Homework 1
Deep Learning

Guillaume Delorme ggd2113
Guillaume Michel gjm2152
Jeremy Yao jy3015
Thomas Luong tl2961

February 25, 2020

Deep Learning Homework 1

Problem 1 :

Let us prove the result by induction.

If we have no hidden layer, the result is obvious. Same if we have one layer.

If we have $n+1$ layers : and assume the result holds for $(n-1)$ layers.

$$\text{input} \longrightarrow \underset{W_1, b_1}{\bigcirc} \xrightarrow{f_1} \cdots \longrightarrow \underset{W_n, b_n}{\bigcirc} \xrightarrow{f_n}$$

the logits $\eta_i(x)$ can be written $\eta_i = W_i^T x + b_i$

because the activation function f_i is linear, $f_i(\eta_i) = A_i \eta_i = A_i W_i^T x + A_i b_i$

$$x \longrightarrow \underset{W_{n-1}, b_{n-1}}{\bigcirc} \xrightarrow{f_{n-1}} \underset{W_n, b_n}{\bigcirc} \xrightarrow{f_n} y \iff A_n\left(W_n^T\left(A_{n-1}W_{n-1}^T x + A_{n-1} b_{n-1}\right) + b_n\right) = y$$

$$\iff A_n W_n^T A_{n-1} W_{n-1}^T x + A_n W_n^T A_{n-1} b_{n-1} + A_n b_n = y$$

$$\iff A_n\left(\tilde{W}_{n-1} x + \tilde{b}_{n-1}\right) = y$$

$$\iff x \longrightarrow \underset{\tilde{W}_{n-1}, \tilde{b}_{n-1}}{\bigcirc} \xrightarrow{f_n} y$$

therefore the network can be rewritten as a combination of $(n-1)$ layers with linear activation functions, which is equivalent to one single linear neural network per our induction assumption.

Hence : we have proven by induction that a combination of any number of linearly activated layers is equivalent to one linear nn.

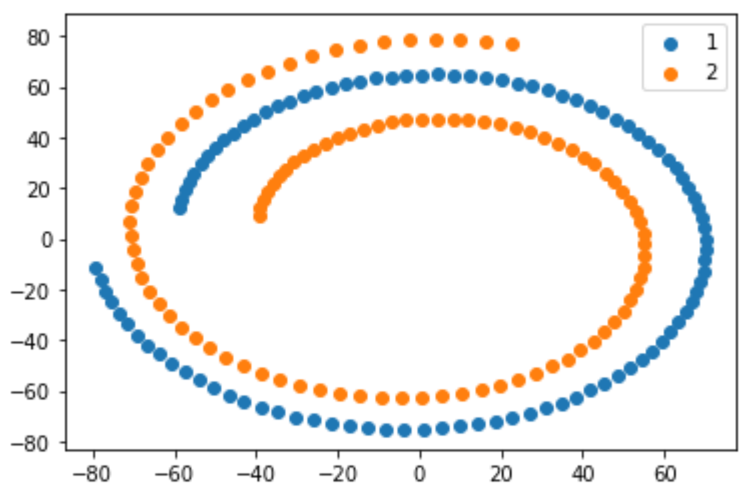
In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
%matplotlib inline
```

Problem 2

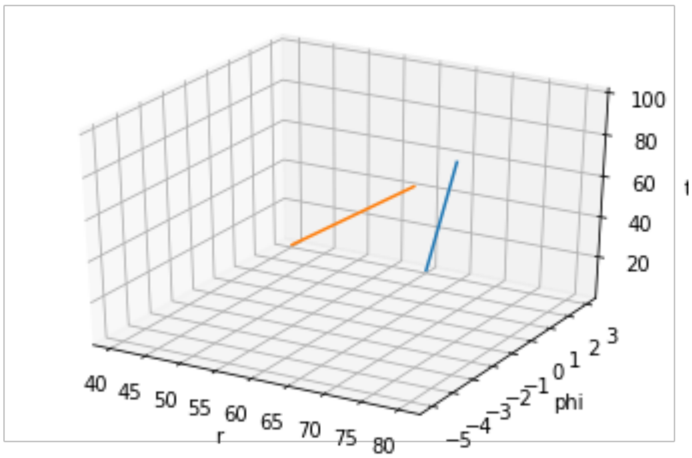
In [2]:
```
t = np.arange(1,101)
```

In [3]:
```
x1 = (60 + 0.2*t)*np.cos(-0.06*t + 3)
y1 = (60 + 0.2*t)*np.sin(-0.06*t + 3)
r1 = 60 + 0.2*t
phi1 = -0.06*t + 3

x2 = (40 + 0.4*t)*np.cos(-0.08*t + 3)
y2 = (40 + 0.4*t)*np.sin(-0.08*t + 3)
r2 = 40 + 0.4*t
phi2 = -0.08*t + 3
```

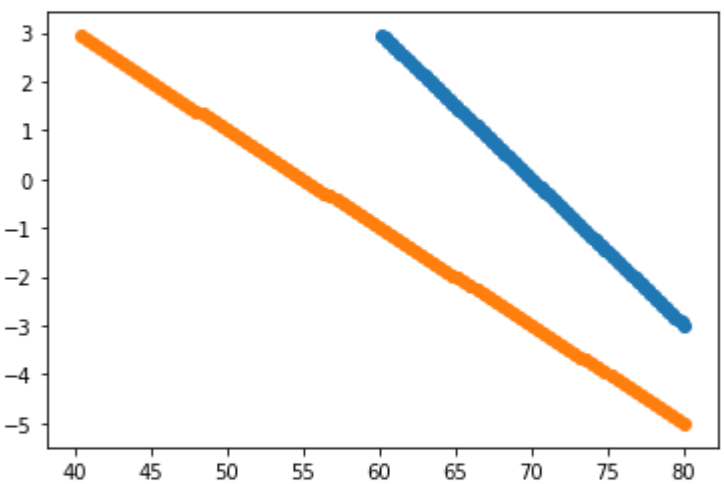
In [4]:
```
plt.scatter(x1,y1,label = '1')
plt.scatter(x2,y2,label='2')
plt.legend()
plt.show()
```



In [5]:
```
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.plot(r1, phi1 ,t)
ax.plot(r2, phi2 ,t)


ax.set_xlabel("r")
ax.set_ylabel("phi")
ax.set_zlabel("t")
plt.show()
```



In [6]:
```
plt.scatter(r1,phi1)
plt.scatter(r2,phi2)
plt.show()
```



The two lines seem to be separable as is, but increasing t would lead to them intersecting at some point, so let's stick with the 3D representation

Using an other definition for the loss

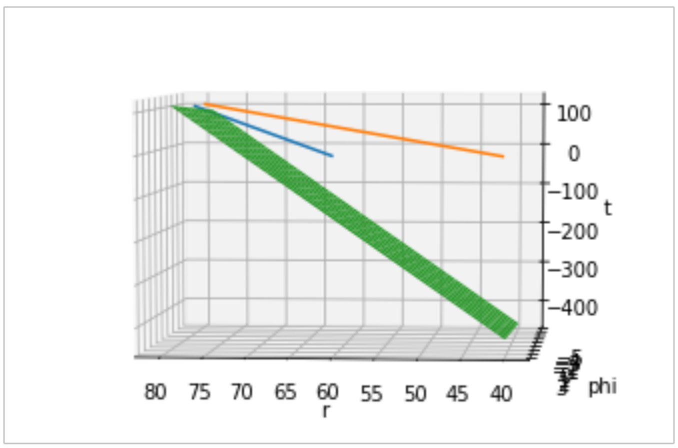
In [7]:
```
def hinge_loss_wiki(w,theta1,theta2):
    x=theta1
    y=theta2
    [w0,w1,w2,w3]=w
    loss=0
    for i in range (len(t)):
        y1=w0 + w1*x[i][0] + w2*x[i][1] +w3*x[i][2]
    if y1>0:
        z=1
    else:
        z=-1
    loss+=(max(0,1-z))

    y2=w0 + w1*y[i][0] + w2*y[i][1] + w3*y[i][2]

    if y2>0:
        z=1
    else:
        z=-1
    loss+=(max(0,1+z))

    return loss
```

In [8]:
```
min_loss = np.inf
theta1 = [[r1[i], phi1[i], t[i]] for i in range(len(t))]
theta2 = [[r2[i], phi2[i], t[i]] for i in range(len(t))]
opt_params = []
for w0 in np.linspace(-2000,0,15):
    for w1 in np.linspace(-1,20,10):
        for w2 in np.linspace(0,10,10):
            for w3 in np.linspace(-10,1,10):
                if w1 != 0 or w2 != 0 or w3 != 0:
                    w=[w0,w1,w2,w3]
                    loss = hinge_loss_wiki(w,theta1, theta2)
                    if loss < min_loss:
                        min_loss = loss
                        opt_params = w
print(opt_params)
```

```
[-1428.5714285714284, 20.0, 5.555555555555555, -1.444444444444442
9]
```

In [9]:

```
r, phi = np.meshgrid(range(40,80), range(-5,4))

t2=-1/opt_params[-1]*(opt_params[0]+opt_params[1]*r+opt_params[2]*

fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.plot(r1, phi1 ,t)
ax.plot(r2, phi2 ,t)

ax.plot_surface(r, phi, t2, alpha=1)

ax.set_xlabel("r")
ax.set_ylabel("phi")
ax.set_zlabel("t")


ax.view_init(5, 95)
plt.show()
```



Problem 3

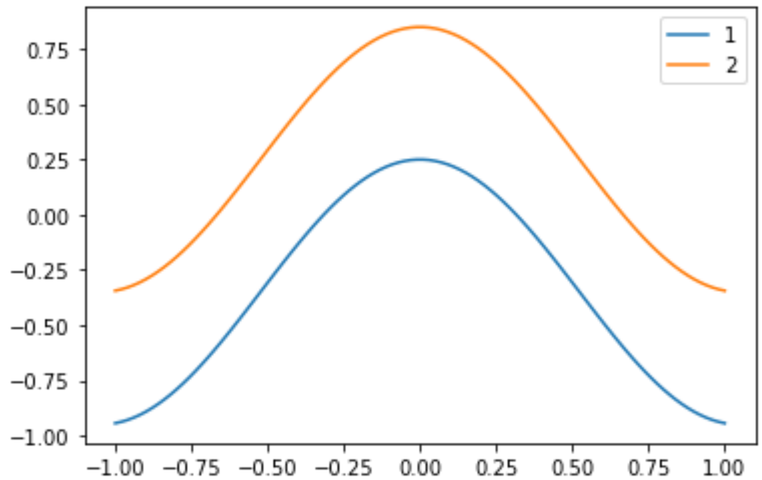
In [10]:

```
x = np.linspace(-1,1,100)
```

In [11]:

```
y1 = 0.6*np.sin(np.pi/2 + 3*x) - 0.35
y2 = 0.6*np.sin(np.pi/2 + 3*x) + 0.25
```

In [12]:
```
plt.plot(x,y1,label='1')
plt.plot(x,y2,label='2')
plt.legend()
plt.show()
```



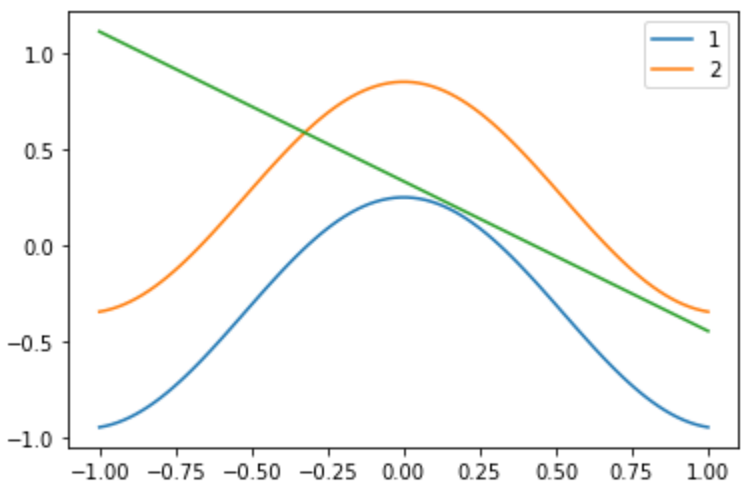
Let's apply the hinge loss to this

In [13]:
```
def hinge_loss_2D(w,theta1,theta2):
    x=theta1
    y=theta2
    [w0,w1,w2]=w
    loss=0
    for i in range (len(t)):
        y1=w0 + w1*x[i][0] + w2*x[i][1]
        if y1>0:
            z=1
        else:
            z=-1
        loss+=(max(0,1-z))

        y2=w0 + w1*y[i][0] + w2*y[i][1]

        if y2>0:
            z=1
        else:
            z=-1
        loss+=(max(0,1+z))

    return loss
```

In [14]:
```
min_loss = np.inf
theta1 = [[x[i], y1[i]] for i in range(len(x))]
theta2 = [[x[i], y2[i]] for i in range(len(t))]
opt_params = []
for w0 in np.linspace(-10,10,20):
    for w1 in np.linspace(-10,20,20):
        for w2 in np.linspace(-10,10,20):
            if w1 != 0 or w2 != 0 :
                w=[w0,w1,w2]
                loss = hinge_loss_2D(w,theta1, theta2)
                if loss < min_loss:
                    min_loss = loss
                    opt_params = w
print(opt_params)
```

```
[1.5789473684210513, -3.6842105263157894, -4.736842105263158]
```

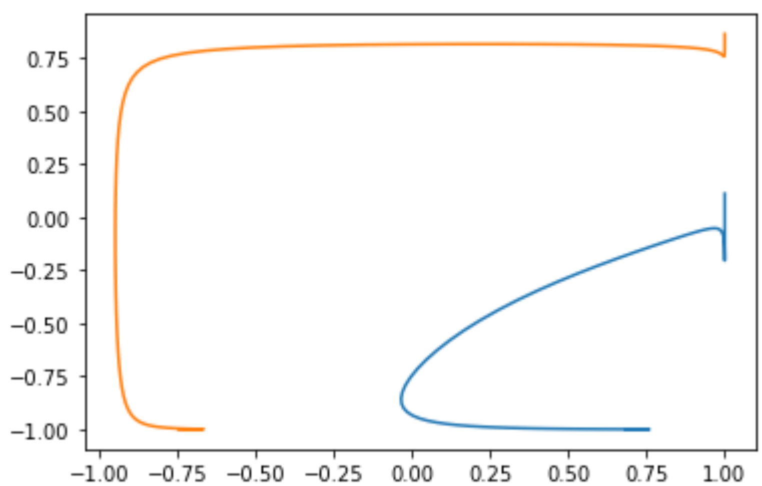
In [15]:
```
plt.plot(x,y1,label='1')
plt.plot(x,y2,label='2')
y=-(opt_params[1]*x+opt_params[0])/opt_params[2]
plt.plot(x,y)
plt.legend()
plt.show()
```



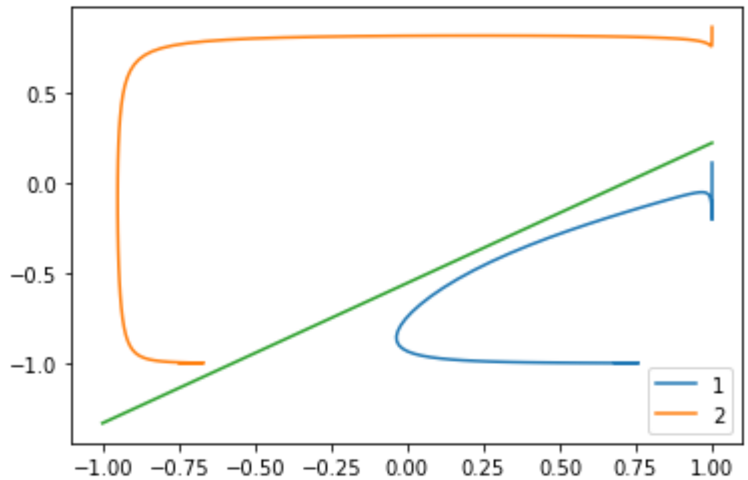
In [16]:
```
def func():
    opt_params = []
    for w11 in np.linspace(-3,3,7):
        for w12 in np.linspace(-3,3,7):
            for w21 in np.linspace(-3,3,7):
                for w22 in np.linspace(-3,3,7):
                    for b1 in np.linspace(-1,1,3):
                        for b2 in np.linspace(-1,1,3):
                            for a in np.linspace(-5,5,10):
                                for b in np.linspace(-5,5,10):
                                    for c in np.linspace(-5,5,10):
                                        x1_hat = np.tanh(w11*x + w
                                        y1_hat = np.tanh(w21*x + w
                                        x2_hat = np.tanh(w11*x + w
                                        y2_hat = np.tanh(w21*x + w
                                        min_loss = np.inf
                                        theta1 = [[x1_hat[i], y1_h
                                        theta2 = [[x2_hat[i], y2_h
                                        if w1 != 0 or w2 != 0 :
                                            w=[a,b,c,w11,w12,w21,w
                                            loss = hinge_loss_2D(w
                                            if loss < min_loss:
                                                min_loss = loss
                                                opt_params = w
                                            if loss == 0:
                                                return opt_params,

opt_params,x1_hat,y1_hat,x2_hat,y2_hat = func()
```

In [17]:
```
plt.plot(x1_hat,y1_hat,label='1')
plt.plot(x2_hat,y2_hat,label='2')
```

Out[17]: [<matplotlib.lines.Line2D at 0x112f96240>]



In [18]:
```
plt.plot(x1_hat,y1_hat,label='1')
plt.plot(x2_hat,y2_hat,label='2')
x = np.linspace(-1,1,100)
y=-(opt_params[1]*x+opt_params[0])/opt_params[2]
plt.plot(x,y)
plt.legend()
plt.show()
```



By looking at the different transformations like we did in part b, it is clear that there are multiple cases where the two curves are linearly separable. Obviously, depending on the order of transformations we use, we will most likely get different separations. Therefore, the problem is quite sensitive to the starting point.