In [1]:
```python
import pandas as pd
import numpy as np
from sklearn.preprocessing import MaxAbsScaler
from sklearn.preprocessing import OneHotEncoder
from scipy.linalg import pinv2
from sklearn.metrics import accuracy_score
from numpy import linalg as LA
from sklearn import preprocessing
import random
import time
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
import pyswarms as ps

from pyswarms.single.global_best import GlobalBestPSO
from pyswarms.utils.functions import single_obj as fx
# Some more magic so that the notebook will reload external python modules;
# see http://stackoverflow.com/questions/1907993/autoreload-of-modules-in-ipython
%load_ext autoreload
%autoreload 2
```

In [2]:
```python
#This function uses parameters like radius, ditance between two moons, number of samples in total and width as per which
#half moon will be generated and the data points will be stored along with their labels.

features=3 #number of attributes +label column
instances=3000
r=10
d=-4
w=4



#First we need to check whether the number of samples are even or not
if (instances%2!=0) :
    print("*****Error****** Number of samples are not valid; They should be even ")
    instances=instances+1

#Matrix initialization of samples with 0 as initial value for x,y and label values.
valuesofSamples=np.zeros((features,instances),dtype=int)
#print(valuesofSamples)

# Boundary condition checking
if (r<w/2):
    print("*****Error******  Radius is not enough")

#Creating random float values of x and y coordinates of half the instances
randomval=np.random.random((2,int(instances/2)))
#print (randomval)

radii=(r-w/2)+w*randomval[0][:]

#Calculating outer radius for one half moon
theta=np.pi*randomval[1][:]

#Creation of datasets for both the half moons
x_class1=np.multiply(radii,np.cos(theta)) #X coordinate for 1st half data points
y_class1=np.multiply(radii,np.sin(theta)) #y coordinate for 1st half data points
label_class1=np.ones((1,len(x_class1)),dtype=int) #providing label as 1 to the entire 1st half moon data
label_class1=np.hstack(label_class1)

x_class2=np.multiply(radii,np.cos(-theta))+r #X coordinate for 2nd half data points
y_class2=np.multiply(radii,np.sin(-theta))-d #y coordinate for 2nd half data points
label_class2=0*np.ones((1,len(x_class2)),dtype=int) #providing label as 0 to the 2nd half moon data
label_class2=np.hstack(label_class2)



#Now we will create a single matrix with all the x,y coordinates of all the points belonging to both the halves
#using a nested list functionality,with their corresponding labels
valuesofSamples[0,:]=np.concatenate([x_class1,x_class2])
valuesofSamples[1,:]=np.concatenate([y_class1,y_class2])

valuesofSamples[2,:]=np.concatenate(([label_class1,label_class2]))

#converting to dataframe and Transposing it to get columns on the top
df=(pd.DataFrame(valuesofSamples)).T

DF=df.rename(columns={0:'x',1:'y',2:'labels'}) #Renaming the column name as per index


DF.sample(frac=1) # Randomizing the whole dataset
DF_train,DF_test=train_test_split(DF, test_size=0.30) #splitting the dataset with 1000 training and 2000 testing data points
#Separating the feature data and label data for the training set
feature_valTrain=DF_train[['x','y']]
Label_train=pd.DataFrame(DF_train['labels'])

#Separating the feature data and label data for the testing set
feature_valTest=DF_test[['x','y']]
Label_test=pd.DataFrame(DF_test['labels'])
```

In [3]:
```python
#Scaling Train and test features
scale=MaxAbsScaler()
Train_features=scale.fit_transform(feature_valTrain)
Test_features=scale.fit_transform(feature_valTest)
print(Train_features[0])
```

```
[0.19047619 0.81818182]
```

In [4]:
```python
#Doing one hot encoding on labels to convert classes to a vector notation or binarizing the labels
encoder=OneHotEncoder(categories='auto')

Train_Label=encoder.fit_transform(Label_train).toarray()
Test_Label=encoder.fit_transform(Label_test).toarray()
```

In [5]:
```python
#Activation function definintion
def sigmoid(y):
    a = 1/(1+np.exp(-y))
    #a=np.maximum(0,y)
    return a
```

In [6]:
```python
# Function to calculate the accuracy using the maximum index of the binary value and comparing that with the original label index
def accuracy(Error,target):
    count=0

    for i in range(len(Error)):

        index=np.argmax(target[i])    # getting the index of the target label
        index_predicted=np.argmax(Error[i]) #getting the index of the predicted label
        if index_predicted==index:
            count=count+1
    predicted_accuracy=(count/len(Error))*100   #calculating the accuracy based on the total correct predictions vs total values
    #print(predicted_accuracy)
    return predicted_accuracy
```

In [7]:
```python
#defining ielm objective function to get the minimum loss which is been compared using Global best FUNCTION OF sWARM OPTIMIZER
# THIS FUNCTION IS CALLED BY EACH PARTICLE WHICH RECEIVES THE LOSS AND THEN THAT LOSS IS KEPT IN A LIST WHICH IS THEN
#COMPARED TO GET THE MINIMUM LOSS AND HENCE THE OPTIMIZED WEIGHTS AND BIASES.
#here loss means maximizing the accuracy. hen 1- accuracy to get the loss

def IELm_objective(dimensions,E):

    a = dimensions[0:2].reshape((len(Train_features[0]),1)) #retrieving weight which is first 2 elements of the array dimensions
    b=dimensions[-1].reshape(1,1)
    predicted_accuracy=0  #Initial value of accuracy
    beta=0
    mul=np.dot(Train_features,a)
    sum=mul+b
    activation=sigmoid(sum) # activation function sigmoid to calculate H
    beta_overall=np.dot(np.linalg.pinv(activation),E) # beta= inverse(H) * Target Label. As our target keeps on changing with new neuron
                                              #hence E
    y=np.dot(activation,beta_overall) # Calculated output
    E=E-y # Updated Training Label or Error
    predicted_accuracy=accuracy(y,Train_Label) #Calculating the accuracy of the predicted labels
    return(1-predicted_accuracy)
```

In [8]:
```python
#Iterating the number of particles and calling the objctive function to get the loss.
def swarm_initializer(input,**kwargs):

    particles = input.shape[0]
    for i in range(particles):
        loss = IELm_objective(input[i],**kwargs) #passing the number of values required after optimization and updated target variable via **kwargs which is updated target label only
    return np.array(loss)
```

In [9]:
```python
L=0 # Number of neurons in hidden layer inititally
max_neuron=10 # Maximum number of Neurons
Accuracy_expected=97 # Expected accuracy or a threshold value that the network should have to break stop adding the neurons
predicted_accuracy=0  #Initial value of accuracy
beta=0
E=Train_Label
 #Initial value of error as training set label, to keep a check on the differnce between the target and the predicted
new_a=np.random.uniform(low=-1,high=1,size=[len(Train_features[0]),1]) #Generating numpy array of weights for the first hidden neuron
new_b=np.random.uniform(low=-1,high=1,size=[1]) #Generating numpy array of bias for the first hidden neuron

# ****** TRAININIG STARTS HERE ***********
#Loop until the hidden layer reaches maximum number of neurons and until the predicted accuracy doesnt surpass the expected accuracy
start_time=time.time()
options = {'c1': 0.5, 'c2': 0.3, 'w':0.9} #Setting the hyper parameter and passing to the optimizer
lb = [-1,-1,-1] #setting the upper and lower limit bounds for the weights and biases
ub = [1,1,1]
bounds=(lb,ub)
while L<max_neuron and predicted_accuracy<Accuracy_expected:
    L+=1

    #for the first neuron using the random produced weights and biases followed by further steps
    if L==1:
        mul=np.dot(Train_features,new_a)
        sum=mul+new_b
        activation=sigmoid(sum) # activation function sigmoid to calculate H
        beta_overall=np.dot(np.linalg.pinv(activation),E) # beta= inverse(H) * Target Label. As our target keeps on changing with new neuron
                                                  #hence E
        y=np.dot(activation,beta_overall) # Calculated output
        E=E-y # Updated Training Label or Error
        predicted_accuracy=accuracy(y,Train_Label) #Calculating the accuracy of the predicted labels
        print("-------- TRAINING ACCURACY ----------")
        print("")
        print(predicted_accuracy,"With", L," Neuron")

    else:
        optimizer = ps.single.GlobalBestPSO(n_particles=20,dimensions=3, options=options, bounds=bounds) #setting number of particles 20 with bounds as [-1,1] for both weights and bias
        cost, weights_bias = optimizer.optimize(swarm_initializer, iters=50,E=E) #calling the iterating function of particles which will run the objective functioin and also passing updated labels for calulcaing the error
        print(cost)
        print(weights_bias)

        a=weights_bias[:2].reshape(len(Train_features[0]),1)
#         a=np.random.uniform(low=-1,high=1,size=[len(Train_features[0]),1]) #creating a new random weights for the second neuron and so on
#         b=np.random.uniform(low=-1,high=1,size=[1]) #creating a new random bias for the second neuron and so on
        b=weights_bias[-1].reshape(1,)
        new_a=np.append(new_a,a,axis=1) # adding the newly created weight to the weight matrix of the previous neuron
        mul=np.dot(Train_features,a)
        new_b=np.append(new_b,b,axis=0) # adding the newly created bias to the bias matrix of the previous neuron
        sum=mul+b
        activation=sigmoid(sum)
        beta=np.dot(pinv2(activation), E)
        y=np.dot(activation,beta)
        E=E-y
        print("")
        beta_overall=np.append(beta_overall,beta,axis=0)
        predicted_accuracy=accuracy(y,Train_Label)
        print(predicted_accuracy,"with", L," Neuron ")
end_time=round(time.time()-start_time,3)
print("")
print("TRAINING TIME",end_time)
```

```
2020-10-30 22:02:51,031 - pyswarms.single.global_best - INFO - Optimize for 50 iters with {'c1': 0.5, 'c2': 0.3, 'w': 0.9}
pyswarms.single.global_best:   0%|          |0/50

-------- TRAINING ACCURACY ----------

49.714285714285715 With 1  Neuron

pyswarms.single.global_best: 100%|██████████|50/50, best_cost=-49.3
2020-10-30 22:02:59,501 - pyswarms.single.global_best - INFO - Optimization finished | best cost: -49.28571428571429, best pos: [0.96129926 0.43586493 0.09866023]
2020-10-30 22:02:59,512 - pyswarms.single.global_best - INFO - Optimize for 50 iters with {'c1': 0.5, 'c2': 0.3, 'w': 0.9}
pyswarms.single.global_best:   2%|█         |1/50, best_cost=-49.3

-49.28571428571429
[0.96129926 0.43586493 0.09866023]

50.28571428571429 with 2  Neuron

pyswarms.single.global_best: 100%|██████████|50/50, best_cost=-49.3
2020-10-30 22:03:05,525 - pyswarms.single.global_best - INFO - Optimization finished | best cost: -49.28571428571429, best pos: [-0.09939794  0.55941817  0.20307257]
2020-10-30 22:03:05,538 - pyswarms.single.global_best - INFO - Optimize for 50 iters with {'c1': 0.5, 'c2': 0.3, 'w': 0.9}
pyswarms.single.global_best:   2%|█         |1/50, best_cost=-49.3

-49.28571428571429
[-0.09939794  0.55941817  0.20307257]

50.28571428571429 with 3  Neuron

pyswarms.single.global_best: 100%|██████████|50/50, best_cost=-49.3
2020-10-30 22:03:11,436 - pyswarms.single.global_best - INFO - Optimization finished | best cost: -49.28571428571429, best pos: [0.95496218 0.00378372 0.34678914]
2020-10-30 22:03:11,449 - pyswarms.single.global_best - INFO - Optimize for 50 iters with {'c1': 0.5, 'c2': 0.3, 'w': 0.9}
pyswarms.single.global_best:   2%|█         |1/50, best_cost=-48.7

-49.28571428571429
[0.95496218 0.00378372 0.34678914]

49.714285714285715 with 4  Neuron

pyswarms.single.global_best: 100%|██████████|50/50, best_cost=-49.3
2020-10-30 22:03:17,205 - pyswarms.single.global_best - INFO - Optimization finished | best cost: -49.28571428571429, best pos: [ 0.89891057  0.09168554 -0.30484746]
2020-10-30 22:03:17,220 - pyswarms.single.global_best - INFO - Optimize for 50 iters with {'c1': 0.5, 'c2': 0.3, 'w': 0.9}
pyswarms.single.global_best:   2%|█         |1/50, best_cost=-49.3

-49.28571428571429
[ 0.89891057  0.09168554 -0.30484746]

49.714285714285715 with 5  Neuron

pyswarms.single.global_best: 100%|██████████|50/50, best_cost=-49.3
2020-10-30 22:03:22,859 - pyswarms.single.global_best - INFO - Optimization finished | best cost: -49.28571428571429, best pos: [ 0.62093105  0.26863987 -0.15717763]
2020-10-30 22:03:22,875 - pyswarms.single.global_best - INFO - Optimize for 50 iters with {'c1': 0.5, 'c2': 0.3, 'w': 0.9}
pyswarms.single.global_best:   2%|█         |1/50, best_cost=-49.3

-49.28571428571429
[ 0.62093105  0.26863987 -0.15717763]

50.28571428571429 with 6  Neuron

pyswarms.single.global_best: 100%|██████████|50/50, best_cost=-49.3
2020-10-30 22:03:29,021 - pyswarms.single.global_best - INFO - Optimization finished | best cost: -49.28571428571429, best pos: [-0.36633787  0.08672393 -0.5779917 ]
2020-10-30 22:03:29,036 - pyswarms.single.global_best - INFO - Optimize for 50 iters with {'c1': 0.5, 'c2': 0.3, 'w': 0.9}
pyswarms.single.global_best:   2%|█         |1/50, best_cost=-48.7

-49.28571428571429
[-0.36633787  0.08672393 -0.5779917 ]

50.28571428571429 with 7  Neuron

pyswarms.single.global_best: 100%|██████████|50/50, best_cost=-48.7
2020-10-30 22:03:34,952 - pyswarms.single.global_best - INFO - Optimization finished | best cost: -48.714285714285715, best pos: [-0.33066761 -0.70556427  0.75334801]
2020-10-30 22:03:34,964 - pyswarms.single.global_best - INFO - Optimize for 50 iters with {'c1': 0.5, 'c2': 0.3, 'w': 0.9}
pyswarms.single.global_best:   2%|█         |1/50, best_cost=-48.7

-48.714285714285715
[-0.33066761 -0.70556427  0.75334801]

49.714285714285715 with 8  Neuron

pyswarms.single.global_best: 100%|██████████|50/50, best_cost=-49.3
2020-10-30 22:03:40,892 - pyswarms.single.global_best - INFO - Optimization finished | best cost: -49.28571428571429, best pos: [-0.30815906 -0.91840134  0.70201206]
2020-10-30 22:03:40,906 - pyswarms.single.global_best - INFO - Optimize for 50 iters with {'c1': 0.5, 'c2': 0.3, 'w': 0.9}
pyswarms.single.global_best:   2%|█         |1/50, best_cost=-49.3

-49.28571428571429
[-0.30815906 -0.91840134  0.70201206]

49.714285714285715 with 9  Neuron

pyswarms.single.global_best: 100%|██████████|50/50, best_cost=-49.3
2020-10-30 22:03:47,053 - pyswarms.single.global_best - INFO - Optimization finished | best cost: -49.28571428571429, best pos: [-0.63682437 -0.04874794  0.95756228]

-49.28571428571429
[-0.63682437 -0.04874794  0.95756228]

50.28571428571429 with 10  Neuron

TRAINING TIME 56.053
```

In [10]:
```python
#Testing
print("-------- TESTING ACCURACY --------")
print("")
start_time=time.time()
h_test=np.dot(Test_features,new_a) #using same weights, bias and beta produced during training phase
sum_test=h_test+new_b
activation_test=sigmoid(sum_test)
y_test=np.dot(activation_test,beta_overall)
testing_accuracy=accuracy(y_test,Test_Label)

print(testing_accuracy,"with", L," neurons ")
end_time=round(time.time()-start_time,3)
print("")
print("TESTING TIME",end_time)
```

```
-------- TESTING ACCURACY --------

84.33333333333334 with 10  neurons

TESTING TIME 0.004
```

In [ ]:

In [ ]: