# Linear Regression (HW1)

Student: 張皓雲
Student Id: 310551031

# Content

## ● Part. 1, Coding (60%):

### 1. (15%) Plot the learning curve of the training, you should find that loss decreases after a few iterations (x-axis=iteration, y-axis=loss, Matplotlib or other plot tools is available to use)

I used the gradient descent method, mini-batch gradient descent method and stohastic gradient descent method to finish this assignment. The loss curves of the three methods are as follows. Obviously, because only one data point is seen in each iteration of stochastic gradient descent, the loss curve of stochastic gradient descent is more oscillating than that of minibatch gradient descent. The loss curve figure, data point with fitting line figure are packing into the compress file.

#### A. Gradient Descent Loss Curve

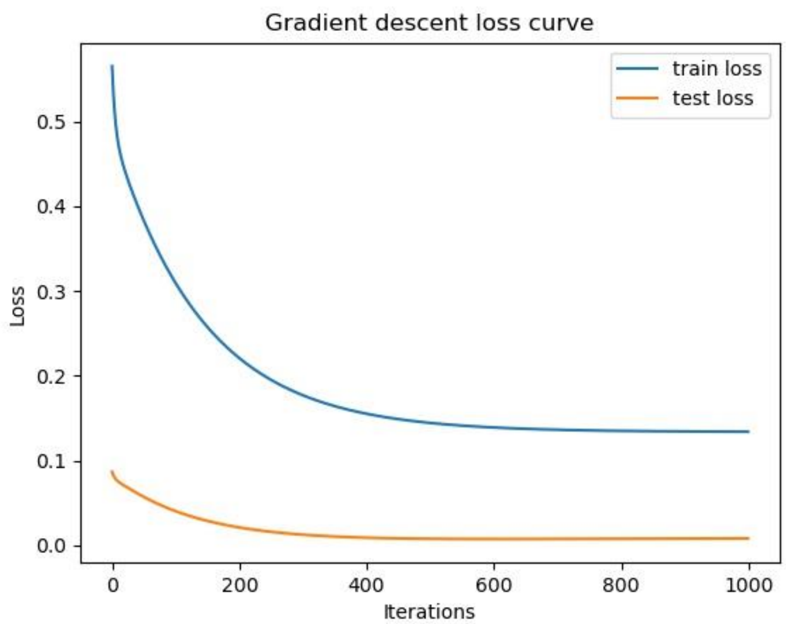



Figure 1. Gradient Descent Loss Curve

### B. Minibatch Gradient Descent Loss Curve

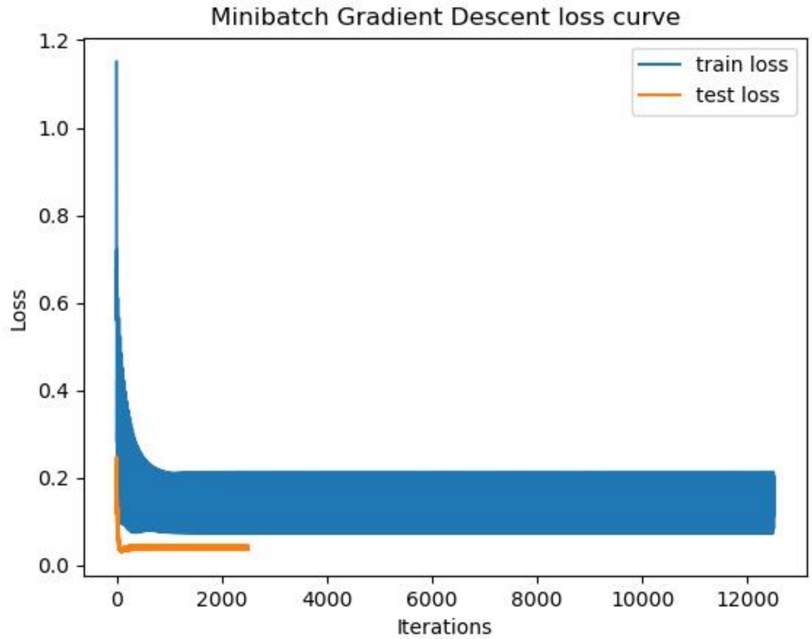



Figure 2. Minibatch Gradient Descent Loss Curve

### C. Stochastic Gradient Descent Loss Curve

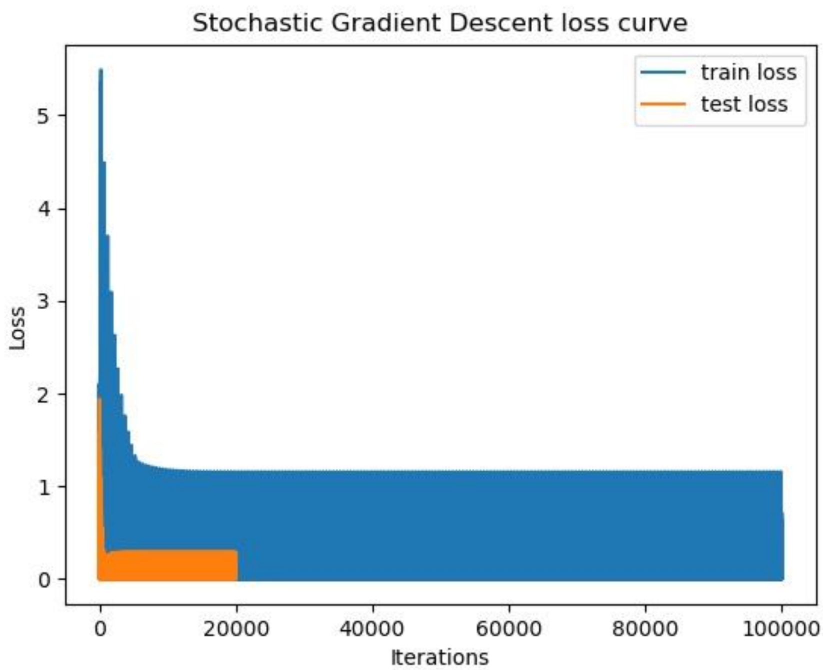



Figure 3. Stochastic Gradient Descent Loss Curve

## 2. (15%) What's the Mean Square Error of your prediction and ground truth (prediction=model(x_test), ground truth=y_test)

In this section, the situation is divided into three parts, which including Gradient Descent, Minibatch Gradient Descent and Stochastic Gradient Descent. The MSE calculation results of the prediction and the ground truth are shown in Figure 7.

### A. Gradient Descent Mean Square Error

This part of the results can be seen in Figure 4. The training loss in Figure 4 refers to the average of the loss results of each batch when training the model, which means the MSE result between Ground Truth and Prediction. The test loss refers to the average value of each batch of loss results when testing the model, that is, the average value of MSE between the ground truth and the prediction. Because in this method, the batch size will be set to the size of the data, so this MSE between Ground Truth and Prediction will be more accurate. The MSE between Ground Truth and Prediction is 0.00830.

```
100%|████████████████████████████████████████| 1/1 [00:00<00:00, 17476.27it/s]
epoch 997, train loss: 0.13386901619226668, test loss: 0.008305728503l3524
100%|████████████████████████████████████████| 1/1 [00:00<00:00, 9709.04it/s]
100%|████████████████████████████████████████| 1/1 [00:00<00:00, 16980.99it/s]
epoch 998, train loss: 0.13386771084797122, test loss: 0.00830720174490206
100%|████████████████████████████████████████| 1/1 [00:00<00:00, 6307.22it/s]
100%|████████████████████████████████████████| 1/1 [00:00<00:00, 14122.24it/s]
epoch 999, train loss: 0.1338664146148669, test loss: 0.008308671067295381
```

Figure 4. Gradient Descent Mean Square Error

## B. Minibatch Gradient Descent Mean Square Error

This part of the results can be seen in Figure 5, and the description is similar to the Figure 4. But in this section, the batch size is 20. The MSE between Ground Truth and Prediction is 0.04124.

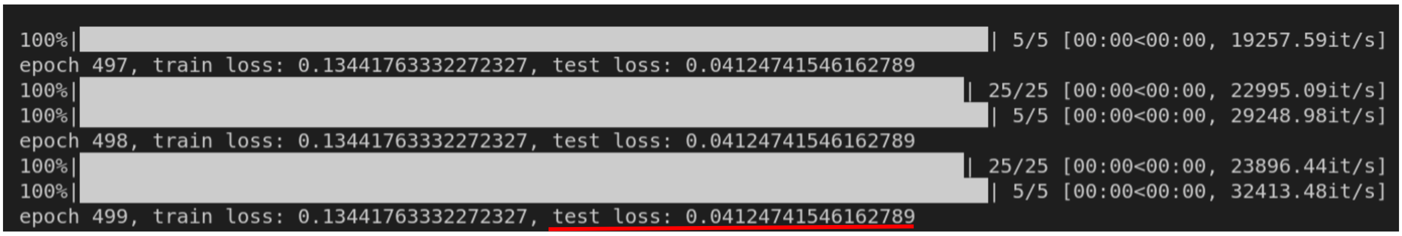



Figure 5. Minibatch Gradient Descent Mean Square Error

## C. Stochastic Gradient Descent Mean Square Error

This part of the results can be seen in Figure 6, and the description is similar to the Figure 4. The MSE between Ground Truth and Prediction is 0.0399215.

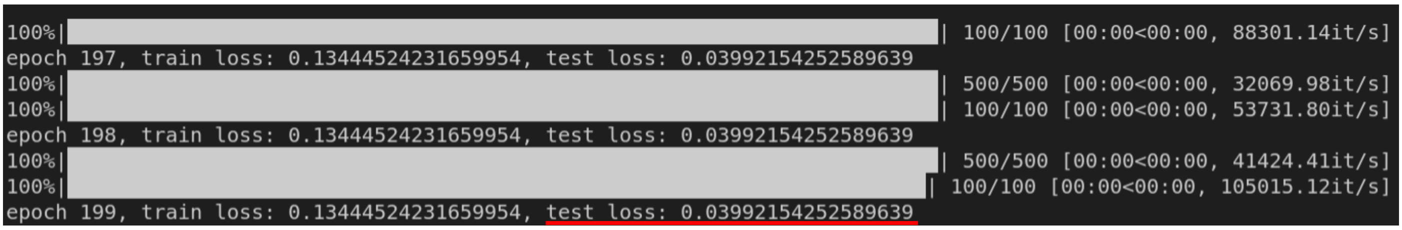



Figure 6. Stochastic Gradient Descent Mean Square Error

## D. Calculate MSE between Prediction and Ground truth

The calculation of MSE (x_test and y_test) can be seen in the code of Figure 7. Line 146 is to find the prediction, which is the process of weight*x_test+bias. As for line 149, it starts to calculate the MSE between prediction and ground truth (y_test). The detailed calculation code of MSE can is shown in Figure 8.

```
142             # testing phase(calculate test loss)
143             for x_batch, y_batch in tqdm(test_batch):
144 
145                 # like y_pred = model(x)
146                 y_pred = np.matmul(x_batch, best_theta)
147 
148                 # calculate mse between prediction and ground truth
149                 loss = criterion(y_pred, y_batch, batch_size)
150 
151                 # save the loss value to plot the figure
152                 test_batch_loss.append(loss)
153                 test_loss_history.append(loss)
```

Figure 7. MSE Calculation

```
def criterion(y_hat, y, m):  # calculate the loss through Mean Square Error
    return 1/(2*m) * np.sum((y_hat - y)**2)
```

Figure 8. MSE Calculation detailed

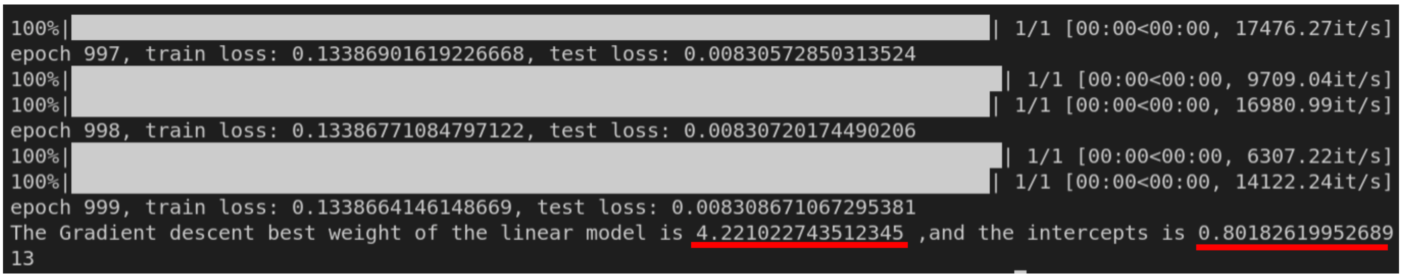
## 3. (15%) What're the weights and intercepts of your linear model?

In this part, apart from dividing this part into gradient descent, Mini batch gradient descent and stochastic gradient descent, I also divide it into **normalize** and **unnormalize** for discussion.

### A. Gradient Descent weights and intercepts

- Normalize

In this assignment, I have saved the best weight and intercepts according to the MSE, and the best weight in gradient descent method is 4.221022, the intercepts is 0.8018.



```
100%|████████████████████████████████████████| 1/1 [00:00<00:00, 17476.27it/s]
epoch 997, train loss: 0.13386901619226668, test loss: 0.008305728503l3524
100%|████████████████████████████████████████| 1/1 [00:00<00:00, 9709.04it/s]
100%|████████████████████████████████████████| 1/1 [00:00<00:00, 16980.99it/s]
epoch 998, train loss: 0.13386771084797122, test loss: 0.00830720174490206
100%|████████████████████████████████████████| 1/1 [00:00<00:00, 6307.22it/s]
100%|████████████████████████████████████████| 1/1 [00:00<00:00, 14122.24it/s]
epoch 999, train loss: 0.1338664146148669, test loss: 0.008308671067295381
The Gradient descent best weight of the linear model is 4.221022743512345 ,and the intercepts is 0.80182619952689
13
```

Figure 9. Gradient Descent weight and intercepts(normalize)

- UnNormalize

The best weight in gradient descent method is 0.8179, the intercepts is 0.7845. In this part, you can see that after normalizing, the weight value will become about 6 times.

```
100%| | 1/1 [00:00<00:00, 8456.26it/s]
100%| | 1/1 [00:00<00:00, 18157.16it/s]
epoch 997, train loss: 0.13368200181610046, test loss: 0.00687029732074477
100%| | 1/1 [00:00<00:00, 10672.53it/s]
100%| | 1/1 [00:00<00:00, 21076.90it/s]
epoch 998, train loss: 0.13368200181610046, test loss: 0.00687029732074477
100%| | 1/1 [00:00<00:00, 8848.74it/s]
100%| | 1/1 [00:00<00:00, 20560.31it/s]
epoch 999, train loss: 0.13368200181610046, test loss: 0.00687029732074477
The Gradient descent best weight of the linear model is 0.8179703770324728 ,and the intercepts is 0.7845650819162
41
```

Figure 10. Gradient Descent weight and intercepts

### B. Minibatch Gradient Descent weights and intercepts

- Normalize

In this assignment, I have saved the best weight and intercepts according to the MSE, and the best weight in minibatch gradient descent method is 4.321306, the intercepts is 0.7733.

```
100%| | 5/5 [00:00<00:00, 19257.59it/s]
epoch 497, train loss: 0.13441763332272327, test loss: 0.04124741546162789
100%| | 25/25 [00:00<00:00, 22995.09it/s]
100%| | 5/5 [00:00<00:00, 29248.98it/s]
epoch 498, train loss: 0.13441763332272327, test loss: 0.04124741546162789
100%| | 25/25 [00:00<00:00, 23896.44it/s]
100%| | 5/5 [00:00<00:00, 32413.48it/s]
epoch 499, train loss: 0.13441763332272327, test loss: 0.04124741546162789
The Minibatch Gradient Descent best weight of the linear model is 4.321306454481022 ,and the intercepts is 0.7733
880702374907
```

Figure 11. Minibatch Gradient Descent weight and intercepts(normalize)

- UnNormalize

The best weight in gradient descent method is 0.80244, the intercepts is 0.75648. In this part, you can see that after normalizing, the weight value will become about 6 times.

```
100%|██████████████████████████████████████████| 25/25 [00:00<00:00, 29232.67it/s]
100%|███████████████████████████████████████████| 5/5 [00:00<00:00, 42711.85it/s]
epoch 497, train loss: 0.13535124205438773, test loss: 0.034816128155235666
100%|██████████████████████████████████████████| 25/25 [00:00<00:00, 29208.25it/s]
100%|███████████████████████████████████████████| 5/5 [00:00<00:00, 42538.58it/s]
epoch 498, train loss: 0.13535124205438773, test loss: 0.034816128155235666
100%|██████████████████████████████████████████| 25/25 [00:00<00:00, 29620.79it/s]
100%|███████████████████████████████████████████| 5/5 [00:00<00:00, 43873.47it/s]
epoch 499, train loss: 0.13535124205438773, test loss: 0.034816128155235666
The Minibatch Gradient Descent best weight of the linear model is 0.8024482255801342 ,and the intercepts is 0.756
4840225346317
```

Figure 12. Minibatch Gradient Descent weight and intercepts

## C. Stochastic Gradient Descent weights and intercepts

- Normalize

In this assignment, I have saved the best weight and intercepts according to the MSE, and the best weight in minibatch gradient descent method is 4.3159, the intercepts is 0.755. In Figure 15, we can see that the saving code, weight and intercepts are saved in the "best_theta" parameter.

```
100%|█████████████████████████████████████████| 100/100 [00:00<00:00, 88301.14it/s]
epoch 197, train loss: 0.13444524231659954, test loss: 0.03992154252589639
100%|█████████████████████████████████████████| 500/500 [00:00<00:00, 32069.98it/s]
100%|█████████████████████████████████████████| 100/100 [00:00<00:00, 53731.80it/s]
epoch 198, train loss: 0.13444524231659954, test loss: 0.03992154252589639
100%|█████████████████████████████████████████| 500/500 [00:00<00:00, 41424.41it/s]
100%|████████████████████████████████████████| 100/100 [00:00<00:00, 105015.12it/s]
epoch 199, train loss: 0.13444524231659954, test loss: 0.03992154252589639
The Stochastic Gradient Descent best weight of the linear model is 4.315912459202712 ,and the intercepts is 0.755
8936600133507
```

Figure 13. Stochastic Gradient Descent weight and intercepts(normalize)

- UnNormalize

The best weight in gradient descent method is 0.796593, the intercepts is 0.7392. In this part, you can see that after normalizing, the weight value will become about 6 times.

```
100%|                                                  | 500/500 [00:00<00:00, 49750.95it/s]
100%|                                                  | 100/100 [00:00<00:00, 99391.09it/s]
epoch 197, train loss: 0.1350201054720081, test loss: 0.035272975863546344
100%|                                                  | 500/500 [00:00<00:00, 48740.37it/s]
100%|                                                  | 100/100 [00:00<00:00, 103768.04it/s]
epoch 198, train loss: 0.1350201054720081, test loss: 0.035272975863546344
100%|                                                  | 500/500 [00:00<00:00, 54989.96it/s]
100%|                                                  | 100/100 [00:00<00:00, 112508.15it/s]
epoch 199, train loss: 0.1350201054720081, test loss: 0.035272975863546344
The Stochastic Gradient Descent best weight of the linear model is 0.7965938046686774 ,and the intercepts is 0.73
92428834352359
```

Figure 14. Stochastic Gradient Descent weight and intercepts

```
for epoch in range(epochs):

    train_batch_loss, test_batch_loss = [], []

    # training phase(calculate train loss)
    for x_batch, y_batch in tqdm(train_batch):

        # something like y_pred = model(x)
        y_pred = np.matmul(x_batch, theta)

        # calculate mse between prediction and ground truth
        loss = criterion(y_pred, y_batch, batch_size)

        # calculate the gradient and do backpropagation
        theta -= lr * \
            compute_gradient(x_batch, y_pred, y_batch, batch_size)

        # save the loss value to plot the figure
        train_batch_loss.append(loss)
        train_loss_history.append(loss)

        # save the best weight and bias
        if min_loss > loss:
            best_theta = theta
```

Figure 15. Saving the best weights and intercepts

## 4. (10%) What's the difference between Gradient Descent, Mini-Batch Gradient Descent, and Stochastic Gradient Descent?

The difference between the three methods is the batch size. The batch size used by gradient descent is the number of training data (500), the batch size of minibatch gradient descent is selected from 1 to 500, and the batch size of stochastic gradient descent is 1. The setting of the hyperparameters including learning rate(lr), batch size(batch_size) ,and number of epochs(epochs) can be seen in the Figure 22.

## A. Gradient Descent weights and intercepts

The figure below is about data point and fitting line, especially during training and testing. During the training stage, because the batch size of the gradient descent is an integer of the data, there should be more epochs during the training.

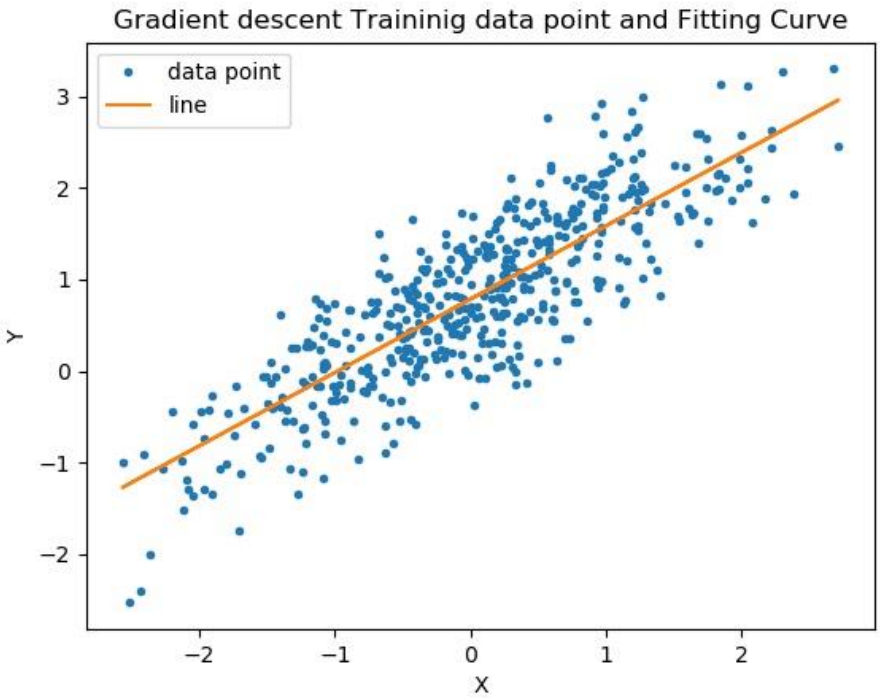



Figure 16. Training data point and fitting line

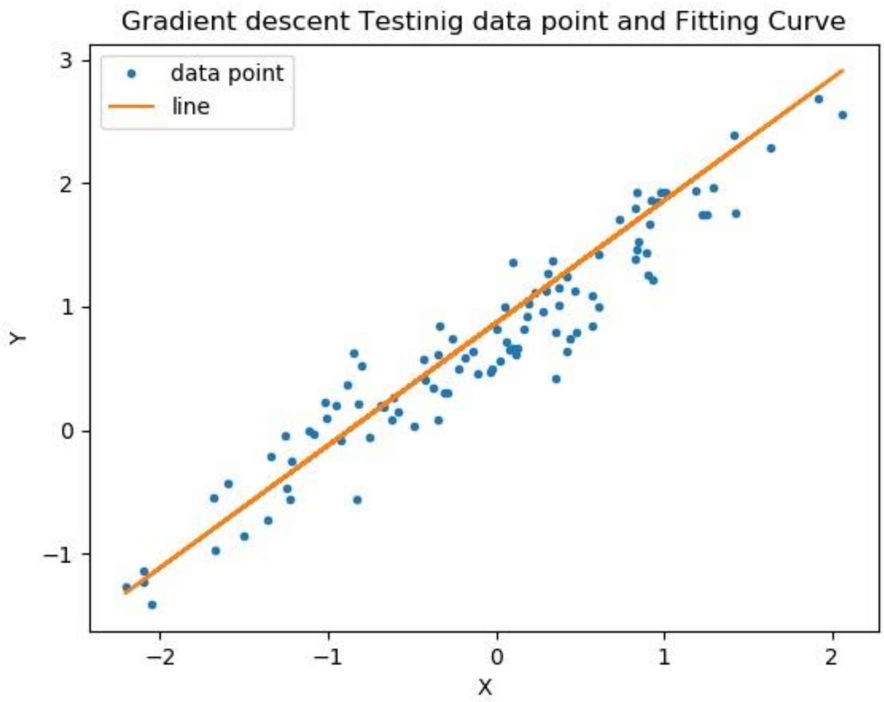



Figure 17. Testing data point and fitting line

**B. Minibatch Gradient Descent weights and intercepts**

The figure below is about data point and fitting line, especially during training and testing. During the training stage, because the batch size of the Minibatch Gradient Descent is the number during 1~whole data, there can have less epochs during the training.

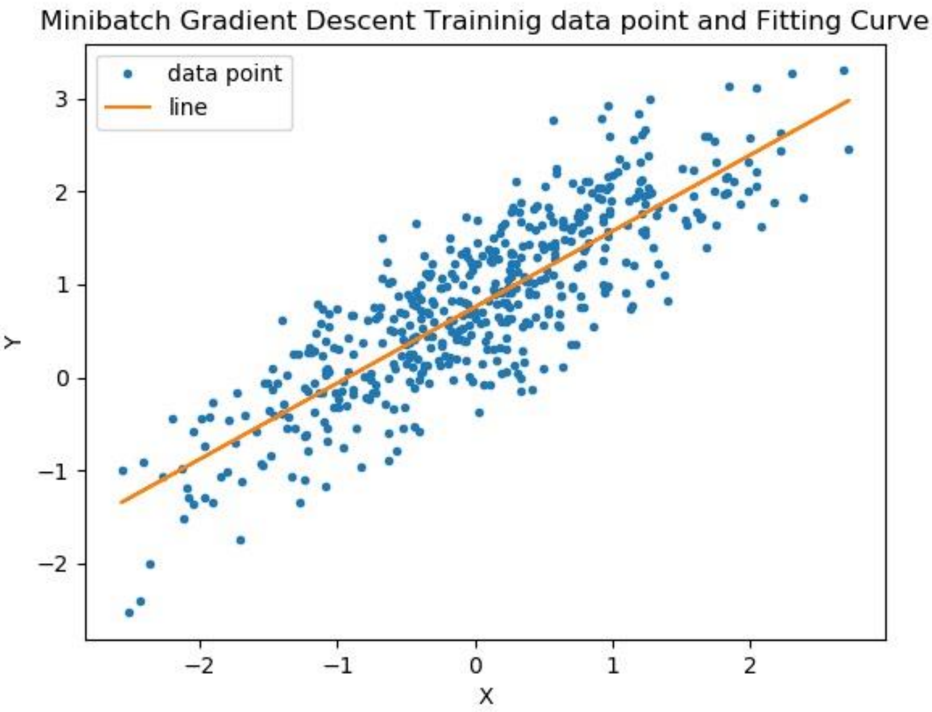



Figure 18. Training data point and fitting line

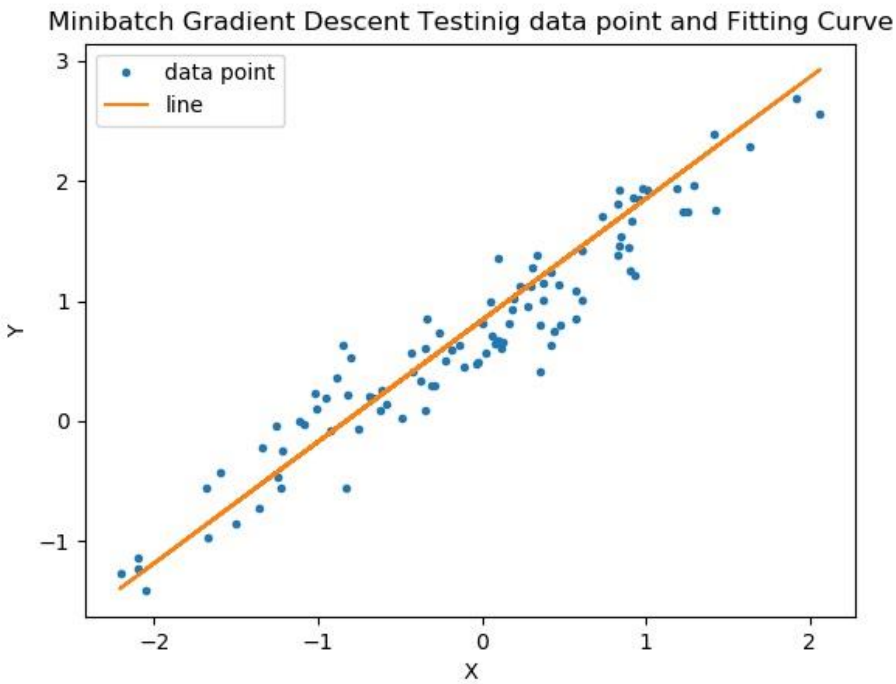



Figure 19. Testing data point and fitting line

## C. Stochastic Gradient Descent weights and intercepts

The figure below is about data point and fitting line, especially during training and testing. During the training stage, because the batch size of the Stochastic Gradient Descent is 1, the number of epochs can less than minibatch gradient descent.

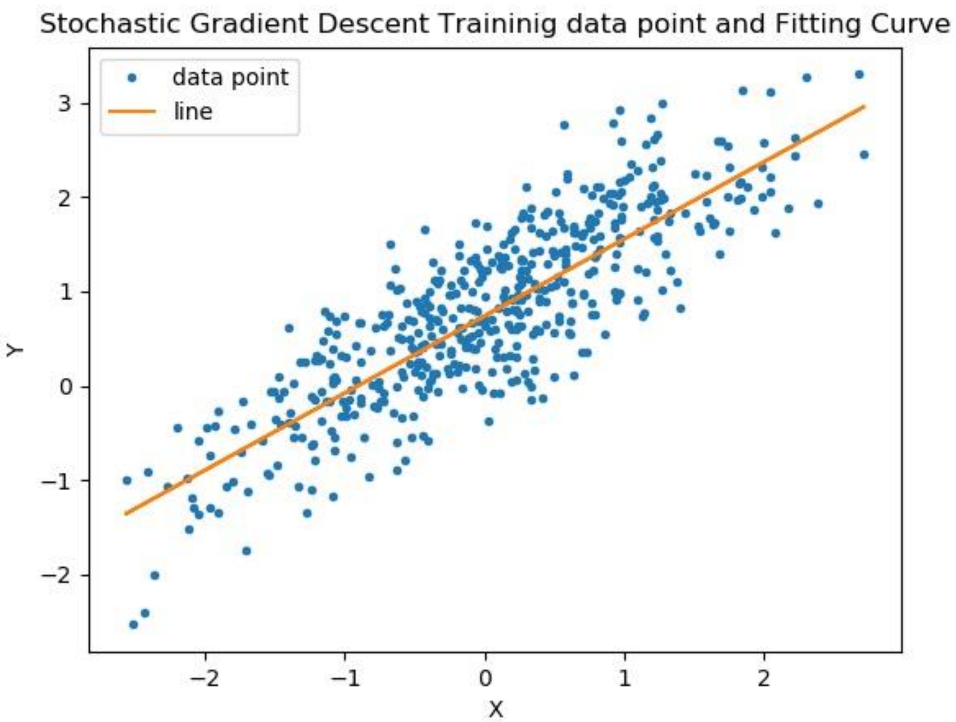



Figure 20. Training data point and fitting line

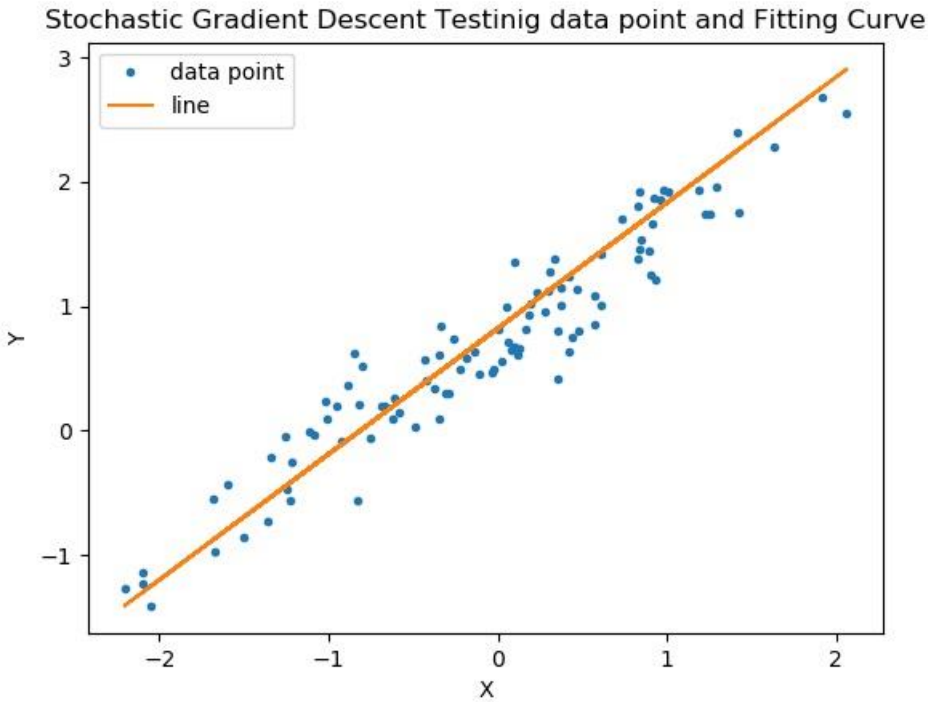



Figure 21. Testing data point and fitting line

```python
    # Gradient descent mode
    if mode == 1:
        lr = 1e-1
        epochs = 1000
        batch_size = 500
        name = "Gradient descent"

    # Minibatch Gradient Descent mode
    elif mode == 2:
        lr = 1e-1
        epochs = 500
        batch_size = 20
        name = "Minibatch Gradient Descent"

    # Stochastic Gradient Descent mode
    elif mode == 3:
        lr = 1e-2
        epochs = 200
        batch_size = 1
        name = "Stochastic Gradient Descent"
```

Figure 22. The hyperparameters in each gradient descent method

- **Part. 2, Questions (40%)**

系所:資料工所. 學號: 31051031 姓名: 張晧雲.

Part 2. Questions.

1.

| | apples | oranges | guavas | 選到機率 |
|---|---|---|---|---|
| R | 3 | 4 | 3 | 0.2 |
| B | 2 | 0 | 2 | 0.4 |
| G | 12 | 4 | 4 | 0.4 |

(1).

$$\underbrace{P(g)}_{\text{guava}} = \underbrace{P(R)}_{\text{Red}} P(g|R) + \underbrace{P(B)}_{\text{Blue}} P(g|B) + \underbrace{P(G)}_{\text{Green}} P(g|G)$$

$$= 0.2 \times 0.3 + 0.4 \times 0.5 + 0.4 \times 0.2$$

$$= 0.34 \quad ※$$

(2).

$$P(\underbrace{B}_{\text{Blue}} | \underbrace{a}_{\text{apple}}) = \frac{P(a|B)\,P(B)}{P(a)} = \frac{P(a|B)\,P(B)}{P(a|R)P(R) + P(a|B)P(B) + P(a|G)P(G)}$$

$$= \frac{0.5 \times 0.4}{0.3 \times 0.2 + 0.5 \times 0.4 + 0.6 \times 0.4}$$

$$= \frac{0.2}{0.5}$$

$$= 0.4 \quad ※$$

2.

(2) $P(x,C_1) \geq P(x,C_2) \quad | \quad x \in R_1$

$P(x,C_2) \geq P(x,C_1) \quad | \quad x \in R_2$

(1) prove.

if $a \leq b$, then $a \leq (ab)^{\frac{1}{2}}$

$(ab)^{\frac{1}{2}} \geq (a \times a)^{\frac{1}{2}} = a$

$\therefore \sqrt{ab} \geq a$

$$\int \{P(x,C_1)\,P(x,C_2)\}^{\frac{1}{2}}\,dx =$$

$$= \int_{R_1} \{P(x,C_1)\,P(x,C_2)\}^{\frac{1}{2}}\,dx + \int_{R_2} \{P(x,C_1)\,P(x,C_2)\}^{\frac{1}{2}}\,dx \geq \underbrace{\int_{R_1} P(x,C_2)\,dx + \int_{R_2} P(x,C_1)}_{=\,P(\text{mistake})}$$

※

3. (1) prove $E(x) = E_y[E_x[x|y]]$

$$= E_y\Big[\sum_x x \cdot P(X=x \mid Y=y)\Big]$$

$$= \sum_y \sum_x x \cdot P(X=x \mid Y=y)\,P(Y=y)$$

$$= \sum_y \sum_x x \cdot P(Y=y \mid X=x)\,P(X=x)$$

$$= \sum_x x \cdot P(X=x) \sum_y P(Y=y \mid X=x)$$

$$= \sum_x x\,P(X=x)$$

$$= E[x]$$

3, (2). prove $Var[x] = E_y[Var_x[x|y]] + Var_y[E_x[x|y]]$

代入. $Var(x) = E[X^2] - [E[X]]^2$.

$E_y[Var_x[x|y]] + Var_y(E_x[x|y]]$

$= E_y[E_x[x^2|y] - (E_x[x|y])^2] + E_y[(E_x[x|y])^2] - (E_y[E_x[x|y]])^2$.

$= E_x[x^2] - E_y[[E_x[x|y]]^2] + E_y[(E_x[x|y])^2] - (E_x[x])^2$

$= E_x[x^2] - (E_x[x])^2$

$= Var_x[x]$

# ● Part. 3, Model Architecture and Method

## 1. Flow chart

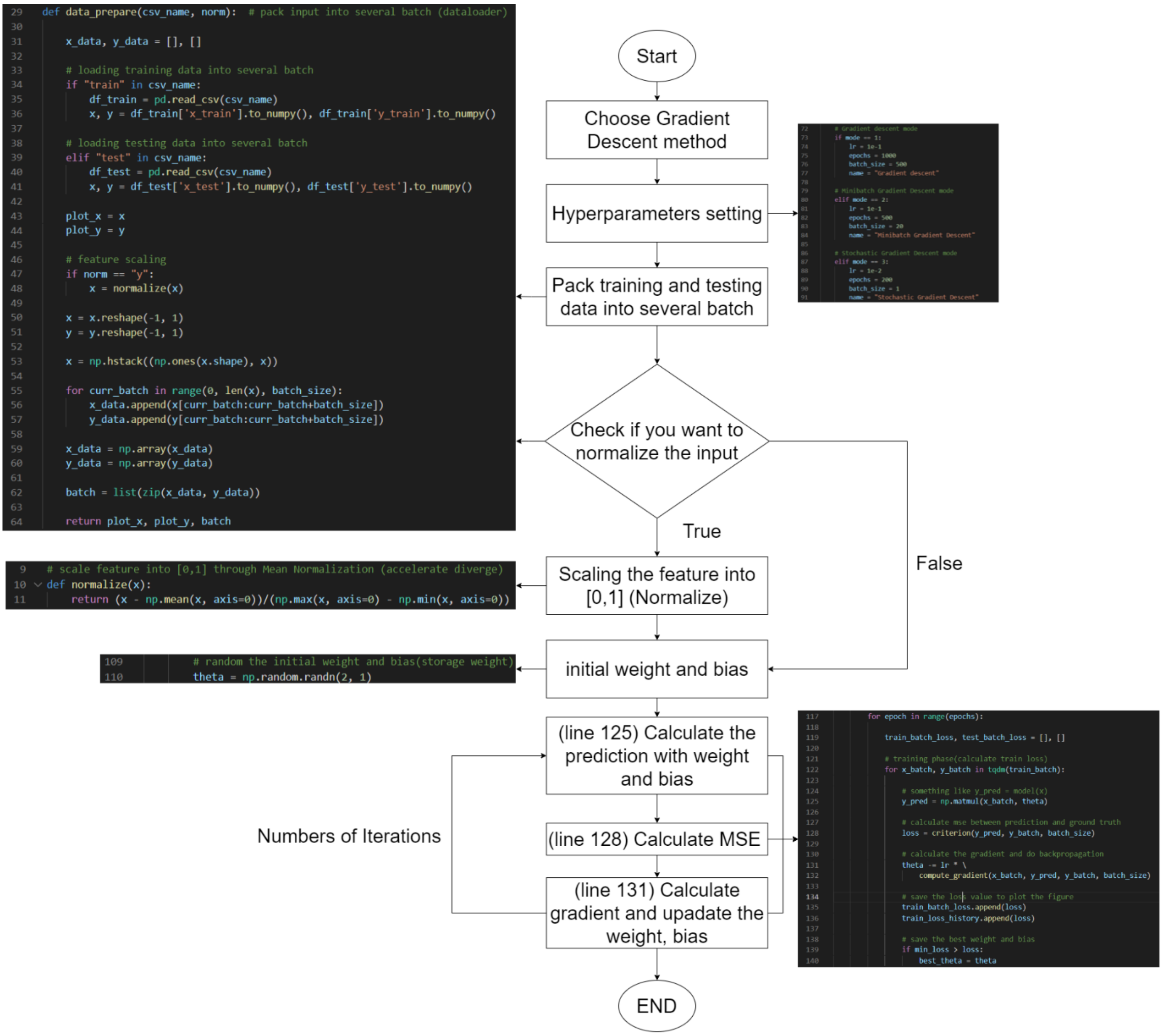



Figure 23. Programming flow chart

## 2. Normaliztion(feature scaling)

Normalization is to subtract the average of all x-pointoordinates from each x - point coordinate and then divide it by the interval between the maximum and minimum values. If use this method, the model will converge more quickly.

```python
# scale feature into [0,1] through Mean Normalization (accelerate diverge)
def normalize(x):
    return (x - np.mean(x, axis=0))/(np.max(x, axis=0) - np.min(x, axis=0))
```

Figure 24. Normalization method