Training models: Gradient Descent

Jens Peter Andersen, Assistant Professor, Roskilde
Michael Claudius, Associate Professor, Roskilde, with respect and gratefullness to Jens Peter Andersen

26-09-2021

Training models: Opening The Black Box

- **It is about understanding what goes on behind the stage when models are trained and fitted to labels!**

1. **Linear regression refreshed**
2. **Define the cost function**
3. **Minimize the cost function, two types**
   - **Closed Form Solution (Normal Equation)**
   - **Gradient Descent Solutions: Batch GD, Stochastic GD (SGD), Mini-Batch GD**
4. **Regularize Linear Models to avoid vulnerabilities**
   - **Overfitting, Underfitting, Learning Curves, Early Stopping**

Closed Form: Evaluation

Typical complexity for closed form computations:

- Training the model with *m* feature instances is complexity *O(m)* – proportional to number of instances m
- Training the model with *n* features is complexity $O(n^{(>2)})$ – proportional to quadratic number of features $n^{(2)}$

  - – e.g. increasing *n* by a factor 2 will increase processing resources needed by $2^2=4$
  - – e.g. increasing *n* by a factor 10 will increase processing resources needed by $10^2=100$! That's 100 times slower

  - Processing resources are time and memory

  - Advantage: Simple to compute
  - Disadvantage: Slow for high number of features (n > 10.000)
    - Examples: predicting on basis of the human genom

  - **We are lucky. Why? There is the Gradient Descent Solutions for these cases.**
  - **BUT let us look at some code first.**

Gradient Descent Solutions

- **We are lucky. There is the Gradient Descent Solution for these cases.**

- **Batch GD,**
  - **Use the whole training set to calculate gradients at each step**
  - **Advantage: simple**
  - **Disadvantage: Slow for a large training set**
- **Stochastic GD (SGD)**
  - **Choose randomly one instance each time and calculate the gradient based on this instance**
  - **Advantage: Fast, will come close to minimum**
  - **Disadvantage: irregular path and bounce around the minimum**
  - **Solution: Needs a good learning schedule**
- **Mini-Batch GD**
  - **Choose randomly a batch; i.e. a set of instances each time and calculate the gradient based on this instances**
  - **Advantage: Fast, will come close to minimum, more regular than SGD**
  - **Disadvantage: bounce around the minimum**
  - **Solution: Needs a good learning schedule**

- **BUT first we will look at the principles behind Gradient Descent**

Gradient descent – General learning approach

- **Applicable for various kinds of models, which includes:**
  - **Linear**
  - **Polynomial**
  - **Logistic regression (classification)**
- **Out of core (memory) learning is possible**
- **Slow when *m* – the number of instances increases**
- **Faster then normal equation, when *n* - the number of model paramters $\theta_1,\dots,\theta_n$ - i.e. number of features increase**

Gradient descent – Minimizing MSE

- Applied in order to <u>optimize</u> by <u>minimizing</u> a so-called cost function – typically MSE in machine learning.
- Here the Mean Square Error is a function of the model parameters – that is $MSE(\theta_1, \ldots, \theta_n)$
- 'Gradient' refers to observing on the slope of the cost function – negative, zero or positive.
- We want to end up in a set of values for $\theta_1, \ldots, \theta_n$ where the slope of the cost function is zero – that means minimum reached.
- 'Descent' refers to that we are adjusting the values of $\theta_1, \ldots, \theta_n$ in the direction where the cost function diminishes

Linear model - Mean squared error (MSE)

Problem: Finding the model parameters θ_0 and θ_1

$\hat{y}= \theta_0 + \theta_1 x_1$

Solution: Finding the model parameters θ_0 and θ_1 by minimizing MSE:

$MSE(\theta_1, \theta_0) = 1/N \ \Sigma_{i=1..N} \ (y_i - (\theta_1 x_i + \theta_0))^2$

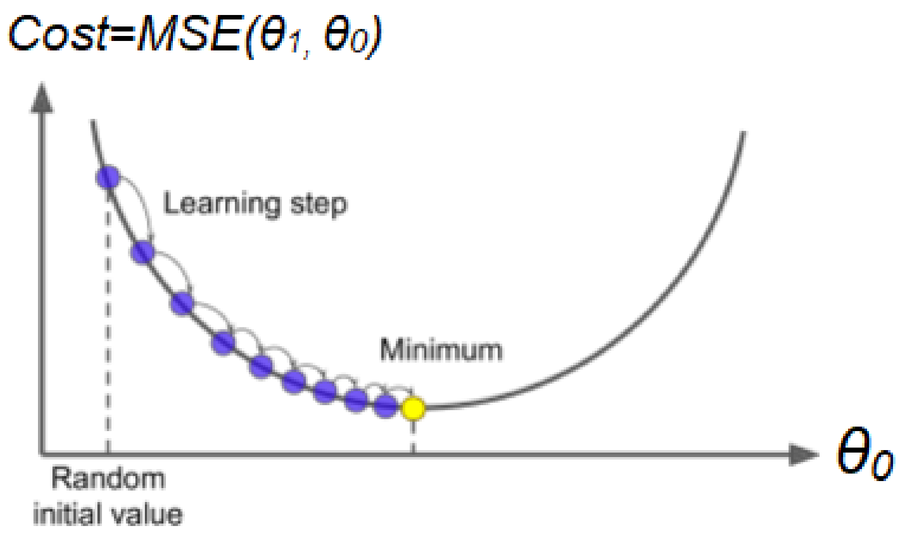
Linear model - Mean squared root error

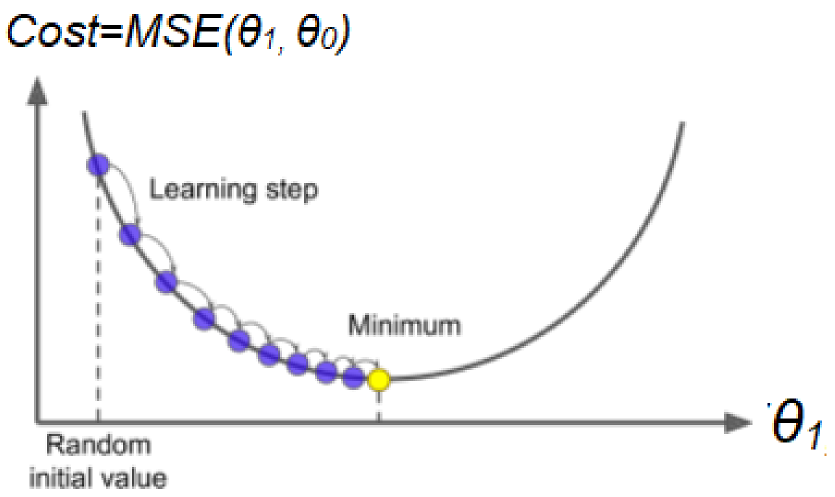
- Mathematically we want to calculate the so-called partially derivative with respect to all model parameters in order to approach the minimum for MSE.

- With 2 model parameters the partially derivatives are expressed like this :
  - $\partial MSE(\theta_{1,}\ \theta_0) / \partial\theta_1$ - Expresses slope in the direction of θ_1
  - $\partial MSE(\theta_{1,}\ \theta_0) / \partial\theta_0$ - Expresses slope in the direction of θ_0
- Don’t worry we will look at the curves soon

Performing Gradient Descent

Now lets watch a video on Gradient Descent: [Gradient Descent Statquest]
Afterwards explain to yourselves in the groups the process on how the cost function Mean Square Error (MSE) goes towards the minimum, use curves below for help.





Performing Gradient Descent – the principle

While (Minimum <u>not</u> reached)

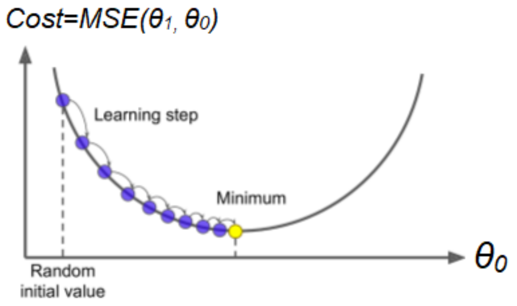
{

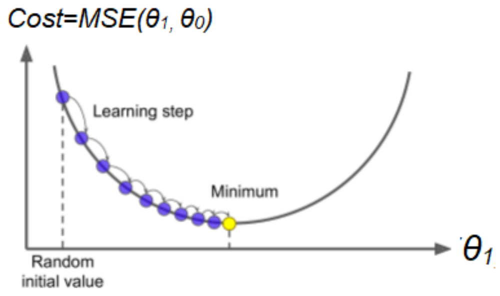
Based on the learning set:

$\theta_0 \ _= = \theta_0$ *- LearningRate ** $\partial MSE(\theta_{1,} \theta_0) / \partial\theta_0$

$\theta_1 \ _= = \theta_1$ *- LearningRate ** $\partial MSE(\theta_{1,} \theta_0) / \partial\theta_1$

}

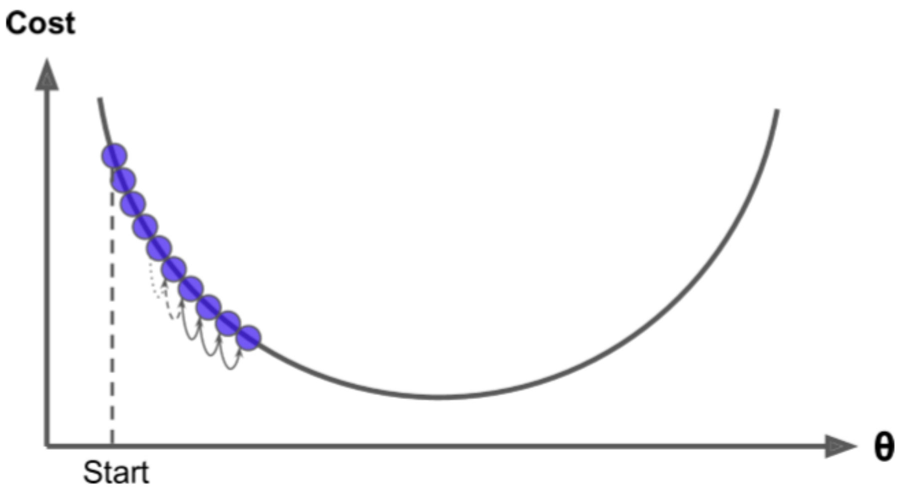




latest.webp

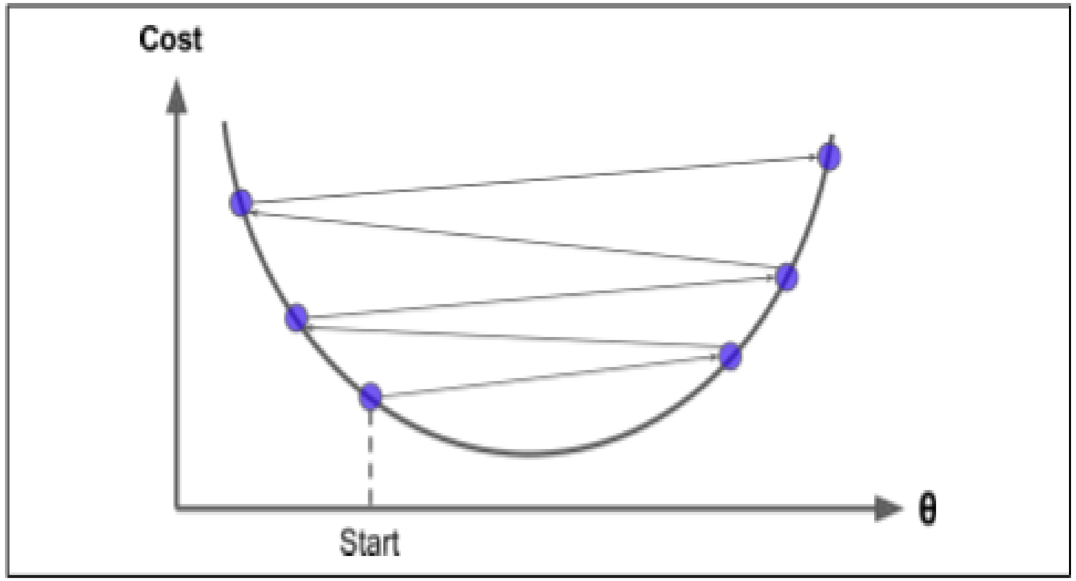
Being too ‘scrooge’ with the learning rate

- Learning rate is <u>too small</u> will make gradient descent <u>too slow</u>
- That is, the model parameters $\theta_{0\ \dots}\ \theta_n$ are changed in small steps slowing down the algorithm
- Eventually $MSE(\theta_{0,\ \dots,}\ \theta_n)$ will <u>converge</u> towards a minimum



Being too ‘greedy/lazy’ with the learning rate

- Learning rate is <u>too big</u> will make gradient descent <u>diverge</u> away from finding the minimum $MSE(\theta_{0, \ldots,}\ \theta_n)$
- That is, the model parameters $\theta_{0\ \ldots}\ \theta_n$ are changing in big steps that makes the learning algorithm get lost



Cost function: Challenges

- **IF the cost function is “not nice” not-convex function with irregular shape with holes, ridges, plateaus then**
- **GD finds a local minimum**
- **GD takes long time to pass a plateau**

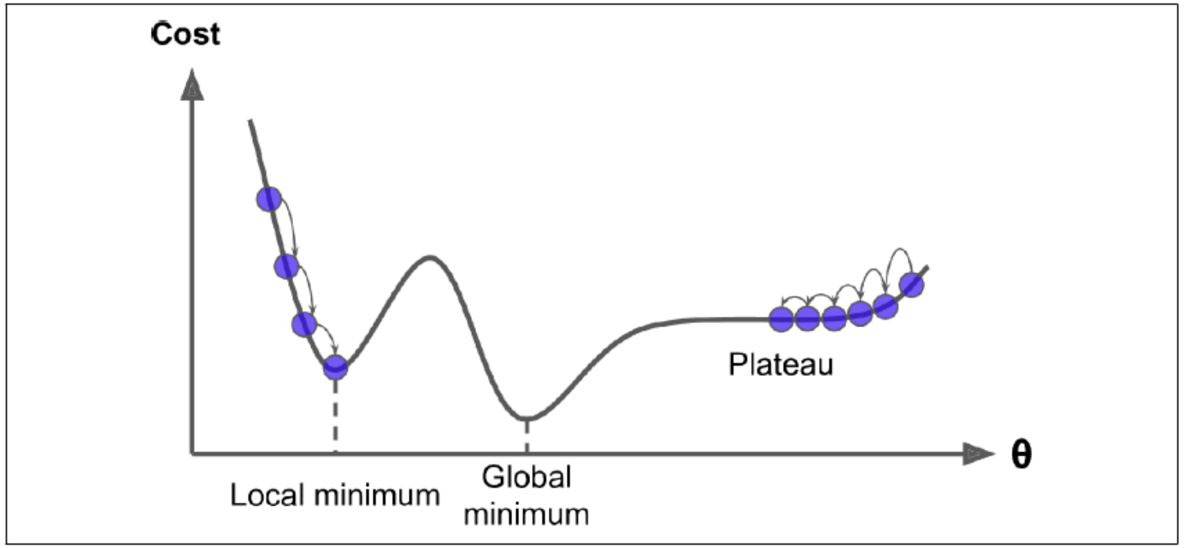
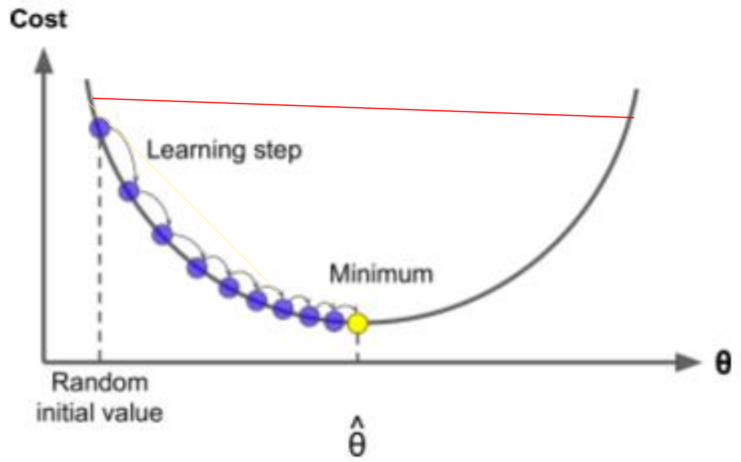


Figure 4-6. Gradient Descent pitfalls

- **BUT we are lucky again. Why? The MSE cost function is a convex function !**

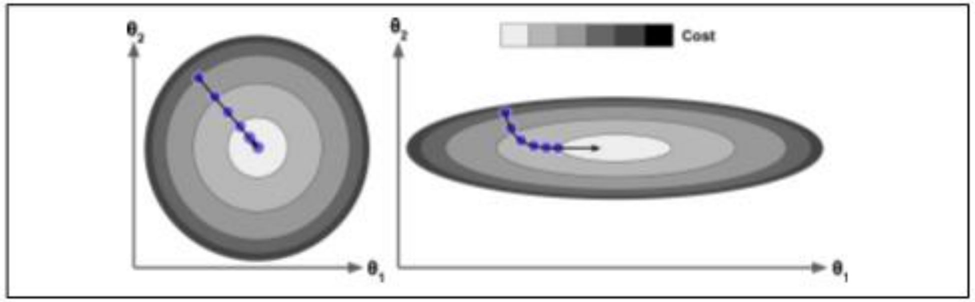
Nice linear regression properties MSE cost function

- The MSE cost function for a Linear Regression model a so-called convex function
- Convex: Red line segment will never cross the curve below
- This means that is has only one global minimum.
- It is a continuous function with a slope that never changes abruptly
- Gradient Descent is then guaranteed to approach arbitrarily close the global minimum.



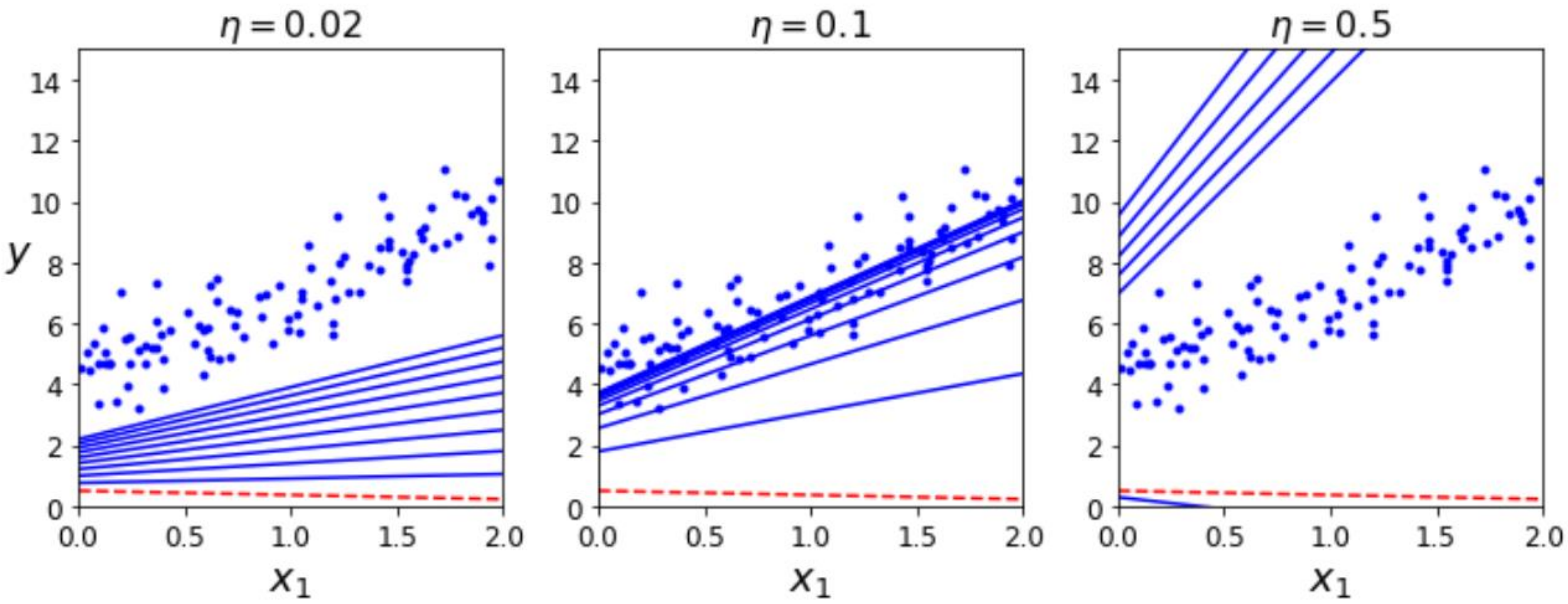
Gradient Descent – Feature scaling needed

- Feature scaling: Features (learning input) have same order of magnitude – e.g. in the the area -1 to 1
- To the left: Feature scaling applied -> Minimum of cost function approached faster
- To the right: Feature scaling not applied -> Minimum of cost function approached slower
- Feature scaling can be obtained by the Scikit-Learn's StandardScaler
- Feature scaling is recommended for gradient descent algorithms



Gradient Descent: the learning rate - example

- To the <u>left</u>: ‘Scrooge’ learning rate $\eta = 0.02$ – too small approaching MSE optimum slowly
- In the <u>middle</u>: ‘Appropriate’ learning rate $\eta = 0.1$ – approaches MSE optimum in reasonable time
- To the right: ‘Lazy’ rate $\eta = 0.5$ – too big missing MSE optimum



Stochastic Gradient Descent: The Principle

- Instead of processing the entire training set, we randomly pick one training set instance at a time
- Faster than the Batch Gradient Descent
- But more erractic

While (Minimum <u>not</u> reached)

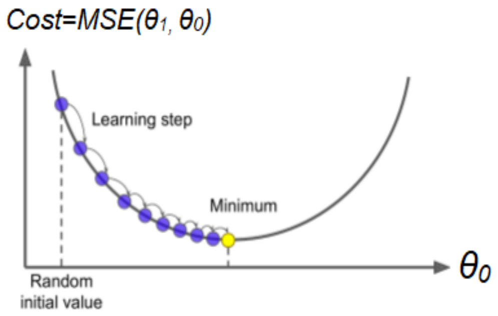
{

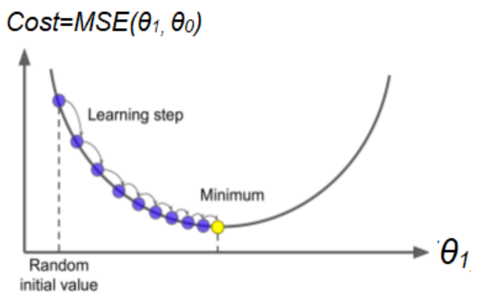
Based on picking one element at a time in the learning set randomly:

$\theta_0 \;_= = \theta_0$ - LearningRate * $\partial MSE(\theta_1, \theta_0) / \partial\theta_0$

$\theta_1 \;_= = \theta_1$ - LearningRate * $\partial MSE(\theta_1, \theta_0) / \partial\theta_1$

}





Mini-Batch Gradient Descent: The principle

- Instead of processing the entire training set, we pick a batch training set instance at a time
- Trade of between batch gradient descent and stochastic gradient descent
- Fast and less erractic

While (Minimum <u>not</u> reached)

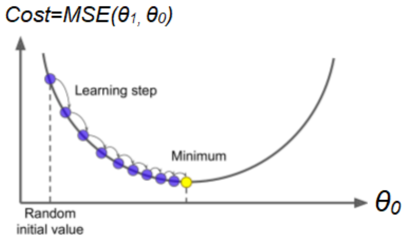
{

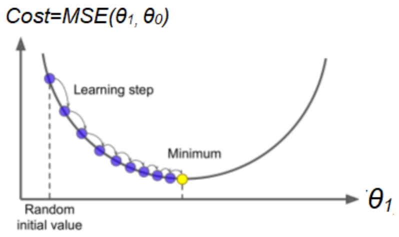
Based on picking batch of elements at a time in the learning set randomly:

$\theta_0 \;_= = \theta_0$ - LearningRate * $\partial MSE(\theta_{1,}\ \theta_0) / \partial\theta_0$

$\theta_1 \;_= = \theta_1$ - LearningRate * $\partial MSE(\theta_{1,}\ \theta_0) / \partial\theta_1$

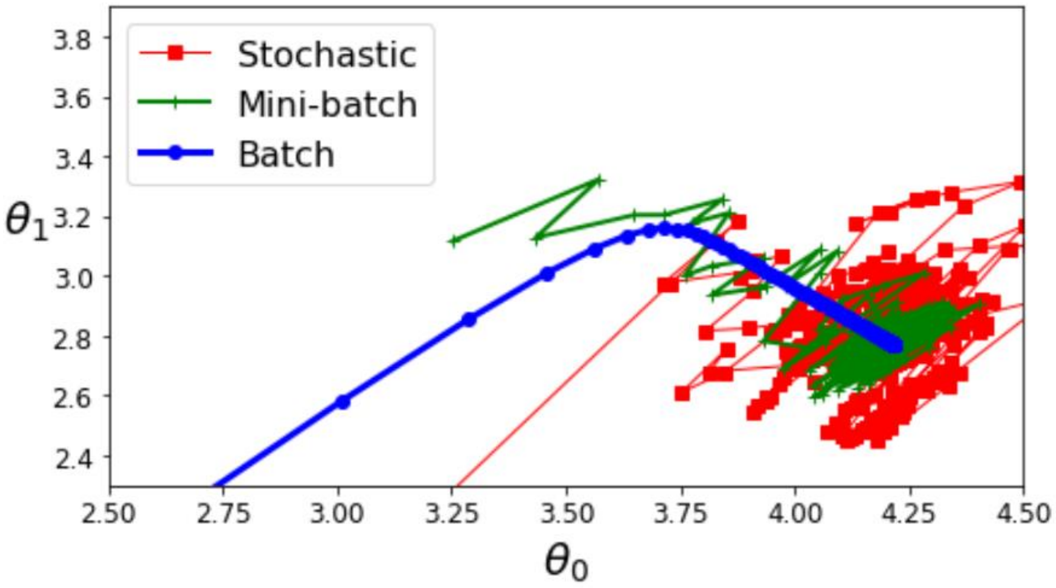
}





Comparing gradient descent approaches

Different paths in development in model parameters



Gradient Descent Evaluation

- We are lucky. There is the Gradient Descent Solution for these cases.

- Batch GD,
  - Use the whole training set to calculate gradients at each step
  - Advantage: Simple
  - Disadvantage: Slow for a large training set
- Stochastic GD (SGD)
  - Choose randomly one instance each time and calculate the gradient based on this instance
  - Advantage: Fast, will come close to minimum
  - Disadvantage: irregular path and bounce around the minimum
  - Solution: Needs a good learning schedule
- Mini-Batch GD
  - Choose randomly a batch; i.e. a set of instances each time and calculate the gradient based on this set of instances
  - Advantage: Fast, will come close to minimum, more regular than SGD
  - Disadvantage: bounce around the minimum
  - Solution: Needs a good learning schedule

- So lets compare the different solutions

Some comparison on linear regression algorithms

Algorithm	Large *m*	Out-of-core support	Large *n*	Hyperparams	Scaling required	Scikit-Learn
Normal Equation	Fast	No	Slow	0	No	n/a
SVD	Fast	No	Slow	0	No	`LinearRegression`
Batch GD	Slow	No	Fast	2	Yes	`SGDRegressor`
Stochastic GD	Fast	Yes	Fast	≥2	Yes	`SGDRegressor`
Mini-batch GD	Fast	Yes	Fast	≥2	Yes	`SGDRegressor`

That’s all folks

- **Don’t let a little Gorilla Math scare you !**