

Exercise1_Solution

November 17, 2020

1 Exercise Sheet 1: Neurons

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

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

Write a function “logistic(h,a=1)” resembling the logistic function and a second function “signtrafunc” the binary (sign-based) transfer function. Also write a function “dlogistic(h,a=1)” resembling the derivative of the logistic function. Similarly, write a “gaussRBF(h,a=1)” resembling the gaussian Radial Basis Function and “dgaussRBF(h,a=1)” its derivative.

Plot each of them in the range of -10 to 10 together with their derivative (except for the sign-based function) with matplotlibs plot function (plt.plot)).

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.

```
[2]: def logistic(h,a=1):
      return 1/(1+np.exp(-a*h))

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

      def dlogistic(h,a=1):
          return a*(logistic(x,a)-np.power(logistic(x,a),2))

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

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

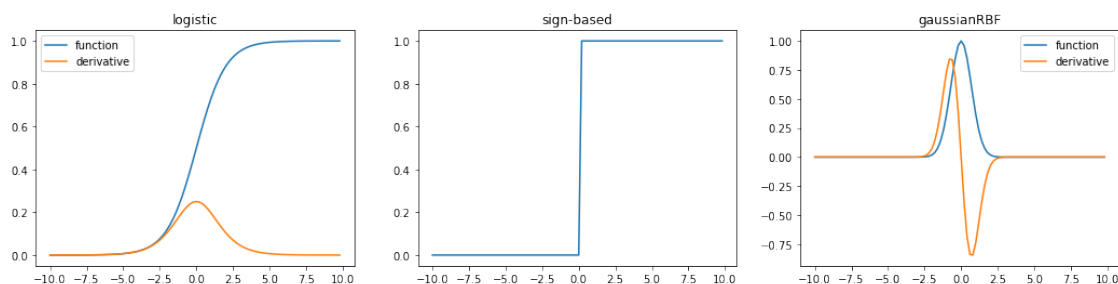
      x=np.arange(-10,10,0.2)

      plt.figure(figsize=[18,4])
      plt.subplot(131)
      plt.title('logistic')
```

```

plt.plot(x,logistic(x))
plt.plot(x,dlogistic(x))
plt.legend(['function','derivative'])
plt.subplot(132)
plt.title('sign-based')
plt.plot(x,signtrafunc(x))
plt.subplot(133)
plt.title('gaussianRBF')
plt.plot(x,gaussRBF(x))
plt.plot(x,dgaussRBF(x))
plt.legend(['function','derivative'])
plt.show()

```



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

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,h)” for the transfer function, “neuron.dtrafunc(self,h)” for the first derivative of the transfer function and the method “neuron.out(self,x)” to calculate the output for inputs x and h respectively.

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

The transfer function and its derivative should best be assigned at construction in order to account for different possible transfer functions. You do not need to set a transfer function, yet, but you may implement the logistic function as the default transfer function.

```

[3]: class neuron:
    def __init__(self,w,b,trafunc=logistic,dtrafunc=dlogistic):
        self.w=np.array(w)
        self.b=np.array(b)
        self.trafunc=trafunc
        self.dtrafunc=dtrafunc

```

```
def out(self, x):
    return self.trafunc(np.dot(self.w,x)-self.b)
```

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

Use instances of the neuron class above to build the equivalents to logical “or” and “and”-functions using the binary (sign-based) transfer function and test them for 2-dimensional input vectors x resembling all possibilities of combinations $([0,0], [1,0], [0,1], [1,1])$ by setting the weights and bias manually.

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

What’s the difference?

We do not need the derivatives, here, so they may be set to anything. In particular, the sign-based function does not have a derivative.

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

```
[4]: ## OR function
w=np.array([1 ,1])
b=0.5

neuron1=neuron(w,b,signtrafunc)

print('Ouput of the OR function with sign-based function for different inputs:')
print(' [0,0]-->'+str(neuron1.out([0,0])))
print(' [1,0]-->'+str(neuron1.out([1,0])))
print(' [0,1]-->'+str(neuron1.out([0,1])))
print(' [1,1]-->'+str(neuron1.out([1,1])))

## OR function logistic
w=np.array([1 ,1])
b=0.5

neuron1=neuron(w,b)

print('Ouput of the OR function with logistic transfer function for different_
->inputs:')
print(' [0,0]-->'+str(neuron1.out([0,0])))
print(' [1,0]-->'+str(neuron1.out([1,0])))
print(' [0,1]-->'+str(neuron1.out([0,1])))
print(' [1,1]-->'+str(neuron1.out([1,1])))

## OR function gaussRBF
w=np.array([1 ,1])
```

```

b=0.5

neuron1=neuron(w,b,gaussRBF)

print('Ouput of the OR function with gaussian RBF transfer function for_
↳different inputs:')
print(' [0,0]-->'+str(neuron1.out([0,0])))
print(' [1,0]-->'+str(neuron1.out([1,0])))
print(' [0,1]-->'+str(neuron1.out([0,1])))
print(' [1,1]-->'+str(neuron1.out([1,1])))

## AND function
w=np.array([1 ,1])
b=1.5
neuron2=neuron(w,b,signtrafunc)

print('Ouput of the AND function with sign-based function for different inputs:
↳')
print(' [0,0]-->'+str(neuron2.out([0,0])))
print(' [1,0]-->'+str(neuron2.out([1,0])))
print(' [0,1]-->'+str(neuron2.out([0,1])))
print(' [1,1]-->'+str(neuron2.out([1,1])))

## AND function logistic
neuron2=neuron(w,b)

print('Ouput of the AND function with logistic transfer function for different_
↳inputs:')
print(' [0,0]-->'+str(neuron2.out([0,0])))
print(' [1,0]-->'+str(neuron2.out([1,0])))
print(' [0,1]-->'+str(neuron2.out([0,1])))
print(' [1,1]-->'+str(neuron2.out([1,1])))

## AND function gaussRBF
neuron2=neuron(w,b,gaussRBF)

print('Ouput of the AND function with gaussian RBF transfer function for_
↳different inputs:')
print(' [0,0]-->'+str(neuron2.out([0,0])))
print(' [1,0]-->'+str(neuron2.out([1,0])))
print(' [0,1]-->'+str(neuron2.out([0,1])))
print(' [1,1]-->'+str(neuron2.out([1,1])))

```

Ouput of the OR function with sign-based function for different inputs:

```

[0,0]-->0.0
[1,0]-->1.0

```

```

[0,1]-->1.0
[1,1]-->1.0
Ouput of the OR function with logistic transfer function for different inputs:
[0,0]-->0.3775406687981454
[1,0]-->0.6224593312018546
[0,1]-->0.6224593312018546
[1,1]-->0.8175744761936437
Ouput of the OR function with gaussian RBF transfer function for different
inputs:
[0,0]-->0.7788007830714049
[1,0]-->0.7788007830714049
[0,1]-->0.7788007830714049
[1,1]-->0.10539922456186433
Ouput of the AND function with sign-based function for different inputs:
[0,0]-->0.0
[1,0]-->0.0
[0,1]-->0.0
[1,1]-->1.0
Ouput of the AND function with logistic transfer function for different inputs:
[0,0]-->0.18242552380635635
[1,0]-->0.3775406687981454
[0,1]-->0.3775406687981454
[1,1]-->0.6224593312018546
Ouput of the AND function with gaussian RBF transfer function for different
inputs:
[0,0]-->0.10539922456186433
[1,0]-->0.7788007830714049
[0,1]-->0.7788007830714049
[1,1]-->0.7788007830714049

```

Answer: While the logistic function delivers a smooth propagation between different outputs, the radial basis function additionally changes the order of the results and is also capable of inverting them. In combination with an inner product as commonly used with logistic transfer function it serves as a dissimilarity measure: the higher the ‘covariance’ the lower the output. RBFs in Neural Networks are thus commonly used in combination with a distance measure instead of the inner product.

1.4 Task 4: The electrical field strength (4 points - analytic derivation)

Analytically derive the electrical field strength $\vec{E} = -\nabla\phi$ of an electrostatic dipole field at origin in an infinite homogeneous otherwise charge-free space 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 $\nabla = \left(\frac{\delta}{\delta x}, \frac{\delta}{\delta y}, \frac{\delta}{\delta z} \right)!$

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

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....

1.4.1 Solution:

$$\vec{E} = -\nabla\phi = -\nabla\left(\frac{1}{4\pi\epsilon_0}\frac{\vec{p}\cdot\vec{r}}{|\vec{r}|^3}\right) = -\nabla\left(\frac{1}{4\pi\epsilon_0}\frac{p_x\cdot x+p_y\cdot y+p_z\cdot z}{(r_x^2+r_y^2+r_z^2)^{\frac{3}{2}}}\right) = -\frac{1}{4\pi\epsilon_0}\begin{pmatrix} \frac{\delta}{\delta x} \\ \frac{\delta}{\delta y} \\ \frac{\delta}{\delta z} \end{pmatrix}\begin{pmatrix} p_x\cdot x+p_y\cdot y+p_z\cdot z \\ (r_x^2+r_y^2+r_z^2)^{\frac{3}{2}} \end{pmatrix}$$

$$\frac{\delta}{\delta x}\left(\frac{\vec{p}\cdot\vec{r}}{|\vec{r}|^3}\right) = \frac{\delta}{\delta x}\left(\frac{p_x\cdot x+p_y\cdot y+p_z\cdot z}{(x^2+y^2+z^2)^{\frac{3}{2}}}\right) = \frac{p_x|r|^3 - \vec{p}\cdot\vec{r}3x(x^2+y^2+z^2)^{\frac{1}{2}}}{|r|^6} = \frac{p_x}{|r|^3} - 3\frac{\vec{p}\cdot\vec{r}x|r|}{|r|^6} = \frac{p_x}{|r|^3} - 3\frac{\vec{p}\cdot\vec{r}\cdot x}{|r|^5}$$

Equivalently for $\frac{\delta}{\delta y}$ and $\frac{\delta}{\delta z}$...

This leads to:

$$\vec{E} = -\frac{1}{4\pi\epsilon_0}\begin{pmatrix} \frac{p_x}{|r|^3} - 3\frac{\vec{p}\cdot\vec{r}\cdot x}{|r|^5} \\ \frac{p_y}{|r|^3} - 3\frac{\vec{p}\cdot\vec{r}\cdot y}{|r|^5} \\ \frac{p_z}{|r|^3} - 3\frac{\vec{p}\cdot\vec{r}\cdot z}{|r|^5} \end{pmatrix} = \frac{1}{4\pi\epsilon_0}\left(3\frac{\vec{p}\cdot\vec{r}}{|r|^5}\cdot\vec{r} - \frac{\vec{p}}{|r|^3}\right)$$

1.5 Task 5: Current density & potential of a single PSP (5 points - programming)

Write a python function resembling the electrical field potential $\phi(\vec{r}, \vec{r}_q)$ and another one for the current density \vec{J} of a current dipole in infinite homogeneous conductive space at point \vec{r}_q in Cartesian coordinates. The function should take the position of the observed point \vec{r} , the position of the dipole \vec{r}_q , the current density produced by the source \vec{Q} and the conductivity σ as inputs and return the resulting scalar potential/the current density of the volumce conduction, respectively. \vec{r}_q should be set to the origin by default.

$$\phi(\vec{r}) = \frac{1}{4\pi\sigma}\frac{\vec{Q}\cdot(\vec{r}-\vec{r}_q)}{|\vec{r}-\vec{r}_q|^3}$$

$$\vec{J} = \frac{1}{4\pi}\left(3\frac{(\vec{r}-\vec{r}_q)\cdot\vec{Q}}{|\vec{r}-\vec{r}_q|^5}\cdot(\vec{r}-\vec{r}_q) - \frac{\vec{Q}}{|\vec{r}-\vec{r}_q|^3}\right)$$

σ is the specific conductivity of the material (a constant) and \vec{Q} is the current density at the source (neuron). For brain tissue, the conductivity is around $\sigma = 0.33\frac{S}{m}$.

Make a plot of the two variables in two dimensions from -10cm to 10cm for both axes in the x/y-plane for a dipole at origin with dipole moment $\vec{Q} = [20, 0]fAm$. Plot the potential using the matplotlib function `plt.contour` for plotting and the current density using the function `plt.streamplot`. Draw both plots into the same figure.

Hints: - Numpy's tensordot might simplify the treatment of multiple positions at once. - The arguments to the plot function call for \vec{J} and ϕ values need the shape corresponding to the XX and YY variables from np.meshgrid. - Increase the number of contours in plt.contour to get a nicer distribution of isopotential lines or better use a logarithmic scaling - you may use contourf to add colors

```
[5]: def phi_dip(r, Q, r_0=None, sigma=0.33):
    r=np.array(r)
    Q=np.array(Q)
    if r_0 is not None:
        r_0=np.array(r_0)
        r=(r.T-r_0).T
    return 1/(4*np.pi*sigma)*(np.tensordot(Q,r,1))/np.power(np.linalg.
↪norm(r,axis=0),3)

def J_dip(r, Q, r_0=None, sigma=0.33):
    r=np.array(r)
    Q=np.array(Q)
    if r_0 is not None:
        r_0=np.array(r_0)
        r=(r.T-r_0).T
    nom=((3*np.tensordot(Q,r,1)*r/np.power(np.linalg.norm(r,axis=0),2)).T-Q).T
    return (1/(4*np.pi)*nom/np.power(np.linalg.norm(r,axis=0),3))

#initialize the variables (x and y in cm, p as a current dipole with 20fAm)
x=np.arange(-10,10,0.1)
y=np.arange(-10,10,0.1)
r_0=np.array([0,0])
Q=np.array([20,0])*1e-15;

#create the meshgrid for calculation and plotting (in SI --> m)
XX,YY=np.meshgrid(x/100,y/100)

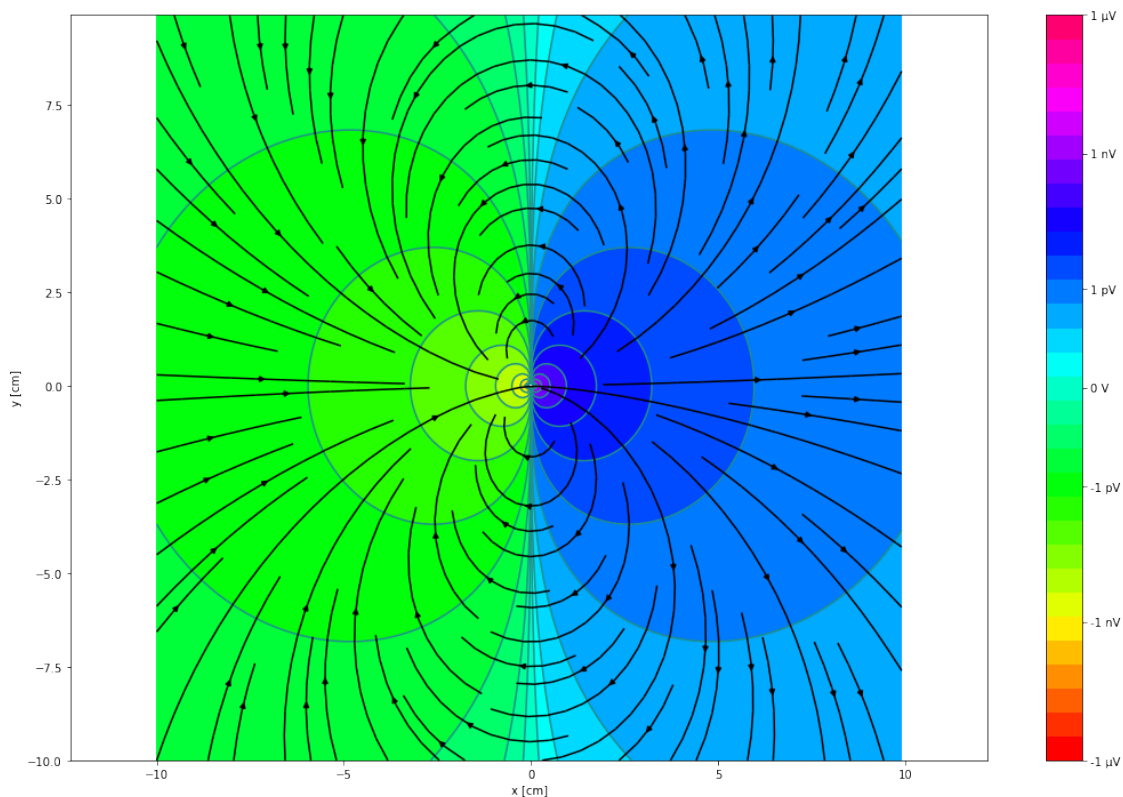
#Calculate the potential on the whole meshgrid
phi=phi_dip([XX,YY],Q,r_0)
#Calculate the current density on the whole meshgrid
J=J_dip([XX,YY],Q,r_0)

#plot the potential with logarithmic contours
fig=plt.figure(figsize=[18,12])
contours=np.logspace(-14,-6,16);
contours=np.concatenate([-np.flip(contours), contours])
potplot=plt.contour(x,y,phi,contours)
#potplot=plt.contourf(x,y,phi,contours,cmap=plt.cm.hsv)
potplot=plt.contourf(x,y,phi,contours,colors=plt.cm.hsv(np.arange(contours.
↪shape[0])/contours.shape[0]))
```

```

#add a colorbar for interpreting the values
cbar=plt.colorbar()
cbartick=[-1e-6,-1e-9,-1e-12,0,1e-12,1e-9,1e-6]
cbarticklab=['-1  $\mu$ V', '-1 nV', '-1 pV', '0 V', '1 pV', '1 nV', '1  $\mu$ V']
cbar.set_ticks(cbartick)
cbar.set_ticklabels(cbarticklab)
#plot the current density
fieldplot=plt.streamplot(x,y,J[0,:,:],J[1,:,:],color='k')
#make axis equal to avoid distortion and set labels
fig.axes[0].axis('equal')
plt.xlabel('x [cm]')
plt.ylabel('y [cm]')
plt.show()

```



```

[6]: # a short demo that shows how important it is to not use for loops in python
      ↪but (numpy) array operations instead
import time
now=time.time()
phi=phi_dip([XX,YY],Q,r_0)
print("execution time tensordot="+str(time.time()-now)+"s")
now=time.time()

```



```
for xx in x:
    for yy in y:
        phi_dip([x,y],Q,r_0)
print("execution time for loops="+str(time.time()-now)+"s")
```

execution time tensordot=0.009901046752929688s

execution time for loops=6.82932448387146s