SHANTANU MANE
03 / 03 / 2023
E - 63
AIML (E)

DEEP LEARNING LAB

Lab Assignment - 2
Deep Learning Lab [CAP - 301]

SHRI RAMDEOBABA COLLEGE OF ENGINEERING AND MANAGEMENT, NAGPUR, 440013

CODE :

```
import matplotlib.pyplot as plt
from mpl_toolkits import mplot3d
import numpy as np

fig = plt.figure(figsize=(10, 7))


class GradientDescentFamilyUpdated:
    """
    :arg x: x
    :arg y: y
    :arg lr: learning rate
    :arg beta: beta
    :arg eps: epsilon
    :arg velocity_x: velocity x
    :arg velocity_y: velocity y

    :de
    """
    x = 0.0
    y = 0.0

    lr, eps = 0.0, 0.0

    velocity_x, velocity_y = 0.0, 0.0

    beta = 0.0

    def __init__(self, x, y, lr, beta, eps):
        """

        :param x:
        :param y:
        :param lr:
        :param beta:
        :param eps:
        """
        self.x = x
        self.y = y
        self.lr = lr
        self.beta = beta
        self.eps = eps

    def Get_Gradient(self, x, y):
        """

        Arguments:
        ----------
        :param x:
        :param y:

        Returns:
        --------
        :returns: dldx, dldy
        """
        dldx = 2 * x
        dldy = 2 * y

        return dldx, dldy
```

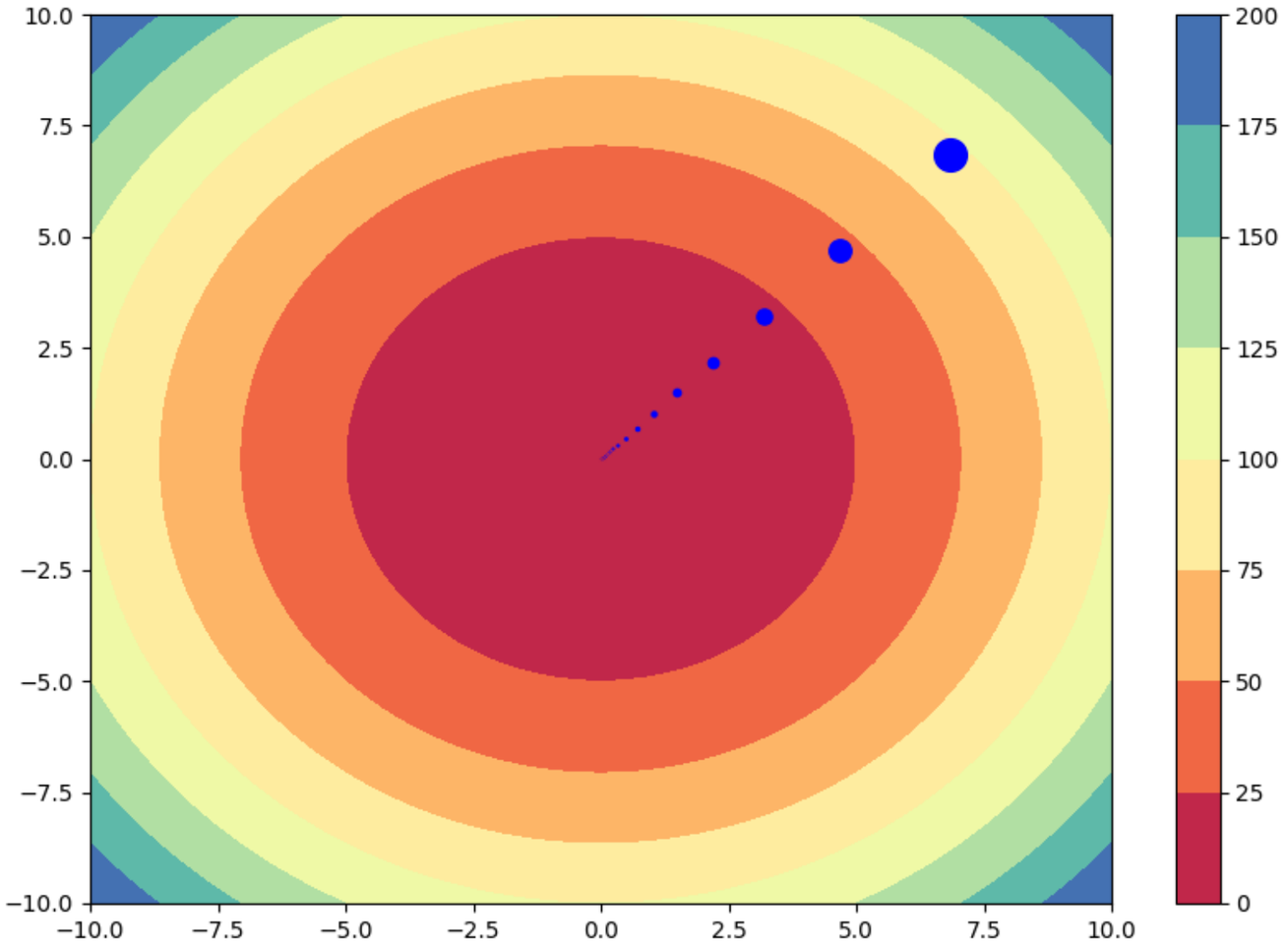
CODE :

```
    def Plot_MeshGrid(self, x, y):
        feature_x = np.linspace(-10.0, 10.0, 101)
        feature_y = np.linspace(-10.0, 10.0, 101)

        # Creating 2-D grid of features
        [X, Y] = np.meshgrid(feature_x, feature_y)

        # fig, ax = plt.subplots(1, 1)

        Z = X ** 2 + Y ** 2

        # plots filled contour plot
        plt.contourf(X, Y, Z, cmap='Spectral')
        plt.colorbar()

    def AdaGrad(self, x: float, y: float) -> None:
        """

        :param x:
        :param y:
        :param lr:
        :return: None
        """
        self.velocity_x, self.velocity_y = 0.0, 0.0
        for epoch in range(30):
            z = x ** 2 + y ** 2

            del_x, del_y = self.Get_Gradient(x, y)

            self.velocity_x += del_x ** 2
            x -= (self.lr / (self.lr * self.eps) ** 0.5) * del_x

            self.velocity_y += del_y ** 2
            y -= (self.lr / (self.lr * self.eps) ** 0.5) * del_y

            plt.scatter(x, y, z, color='b')

        print(x, y)

    def RMSProp(self, x: float, y: float) -> None:
        """

        :param x:
        :param y:
        :return: None
        """
        self.velocity_x, self.velocity_y = 0.0, 0.0
        for epoch in range(35):
            z = x ** 2 + y ** 2

            del_x, del_y = self.Get_Gradient(x, y)

            self.velocity_x = (self.beta * self.velocity_x) + ((1 - self.beta) * del_x **
2)          x -= (self.lr / (self.lr * self.eps) ** 0.5) * del_x

            self.velocity_y = (self.beta * self.velocity_y) + ((1 - self.beta) * del_y **
2)          y -= (self.lr / (self.lr * self.eps) ** 0.5) * del_y

            plt.scatter(x, y, z, color='y')

        print(x, y)

    def Show_Plot(self):
        plt.show()
```

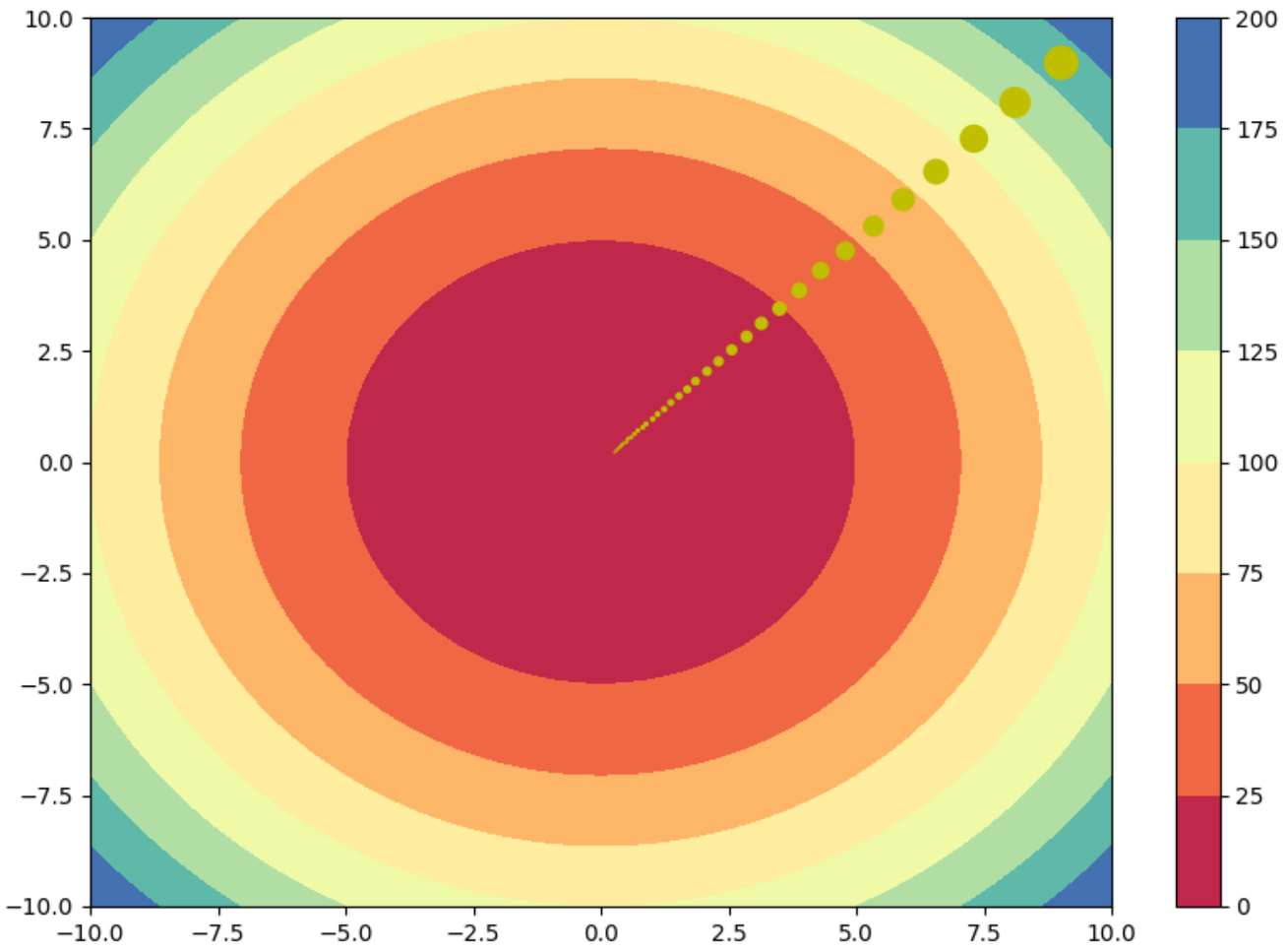
CODE :

```
from Gradient_Descent_AdaGrad_RMSProp import GradientDescentFamilyUpdated

if __name__ == '__main__':
    gd: GradientDescentFamilyUpdated = GradientDescentFamilyUpdated(0.0, 0.0, 0.01, 0.3,
0.4)gd.Plot_MeshGrid(0.0, 0.0)
    gd.AdaGrad(10, 10)
    gd.RMSProp(10, 10)
    gd.Show_Plot()
```

OUTPUT :



ADAGRAD



RMSPROP