UiO : Department of Informatics
University of Oslo

INF 5860 Machine learning for image classification

Lecture 2 : Linear classification and regression – part 1: Regression

Anne Solberg

January 27, 2017

Today's topics

- Self-study: Linear algebra (Chapter 2)
- Linear regression
  - Deep Learning Chap 5.1
- Introduction to loss functions and minimization
- Read section1-3 in http://cs229.stanford.edu/notes/cs229-notes1.pdf
- Gradient descent
- Briefly about polynomial regression

Introduction

- Linear regression has many similarities to neural nets, and it is easy to explain the role of learning a loss function on a data set.
- Classification can be viewed as a regression problem.
  - Classification: estimate the class label k=1….K
  - Regression: estimate a continuous variable y
  - Both methods use training data to estimate the parameters.
- Linear mappings $\theta^T x$ are the fundament in both.

  This is also the basic operation of a node in a neural net

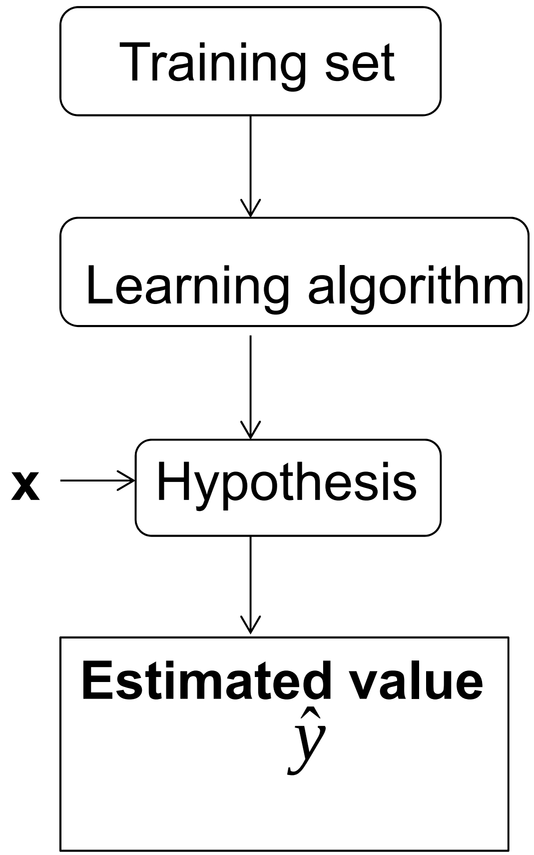
The linear regression problem

- Task T: predict the true values y based on data vector **x** from a training data set.
- In regression, we want to predict y (a continuous number) based on data **x**.
  - Example: predict the development of the population in Norway based on measurements from 1990-2010.
- Predictions are based on the estimated values, and a linear hypothesis

A straight line

$$\text{Hypothesis} \quad : \hat{y} = w^T \mathbf{x}$$

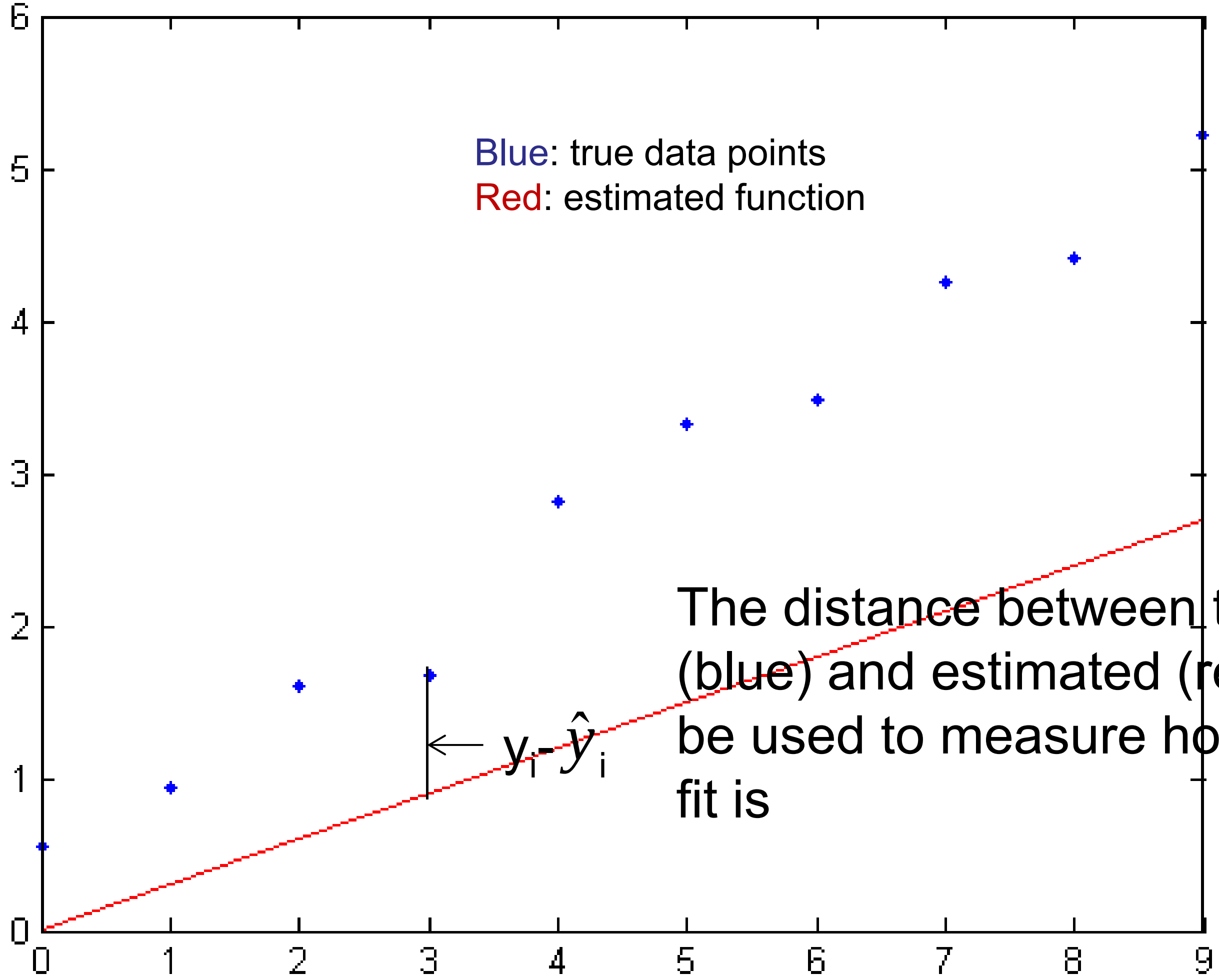
- Learning will be based on comparing y and $\hat{y}$



Linear regression: training data set

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \end{bmatrix} \qquad \mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix}$$

- Want to estimate **y** based on data **x**.
- Given m training samples where x and y are known.
- If x_i has one variable pr. sample (e.g. one gray level), this is called univariate regression.



The distance between the true (blue) and estimated (red) value will be used to measure how good the fit is

Error measure for learning linear regression: Mean square error(MSE)

- Mean square error over the training data set
- Training data: a set of m samples **x**={x_i,i,i==1..m}
- x_i can consist of one of more variables/features, e.g. several measurements.

$$J(\theta) = MSE = \frac{1}{2m}\sum_{i=1}^{m}\left(\hat{y}_i - y_i\right)^2$$

$$\text{In vector form}: \frac{1}{2m}\left\|\hat{\mathbf{y}} - \mathbf{y}\right\|_2^2 \quad \text{L2 - norm}$$

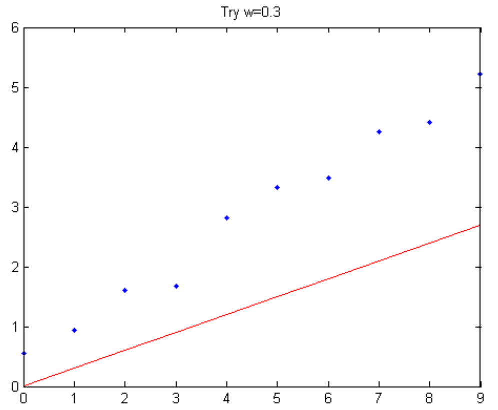
θ is the parameter we want to fit, the parameters of a line

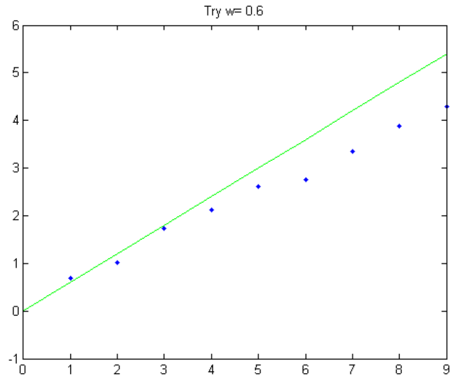
Goal: find w that minimize MSE

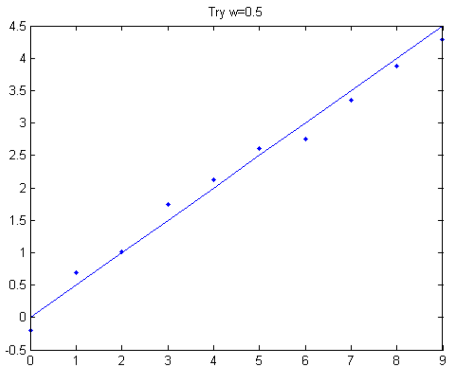
$$MSE = \frac{1}{2m}\sum_{i=1}^{m}\left(\hat{y}_i - y_i\right)^2 = \frac{1}{2m}\sum_{i=1}^{m}\left(w^T x_i - y_i\right)^2$$

- For regression, there is an analytical solution to this, because MSE is a quadratic function.
- We minimize the error by derivation, and setting the derivative to zero.
- For large data sets, it will be better to solve this iteratively using gradient descent optimization.

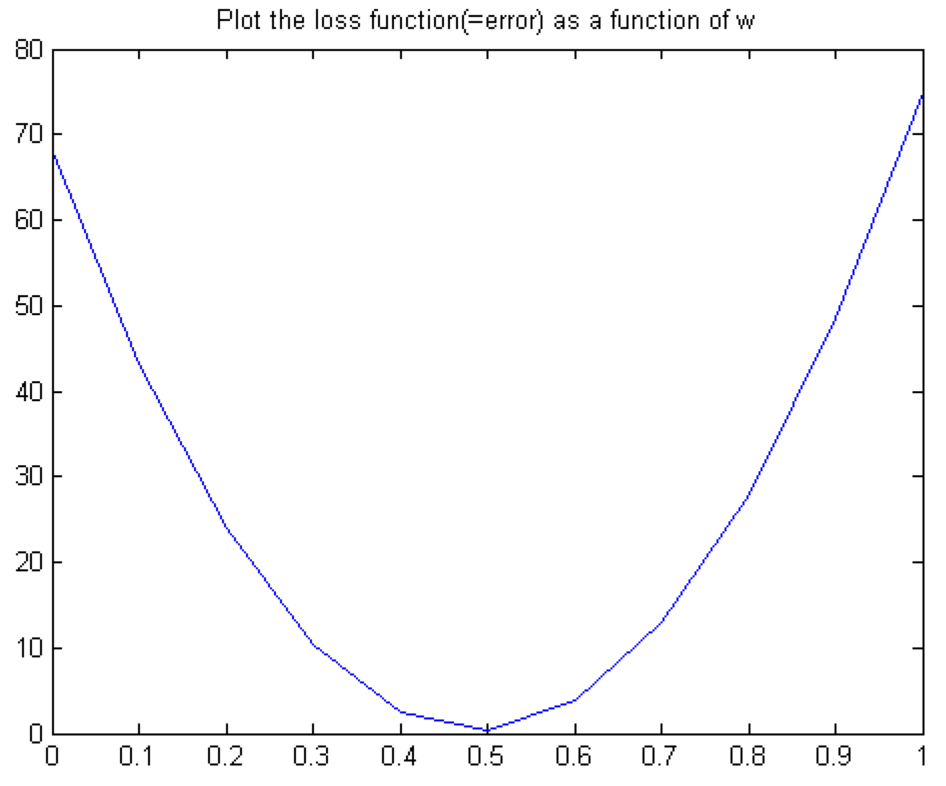
Towards an iterative solution: try different values of w and see how they fit



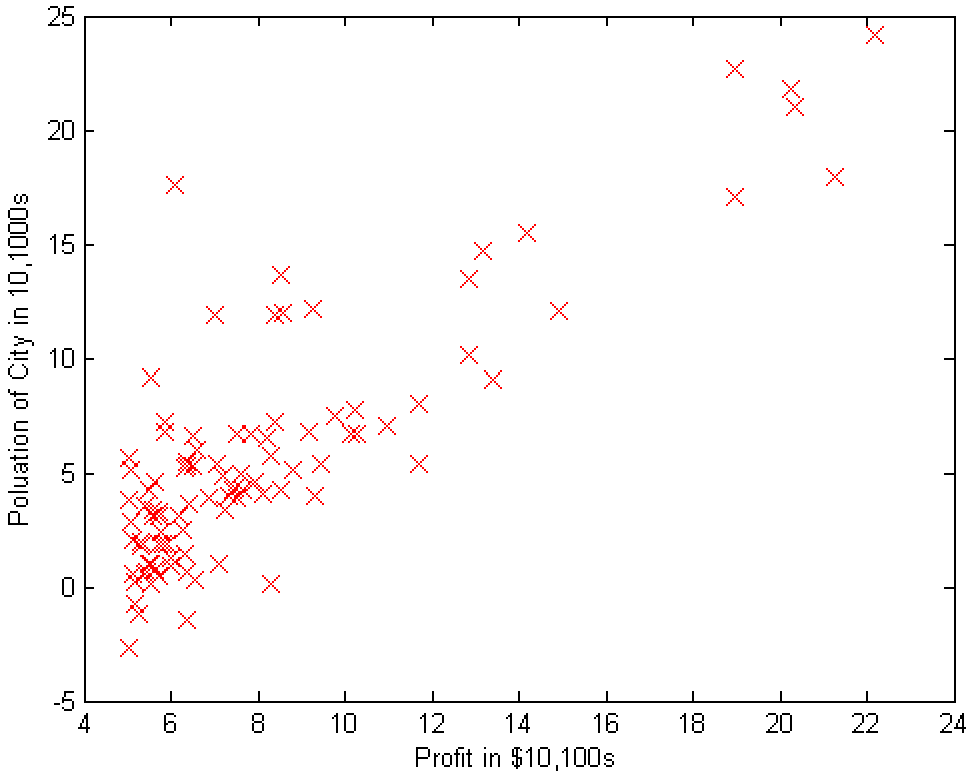




See how the loss (MSE) changes with varying w for y=wx

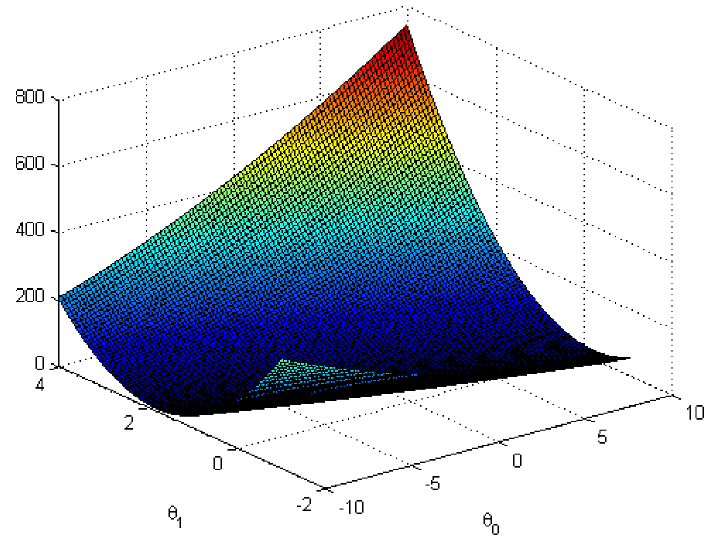


Example of J(θ^0,θ^1) for a general line ($y= \theta^1 x+ \theta^0$)



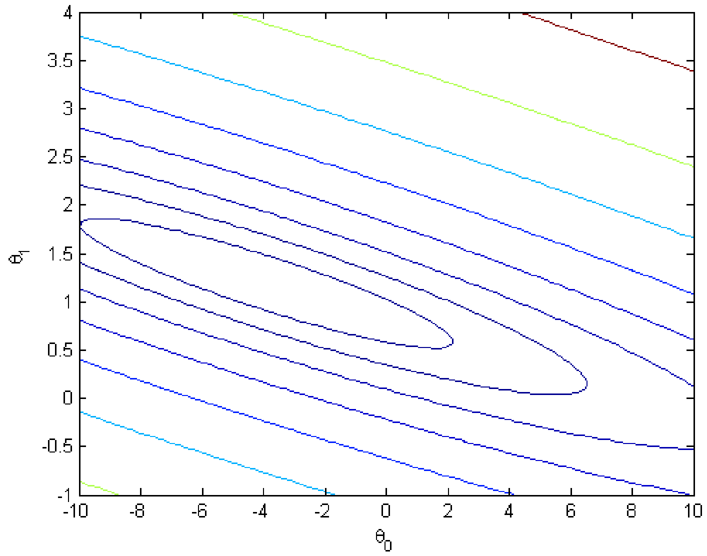
Since we know the formula for J(θ^0,θ^1), we can plot it

- Make a grid of values for θ^0,θ^1
- Compute J(θ^0,θ^1) and visualize
- Note: only valid for this data set x_1,..x_m



- Alternatively, we can plot the contours of J(θ^0,θ^1)

- Now we need an algorithm to find the minimum.

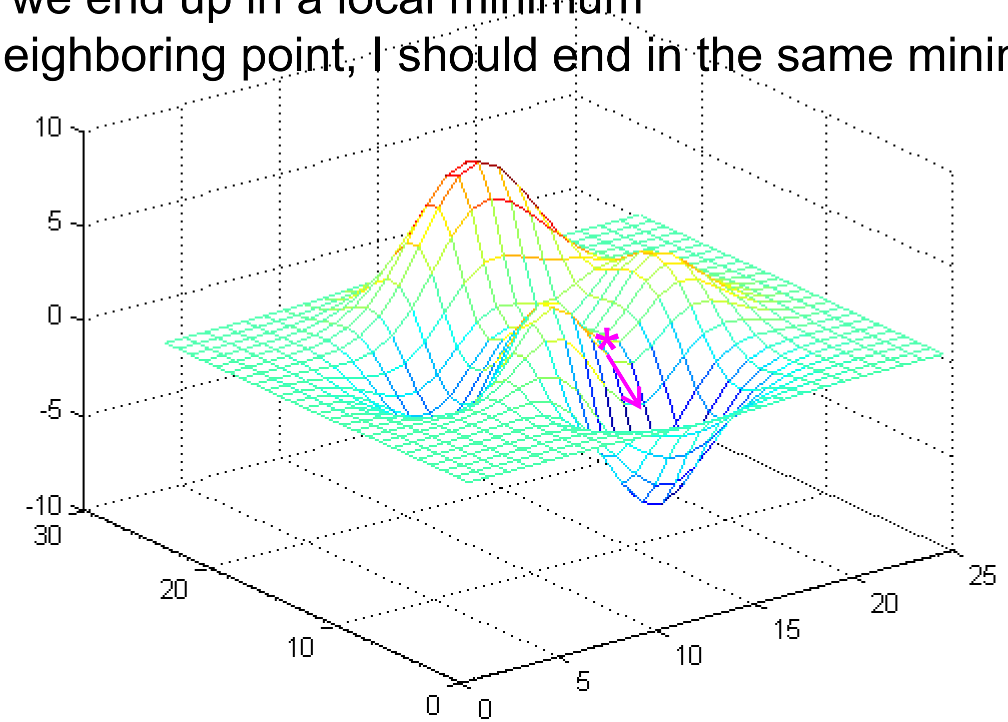


Gradient descent minimization

- Let’s see how gradient descent can be used to find w that mimize MSE.
- Read Section 4.3 in Deep Learning.

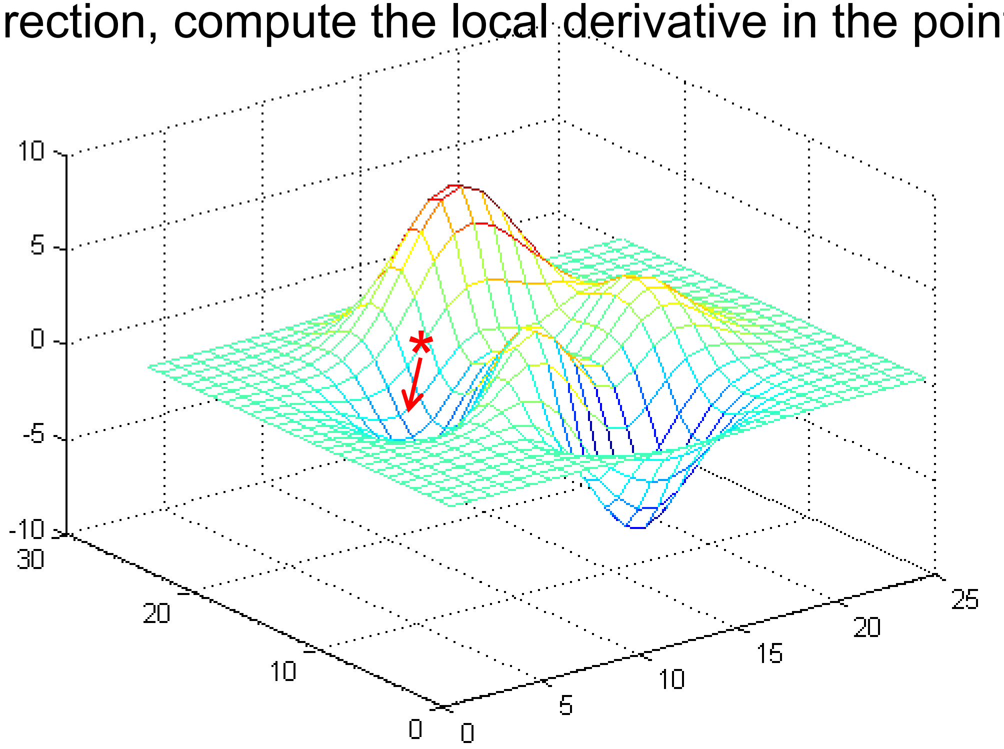
Gradient descent intuition

Start from a point and take a step downhill in the steepest possible direction
Repeat this until we end up in a local minimum
If I start from a neighboring point, I should end in the same minimum



Gradient descent intuition

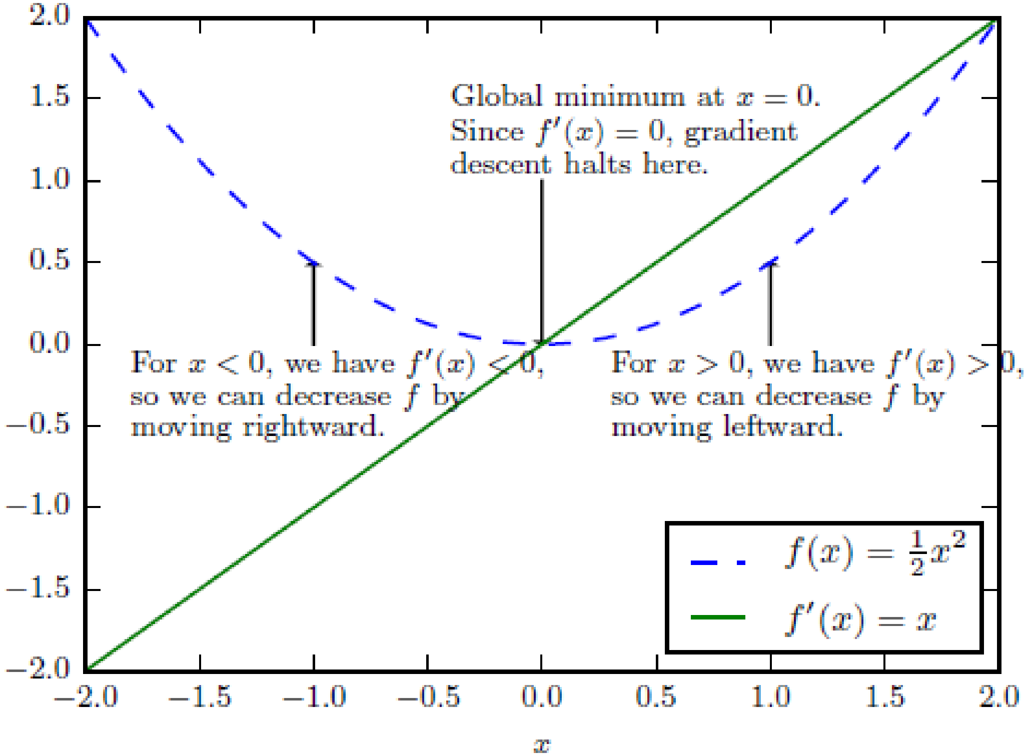
If we start from a different point we might end up in another local minimum
For finding the direction, compute the local derivative in the point



Iterative minimization outline

- Have a function $J(\theta^0,\theta^1)$ (can be generalized to more than two parameters)
- Want to find θ^0,θ^1 that minimize $J(\theta^0,\theta^1)$
- Outline
  1. Start with some value of θ^0,θ^1 (e.g. $\theta^0=0,\theta^1=0$)
  2. Compute $J(\theta^0,\theta^1)$ for the given value of θ^0,θ^1 and change θ^0,θ^1 in a manner that will decrease $J(\theta^0,\theta^1)$
  3. Repeat step 2 until we hopefully end up in a minimum

Illustration of gradients/derivatives



Gradient descent principle

- Given a function f
- The directional derivative in direction u is the slope of f in direction u.
- To iteratively minimize f, we want to find the direction in which f decreases the fastest:

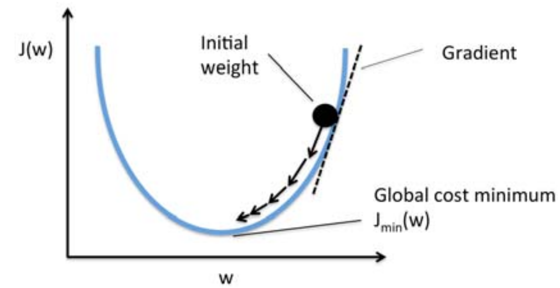
$$\min_{u,u^T u=1} \mathbf{u}^{\mathrm{T}} \nabla_x f(\mathbf{x})$$

$$= \min_{u,u^T u=1} \|\mathbf{u}\|_2 \|\nabla_x f(\mathbf{x})\|_2 \cos\theta$$

- Ignoring terms that do not depend on u, and using $||u||^2=1$, this is simplified to $\min_{\mathbf{u}} \cos(\theta)$, where θ is the angle between **u** and the gradient.
- This is mimized when u points in **the opposite direction** as the gradient.
- **So we can minimize f by taking a step in the direction of the negative gradient.**

- The gradient descent propose a new point $x' = x - \varepsilon \nabla_x f(x)$ where ε is the **learning rate**.
- If ε is too small, the algorithm converges too slow.
- It ε is too large, it may fail to converge, or diverge.

Back to linear regression



- The simplest case

$$\text{Hypothesis} \quad : \hat{y} = w^T \mathbf{x}$$

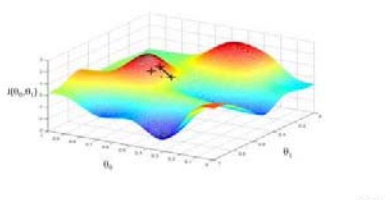
- Gives a function of one variable w

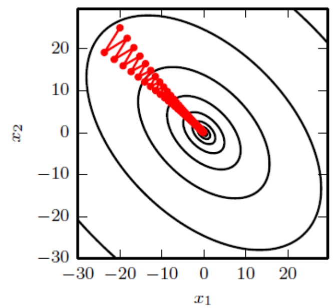
- Considering an offset b:

$$\text{Hypothesis} \quad : \hat{y} = w^T \mathbf{x} + b$$

- Gives a function of two variables w and b

[redacted]





Gradient descent for linear regression

- Let $\theta = \begin{bmatrix} \theta_1 = w \\ \theta_2 = b \end{bmatrix}$ be the parameter vector for the unknown two parameters w and b (Model: $w^T x + b$)
- We want the minimize the criterion function $J(\theta_{1,} \theta_2)$=MSE

$$MSE = \frac{1}{2m} \sum_i (\hat{y}_i - y_i)^2 = \frac{1}{2m} \sum_i \left(w^T x_i + b - y_i \right)^2$$

- Two parameters w, and b.
- Compute the derivative of $J(\theta_{1,} \theta_2)$ with respect two each of them, and set the derivative to 0.
- Note that this is quadratic (and convex) function so there are no local minima.

Gradient descent for linear regression Univariate x – a single feature/gray level

$$\frac{\partial}{\partial w} J(w,b) = \frac{\partial}{\partial w} \frac{1}{2m} \sum_i \left(w\ x_i + b - y_i\right)^2$$

$$= \frac{2}{2m} \sum_i \left(w\ x_i + b - y_i\right) x_i$$

$$\frac{\partial}{\partial b} J(w,b) = \frac{\partial}{\partial b} \frac{1}{2m} \sum_i \left(w\ x_i + b - y_i\right)^2$$

$$= \frac{2}{2m} \sum_i \left(w\ x_i + b - y_i\right)$$

Here we use the chain rule

- Here we sum the gradient over all x_i in the training data set.
- This is called **batch gradient descent.**

Gradient descent algorithm for one variable x

Gradient descent

repeat until convergence

for j=0:1

$$\theta^j = \theta^j - \varepsilon \frac{\partial}{\partial \theta^j} J(\theta^0, \theta^1)$$

Update θ^0, θ^1 simultaneously

Linear regression model

$$\hat{y} = w^T \mathbf{x} + b$$
$$= \theta^0 + \theta^1 x$$

$$J(\theta_0, \theta_1) = \frac{1}{2m} \sum_i (\hat{y}_i - y_i)^2 = \frac{1}{2m} \sum_i (w^T x_i + b - y_i)^2$$

Gradient descent example on the whiteboard

$$x = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} \quad y = \begin{bmatrix} 1 \\ 1.5 \\ 2.5 \end{bmatrix}$$

$$J(\theta_1, \theta_2) = \frac{1}{2m} \sum_i \left(\hat{y}_i - y_i \right)^2$$

Compute the loss function for $\theta = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$

Compute θ after one iteration

The linear regression problem, one variable

Hypothesis: $h(\theta) = \hat{y} = \theta^0 + \theta^1 x$

Parameters: θ^0 and θ^1

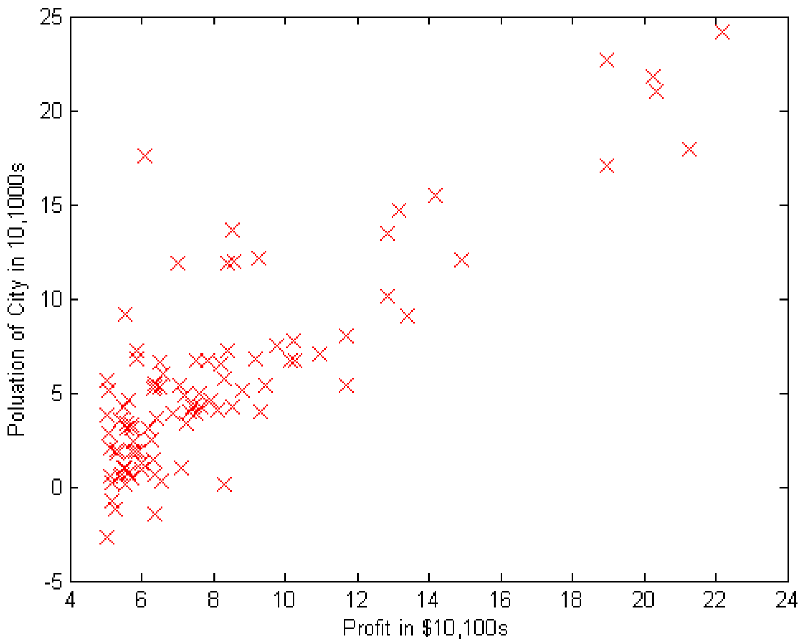
Cost function: $J(\theta^0, \theta^1) = \frac{1}{2m} \sum_{i=1}^{m} (\hat{y}_i - y_i)^2$

Goal: $\underset{\theta^0, \theta^1}{\text{minimize}} \, J(\theta^0, \theta^1)$

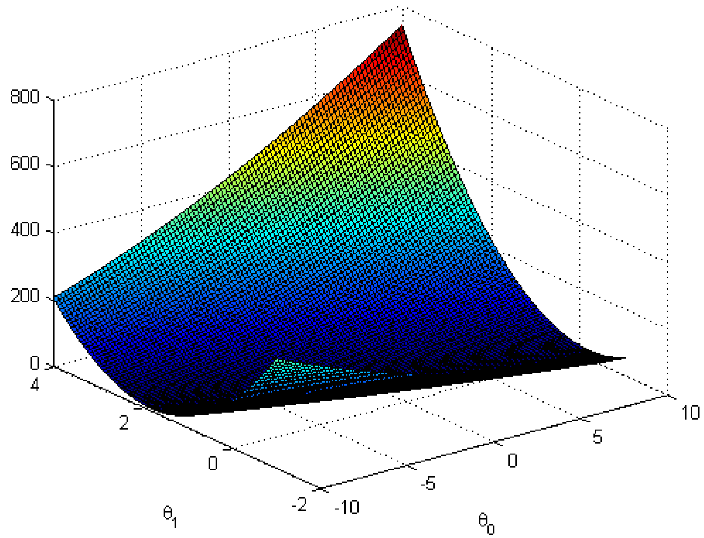
Gradient descent solution:
repeat until convergence
for j=0:1

$$\theta^j = \theta^j - \varepsilon \frac{\partial}{\partial \theta^j} J(\theta_1, \theta_2)$$

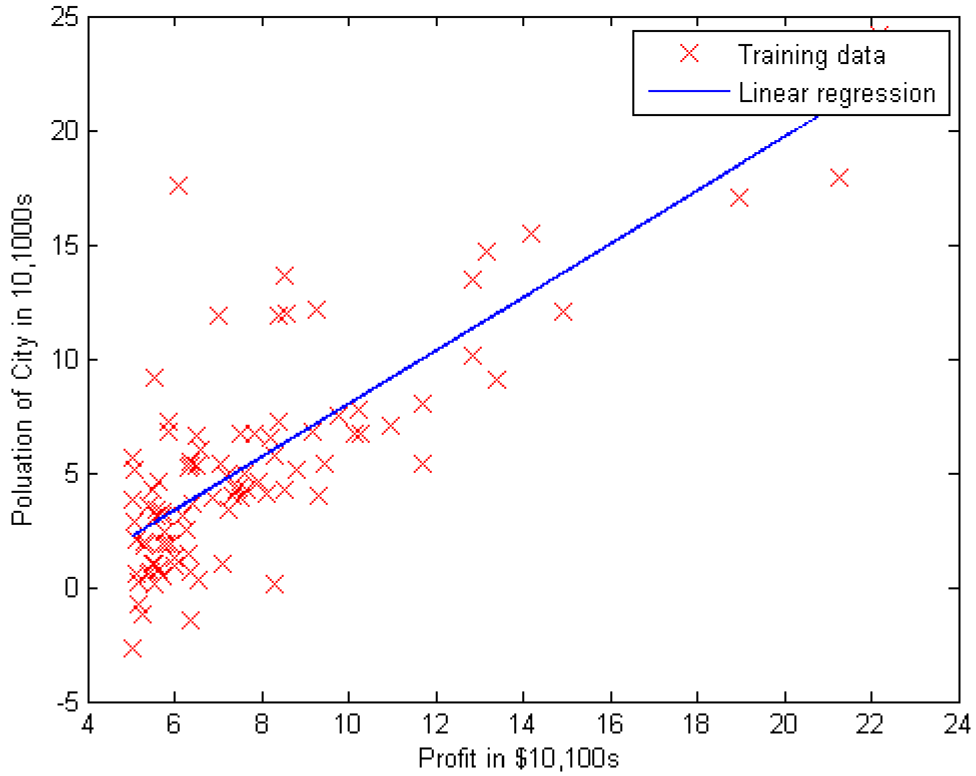
Back to the example



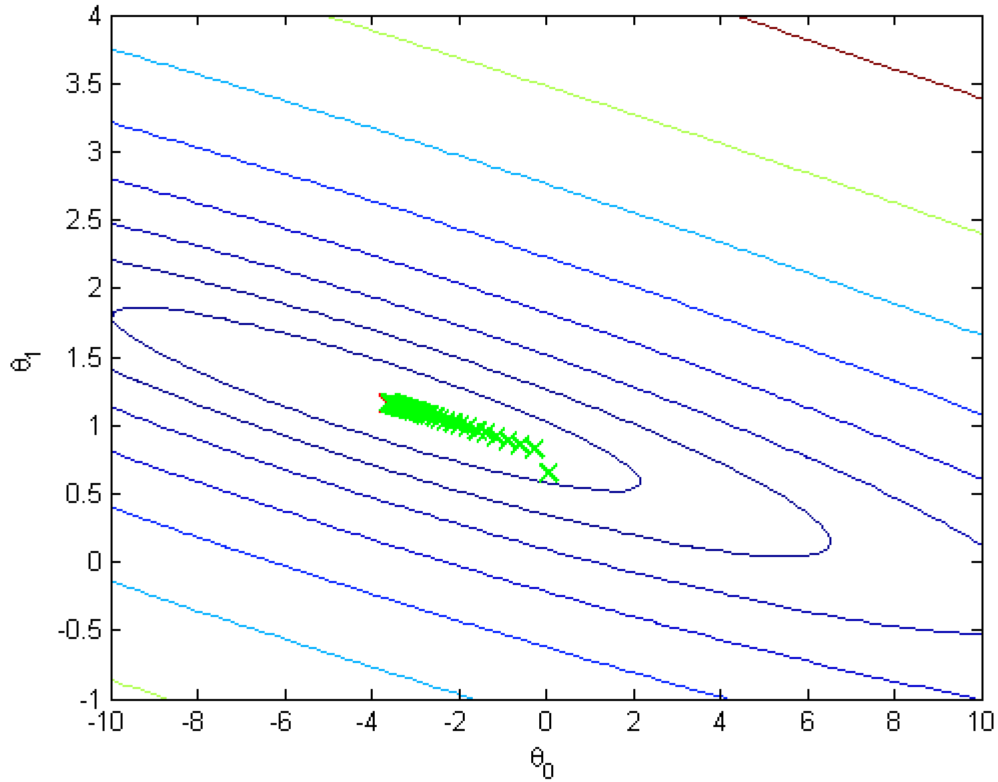
Cost function J



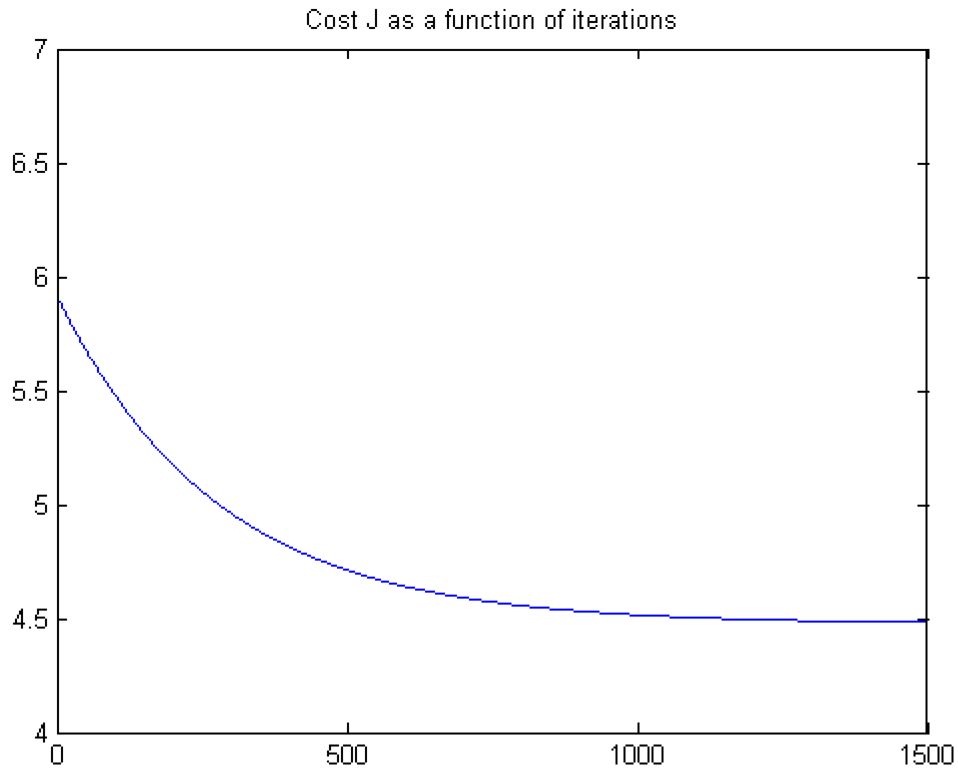
The result from gradient descent (-3.63,.16)



The value of J overlaid the values of θ^0,θ^1 after every 50th iteration



J as a function of iterations



Multiple features/variables x.

- Example: predict house price as function of 4 features (size, number of bedrooms, number of floor, age):

Size (feet^2) X^1	Number of bedrooms x^2	Number of floors x^3	Age (years) x^4	Price (1000$) y
2104	5	1	45	460
1416	3	2	41	232
1534	3	2	30	315
852	2	1	36	178

- Notation:
  - n. number of features
  - x_i : vector of n features for sample i
  - x_i^j : value of feature j for sample i

- Hypothesis:

$$h_\theta(x) = \theta^0 \quad + \theta^1 x^{(1)} + \theta^2 x^{(2)} + \theta^3 x^{(3)} + \theta^4 x^{(4)}$$

$$\text{An example}: h_\theta(x) = 80 \quad + 0.1x^{(1)} + 0.01x^{(2)} + 3x^{(3)} - 2x^{(4)}$$

Linear regression with multiple variables

- So, if we want to predict y based on n measured features, $x^{(1)}$, $x^{(2)}$, $x^{(3)}$….. $x^{(n)}$
- Example: color image with R,G,B values (n=3)

$$\hat{y} = \theta^0 + \theta^1\ x^{(1)} + \theta^2\ x^{(2)} + \theta^3\ x^{(3)} + \cdots + \theta^n\ x^{(n)}$$

- Trick: For convenience, define $x^{(0)}$=1 for compact notation

$$x_i = \begin{bmatrix} x_i^{(0)} = 1 \\ x_i^{(1)} \\ \vdots \\ x_i^{(n)} \end{bmatrix} \quad \theta = \begin{bmatrix} \theta^0 \\ \theta^1 \\ \vdots \\ \theta^n \end{bmatrix}$$

x is a (n+1)x1 matrix
θ is a (n+1)x1 matrix
θ^T is a 1x(n+1) matrix

$$\hat{y} = \theta^0 x^{(0)} + \theta^1\ x^{(1)} + \theta^2\ x^{(2)} + \theta^3 x^{(3)} + \cdots + \theta^n x^{(n)}$$

$$= \theta^T x$$

The design matrix X

- For vector-implementations of e.g. prediction, it is convenient to organize the measurements in a matrix X called the design matrix:
- If n=2, x_i is the 2 measurements for sample i
- In the X-matrix below, each row consists of x_i^T

$$\mathbf{X} = \begin{bmatrix} 1 & \text{all n features for training samle 1} \\ 1 & \text{all n features for training sample 2} \\ & \vdots \\ 1 & \text{all n features for training sample m} \end{bmatrix}$$

Generalize the gradient descent to more features/variables

Gradient descent

repeat until convergence

$$\theta^0 = \theta^0 - \varepsilon \frac{1}{m} \sum_i (\hat{y}_i - y_i) x_i^{(0)}$$

$$\theta^1 = \theta^1 - \varepsilon \frac{1}{m} \sum_i (\hat{y}_i - y_i) x_i^{(1)}$$

$$\theta^2 = \theta^2 - \varepsilon \frac{1}{m} \sum_i (\hat{y}_i - y_i) x_i^{(2)}$$

$$\vdots$$

$$J(\theta_1) = \frac{1}{m} \sum_i (\hat{y}_i - y_i)^2 = \frac{1}{m} \sum_i \left(\theta^T x_i - y_i\right)^2$$

Remember that $x^0=1$

Update all θ^j simultaneously

Multivariate gradient descent

Gradient descent

repeat until convergence

for j=0:n

$$\theta^{j} = \theta^{j} - \varepsilon \frac{1}{m} \sum_{i} \left(\theta^{T} x_{i} - y_{i} \right) x_{i}^{(j)}$$

$$J(\theta_1) = \frac{1}{m} \sum_{i} (\hat{y}_i - y_i)^2 = \frac{1}{m} \sum_{i} \left(\theta^{T} x_i - y_i \right)^2$$

Remember that $x^0=1$

Update all θj simultaneously

Implementing gradient descent

- For simplicity: keep a for-loop over j for the n features to estimate

$$\theta^j = \theta^j - \varepsilon \frac{1}{m} \sum_i \left(\theta^T x_i - y_i \right) x_i^{(j)}$$

- The sum over all samples x_i can be done on vectors using np.sum() and other vector operations.

Gradient descent in practice: finding the learning rate

- How do we make sure that the optimization runs correctly?
  - Make sure J decreases! Plot J as a function of the number of iterations
    - Computation of J should be vectorized

$$J(\theta_0,\theta_1)=\frac{1}{m}\sum_i(\hat{y}_i-y_i)^2=\frac{1}{m}\sum_i\left(w^T x_i+b-y_i\right)^2$$

  - If ε is too small: slow convergence
  - If ε is too large: may not decrease, may not converge
  - ε is a number between 0 and 1, often close to 0 (try 0.001,…0.01,….0.1,….1)

Solving the regression problem analytically using the normal equation

- Aggregate all the m n-dimensional training samles into a matrix X (called design matrix)

$$\mathbf{X} = \begin{bmatrix} 1 & \text{all n features for training samle 1} \\ 1 & \text{all n features for training sample 2} \\ & \vdots \\ 1 & \text{all n features for training sample m} \end{bmatrix} \quad y = \begin{bmatrix} \text{true value for training samle 1} \\ \text{true value for training sample 2} \\ \vdots \\ \text{true value for training sample m} \end{bmatrix}$$

Note the column of 1's in X

X: matrix with m rows (nof samples) and n+1 columns (mx(n+1))

X^TX will be size nxn

$$\theta = \left(X^T X\right)^{-1} X^T y$$

Comparing gradient descent with the normal equation

Gradient descent

- Need to choose ε
- Needs many iterations
- Works well even when n is large (for images of size 256x256 $n=256^2$

Normal equation

- No need to choose ε
- No iterations
- Need to compute $(X^TX)^{-1}$ (size nxn)
- Slow if n is very large
- X^TX can be non-invertible e.g. if features are linearly dependent (then use pseudo-inverse)

Gradient descent and feature scaling

- What if the features have different scale?
- x^1=size in square feet (0-2000)
- x^2=number of bedrooms (1-5)
- Draw J as a function of θ^j

- Scale the data so they have the same mean=0 and standard deviation σ=1)
- $(x^1-\mu^1)/\ \sigma^1$ (mean of feature 1 over all samples in data set).
- $(x^2-\mu^2)/\ \sigma^2$

Some statistics beyond the least squares loss function

- Statistician will derive the least square loss function based on the maximum likelihood principle.
- Here is a very short introduction to how:
- Assume the measurements y_i are random variables related to x_i as:

$$y_i = \theta^T x_i + \eta_i$$

- η_i is a noise term, Gaussian noise with zero mean and variance σ^2

$$p(\eta_i) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{\eta_i^2}{2\sigma^2}\right)$$

- The yi's will then have the conditional distribution

$$p(y_i \mid x_i, \theta) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(y_i - \theta^T x_i)^2}{2\sigma^2}\right)$$

Some statistics beyond the least squares loss function

- Given m samples, how likely is a certain value of θ?
- This is studied in terms of the likelihood function L(θ,X,y) given the training data set.

$$L(\theta) = L(\theta, X, y) = p(y \mid X, \theta)$$

How likely is it that we observe y for a given value θ of and the data X.

- When the noise η_i is independent from sample to sample, we get

$$L(\theta) = \prod_{i=1}^{m} p(y_i \mid x_i, \theta) = \prod_{i=1}^{m} \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(y_i - \theta^T x_i)^2}{2\sigma^2} \right)$$

- The «best guess» of θ is the value of θ that maximize the likehoodfunction L(θ)
- Often it is easier to optimize the logarithm of the likelihood, called log-likelihoood
- It can be shown that maximizing L is equivalent to minimizing the MSE loss function.

The linear regression problem, summary

Hypothesis: $h(\theta) = \hat{y} = \theta^T x$

Parameters: $\theta^j,\ j = 0..n$

Cost function: $J(\theta^0) = \frac{1}{2m}\sum_{i=1}^{m}\left(\hat{y}_i - y_i\right)^2$

Goal: $\underset{\theta^0}{\text{minimize}}\, J(\theta)$

Gradient descent solution:
repeat until convergence
 for j=0:n

$$\theta^j = \theta^j - \varepsilon \frac{\partial}{\partial \theta^j} J(\theta_1, \theta_2)$$

Summary continued

- Take care to find a good value of the learning rate!
  - Visualize J as a function of iterations

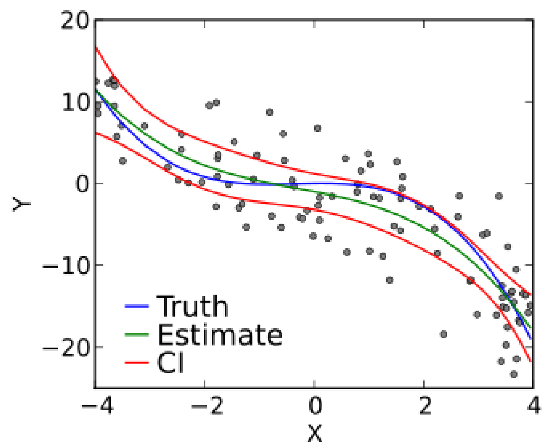
- Consider feature scaling if the range of the features are different

Polynomial regression

- If a linear model is not sufficient, we can extend to allow higher-order terms or cross-terms between the variables by changing our hypothesis $h_\theta(x)$

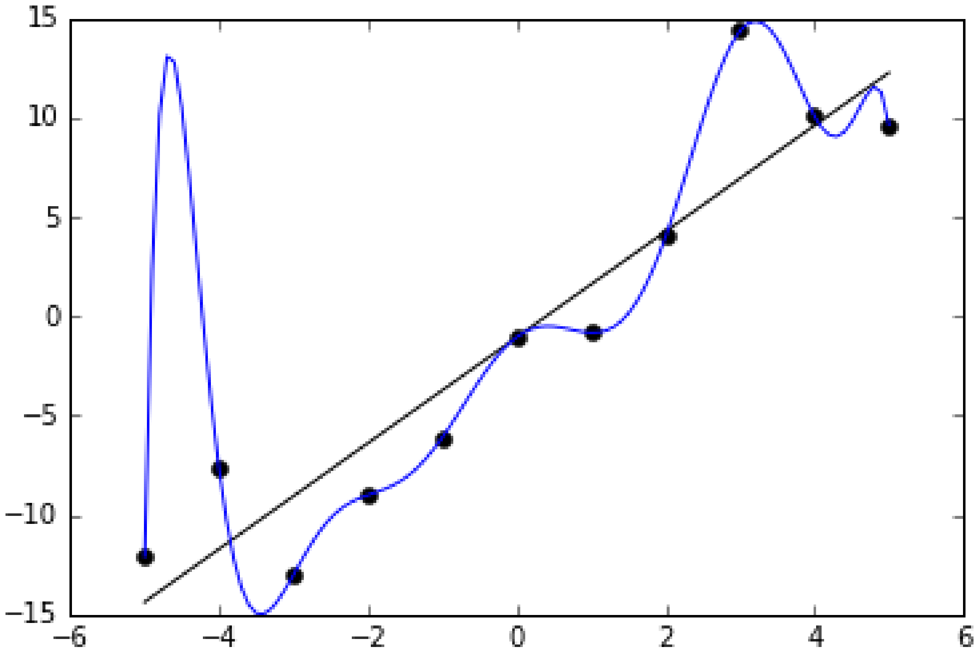
$$h_\theta(x) = \theta^0 + \theta^1 x^1 + \theta^2 (x^1)^2 + \theta^3 (x^1)^3 \ldots$$

$$h_\theta(x) = \theta^0 + \theta^1 x^1 + \theta^2 \sqrt{x^1}$$



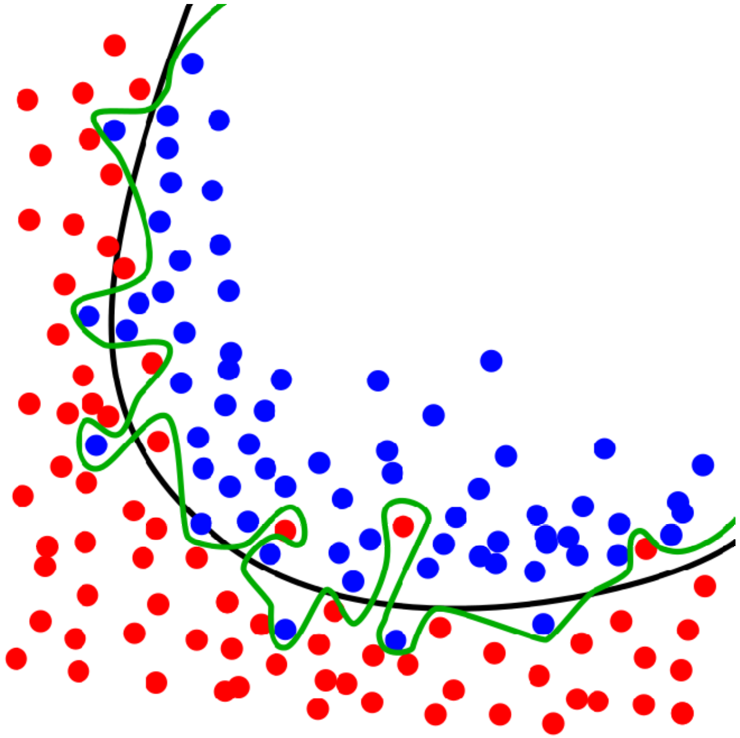
The danger of overfitting

A higher-order model can easily overfit the training data



Overfitting for classification

- Overfitting must be avoided for classifiation also – this is partly why we start with simple linear models



Learning goals – linear regression

- Be able to set up the problem:
  - Hypothesis, parameters, cost function, goal
- Understand gradient descent for this problem
- From exercises:
  - Be able to solve by hand simple problems
  - Implement gradient descent to solve the linear regression problem.
- Know the practical details about feature scaling and setting the learning rate.

Next two weeks:

- Next week: The challenge of generalization
  - The art of not overfitting to training data in general
- In two week we continue with:
  - From regression to classification
  - Logistic regression
    - Regression to solve a 2-class classification problem.
  - Generalizing to K classes
    - Softmax
    - Support vector machine classifiers
  - Reading material
    - http://cs231n.github.io/classification/
    - http://cs229.stanford.edu/notes/cs229-notes1.pdf