

# ML HW1

## 1 Linear Algebra Review

Machine Learning HW1

1.

1.1

$$\begin{aligned} 2x_1 + 2x_2 + 3x_3 &= 1 \\ -3x_1 + 6x_2 + 3x_3 &= 6 \\ 5x_1 - 4x_2 &= -5 \\ -5x_1 - 5x_2 &= -5 \\ x_2 &= 0 \end{aligned}$$

$$5x_1 = -5 \Rightarrow x_1 = -1$$

$$1 + x_3 = 2 \Rightarrow x_3 = 1$$

$$x_1 = -1, x_2 = 0, x_3 = 1$$

1.2

$$\begin{bmatrix} 2 & 2 & 3 \\ 1 & -1 & 0 \\ -1 & 2 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 1 \\ -1 \\ 2 \end{bmatrix}$$

1.3. 3

$$1.4. \begin{bmatrix} 2 & 2 & 3 & | & 1 & 0 & 0 \\ 1 & -1 & 0 & | & 0 & 1 & 0 \\ -1 & 2 & 1 & | & 0 & 0 & 1 \end{bmatrix} \quad A^{-1} = \begin{bmatrix} 1 & 4 & -3 \\ 1 & -5 & -3 \\ -1 & 6 & 4 \end{bmatrix}$$

$$\det(A) = -2 + 6 - 3 - 0 - 2$$

$$= \begin{bmatrix} 5 & -4 & 0 & | & 1 & 0 & -3 \\ 1 & -1 & 0 & | & 0 & 1 & 0 \\ -1 & 2 & 1 & | & 0 & 0 & 1 \end{bmatrix} = -1$$

$$\therefore = \begin{bmatrix} 1 & 0 & 0 & | & 1 & -4 & -3 \\ -1 & -1 & 0 & | & 0 & 1 & 0 \\ -1 & 2 & 1 & | & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & | & 1 & -4 & -3 \\ 0 & 1 & 0 & | & 1 & -5 & -3 \\ 0 & 2 & 1 & | & 1 & -4 & -2 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & | & 1 & -4 & -3 \\ 0 & 1 & 0 & | & 1 & -5 & -3 \\ 0 & 0 & 1 & | & -1 & 6 & 4 \end{bmatrix}$$

$$1.5 \quad A \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 1 \\ -1 \\ 2 \end{bmatrix}$$

$$\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = A^{-1} \begin{bmatrix} 1 \\ -1 \\ 2 \end{bmatrix} = \begin{bmatrix} 1 & -4 & -3 \\ 1 & -5 & -3 \\ -1 & 6 & 4 \end{bmatrix} \begin{bmatrix} 1 \\ -1 \\ 2 \end{bmatrix} = \begin{bmatrix} -1 \\ 0 \\ 1 \end{bmatrix}$$

$$1.6 \quad \langle x, b \rangle = [-1, 0, 1] \begin{bmatrix} 1 \\ -1 \\ 2 \end{bmatrix} = 1 \times 0 + 0 \times (-1) + 1 \times 2 = \begin{bmatrix} -1 & 0 & -2 \\ 0 & 0 & 0 \\ 1 & -1 & 2 \end{bmatrix}$$

$$1.7 \quad \|b\|_1 = 1 + 1 + 2 = 4$$

$$\|b\|_2 = \sqrt{1^2 + (-1)^2 + 2^2} = \sqrt{6}$$

$$\|b\|_\infty = 2$$

$$1.8 \quad A = \begin{bmatrix} 2 & 2 & 3 \\ 1 & -1 & 0 \\ -1 & 2 & 1 \\ -1 & 2 & 1 \end{bmatrix} \quad \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 1 \\ -1 \\ 2 \\ 1 \end{bmatrix} \quad A^T = \begin{bmatrix} 1 & -4 & -1.5 & -1.5 \\ 1 & -4 & -1.5 & -1.5 \\ -1 & 6 & 2 & 2 \end{bmatrix}$$

1.12 Yes, because  $B^{-1}B = I$

$$1.9 \quad \text{rank}(A_t) = 3$$

$$1.10 \quad \text{No, because } -x_1 + 2x_2 + x_3 \text{ can't be both 2 and 1 at the same time}$$

$$\frac{\partial (y^T A y)}{\partial y} = y^T (A^T + A)$$

$$= \begin{bmatrix} 1, 2, 3 \\ 3, 2, 2 \\ 2, 2, 2 \end{bmatrix} \begin{bmatrix} 4 & 3 & 2 \\ 3 & 2 & 2 \\ 2 & 2 & 2 \end{bmatrix}$$

## 2 Linear Regression Model Fitting (Programming)

```

1 # Machine Learning HW1
2 import matplotlib.pyplot as plt
3 import numpy as np
4 # more imports
5
6
7 # Parse the file and return 2 numpy arrays
8
9
10 def load_data_set(filename):
11     arr = np.loadtxt(filename)
12     x, y = np.split(arr, [-1], axis=1)
13     # plt.plot(x[:, 1], y, '.')

```

```

14     #     plt.show()
15     return x, y
16
17 # Find theta using the normal equation
18
19
20 def normal_equation(x, y):
21     x_t = np.transpose(x)
22     theta = np.dot(np.dot(np.linalg.inv(np.dot(x_t, x)), x_t), y)
23     return theta
24
25 # Find thetas using stochastic gradient descent
26 # Don't forget to shuffle
27
28
29 def stochastic_gradient_descent(x, y, learning_rate, num_iterations):
30     # your code
31     thetas = []
32     theta = np.array([
33         [1],
34         [1]
35     ])
36     for _ in range(num_iterations):
37         x, y = unison_shuffled_copies(x, y)
38         for row, label in zip(x, y):
39             theta = theta - learning_rate * [
40                 row[:, np.newaxis] * (np.dot(row, theta) - label)
41             ]
42             thetas.append(theta)
43     # print(thetas)
44     return thetas
45
46 # Find thetas using gradient descent
47
48 def gradient_descent(x, y, learning_rate, num_iterations):
49     thetas = []
50     theta = np.array([
51         [1],
52         [1]
53     ])
54     for _ in range(num_iterations):
55         gradient_sum = np.array([
56             [0.0],
57             [0.0]
58         ])
59         for row, label in zip(x, y):
60             gradient_sum = gradient_sum - learning_rate * [
61                 row[:, np.newaxis] * (np.dot(row, theta) - label)
62             ]
63             theta = theta + gradient_sum
64             thetas.append(theta)
65     return thetas
66
67 def unison_shuffled_copies(a, b):
68     assert len(a) == len(b)
69     p = np.random.permutation(len(a))
70     return a[p], b[p]
71
72 # Find thetas using minibatch gradient descent
73 # Don't forget to shuffle
74
75
76 def minibatch_gradient_descent(x, y, learning_rate, num_iterations, batch_size):
77     thetas = []
78     theta = np.array([
79         [1],
80         [1]
81     ])
82     gradient_sum = np.array([

```

```

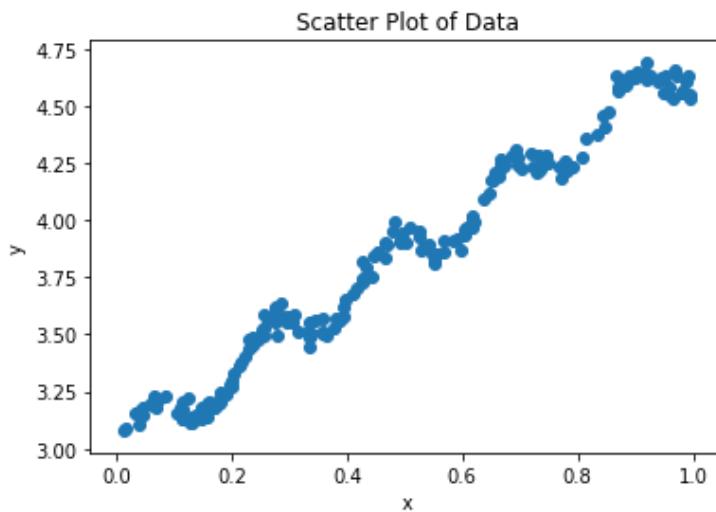
83         [0],
84         [0]
85     ])
86     count = 0
87     for _ in range(num_iterations):
88         x, y = unison_shuffled_copies(x, y)
89         # print(x, y)
90         for row, label in zip(x, y):
91             count += 1
92             gradient_sum = gradient_sum - learning_rate * [
93                 row[:, np.newaxis] * (np.dot(row, theta) - label)
94             ]
95             if count == batch_size:
96                 count = 0
97                 theta = theta + gradient_sum
98                 gradient_sum = np.array([
99                     [0],
100                    [0]
101                ])
102                # print(theta)
103                theta = theta + gradient_sum
104                thetas.append(theta)
105                # print(thetas)
106
107    # Given an array of x and theta predict y
108
109
110    def predict(x, theta):
111        # your code
112        y_predict = np.dot(x, theta)
113        return y_predict
114
115    # Given an array of y and y_predict return loss
116
117
118    def get_loss(y, y_predict):
119        # your code
120        diff = y - y_predict
121        loss = np.dot(diff.T, diff) / len(y)
122        return loss
123
124    # Given a list of thetas one per epoch
125    # this creates a plot of epoch vs training error
126
127
128    def plot_training_errors(x, y, thetas, title):
129        accuracies = []
130        epochs = []
131        losses = []
132        epoch_num = 1
133        for theta in thetas:
134            losses.append(get_loss(y, predict(x, theta)))
135            epochs.append(epoch_num)
136            epoch_num += 1
137        plt.scatter(epochs, losses)
138        plt.xlabel("epoch")
139        plt.ylabel("loss")
140        plt.title(title)
141        plt.show()
142
143    # Given x, y, y_predict and title,
144    # this creates a plot
145
146
147    def plot(x, y, theta, title):
148        # plot
149        y_predict = predict(x, theta)
150        plt.scatter(x[:, 1], y)
151        plt.plot(x[:, 1], y_predict)

```

```
152     plt.xlabel("x")
153     plt.ylabel("y")
154     plt.title(title)
155     plt.show()
```

## Data Set

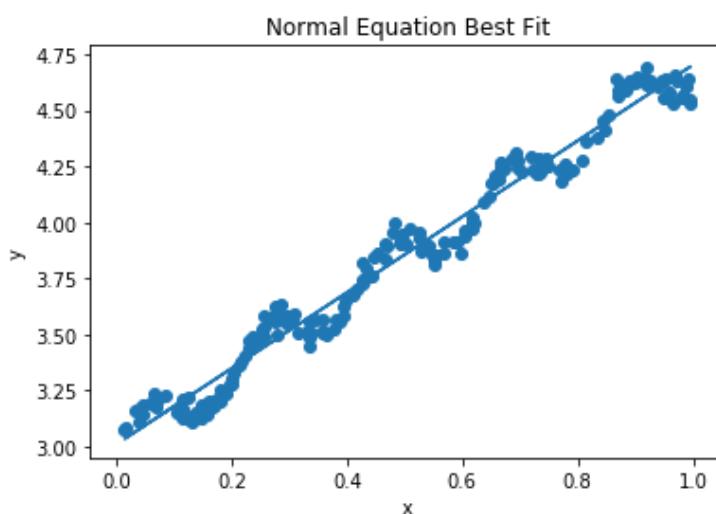
```
1 x, y = load_data_set('regression-data.txt')
2 # plot
3 plt.scatter(x[:, 1], y)
4 plt.xlabel("x")
5 plt.ylabel("y")
6 plt.title("Scatter Plot of Data")
7 plt.show()
```



## Normal Equation

```
1 theta = normal_equation(x, y)
2 print('theta={}'.format(theta))
3 plot(x, y, theta, "Normal Equation Best Fit")
```

```
1 theta=[[3.00774324]
2 [1.69532264]]
```

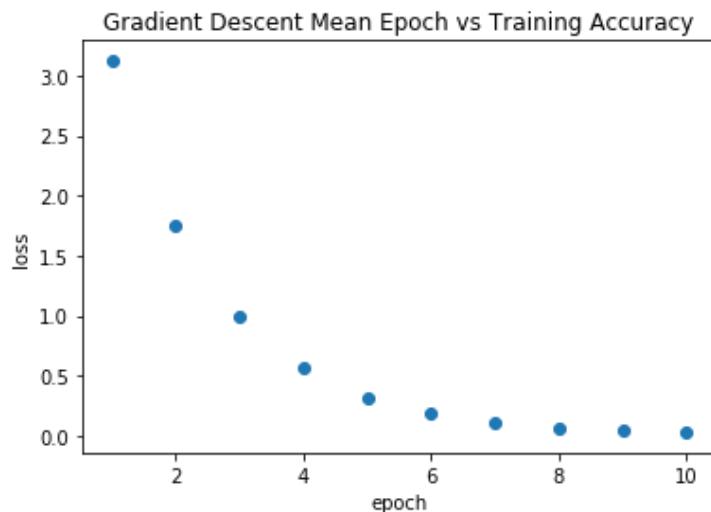
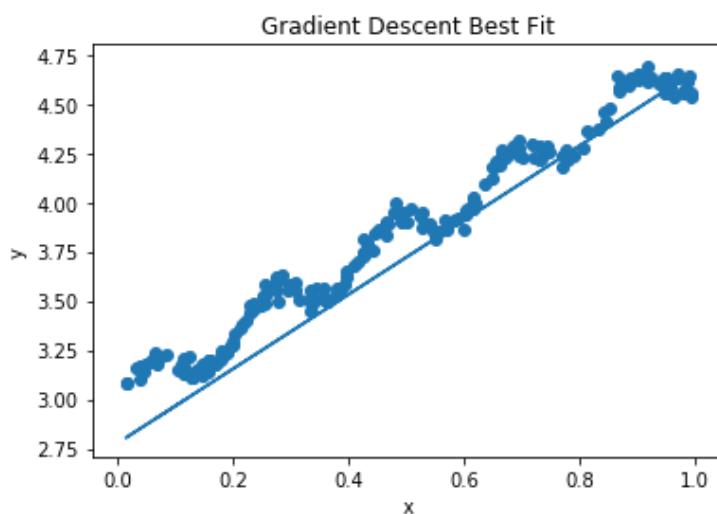


## Gradient Descent

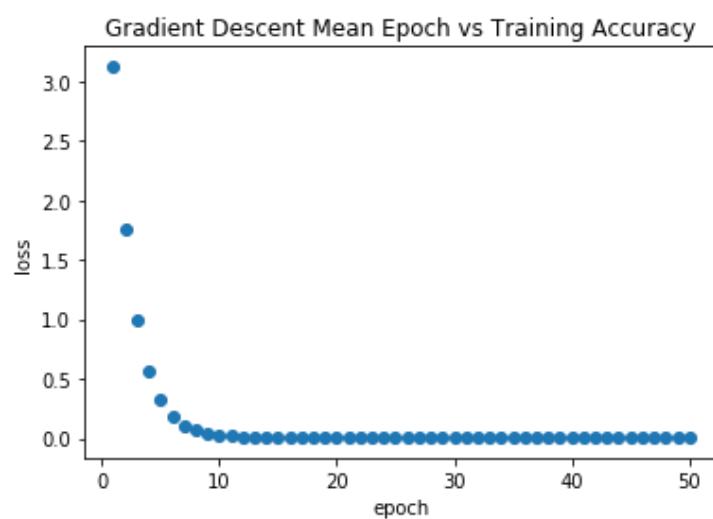
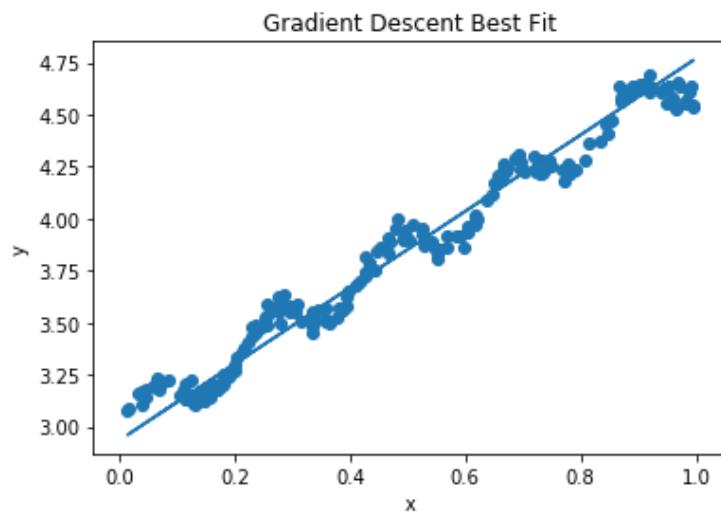
## Results

```
1 n_iteration_list = [10, 50]
2 learning_rate_list = [0.001, 0.005, 0.01, 0.05, 0.1, 0.3]
3 for learning_rate in learning_rate_list:
4     for n_iter in n_iteration_list:
5         print('learning rate: {}, number of iterations: {}'.format(learning_rate,
n_iter))
6         thetas = gradient_descent(x, y, learning_rate, n_iter)
7         plot(x, y, thetas[-1], "Gradient Descent Best Fit")
8         plot_training_errors(x, y, thetas, "Gradient Descent Mean Epoch vs Training
Accuracy")
```

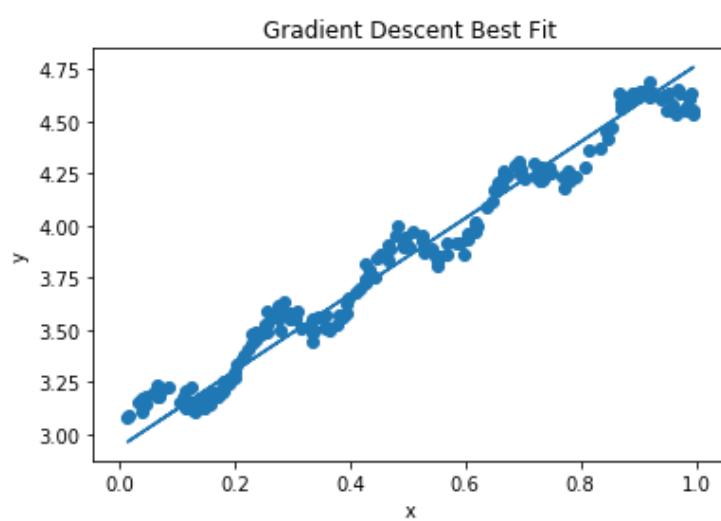
```
1 learning rate: 0.001, number of iterations: 10
```

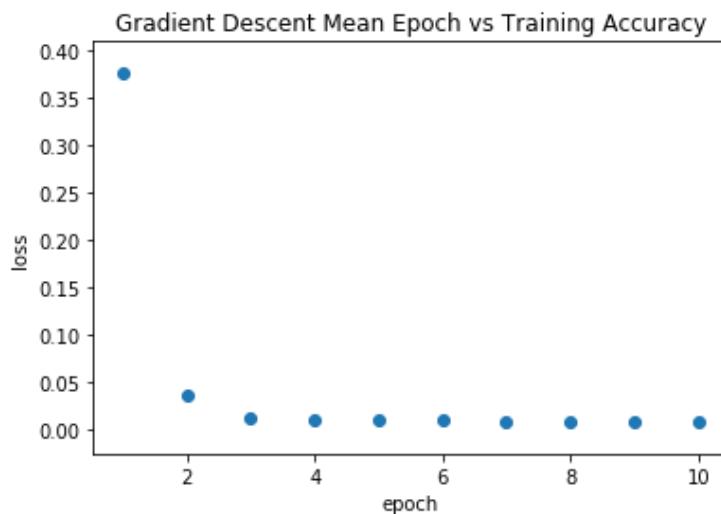


```
1 learning rate: 0.001, number of iterations: 50
```

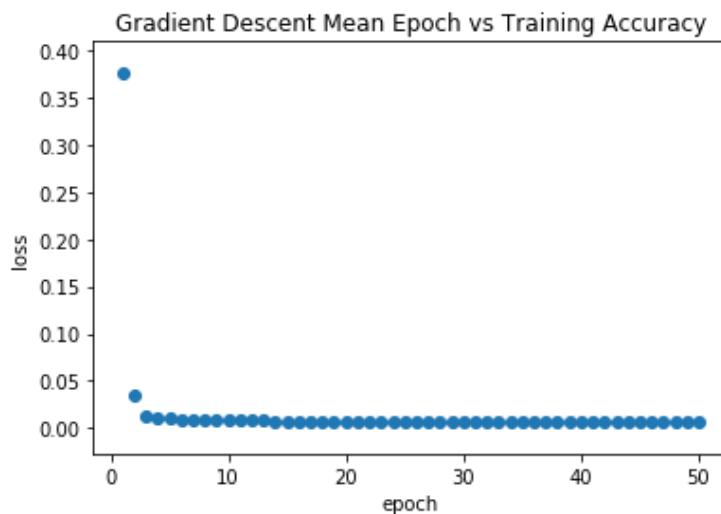
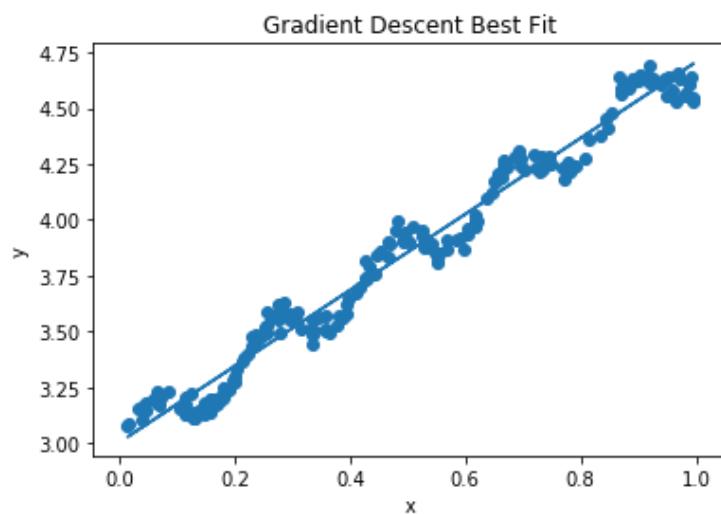


```
1 learning rate: 0.005, number of iterations: 10
```

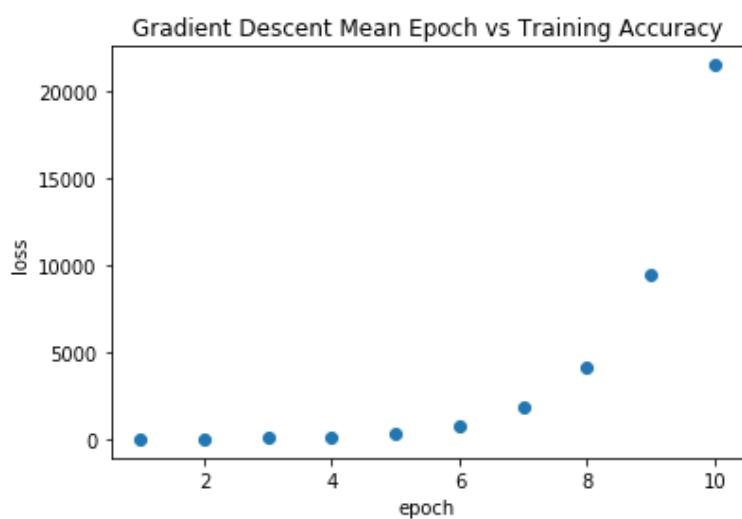
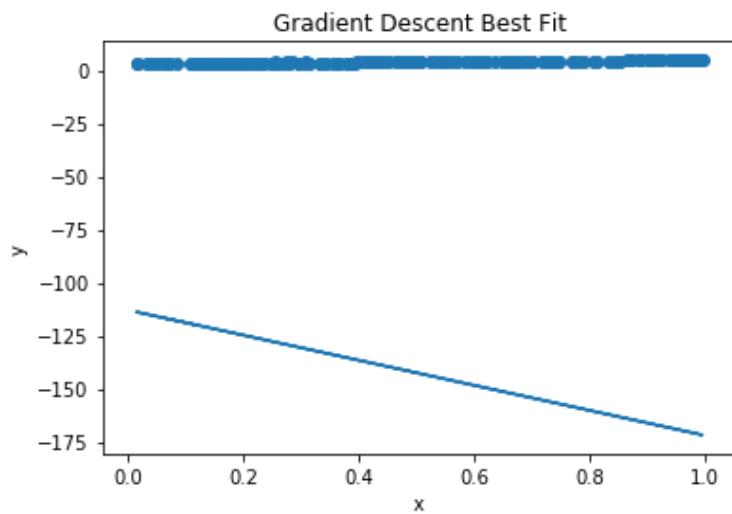




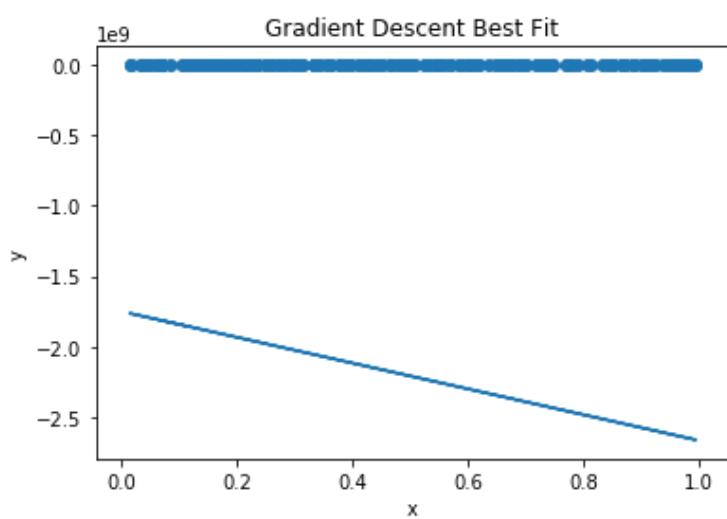
```
1 learning rate: 0.005, number of iterations: 50
```

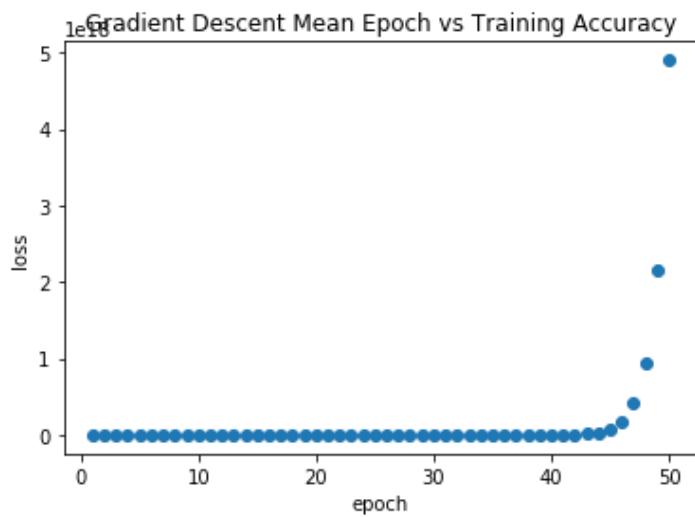


```
1 learning rate: 0.01, number of iterations: 10
```

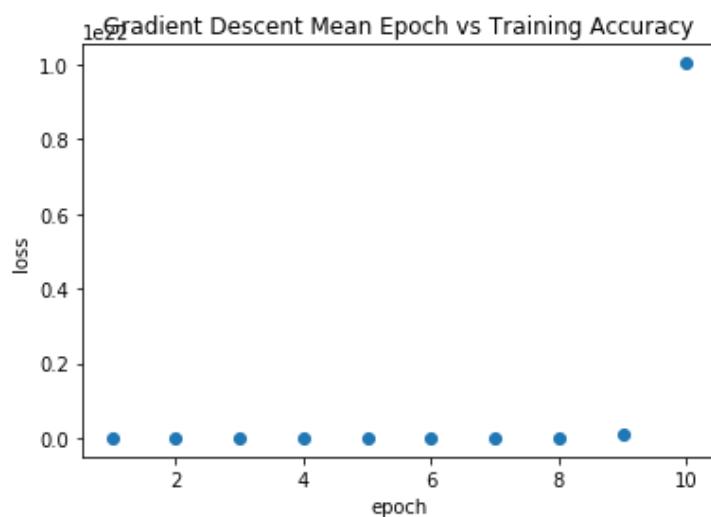
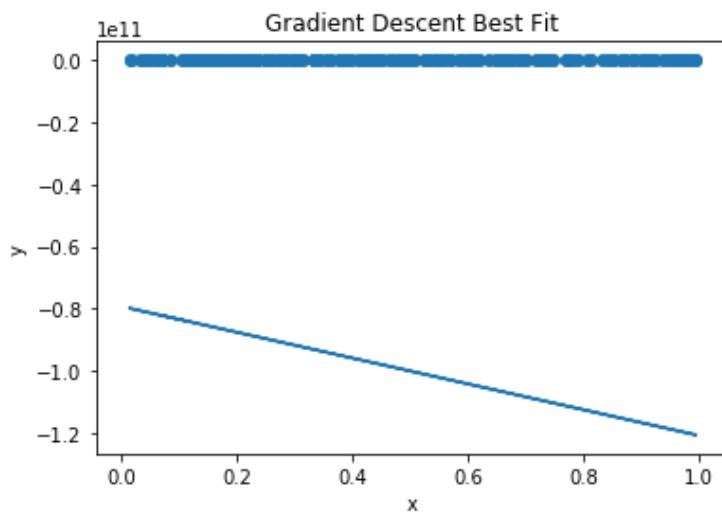


```
1 learning rate: 0.01, number of iterations: 50
```

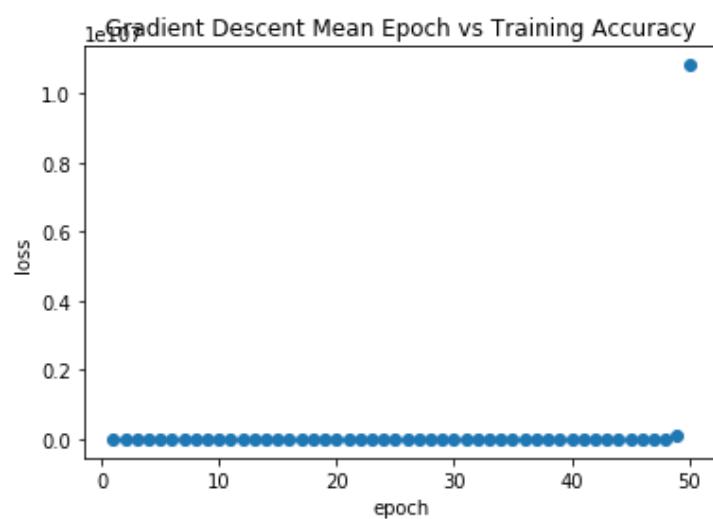
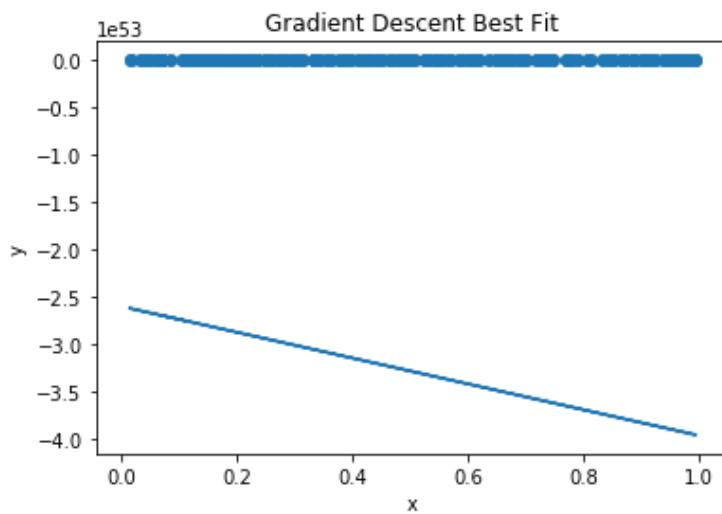




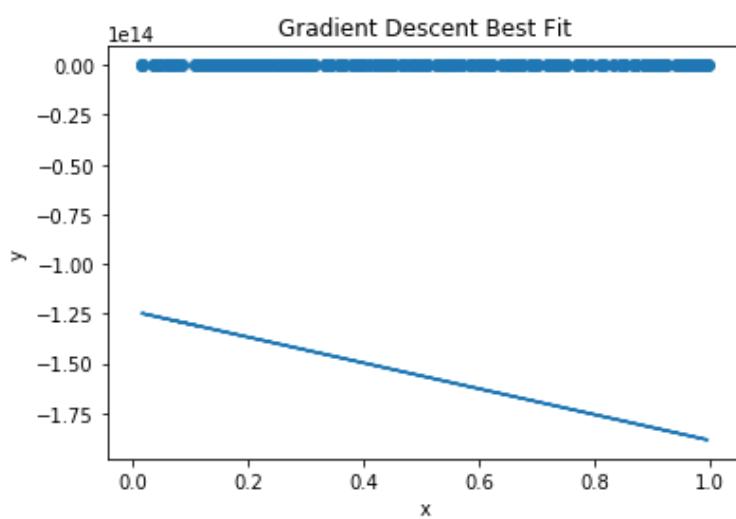
```
1 learning rate: 0.05, number of iterations: 10
```

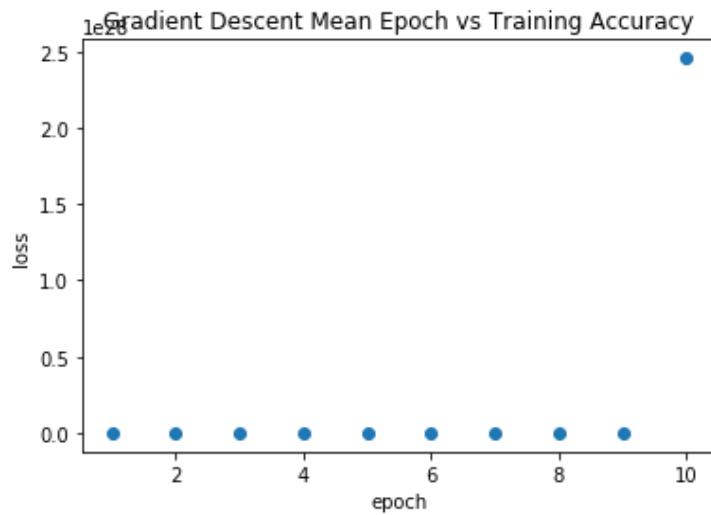


```
1 learning rate: 0.05, number of iterations: 50
```

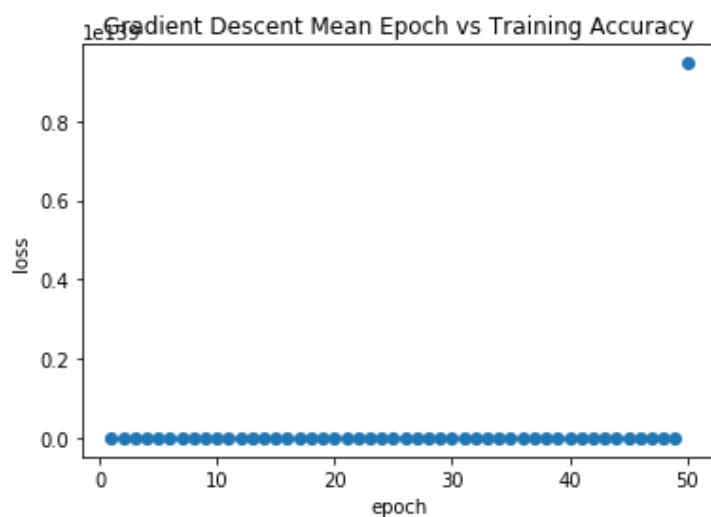
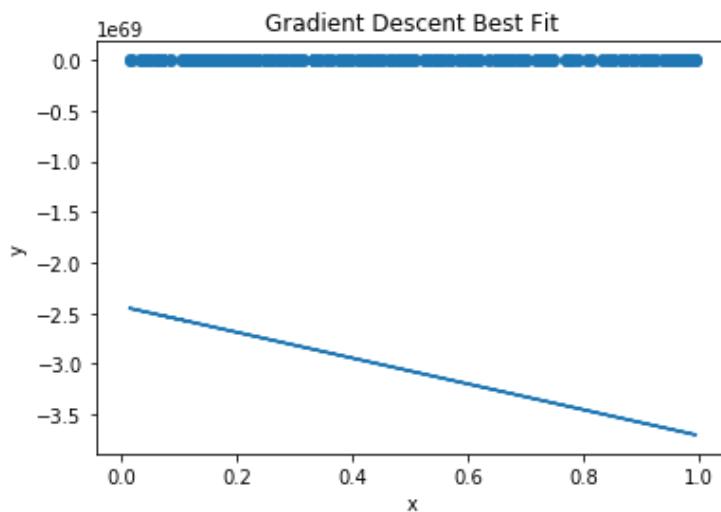


```
1 learning rate: 0.1, number of iterations: 10
```

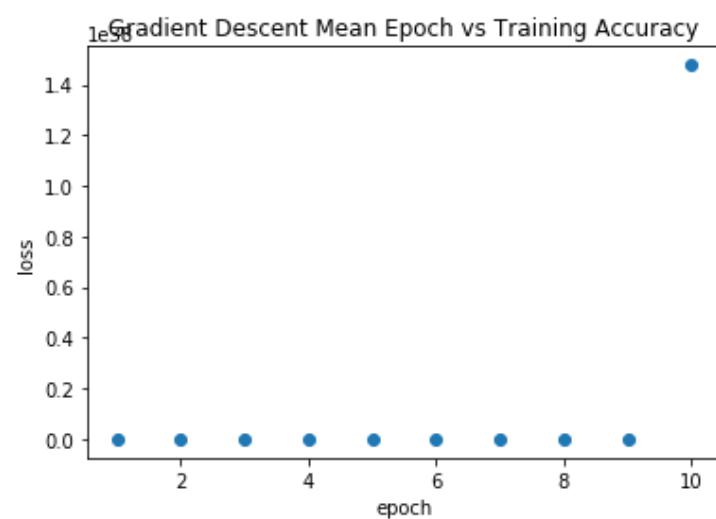
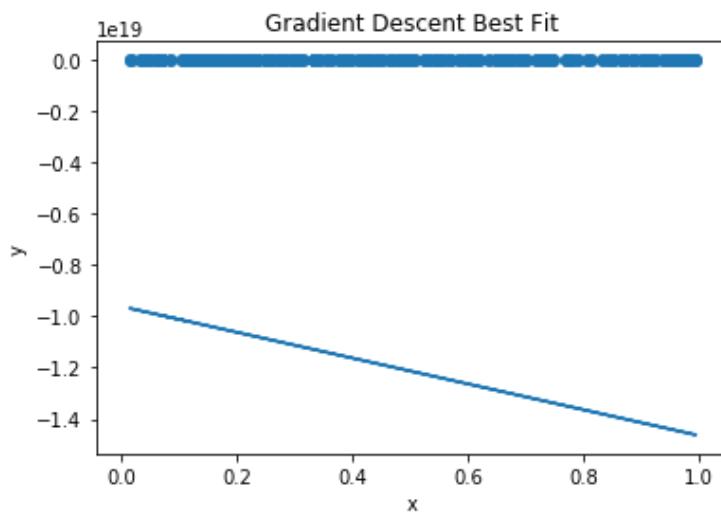




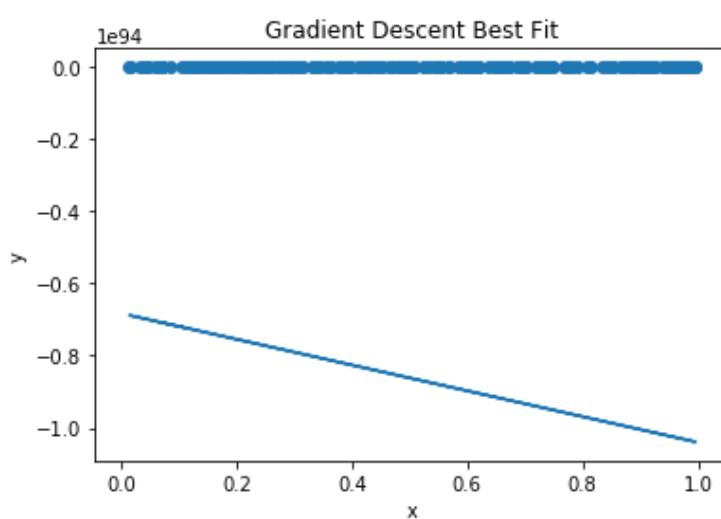
```
1 learning rate: 0.1, number of iterations: 50
```

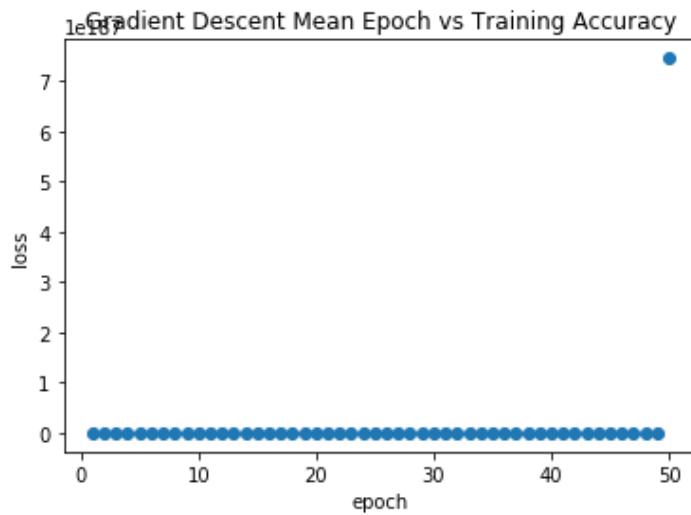


```
1 learning rate: 0.3, number of iterations: 10
```



```
1 learning rate: 0.3, number of iterations: 50
```





## Observation

As shown above, too large a learning rate resulted in the loss function not converging, with 0.001 being a small enough learning rate for the algorithm to get very close to the minimum point.

## Stochastic Gradient Descent

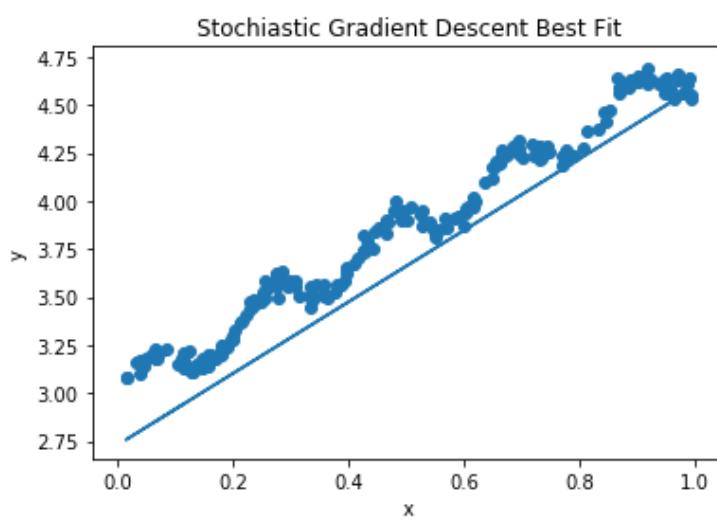
### Results

```

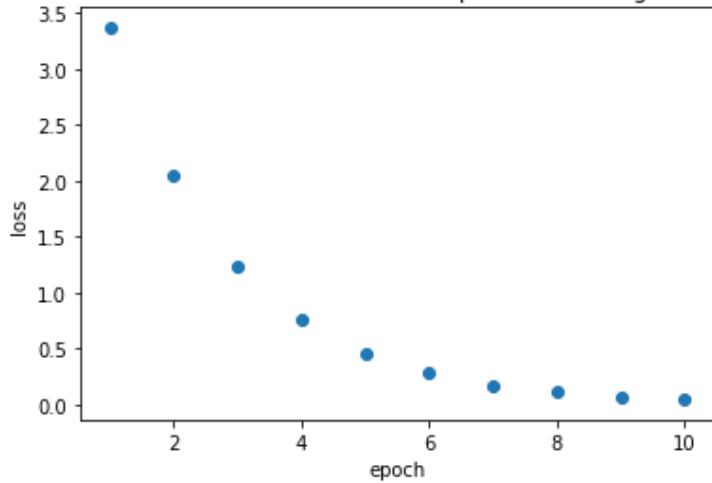
1 n_iteration_list = [10, 50]
2 learning_rate_list = [0.001, 0.005, 0.01, 0.05, 0.1, 0.3]
3 for learning_rate in learning_rate_list:
4     for n_iter in n_iteration_list:
5         print('learning rate: {}, number of iterations: {}'.format(learning_rate,
n_iter))
6         thetas = stochastic_gradient_descent(x, y, learning_rate, n_iter)
7         plot(x, y, thetas[-1], "Stochastic Gradient Descent Best Fit")
8         plot_training_errors(x, y, thetas, "Stochastic Gradient Descent Mean Epoch vs
Training Accuracy")

```

```
1 learning rate: 0.001, number of iterations: 10
```

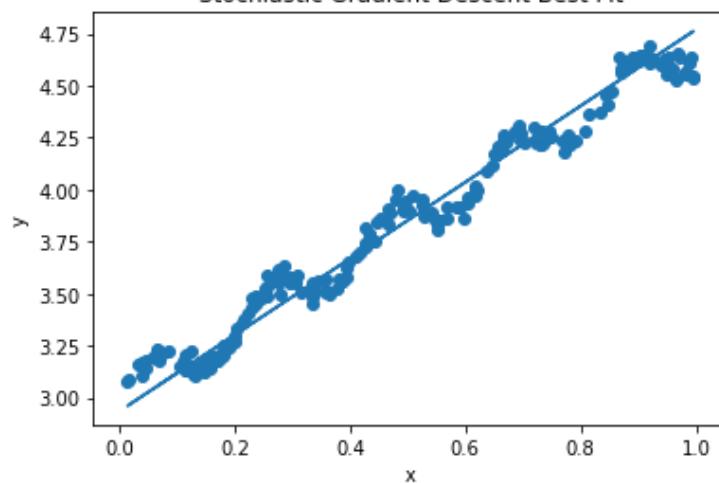


Stochiastic Gradient Descent Mean Epoch vs Training Accuracy

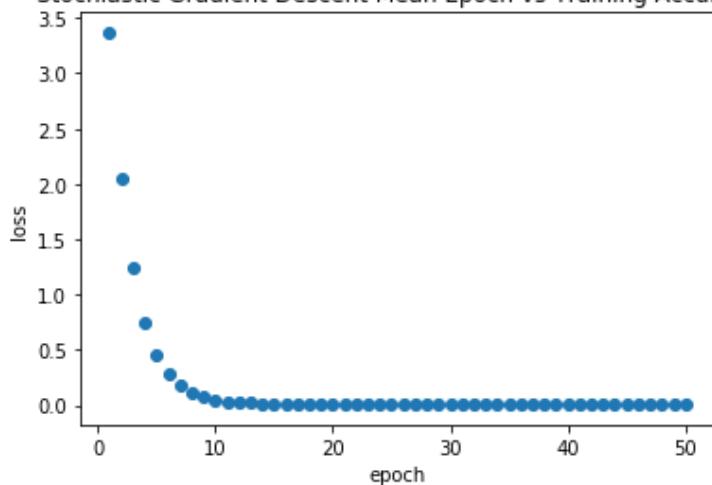


```
1 learning rate: 0.001, number of iterations: 50
```

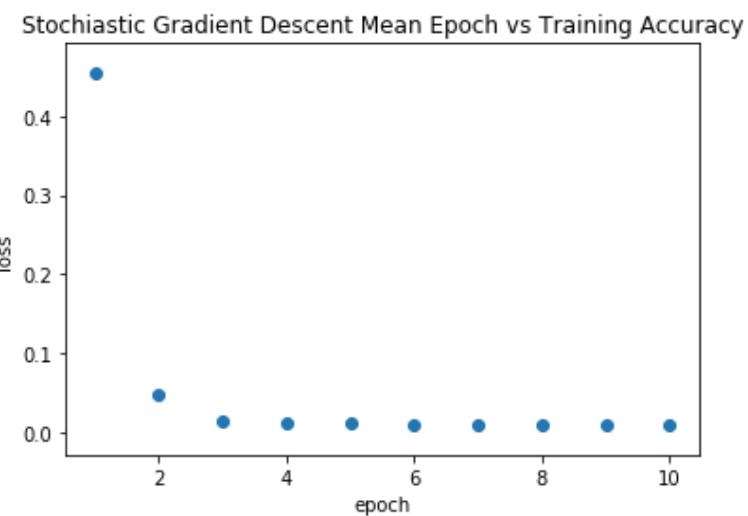
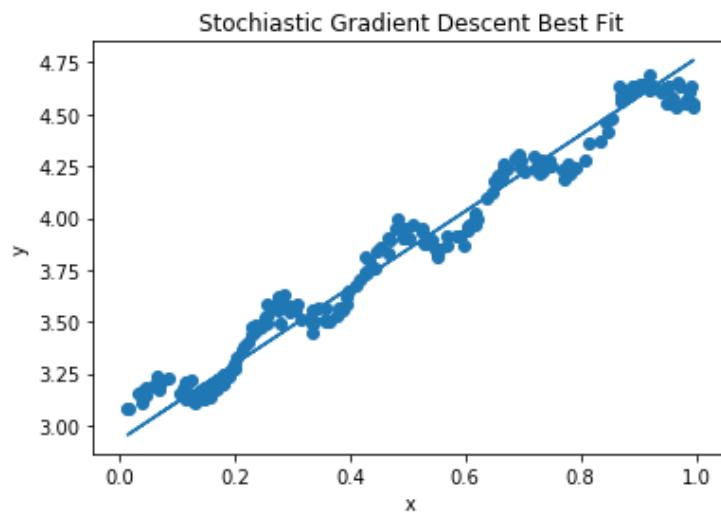
Stochiastic Gradient Descent Best Fit



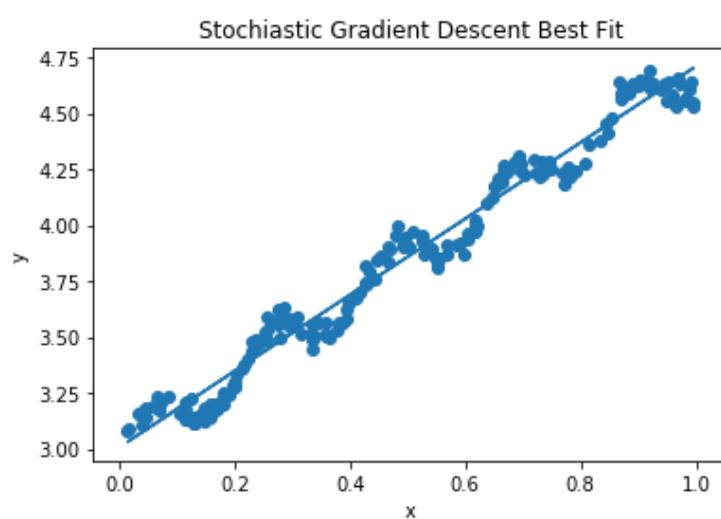
Stochiastic Gradient Descent Mean Epoch vs Training Accuracy



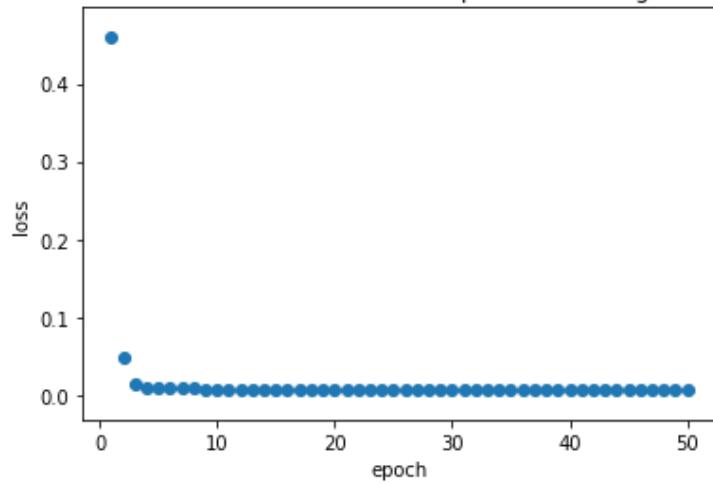
```
1 learning rate: 0.005, number of iterations: 10
```



```
1 learning rate: 0.005, number of iterations: 50
```

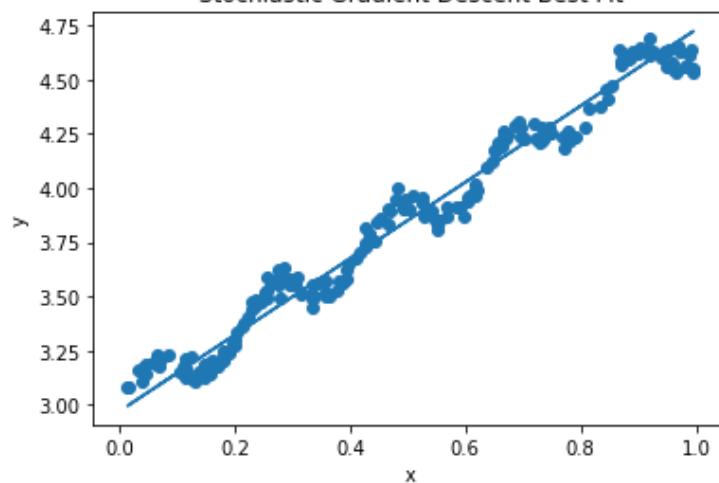


Stochiastic Gradient Descent Mean Epoch vs Training Accuracy

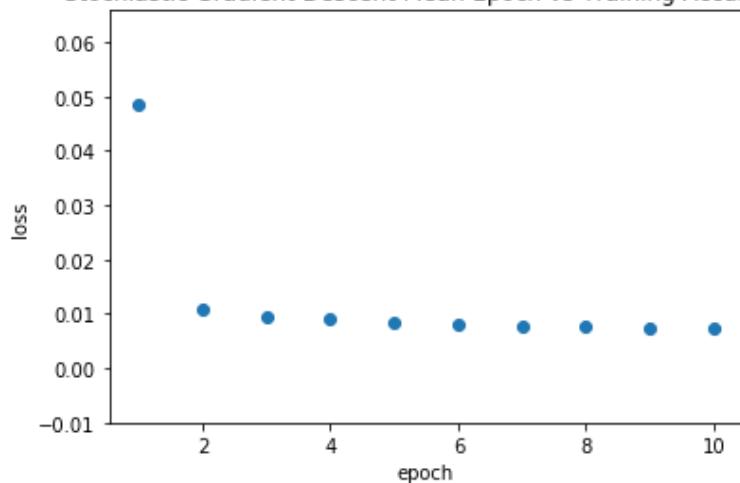


```
1 learning rate: 0.01, number of iterations: 10
```

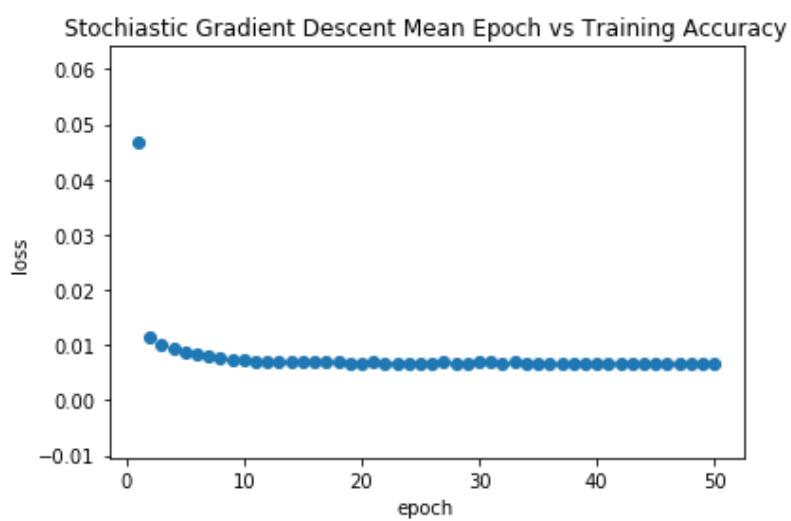
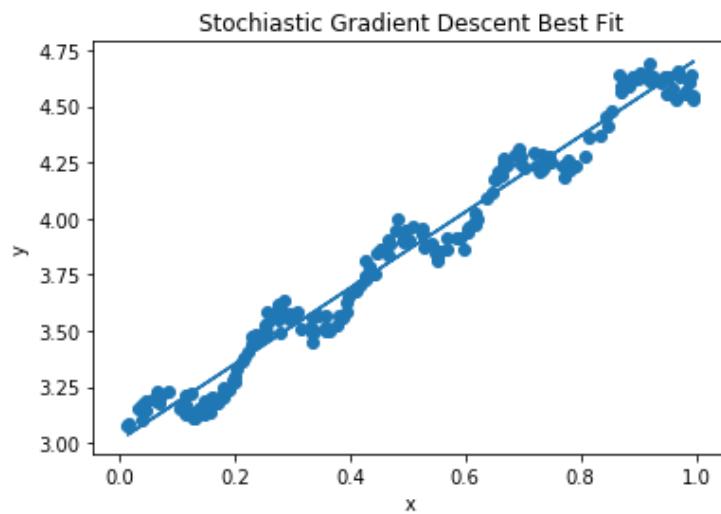
Stochiastic Gradient Descent Best Fit



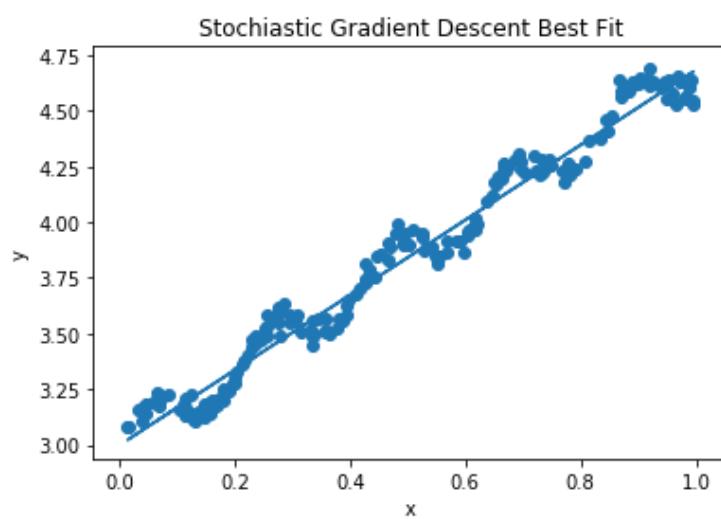
Stochiastic Gradient Descent Mean Epoch vs Training Accuracy



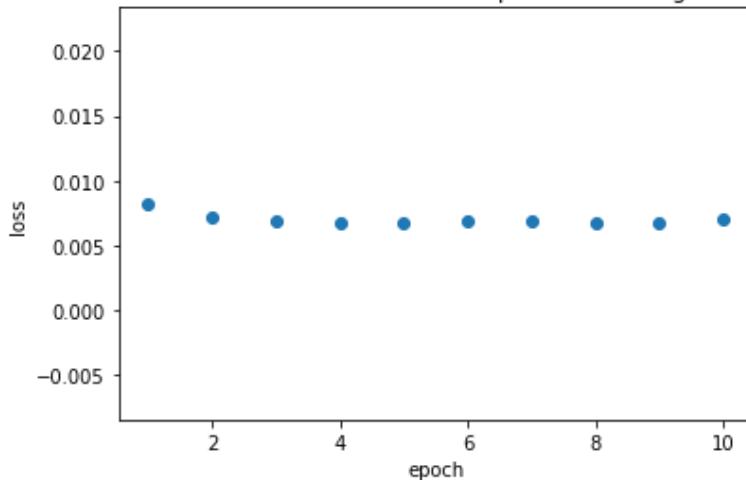
```
1 learning rate: 0.01, number of iterations: 50
```



```
1 learning rate: 0.05, number of iterations: 10
```

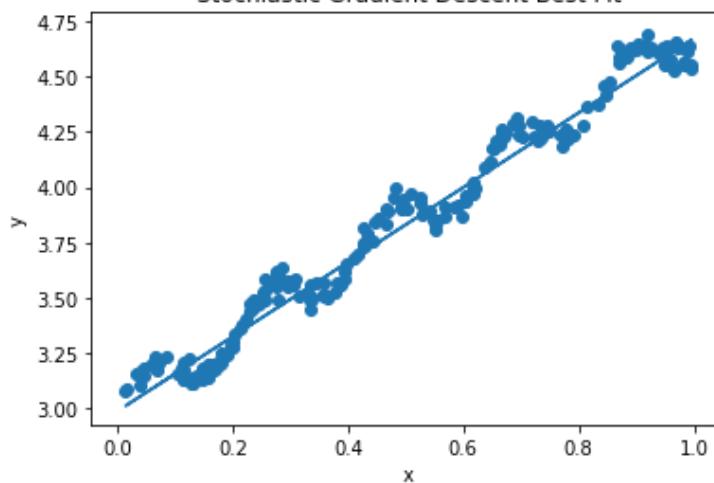


Stochastic Gradient Descent Mean Epoch vs Training Accuracy

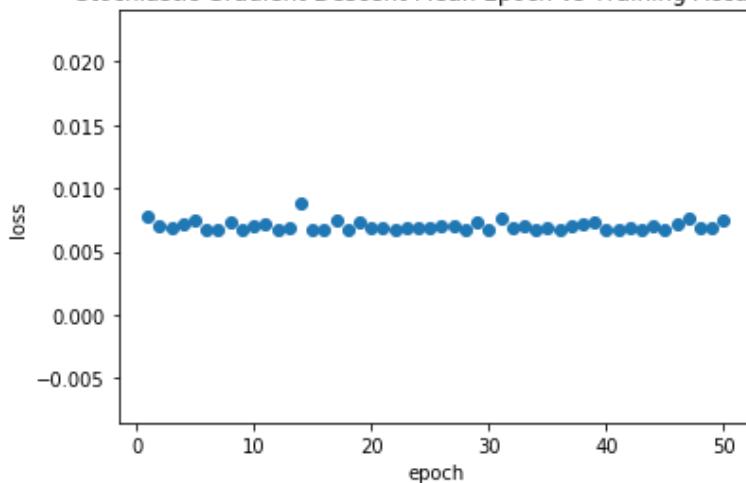


```
1 learning rate: 0.05, number of iterations: 50
```

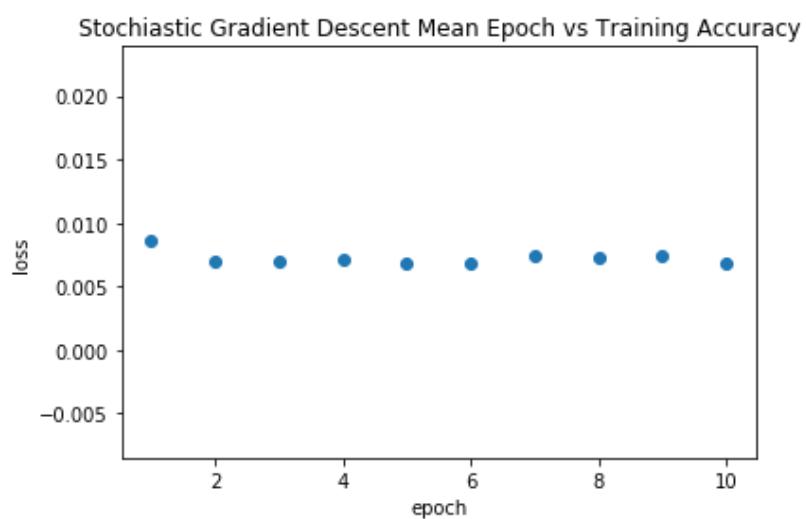
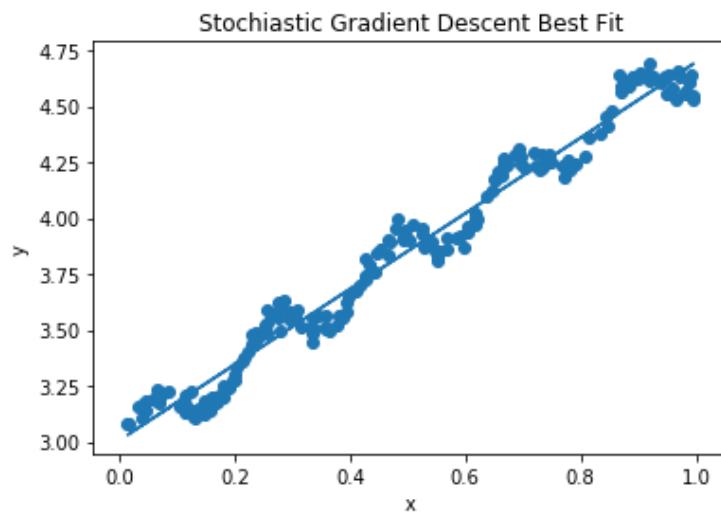
Stochastic Gradient Descent Best Fit



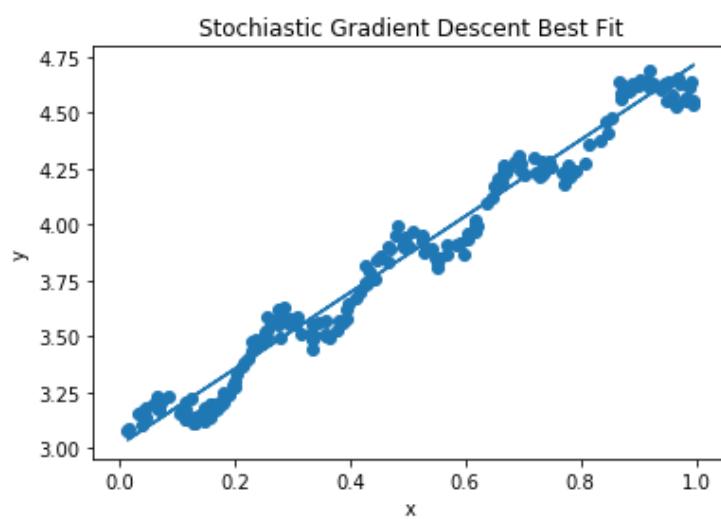
Stochastic Gradient Descent Mean Epoch vs Training Accuracy

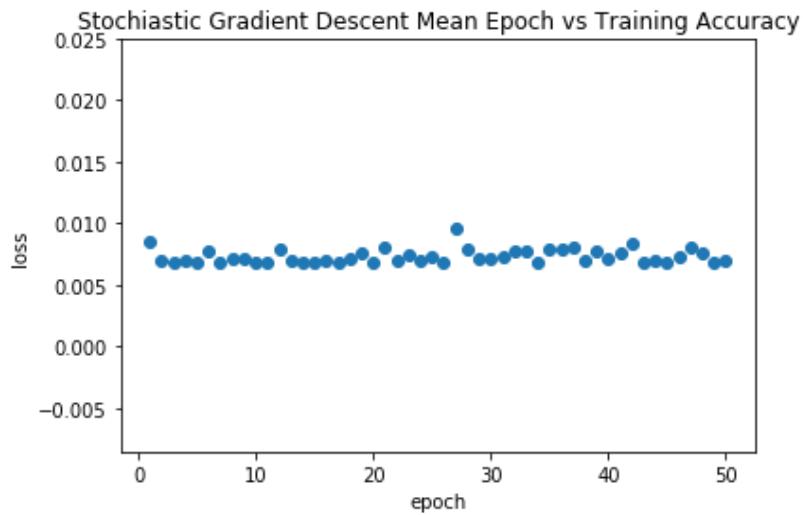


```
1 learning rate: 0.1, number of iterations: 10
```

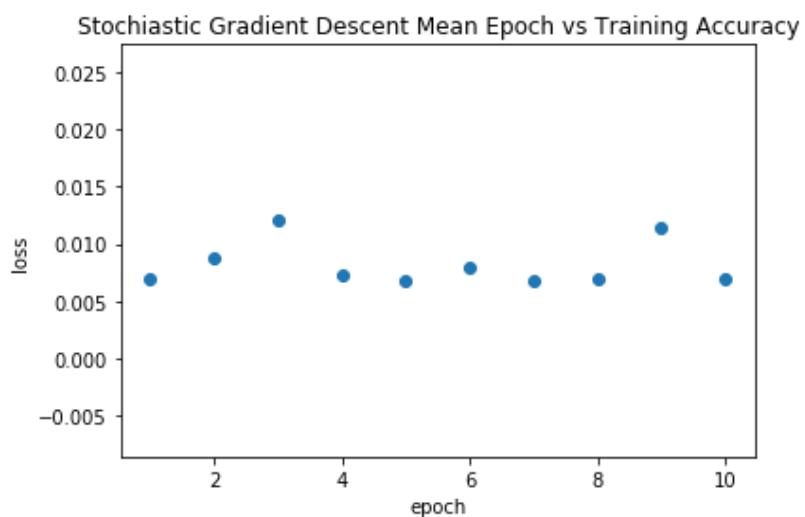
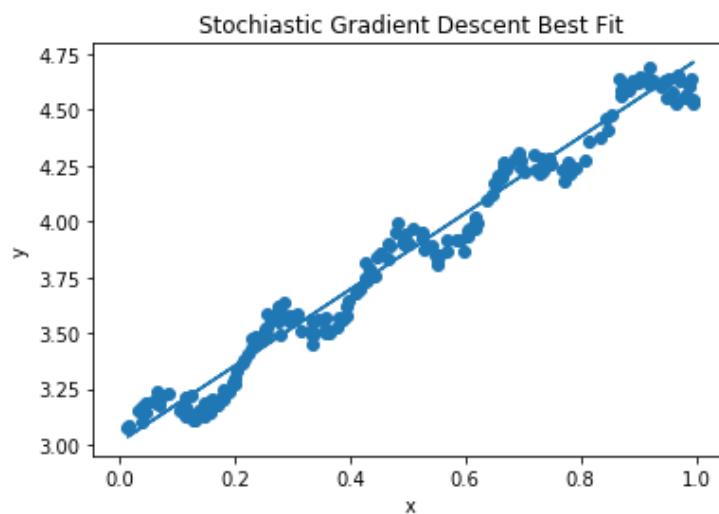


```
1 learning rate: 0.1, number of iterations: 50
```

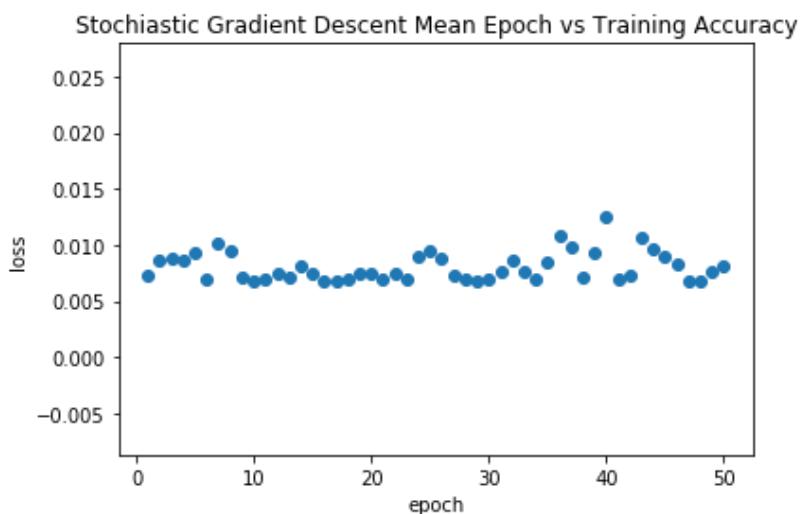
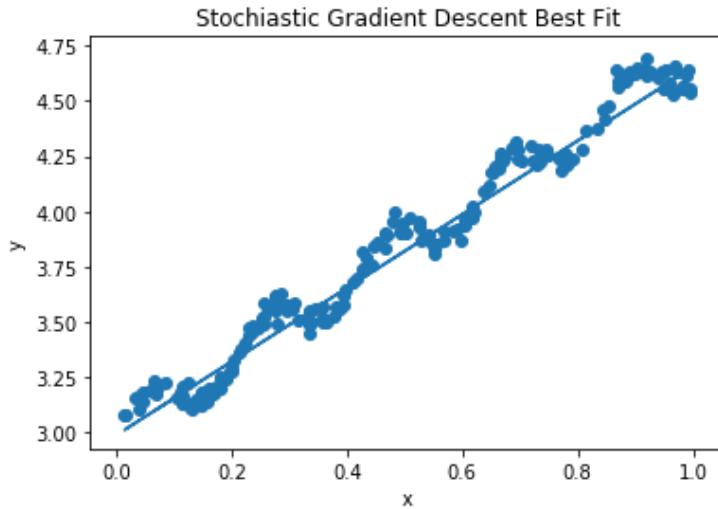




```
1 learning rate: 0.3, number of iterations: 10
```



```
1 learning rate: 0.3, number of iterations: 50
```



## Observation

As we can see in the above figure, even with a learning rate of 0.3, SGD still managed to fit pretty well to the data points. We can also see the loss fluctuated slightly but didn't diverge too much.

## Minibatch Gradient Descent

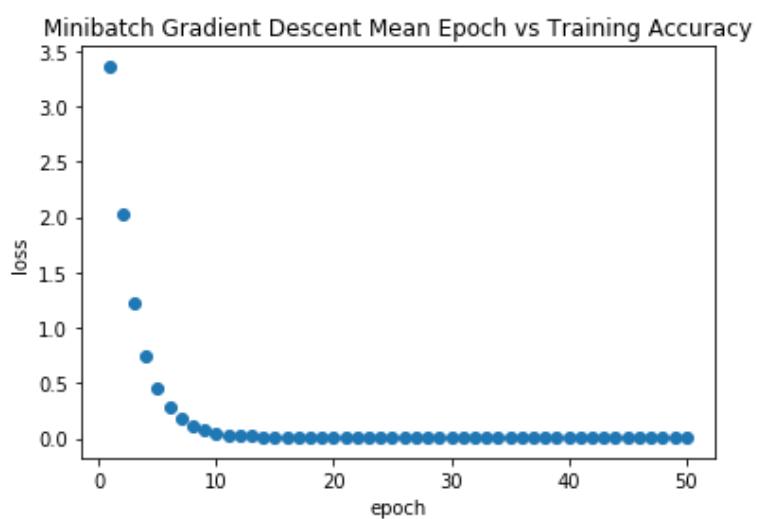
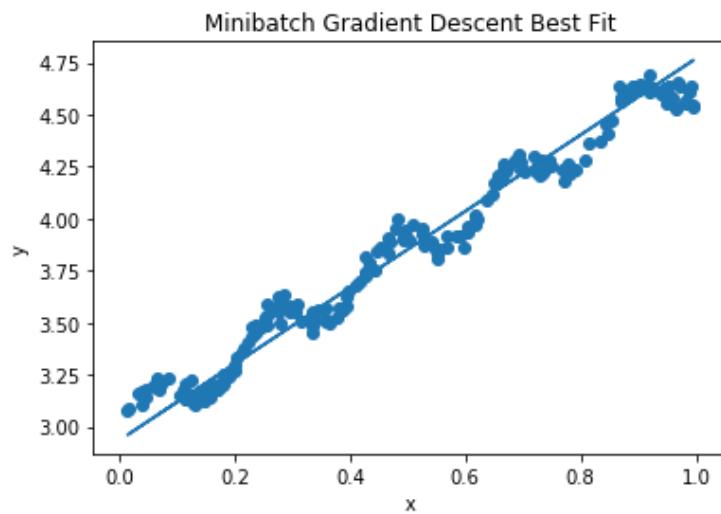
### Result

```

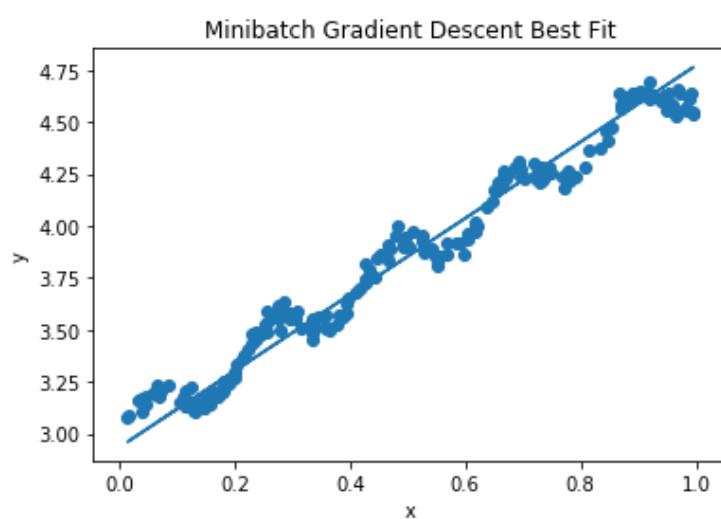
1  learning_rate_list = [0.001, 0.005, 0.01, 0.05, 0.1, 0.3]
2  for learning_rate in learning_rate_list:
3      for batch_size in [10, 50, 100]:
4          print('learning rate: {}, batch size: {}, number of iterations:
5              50'.format(learning_rate, batch_size))
5          thetas = minibatch_gradient_descent(x, y, learning_rate, 50, batch_size)
6          plot(x, y, thetas[-1], "Minibatch Gradient Descent Best Fit")
7          # print(thetas)
8          plot_training_errors(
9              x, y, thetas, "Minibatch Gradient Descent Mean Epoch vs Training Accuracy")

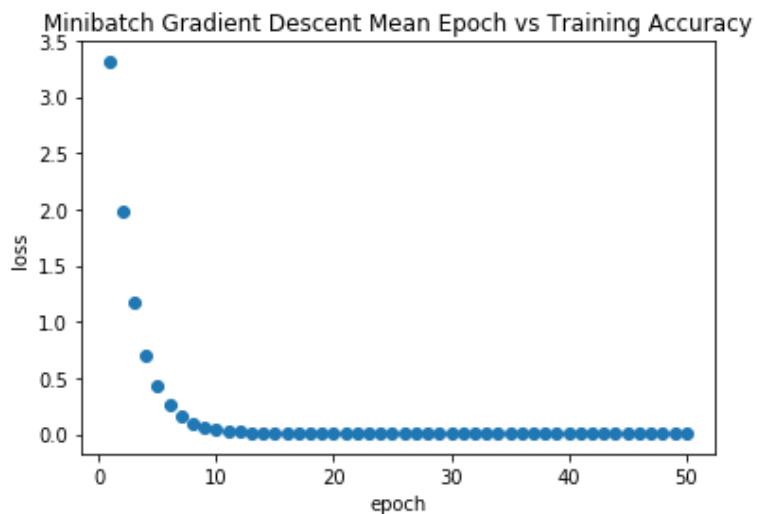
```

```
1  learning rate: 0.001, batch size: 10, number of iterations: 50
```

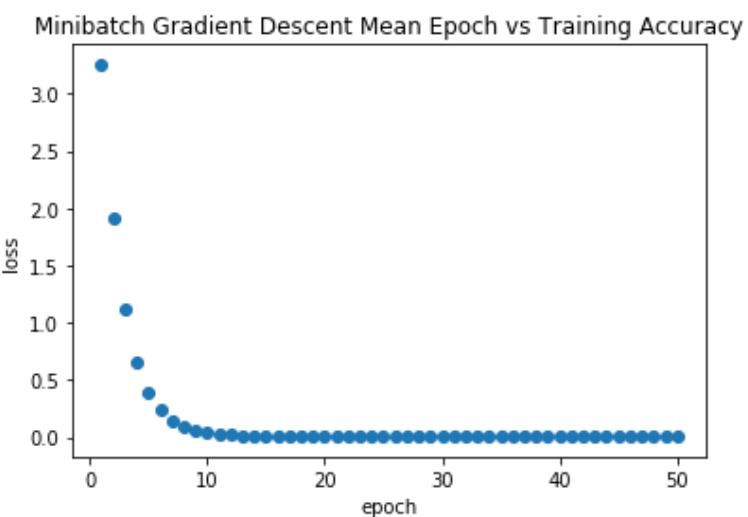
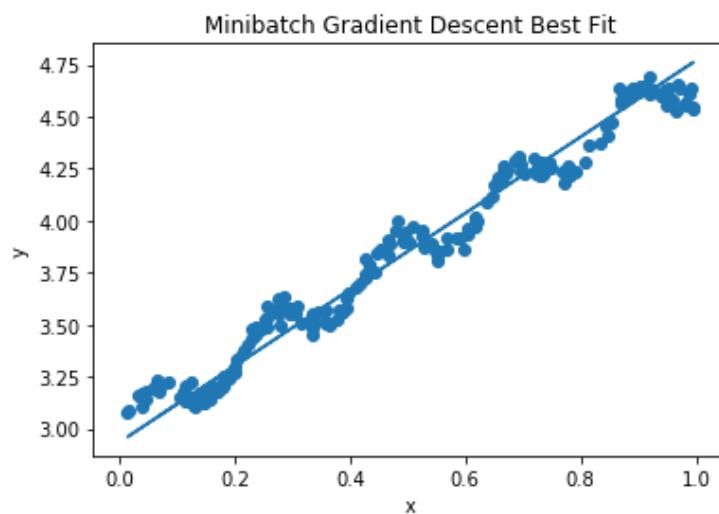


```
1 learning rate: 0.001, batch size: 50, number of iterations: 50
```

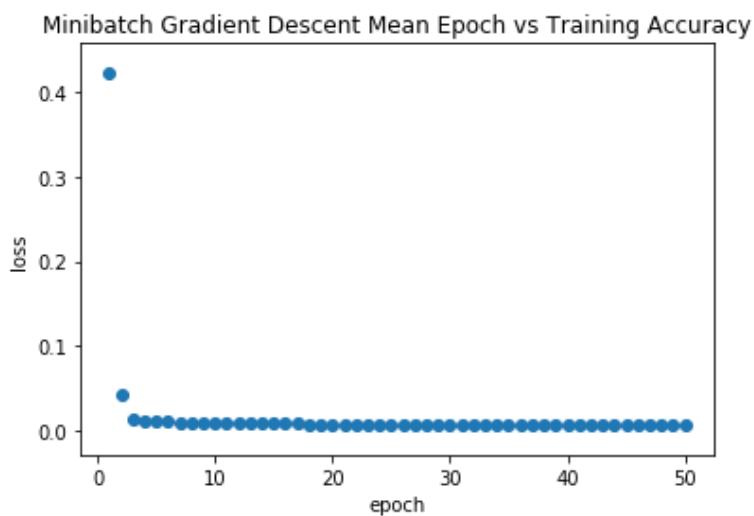
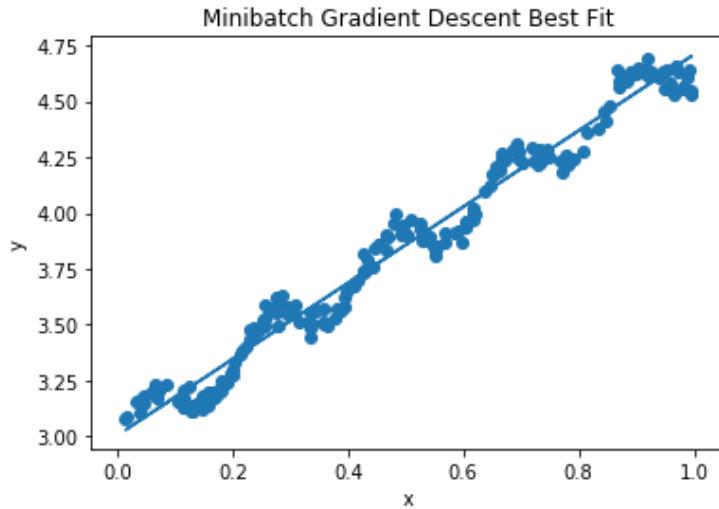




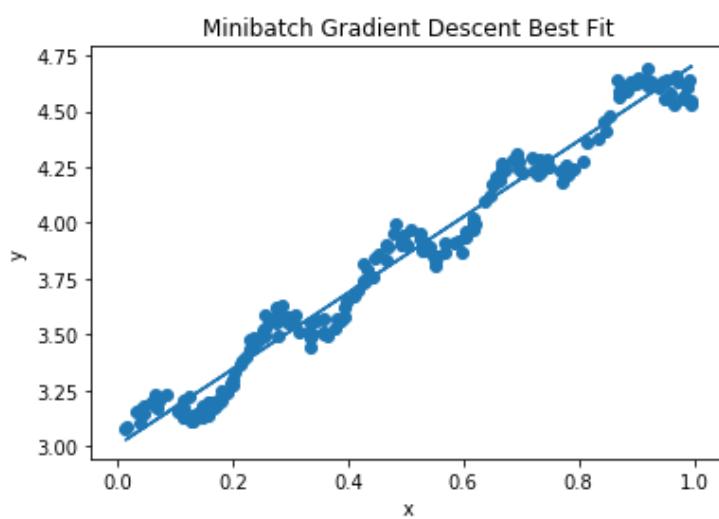
```
1 learning rate: 0.001, batch size: 100, number of iterations: 50
```

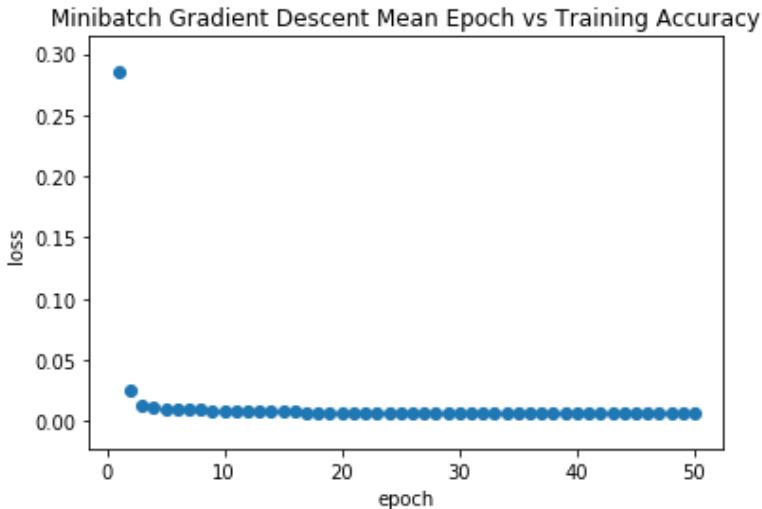


```
1 learning rate: 0.005, batch size: 10, number of iterations: 50
```

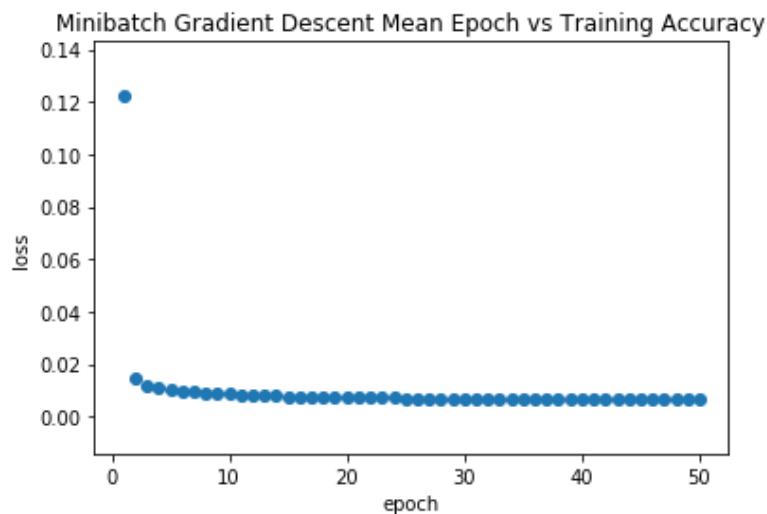
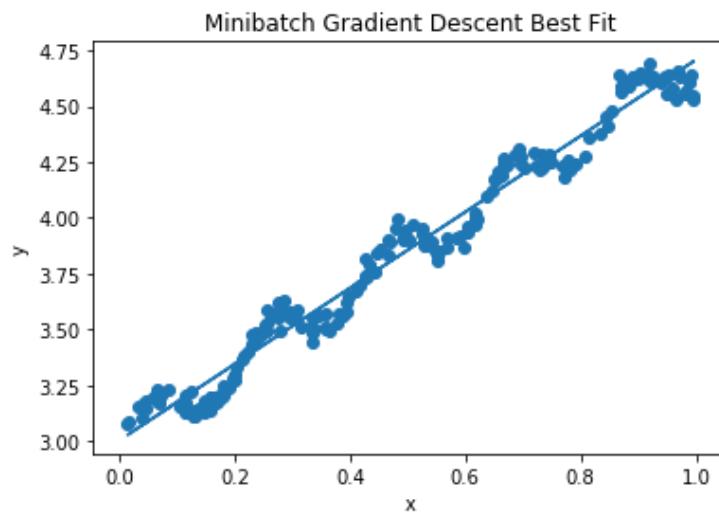


```
1 learning rate: 0.005, batch size: 50, number of iterations: 50
```

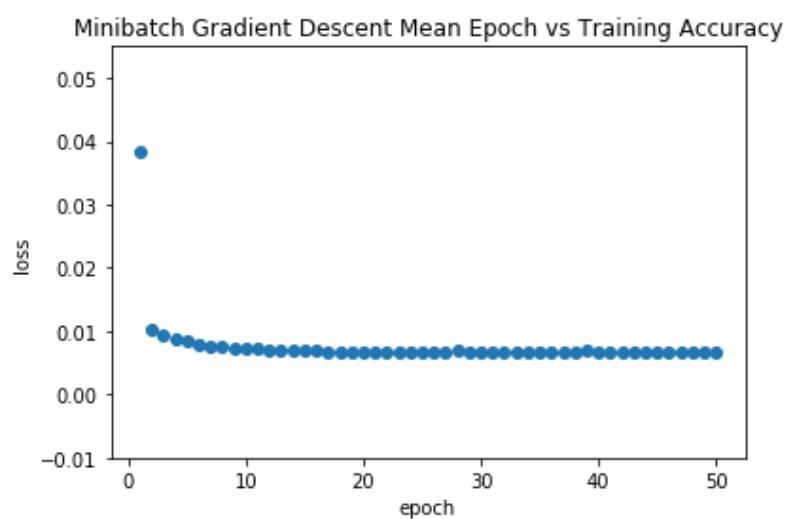
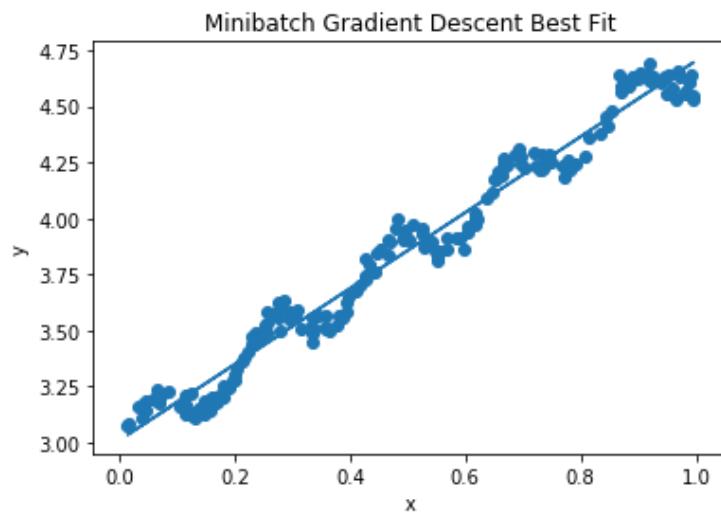




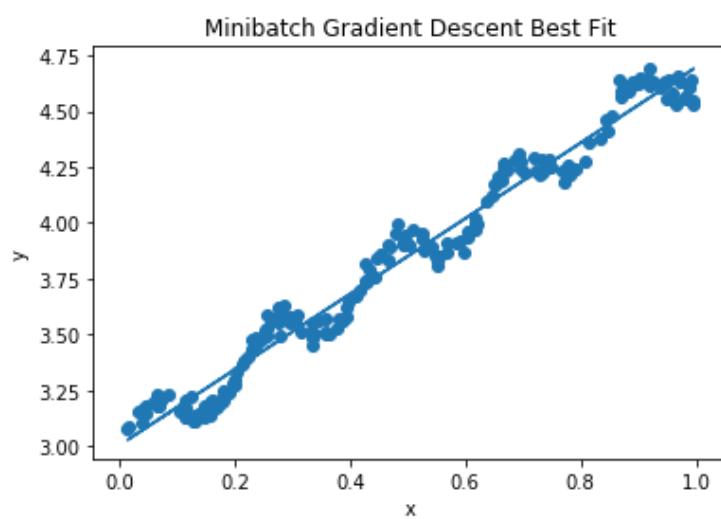
```
1 learning rate: 0.005, batch size: 100, number of iterations: 50
```



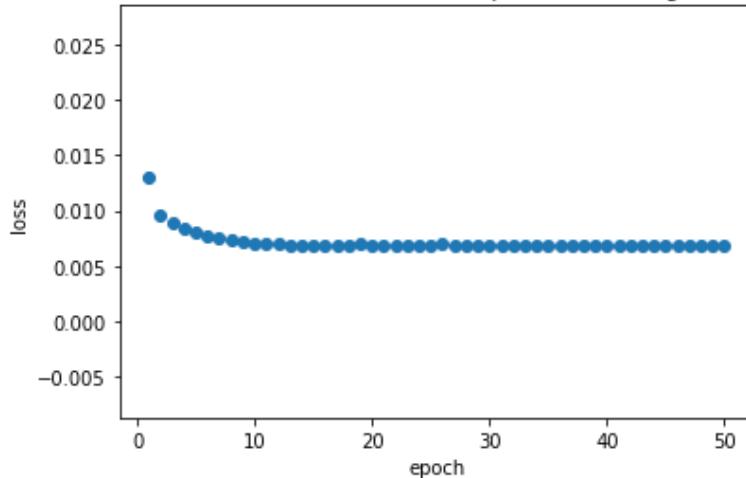
```
1 learning rate: 0.01, batch size: 10, number of iterations: 50
```



```
1 learning rate: 0.01, batch size: 50, number of iterations: 50
```

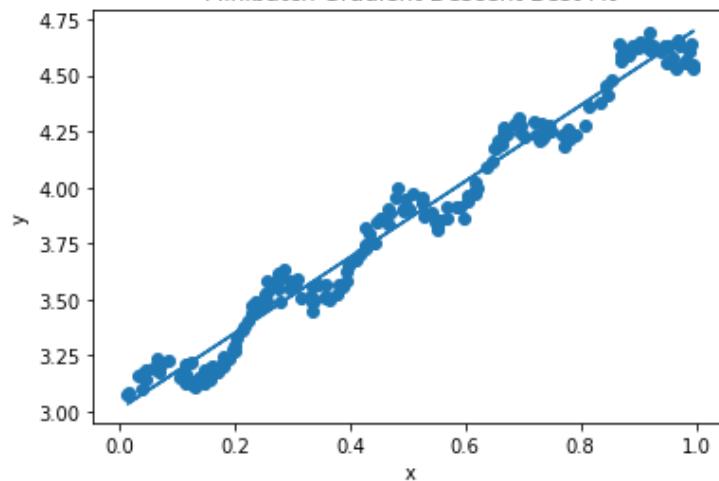


Minibatch Gradient Descent Mean Epoch vs Training Accuracy

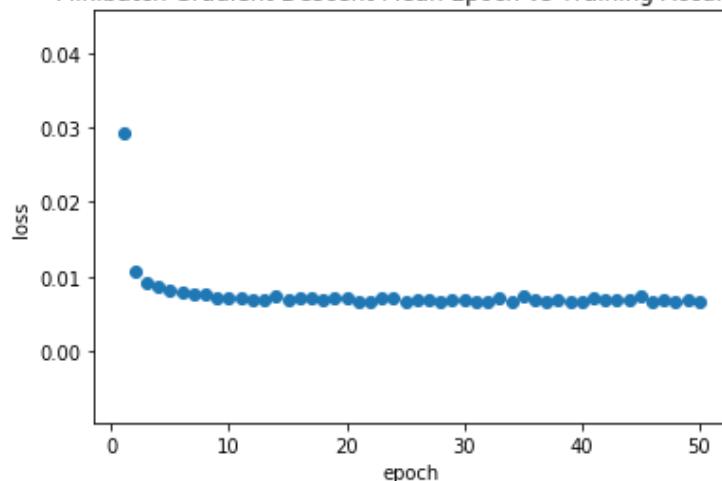


```
1 learning rate: 0.01, batch size: 100, number of iterations: 50
```

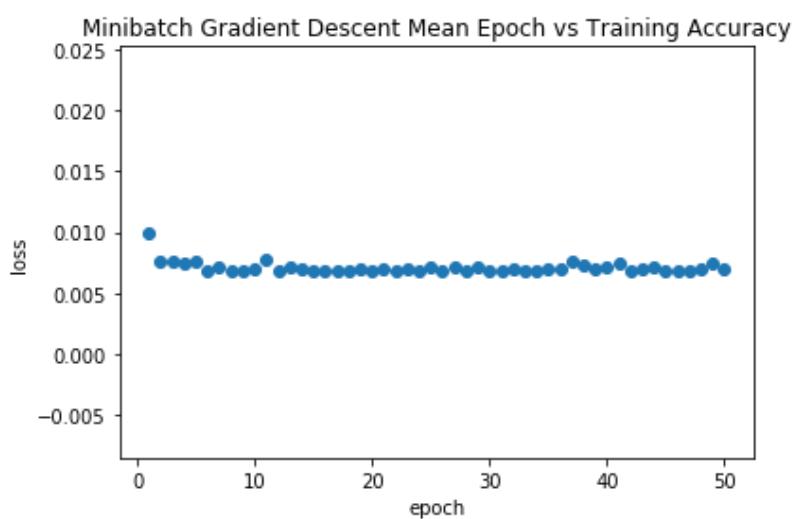
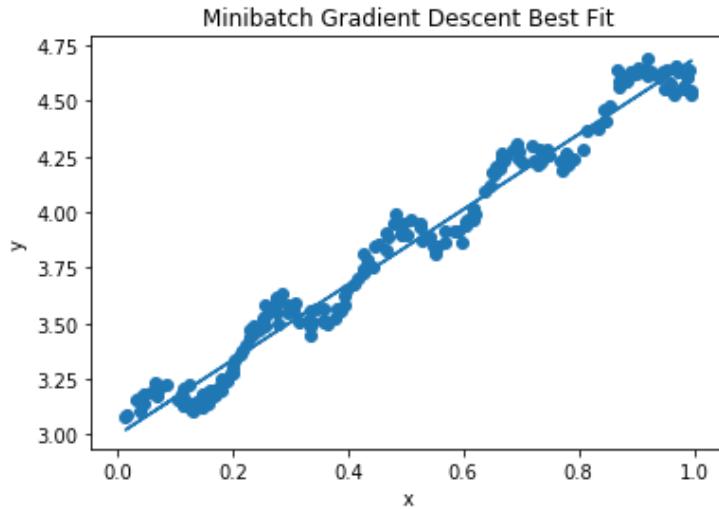
Minibatch Gradient Descent Best Fit



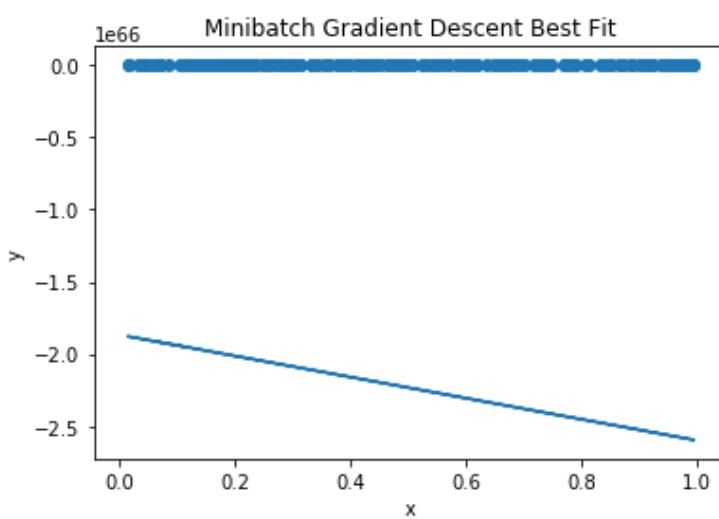
Minibatch Gradient Descent Mean Epoch vs Training Accuracy



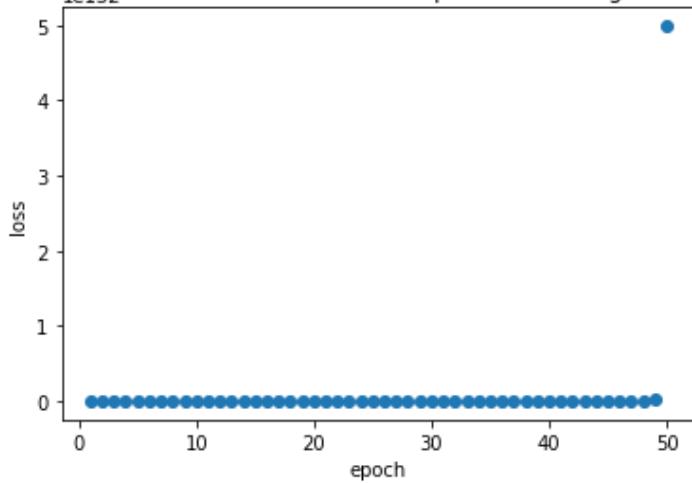
```
1 learning rate: 0.05, batch size: 10, number of iterations: 50
```



```
1 learning rate: 0.05, batch size: 50, number of iterations: 50
```

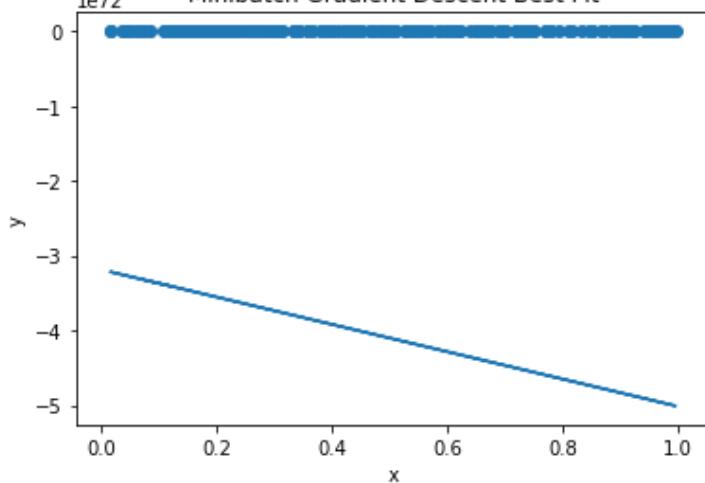


Minibatch Gradient Descent Mean Epoch vs Training Accuracy

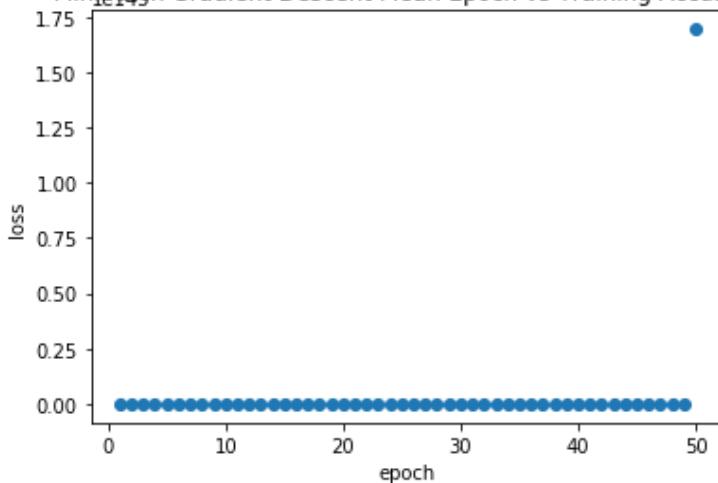


```
1 learning rate: 0.05, batch size: 100, number of iterations: 50
```

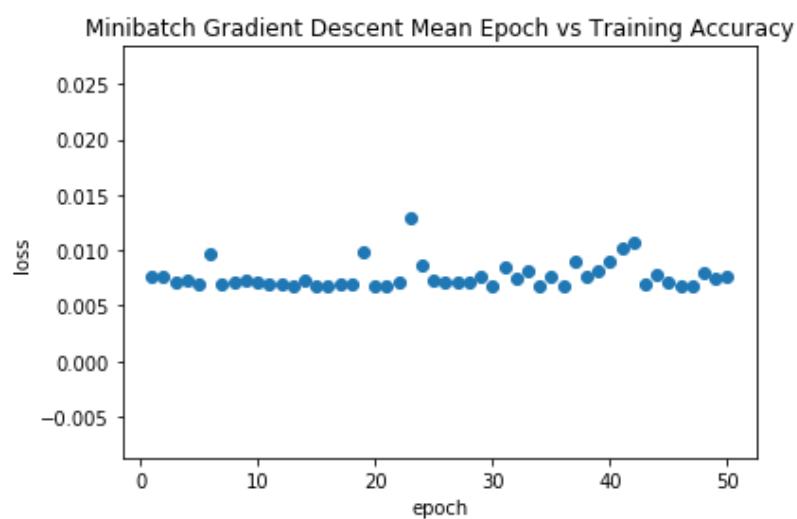
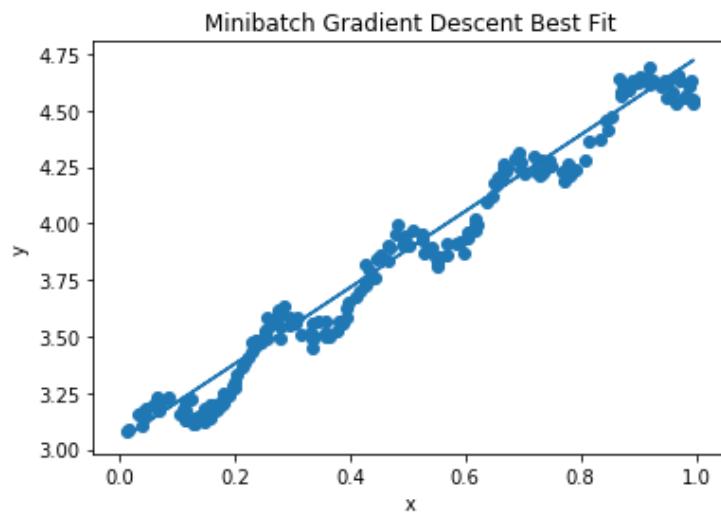
Minibatch Gradient Descent Best Fit



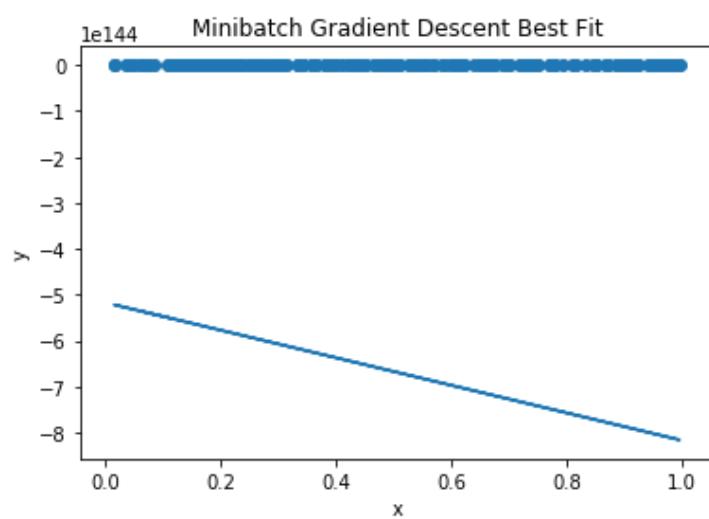
Minibatch Gradient Descent Mean Epoch vs Training Accuracy



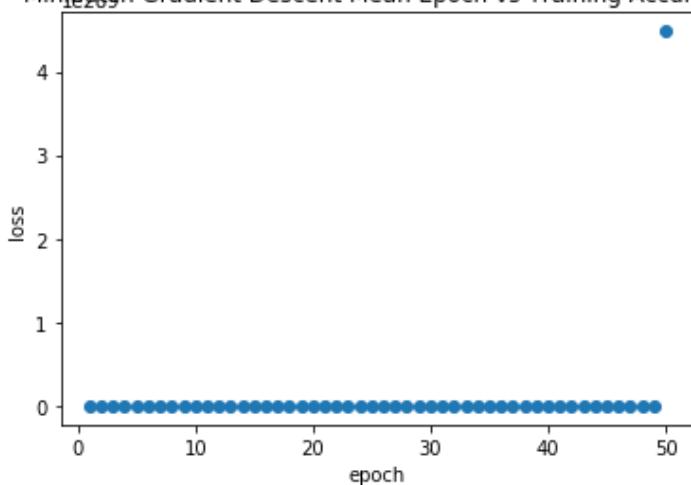
```
1 learning rate: 0.1, batch size: 10, number of iterations: 50
```



```
1 learning rate: 0.1, batch size: 50, number of iterations: 50
```

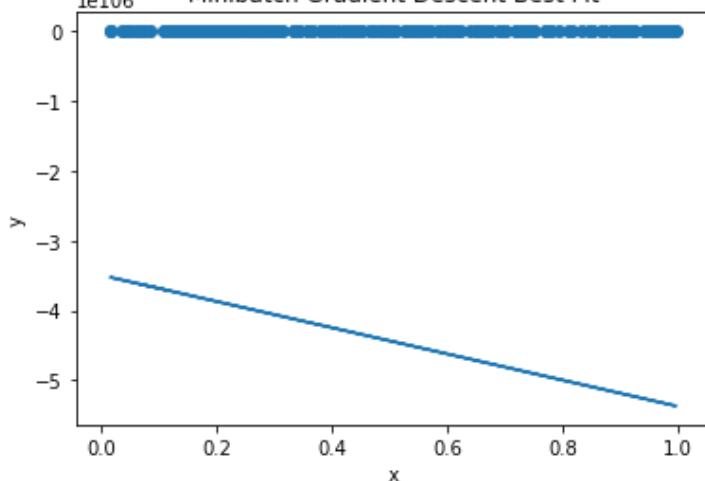


Minibatch Gradient Descent Mean Epoch vs Training Accuracy

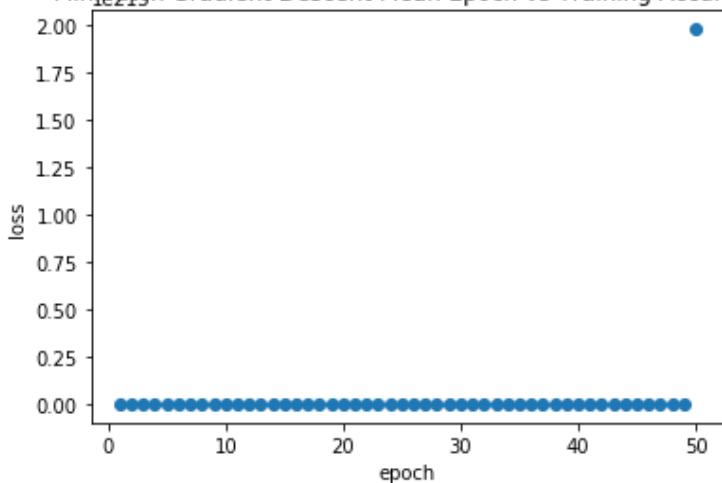


```
1 learning rate: 0.1, batch size: 100, number of iterations: 50
```

Minibatch Gradient Descent Best Fit

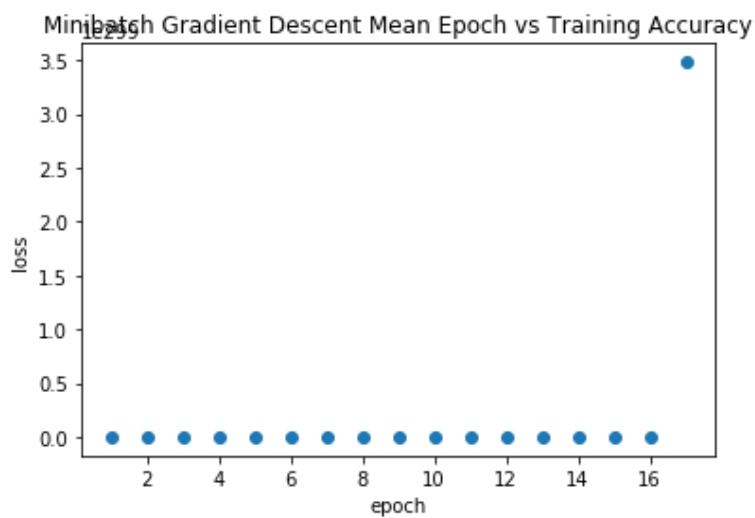
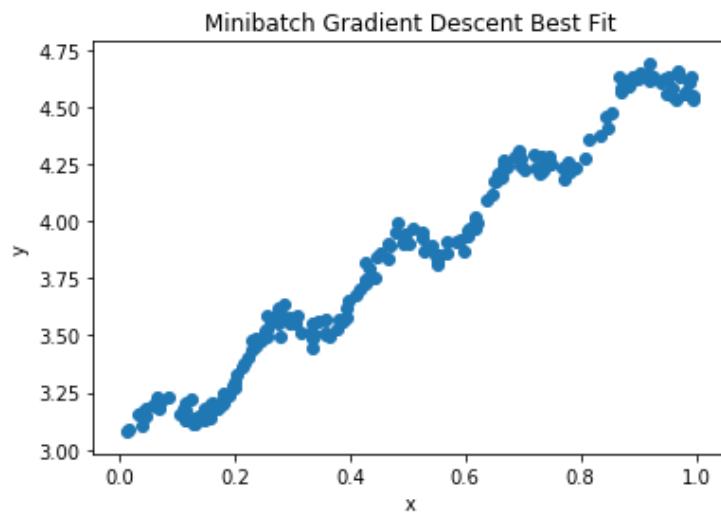


Minibatch Gradient Descent Mean Epoch vs Training Accuracy

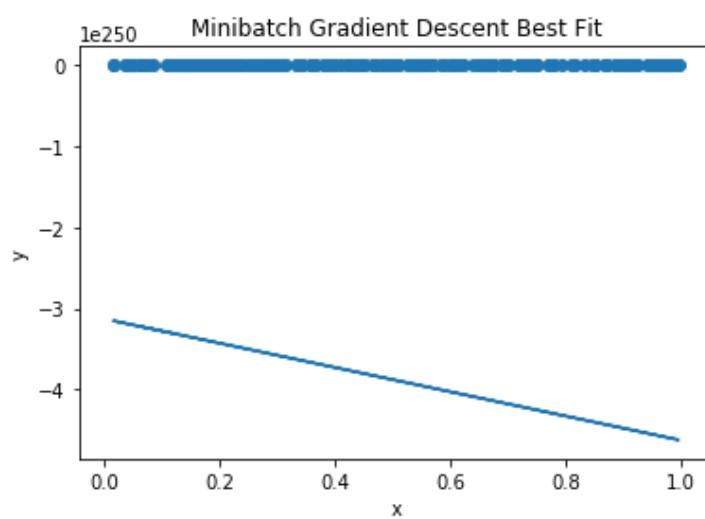


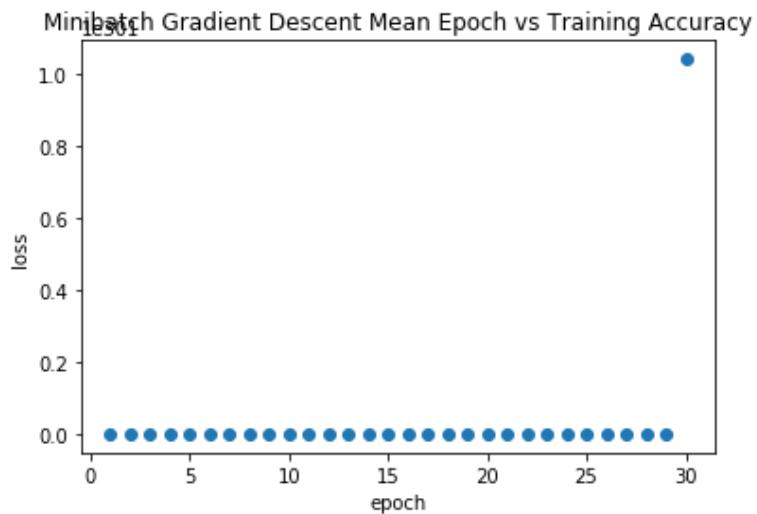
```
1 learning rate: 0.3, batch size: 10, number of iterations: 50
```

```
1  C:\Users\james\Anaconda3\lib\site-packages\ipykernel_launcher.py:93: RuntimeWarning:  
    overflow encountered in subtract  
2  C:\Users\james\Anaconda3\lib\site-packages\ipykernel_launcher.py:96: RuntimeWarning:  
    invalid value encountered in add
```

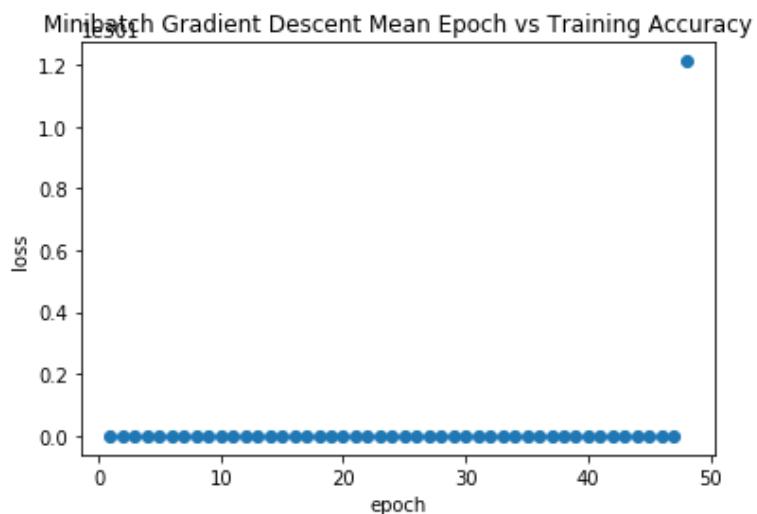
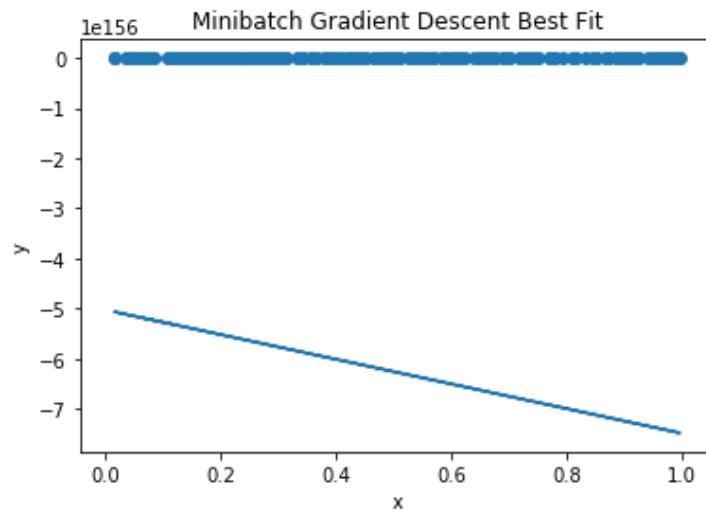


```
1  learning rate: 0.3, batch size: 50, number of iterations: 50
```





```
1 learning rate: 0.3, batch size: 100, number of iterations: 50
```



## Observation

From observing the results above, we can see when the batch size is larger, we need to tone down the learning rate in order to avoid the diverging.

### 3 Sample Exam Questions:

#### 3.1

The MSE will be 0 as every point in the data set sits perfectly on the same line, so we can use linear regression to find a line to fit the data set perfectly.

#### 3.2

(a)

3.2	X	Y
P <sub>1</sub>	0	2
P <sub>2</sub>	2	2
P <sub>3</sub>	3	1

(a)

$$\text{Leave } P_1: y = -x + 4$$

"Error

$$(4 - 2)^2 = 4$$

$$\text{Leave } P_2: y = -\frac{1}{3}x + 2 \quad \left(-\frac{2}{3} + 2 - 2\right)^2 = \frac{4}{9}$$

$$\text{Leave } P_3: y = 2$$

$$(2 - 1)^2 = 1$$

$$\text{Mean error} = \frac{(4 + \frac{4}{9} + 1)}{3} = \frac{\frac{49}{9}}{3} = \frac{49}{27}$$

(b)

$$b) \beta = \beta_0$$

Leave  $\rho$ ,  $J(\beta_0) = (\beta_0 - 2)^2 + (\beta_0 - 1)^2$

$$= \beta_0^2 - 4\beta_0 + 4 + \beta_0^2 - 2\beta_0 + 1$$
$$= 2\beta_0^2 - 6\beta_0 + 5$$

$$\frac{dJ}{d\beta_0} = 4\beta_0 - 6$$
$$\beta_0 = \frac{3}{2}$$

$$\text{error} = \left(\frac{3}{2} - 2\right)^2 = \frac{1}{4}$$

Leave  $P_2$ :  $\beta_0 = \frac{3}{2}$  (mid point of  $P_1$  and  $P_3$ )

$$\text{error: } \left(\frac{3}{2} - 2\right)^2 = \frac{1}{4}$$

Leave  $P_3$ :  $\beta_0 = 2$

$$\text{error: } (2 - 1)^2 = 1$$

$$\text{mean error: } \left(\frac{1}{4} + \frac{1}{4} + 1\right) \div 3 = \frac{1}{2}$$

(c) The second model, since it produced a smaller error

### 3.3

(e) A, because the smaller the training data set is, the easier it is for us to find a line that fits the data points. Two points, for example, can be perfectly fit with 0 training error.

(f) B, because as we increase the size of the training data set, the model should generalize better and be able to make better predictions on the testing data.