September 16, 2022

Report
CSC 737: Assignment 1

Contents

1 Implementation

The implementation of linear regression with Gradient Descent in Python . I made the separate code for each Mini Batch Gradient Descent ,Stochastic Gradient Descent , Batch Gradient Descent. Each function take the four

parameters for the further working,They are dataFilePath ,**m** which is slope of the line **b** which is the intercept of the linear regression prediction line and number of epochs and Batch size for the mini gradient Descent.

```
def mini_batch_gradient_descent(self, data_path, learning_rate, epochs, batch_size):
    data = pd.read_csv(data_path)

    if batch_size == data.shape[0]:
        # case when batch size equal to the total number of example
        self.batch_gradient_descent(data_path, learning_rate, epochs)
    elif batch_size == 1:
        # cost When batch size equal to the 1
        self.stochastic_gradient_descent(data_path, learning_rate, epochs)
    else:

        Y = np.array(data.y)
        X = np.array(data.drop('y', axis=1))

        no_of_features = X.shape[1]
        no_of_sample = data.shape[0]

        m = np.ones(shape=(no_of_features))
        b = 0
        epochs_list, cost_list = [], []

        for epoch in range(0, epochs):
            # sample_index for mini batch
            sample_index = np.random.randint(0, no_of_sample - 1, batch_size)
            # predict line of linear regression
            batch_X = X[sample_index]
            batch_Y = Y[sample_index]

            guess = np.dot(m, batch_X.T) + b

            m_grad = -(2 / no_of_sample) * (batch_X.T.dot(batch_Y - guess))
            b_grad = -(2 / no_of_sample) * np.sum(((batch_Y - guess)))

            # update the slope and baise after taking derivatives
            m = m - learning_rate * m_grad
            b = b - learning_rate * b_grad

            # computing mean square error as cost function
            cost = np.mean(np.square(batch_Y - guess))

            if epoch % 10 == 0:
                cost_list.append(cost)
                epochs_list.append(epoch)
            # print(f'M : {m} ,B : {b} , Cost : {cost}' )

        # visualization of cost against each epocsh
        self.visualization_dataset(epochs_list, cost_list, 'Mini Batch Gradient Descent')

        return m ,b
```

Figure 1: Mini Batch Gradient Descent

```python
def stochastic_gradient_descent(self, data_path, learning_rate, epochs):

    data = pd.read_csv(data_path)

    Y = np.array(data.y)
    X = np.array(data.drop('y', axis=1))

    no_of_features = X.shape[1]
    no_of_sample = data.shape[0]

    m = np.ones(shape=(no_of_features))
    b = 0
    epochs_list, cost_list = [], []

    for epoch in range(0, epochs):
        random_sample = np.random.randint(0, no_of_sample - 1)
        X_saample = X[random_sample]
        y_sample = Y[random_sample]

        # predict line of linear regression
        guess = np.dot(m, X_saample.T) + b

        m_grad = -(2 / no_of_sample) * (X_saample.T.dot(y_sample - guess))
        b_grad = -(2 / no_of_sample) * (y_sample - guess)

        # update the slope and baise after taking derivatives
        m = m - learning_rate * m_grad
        b = b - learning_rate * b_grad

        # computing mean square error
        cost = np.mean(np.square(y_sample - guess))

        if epoch % 100 == 0:
            cost_list.append(cost)
            epochs_list.append(epoch)
    print(f'M : {m} ,B : {b} , Cost : {cost}')

    self.visualization_dataset(epochs_list, cost_list, 'Stochastic Gradient Descent')
    return m ,b
```

Figure 2: Stochastic Gradient Descent

```python
def batch_gradient_descent(self, data_path, learning_rate, epochs):

    data = pd.read_csv(data_path)

    Y = np.array(data.y)
    X = np.array(data.drop('y', axis=1))

    no_of_features = X.shape[1]
    no_of_sample = data.shape[0]

    m = np.ones(shape=(no_of_features))
    b = 0
    epochs_list, cost_list = [], []

    for epoch in range(0, epochs):
        # predict line of linear regression
        guess = np.dot(m, X.T) + b

        m_grad = -(2 / no_of_sample) * (X.T.dot(Y - guess))
        b_grad = -(2 / no_of_sample) * np.sum(((Y - guess)))

        # update the slope and baise after taking derivatives
        m = m - learning_rate * m_grad
        b = b - learning_rate * b_grad

        # computing mean square error as cost function
        cost = np.mean(np.square(Y - guess))

        if epoch % 10 == 0:
            cost_list.append(cost)
            epochs_list.append(epoch)
        # print(f'M : {m} ,B : {b} , Cost : {cost}' )

    self.visualization_dataset(epochs_list, cost_list, 'Batch Gradient Descent')
    return m ,b
```

Figure 3: Batch Gradient Descent

2 Objective

2.1 Experimental Setup

2.2 How to initialize the weights and bias?

This experiment will evaluate what the initial guess for weight and bias need be. Use a mini-batch stochastic gradient descent algorithm of batch size 50 and learning rate of

0.02. Initialize the weights all zeros, all uniform random and all random from normal distribution. For all the three type of initialization, vary the number of iterations, record the loss and plot the loss against the number of iterations. In Implementation of linear regression using Gradient Descent i forced myself to use Numpy as much as possible.For side the of Dataset i generated the data according to the requirement and save into the csv file.

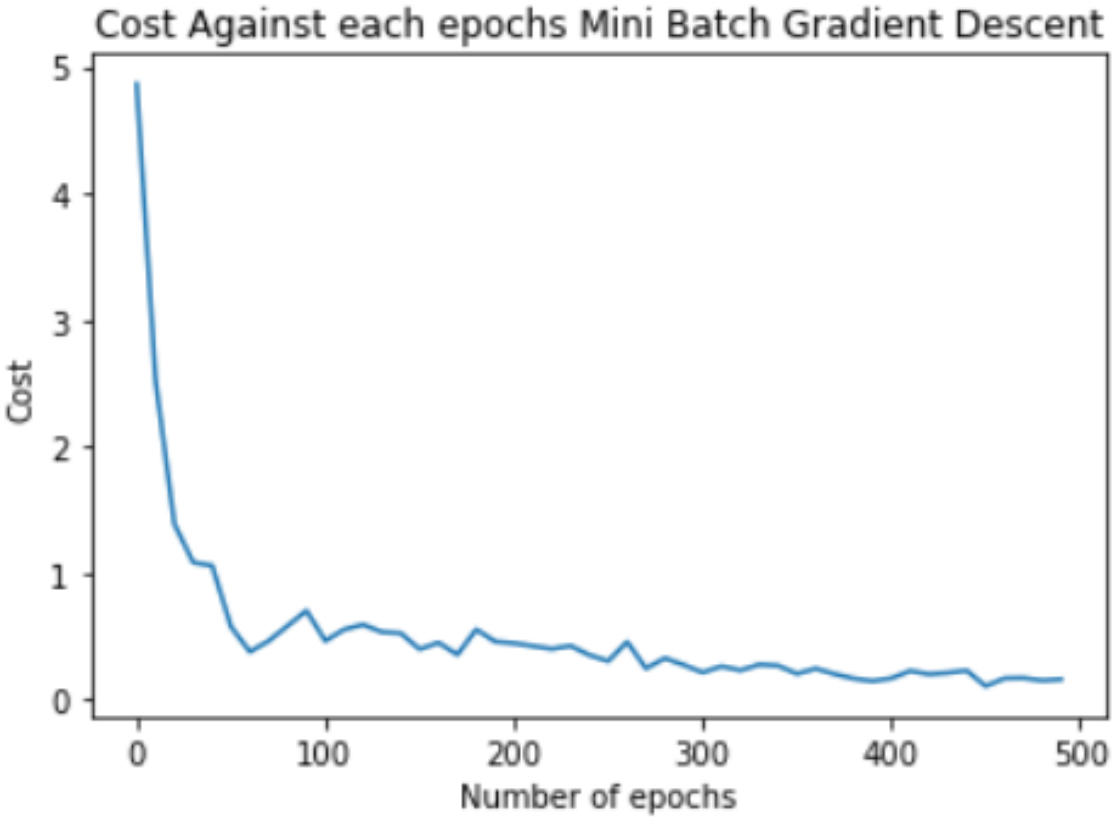

Figure 4: Mini Batch Gradient Descent

2.3 Comparing stochastic, batch and mini-batch gradient descent

This experiment will evaluate how time to converge is affected by the batch size. Vary the batch size to be 1, 50, 200 and 1000, and record the loss with increased number of iterations. Plotted the loss against the number of iterations for the different batch sizes. Use a learning rate of 0.02. Use random initialization of weights and bias.

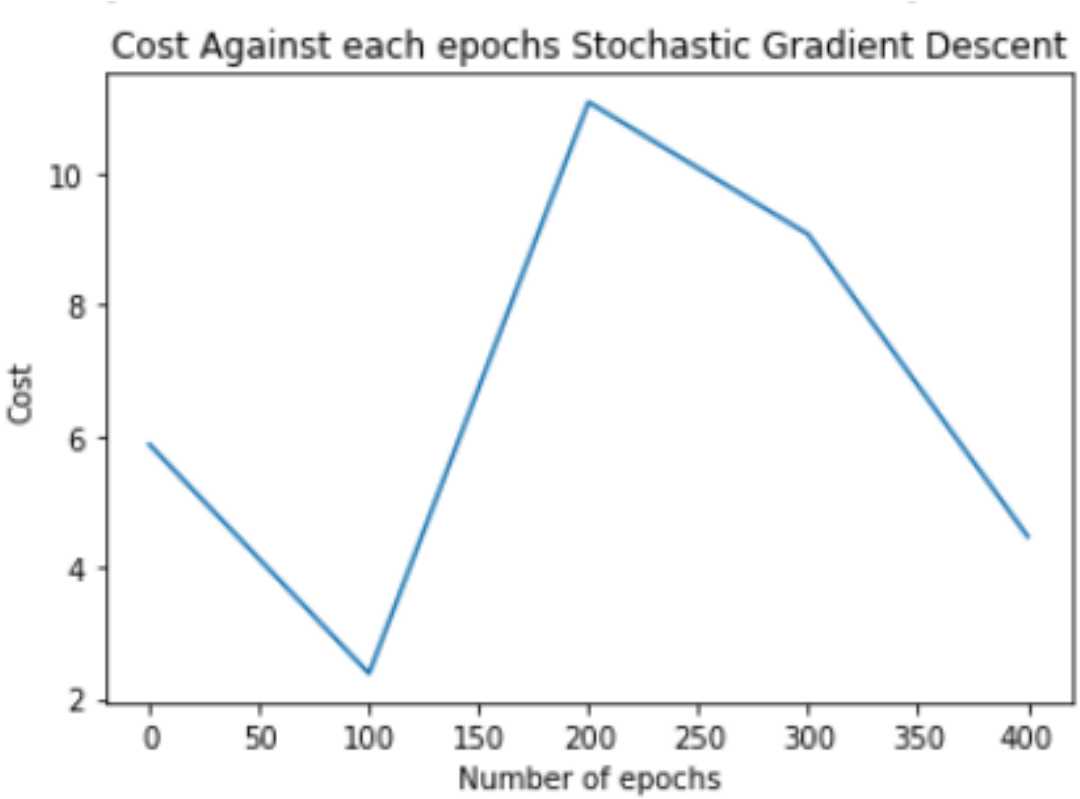

Figure 5: Stochastic Gradient Descent of batch Size 1

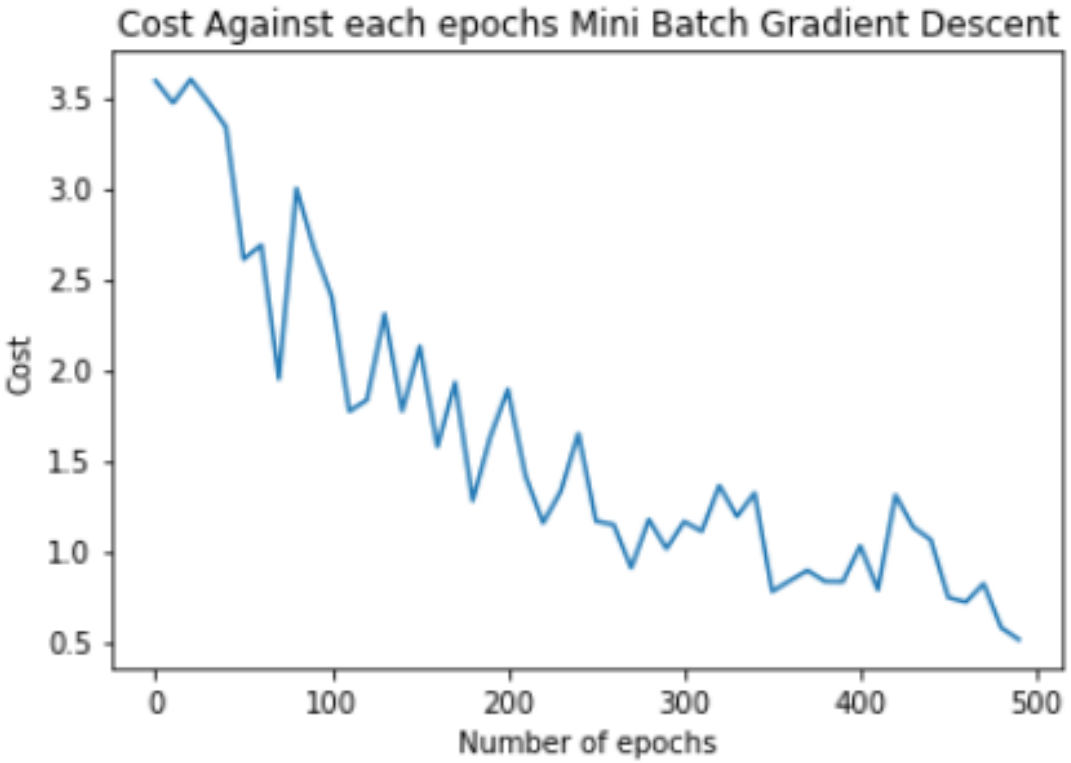

Figure 6: Mini Batch Gradient Descent of batch size 50

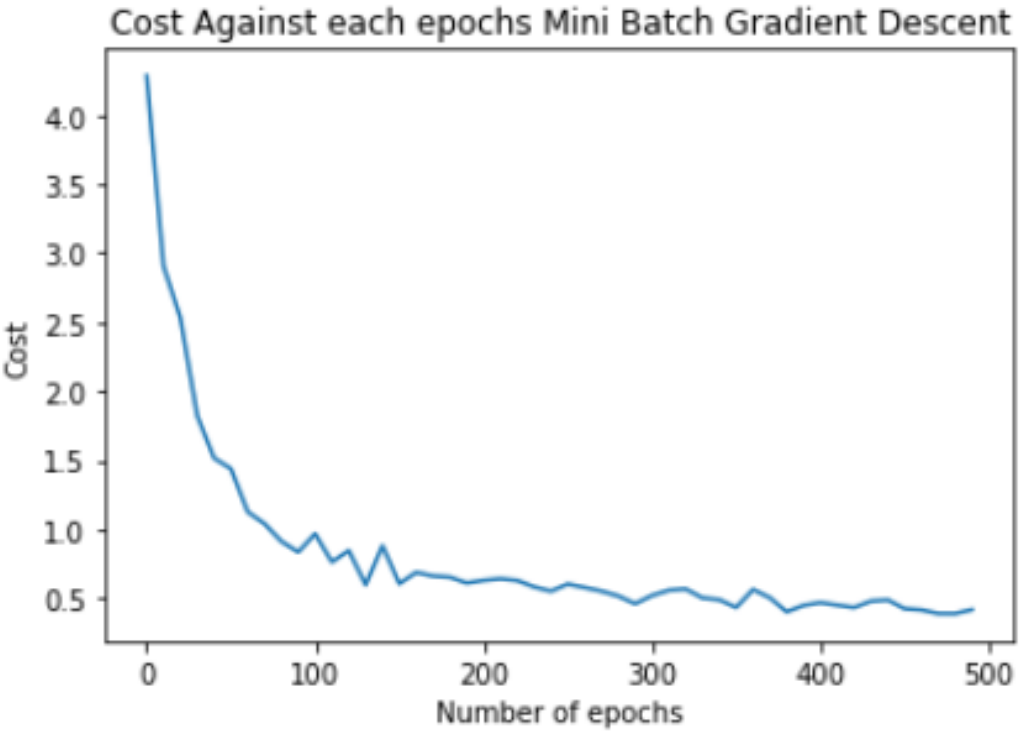

Figure 7: Mini Batch Gradient Descent of batch size 500

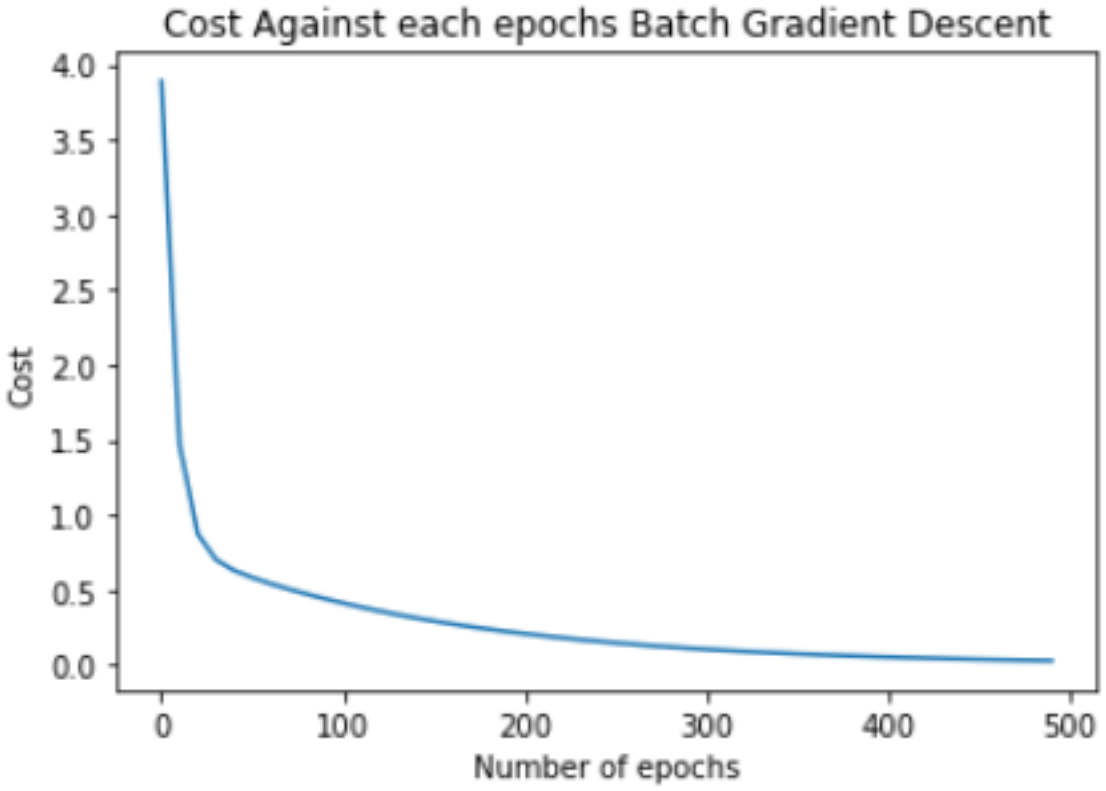

Figure 8: Batch Gradient Descent of batch size 1000

2.4 How learning rate affect the time to converge

In this experiment i am going to check the cost on the different learning rate and same the batch size.We know if set the learning rate is very high we might be lost the global minima and if we set the learning rate is very low then it will take time to get the local minimum.So on this experiment i am going to different result on cost for different learning rate 0.01, 0.05, 0.1, 0.2, 0.3 save after each iteration.

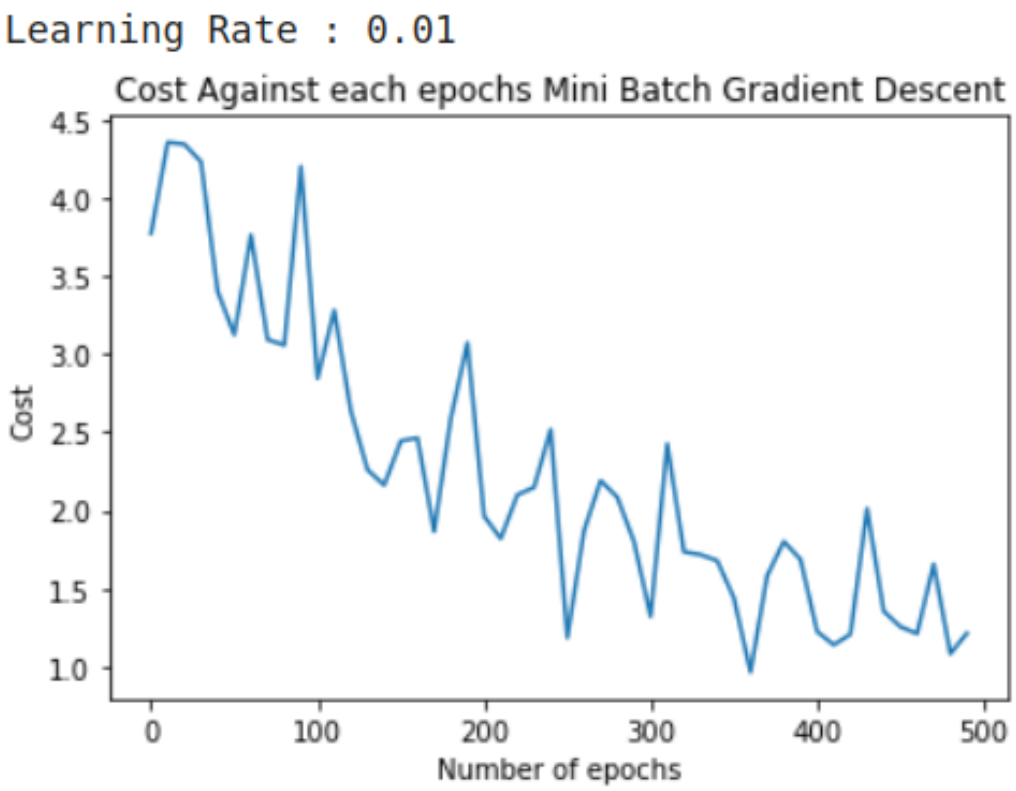

Figure 9: Mini Batch Gradient Descent of batch size 50

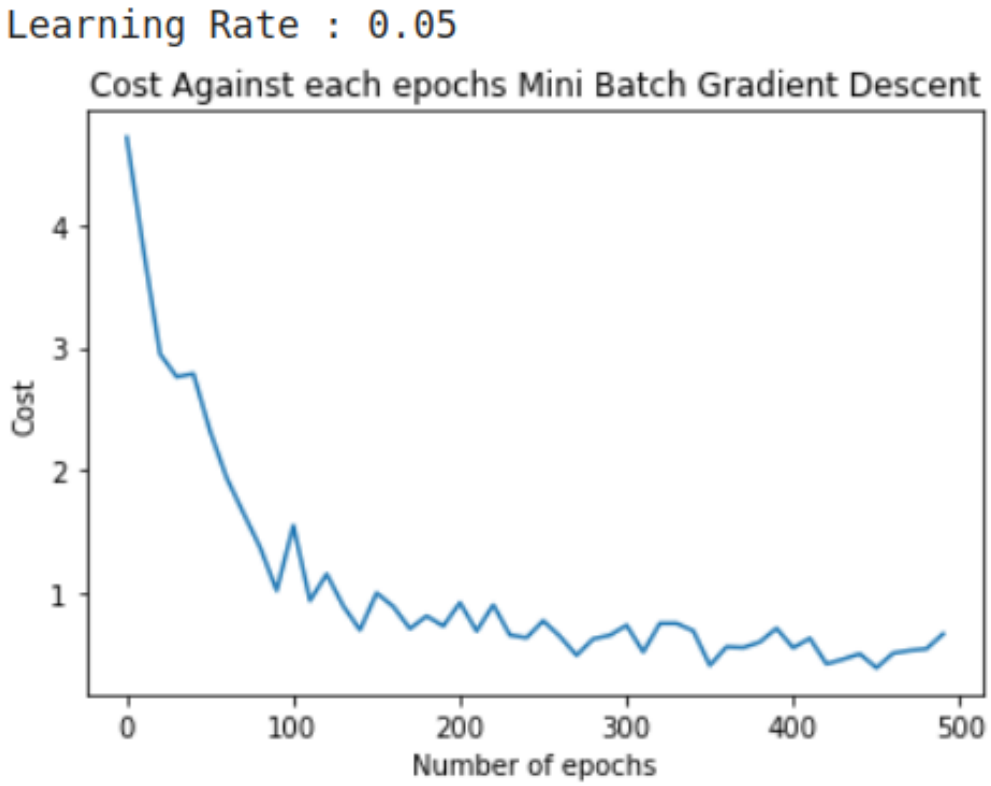

Figure 10: Mini Batch Gradient Descent of batch size 50

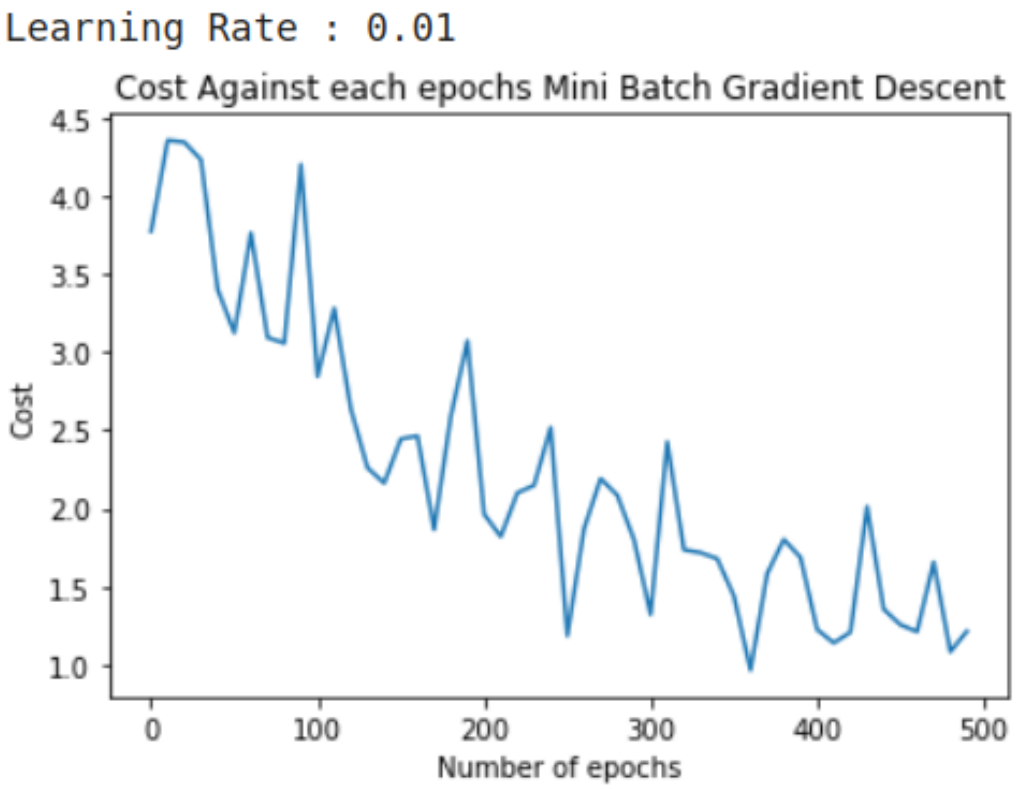

Figure 11: Min Batch Gradient Descent of batch size 50

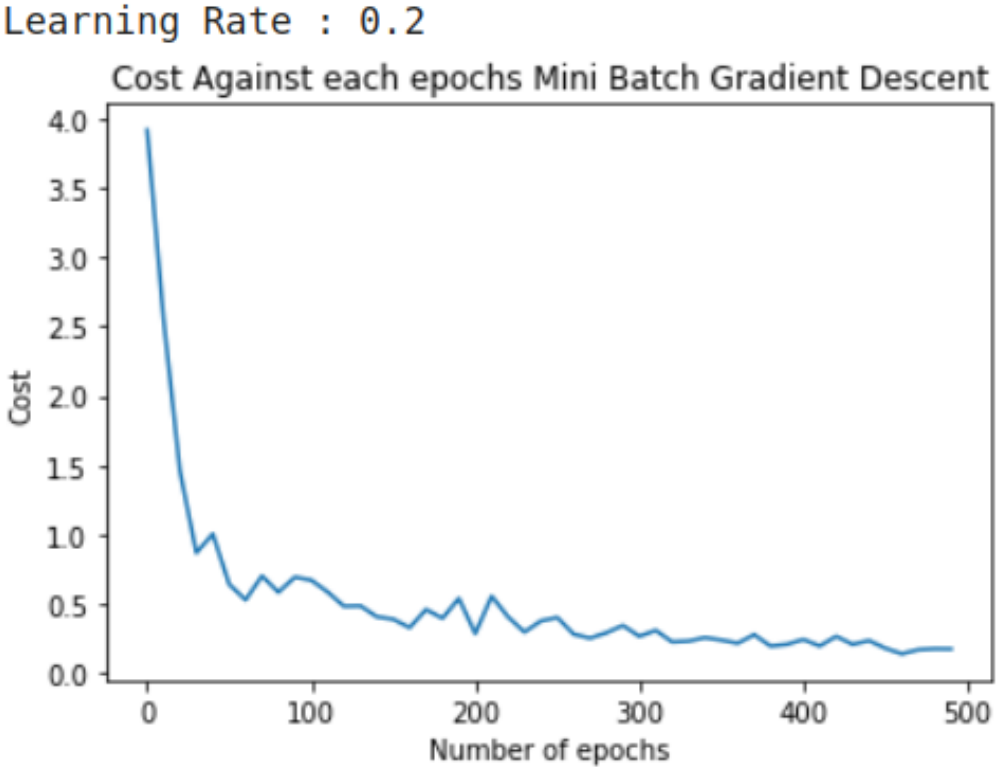

Figure 12: Mini Batch Gradient Descent of batch size 50

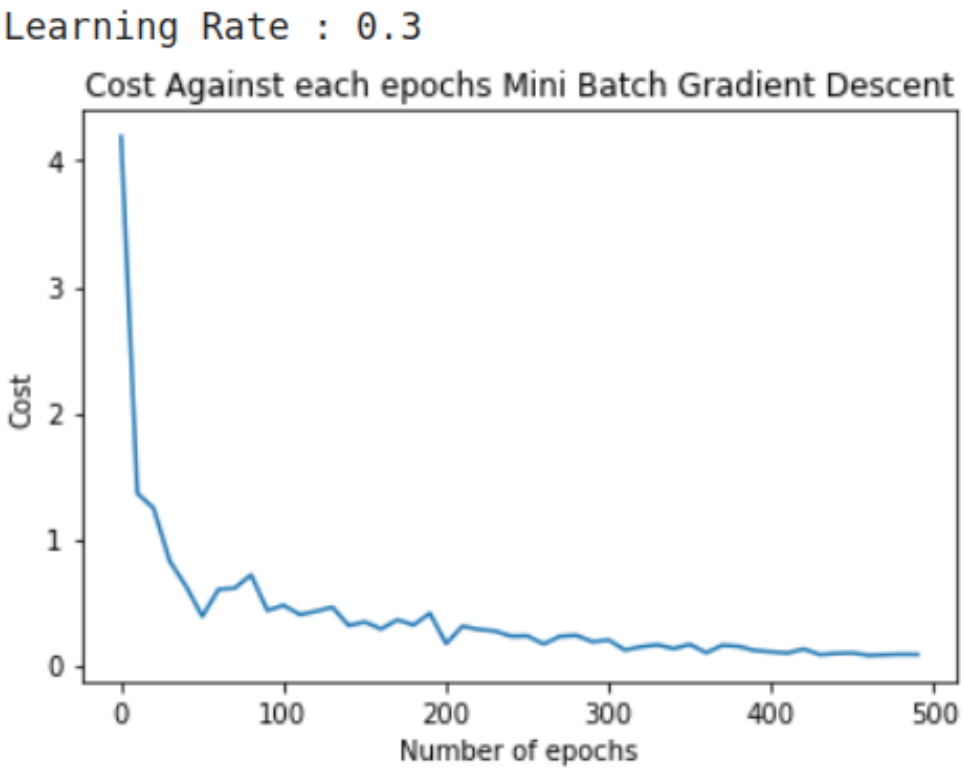

Figure 13: Mini Batch Gradient Descent of batch size 50

3 Conclusions

We have to used the different type Gradient descent in different situation . if we have large Datasets the the we should use the mini batch gradient descent and stochastic gradient descent and if the dataset is small then we should use the Batch gradient descent.We shuld choose the low learning rate 0.01 because we choose the large learning rate the it will take long step toward global minimun and lost the

global minimum.