Principles of
Machine Learning

Week 8: Linear Regression in One and Multiple Variables (Part 1)

Learning Objectives

After successfully completing this topic, you will be able to …

- Explain what Linear Regression is and its use in ML
- Describe and implement a Gradient Descent algorithm for learning LR parameters with one or multiple variables
- Describe and implement extensions such as Stochastic GD, regularisation, and polynomial regression
- Considering the characteristics of linear regression, recommend when it would be appropriate to an application
- Discuss and apply feature engineering methods including feature scaling, feature reduction, and transformation methods (e.g. PCA)

Structure of Videos for This Topic

Week 8

- Linear regression; closed-form solution for linear regression
- Gradient descent for linear regression with one input variable
- Multiple Linear Regression: solving in closed form and with gradient descent

Week 9

- Feature scaling; polynomial regression
- Bias, variance, underfitting and overfitting in regression
- Gradient descent with regularisation; dimension reduction
- Feature engineering approaches including transformations and subset selection

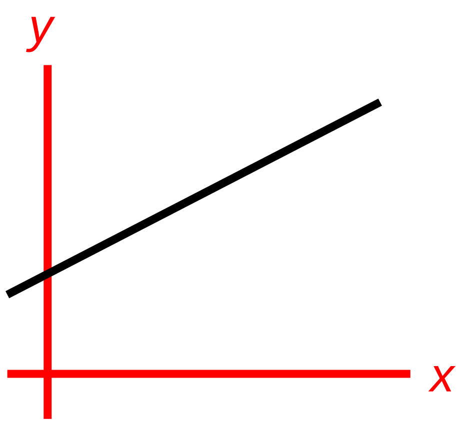
Part 1:
Linear Regression

Linear Regression Overview

- Given target value (y) and set of attribute values (x_m), goal of Linear Regression is to find an equation that describes the target in terms of the attributes
- Equation is of the form:

  $$y = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \ldots + \theta_m x_m$$

  where θ_m is the weight associated with attribute x_m



- This is the equation of a line
  - Compare to a 2D line, where x and y are dimensions: $y = a + b\,x$
  - Now have higher dimensions, instead of x we have x_1, x_2, etc
  - Use θ_0 as intercept rather than a, and θ_1 relates to slope in dimension x_1

Linear Regression Overview

- Have to find equation of this form:

$$y = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \ldots + \theta_m x_m$$

where θ_m is the weight associated with attribute x_m

- In other words, have to find values of θ_0 θ_1 θ_2 etc

- Weights found using least squares fit
  - For a given training set, the weights are found that minimize the squared error between predicted and actual target values
  - If a weight is 0, that attribute has no effect on outcome
  - Assumes that there is little/no correlation between dimensions

Linear Regression Overview

- Training data consists of sets of values for $x_1 \; x_2 \ldots y$

Size (m^2)	# Beds	# Floors	Age (yrs)	Price (k€)
195	5	1	40	450
130	3	2	35	220
140	3	2	26	310
80	2	1	30	170
180	5	2	38	400
x_1	x_2	x_3	x_4	y

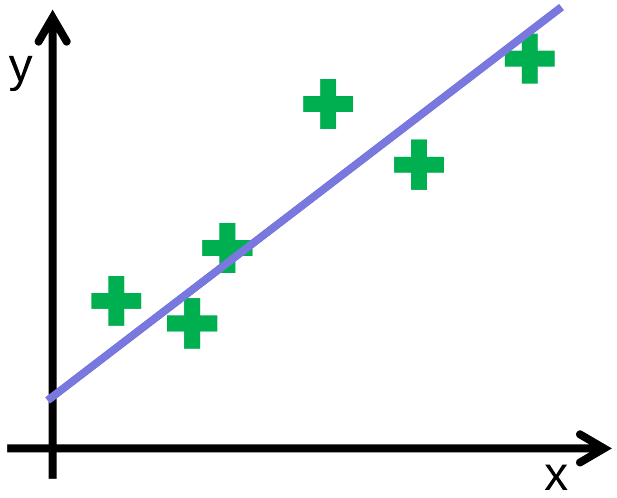
- We formulate a hypothesis of the form

  $h_{\boldsymbol{\theta}}(\boldsymbol{x}) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \ldots + \theta_m x_m$

  where $\boldsymbol{x}$ is vector $[x_1 \; x_2 \; \ldots]$ and $\boldsymbol{\theta}$ is vector $[\theta_0 \; \theta_1 \; \theta_2 \ldots]$

- **Learning objective:** find values for $\boldsymbol{\theta}$ such that the value output by $h_{\boldsymbol{\theta}}(\boldsymbol{x})$ for a given vector $\boldsymbol{x}$ is as close to y as possible

Linear Regression in 1 Variable



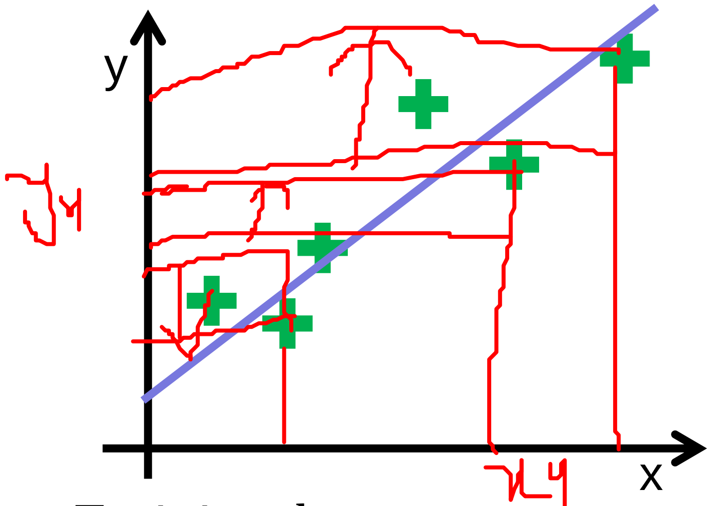
Training data: green +

Hypothesis: blue line.

Hypothesis $h_\theta(x)$
has the form $h_\theta(x) = \theta_0 + \theta_1 x$

θ_0, θ_1 are the **model parameters/weights**.

Linear Regression in 1 Variable



Training data: green +

Hypothesis: blue line.

Hypothesis $h_\theta(x)$
has the form $h_\theta(x) = \theta_0 + \theta_1 x$

θ_0, θ_1 are the **model parameters/weights**.

Learning objective: choose θ_0, θ_1 such that for each training example $(x^{(i)}, y^{(i)})$, $h_\theta(x^{(i)})$ is as close as possible to $y^{(i)}$ on average.

One way to formalise this is mean squared error:

$$\min_{\theta_0,\theta_1} \frac{1}{2N} \sum_{i=1}^{N} \left(h_\theta(x^{(i)}) - y^{(i)}\right)^2$$

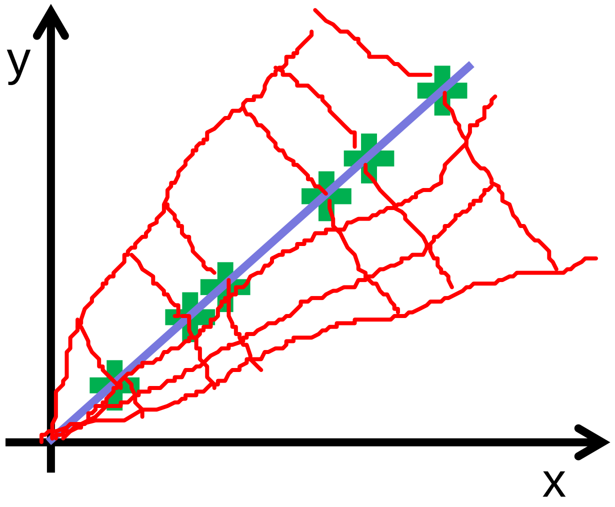
This defines a **squared error cost function**, $J(\theta_0, \theta_1)$:

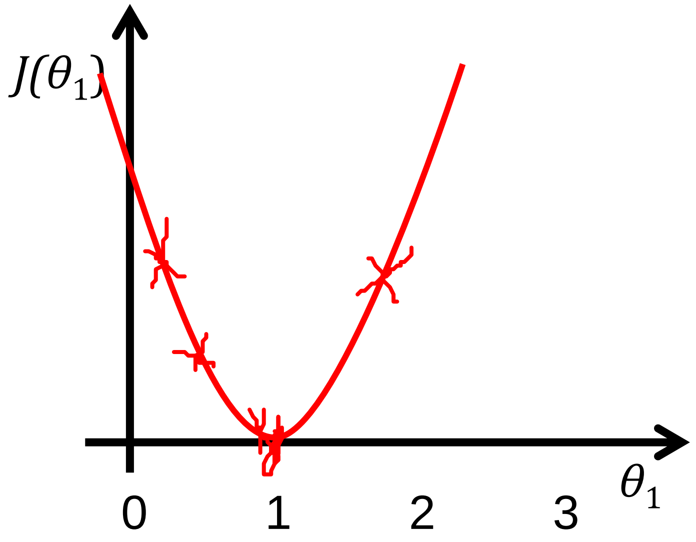
$$J(\theta_0, \theta_1) = \frac{1}{2N} \sum_{i=1}^{N} \left(h_\theta(x^{(i)}) - y^{(i)}\right)^2$$

Minimize J to find the optimal hypothesis.

Note: could use a different cost function; may yield slightly different results. Squared loss dates to Gauss.

Cost Function Behaviour [1]





Simplified case:
data values are $y=x$

Hypothesis form is $h_\theta(x) = \theta_1 x$

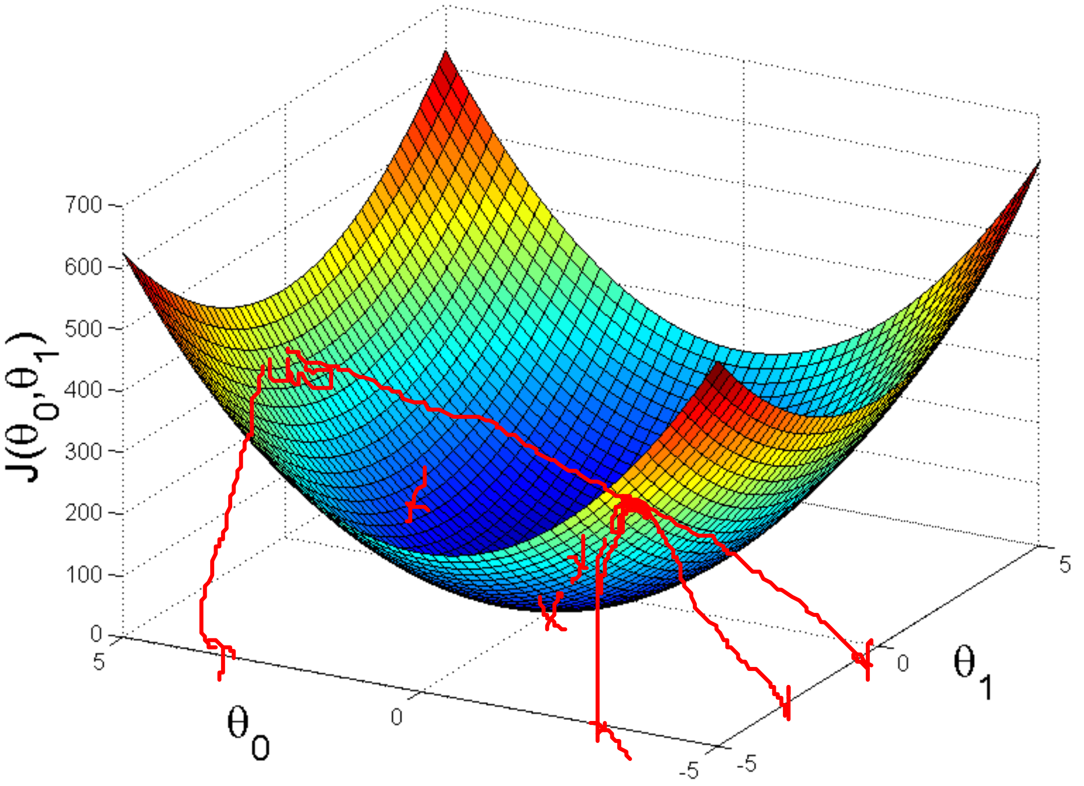
$\theta_0 = 0$ (intercept is 0)
Only vary θ_1

If there is a straight line that goes through the data, it is defined by the parameters of $\boldsymbol{\theta}$ where $J(\boldsymbol{\theta}) = 0$, irrespective of form of J.

If not, min value of J will be positive and will depend on specific cost fn used.

Either way, J is convex, has single minimum.

Cost Function Behaviour [2]



If we need to optimize θ_0 and θ_1:
still have single minimum, still a convex optimization problem.

This also holds when we move to linear regression in multiple variables, with more θ_i.

How Do We Optimize the Parameters?

There are two main ways to find θ_0 and θ_1:

1. Closed form solution:
   – Because it is a convex optimization problem, the solution has a unique form
2. Gradient Descent:
   – General-purpose algorithm for finding a local minimum of any continuous differentiable function
   – For convex hull, it can always find the correct answer if its parameters are reasonable
   – Iterative unlike closed form and therefore, not as efficient as closed form
   – Advantage: Applicable to many optimisation tasks

Either way, need partial derivatives of $J(\theta_0, \theta_1)$ …

Cost Function Derivatives

Given a cost function $J(\theta_0, \theta_1 \dots \theta_m)$,

its minimum value w.r.t. all θ_i is found when all of its partial derivatives are zero.

In this case:

$$J(\theta_0, \theta_1) = \frac{1}{2N}\sum_{i=1}^{N}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)^2 = \frac{1}{2N}\sum_{i=1}^{N}\left(\theta_0 + \theta_1 x^{(i)} - y^{(i)}\right)^2$$

Partial derivatives:

$$\frac{\partial}{\partial\theta_0}J(\theta_0, \theta_1) = \frac{1}{N}\sum_{i=1}^{N}\left(\theta_0 + \theta_1 x^{(i)} - y^{(i)}\right) = 0$$

$$\frac{\partial}{\partial\theta_1}J(\theta_0, \theta_1) = \frac{1}{N}\sum_{i=1}^{N}\left(\theta_0 + \theta_1 x^{(i)} - y^{(i)}\right)x^{(i)} = 0$$

Cost Function: Closed Form Solution

The partial derivatives have a unique solution:

$$\theta_0 = \frac{1}{N}\sum y^{(i)} - \frac{\theta_1}{N}\sum x^{(i)}$$

$$\theta_1 = \frac{N\sum x^{(i)}y^{(i)} - \left(\sum x^{(i)}\right)\left(\sum y^{(i)}\right)}{N\sum\left(\underline{x^{(i)}}^{2}\right) - (\underline{\sum x^{(i)}})^2}$$

This can be computed with a small amount of code or even in a spreadsheet.

LinRegClosedForm.xlsx

Part 2:
Gradient Descent

Gradient Descent

- General-purpose method that works **beyond linear regression**: does not require closed-form solution
- Make initial guess; take incremental steps 'downhill' with step size controlled by learning rate α until little/no change

Batch Gradient Descent **Algorithm:**

initialise $\boldsymbol{\theta}$ to any set of valid initial values

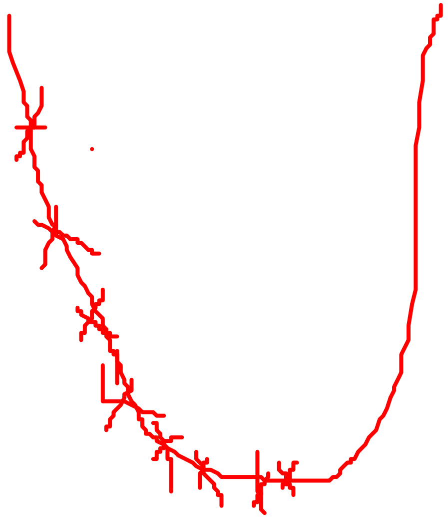
repeat until convergence (or time limit reached):

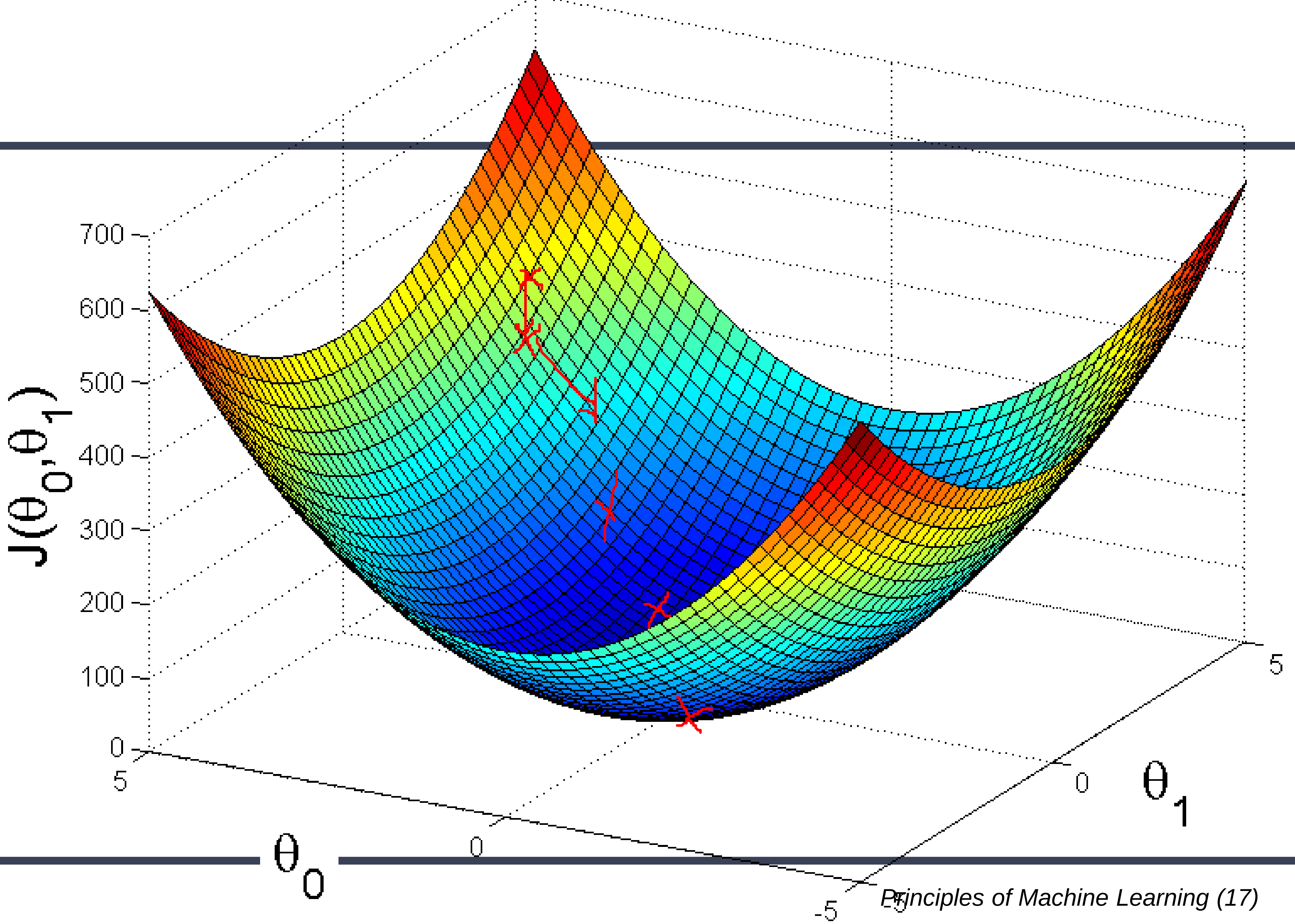
later slide

simultaneously foreach θ_j in $\boldsymbol{\theta}$ do:

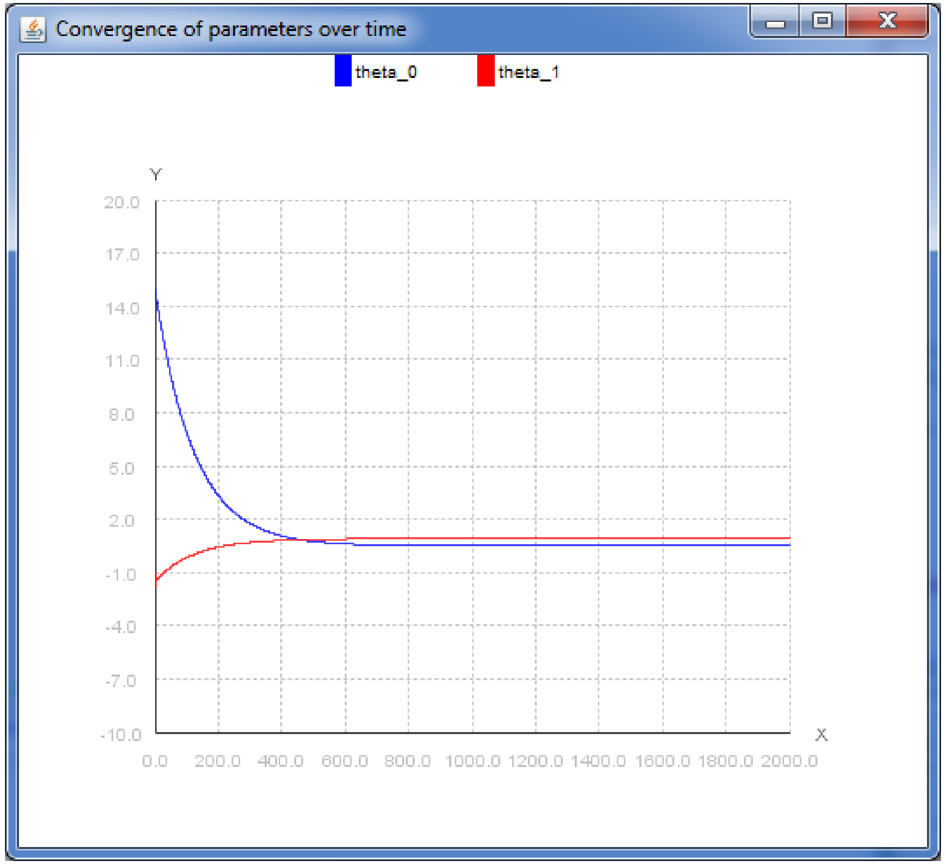
$$\theta_j \leftarrow \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\boldsymbol{\theta})$$

Gradient Descent Illustration

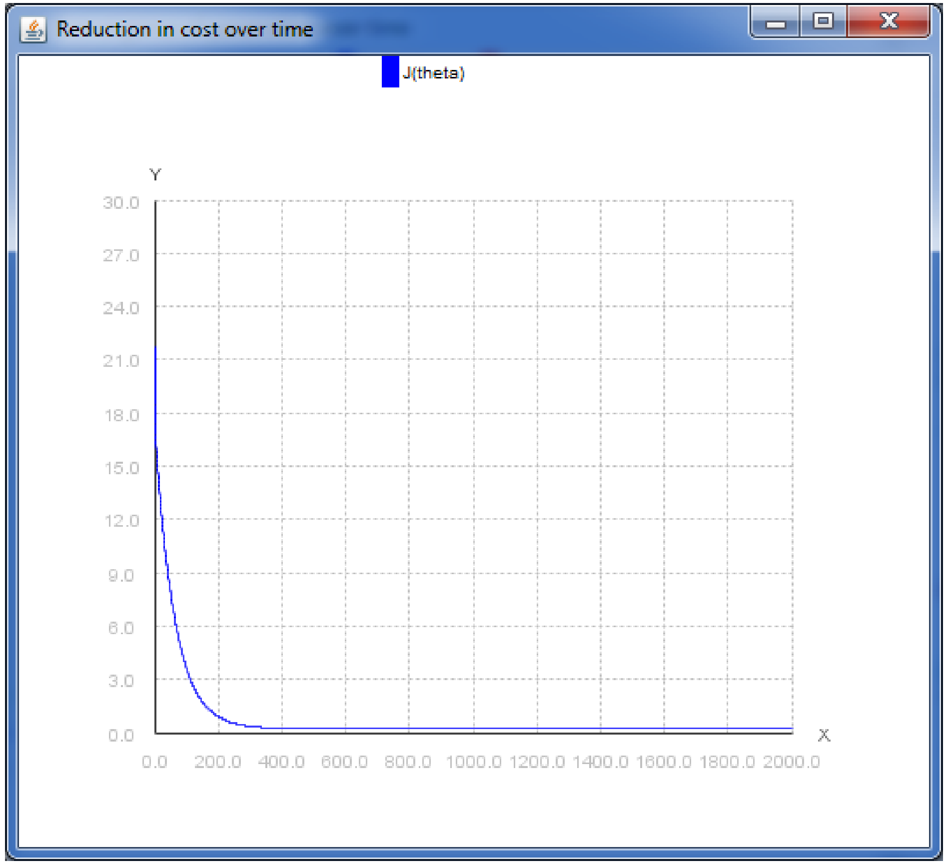




Checking for Convergence



Changes to θ_i will be initially large, reducing as iterations proceed.
If progress is slow, increase learning rate.



Cost **must decrease** in every iteration, otherwise α is too large

Gradient Descent Illustration

- I have written two implementations of Gradient Descent for Linear Regression in one variable, both of which are on Blackboard

- Java version:
  GradientDescent.java

- Python (Jupyter Notebook) version:
  SimpleLinearRegression.ipynb

- We will examine the second one …

Stochastic Gradient Descent

- Rather than using all data together ("batch") in the update function, randomly select a single example for updating $\boldsymbol{\theta}$
  - Keep repeating until convergence
  - Partial derivatives are simplified: only 1 example so N=1, no Σ

$$\frac{\partial}{\partial \theta_1} J(\boldsymbol{\theta}) = \cancel{\frac{1}{N}} \cancel{\sum_{i=1}^{N}} \left(h_\theta(\boldsymbol{x}^{(i)}) - y^{(i)} \right) x_1^{(i)}$$

Stochastic Gradient Descent

- Can be faster than batch gradient descent
  - Convergence is not guaranteed

- Works well in **online learning**
  - New data arriving all of the time
  - May be ***high velocity*** data
  - As long as all data are identically drawn from same distribution, just sample the data stream as fast as possible for updates

Part 3:
Multiple Linear Regression

Multiple Linear Regression [1]

- Usually, have multiple input variables that relate to a quantity of interest

	Size (m^2)	# Beds	# Floors	Age (yrs)	Price (k€)
	195	5	1	40	450
	130	3	2	35	220
$\boldsymbol{x}^{(3)}$ $y^{(3)}$	140	3	2	26	310
	80	2	1	30	170
	180	5	2	38	400
	x_1	x_2	x_3	x_4	y

N no. of cases

$x_2^{(3)}$

No. of attributes: m

Now, $\boldsymbol{x}^{(i)}$ is a feature vector of length m, values from one row

Individual features denoted $x_j^{(i)}$

Multiple Linear Regression [2]

Hypothesis is now of the form:

$h_{\boldsymbol{\theta}}(\boldsymbol{x}) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \ldots + \theta_m x_m$

To simplify notation, add a dummy input variable $x_0 = 1$.

Then: $h_{\boldsymbol{\theta}}(\boldsymbol{x}^{(i)}) = \boldsymbol{\theta} \cdot \boldsymbol{x}^{(i)} = \sum \theta_j \, x_j^{(i)}$

Cost function essentially same as before, now using vector notation:

$$J(\boldsymbol{\theta}) = \frac{1}{2N} \sum_{i=1}^{N} \left(h_{\boldsymbol{\theta}}(\boldsymbol{x}^{(i)}) - y^{(i)} \right)^2$$

Reminder of vector dot product:

a = <a_1 a_2 a_3 a_4>
b = <b_1 b_2 b_3 b_4>

a . b = $a_1 b_1 + a_2 b_2 + \ldots$

Multiple LR: Closed Form Solution

As before, can be solved in closed form.

Let $\mathbf{X}$ be full data matrix of inputs: row i corresponds to $\boldsymbol{x}^{(i)}$

Let $\boldsymbol{y}$ be vector of all outputs

Then the optimal vector of values for $\boldsymbol{\theta}$ that minimises squared error is found from:

$$\boldsymbol{\theta}^* = (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \boldsymbol{y}$$

Non-iterative, but requires inverting a matrix of size $m \times m$ (m = no. of attributes): $O(m^3)$

Gradient descent faster for large m (e.g. >1000).

Multiple LR: Gradient Descent

The algorithm is **the same** for Multiple LR as before:

initialise $\boldsymbol{\theta}$ to any set of valid initial values

repeat until convergence or until limit is reached:

  simultaneously foreach θ_j in $\boldsymbol{\theta}$ do:

$$\theta_j \leftarrow \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\boldsymbol{\theta})$$

The partial derivatives are **different**: can be verified to be direct generalisation of the single-variable case:

$$\frac{\partial}{\partial \theta_j} J(\boldsymbol{\theta}) = \frac{1}{N} \sum_{i=1}^{N} \left(h_\theta(\boldsymbol{x}^{(i)}) - y^{(i)} \right) x_j^{(i)}$$

Multiple LR: Gradient Descent

- Let's compare the partial derivatives …
- Single variable case:

$$\frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1) = \frac{1}{N} \sum_{i=1}^{N} \left(h_\theta(x^{(i)}) - y^{(i)} \right)$$

$$\frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1) = \frac{1}{N} \sum_{i=1}^{N} \left(h_\theta(x^{(i)}) - y^{(i)} \right) x^{(i)}$$

- Multiple variable case:

$$\frac{\partial}{\partial \theta_j} J(\boldsymbol{\theta}) = \frac{1}{N} \sum_{i=1}^{N} \left(h_{\boldsymbol{\theta}}(\boldsymbol{x}^{(i)}) - y^{(i)} \right) x_j^{(i)}$$

End of

Linear Regression in One and Multiple Variables (Part 1)