SHANTANU MANE
28 / 02 / 2023
E - 63
AIML (E)

DEEP LEARNING LAB

Lab Assignment - 1
Deep Learning Lab *[CAP - 301]*

SHRI RAMDEOBABA COLLEGE OF ENGINEERING AND MANAGEMENT, NAGPUR, 440013

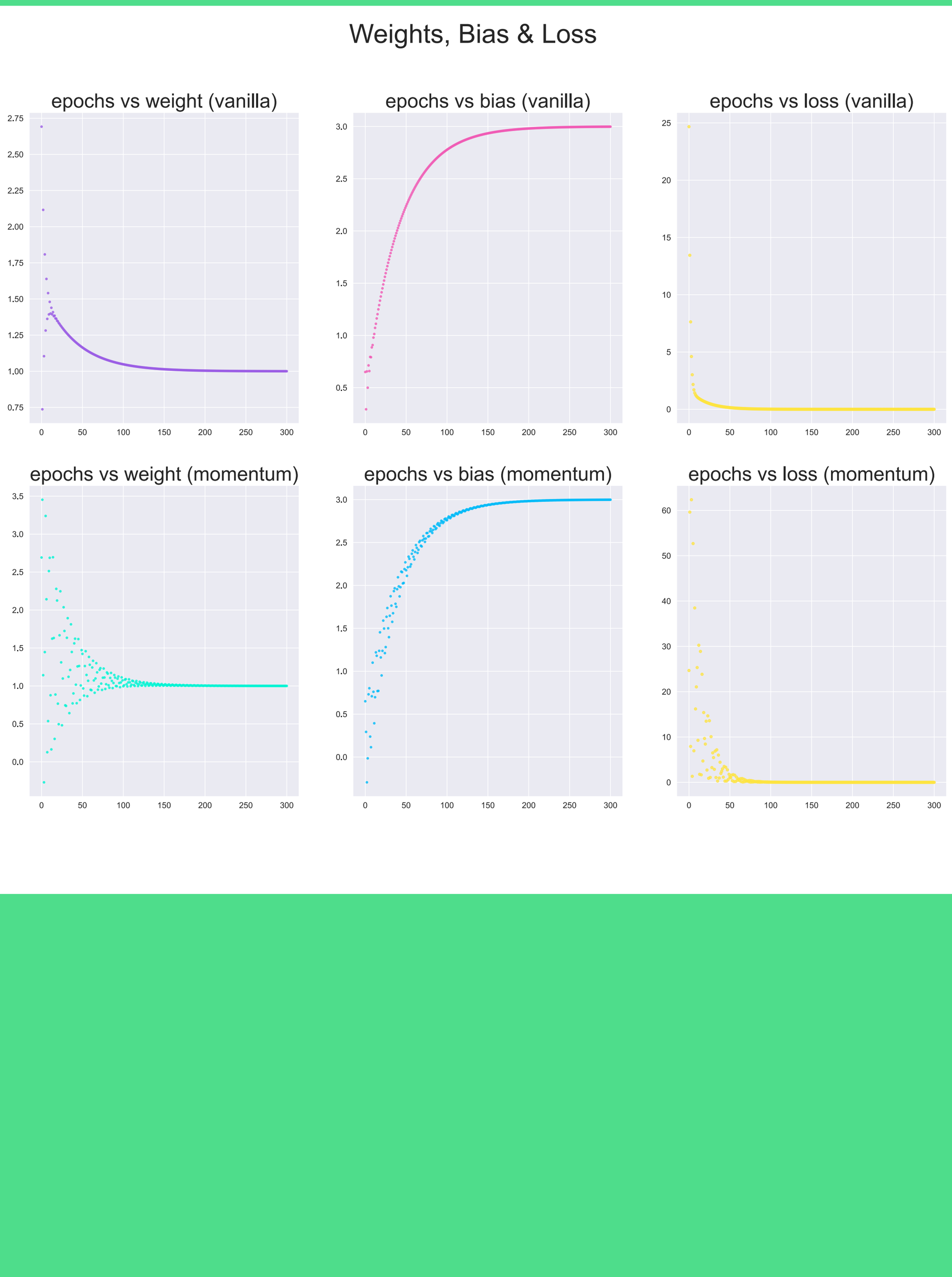
CODE :

```
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns

sns.set_style('darkgrid')
fig, axs = plt.subplots(2, 3, figsize=(20, 15))
fig.suptitle("Weights, Bias & Loss", fontsize=32)


class GradientDescentFamily:
    w = 0.0
    b = 0.0
    velocity = 0.0
    velocity_b = 0.0

    X = np.array([])
    Y = np.array([])

    lr = 0.0
    gamma = 0.0

    N = 0

    colors = ["#9b5de5", "#f15bb5", "#fee440", "#00bbf9", "#00f5d4"]

    def __init__(self, w, b, lr, gamma):
        self.w = w
        self.b = b

        self.lr = lr
        self.gamma = gamma

        self.X = np.array([1, 3.5, 6])
        self.Y = np.array([4, 6.5, 9])

        self.N = self.X.shape[0]

    def Get_Gradient(self):
        dldw = 0.0
        dldb = 0.0

        # N = self.X.shape[0]

        dldw += -2 * self.X * (self.Y - (self.w * self.X + self.b))
        dldb += -2 * (self.Y - (self.w * self.X + self.b))

        return dldw, dldb

    def Vanilla_Gradient_Descent(self):
        loss = 0.0
        self.w = 0.0
        self.b = 0.0
        for epoch in range(301):
            dldw, dldb = self.Get_Gradient()

            self.w -= self.lr * np.sum(dldw) / self.N
            self.b -= self.lr * np.sum(dldb) / self.N

            y_pred = self.w * self.X + self.b
            loss = np.sum((self.Y - y_pred) ** 2) / self.N
```

CODE :

```
            axs[0, 0].scatter(epoch, self.w, color=self.colors[0], s=5, alpha=0.7)
            axs[0, 0].set_title("epochs vs weight (vanilla)", fontsize=24)
            axs[0, 1].scatter(epoch, self.b, color=self.colors[1], s=5, alpha=0.7)
            axs[0, 1].set_title("epochs vs bias (vanilla)", fontsize=24)
            axs[0, 2].plot(epoch, loss, color=self.colors[2], marker='.', alpha=0.7)
            axs[0, 2].set_title("epochs vs loss (vanilla)", fontsize=24)

        print(f'w: {self.w}, b : {self.b}, final loss : {loss}')

    def Momentum_Gradient_Descent(self):
        self.w = 0.0
        self.b = 0.0
        loss = 0.0

        for epoch in range(301):
            dldw, dldb = self.Get_Gradient()

            self.velocity = self.gamma * self.velocity + self.lr * np.sum(dldw) / self.N
            self.w -= self.velocity

            self.b -= self.lr * np.sum(dldb) / self.N

            y_pred = self.w * self.X + self.b
            loss = np.sum((self.Y - y_pred) ** 2) / self.N

            axs[1, 0].scatter(epoch, self.w, color=self.colors[4], s=5, alpha=0.7)
            axs[1, 0].set_title("epochs vs weight (momentum)", fontsize=24)
            axs[1, 1].scatter(epoch, self.b, color=self.colors[3], s=5, alpha=0.7)
            axs[1, 1].set_title("epochs vs bias (momentum)", fontsize=24)
            axs[1, 2].plot(epoch, loss, color=self.colors[2], marker='.', alpha=0.7)
            axs[1, 2].set_title("epochs vs loss (momentum)", fontsize=24)

        print(f'w: {self.w}, b : {self.b}, final loss : {loss}')


if __name__ == '__main__':
    gd: GradientDescentFamily = GradientDescentFamily(0.0, 0.0, 0.05, 1.009)

    gd.Vanilla_Gradient_Descent()
    gd.Momentum_Gradient_Descent()

    plt.show()
```

OUTPUT :


Weights, Bias & Loss
epochs vs weight (vanilla)
epochs vs bias (vanilla)
epochs vs loss (vanilla)
epochs vs weight (momentum)
epochs vs bias (momentum)
epochs vs loss (momentum)