# EE550 - Implementation of 3-D Winner-Take-All Network

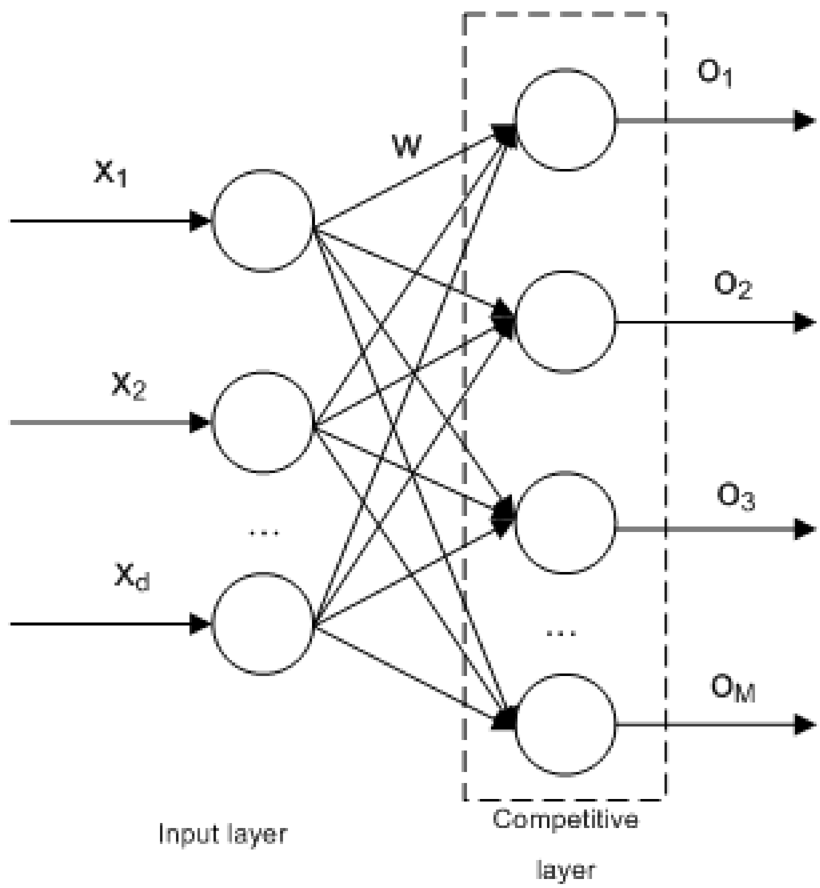
May 4, 2017

**Mine Melodi Çalışkan - 2015705009**

Competitive neural networks learn to categorize input pattern vectors. Each category of inputs actives a different output neuron. The categories formed are based on similarities between the input vectors. Similar, that is correlated , input vectors active the same output neuron. In that the learning is based on similarities in the input space, and there is no external teacher which forces classification, this is an unsuperviesd network.

```
In [29]: from IPython.core.display import Image
         Image(filename='img/competitive.png')
```

```
Out[29]:
```



Single layer of output units with $O_i$ being the output units.

Outputs are fully connected to a set of inputs denoted by $\xi_i$ with weight connection$w_{ij},$ where $w_{ij} > 0$

Binary inputs and outputs are considered. Only one output unit, called the winner, can be become active at a time.

The winner is the unit with the output with the largest net input.

$$h_i = \sum w_{ij}\xi_j = \vec{w_{i^*}}\vec{\xi}$$

$$\vec{w_{i^*}} = [w_{i1} \ldots w_{iN}] \quad , \vec{\xi} = [\xi_1 \ldots \xi_N]$$

Hence, $\vec{w_{i^*}}\vec{\xi} \geq \vec{w_i}\vec{\xi}, \ \forall i$ defines the winning unit $i^*$ with $O_{i^*} = 1$

If all the weights are normalized to $||w_i|| = 1, \forall i$

Then, $||\vec{w_{i^*}} - \vec{\xi}|| \leq ||\vec{w_i} - \vec{\xi}||, \forall i$

Winner is the unit with normalized $\vec{w}$ closest to the input vector $\vec{\xi}$

**Algorithm**

- Choose k weight vectors with the same dimensionality as the input vector
- Choose an input vector $\xi$ and calculate the distance between $\xi$ and all weight vectors.
- Find the index $i^*$ of the winner neuron and set its output activation to 1 and the activation of all other competitive neurons to 0.
- Adapt the weight vector of the winner only, using : $\Delta w_{i^*j} = \eta({\xi_j}^{\mu} - w_{i^*j})$ which moves $\vec{w_j}$ toward $\vec{\xi^{\mu}}$
- Choose another input vector and go to step 2
- At the end of the epoch, check if the stopping criteria is satisfied:

```
* No more changes in the position of the weight vectors
* Or maximum number of epochs is reached
```

In the code I used the identity $||x-y||^2 = ||x||^2 - 2* < x, y > +||y||^2$

and determined the winner unit as which minimizes the squared distance.

**Data Preparation**

We will generate uniformly distributed random data points on unit sphere.
To obtain such points we choose u, v to be random variates on (0,1). Then

$$\theta = 2\pi u \quad , \quad \phi = cos^{-1}(2v - 1)$$

gives the spherical coordinates for a set of points which are uniformly distributed over $\mathbb{S}^{\nvDash}$.

We can pick $u = cos\phi$ to be uniformly distributed and obtain the points.

$$x = \sqrt{1 - u^2}cos\theta$$

$$y = \sqrt{1 - u^2}sin\theta$$

$$z = u$$

with $\theta \in [0, 2\pi)$ and $u \in [-1, 1]$, which are also uniformly distributed over $\mathbb{S}^{\nvDash}$

In [1]:
```
import numpy as np
import pandas as pd
import math

def unit_sphere_data_generator(M):

    np.random.seed(56)
    u = np.random.uniform(0,1,M)
    v = np.random.uniform(0,1,M)
    theta = 2 * np.pi * u
    phi= np.arccos((2*v - 1.0))


    c=np.zeros((M,3))


    c[:,0] = np.cos(theta) * np.sin(phi)
    c[:,1] = np.sin(theta) * np.sin(phi)
    c[:,2] =np.cos(phi)



    sphere_points = pd.DataFrame(c)

    ##filter if there are any duplicates
    sphere_points_no_duplicates = sphere_points.drop_duplicates()

    generators = sphere_points_no_duplicates.as_matrix()

    return generators
```

In our case, we need 3 distinct data groups to make clustering easier.

With the above generator function, I decided to generate uniformly distributed data points on unit sphere and then apply 3 different restrictions and choose 50 points for each different group.

```
In [2]: points_on_sphere=unit_sphere_data_generator(2000)
```

```
In [3]: distinct_data1=[]
        distinct_data2=[]
        distinct_data3=[]
        for i in points_on_sphere:
            #If all coordinates are positive and above 0.05
            if np.all(i>0.05):
                distinct_data1.append(i)
            #If all coordinates are negative and belove -0.05
            elif np.all(i<-0.05):
                distinct_data2.append(i)
            #If x-axis negative and the others positive
            #and between provided values
            elif i[0]<0 and i[1]>0 and i[1]<0.8 and i[2]>0 and i[2]<0.5:
                distinct_data3.append(i)
```

```
In [4]: #take 50 points from first area
        distinct_data1=distinct_data1[:50]

        #take 40 points for train
        data1_train=distinct_data1[:40]

        #take remaining 10 points for test
        data1_test=distinct_data1[40:]

        #similar steps for the second and the third area's points.

        distinct_data2=distinct_data2[:50]
        data2_train=distinct_data2[:40]
        data2_test=distinct_data2[40:]


        #np.random.shuffle(distinct_data3)
        distinct_data3=distinct_data3[:50]
        data3_train=distinct_data3[:40]
        data3_test=distinct_data3[40:]
```

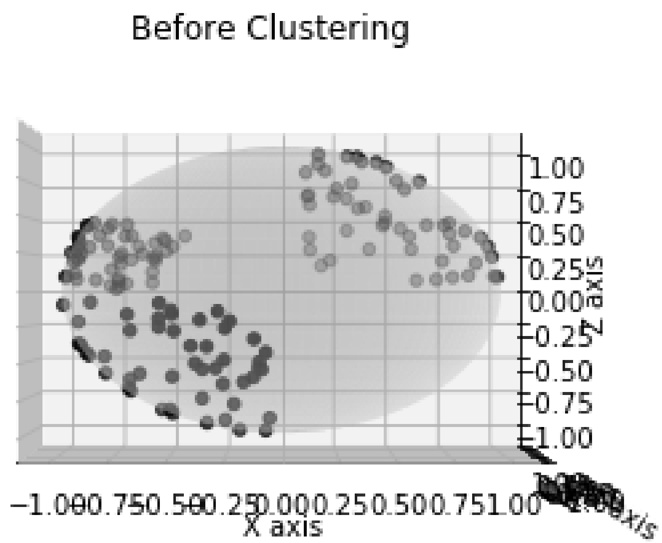
```
In [5]: #collect training data points
        training_datas=np.vstack((data1_train,data2_train,data3_train))

        #shuffle training data
        np.random.seed(91)
        np.random.shuffle(training_datas)
```

In [6]:
```
#collect testing data points
testing_datas=np.vstack((data1_test,data2_test,data3_test))

#shuffle testing data
np.random.seed(101)
np.random.shuffle(testing_datas)
```

In [7]:
```
#for ploting all data
all_data_points=np.vstack((training_datas,testing_datas))
```

In [26]:
```
import numpy as np
import matplotlib.pyplot as plt
import mpl_toolkits.mplot3d.axes3d as axes3d
from mpl_toolkits.mplot3d import Axes3D
from matplotlib import cm
```

In [9]:
```
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
plt3d=plt.subplot(projection='3d')

#plot unit sphere
u = np.linspace(0,2*np.pi, 100)
v = np.linspace(0, np.pi, 100)
x = np.outer(np.cos(u),np.cos(v))
y = np.outer(np.cos(u),np.sin(v))
z = np.outer(np.sin(u),np.ones(np.size(v)))

plt3d.plot_surface(x, y, z, color='w', alpha=0.3)


#plot data points on the unit sphere
plt3d.scatter(all_data_points[:,0],all_data_points[:,1],
              all_data_points[:,2], c='k')



plt3d.set_xlabel('X axis')
plt3d.set_ylabel('Y axis')
plt3d.set_zlabel('Z axis')
plt3d.view_init(0,-90)
plt.title('Before Clustering')
plt.show()
```




**Train Phase**

```
In [10]: import numpy as np
         from sklearn.preprocessing import normalize

         def CompetitiveNetwork(X,epochs,eta=0.01):
             '''
             X:Input data points
             epochs: number of epochs
             eta: learning rate
             '''

             #number of data points
             N=X.shape[0]

             #it's convenient to choose
             #initial 3 weights randomly from data points
             W=X[np.random.choice(np.arange(len(X)), 3), :]

             #store weights to plot weights trajectories
             trj=[W]


             X2=(X**2).sum(axis=1)[:,np.newaxis]


             for epoch in range(epochs):
```

```
            #to store datas with assigned labels
            assigned_labels=[]
            for i in range(N):
                distance=X2[i:i+1].T-2*np.dot(W,X[i:i+1,:].T)
                +(W**2).sum(axis=1)[:,np.newaxis]
                output=(distance==distance.min(axis=0)[np.newaxis,:]).T
                output=output.astype("int")
                assigned_labels.append([X[i:i+1,:],output])
                #output=[1,0,0] if first class is winner and so on.

                #So multiplication with "output" in below,
                #provides update for the winner weight only.

                #weight update
                W+= eta*(np.dot(output.T,X[i:i+1,:])
                         -output.sum(axis=0)[:,np.newaxis]*W)

                #normalize the weights
                W=normalize(W, norm='l2', axis=1)


            trj.append(W)
            #if weights doesn't change then break
            if (trj[epoch]==trj[epoch-1]).all():
                break

        return W,trj,assigned_labels
```

In [11]:
```
XX=training_datas
#epochs: 100
```

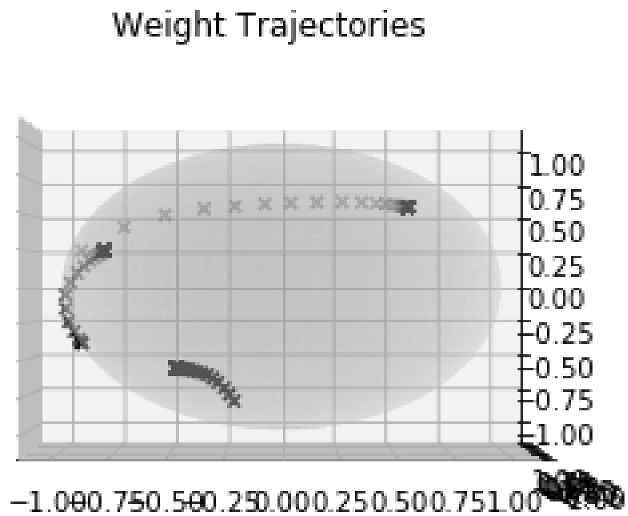
In [12]:
```
centers,trajectories,labelled_data=CompetitiveNetwork(XX,100,eta=0.01)
```

In [13]:
```
centers
```

Out[13]:
```
array([[-0.82790793,  0.49882935,  0.25639371],
       [ 0.58848857,  0.57000613,  0.57338837],
       [-0.4658794 , -0.69241838, -0.5509203 ]])
```

In [14]:
```
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
plt3d=plt.subplot(projection='3d')

#plot unit sphere
plt3d.plot_surface(x, y, z, color='w', alpha=0.3)

#plot weight trajectories
for i in trajectories:
    plt3d.scatter(i[:,0],i[:,1],i[:,2], c='k', marker='x')
```

```
    plt3d.view_init(0,-90)
plt.title('Weight Trajectories')
plt.show()
```

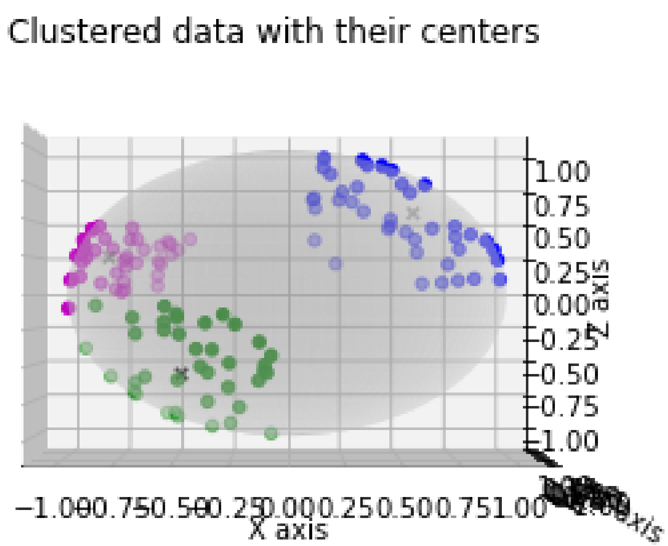



In [15]:
```
#labelled_data has a form as given: [data_point, class]
#for example:
#[array([[-0.89138016,  0.31313103,  0.3277047 ]]), array([[ 1.,  0.,  0.
#I will assign classes as follows:
#pink class --> [1,0,0]
#blue class --> [0,1,0]
#green class --> [0,0,1]
```

In [16]:
```
pink_class=[]
blue_class=[]
green_class=[]
for i in labelled_data:
    if np.all(i[1][0]==[1,0,0]):
        pink_class.append(i[0][0])
    elif np.all(i[1][0]==[0,1,0]):
        blue_class.append(i[0][0])
    elif np.all(i[1][0]==[0,0,1]):
        green_class.append(i[0][0])
```

In [17]:
```
pink_class=np.array(pink_class)
blue_class=np.array(blue_class)
green_class=np.array(green_class)
```

In [18]:
```
#plot resulted clusters with the last weights

fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
plt3d=plt.subplot(projection='3d')

#plot unit sphere
plt3d.plot_surface(x, y, z, color='w', alpha=0.3)

#plot data points
plt3d.scatter(pink_class[:,0],pink_class[:,1],pink_class[:,2], c='m')
plt3d.scatter(green_class[:,0],green_class[:,1],green_class[:,2], c='g')
plt3d.scatter(blue_class[:,0],blue_class[:,1],blue_class[:,2], c='b')


#plot centers
plt3d.scatter(centers[:,0],centers[:,1],centers[:,2], c='k', marker='x')



plt3d.set_xlabel('X axis')
plt3d.set_ylabel('Y axis')
plt3d.set_zlabel('Z axis')
plt3d.view_init(0,-90)
plt.title('Clustered data with their centers')
plt.show()
```




**Test phase for unseen datas**

In [19]:
```
#In the test step we classify datas with provided centers
#from training step.
#In the algorithm, determined centers should be given
#and we no longer update weights.
def Classify(testX,c):
    '''
    testX: testing datas
    c: centers
    '''
    classes=[]
    X2=(testX**2).sum(1)[:,np.newaxis]
    for i in range(0,testX.shape[0],1):
        distance=X2[i:i+1].T-2*np.dot(c,testX[i:i+1,:].T)
        +(c**2).sum(1)[:,np.newaxis]
        output=(distance==distance.min(0)[np.newaxis,:]).T
        output=output.astype("int")
        classes.append([testX[i:i+1,:],output])
    return classes
```

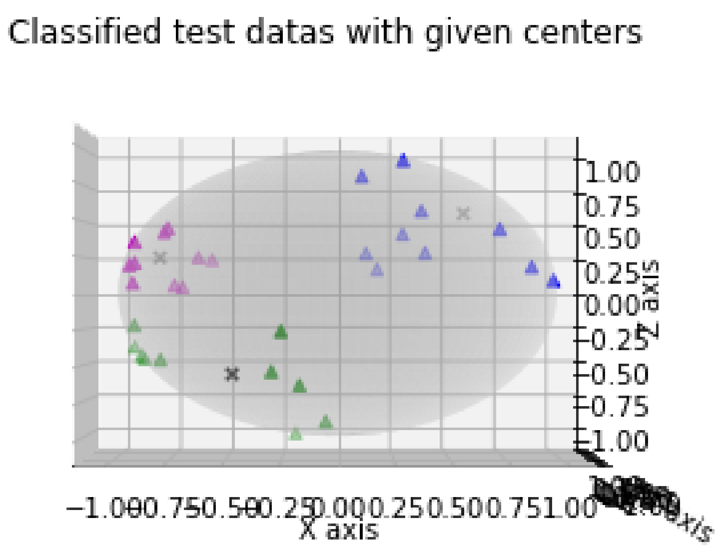
In [20]:
```
test_classes=Classify(testing_datas,centers)
```

In [21]:
```
test_pink_class=[]
test_blue_class=[]
test_green_class=[]
for i in test_classes:
    if np.all(i[1][0]==[1,0,0]):
        test_pink_class.append(i[0][0])
    elif np.all(i[1][0]==[0,1,0]):
        test_blue_class.append(i[0][0])
    elif np.all(i[1][0]==[0,0,1]):
        test_green_class.append(i[0][0])
```

In [22]:
```
test_pink_class=np.array(test_pink_class)
test_blue_class=np.array(test_blue_class)
test_green_class=np.array(test_green_class)
```

In [23]:
```
#plot resulted clusters with the last weights

fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
plt3d=plt.subplot(projection='3d')

#plot unit sphere
plt3d.plot_surface(x, y, z, color='w', alpha=0.3)

#plot data points
plt3d.scatter(test_pink_class[:,0],test_pink_class[:,1],
              test_pink_class[:,2], c='m',marker='^')
plt3d.scatter(test_green_class[:,0],test_green_class[:,1],
```

```
                  test_green_class[:,2], c='g',marker='^')
plt3d.scatter(test_blue_class[:,0],test_blue_class[:,1],
              test_blue_class[:,2], c='b',marker='^')


#plot centers
plt3d.scatter(centers[:,0],centers[:,1],centers[:,2], c='k', marker='x')



plt3d.set_xlabel('X axis')
plt3d.set_ylabel('Y axis')
plt3d.set_zlabel('Z axis')
plt3d.view_init(0,-90)
plt.title('Classified test datas with given centers')
plt.show()
```




We can observe that test data points are classified as expected color classes.

In [24]: test_classes

```
Out[24]: [[array([[-0.8047478 ,  0.40411486,  0.43482428]]), array([[1, 0, 0]])],
          [array([[-0.72851132,  0.68352227,  0.04548138]]), array([[1, 0, 0]])],
          [array([[ 0.30815716,  0.84808556,  0.4310337 ]]), array([[0, 1, 0]])],
          [array([[-0.29111062, -0.79611131, -0.53052934]]), array([[0, 0, 1]])],
          [array([[ 0.30130853,  0.16364344,  0.93937958]]), array([[0, 1, 0]])],
          [array([[-0.94651441,  0.31416908,  0.07354092]]), array([[1, 0, 0]])],
          [array([[-0.65448764,  0.71002504,  0.25982759]]), array([[1, 0, 0]])],
```

```
[array([[-0.91845213, -0.13290477, -0.37253458]]), array([[0, 0, 1]])],
[array([[-0.59203932,  0.76816715,  0.24373894]]), array([[1, 0, 0]])],
[array([[-0.7868402 ,  0.40479248,  0.46586   ]]), array([[1, 0, 0]])],
[array([[-0.05292391, -0.46998573, -0.88108596]]), array([[0, 0, 1]])],
[array([[ 0.98884337,  0.11895477,  0.08965802]]), array([[0, 1, 0]])],
[array([[ 0.11337908,  0.52851915,  0.84131605]]), array([[0, 1, 0]])],
[array([[-0.88465814, -0.17202327, -0.43334509]]), array([[0, 0, 1]])],
[array([[-0.76422852,  0.64231939,  0.0581426 ]]), array([[1, 0, 0]])],
[array([[-0.79310362, -0.40132383, -0.45817664]]), array([[0, 0, 1]])],
[array([[ 0.75519588,  0.46604602,  0.46095584]]), array([[0, 1, 0]])],
[array([[-0.19022115, -0.08060218, -0.9784269 ]]), array([[0, 0, 1]])],
[array([[ 0.18764394,  0.9664422 ,  0.1754401 ]]), array([[0, 1, 0]])],
[array([[-0.93682652,  0.27450029,  0.21680791]]), array([[1, 0, 0]])],
[array([[ 0.41468672,  0.8607686 ,  0.29514801]]), array([[0, 1, 0]])],
[array([[ 0.90074648,  0.39124308,  0.18863889]]), array([[0, 1, 0]])],
[array([[-0.16816403, -0.76288585, -0.62428042]]), array([[0, 0, 1]])],
[array([[ 0.39446345,  0.69770719,  0.59799938]]), array([[0, 1, 0]])],
[array([[-0.8713677 , -0.18258729, -0.45539018]]), array([[0, 0, 1]])],
[array([[-0.9285584 ,  0.08478633,  0.36137317]]), array([[1, 0, 0]])],
[array([[-0.95727458,  0.20891195,  0.19995296]]), array([[1, 0, 0]])],
[array([[ 0.13504707,  0.94665967,  0.29257094]]), array([[0, 1, 0]])],
[array([[-0.91211799, -0.34564096, -0.22039304]]), array([[0, 0, 1]])],
[array([[-0.24812484, -0.93422701, -0.25623028]]), array([[0, 0, 1]])]]
```