CS7267 MACHINE LEARNING

OPTIMIZATION

* This lecture is based on Kyle Andelin's slides

Mingon Kang, Ph.D.
Computer Science, Kennesaw State University

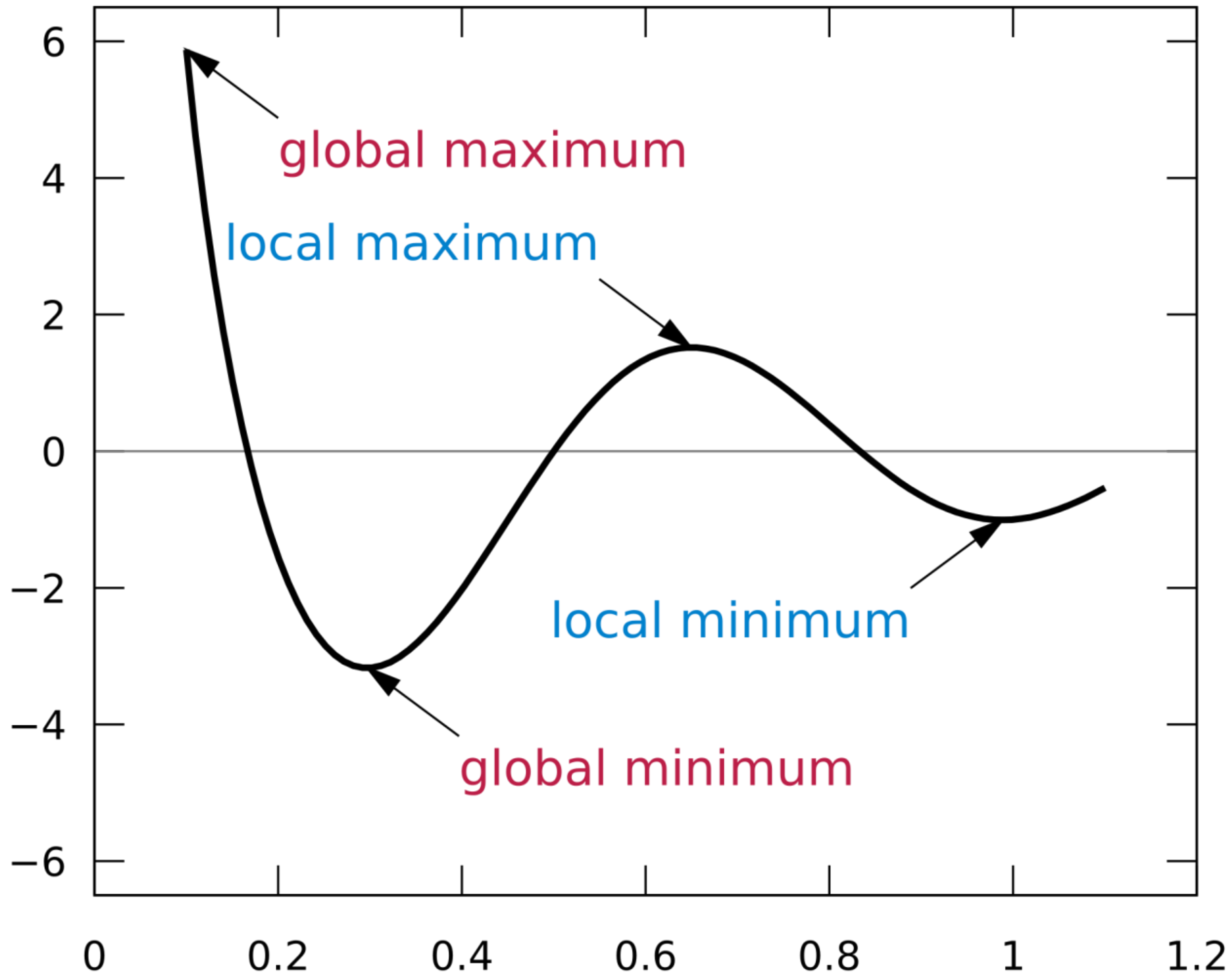
Optimization

- Consider a function $f(.)$ of p numbers of variables:

$$y = f\left(x_1, x_2, \dots, x_p\right)$$

- Find $x_1, x_2, \dots, x_p$ that maximizes or minimizes y
- Usually, minimize a cost/loss function or maximize profit/likelihood function.

Global/Local Optimization



Gradient

- Single variable:
  - The derivative: slope of the tangent line at a point x_0

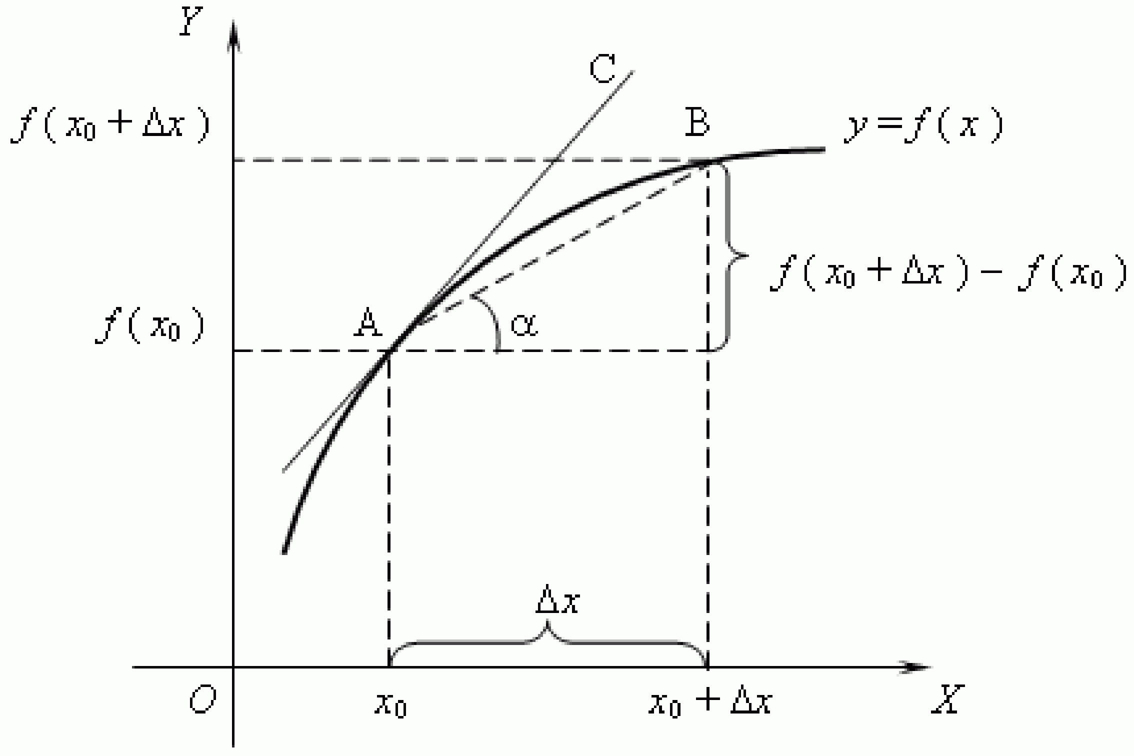
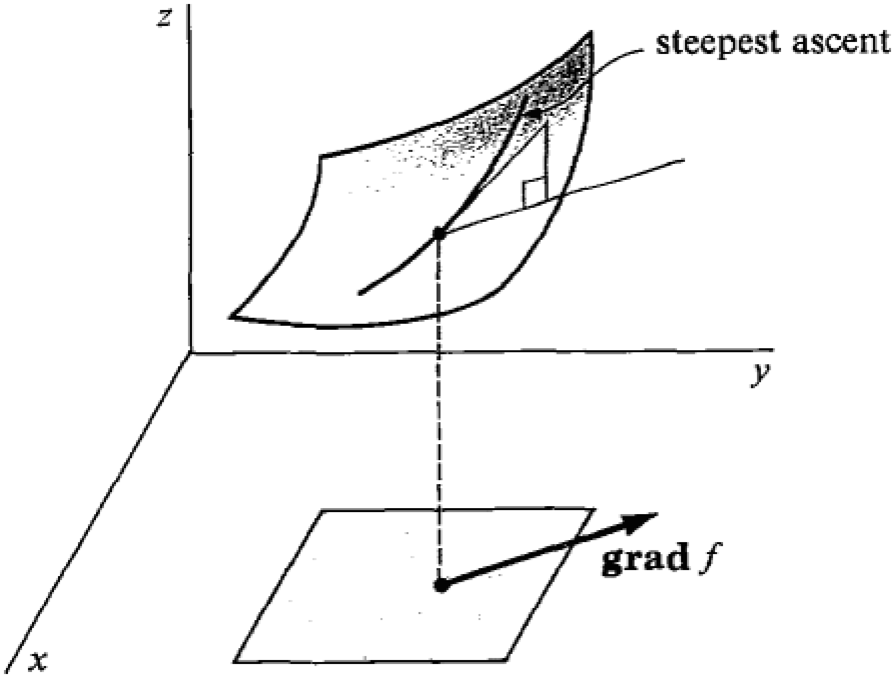


Fig. 1

Gradient

- Multivariable:
  - $\nabla f = \left\langle \frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \ldots, \frac{\partial f}{\partial x_n} \right\rangle$
  - A vector of partial derivatives with respect to each of the independent variables
- ∇f points in the direction of greatest rate of change or "steepest ascent"
- Magnitude (or length) of ∇f is the greatest rate of change

Gradient



The general idea

- We have k parameters $\theta_1, \theta_2, \dots, \theta_k$ we'd like to train for a model – with respect to some error/loss function $J(\theta_1, \dots, \theta_k)$ to be minimized
- Gradient descent is one way to iteratively determine the optimal set of parameter values:
  1. Initialize parameters
  2. Keep changing values to reduce $J(\theta_1, \dots, \theta_k)$
  - ∇J tells us which direction increases J the most
  - We go in the opposite direction of ∇J

To actually descend…

Set initial parameter values $\theta_1^0, \dots, \theta_k^0$

while(not converged) {

calculate ∇J (i.e. evaluate $\frac{\partial J}{\partial \theta_1}, \dots, \frac{\partial J}{\partial \theta_k}$)

do {

$$\theta_1 := \theta_1 - \alpha \frac{\partial J}{\partial \theta_1}$$

$$\theta_2 := \theta_2 - \alpha \frac{\partial J}{\partial \theta_2}$$

$$\vdots$$

$$\theta_k := \theta_k - \alpha \frac{\partial J}{\partial \theta_k}$$

}

}

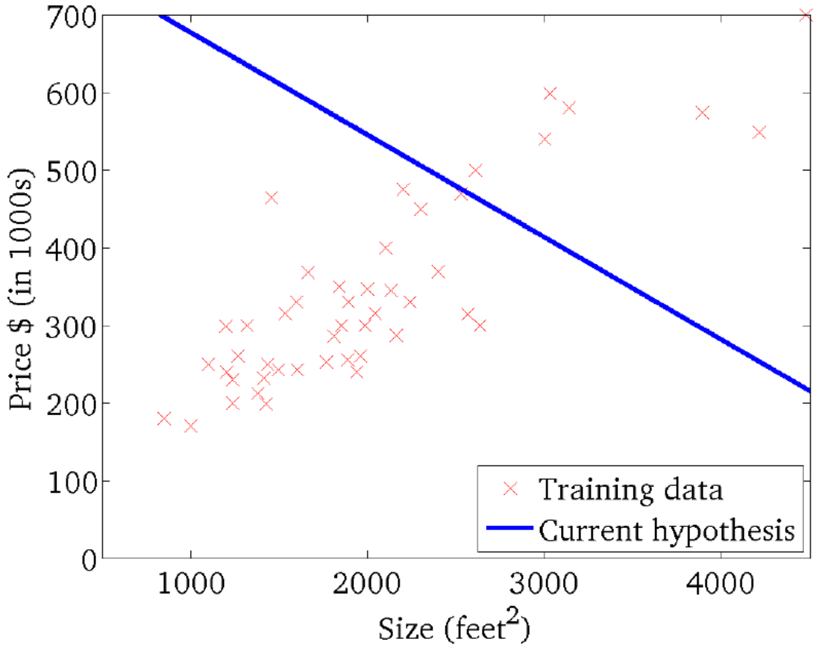
Where α is the 'learning rate' or 'step size'

- Small enough α ensures $J\left(\theta_1^i, \dots, \theta_k^i\right) \leq J(\theta_1^{i-1}, \dots, \theta_k^{i-1})$

After each iteration:

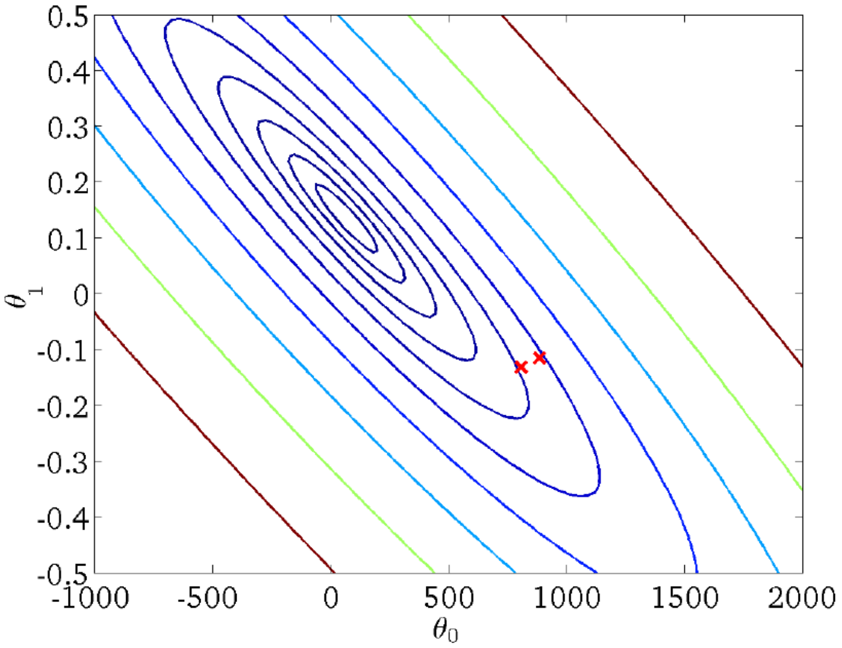
$h_\theta(x)$

(for fixed θ_0, θ_1, this is a function of x)



$J(\theta_0, \theta_1)$

(function of the parameters θ_0, θ_1)

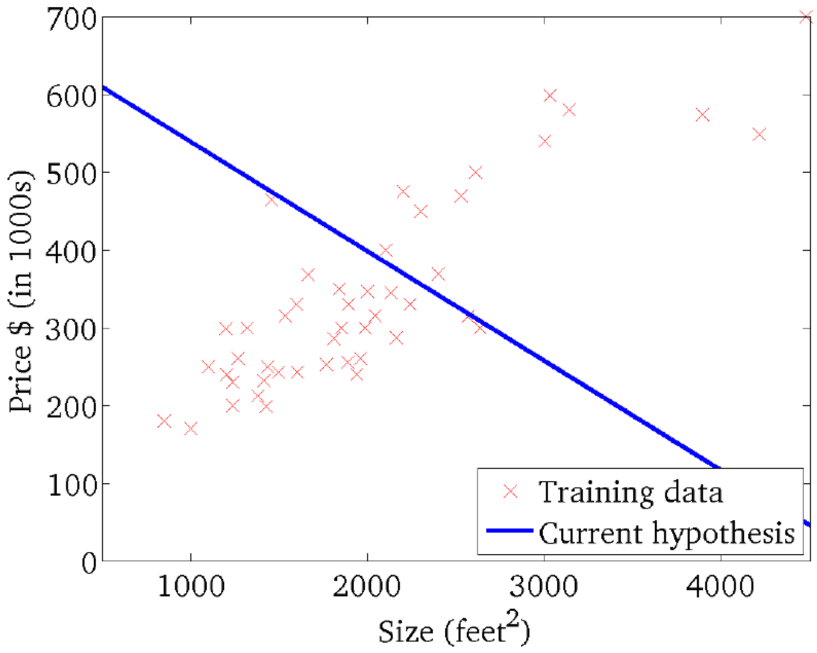


Picture credit: Andrew Ng, Stanford University, Coursera Machine Learning, Lecture 2 Slides

After each iteration:

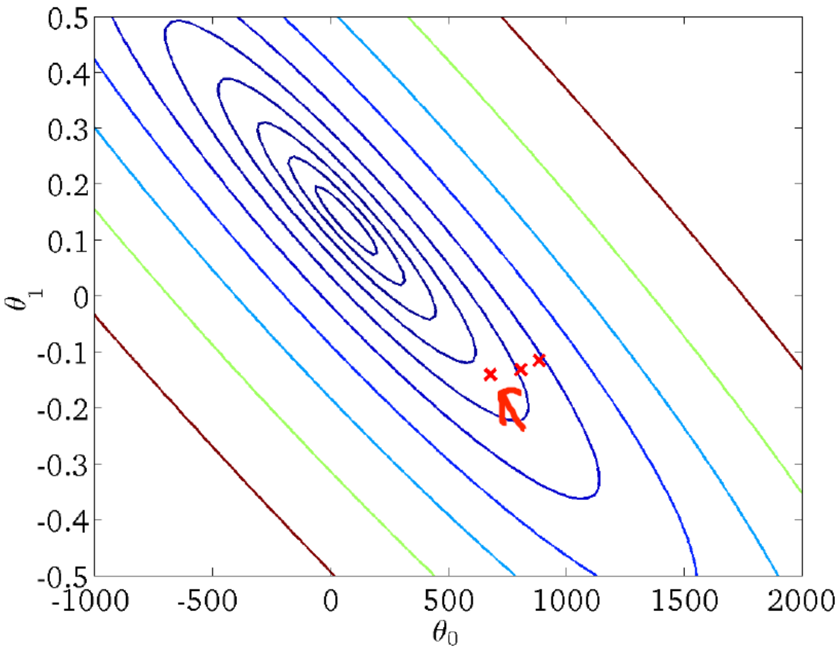
$$h_\theta(x)$$

(for fixed θ_0, θ_1, this is a function of x)



$$J(\theta_0, \theta_1)$$

(function of the parameters θ_0, θ_1)

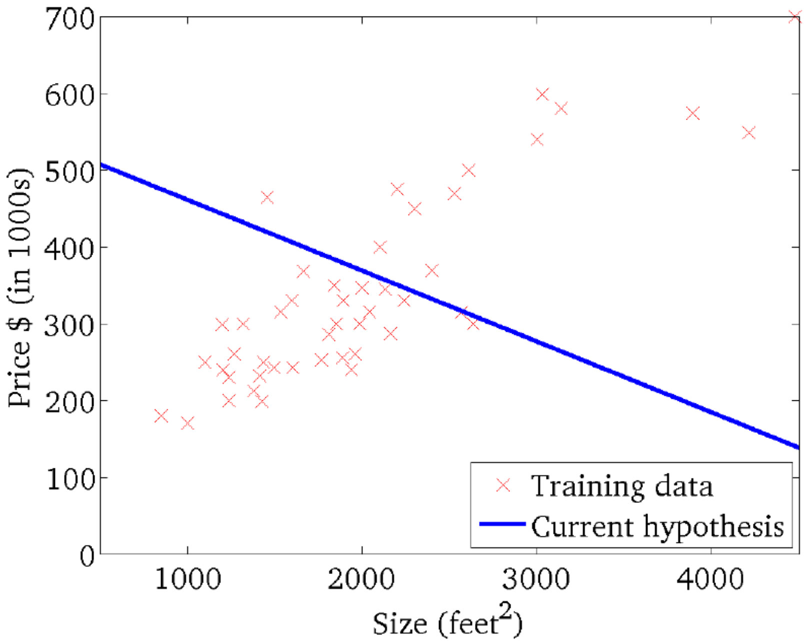


Picture credit: Andrew Ng, Stanford University, Coursera Machine Learning, Lecture 2 Slides

After each iteration:

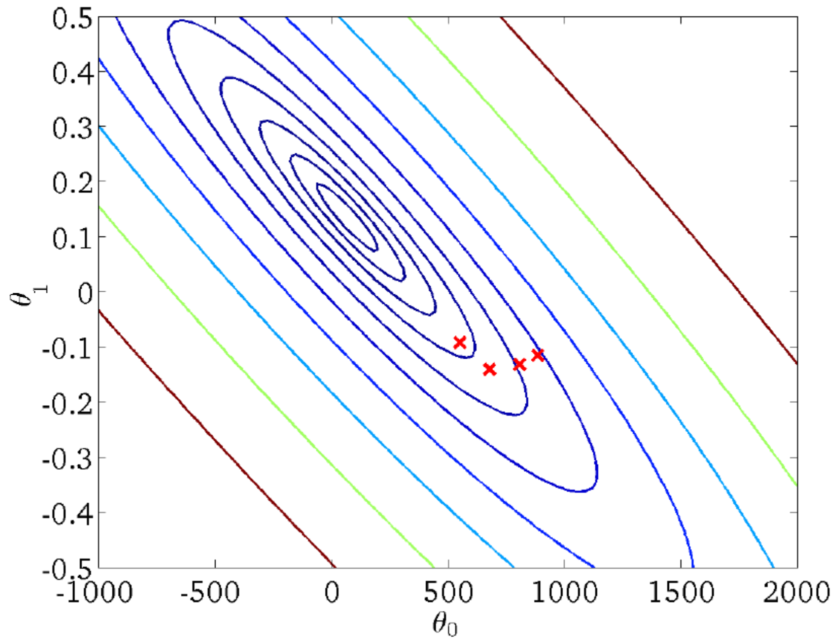
$$h_\theta(x)$$

(for fixed θ_0, θ_1, this is a function of x)



$$J(\theta_0, \theta_1)$$

(function of the parameters θ_0, θ_1)

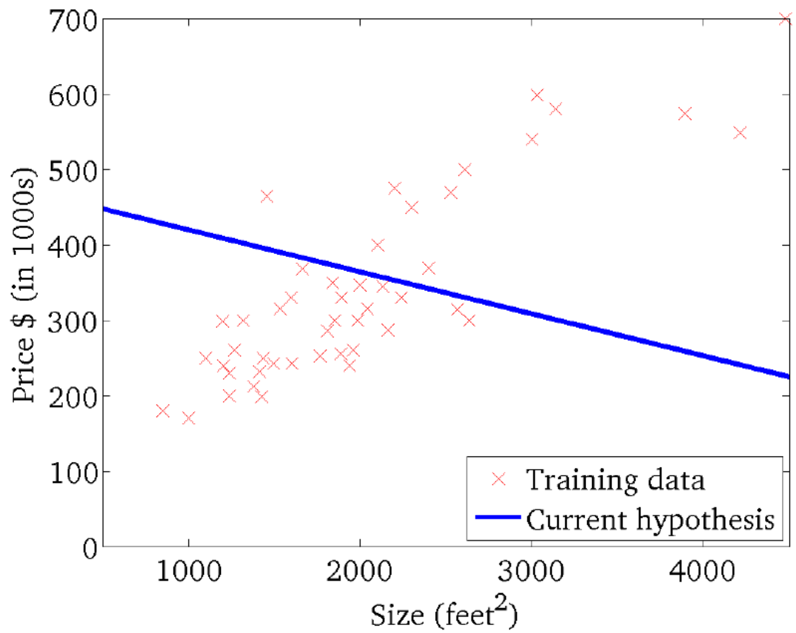


Picture credit: Andrew Ng, Stanford University, Coursera Machine Learning, Lecture 2 Slides

After each iteration:

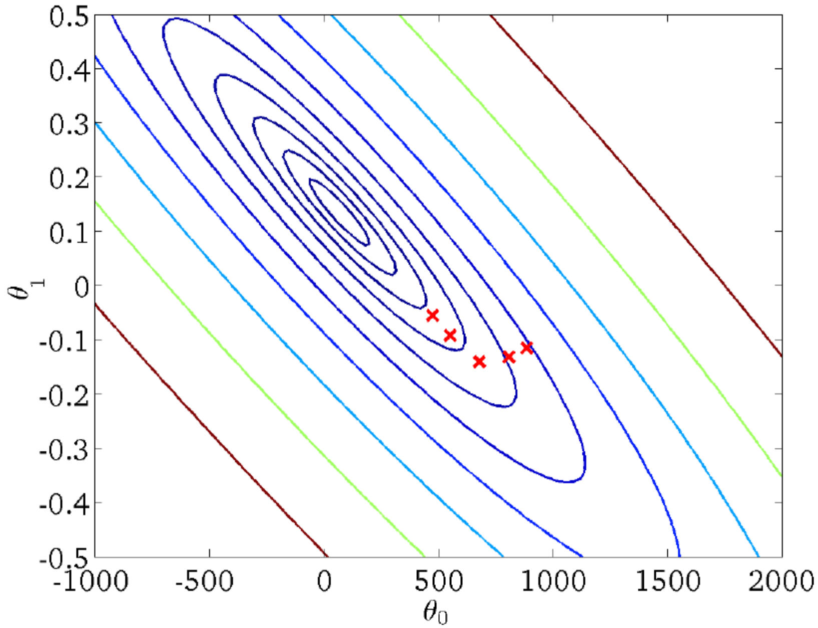
$h_\theta(x)$

(for fixed θ_0, θ_1, this is a function of x)



$J(\theta_0, \theta_1)$

(function of the parameters θ_0, θ_1)

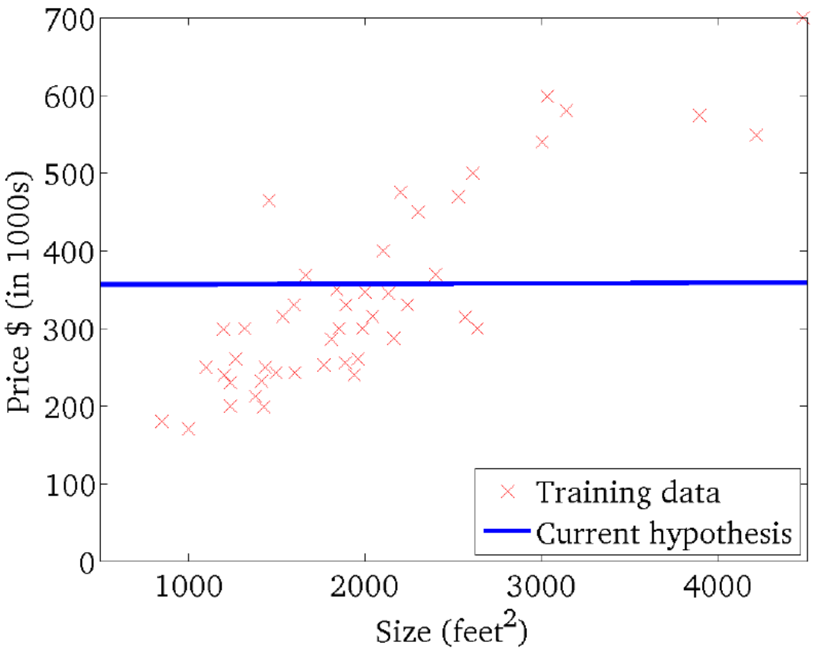


Picture credit: Andrew Ng, Stanford University, Coursera Machine Learning, Lecture 2 Slides

After each iteration:

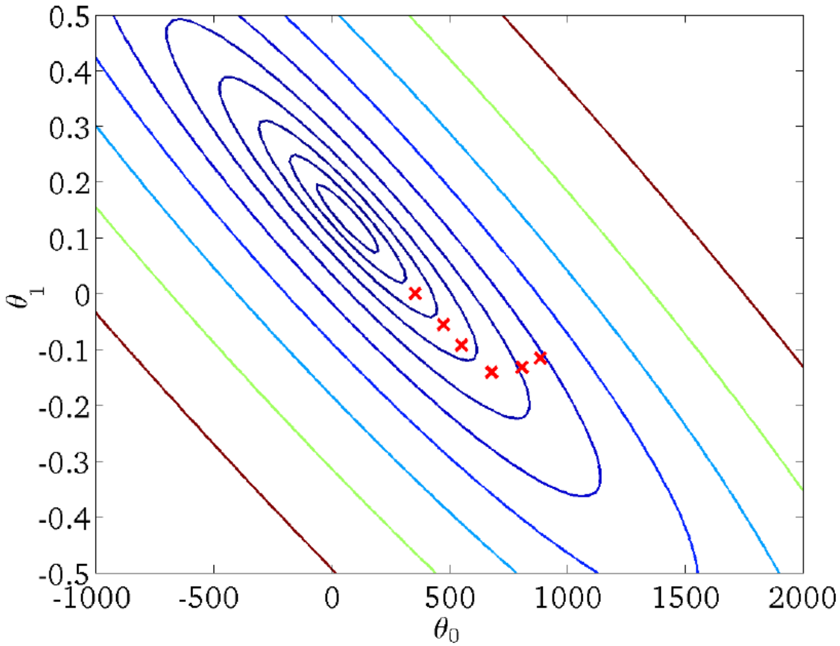
$h_\theta(x)$

(for fixed θ_0, θ_1, this is a function of x)



$J(\theta_0, \theta_1)$

(function of the parameters θ_0, θ_1)

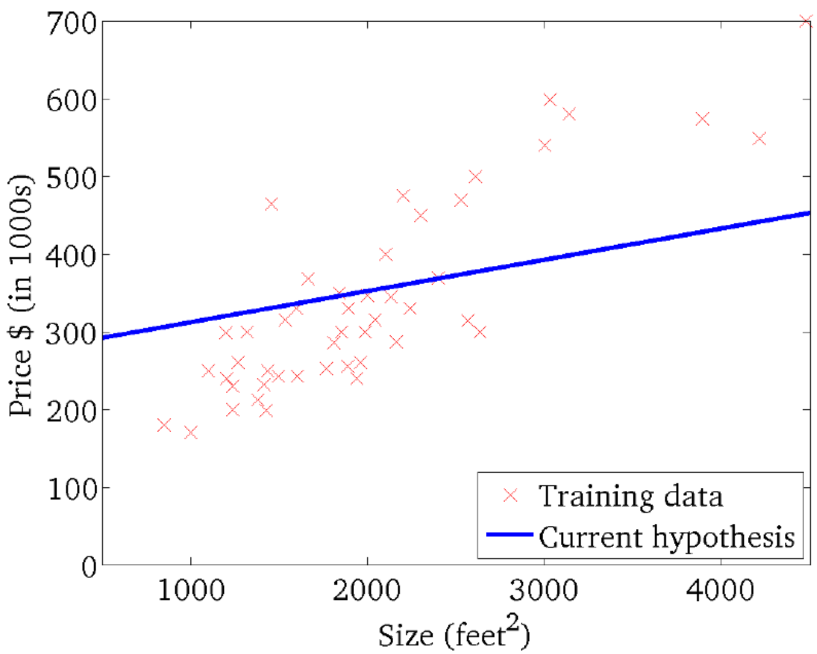


Picture credit: Andrew Ng, Stanford University, Coursera Machine Learning, Lecture 2 Slides

After each iteration:

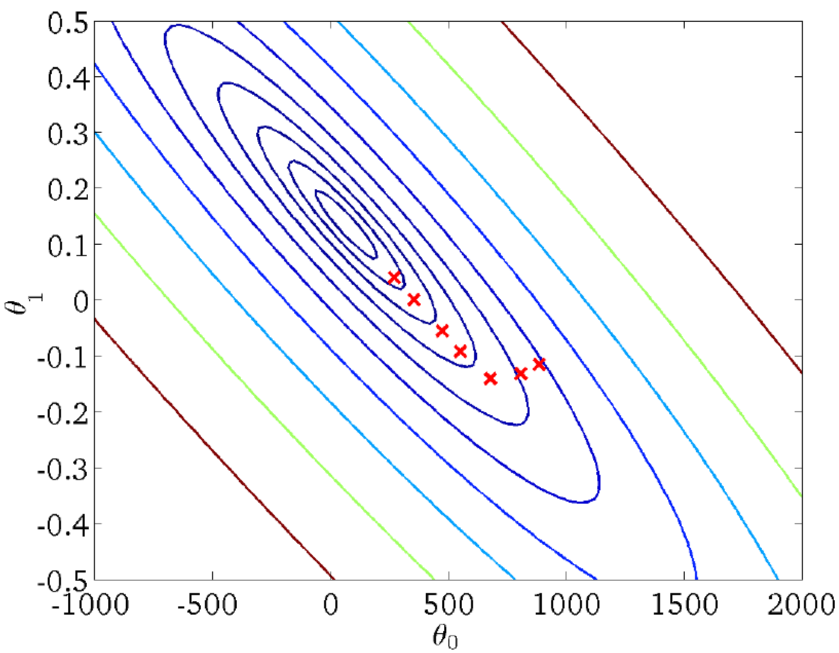
$h_\theta(x)$

(for fixed θ_0, θ_1, this is a function of x)



$J(\theta_0, \theta_1)$

(function of the parameters θ_0, θ_1)

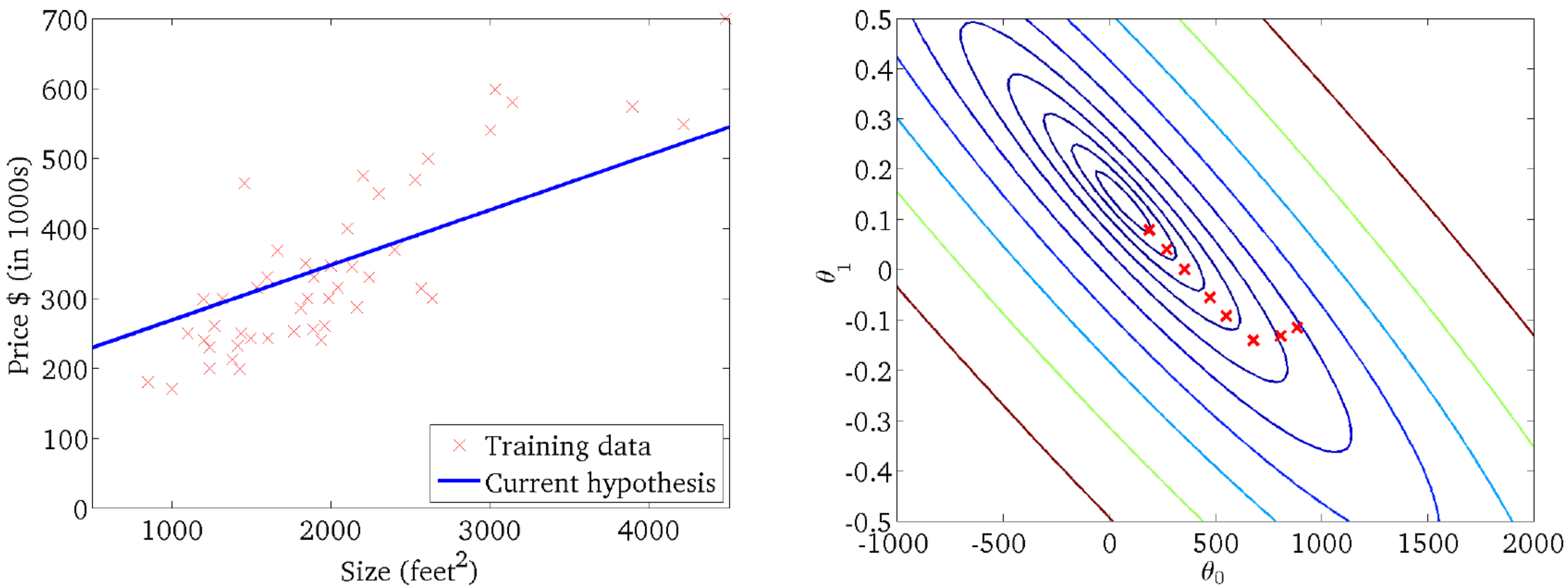


Picture credit: Andrew Ng, Stanford University, Coursera Machine Learning, Lecture 2 Slides

After each iteration:

$h_\theta(x)$ | $J(\theta_0, \theta_1)$

(for fixed θ_0, θ_1, this is a function of x) | (function of the parameters θ_0, θ_1)



Picture credit: Andrew Ng, Stanford University, Coursera Machine Learning, Lecture 2 Slides

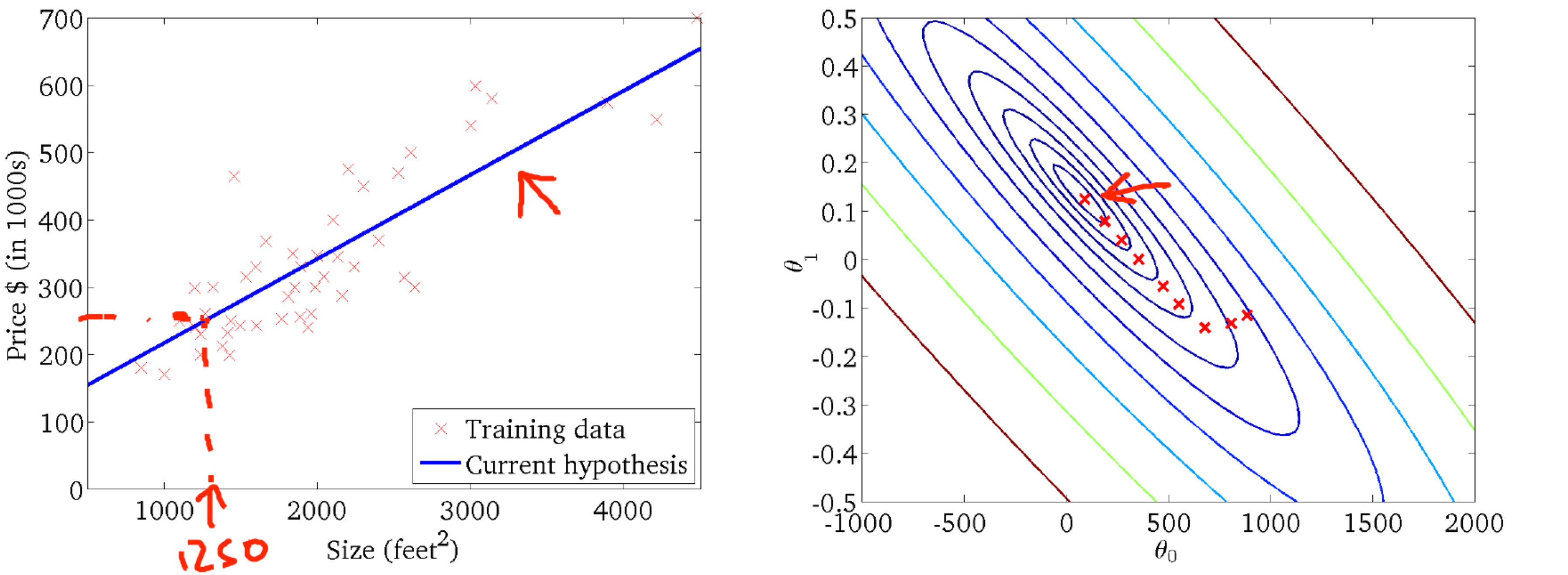
After each iteration:

$h_\theta(x)$

(for fixed θ_0, θ_1, this is a function of x)

$J(\theta_0, \theta_1)$

(function of the parameters θ_0, θ_1)

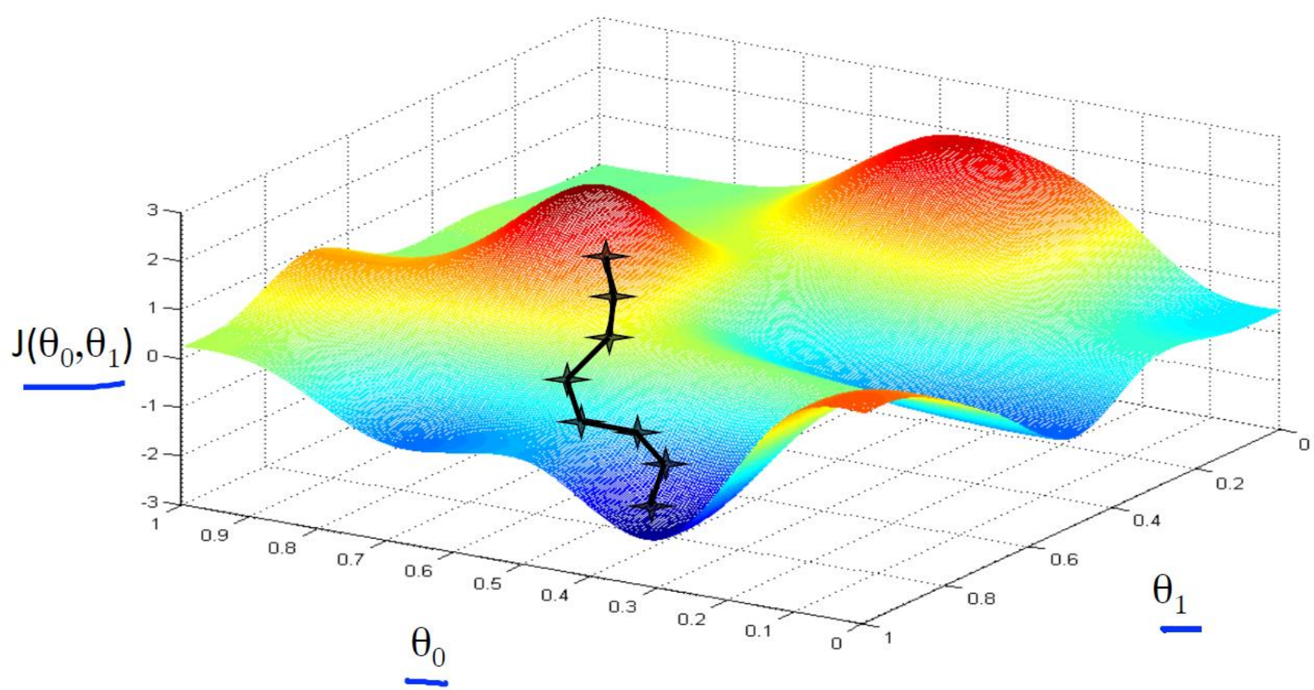


Picture credit: Andrew Ng, Stanford University, Coursera Machine Learning, Lecture 2 Slides

Issues

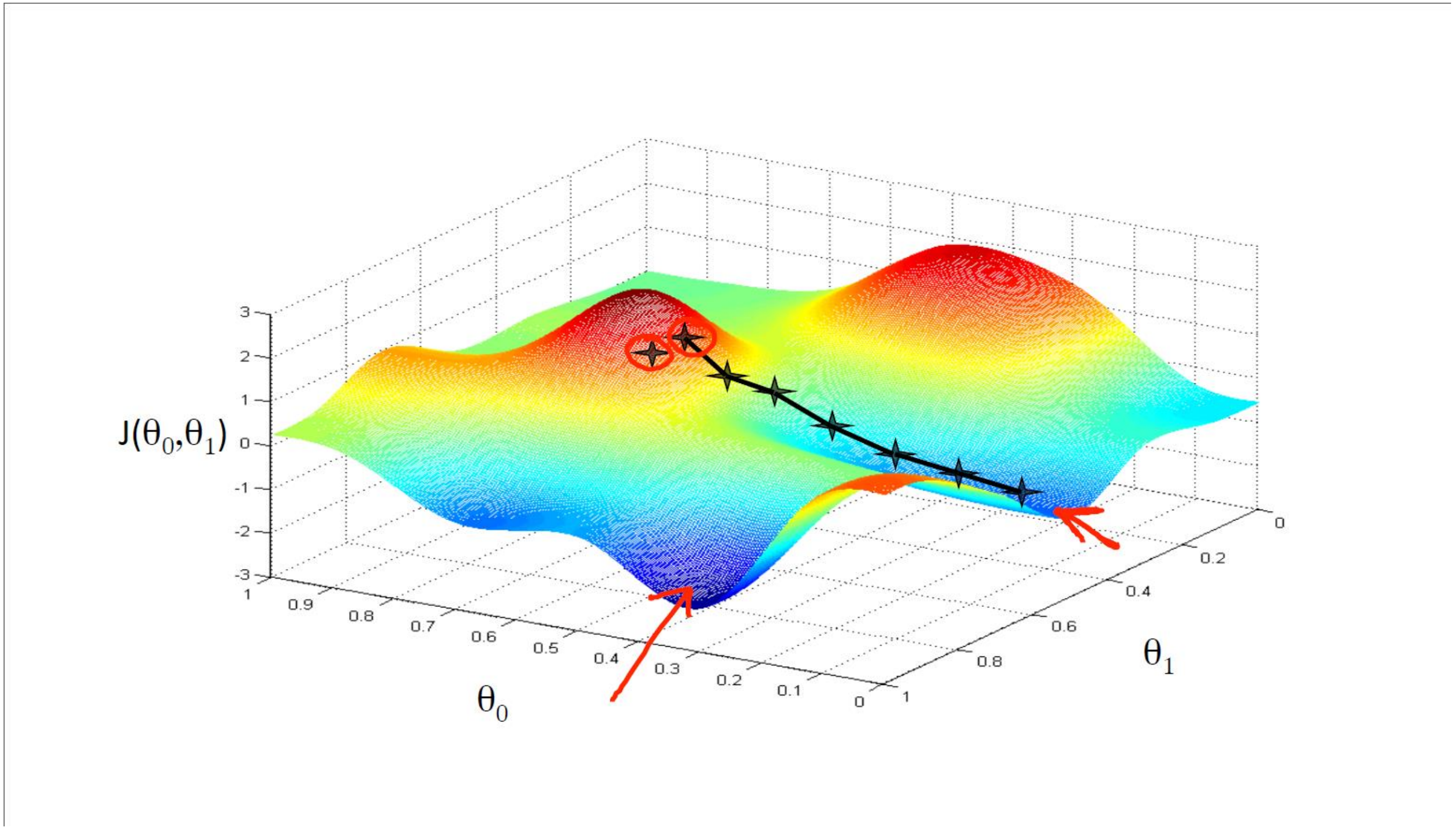
- Convex objective function guarantees convergence to global minimum
- Non-convexity brings the possibility of getting stuck in a local minimum
  - Different, randomized starting values can fight this

Initial Values and Convergence



Picture credit: Andrew Ng, Stanford University, Coursera Machine Learning, Lecture 2 Slides

Initial Values and Convergence



Picture credit: Andrew Ng, Stanford University, Coursera Machine Learning, Lecture 2 Slides

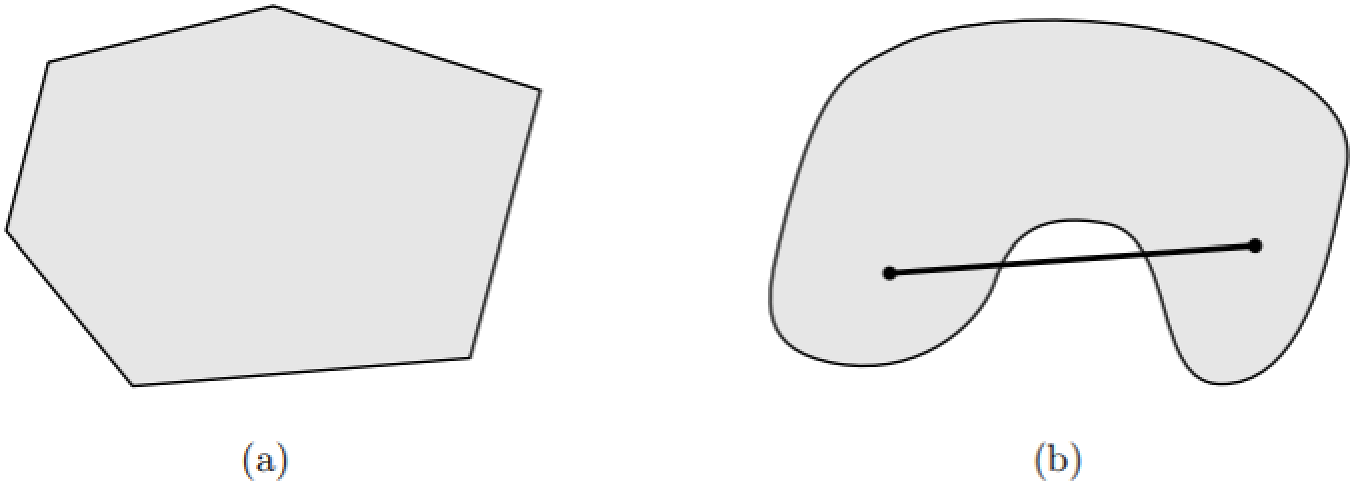
Issues cont.

- Convergence can be slow
  - Larger learning rate α can speed things up, but with too large of α, optimums can be 'jumped' or skipped over - requiring more iterations
  - Too small of a step size will keep convergence slow
  - Can be combined with a *line search* to find the optimal α on every iteration

Convex set

- Definition
  - A set C is convex if, for any $x, y \in C$ and $\theta \in \Re$ with $0 \leq \theta \leq 1, \theta x + (1-\theta)y \in C$
  - If we take any two elements in C, and draw a line segment between these two elements, then every point on that line segment also belongs to C

Convex set

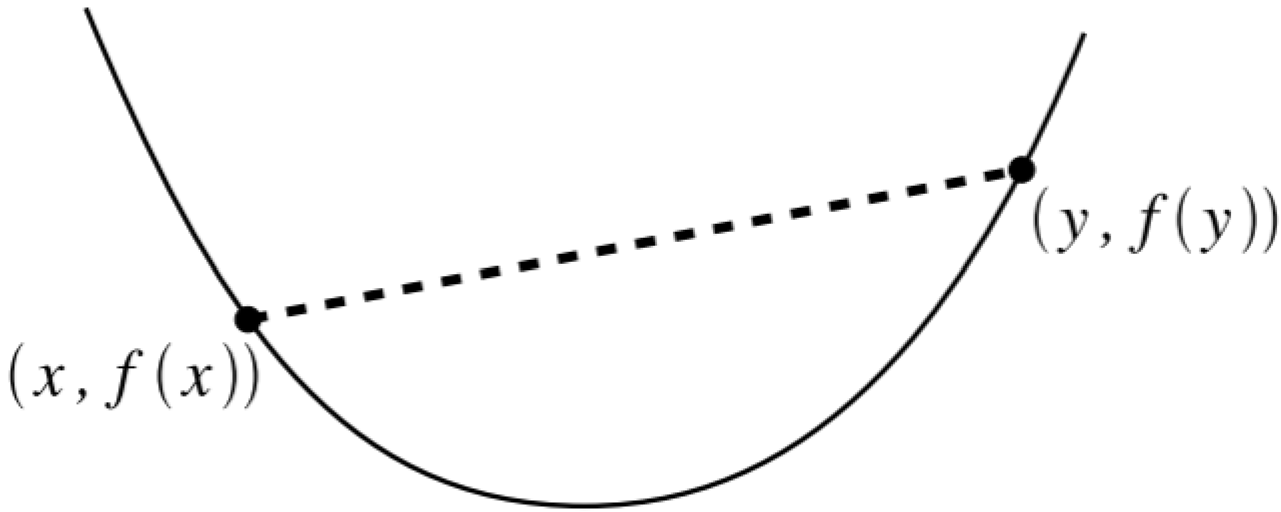


Examples of a convex set (a) and a non-convex set

Convex functions

- Definition
  - A function $f: \Re_n \rightarrow \Re$ is convex if its domain (denoted $D(f)$) is a convex set, and if, for all $x, y \ \in D(f)$ and $\theta \in R, 0 \leq \theta \leq 1, f(\theta x + (1-\theta)y) \leq \theta f(x) + (1-\theta)f(y)$.
  - If we pick any two points on the graph of a convex function and draw a straight line between them, the n the portion of the function between these two points will lie below this straight line

Convex function

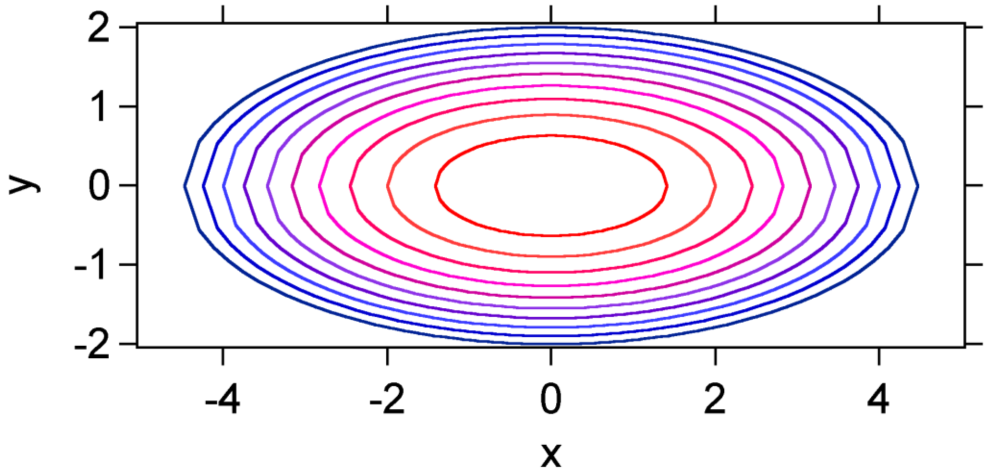


The line connecting two points on the graph must lie above the function

Steepest Descent Method

$$f(x_1, x_2) = x_1^2 + 5x_2^2$$

- Contours are shown below



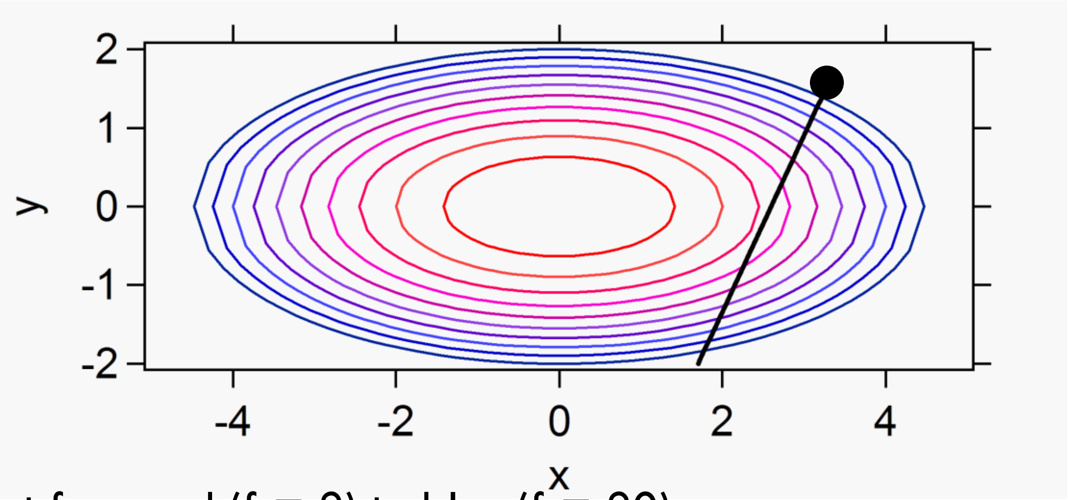
Steepest Descent

The gradient at the point (x_1^1, x_2^1) is

$$\nabla f(x_1^1, x_2^1) = (2x_1^1,\ 10x_2^1)^T$$

If we choose $x_1^1 = 3.22$, $x_2^1 = 1.39$ as the starting point represented by the black dot on the figure, the black line shown in the figure represents the direction for a line search.

$$-\nabla f(x_1^1, x_2^1) = (-6.44, -13.9)^T$$



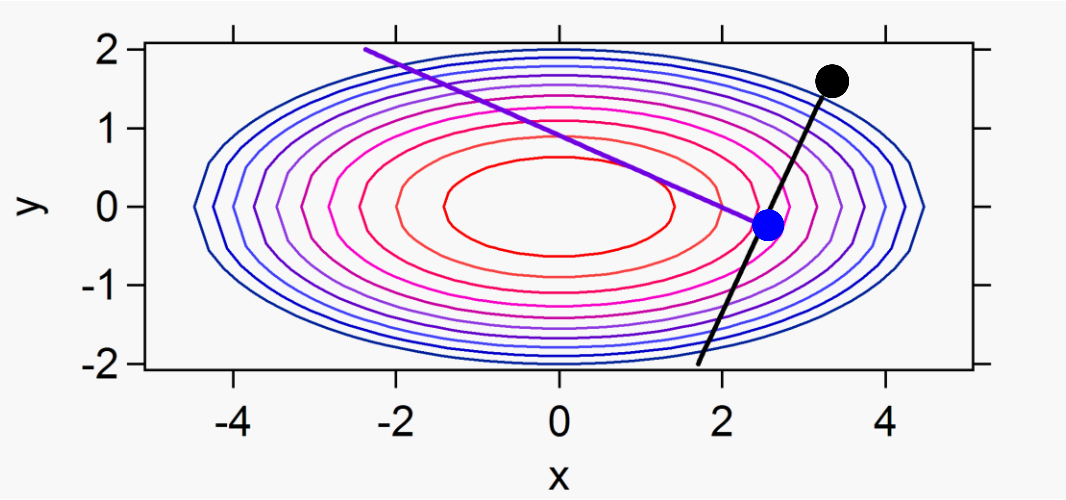
Contours represent from red (f = 2) to blue (f = 20).

Steepest Descent

Now, the question is how big should the step be along the direction of the gradient? We want to find the minimum along the line before taking the next step.

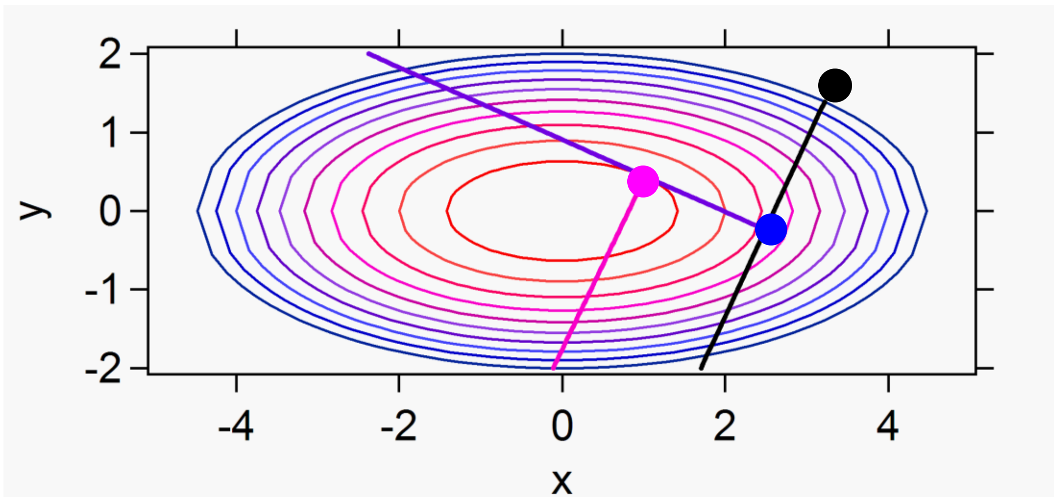
The minimum along the line corresponds to the point where the new direction is orthogonal to the original direction.

The new point is $(x_1^2, x_2^2) = (2.47,-0.23)$ shown in blue .



Steepest Descent

By the third iteration we can see that from the point (x_1^2, x_2^2) the new vector again misses the minimum, and here it seems that we could do better because we are close.



Steepest descent is usually used as the first technique in a minimization procedure, however, a robust strategy that improves the choice of the new direction will greatly enhance the efficiency of the search for the minimum.

Numerical Optimization

- Numerical Optimization
  - Steepest descent
  - Newton Method
  - Gauss–Newton algorithm
  - Levenberg–Marquardt algorithm
  - Line Search Methods
  - Trust-Region Methods

See: http://home.agh.edu.pl/~pba/pdfdoc/Numerical_Optimization.pdf