Gradient Descent

Luca Brodo – Summer Semester 2021

Deep Learning

Gradient Descent

- Algorithm for optimization of Machine Learning models

- Performance plays a huge role when it comes to training models
- By knowing how it works, ad – hoc solutions can be implemented

Gradient Descent

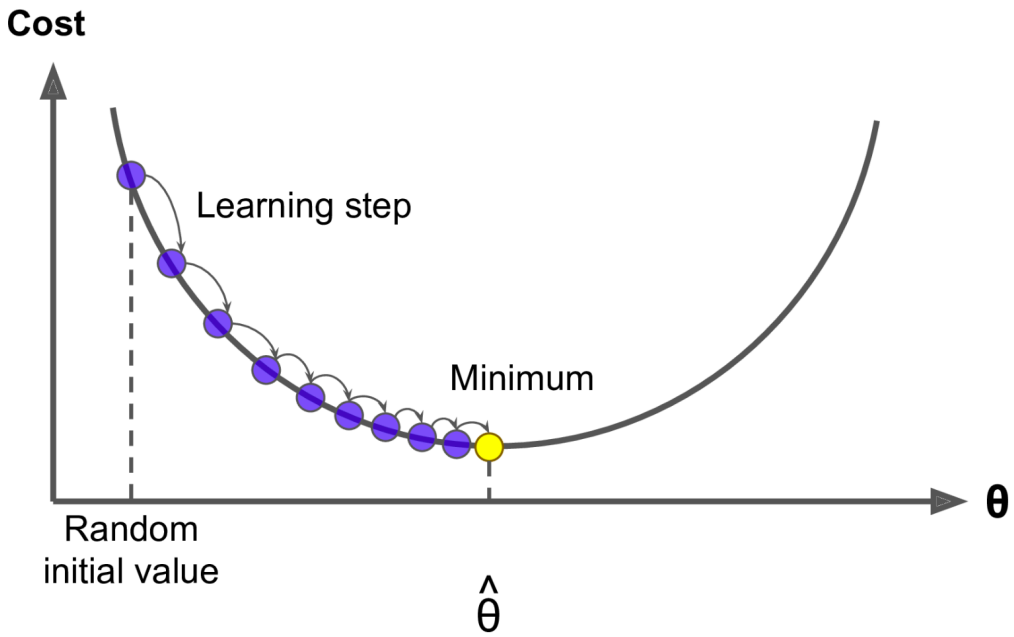
- This algorithm is usually attributed to Cauchy, who first suggested it in 1847

- Basic idea:
  - Finding local minimum through an iterative process
  - The step opposite direction of the gradient of the loss function is taken

$$\mathbf{a_n} = \mathbf{a_{n-1}} - \alpha \nabla F(\mathbf{a_{n-1}})$$

Gradient Descent

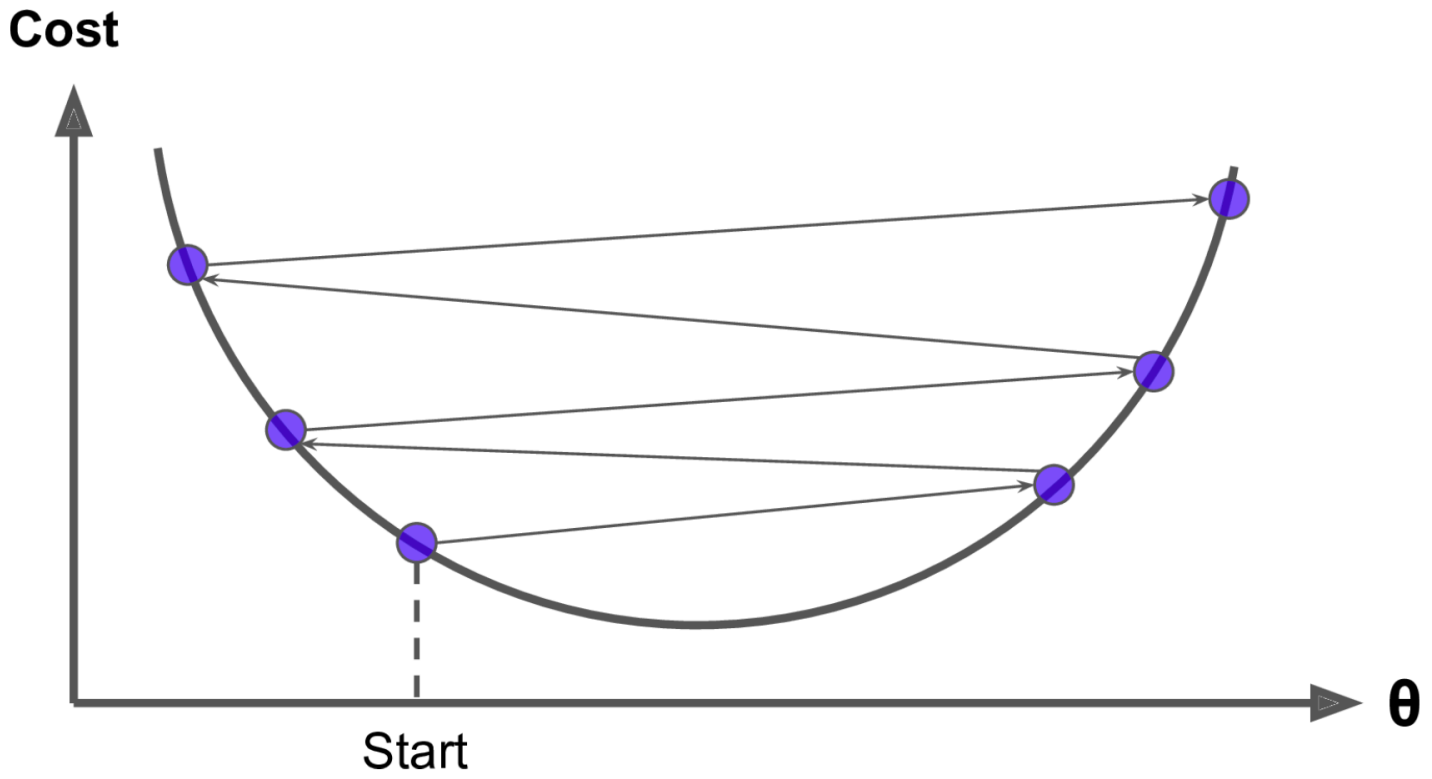
- Calculate the minimum of the cost function
- Steps = Learning Rate



$$\mathbf{a_n} = \mathbf{a_{n-1}} - \alpha \nabla F(\mathbf{a_{n-1}}) \quad (1)$$

Incorrect Value for the learning rate

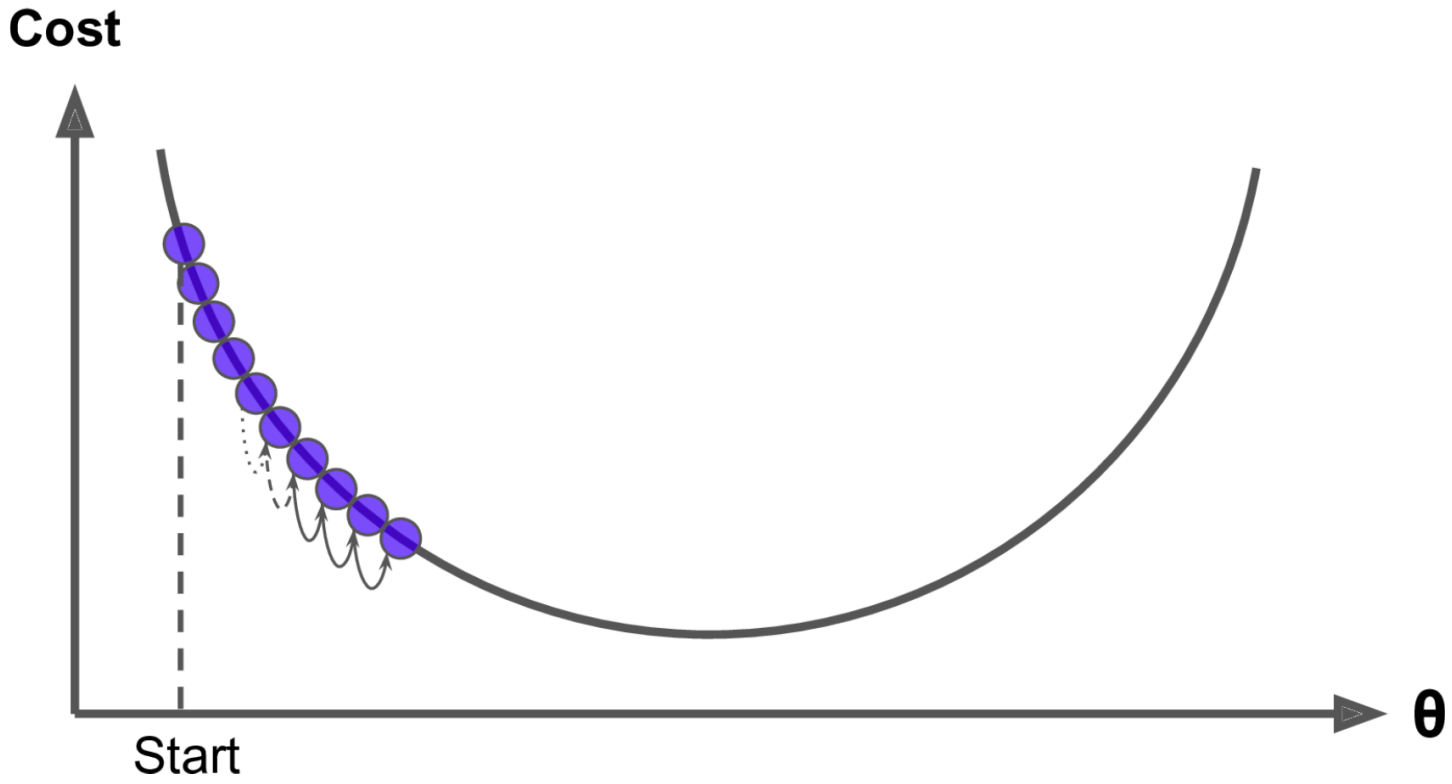
- Overshooting:



(1)

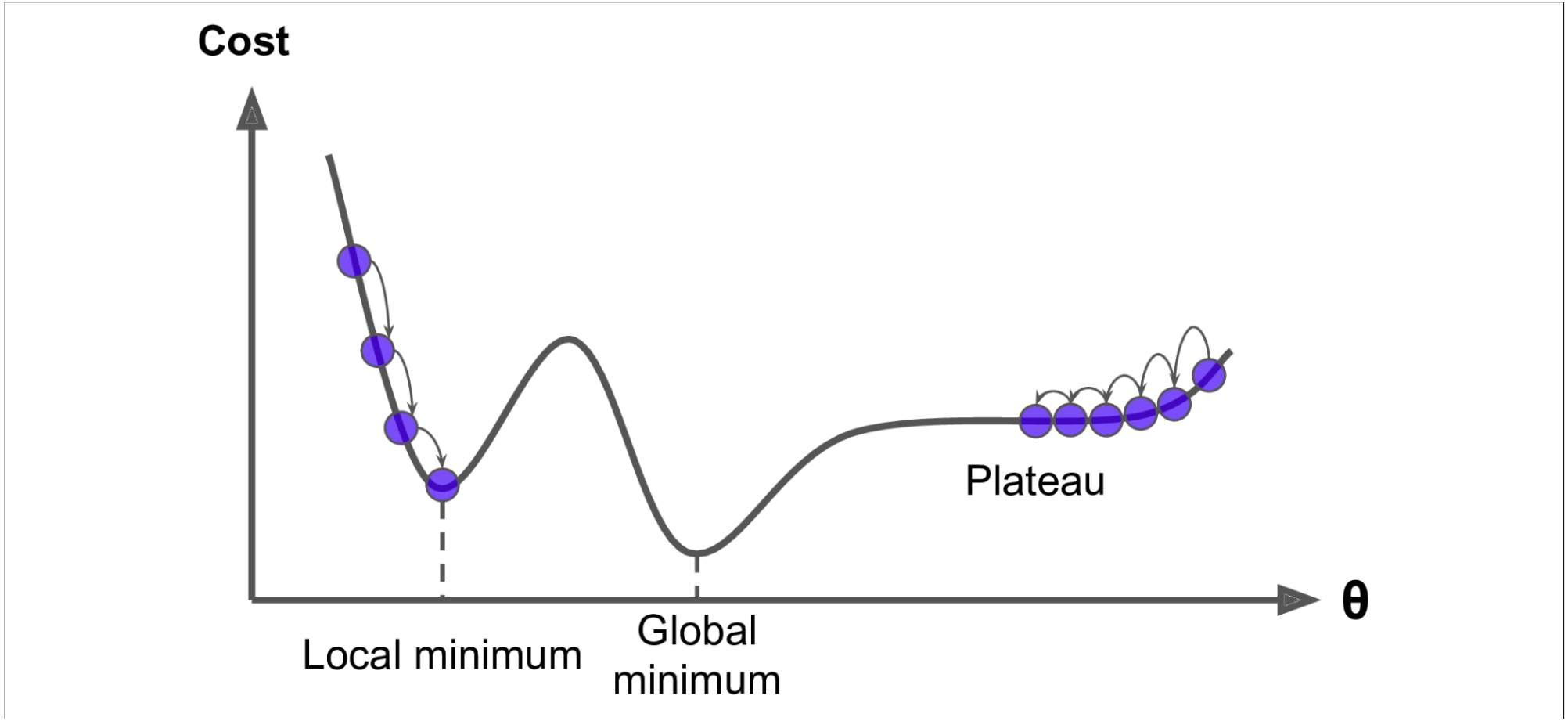
Incorrect Value for the learning rate

- Undershooting



(1)

Gradient Descent Pitfalls



(1)

3 Variations:

- Batch Gradient Descent
- Stochastic Gradient Descent
- Mini-Batch Gradient Descent

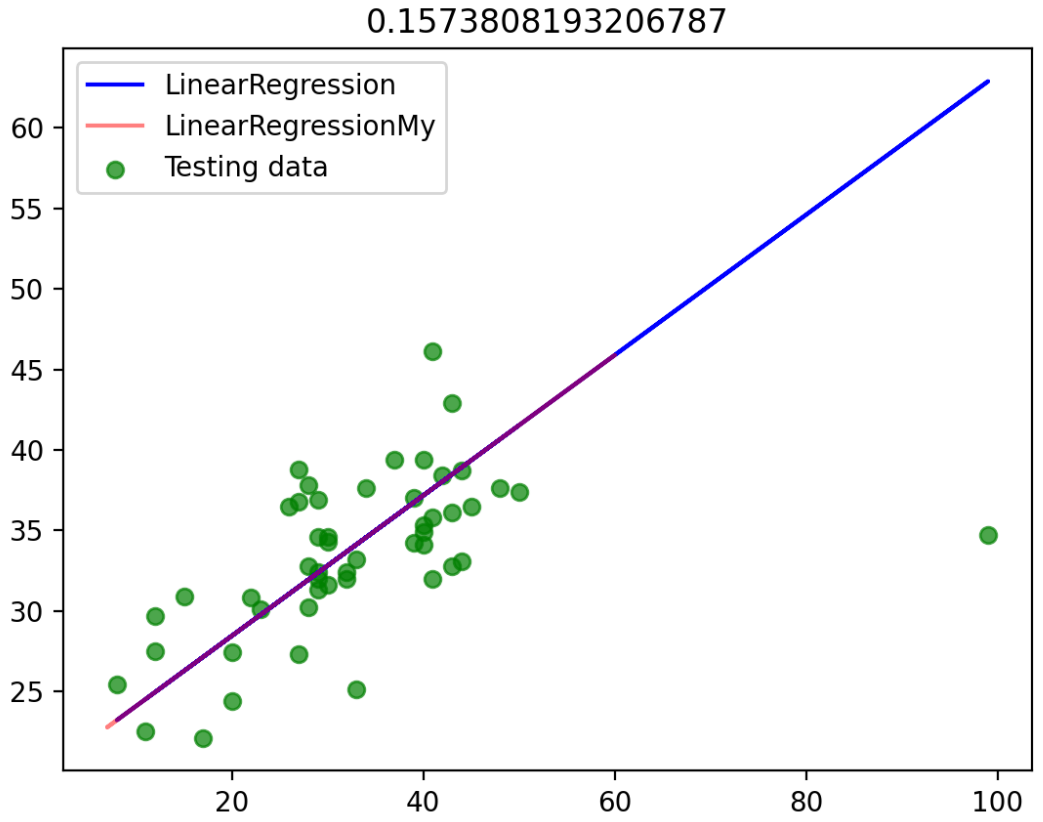
Batch Gradient Descent

$$\mathbf{a_n} = \mathbf{a_{n-1}} - \alpha \nabla F(\mathbf{a_{n-1}})$$

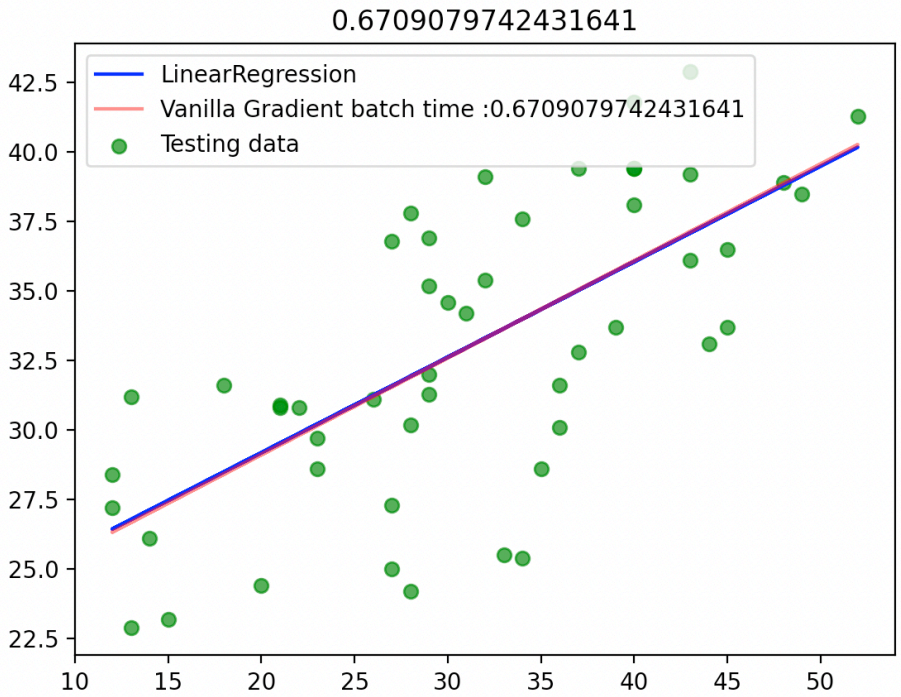
$$\theta^{(i)} = \theta^{(i-1)} - \alpha \nabla \mathscr{L}(\theta; \tau)$$

- Optimizes the parameters using the whole data set at every iteration
- 0(1/$\mathcal{E}$) iterations to reach the optimum => $\mathcal{E}$ depending on the shape of the cost function

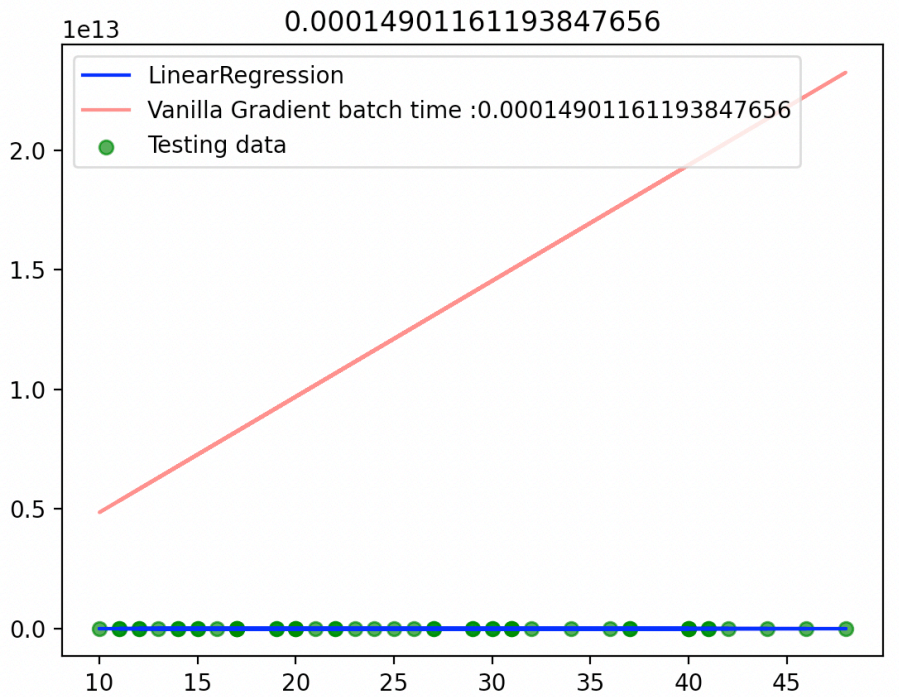
Result


0.1573808193206787
LinearRegression
LinearRegressionMy
Testing data
60
55
50
45
40
35
30
25
20
40
60
80
100

Smaller learning rate



Bigger learning rate



Characteristics

- The optimal minimum is guaranteed to be found
- Very solid
- Works with all type of loss functions

But:

- Slow
- Not suitable for large datasets

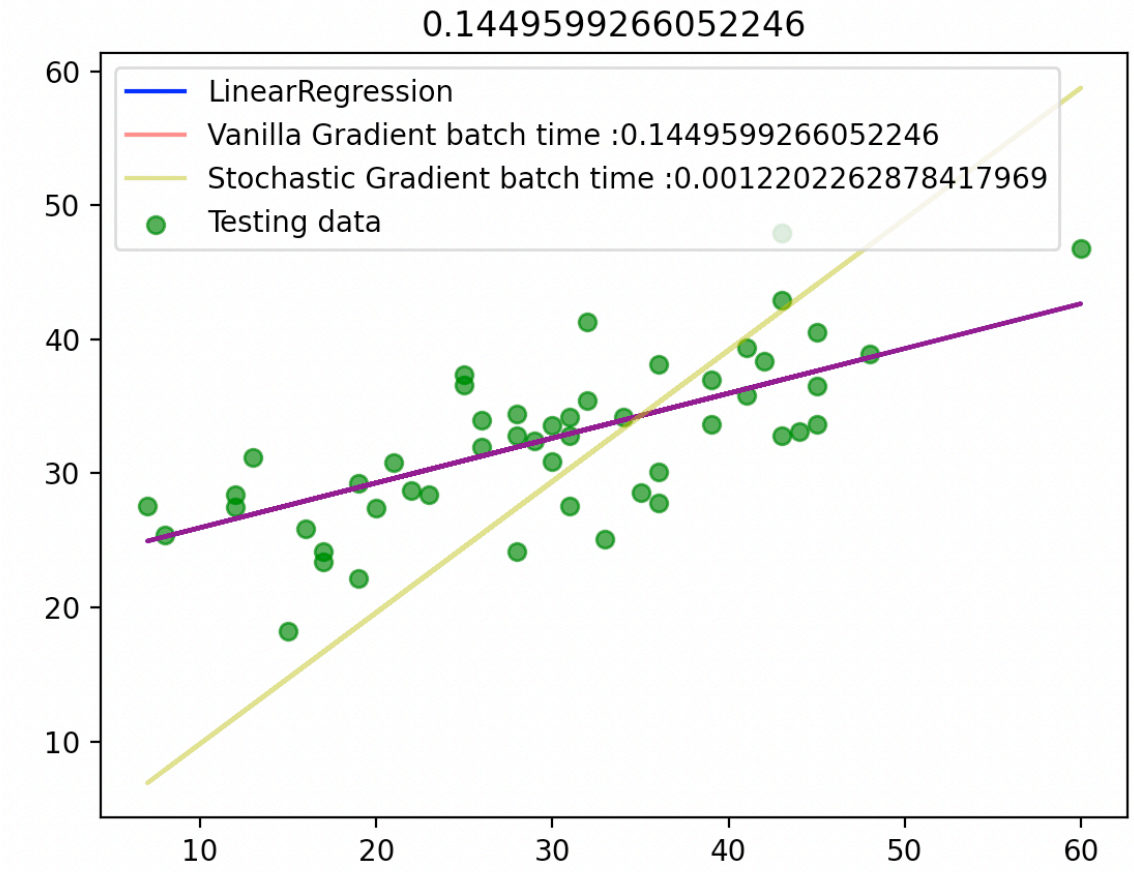
Stochastic Gradient Descent

$$\mathbf{a_n} = \mathbf{a_{n-1}} - \alpha \nabla F(\mathbf{a_{n-1}})$$

$$\theta^{(i)} = \theta^{(i-1)} - \alpha \nabla \mathscr{L}(\theta; (x_i, y_i))$$

- Batch version too slow
  - Using the entire dataset introduces unwanted redundancy
- Solution →
  - Use only one instance of the dataset
  - At every iteration, select one instance randomly

Result


0.1449599266052246
LinearRegression
Vanilla Gradient batch time :0.1449599266052246
Stochastic Gradient batch time :0.0012202262878417969
Testing data
60
50
40
30
20
10
10
20
30
40
50
60

Drawbacks

- Randomness → much less solid
  - Possibility to jump out of the local minimum
- Depends on the type of cost function

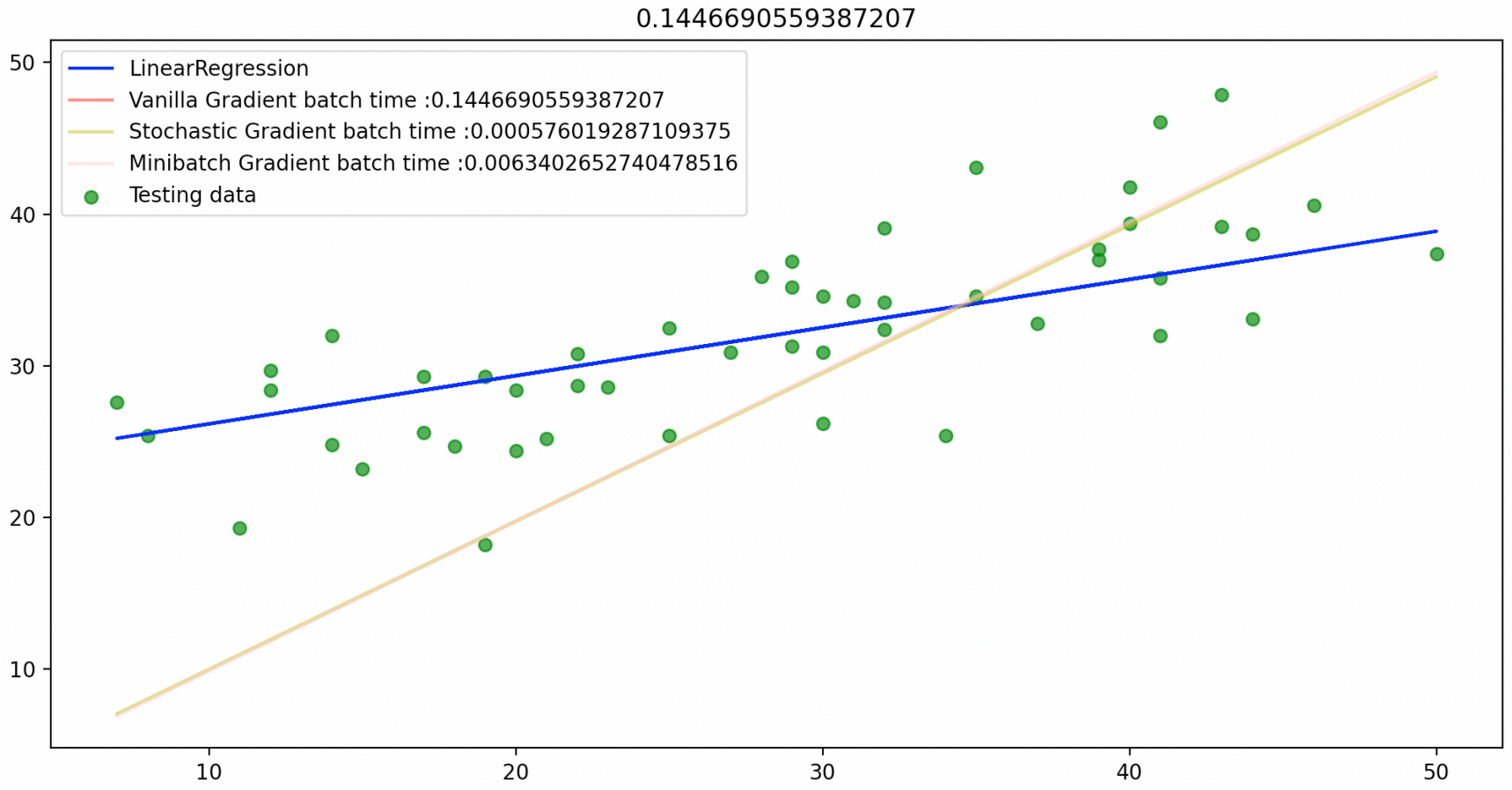
- One possible solution
  - Reduce the learning grade gradually at every iteration

Mini-Batch Gradient Descent

$$\theta^{(i)} = \theta^{(i-1)} - \alpha \nabla \mathscr{L}(\theta; \beta)$$

- Select batch from the dataset
- Use this batch for the optimization

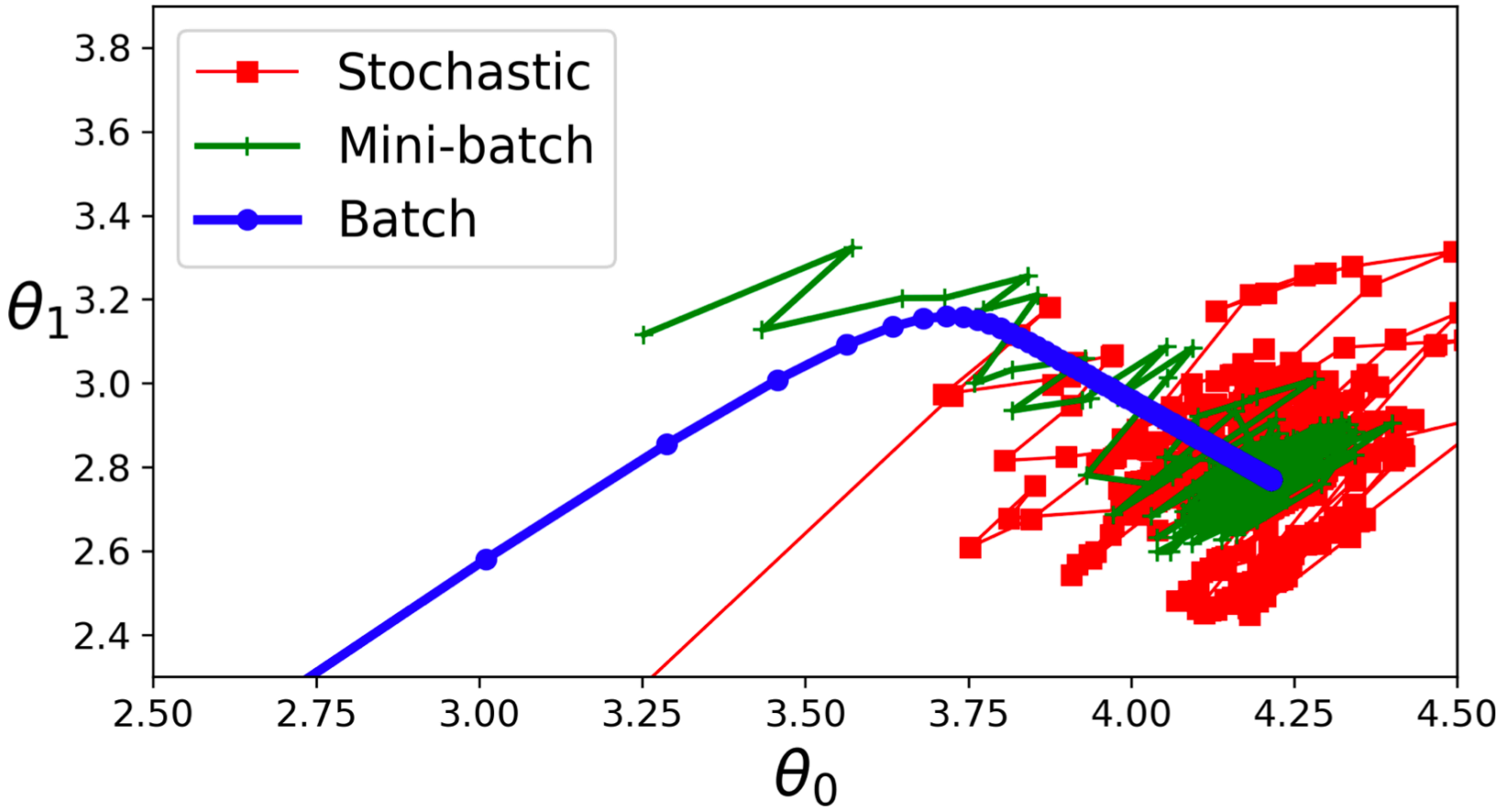
Result


0.1446690559387207
LinearRegression
Vanilla Gradient batch time :0.1446690559387207
Stochastic Gradient batch time :0.000576019287109375
Minibatch Gradient batch time :0.0063402652740478516
Testing data
50
40
30
20
10
10
20
30
40
50

Characteristics

- Randomness still plays a major role
- Faster than the batch version, more stable than the stochastic
- The form of the loss function has an impact on the performance
- Suitable for large dataset

Comparison



(1)

Comparison and conclusion

- Very solid optimization algorithm
- Huge room for improvements for ad-hoc solutions
- Not a work-for-all solution
- Important trade off to consider:
  - Faster solutions but higher possibility to overshoot