ECE 657 Assignment 2 – Group 80

Rumna Samanta – 20883387
rsamanta@uwaterloo.ca

Juhi Vasudev Bachani – 20979706
jvbachani@uwaterloo.ca

Frank Huang Huang – 20433010
g7huang@uwaterloo.ca

June 22, 2022

Problem 1: Deriving formulas for stochastic Gradient_Based method for training an RBF NN:

Problem 1

$$w(n+1) = w(n) - \mu_w \frac{\partial}{\partial w} J(n) \Big|_{w=w(n)}$$

$$\frac{\partial}{\partial w} J(n) = \frac{1}{2}\left[y_d(n) - \sum_{k=1}^{N} w_k(n)\, \phi\{x(n), c_k, \sigma_k\} \right]^2$$

Let $G_k(n)$ denotes $\phi\{x(n), c_k, \sigma_k\}$, then we will have:

$$\frac{\partial}{\partial w} J(n) = \frac{1}{2}\left[y_d(n) - \sum_{k=1}^{N} w_k(n)\, G_k(n) \right]^2$$

$$= \frac{1}{2}\cdot 2\left(y_d(n) - \sum_{k=1}^{N} w_k(n)\, G_k(n) \right)\cdot \frac{\partial}{\partial w}\left(y_d(n) - \sum_{k=1}^{N} w_k(n)\cdot G_k(n) \right)$$

$$= \left(y_d(n) - \sum_{k=1}^{N} W_k(n)\, G_k(n) \right)\left(0 - \sum_{k=1}^{N} G_k(n) \right)$$

$$= -\left(y_d(n) - \sum_{k=1}^{N} w_k(n)\, \phi\{x(n), c_k, \sigma_k\} \right)\cdot \sum_{k=1}^{N} \phi\{x(n), c_k, \sigma_k\}$$

$$= -e(n)\, \psi(n)$$

$$\therefore w(n+1) = w(n) - (\mu_w - e(n)\, \psi(n))$$

$$= w(n) + \mu_w\, e(n)\, \psi(n)$$

$$c_k(n+1) = c_k(n) - \mu_k \frac{\partial}{\partial c_k} J(n) \Big|_{c_k = c_k(n)}$$

$$J(n) = \frac{1}{2}\left[y_d(n) - \sum_{k=1}^{N} w_k(n) \exp\left(-\frac{\|x(n) - c_k(n)\|^2}{2\sigma_k^2(n)} \right) \right]^2$$

$$\frac{\partial J(n)}{\partial c_k} = \frac{1}{2}\cdot 2\left[y_d(n) - \sum_{k=1}^{N} w_k(n) \exp\left(-\frac{\|x(n) - c_k(n)\|^2}{2\sigma_k^2(n)} \right) \right] \cdot \left[0 - \frac{\partial}{\partial c_k} \sum_{k=1}^{N} W_k(n) \exp\left(-\frac{\|x(n) - c_k(n)\|^2}{2\sigma_k^2(n)} \right) \right]$$

$$= 1\cdot e(n)\left(-\frac{\partial}{\partial c_k} \sum_{k=1}^{N} w_k(n)\cdot \exp\left(-\frac{\|x(n) - c_k(n)\|^2}{2\sigma_k^2(n)} \right) \right)$$

$$\therefore w(n+1) = w(n) - (\mu_w - e(n)\psi(n))$$
$$= w(n) + \mu_w e(n)\psi(n)$$

$$c_k(n+1) = c_k(n) - \mu_c \frac{\partial}{\partial c_k} J(n) \Big|_{c_k = c_k(n)}$$

$$J(n) = \frac{1}{2}\left[y_d(n) - \sum_{k=1}^{N} w_k(n) \exp\left(-\frac{\|x(n)-c_k(n)\|^2}{2\sigma_k^2(n)}\right)\right]^2$$

$$\frac{\partial J(n)}{\partial c_k} = \frac{1}{2}\cdot 2\left[y_d(n) - \sum_{k=1}^{N} w_k(n) \exp\left(-\frac{\|x(n)-c_k(n)\|^2}{2\sigma_k^2(n)}\right)\right]$$
$$\cdot \left[0 - \frac{\partial}{\partial c_k} \sum_{k=1}^{N} w_k(n) \exp\left(-\frac{\|x(n)-c_k(n)\|^2}{2\sigma_k^2(n)}\right)\right]$$

$$= 1 \cdot e(n) \left(-\frac{\partial}{\partial c_k} \sum_{k=1}^{N} w_k(n) \cdot \exp\left(-\frac{\|x(n)-c_k(n)\|^2}{2\sigma_k^2(n)}\right)\right)$$

$$= - e(n) \cdot w_k(n) \frac{\partial}{\partial c_k} \exp\left(-\frac{\|x(n)-c_k(n)\|^2}{2\sigma_k^2(n)}\right)$$

$$= - e(n) \cdot w_k(n) \exp\left(-\frac{\|x(n)-c_k(n)\|^2}{\sigma_k^2(n)}\right) \cdot \left(\frac{\partial}{\partial c_k} - \frac{(x(n)-c_k(n))^2}{2\sigma_k^2(n)}\right)$$

$$= - e(n) \cdot w_k(n)\, \phi\{x(n), c_k(n), \sigma_k\} \left(\cancel{-x^2}\, 2x\right.$$
$$\cdot \left(\frac{\partial}{\partial c_k} - \frac{x^2(n) - 2x(n)c_k(n) + c_k(n)^2}{2\sigma_k^2(n)}\right)$$

$$= - e(n) \cdot w_k(n)\, \phi\{x(n), c_k(n), \sigma_k\} \cdot \left(\frac{-2x(n) + 2c_k(n)}{2\sigma_k^2(n)}\right)$$

$$= \frac{e(n) \cdot w_k(n)}{\sigma_k^2(n)} \phi\{x(n), c_k(n), \sigma_k\} \cdot (x(n) + c_k(n))$$

$$= -e(n)\cdot w_k(n)\frac{\partial}{\partial c_k}\exp\left(-\frac{\|x(n)-c_k(n)\|^2}{2\sigma_k^2(n)}\right)$$

$$= -e(n)\cdot w_k(n)\exp\left(-\frac{\|x(n)-c_k(n)\|^2}{\sigma_k^2(n)}\right)\cdot\left(\frac{\partial}{\partial c_k}-\frac{(x(n)-c_k(n))^2}{2\sigma_k^2(n)}\right)$$

$$= -e(n)\cdot w_k(n)\,\phi\{x(n), c_k(n), \sigma_k\}\,(\cancel{-x^2-2x}$$

$$\cdot\left(\frac{\partial}{\partial c_k}-\frac{x^2(n)-2x(n)c_k(n)+c_k(n)^2}{2\sigma_k^2(n)}\right)$$

$$= -e(n)\cdot w_k(n)\phi\{x(n), c_k(n), \sigma_k\}\cdot\left(\frac{-2x(n)+2c_k(n)}{2\sigma_k^2(n)}\right)$$

$$= \frac{e(n)\cdot w_k(n)}{\sigma_k^2(n)}\phi\{x(n), c_k(n), \sigma_k\}\cdot(x(n)+c_k(n))$$

$$\therefore\ c_k(n+1) = c_k(n) - \mu_c\cdot\frac{e(n)\cdot w_k(n)}{\sigma_k^2(n)}\phi\{x(n), c_k(n), \sigma_k\}(x(n)+c_k(n))$$

$$\sigma_k(n+1) = \sigma_k(n) - \mu_\sigma\frac{\partial}{\partial\sigma_k}J(n)\Big|_{\sigma_k=\sigma_k(n)}$$

$$\frac{\partial}{\partial\sigma_k}J(n) = \frac{1}{2}\cdot 2\left[y_d(n) - \sum_{k=1}^{N} w_k(n)\exp\left(-\frac{\|x(n)-c_k(n)\|^2}{2\sigma_k^2(n)}\right)\right]$$

$$\cdot\left[0 - \frac{\partial}{\partial\sigma_k}\sum_{k=1}^{N} w_k(n)\exp\left(-\frac{\|x(n)-c_k(n)\|^2}{\sigma_k^2(n)}\right)\right]$$

$$= 1\cdot e(n)\left(-\frac{\partial}{\partial\sigma_k}\sum_{k=1}^{N} w_k(n)\cdot\exp\left(-\frac{\|x(n)-c_k(n)\|^2}{\sigma_k^2(n)}\right)\right)$$

$$= -e(n)\,w_k(n)\cdot\exp\left(-\frac{\|x(n)-c_k(n)\|^2}{\sigma_k^2(n)}\right)$$

$$= -e(n)\cdot w_k(n)\cdot\exp\left(-\frac{\|x(n)-c_k(n)\|^2}{\sigma_k^2(n)}\right)\cdot\left(\frac{\partial}{\partial\sigma_k}-\frac{\|x(n)-c_k(n)\|^2}{\sigma_k^2(n)}\right)$$

$$= \cancel{-e(n)\cdot w_k(n)\phi\{x(n), c_k(n), \sigma_k\}\cdot\left(\frac{\partial}{\partial\sigma_k}-\frac{x^2(n)-2x(n)c_k(n)+c_k(n)^2}{\sigma_k^2(n)}\right)}$$

$$= -e(n)\cdot w_k(n)\cdot\exp\left(-\frac{\|x(n)-c_k(n)\|^2}{\sigma_k^2(n)}\right)\cdot\left(\frac{\|x(n)-c_k(n)\|^2}{2\sigma_k^3(n)}\right)$$

$$\sigma_k(n+1) = \sigma_k(n) - \mu_\sigma \frac{\partial}{\partial \sigma_k} J(n) \Big|_{\sigma_k = \sigma_k(n)}$$

$$\frac{\partial}{\partial \sigma_k} J(n) = \frac{1}{2} \cdot 2 \left[y_d(n) - \sum_{k=1}^{N} w_k(n) \exp\left(- \frac{\| x(n) - c_k(n) \|^2}{2\sigma_k^2(n)} \right) \right]$$

$$\cdot \left[0 - \frac{\partial}{\partial \sigma_k} \sum_{k=1}^{N} w_k(n) \exp\left(- \frac{\| x(n) - c_k(n) \|^2}{\sigma_k^2(n)} \right) \right]$$

$$= 1 \cdot e(n) \left(- \frac{\partial}{\partial \sigma_k} \sum_{k=1}^{N} w_k(n) \cdot \exp\left(- \frac{\| x(n) - c_k(n) \|^2}{\sigma_k^2(n)} \right) \right)$$

$$= - e(n)\, w_k(n) \cdot \exp\left(- \frac{\| x(n) - c_k(n) \|^2}{\sigma_k^2(n)} \right)$$

$$= - e(n) \cdot w_k(n) \cdot \exp\left(- \frac{\| x(n) - c_k(n) \|^2}{\sigma_k^2(n)} \right) \cdot \left(\frac{\partial}{\partial \sigma_k} - \frac{\| x(n) - c_k(n) \|^2}{\sigma_k^2(n)} \right)$$

$$= \cancel{- e(n) \cdot w_k(n)\, \phi\{ x(n), c_k(n), \sigma_k \} \cdot \left(\frac{\partial}{\partial \sigma_k} - \frac{x^2(n) - 2x(n)c_k(n) + c_k(n)^2}{\sigma_k^2(n)} \right)}$$

$$= - e(n) \cdot w_k(n) \cdot \exp\left(- \frac{\| x(n) - c_k(n) \|^2}{\sigma_k^2(n)} \right) \cdot \left(\frac{\| x(n) - c_k(n) \|^2}{2\sigma_k^3(n)} \right)$$

$$= - \frac{e(n) \cdot w_k(n)}{2\sigma_k^3(n)} \phi\{ x(n), c_k(n), \sigma_k \} \cdot \| x(n) - c_k(n) \|^2$$

$$\therefore \sigma_k(n+1) = \sigma_k(n) + \mu_\sigma \frac{e(n) \cdot w_k(n)}{2\sigma_k^3(n)} \phi\{ x(n), c_k(n), \sigma_k \} \| x(n) - c_k(n) \|^2$$

Problem 3: Python Code that creates an RBF Network

```
# Import the required libraries
import numpy as np
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt
```

```
# Define a function to split the train and test data in to 80, 20 ratio
def split_data(data=None):
  train=80
  test=20
  rows, cols= data.shape
  train_index=int((train/100)*rows)
  train_data=data[:train_index]
  test_data=data[train_index:]
  return train_data, test_data
```

```
# Function to generate training set as given in the assignment
def training_set(i_range=21,j_range=21):
    res=list()
    data=np.array([[-2+0.2*i, -2+0.2*j] for i in range(i_range) for j in range(j_range)])
    for x in data:
      if x[0]**2+x[1]**2<=1:
        res.append([x[0],x[1],1])
      elif x[0]**2+x[1]**2>1:
        res.append([x[0],x[1],-1])

    return np.array(res)
```

```python
# Print the training set generated
training_set()
```

```
array([[-2. , -2. , -1. ],
       [-2. , -1.8, -1. ],
       [-2. , -1.6, -1. ],
       ...,
       [ 2. ,  1.6, -1. ],
       [ 2. ,  1.8, -1. ],
       [ 2. ,  2. , -1. ]])
```

```python
# Function to generate X,Y from a given dataset
def generate_X_Y(data):
  rows, columns = data.shape
  X = data[:rows, :-1]
  Y = data[:rows, -1]
  return X , Y
```

```python
# Function to calculate accuracy
def calculate_acc(y, y_pred):
  acc=np.mean(y == y_pred)*100
  return acc
```

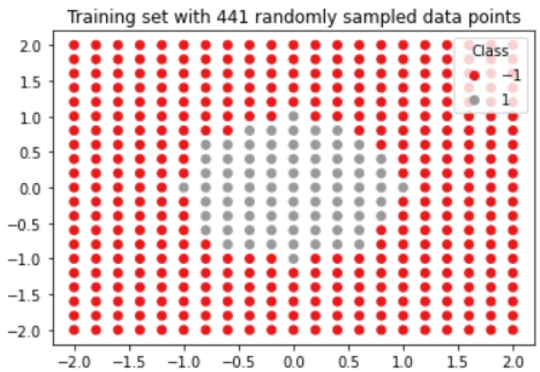
```python
# Fucntion to calculate mean square error
def calculate_mean_sqr_error(y, y_pred):
  m=np.mean((y-y_pred)**2)
  return m
```

```
[ ]  # Class that includes implemention of RBF Neural Network
     class RBFNeuralNetwork:
       def __init__(self, spread=0):
         self._spread_method=spread
         self._centers=None
         self._weights=None

       def spread_getter(self):
         return self._spread_method

       def spread_setter(self,spread):
         self._spread_method=spread

       def centers_getter(self):
         return self._centers

       def centers_setter(self,centers):
         self._centers=centers

       def weights_getter(self):
         return self._weights

       def weights_setter(self, weights):
         self._weights=weights

       def fit(self, x, y, kmeans_param=False, random_center_param=False):
         if not random_center_param and not kmeans_param:
           self.centers_setter(x)
         else:
           if kmeans_param:
             kmeans_model=KMeans(n_clusters=150, random_state=0)
             self.centers_setter(kmeans_model.fit(x).cluster_centers_)
```

```
def weights_setter(self, weights):
  self._weights=weights

def fit(self, x, y, kmeans_param=False, random_center_param=False):
  if not random_center_param and not kmeans_param:
    self.centers_setter(x)
  else:
    if kmeans_param:
      kmeans_model=KMeans(n_clusters=150, random_state=0)
      self.centers_setter(kmeans_model.fit(x).cluster_centers_)
    else:
      self.centers_setter(x[np.random.choice(x.shape[0],150)])

  G=self.activation_func(x)
  self.weights_setter(np.dot(np.linalg.pinv(G),y))

def predict(self, x):
  G=self.activation_func(x)
  return np.sign(np.dot(G, self.weights_getter()))

# Gaussian  Kernel function. Here x is the input vector and center is the center of thr radial function

def kernel(self, x, center):
  s=2*self.spread_getter()**2
  d=np.linalg.norm(x- center) ** 2
  r=np.exp(- d ** 2 / s)
  return r

def activation_func(self, x):
  g=np.zeros((x.shape[0],self.centers_getter().shape[0]))
  for i in range(x.shape[0]):
```

```
# Gaussian  Kernel function. Here x is the input vector and center is the center of thr radial function

def kernel(self, x, center):
  s=2*self.spread_getter()**2
  d=np.linalg.norm(x- center) ** 2
  r=np.exp(- d ** 2 / s)
  return r

def activation_func(self, x):
  g=np.zeros((x.shape[0],self.centers_getter().shape[0]))
  for i in range(x.shape[0]):
    for j in range(self.centers_getter().shape[0]):
      g[i,j] = self.kernel(x[i], self.centers_getter()[j])
  return g
```

```
[ ]  # Generating a pictorial representation of the training set with 441 randomly sampled data points

     training_data=training_set()

     model_plot=plt.scatter(training_data[:,0],training_data[:,1], c=training_data[:,2], cmap='Set1')
     plt.legend(*model_plot.legend_elements(), title="Class")
     plt.title("Training set with 441 randomly sampled data points")
     plt.show()
```



Part 1: Carry out the design of RBF NN based on Gaussian kernel functions with constant spread function and using all the points in the training set as centers of the RB functions. Compare the performance results (mean square error) as you vary the spread parameter while keeping it the same for all kernel functions

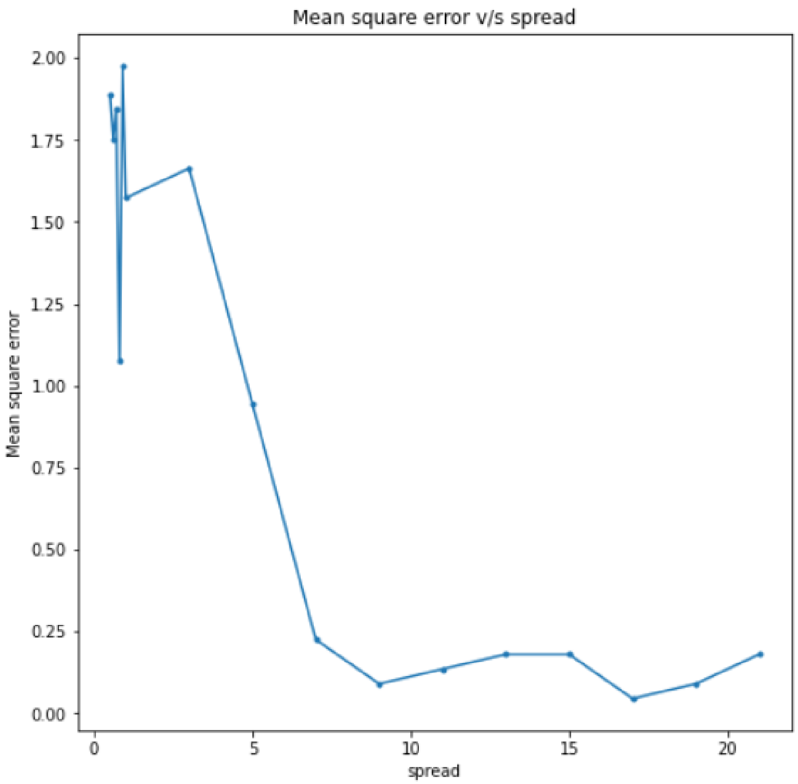
```
[ ]  # Defining a range of constant sigmas
     sigmas=[0.5,0.6,0.7,0.8,0.9,1,3,5,7,9,11,13,15,17,19,21]

     # RBF Neural Network
     rbfNeuralNetwork = RBFNeuralNetwork()

     # Dictionary to store the accuracy values and mean square error of the model
     acc_dict ={
         "train_acc": list(),
         "test_acc": list(),
         "error": list()
     }

     for s in sigmas:
       rbfNeuralNetwork.spread_setter(s)
       np.random.shuffle(training_data)
       train_data, test_data =split_data(training_data)

       X_train, Y_train = generate_X_Y(train_data)
       X_test, Y_test = generate_X_Y(test_data)
```

```
for s in sigmas:
  rbfNeuralNetwork.spread_setter(s)
  np.random.shuffle(training_data)
  train_data, test_data =split_data(training_data)

  X_train, Y_train = generate_X_Y(train_data)
  X_test, Y_test = generate_X_Y(test_data)

  rbfNeuralNetwork.fit(X_train, Y_train)
  pred=rbfNeuralNetwork.predict(X_train)
  acc_dict["train_acc"].append(calculate_acc(X_train, pred))

  pred=rbfNeuralNetwork.predict(X_test)
  acc_dict["test_acc"].append(calculate_acc(Y_test, pred))
  acc_dict["error"].append(calculate_mean_sqr_error(Y_test, pred))
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:3: DeprecationWarning: elementwise comparison failed; this will raise an error in the future.
  This is separate from the ipykernel package so we can avoid doing imports until
```

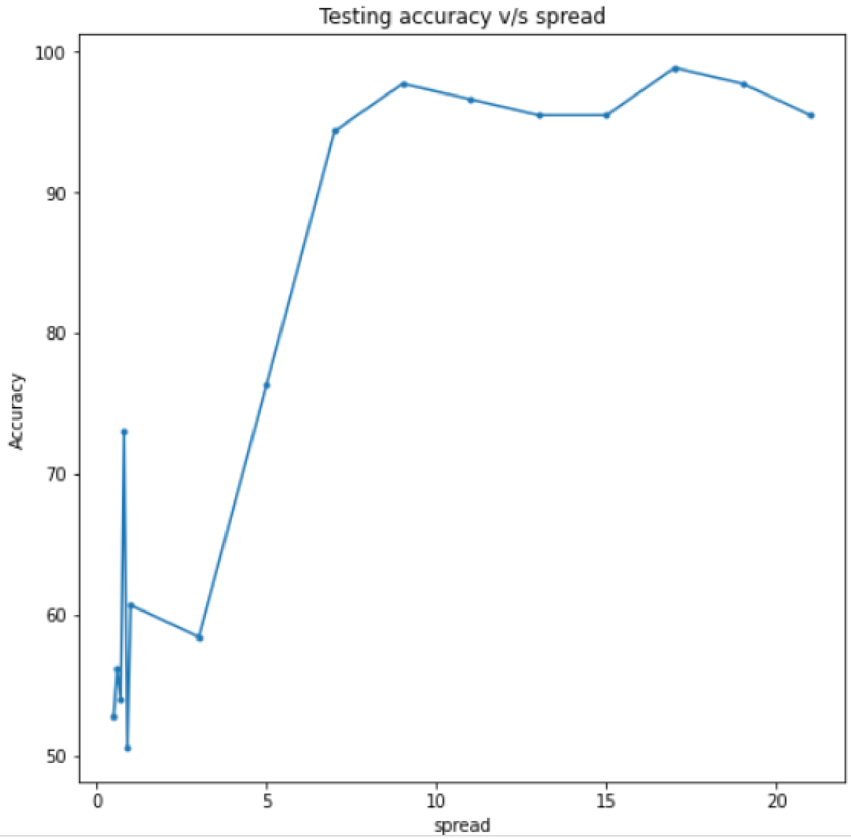
```python
figure, img1 = plt.subplots(figsize = (8,8))
img1.set_title("Mean square error v/s spread")
img1.plot(sigmas, acc_dict["error"],marker='.',)
img1.set_xlabel("spread")
img1.set_ylabel("Mean square error")

print("The lowest mean square error is {0}".format(min(acc_dict["error"])))
plt.show()
```

```
The lowest mean square error is 0.0449438202247191
```



```
[ ] figure, img2 = plt.subplots(figsize = (8,8))
    img2.set_title("Testing accuracy v/s spread")
    img2.plot(sigmas, acc_dict["test_acc"],marker='.',)
    img2.set_xlabel("spread")
    img2.set_ylabel("Accuracy")

    print("The Maximum accuracy is {0}".format(max(acc_dict["test_acc"])))
    plt.show()
```

```
The Maximum accuracy is 98.87640449438202
```



Comparing both these plot images we can say that the maximum accuracy which is 98.8% is achieved when the spread is 17. Also, at that spread value, the mean square error is the lowest which is 0.044. Thus, we can conclude that mean sqaure error vallue changes as the spread changes.

Also, when the spread values are largely spaced then then the data points are described very well. However, if we keep increasing the spread parameters then then different centers of the kernel functions would overlap. This would results in increasing the mean square error value.

Part 2: Perform the design of the RBF NN, using this time only 150 centers, choosing the centers using two approaches:

a) Randomly select the centers from the input data.

```
[ ] # RBF Neural Network
    rbf_part2_a=RBFNeuralNetwork()

    # Set the spread value to 15
    rbf_part2_a.spread_setter(15)
    data_part2_a=training_set()
    np.random.shuffle(data_part2_a)

    # Split the data
    train_data_part2_a, test_data_part2_a = split_data(data_part2_a)

    # Get X_train, X_test, Y_train, Y_test
    X_train, Y_train = generate_X_Y(train_data_part2_a)
    X_test, Y_test = generate_X_Y(test_data_part2_a)
```

```
[ ] # RBF Neural Network
    rbf_part2_a=RBFNeuralNetwork()

    # Set the spread value to 15
    rbf_part2_a.spread_setter(15)
    data_part2_a=training_set()
    np.random.shuffle(data_part2_a)

    # Split the data
    train_data_part2_a, test_data_part2_a = split_data(data_part2_a)

    # Get X_train, X_test, Y_train, Y_test
    X_train, Y_train = generate_X_Y(train_data_part2_a)
    X_test, Y_test = generate_X_Y(test_data_part2_a)

    # Fit the model and calculate accuracy
    rbf_part2_a.fit(X_train, Y_train, kmeans_param = False, random_center_param = True)
    pred_part2_a=rbf_part2_a.predict(X_train)
    part2_a_acc_train=calculate_acc(Y_train, pred_part2_a)

    pred_part2_a=rbf_part2_a.predict(X_test)
    part2_a_acc_test=calculate_acc(Y_test, pred_part2_a)
    part2_a_random_error=calculate_mean_sqr_error(Y_test,pred_part2_a)

    print("For Random center selection model the Accuracy is {0} and mean squared error is {1} ".format(part2_a_acc_test,part2_a_random_error))

    For Random center selection model the Accuracy is 96.62921348314607 and mean squared error is 0.1348314606741573
```

Part 2: b) Use K-Means algorithm to find the centers.

```
[ ] # RBF Neural Network
    rbf_part2_b=RBFNeuralNetwork()

    # Set the spread value to 15
    rbf_part2_b.spread_setter(15)
    data_part2_b=training_set()
    np.random.shuffle(data_part2_b)

    # Split the data
    train_data_part2_b, test_data_part2_b = split_data(data_part2_b)

    # Get X_train, X_test, Y_train, Y_test
    X_train, Y_train = generate_X_Y(train_data_part2_b)
    X_test, Y_test = generate_X_Y(test_data_part2_b)

    # Fit the model and calculate accuracy
    rbf_part2_b.fit(X_train, Y_train, kmeans_param = True, random_center_param = True)
    pred_part2_b=rbf_part2_b.predict(X_train)
    part2_b_acc_train=calculate_acc(Y_train, pred_part2_b)

    pred_part2_b=rbf_part2_b.predict(X_test)
    part2_b_acc_test=calculate_acc(Y_test, pred_part2_b)
    part2_b_random_error=calculate_mean_sqr_error(Y_test,pred_part2_b)

    print("For KMeans model the Accuracy is {0} and mean squared error is {1} ".format(part2_b_acc_test,part2_b_random_error))
```

```
For KMeans model the Accuracy is 98.87640449438202 and mean squared error is 0.0449438202247191
```

Conclusion: From the above accuracy results we can see that random selection model (Accuracy: 98.8%) performs better than the model prepared in part B_a (Accuracy: 96.6%).

Also, if we find the centers by using KMeans algorithm then the RBF model performs well (Accuracy: 98.8%).

So, we can say that random selection model and KMeans model are able to define boundaries very well and these models are least affected by the sigma values. However, considering all the input data as centers, the range of sigma values played an important role.

Problem 4: Design a Kohonen self organizing map (SOM)

```
#Importing Libraries
import numpy as np
import matplotlib.pyplot as plt
```

```
[ ] #values given in question
    Total_epochs = 1000
    Alpha_0 = 0.8
    Epoch_list = [20,40,100,1000]
    Sigmas = [1, 10, 30, 50, 70]
    size = 100
```

We are using 24 colors some shades of red, green, blue, with some yellow, teal and pink

For Red - Red, Tomato, Salmon, Maroon

For Green - Lime, Green, Dark Green, Pale Green

For Blue - Blue, Royal Blue, Navy, Steel blue

For Yellow - Yellow, Olive, Khaki, Yellow Green

For Teal - Teal, DarkCyan, Cyan

For Pink - Pink, HotPink, DeepPink, Mediumvioletred, Palevioletred

```
[ ] #Training data consist of 24 colors. It is selected from the "RGB Color Table: Basic Colors" section of this page: https://www.rapidtables.com/web/color/RGB_Color.html
    def generate_colors():
        colors = np.array(
            [(255,0,0),(255,99,71),(250,128,144),(128,0,0),
              (0,255,0),(0,128,0),(0,100,0),(152,251,152),
              (0,0,255),(65,105,225),(0,0,128),(70,130,180),
              (255,255,0),(128,128,0),(240,230,140),(154,205,50),
              (0,128,128),(0,139,139),(0,255,255),
              (255,192,203),(255,105,180),(255,20,147),(199,21,133),(219,112,147)]
            )
        #calibrate the color codes to values between 0 and 1, instead of being between 0 and 255)
        return colors/255
```

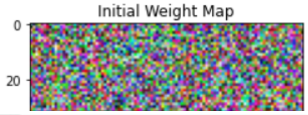
```
[ ] #Printing the calibrated input
    generate_colors()
```

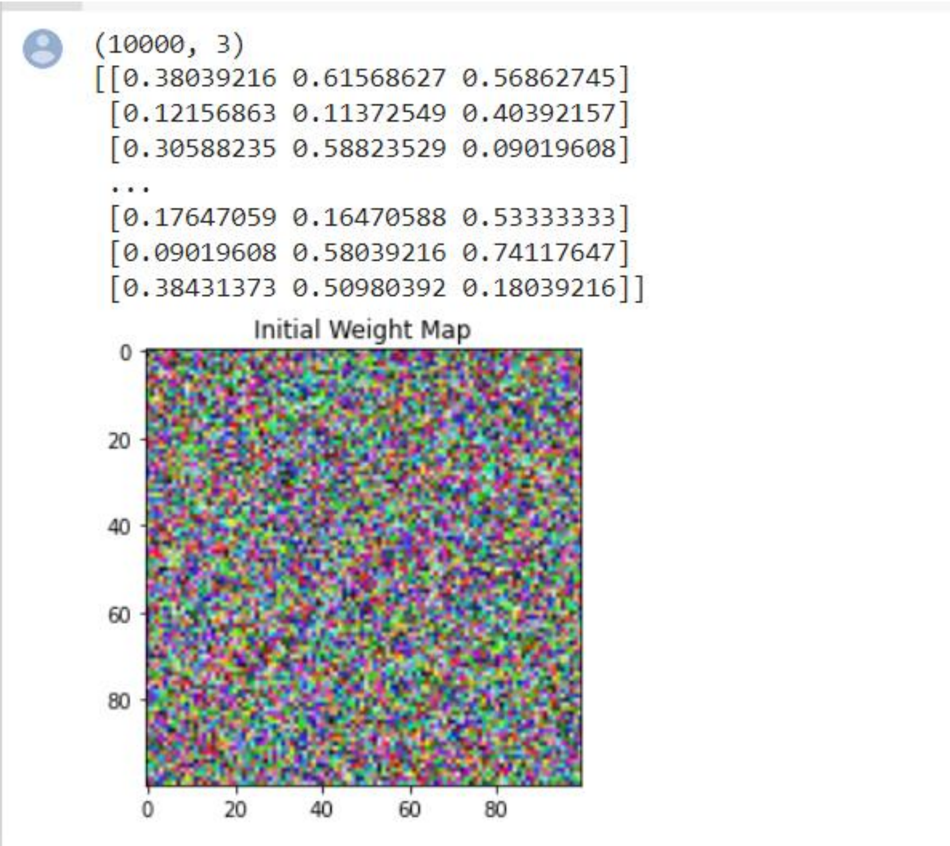
```
array([[1.        , 0.        , 0.        ],
       [1.        , 0.38823529, 0.27843137],
       [0.98039216, 0.50196078, 0.56470588],
       [0.50196078, 0.        , 0.        ],
       [0.        , 1.        , 0.        ],
       [0.        , 0.50196078, 0.        ],
       [0.        , 0.39215686, 0.        ],
       [0.59607843, 0.98431373, 0.59607843],
       [0.        , 0.        , 1.        ],
       [0.25490196, 0.41176471, 0.88235294],
       [0.        , 0.        , 0.50196078],
       [0.2745098 , 0.50980392, 0.70588235],
       [1.        , 1.        , 0.        ],
       [0.50196078, 0.50196078, 0.        ],
       [0.94117647, 0.90196078, 0.54901961],
       [0.60392157, 0.80392157, 0.19607843],
       [0.        , 0.50196078, 0.50196078],
       [0.        , 0.54509804, 0.54509804],
       [0.        , 1.        , 1.        ],
       [1.        , 0.75294118, 0.79607843],
       [1.        , 0.41176471, 0.70588235],
       [1.        , 0.07843137, 0.57647059],
       [0.78039216, 0.08235294, 0.52156863],
       [0.85882353, 0.43921569, 0.57647059]])
```

SOM is a data visualization technique and is unsupervised Learning. It will give idea about changes with reduction of dimensionality.

[ ]
```
# randomly initialized weights
weights = np.empty([size*size,3])

for i in range(0,size*size):
    weights[i][0] = np.random.randint(0,256)/255
    weights[i][1] = np.random.randint(0,256)/255
    weights[i][2] = np.random.randint(0,256)/255

print(weights.shape)
print(weights)
plt.title("Initial Weight Map")
plt.imshow(weights.reshape(size,size,3))
plt.show()
```

```
(10000, 3)
[[0.38039216 0.61568627 0.56862745]
 [0.12156863 0.11372549 0.40392157]
 [0.30588235 0.58823529 0.09019608]
 ...
 [0.17647059 0.16470588 0.53333333]
 [0.09019608 0.58039216 0.74117647]
 [0.38431373 0.50980392 0.18039216]]
```





```
(10000, 3)
[[0.38039216 0.61568627 0.56862745]
 [0.12156863 0.11372549 0.40392157]
 [0.30588235 0.58823529 0.09019608]
 ...
 [0.17647059 0.16470588 0.53333333]
 [0.09019608 0.58039216 0.74117647]
 [0.38431373 0.50980392 0.18039216]]
```

```
[ ] #Defining SOMModel class

    class SOMModel:
      def __init__(self, sigma):
        self.sigma = sigma
        # Initialize random weights and calibrate between 0 and 1.
        # Neuron grid is 100 * 100 => [10000, 2], so weight vector will be [10000, 3]
        self.wt = np.random.randint(0, 256, np.array([10000, 3]))/255
        self.x_vector, self.y_vector = np.meshgrid(np.linspace(0, size - 1, size), np.linspace(0, size - 1, size))
        #self.coordinates = np.c_[self.xv.ravel(), self.yv.ravel()]
        self.coords= np.c_[self.x_vector.ravel(), self.y_vector.ravel()]
        self.ax = None

      def sigma_setter(self, sigma):
        self.sigma = sigma

      def ax_setter(self, axes):
        # Flattening to 1 dimension.
        self.ax = axes.flatten()

      def set_initial_weight(self):
        self.wt = weights

      """
      N = exp (-d_i,j ^2 / 2 sigma ^ 2 (k))
      """
```


```
  """
  N = exp (-d_i,j ^2 / 2 sigma ^ 2 (k))
  """
```

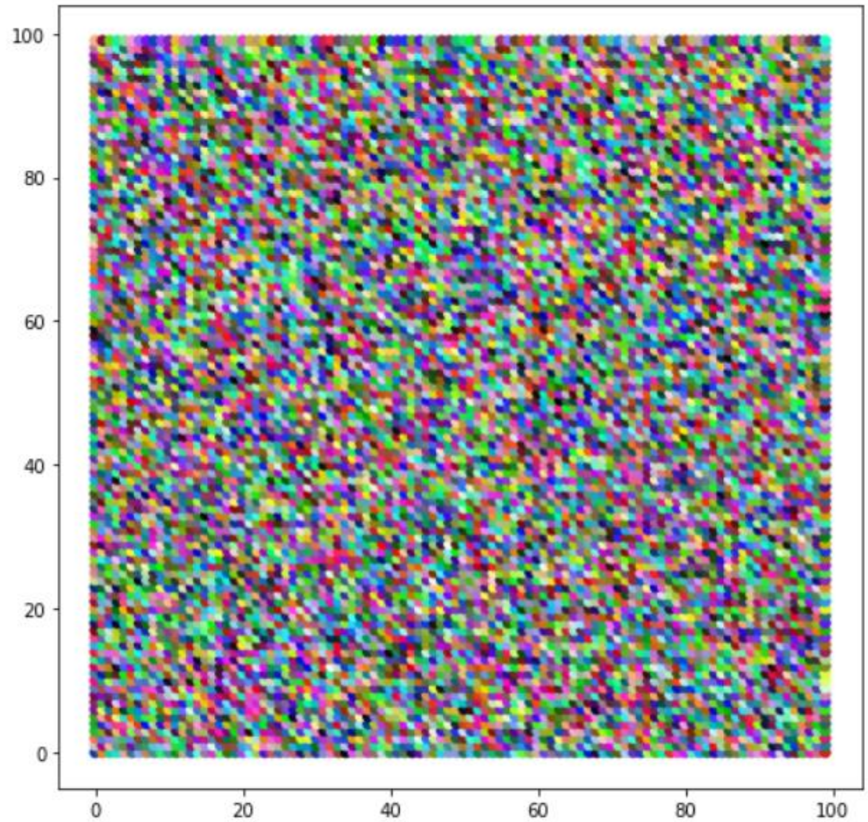

```
#Defining Neighbourhood
  def neighbour(self, k, winner):
    e = np.exp(-k/1000)
    sigma_new = self.sigma * e
    winner_neuron_coordinates = (int(winner / 100), winner % 100)
    #Using Euclidean equation to find the distance
    distance = np.linalg.norm(self.coords - winner_neuron_coordinates, axis = 1) ** 2
    return np.exp(- distance/ (2 * (sigma_new**2)))

  def model_plot(self, axis, ep):
    res = False
    if ep in Epoch_list:
      res = True
      self.ax[axis].scatter(self.x_vector.ravel(), self.y_vector.ravel(), c = self.wt, s = 20)
      self.ax[axis].set_title('Plot for Epoch {0}'.format(ep))
    return res
```

```python
def train(self, ip):
  a_index = 0
  lr = Alpha_0
  for ep in range(1, Total_epochs + 1):

    for x in ip:
      winner_neuron = np.argmin(np.linalg.norm(x - self.wt, axis = 1))
      # Get the neighbourhood
      neighbour = self.neighbour(ep, winner_neuron)
      # Updating the weights according to wij(k+ 1) =  wij(k) +(k)[x - wij(k)]
      self.wt += lr * neighbour.reshape(-1, 1) * (x - self.wt)

    if self.model_plot(a_index, ep):
      a_index +=1

    # Reduce the learning rate.  l(k) = l0 exp(-k/t)
    lr = Alpha_0 * np.exp(-ep/Total_epochs)
```

```python
[ ] generated_colors = generate_colors()
    SomModel = SOMModel(1)
    plot_array = list()

    fig, img = plt.subplots(figsize = (8, 8))
    fig.suptitle('Random weights', fontsize = 10)
    img.scatter(SomModel.x_vector.ravel(), SomModel.y_vector.ravel(), s = 30, c = SomModel.wt)
    plot_array.append([fig, img])

    for i, s in enumerate(Sigmas):
      figure, img1 = plt.subplots(2, 2, figsize = (10, 10))
      figure.suptitle('Sigma {0}'.format(s), fontsize = 20)
      plot_array.append([figure, img1])

      SomModel.sigma_setter(s)
      SomModel.set_initial_weight()
      SomModel.ax_setter(img1)
      SomModel.train(generated_colors)
    plt.show()
```

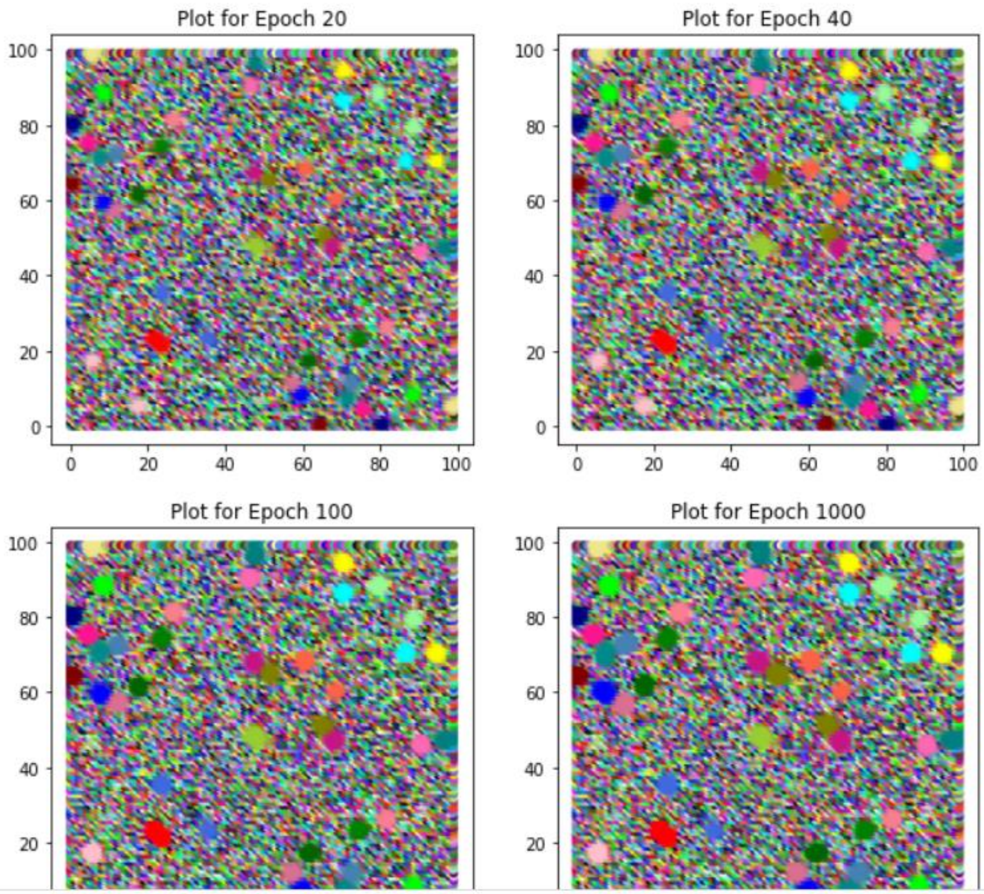
[ ]

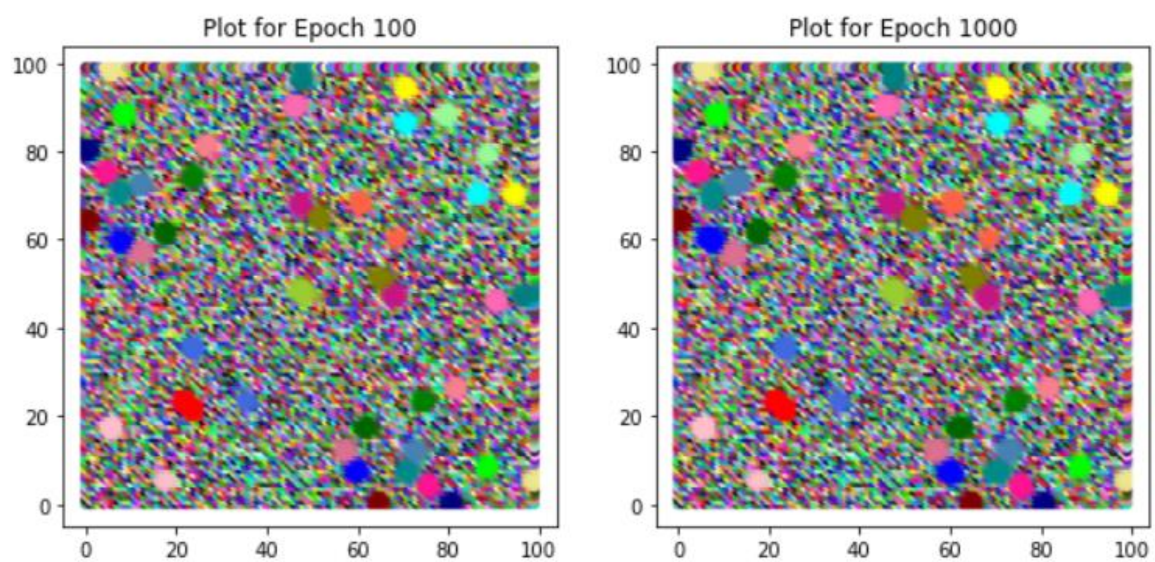
Random weights



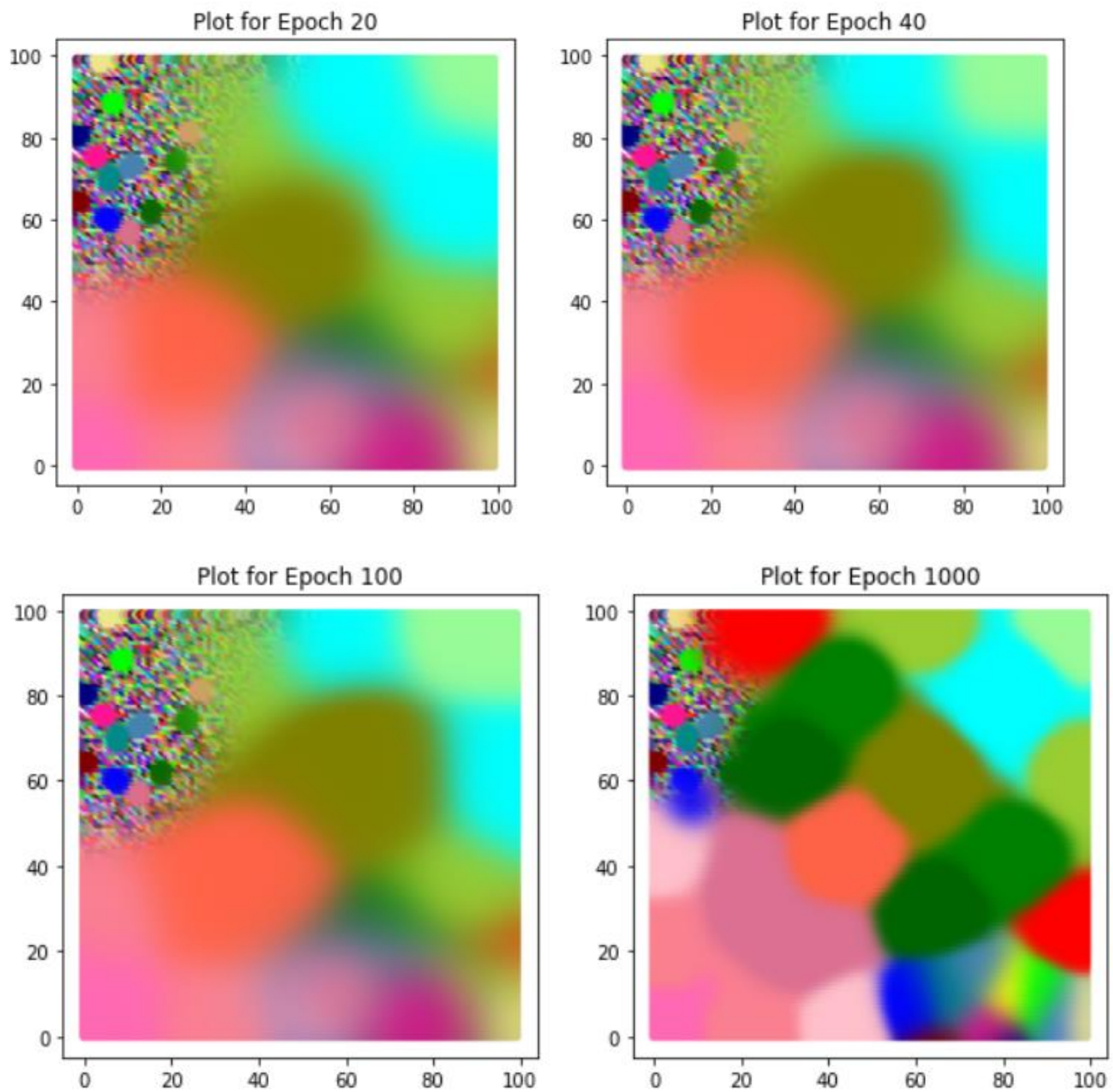
0 20 40 60 80 100

Sigma 1



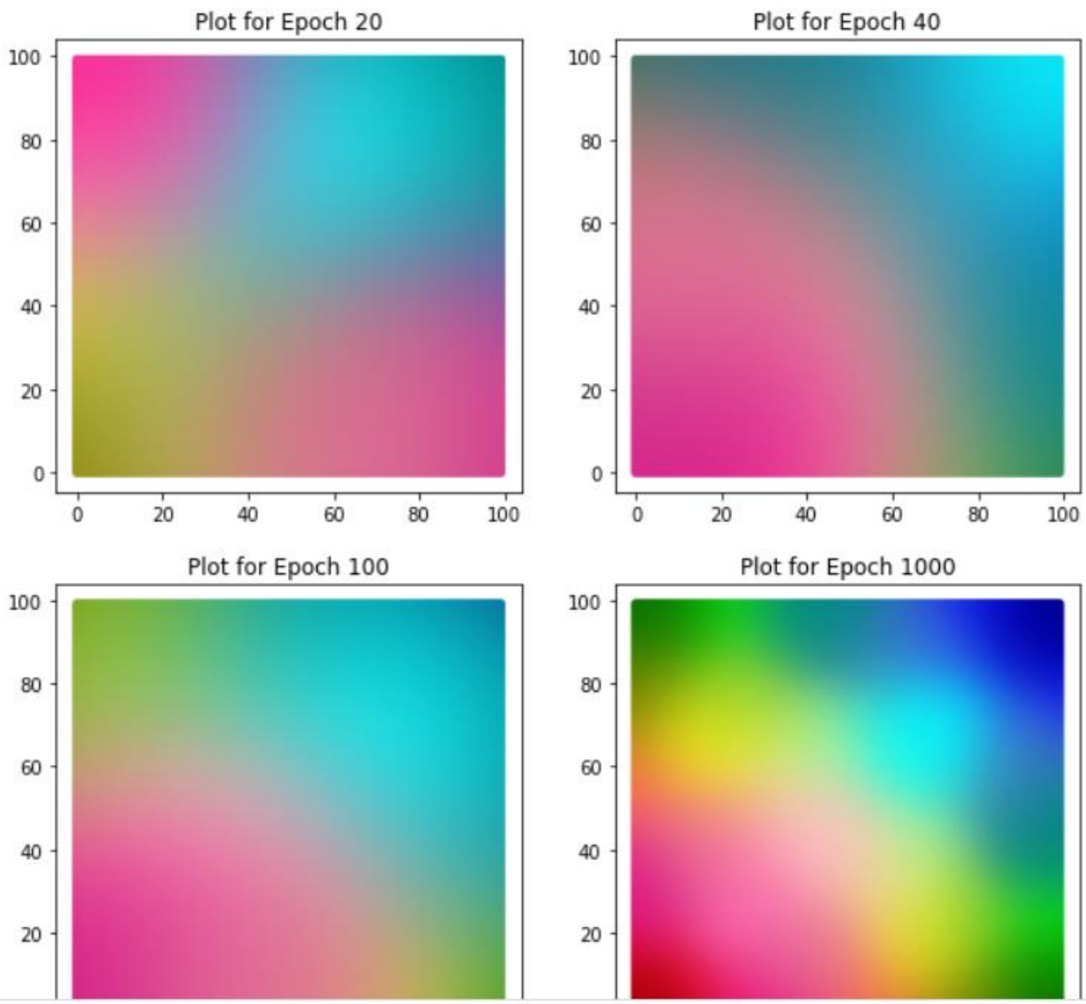


Sigma 10

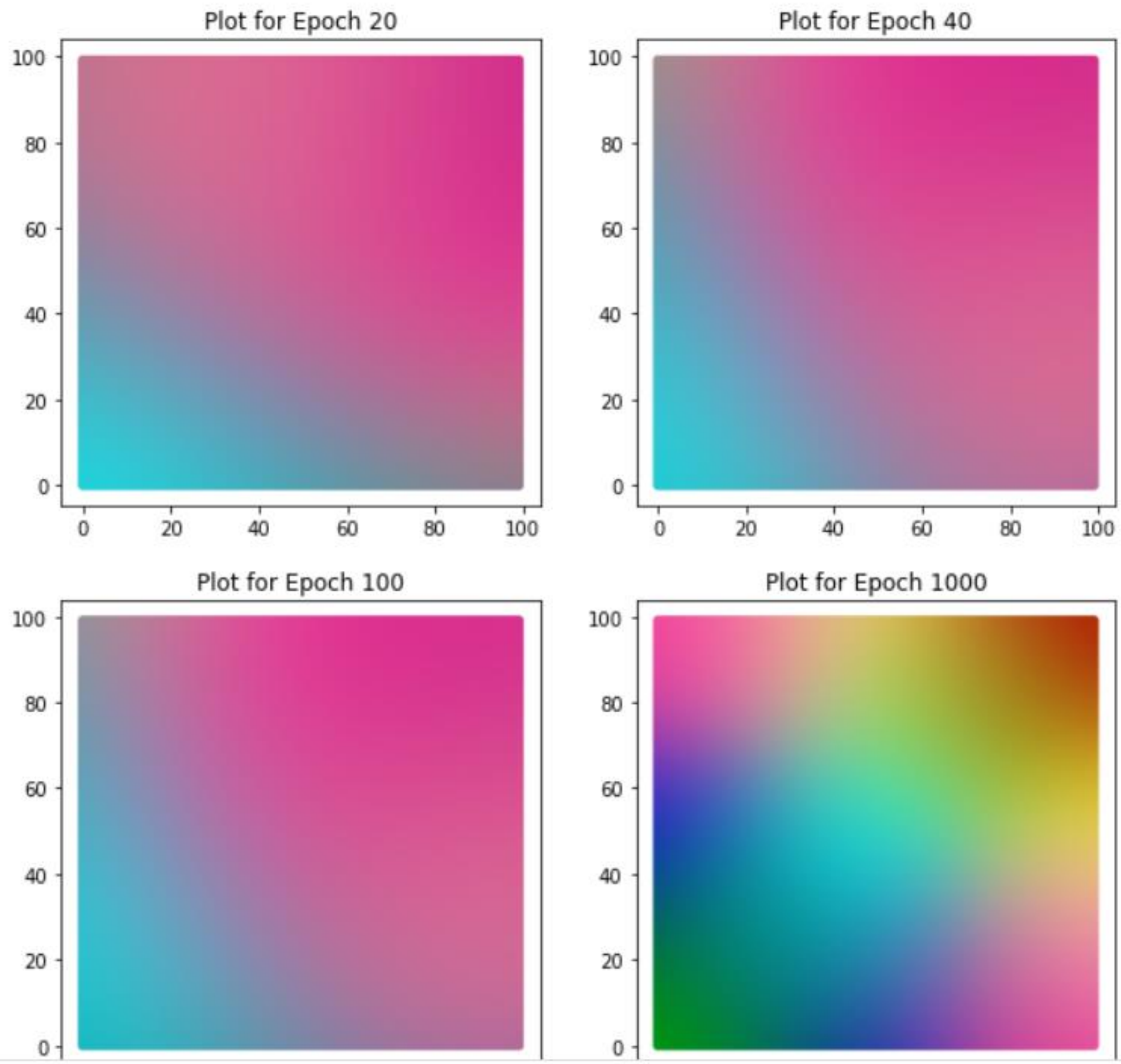


Plot for Epoch 20
Plot for Epoch 40
Plot for Epoch 100
Plot for Epoch 1000
0
20
40
60
80
100

Sigma 30

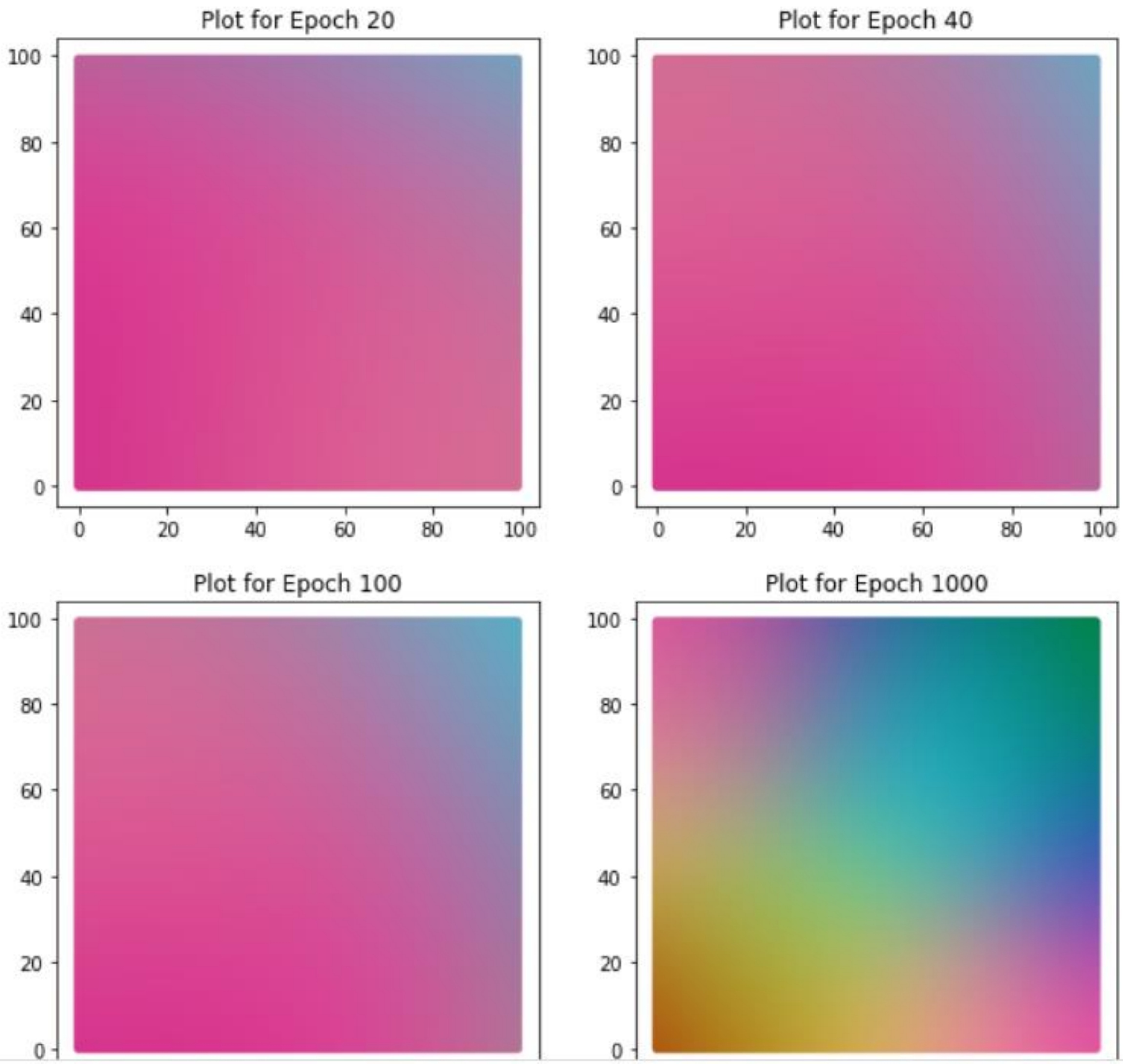

Plot for Epoch 20
Plot for Epoch 40
Plot for Epoch 100
Plot for Epoch 1000
0
20
40
60
80
100

Sigma 50



Plot for Epoch 20
Plot for Epoch 40
Plot for Epoch 100
Plot for Epoch 1000
0
20
40
60
80
100

Sigma 70



Effect of changing Sigma and Epoch:

For Sigma=1:There is not much Impact by changing the Value of epochs from 20 to 1000. But the colors that are close to each other are grouped together

For Sigma=10: So As the Value of sigma increased we can see the graph getting distorted. As sigma increases the distance from neighboorhood increases. But Increasing epoch here is making difference as we can see colours more seperable.

For Sigma 30 : we can see the seperability of colours is becoming unclear and also the boundaries to seperate 2 colours is not visible. For Sigma= 50: The graphs are getting more unclear with increase in Sigma. For Sigma = 100 : The graph is very unclear. The boundaries are not at all visible.

here Our neighbourhood function is decaying, so the distance with neighbour will increase with time, increasing sigma values here will make graph unclear but will also group similar colours. And increasing Epoch overall will help us to see colours more clearly.