

Geo Data Science

Linear Regression & Gradient Descent

Prof. Dr. Martin Kada

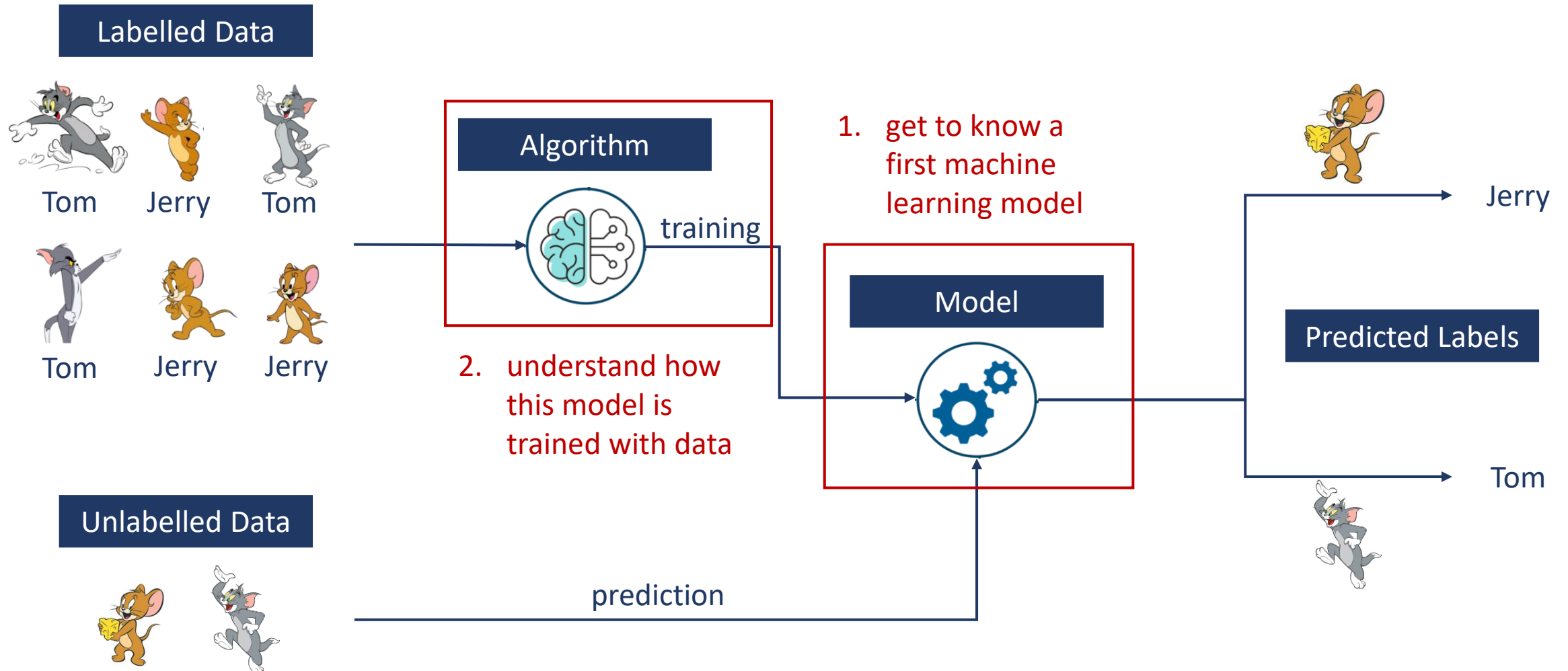
Chair Methods of Geoinformation Science (GIS)
Institute of Geodesy and Geoinformation Science

Copyright Notice

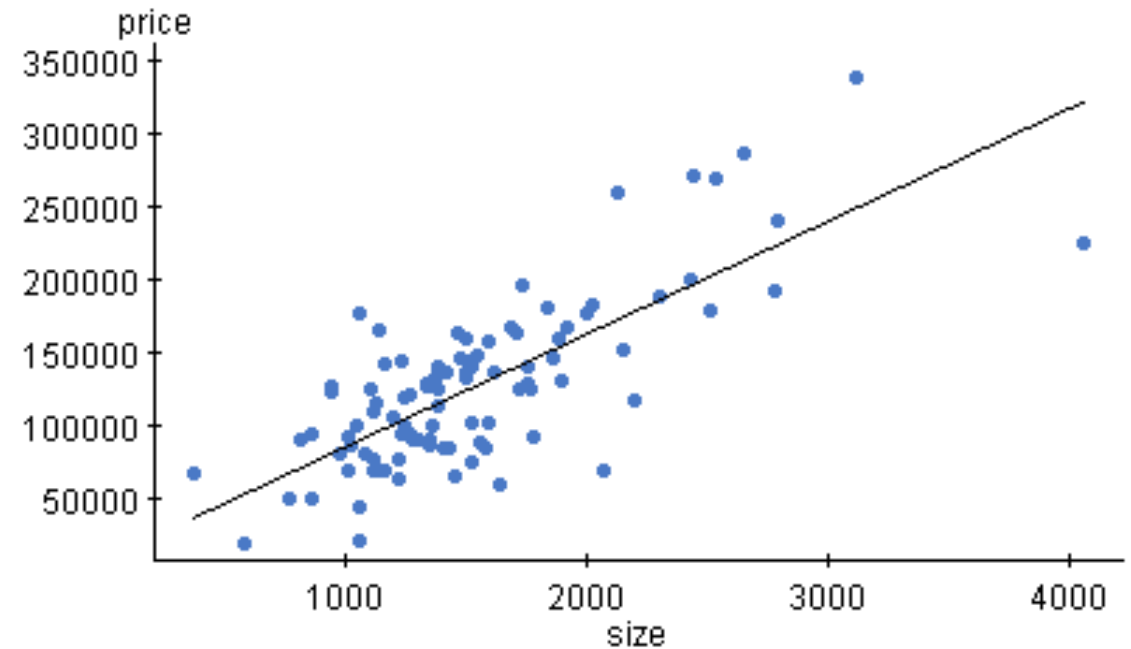
The teaching materials for this course and all elements contained therein are protected by international copyright laws. They may only be used for study purposes for the corresponding course.

Any reproduction and redistribution of the course materials without written permission is prohibited, other than the following: You may print or download them for your own personal use while attending the course.

Content of this Lecture



Content of this Lecture



- Learn a model by fitting a (straight) line through all (training) examples → **linear regression**
- Predict the outcome for an unseen dataset by substituting the input values into the model

Linear Regression

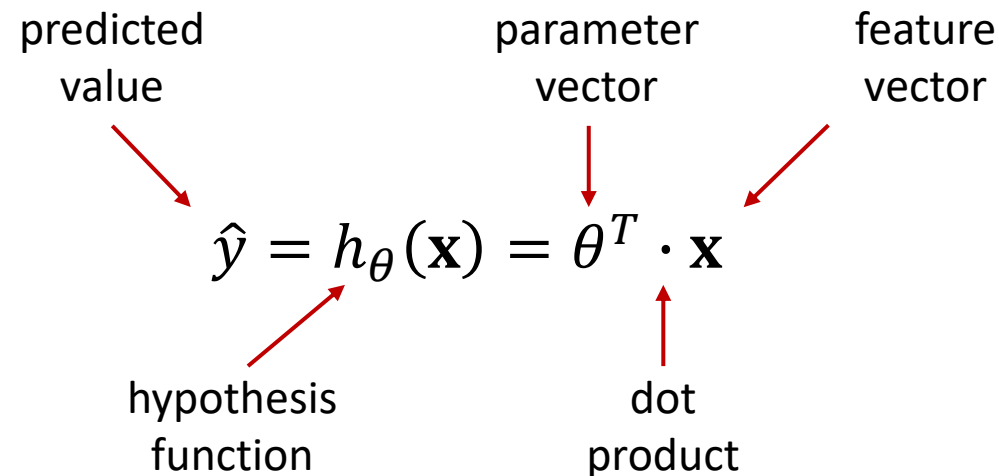
- Linear model that predicts a target value by computing a weighted sum of the input features plus a bias term (also called intercept term)

The diagram illustrates the linear regression equation $\hat{y} = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \dots + \theta_n x_n$ with the following annotations:

- predicted value**: Points to \hat{y} .
- model parameter**: Points to the coefficients $\theta_0, \theta_1, \theta_2, \dots, \theta_n$.
- bias term (intercept term)**: Points to θ_0 .
- feature values**: Points to the input features x_1, x_2, \dots, x_n .
- number of features**: Points to the index n .

Linear Regression

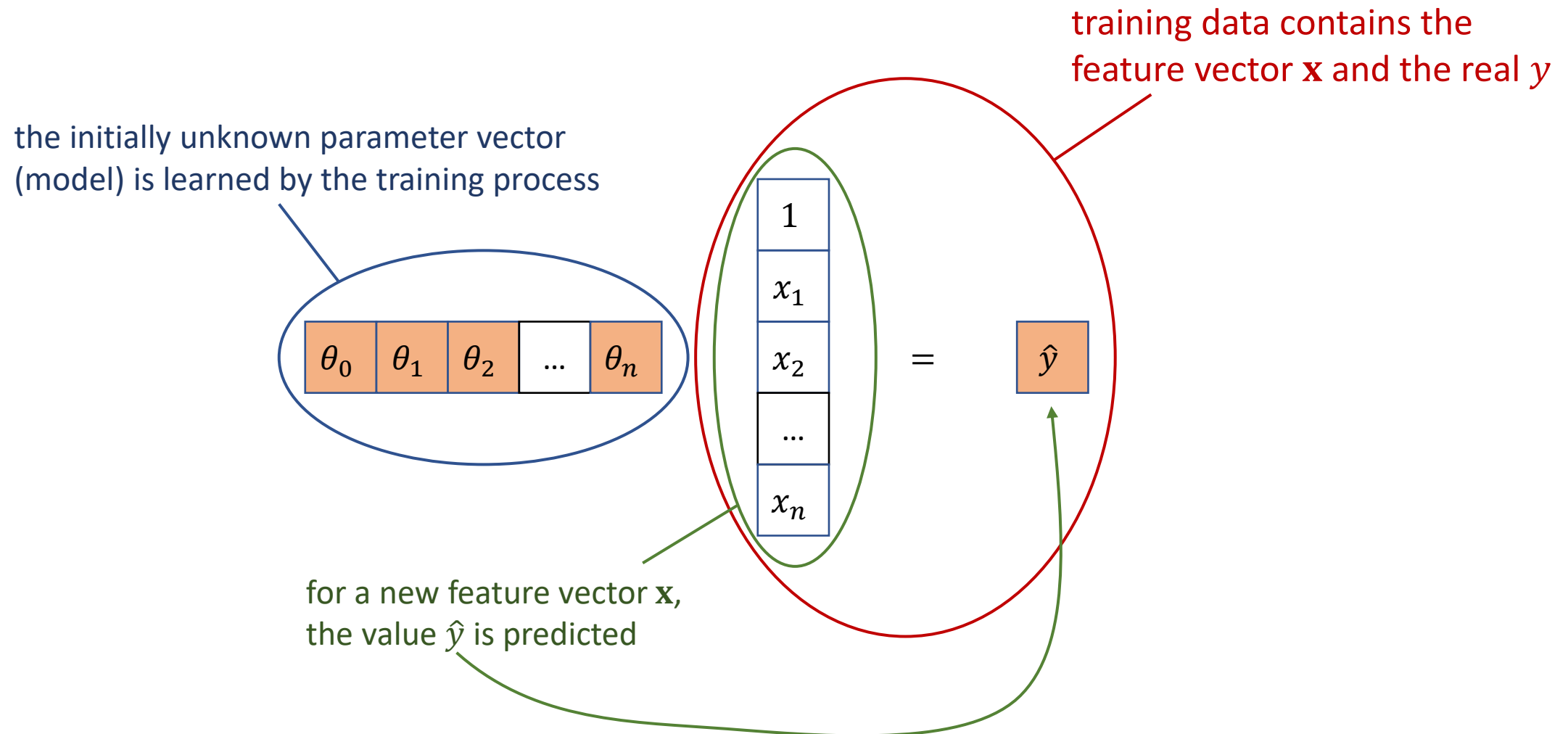
- In vectorized form:



The diagram illustrates the vectorized form of the linear regression equation: $\hat{y} = h_{\theta}(\mathbf{x}) = \theta^T \cdot \mathbf{x}$. Red arrows point from descriptive labels to the corresponding parts of the equation: 'predicted value' points to \hat{y} , 'parameter vector' points to θ^T , 'feature vector' points to \mathbf{x} , 'hypothesis function' points to h_{θ} , and 'dot product' points to the dot operator \cdot .

- Please note:
 - The parameter vector contains the bias term θ_0 and the feature vector contains x_0 (equal to 1)
 - Transpose of θ gives a row vector (instead of a column vector)

Linear Regression



Training of a Linear Regression Model

- Goal: setting the model parameter vector θ to best fit the training data \mathbf{X}
- Requires: **cost function** that measures how well the model fits \mathbf{X}
 - Most common is the Root Mean Square Error (RMSE)
 - **Mean Square Error (MSE)** is simpler to minimize and leads to the same results

$$\text{RMSE}(\mathbf{X}, h_{\theta}) = \sqrt{\frac{1}{m} \sum_{i=1}^m (\theta^T \cdot \mathbf{x}^{(i)} - y^{(i)})^2}$$

$\text{MSE}(\mathbf{X}, h_{\theta}) = \frac{1}{m} \sum_{i=1}^m (\theta^T \cdot \mathbf{x}^{(i)} - y^{(i)})^2$

$\overset{\text{training data of size } m}{\nearrow}$ $\overset{\text{target values}}{\nwarrow}$ $\overset{i^{\text{th}} \text{ training dataset}}{\nearrow}$

Mean Square Error (MSE) Cost Function

because the **square root** is monotonously increasing, larger errors stay larger than smaller errors → by leaving out the square root, the cost function has still the same effect

squared error, so that large errors influence the cost function more than small errors → prefer models that always make small errors, than models that often give perfect predictions, but sometimes make big errors

$$\text{MSE}(\mathbf{X}, h_{\theta}) = \frac{1}{m} \sum_{i=1}^m (\hat{y}^{(i)} - y^{(i)})^2$$

we can substitute $\hat{y}^{(i)}$, because of $\hat{y}^{(i)} = \theta^T \cdot \mathbf{x}^{(i)}$
(more general with respect to other models)

Normal Equation

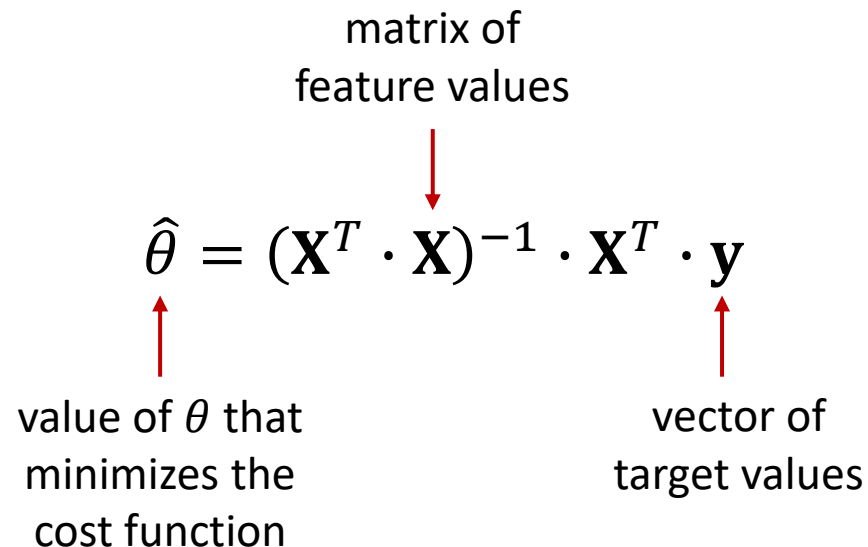
- Closed-form to directly compute the model parameters that best fit the model to the training data
 - $\hat{\theta}$ minimizes the cost function over the training data \mathbf{X}

$$\hat{\theta} = (\mathbf{X}^T \cdot \mathbf{X})^{-1} \cdot \mathbf{X}^T \cdot \mathbf{y}$$

matrix of
feature values

value of θ that
minimizes the
cost function

vector of
target values



Normal Equation

- Computing the inverse of $\mathbf{X}^T \cdot \mathbf{X}$, which is an $n \times n$ matrix, has a time complexity of about $O(n^{2.4})$ to $O(n^3)$
 - The computation of the normal equation is very time consuming when the number of features (n) gets large
- Linear with regard to the number of training data instances m
- Predictions from a trained model is linear with regard to both the number of prediction instances and the number of features
- Not for all machine learning models, a normal equation can be derived

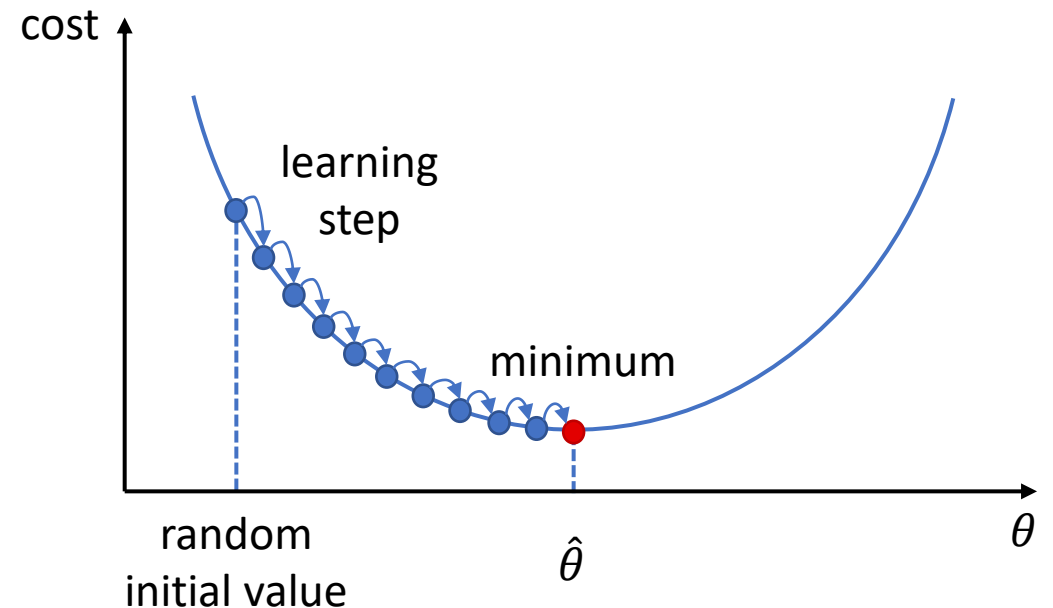
Gradient Descent

- Generic optimization algorithm for finding optimal solutions for a wide range of problems by iteratively tweaking the model parameters in order to minimize the cost function
- Idea:
 - Go downhill in the direction of the steepest slope until you reach a valley
 - Measure the **local gradient** of the cost function with regard to the parameter vector θ and go in the direction of descending gradient until a minimum is reached



Gradient Descent

- Algorithm:
 - Initialize θ with (small) random values (random initialization)
 - Gradually improve θ by decreasing the cost function
 - Stop when θ converges to a minimum



- In order to make a learning step, the gradient of the cost function with respect to each model parameter θ_j is computed
 - Determines how much the cost function changes if you change θ_j a little
- **Partial derivatives** of the cost function:

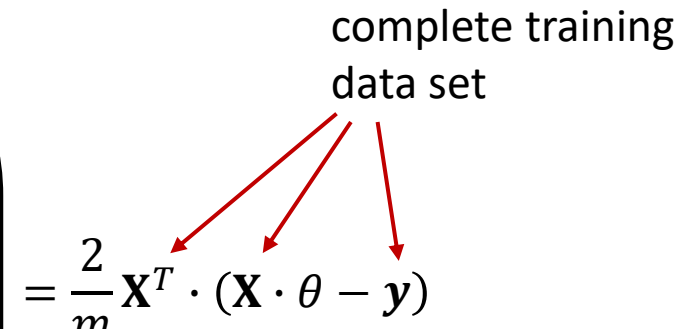
$$\frac{\partial}{\partial \theta_j} \text{MSE}(\theta) = \frac{2}{m} \sum_{i=1}^m (\theta^T \cdot \mathbf{x}^{(i)} - y^{(i)}) x_j^{(i)}$$

Batch Gradient Descent

- Gradient vector of the cost function:

$$\nabla_{\theta} \text{MSE}(\theta) = \begin{pmatrix} \frac{\partial}{\partial \theta_0} \text{MSE}(\theta) \\ \frac{\partial}{\partial \theta_1} \text{MSE}(\theta) \\ \vdots \\ \frac{\partial}{\partial \theta_n} \text{MSE}(\theta) \end{pmatrix} = \frac{2}{m} \mathbf{X}^T \cdot (\mathbf{X} \cdot \theta - \mathbf{y})$$

complete training
data set



- The gradient vector $\nabla_{\theta} \text{MSE}(\theta)$ contains all the partial derivatives of the cost function (one for each model parameter)
- Formula involves calculations over the full training data set $\mathbf{X} \rightarrow$ **Batch Gradient Descent**

Batch Gradient Descent

- Gradient Descent step:

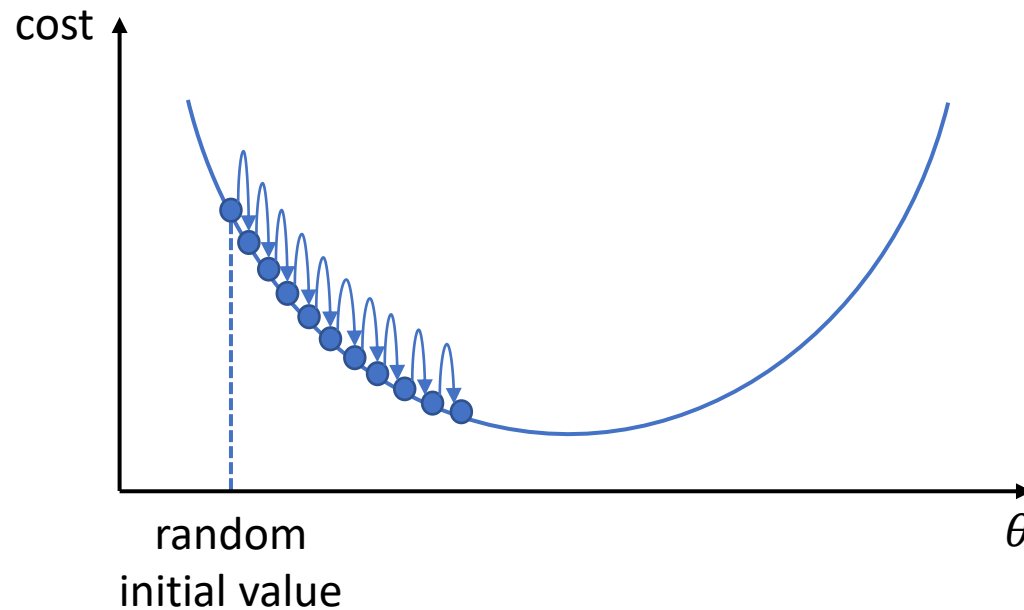
$$\theta^{(next\ step)} = \theta - \eta \nabla_{\theta} \text{MSE}(\theta)$$

learning rate

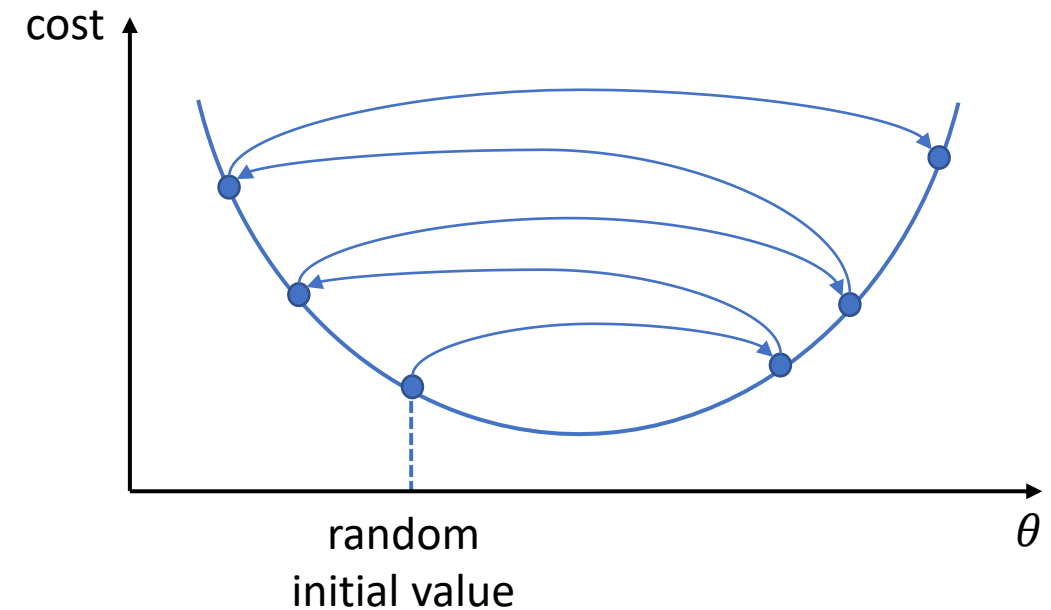
- As the gradient vector points uphill, the Gradient Descent step goes in the opposite direction

Gradient Descent

- The hyperparameter **learning rate η** determines how well the algorithm converges



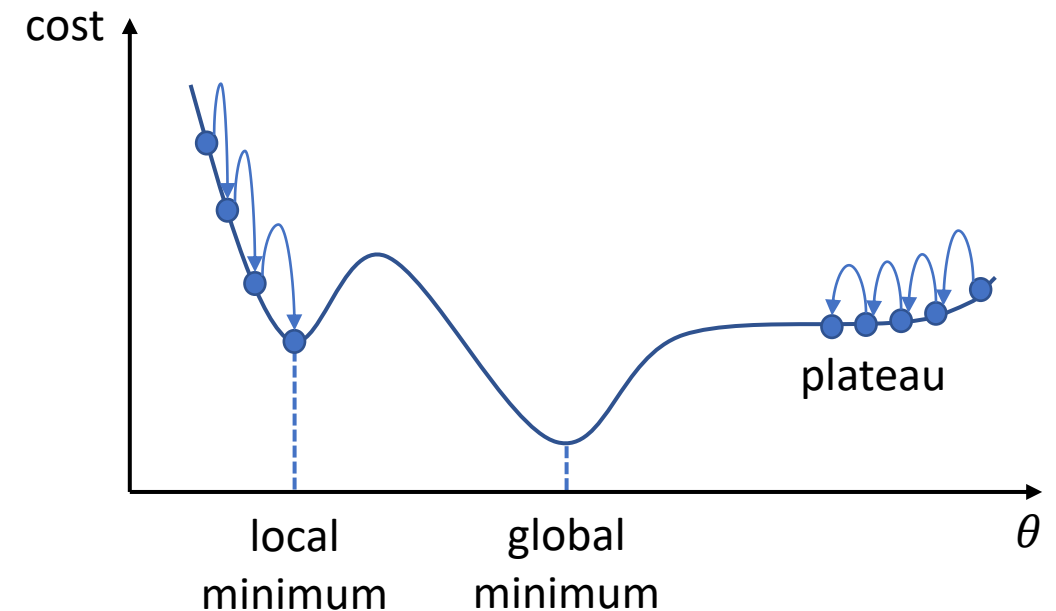
If the **learning rate is too small**, then the algorithm will take many iterations to converge



If the **learning rate is too high**, then the algorithm might overjump the minimum, possibly ending up even further away from the minimum than before → **algorithm diverges** and fails to find a good solution

Gradient Descent

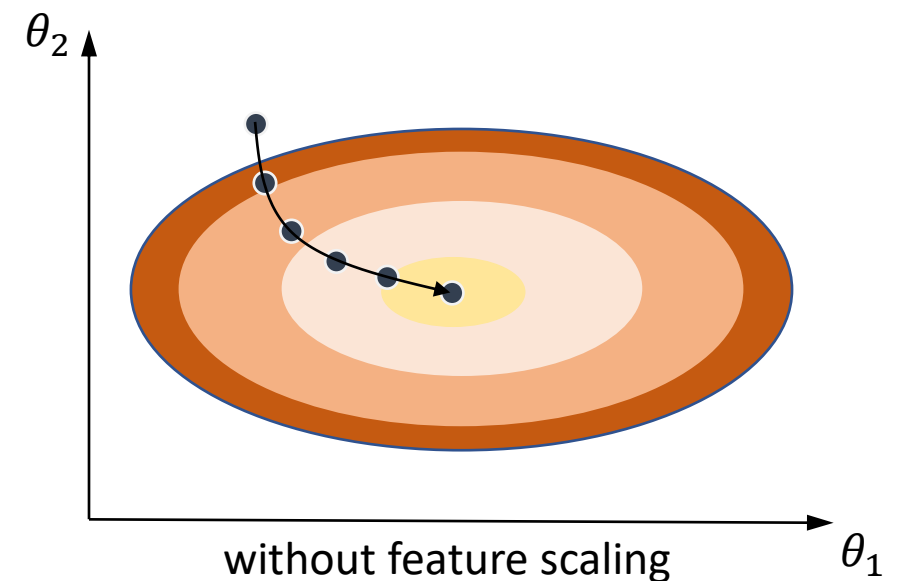
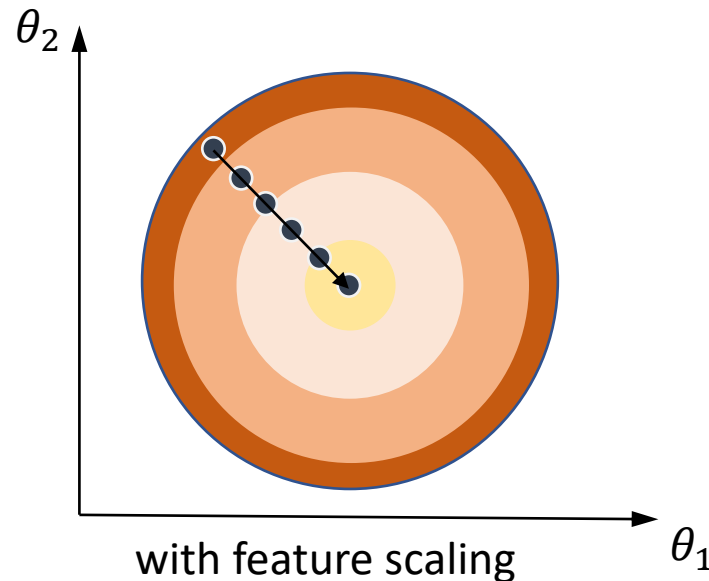
- Cost function might have valleys, ridges, plateaus and other irregular shapes → makes it difficult to converge to the minimum
- Challenges:
 - Getting stuck in a **local minimum**, which is not as good as the **global minimum**
 - Taking very long to cross a **plateau** and unwillingly stopping too early before the global minimum is reached



- MSE cost function for linear regression is
 - Convex
 - The line segment between any two points on the curve does not cross the curve
 - Implies that there are no local minima, just one global minimum
 - Continuous without its slope changing abruptly
 - The derivative is Lipschitz continuous
- As a consequence, it is guaranteed that gradient descent approaches arbitrarily close the global minimum (if the learning rate allows it)

Gradient Descent

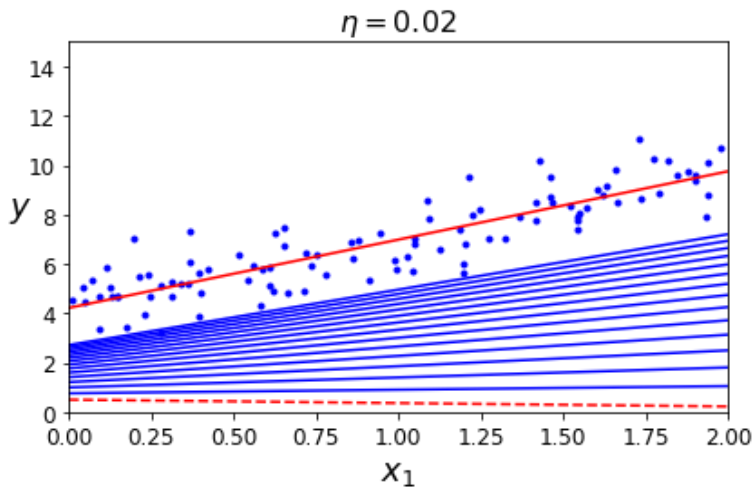
- Cost functions for models with two or more parameters
 - The bowl-like cost function can be elongated if the features have different scales



- Feature should have similar scale or else it will take much longer to converge
- Training a model is searching the model's parameter space for a combination of model parameters that minimizes a cost function (over the training data)

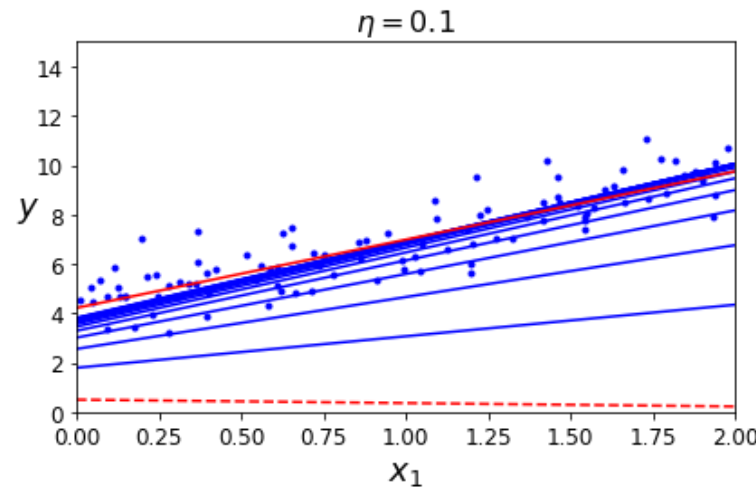
Batch Gradient Descent

- Multiplying the gradient vector by the learning rate η determines the size of the learning step



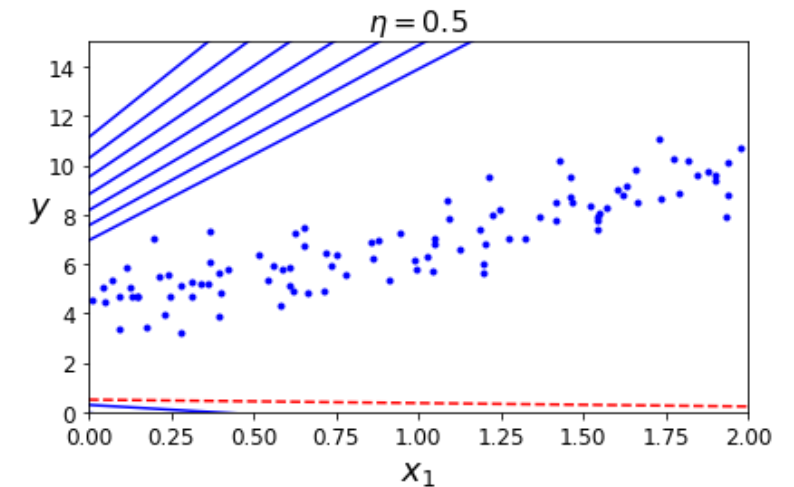
learning rate is too low

→ algorithm will eventually reach the minimum, but it takes a long time



learning rate is good

→ algorithm converges in just a few iterations



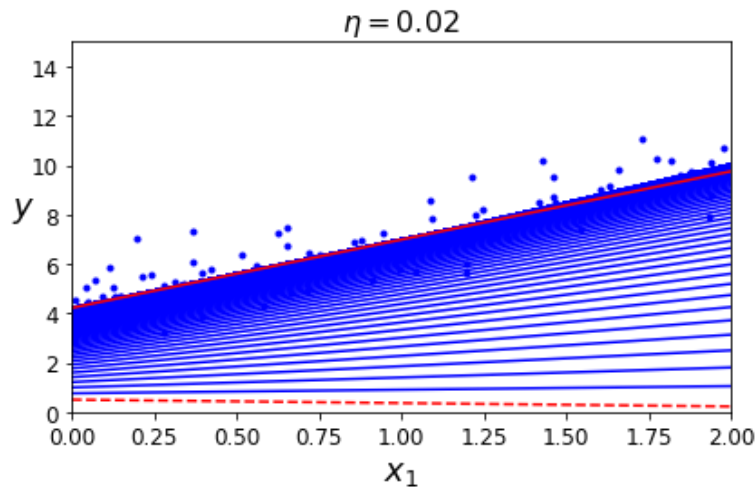
learning rate is too high

→ algorithm diverges, the model jumps around and gets further and further from the solution at every step

- Use grid search to find a good learning rate

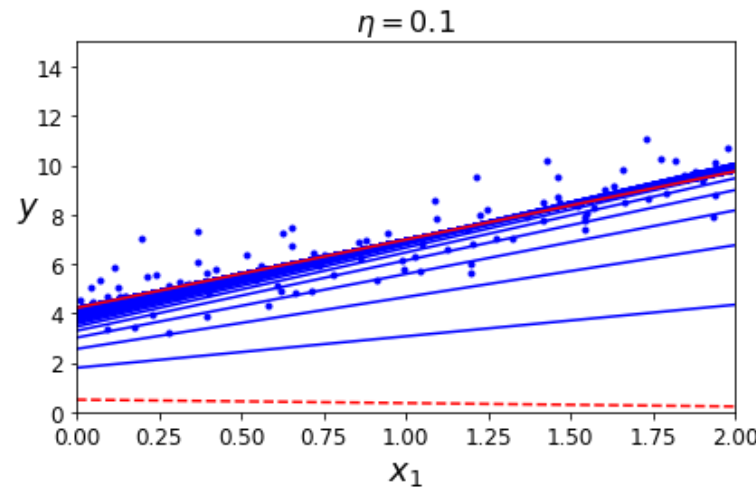
Batch Gradient Descent

- Multiplying the gradient vector by the learning rate η determines the size of the learning step



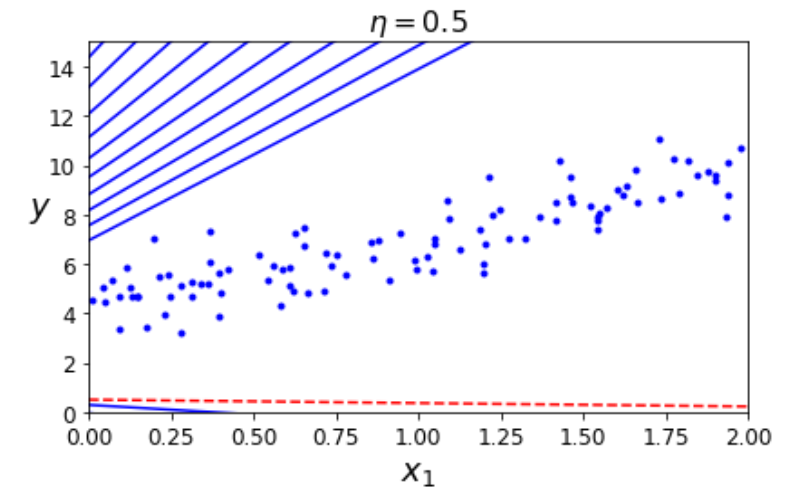
learning rate is too low

→ algorithm will eventually reach the minimum, but it takes a long time



learning rate is good

→ algorithm converges in just a few iterations



learning rate is too high

→ algorithm diverges, the model jumps around and gets further and further from the solution at every step

- Use grid search to find a good learning rate

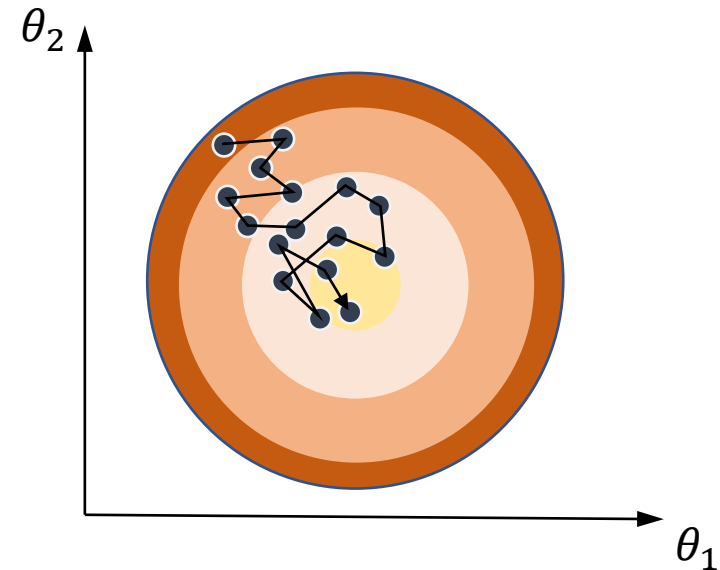
- Number of iterations:
 - Too few, the model parameters will still be far from the optimal solution when the algorithm stops
 - Too many, the algorithm wastes time while the model parameters do not change anymore
- Solution:
 - Set a very large number of iterations, but interrupt the algorithm when the gradient vector becomes tiny, i.e. the norm becomes smaller than a tiny number ε (called tolerance) \rightarrow algorithm has (almost) reached the minimum

Stochastic Gradient Descent

- Batch Gradient Descent uses the whole batch of training data at every step → very slow on large training data sets
 - However, scales well with the number of features
- **Stochastic Gradient Descent** picks one random training data instance at every step and computes the gradients only on that single instance
 - Faster as few computations are necessary for each iteration (gradient step)
 - Possible to train on huge data sets as only one instance needs to be in memory

Stochastic Gradient Descent

- Batch Gradient Descent
 - Cost function gently decreases until it reaches the minimum
- Stochastic Gradient Descent
 - Cost function bounces around and only decreases on average
 - Ends up very close to the minimum, where it continues to bounce around and never reaches it
 - This irregular behavior can help to jump out of a local minimum, therefore having a better chance to find the global minimum



- To take advantage of and mitigate the drawbacks of randomness:
 - Start training with large learning steps
→ quick progress and escape local minima
 - Gradually reduce the learning steps
→ allows to settle at the global minimum
- } simulated annealing
- **Learning schedule** determines the learning rate at each iteration
 - Learning rate reduced too quickly
→ may get stuck in local minimum or stop early
 - Learning rate reduced too slowly
→ jumps around the minimum and end up with a suboptimal solution

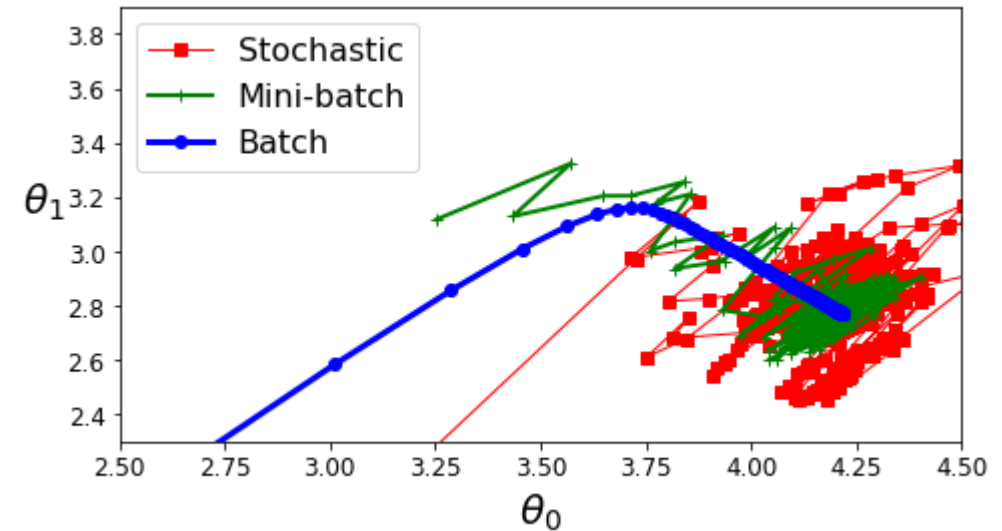
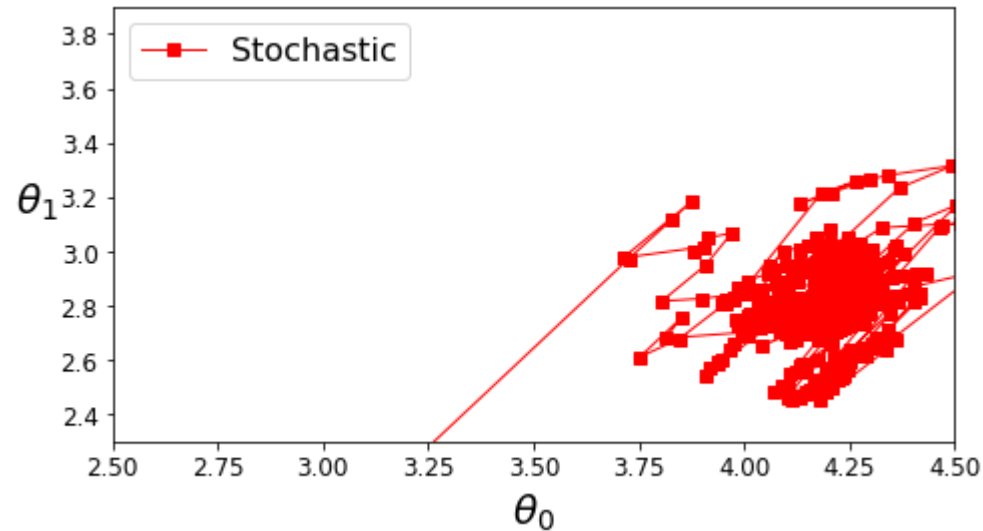
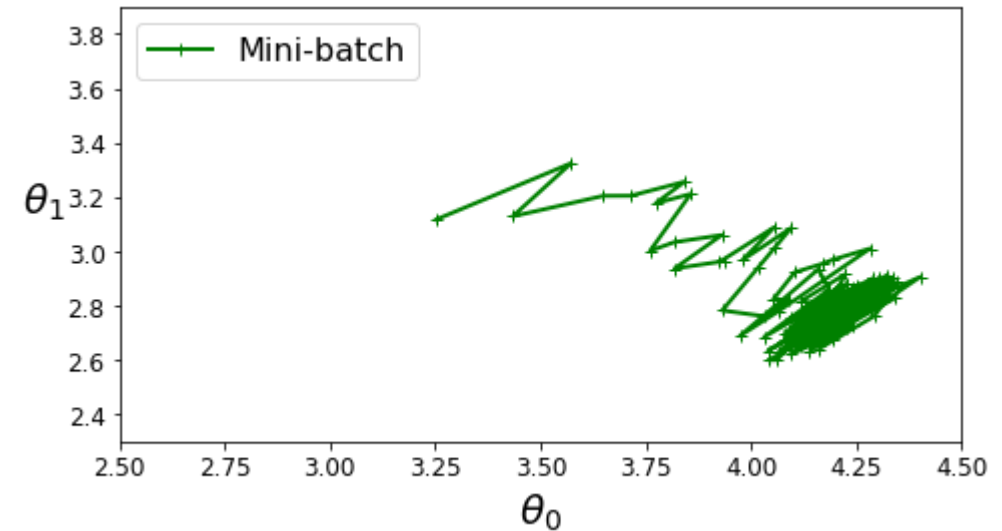
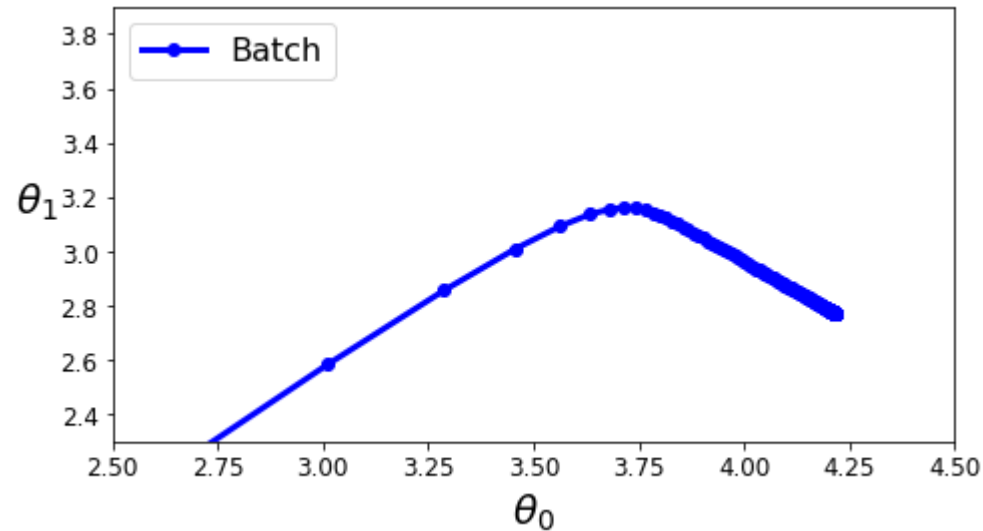
Stochastic Gradient Descent

- Learning is performed in several rounds, called **epochs**:
 - An epoch mostly refers to a complete presentation of the data set to the algorithm
 - Since Stochastic Gradient Descent randomly chooses data elements, it cannot be guaranteed that this ever happens → epoch sometimes refers to the number of training steps with the same learning rate
 - Alternatively, the training data is shuffled at each epoch and the algorithm goes through the complete data set instance by instance → converges more slowly
- Stochastic Gradient Descent generally needs fewer epochs to converge than Batch Gradient Descent

Mini-Batch Gradient Descent

- In Mini-Batch Gradient descent the gradients are computed on small sets of random instances called mini-batches
 - Progress is less erratic in parameter space than Stochastic Gradient Descent
 - Will end up walking around closer to the minimum
 - But harder to escape from local minima

Comparison of all Training Algorithms



Comparison of all Training Algorithms

Algorithm	Large m (size of training data set)	Out-of-core	Large n (number of features)	Hyper-parameters	Scaling required	Scikit-Learn
Normal Equation	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

Thank you for your attention!