \frac{1}{4\pi\epsilon_0} \left(3 \frac{(\vec{r} - \vec{r}_0) \cdot \vec{p}}{|\vec{r} - \vec{r}_0|^5} \cdot (\vec{r} - \vec{r}_0) - \frac{\vec{p}}{|\vec{r} - \vec{r}_0|^3} \right)$$

Make a plot of the two variables in two dimensions from -10 to 10 for both axes in the x/y-plane for a dipole at origin with dipole moment $\vec{p} = [1, 0]$. Plot the potential as decibel values using the matplotlib function `plt.contour/contourf` and the field strength using the function `plt.streamplot`. Draw both plots into the same figure (`plt.hold('on')`). The arguments to the plot function call for \vec{E} - and ϕ - values need to have the shape corresponding to the XX and YY variables from `np.meshgrid` (see below).

Hint: You can find ϵ_0 in the scipy constants. Numpy's `tensordot` might simplify the treatment of multiple positions at once.

```
In [6]: x=np.arange(-10,10,0.1)
        y=np.arange(-10,10,0.1)

        XX,YY=np.meshgrid(x,y)

        p = np.array([1,0]) #dipole moment
        r = np.array([XX,YY])

        def phi_dip(r, p, r_0 = np.zeros((2,200,200))):
            k = 1 / 4 * constants.pi * constants.epsilon_0
            return k * np.tensordot(p,r-r_0, axes = 1) / (np.linalg.norm(r-r_0,axis = 0))**3

        def E_dip (r,p,r_0 = np.array([0,0])):
            k = 1 / 4 * constants.pi * constants.epsilon_0
            div = (np.linalg.norm(r-r_0[:,np.newaxis, np.newaxis],axis = 0))**3

            h = np.zeros((2,div.shape[0],div.shape[1]))
            h[0,:,:] = p[0] / div
            h[1,:,:] = p[1] / div
            return k* (3 * np.tensordot(p,r-r_0[:,np.newaxis, np.newaxis], axes = 1) / (np.lin:

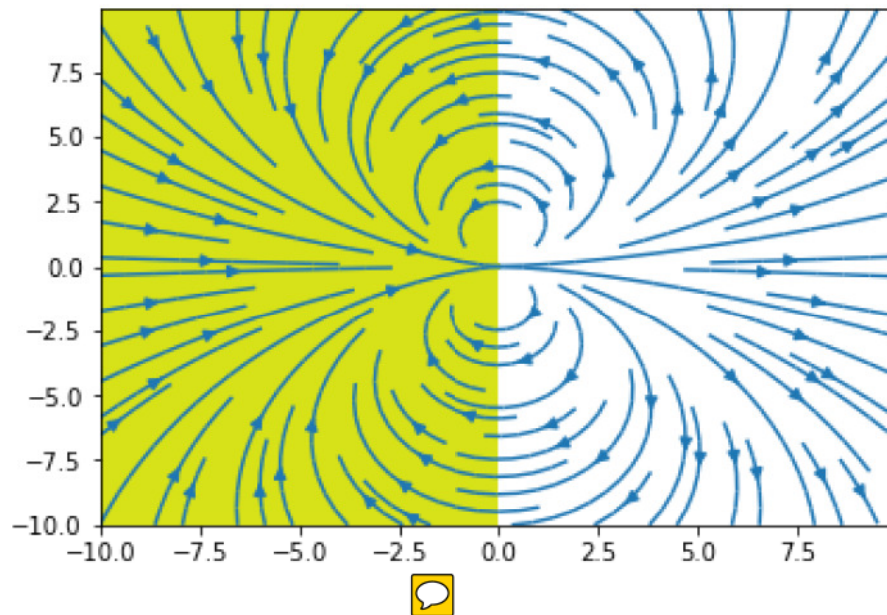
        #Calculate the field and potential on the whole meshgrid
        phi=phi_dip(r,p)

        E=E_dip(r,p)

        plt.contourf(XX, YY, phi)

        plt.streamplot(x, y, E[0], E[1])

Out [6]: <matplotlib.streamplot.StreamplotSet at 0x11328f208>
```



DanielMiklody 30.04.2018 16:09:55
Notiz

```
In [243]: # x=np.arange(-10,10,0.1)
# y=np.arange(-10,10,0.1)

# XX,YY=np.meshgrid(x,y)

# p = np.array([1,0]) #dipole moment
# r = np.array([XX,YY])

# def phi_dip(r, p, r_0 = np.zeros((2,200,200))):
#     k = 1 / 4 * constants.pi * constants.epsilon_0
#     return k * np.tensordot(p,r-r_0, axes = 1) / (np.linalg.norm(r-r_0,axis = 0))*

# phi=phi_dip(r,p)

# print(phi)
```