

# 10-701 Introduction to Machine Learning

---

## Linear Regression

Readings:  
Bishop, 3.1  
Murphy, 7

Matt Gormley  
Lecture 4  
September 19, 2016

# Reminders

- Homework 1:
  - due 9/26/16
- Project Proposal:
  - due 10/3/16
  - start early!

# Outline

- **Linear Regression**
  - Simple example
  - Model
- **Learning**
  - Gradient Descent
  - SGD
  - Closed Form
- **Advanced Topics**
  - Geometric and Probabilistic Interpretation of LMS
  - L<sub>2</sub> Regularization
  - L<sub>1</sub> Regularization
  - Features
  - Locally-Weighted Linear Regression
  - Robust Regression

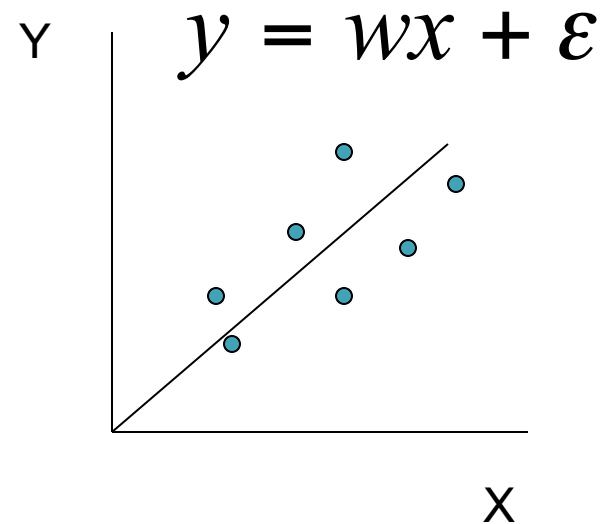
# Outline

- **Linear Regression**
  - Simple example
  - Model
- **Learning (aka. Least Squares)**
  - Gradient Descent
  - SGD (aka. Least Mean Squares (LMS))
  - Closed Form (aka. Normal Equations)
- **Advanced Topics**
  - Geometric and Probabilistic Interpretation of LMS
  - L<sub>2</sub> Regularization (aka. Ridge Regression)
  - L<sub>1</sub> Regularization (aka. LASSO)
  - Features (aka. non-linear basis functions)
  - Locally-Weighted Linear Regression
  - Robust Regression

# Linear regression in 1D

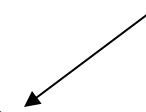
- Our goal is to estimate  $w$  from a training data of  $\langle x_i, y_i \rangle$  pairs
- Optimization goal: minimize squared error (least squares):

$$\arg \min_w \sum_i (y_i - wx_i)^2$$



- Why least squares?
  - minimizes squared distance between measurements and predicted line
  - has a nice probabilistic interpretation
  - the math is pretty

see HW



# Solving Linear Regression in 1D

- To optimize – closed form:
- We just take the derivative w.r.t. to  $w$  and set to 0:

$$\frac{\partial}{\partial w} \sum_i (y_i - wx_i)^2 = 2 \sum_i -x_i(y_i - wx_i) \Rightarrow$$

$$2 \sum_i x_i(y_i - wx_i) = 0 \Rightarrow 2 \sum_i x_i y_i - 2 \sum_i w x_i x_i = 0$$

$$\sum_i x_i y_i = \sum_i w x_i^2 \Rightarrow$$

$$w = \frac{\sum_i x_i y_i}{\sum_i x_i^2}$$

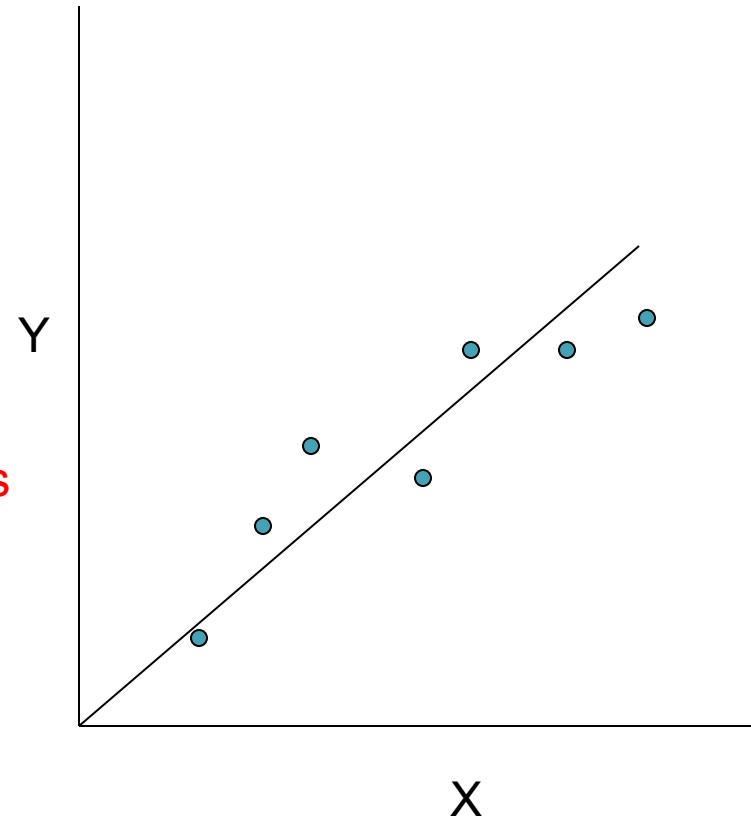
# Linear regression in 1D

- Given an input  $x$  we would like to compute an output  $y$
- In linear regression we assume that  $y$  and  $x$  are related with the following equation:

What we are trying to predict

Observed values

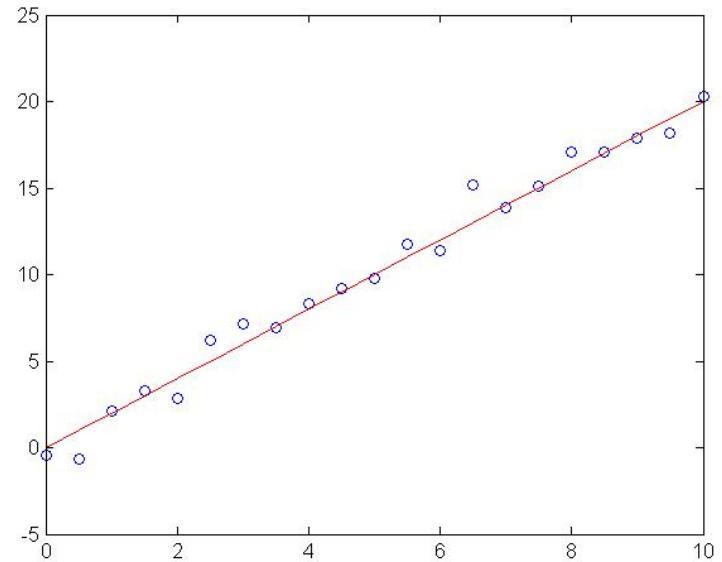
$$y = wx + \varepsilon$$



where  $w$  is a parameter and  $\varepsilon$  represents measurement error or other noise

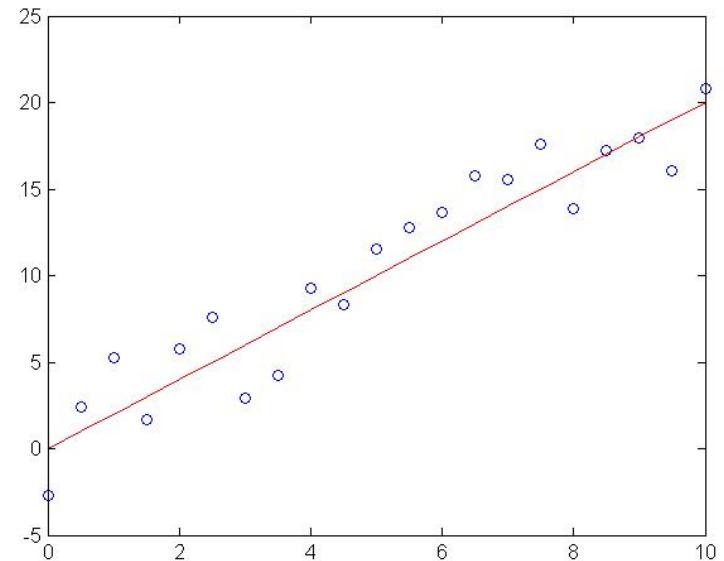
# Regression example

- Generated:  $w=2$
- Recovered:  $w=2.03$
- Noise:  $std=1$



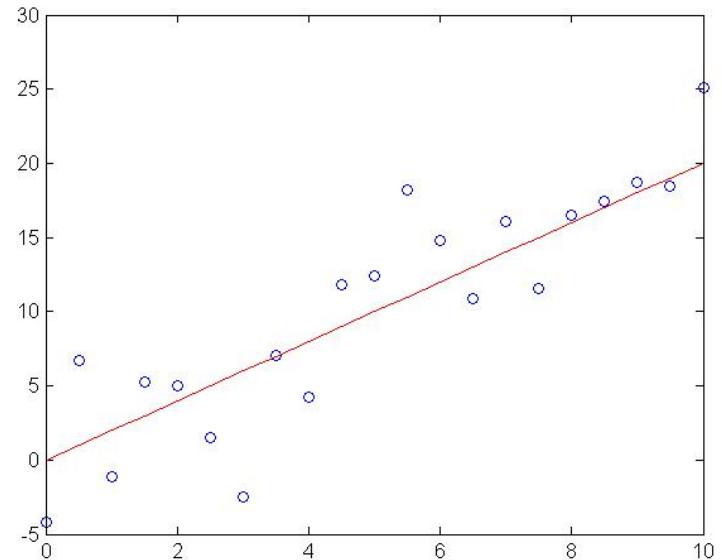
# Regression example

- Generated:  $w=2$
- Recovered:  $w=2.05$
- Noise:  $std=2$



# Regression example

- Generated:  $w=2$
- Recovered:  $w=2.08$
- Noise:  $\text{std}=4$



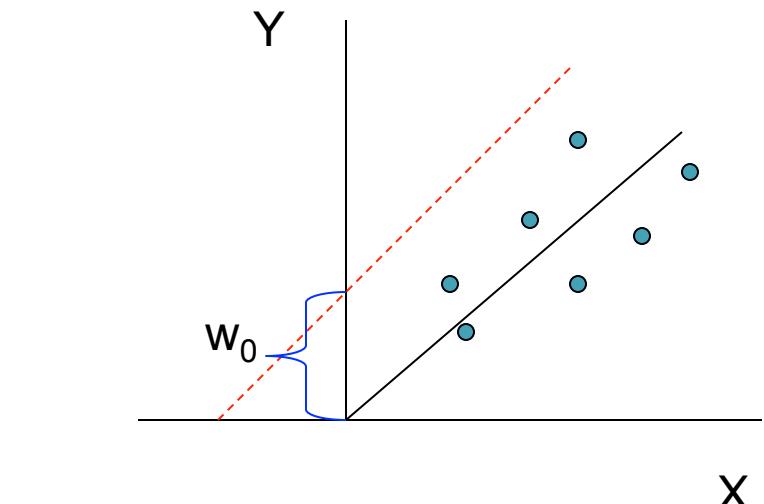
# Bias term

- So far we assumed that the line passes through the origin
- What if the line does not?
- No problem, simply change the model to

$$y = w_0 + w_1 x + \varepsilon$$

- Can use least squares to determine  $w_0$ ,  $w_1$

$$w_0 = \frac{\sum_i y_i - w_1 x_i}{n}$$



$$w_1 = \frac{\sum_i x_i (y_i - w_0)}{\sum_i x_i^2}$$

# Linear Regression

**Data:** Inputs are continuous vectors of length  $K$ . Outputs are continuous scalars.

$$\mathcal{D} = \{\mathbf{x}^{(i)}, y^{(i)}\}_{i=1}^N \text{ where } \mathbf{x} \in \mathbb{R}^K \text{ and } y \in \mathbb{R}$$



What are some  
example problems of  
this form?

# Linear Regression

**Data:** Inputs are continuous vectors of length K. Outputs are continuous scalars.

$$\mathcal{D} = \{\mathbf{x}^{(i)}, y^{(i)}\}_{i=1}^N \text{ where } \mathbf{x} \in \mathbb{R}^K \text{ and } y \in \mathbb{R}$$

**Prediction:** Output is a linear function of the inputs.

$$\hat{y} = h_{\theta}(\mathbf{x}) = \theta_1 x_1 + \theta_2 x_2 + \dots + \theta_K x_K$$

$$\hat{y} = h_{\theta}(\mathbf{x}) = \boldsymbol{\theta}^T \mathbf{x}$$

(We assume  $x_1$  is 1)

**Learning:** finds the parameters that minimize some objective function.

$$\boldsymbol{\theta}^* = \operatorname{argmin}_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$$

# Least Squares

**Learning:** finds the parameters that minimize some objective function.

$$\boldsymbol{\theta}^* = \operatorname{argmin}_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$$

We minimize the sum of the squares:

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

Why?

1. Reduces distance between true measurements and predicted hyperplane (line in 1D)
2. Has a nice probabilistic interpretation

# Least Squares

**Learning:** finds the parameters that minimize some objective function.

$$\theta^* = \operatorname{argmin}_{\theta} J(\theta)$$

We minimize the sum of the squares:

This is a very general optimization setup.

We could solve it in lots of ways. Today, we'll consider three ways.

$$\sum (\theta^T \mathbf{x}^{(i)} - y^{(i)})^2$$

1

between true measurements and line in 1D)  
interpretation

# Least Squares

**Learning:** Three approaches to solving  $\theta^* = \operatorname{argmin}_{\theta} J(\theta)$

**Approach 1:** Gradient Descent

(take larger – more certain – steps opposite the gradient)

**Approach 2:** Stochastic Gradient Descent (SGD)

(take many small steps opposite the gradient)

**Approach 3:** Closed Form

(set derivatives equal to zero and solve for parameters)

# Learning as optimization

- The main idea today
  - Make two assumptions:
    - the classifier is linear, like naïve Bayes
      - ie roughly of the form  $f_w(x) = \text{sign}(x \cdot w)$
    - we want to find the classifier  $w$  that “fits the data best”
  - Formalize as an *optimization problem*
    - Pick a loss function  $J(w, D)$ , and find
$$\operatorname{argmin}_w J(w)$$
    - OR: Pick a “goodness” function, often  $\Pr(D|w)$ , and find the  $\operatorname{argmax}_w f_D(w)$

# Learning as optimization: warmup

Find: optimal (MLE) parameter  $\theta$  of a binomial

Dataset:  $D=\{x_1, \dots, x_n\}$ ,  $x_i$  is 0 or 1,  $k$  of them are 1

$$P(D|\theta) = \text{const} \prod_i \theta^{x_i} (1-\theta)^{1-x_i} = \theta^k (1-\theta)^{n-k}$$

$$\frac{d}{d\theta} P(D|\theta) = \frac{d}{d\theta} (\theta^k (1-\theta)^{n-k})$$

$$\begin{aligned} &= \left( \frac{d}{d\theta} \theta^k \right) (1-\theta)^{n-k} + \theta^k \frac{d}{d\theta} (1-\theta)^{n-k} \\ &= k \theta^{k-1} (1-\theta)^{n-k} + \theta^k (n-k) (1-\theta)^{n-k-1} (-1) \end{aligned}$$

$$= \theta^{k-1} (1-\theta)^{n-k-1} (k(1-\theta) - \theta(n-k))$$

# Learning as optimization: warmup

Goal: Find the best parameter  $\theta$  of a binomial

Dataset:  $D = \{x_1, \dots, x_n\}$ ,  $x_i$  is 0 or 1,  $k$  of them are 1

$$\frac{\partial}{\partial \theta} P(D) = \theta^{k-1} (1 - \theta)^{n-k-1} (k(1 - \theta) - \theta(n - k)) = 0$$

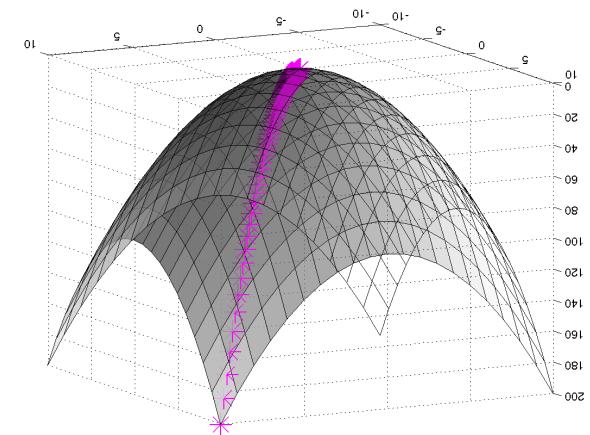
$\theta = 0$        $\theta = 1$

$\downarrow$        $\downarrow$        $\downarrow$

$$k - k\theta - n\theta + k\theta = 0$$
$$\rightarrow n\theta = k$$
$$\rightarrow \theta = k/n$$

# Learning as optimization with gradient ascent

- Goal: Learn the parameter  $w$  of  
...
- Dataset:  $D=\{(x_1, y_1), \dots, (x_n, y_n)\}$
- Use your model to define
  - $\Pr(D|w) = \dots$
- Set  $w$  to maximize Likelihood
  - Usually we use numeric methods to find the optimum
  - i.e., **gradient ascent**: repeatedly take a small step in the direction of the gradient

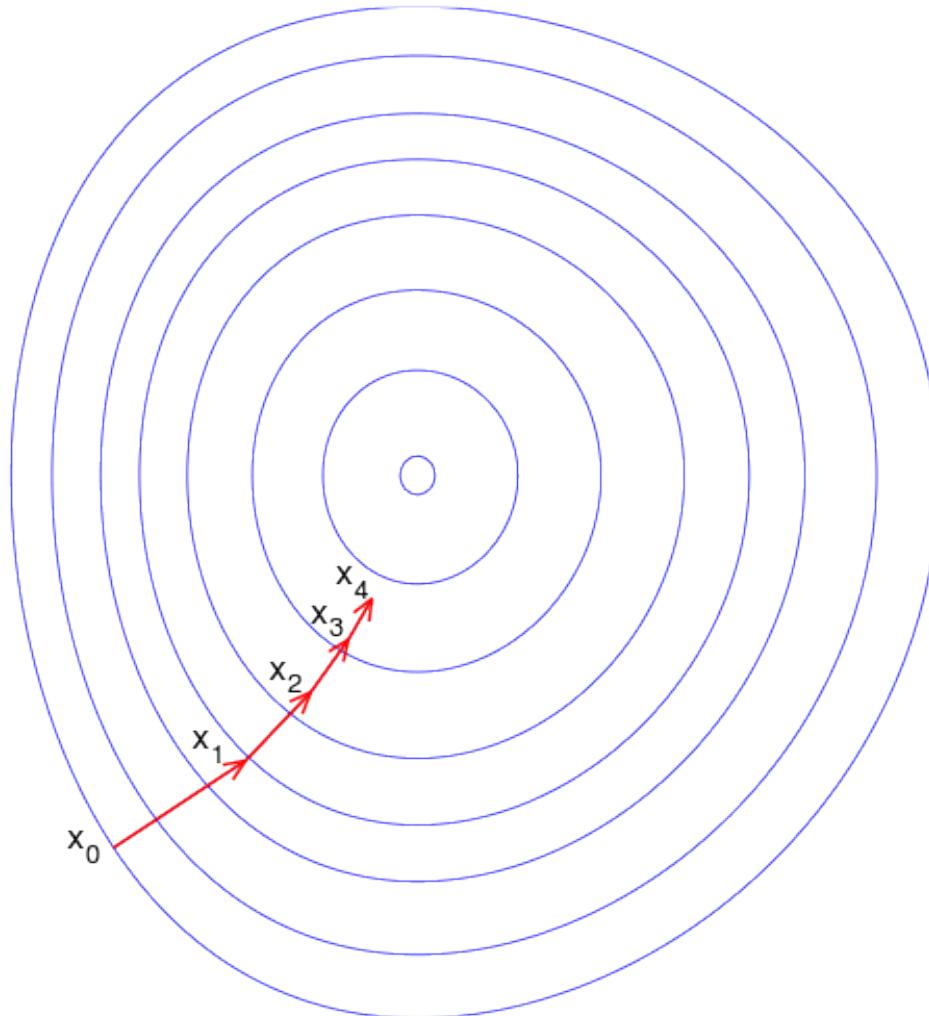


Difference between ascent and descent is only the sign: I'm going to be sloppy here about that.

# Gradient ascent

To find  $\operatorname{argmin}_x f(x)$ :

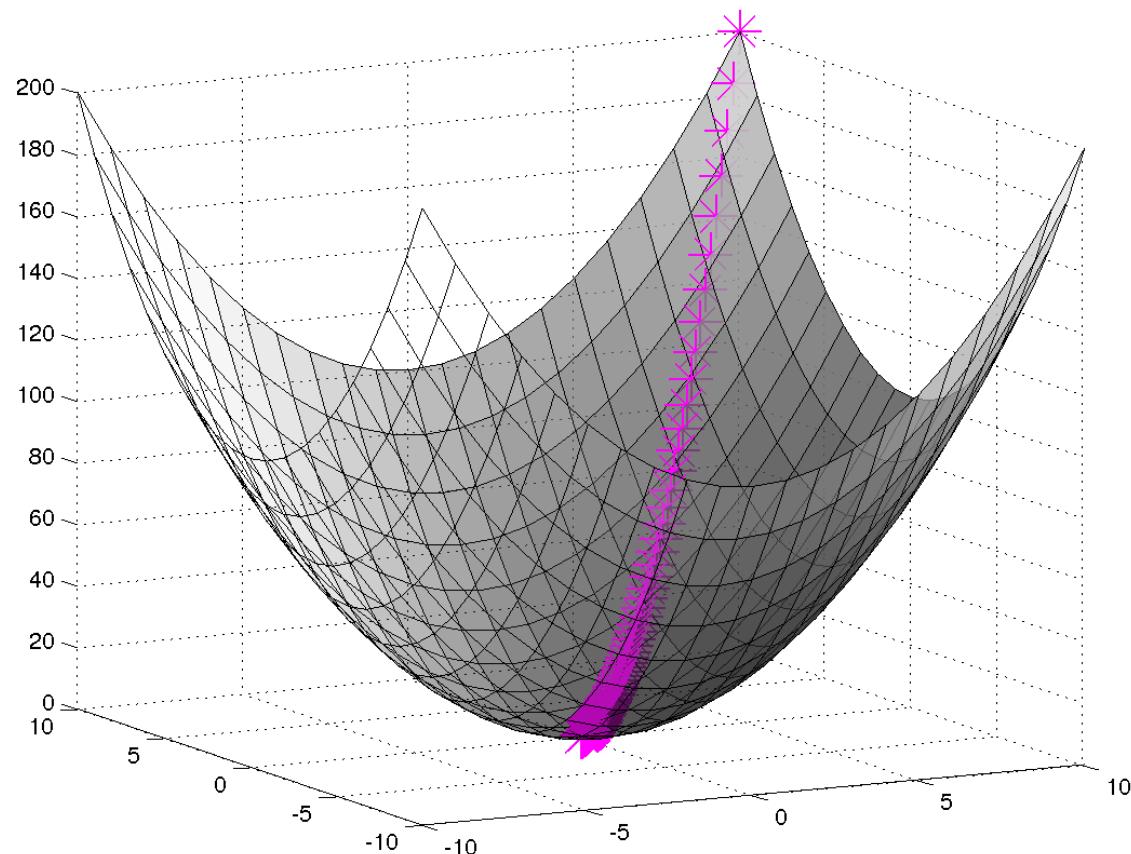
- Start with  $x_0$
- For  $t=1\dots$ 
  - $x_{t+1} = x_t + \lambda f'(x_t)$   
where  $\lambda$  is small



# Gradient descent

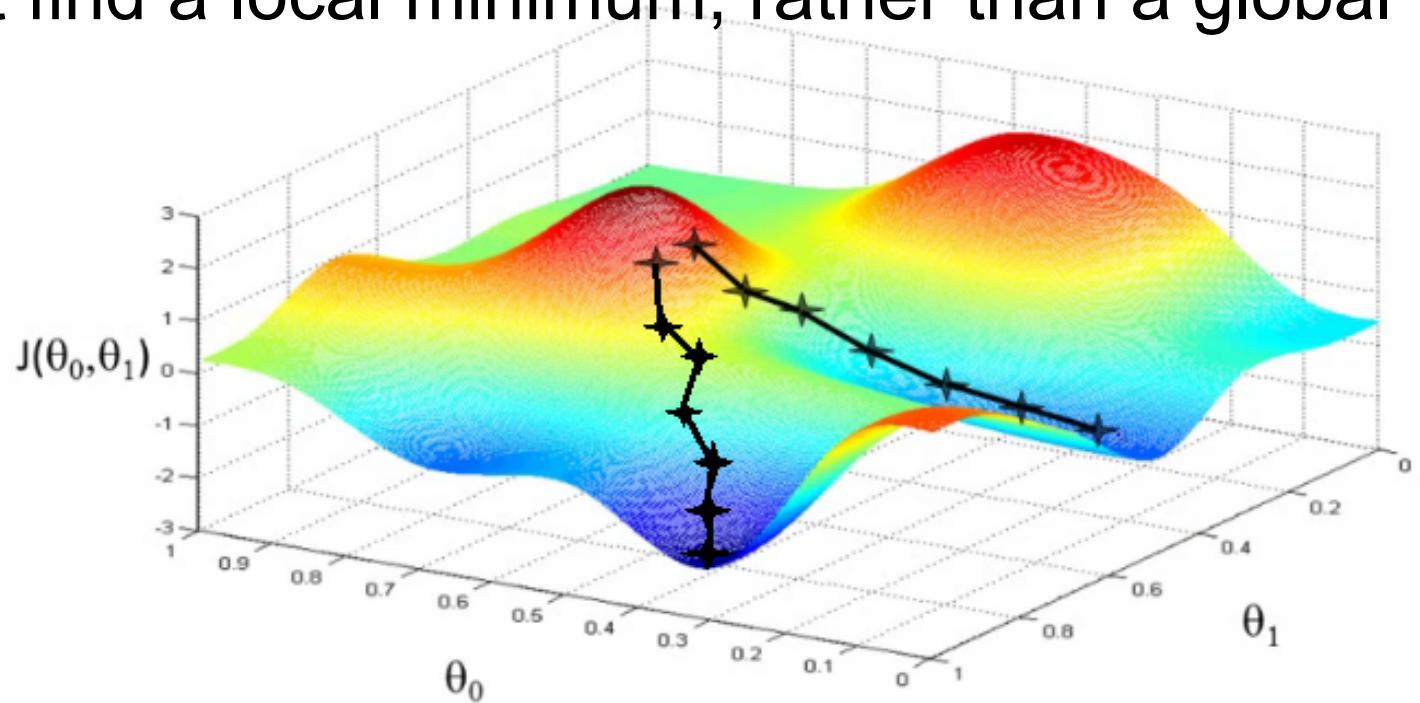
Likelihood: ascent

Loss: descent



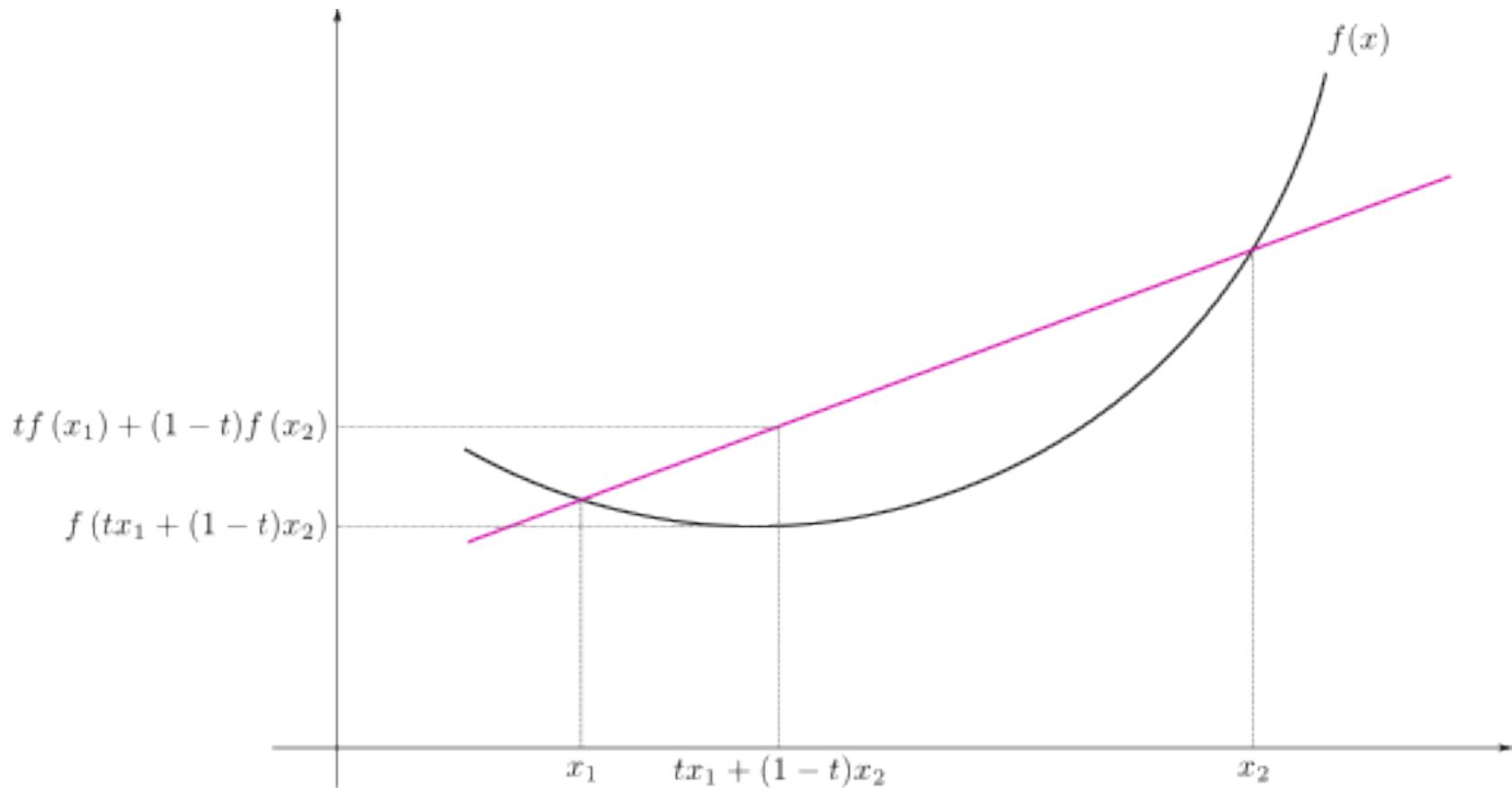
# Pros and cons of gradient descent

- Simple and often quite effective on ML tasks
- Often very scalable
- Only applies to smooth functions (differentiable)
- Might find a local minimum, rather than a global one



# Pros and cons of gradient descent

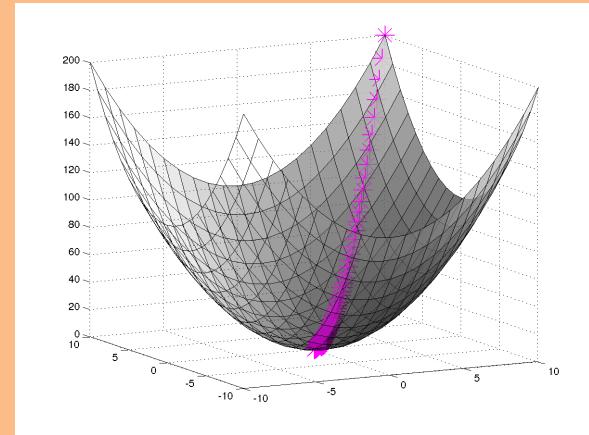
There is only one local optimum if the function is *convex*



# Gradient Descent

## Algorithm 1 Gradient Descent

```
1: procedure GD( $\mathcal{D}$ ,  $\theta^{(0)}$ )
2:    $\theta \leftarrow \theta^{(0)}$ 
3:   while not converged do
4:      $\theta \leftarrow \theta + \lambda \nabla_{\theta} J(\theta)$ 
5:   return  $\theta$ 
```



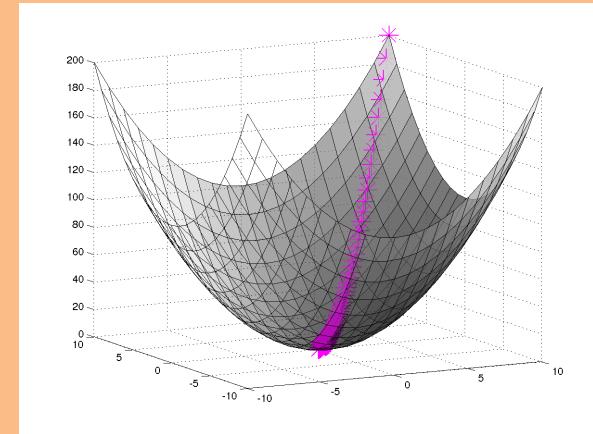
In order to apply GD to Linear Regression all we need is the **gradient** of the objective function (i.e. vector of partial derivatives).

$$\nabla_{\theta} J(\theta) = \begin{bmatrix} \frac{d}{d\theta_1} J(\theta) \\ \frac{d}{d\theta_2} J(\theta) \\ \vdots \\ \frac{d}{d\theta_N} J(\theta) \end{bmatrix}$$

# Gradient Descent

## Algorithm 1 Gradient Descent

```
1: procedure GD( $\mathcal{D}$ ,  $\theta^{(0)}$ )
2:    $\theta \leftarrow \theta^{(0)}$ 
3:   while not converged do
4:      $\theta \leftarrow \theta + \lambda \nabla_{\theta} J(\theta)$ 
5:   return  $\theta$ 
```



There are many possible ways to detect **convergence**. For example, we could check whether the L2 norm of the gradient is below some small tolerance.

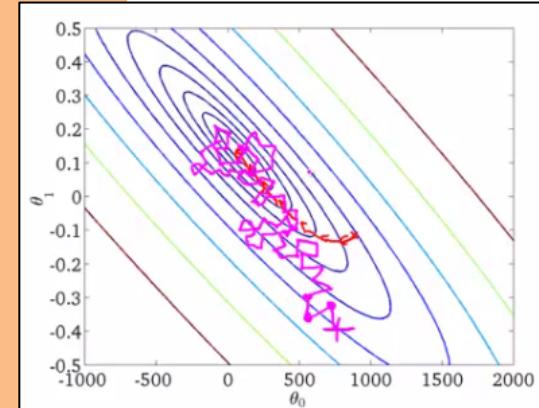
$$\|\nabla_{\theta} J(\theta)\|_2 \leq \epsilon$$

Alternatively we could check that the reduction in the objective function from one iteration to the next is small.

# Stochastic Gradient Descent (SGD)

## Algorithm 2 Stochastic Gradient Descent (SGD)

```
1: procedure SGD( $\mathcal{D}$ ,  $\theta^{(0)}$ )
2:    $\theta \leftarrow \theta^{(0)}$ 
3:   while not converged do
4:     for  $i \in \text{shuffle}(\{1, 2, \dots, N\})$  do
5:        $\theta \leftarrow \theta + \lambda \nabla_{\theta} J^{(i)}(\theta)$ 
6:   return  $\theta$ 
```



Applied to Linear Regression, SGD is called the **Least Mean Squares (LMS)** algorithm

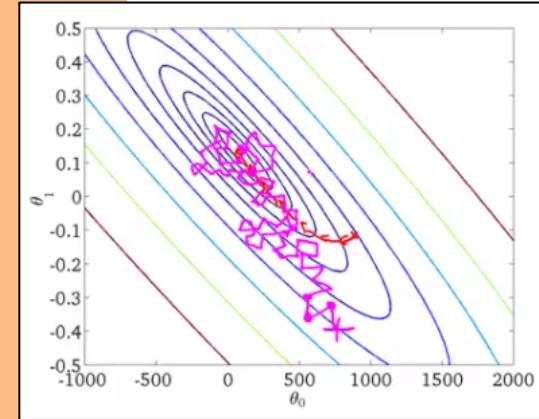
We need a per-example objective:

Let  $J(\theta) = \sum_{i=1}^N J^{(i)}(\theta)$   
where  $J^{(i)}(\theta) = \frac{1}{2}(\theta^T \mathbf{x}^{(i)} - y^{(i)})^2$ .

# Stochastic Gradient Descent (SGD)

## Algorithm 2 Stochastic Gradient Descent (SGD)

```
1: procedure SGD( $\mathcal{D}$ ,  $\theta^{(0)}$ )
2:    $\theta \leftarrow \theta^{(0)}$ 
3:   while not converged do
4:     for  $i \in \text{shuffle}(\{1, 2, \dots, N\})$  do
5:       for  $k \in \{1, 2, \dots, K\}$  do
6:          $\theta_k \leftarrow \theta_k + \lambda \frac{d}{d\theta_k} J^{(i)}(\theta)$ 
7:   return  $\theta$ 
```



Applied to Linear Regression, SGD is called the **Least Mean Squares (LMS)** algorithm

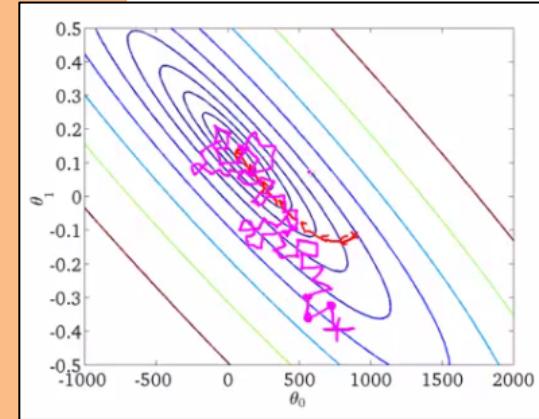
We need a per-example objective:

Let  $J(\theta) = \sum_{i=1}^N J^{(i)}(\theta)$   
where  $J^{(i)}(\theta) = \frac{1}{2}(\theta^T \mathbf{x}^{(i)} - y^{(i)})^2$ .

# Stochastic Gradient Descent (SGD)

## Algorithm 2 Stochastic Gradient Descent (SGD)

```
1: procedure SGD( $\mathcal{D}$ ,  $\theta^{(0)}$ )
2:    $\theta \leftarrow \theta^{(0)}$ 
3:   while not converged do
4:     for  $i \in \text{shuffle}(\{1, 2, \dots, N\})$  do
5:       for  $k \in \{1, 2, \dots, K\}$  do
6:          $\theta_k \leftarrow \theta_k + \lambda \frac{d}{d\theta_k} J^{(i)}(\theta)$ 
7:   return  $\theta$ 
```



Applied to Linear Regression, SGD is called the **Least Mean Squares (LMS)** algorithm.

We need a per-example objective

Let  $J(\theta) = \sum_{i=1}^N J^{(i)}(\theta)$

where  $J^{(i)}(\theta) = \frac{1}{2}(\theta^T \mathbf{x}^{(i)} - y^{(i)})^2$ .

Let's start by calculating this partial derivative for the Linear Regression objective function.

# Partial Derivatives for Linear Reg.

Let  $J(\boldsymbol{\theta}) = \sum_{i=1}^N J^{(i)}(\boldsymbol{\theta})$

where  $J^{(i)}(\boldsymbol{\theta}) = \frac{1}{2}(\boldsymbol{\theta}^T \mathbf{x}^{(i)} - y^{(i)})^2$ .

$$\begin{aligned}\frac{d}{d\theta_k} J^{(i)}(\boldsymbol{\theta}) &= \frac{d}{d\theta_k} \frac{1}{2}(\boldsymbol{\theta}^T \mathbf{x}^{(i)} - y^{(i)})^2 \\ &= \frac{1}{2} \frac{d}{d\theta_k} (\boldsymbol{\theta}^T \mathbf{x}^{(i)} - y^{(i)})^2 \\ &= (\boldsymbol{\theta}^T \mathbf{x}^{(i)} - y^{(i)}) \frac{d}{d\theta_k} (\boldsymbol{\theta}^T \mathbf{x}^{(i)} - y^{(i)}) \\ &= (\boldsymbol{\theta}^T \mathbf{x}^{(i)} - y^{(i)}) \frac{d}{d\theta_k} \left( \sum_{k=1}^K \theta_k x_k^{(i)} - y^{(i)} \right) \\ &= (\boldsymbol{\theta}^T \mathbf{x}^{(i)} - y^{(i)}) x_k^{(i)}\end{aligned}$$

# Partial Derivatives for Linear Reg.

Let  $J(\boldsymbol{\theta}) = \sum_{i=1}^N J^{(i)}(\boldsymbol{\theta})$

where  $J^{(i)}(\boldsymbol{\theta}) = \frac{1}{2}(\boldsymbol{\theta}^T \mathbf{x}^{(i)} - y^{(i)})^2$ .

$$\begin{aligned}\frac{d}{d\theta_k} J^{(i)}(\boldsymbol{\theta}) &= \frac{d}{d\theta_k} \frac{1}{2}(\boldsymbol{\theta}^T \mathbf{x}^{(i)} - y^{(i)})^2 \\ &= \frac{1}{2} \frac{d}{d\theta_k} (\boldsymbol{\theta}^T \mathbf{x}^{(i)} - y^{(i)})^2 \\ &= (\boldsymbol{\theta}^T \mathbf{x}^{(i)} - y^{(i)}) \frac{d}{d\theta_k} (\boldsymbol{\theta}^T \mathbf{x}^{(i)} - y^{(i)}) \\ &= (\boldsymbol{\theta}^T \mathbf{x}^{(i)} - y^{(i)}) \frac{d}{d\theta_k} \left( \sum_{k=1}^K \theta_k x_k^{(i)} - y^{(i)} \right) \\ &= (\boldsymbol{\theta}^T \mathbf{x}^{(i)} - y^{(i)}) x_k^{(i)}\end{aligned}$$

# Partial Derivatives for Linear Reg.

Let  $J(\boldsymbol{\theta}) = \sum_{i=1}^N J^{(i)}(\boldsymbol{\theta})$

where  $J^{(i)}(\boldsymbol{\theta}) = \frac{1}{2}(\boldsymbol{\theta}^T \mathbf{x}^{(i)} - y^{(i)})^2$ .

$$\frac{d}{d\theta_k} J^{(i)}(\boldsymbol{\theta}) = (\boldsymbol{\theta}^T \mathbf{x}^{(i)} - y^{(i)}) x_k^{(i)}$$



Used by SGD  
(aka. LMS)

$$\frac{d}{d\theta_k} J(\boldsymbol{\theta}) = \frac{d}{d\theta_k} \sum_{i=1}^N J^{(i)}(\boldsymbol{\theta})$$

$$= \sum_{i=1}^N (\boldsymbol{\theta}^T \mathbf{x}^{(i)} - y^{(i)}) x_k^{(i)}$$

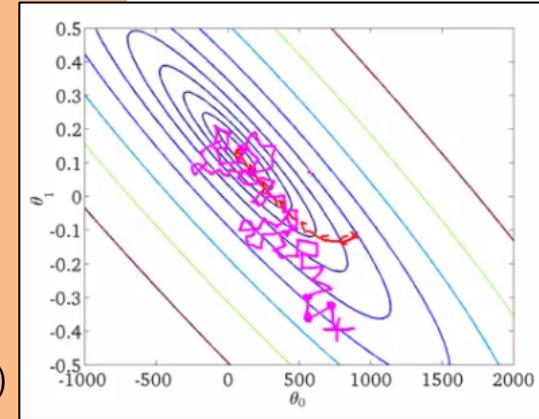


Used by  
Gradient  
Descent

# Least Mean Squares (LMS)

## Algorithm 3 Least Mean Squares (LMS)

```
1: procedure LMS( $\mathcal{D}$ ,  $\theta^{(0)}$ )
2:    $\theta \leftarrow \theta^{(0)}$ 
3:   while not converged do
4:     for  $i \in \text{shuffle}(\{1, 2, \dots, N\})$  do
5:       for  $k \in \{1, 2, \dots, K\}$  do
6:          $\theta_k \leftarrow \theta_k + \lambda(\theta^T \mathbf{x}^{(i)} - y^{(i)}) x_k^{(i)}$ 
7:   return  $\theta$ 
```



Applied to Linear Regression, SGD is called the **Least Mean Squares (LMS)** algorithm

# Least Squares

**Learning:** Three approaches to solving  $\theta^* = \operatorname{argmin}_{\theta} J(\theta)$

**Approach 1:** Gradient Descent

(take larger – more certain – steps opposite the gradient)

**Approach 2:** Stochastic Gradient Descent (SGD)

(take many small steps opposite the gradient)

**Approach 3:** Closed Form

(set derivatives equal to zero and solve for parameters)



# Background: Matrix Derivatives

- For  $f: \mathbb{R}^{m \times n} \mapsto \mathbb{R}$ , define:

$$\nabla_A f(A) = \begin{bmatrix} \frac{\partial}{\partial A_{11}} f & \dots & \frac{\partial}{\partial A_{1n}} f \\ \vdots & \ddots & \vdots \\ \frac{\partial}{\partial A_{1m}} f & \dots & \frac{\partial}{\partial A_{mn}} f \end{bmatrix}$$

- Trace:

$$\text{tr}A = \sum_{i=1}^n A_{ii} \ , \quad \text{tr}a = a \ , \quad \text{tr}ABC = \text{tr}CAB = \text{tr}BCA$$

- Some fact of matrix derivatives (without proof)

$$\nabla_A \text{tr}AB = B^T \ , \quad \nabla_A \text{tr}ABA^TC = CAB + C^T AB^T \ , \quad \nabla_A |A| = |A| (A^{-1})^T$$



# The normal equations

- Write the cost function in matrix form:

$$\begin{aligned}
 J(\theta) &= \frac{1}{2} \sum_{i=1}^n (\mathbf{x}_i^T \theta - y_i)^2 \\
 &= \frac{1}{2} (\mathbf{X}\theta - \bar{y})^T (\mathbf{X}\theta - \bar{y}) \\
 &= \frac{1}{2} (\theta^T \mathbf{X}^T \mathbf{X}\theta - \theta^T \mathbf{X}^T \bar{y} - \bar{y}^T \mathbf{X}\theta + \bar{y}^T \bar{y})
 \end{aligned}$$

- To minimize  $J(\theta)$ , take derivative and set to zero:

$$\begin{aligned}
 \nabla_{\theta} J &= \frac{1}{2} \nabla_{\theta} \text{tr}(\theta^T \mathbf{X}^T \mathbf{X}\theta - \theta^T \mathbf{X}^T \bar{y} - \bar{y}^T \mathbf{X}\theta + \bar{y}^T \bar{y}) \\
 &= \frac{1}{2} (\nabla_{\theta} \text{tr} \theta^T \mathbf{X}^T \mathbf{X}\theta - 2 \nabla_{\theta} \text{tr} \bar{y}^T \mathbf{X}\theta + \nabla_{\theta} \text{tr} \bar{y}^T \bar{y}) \\
 &= \frac{1}{2} (\mathbf{X}^T \mathbf{X}\theta + \mathbf{X}^T \mathbf{X}\theta - 2 \mathbf{X}^T \bar{y}) \\
 &= \mathbf{X}^T \mathbf{X}\theta - \mathbf{X}^T \bar{y} = 0
 \end{aligned}$$

$$\mathbf{X} = \begin{bmatrix} \cdots & \mathbf{x}_1 & \cdots \\ \cdots & \mathbf{x}_2 & \cdots \\ \vdots & \vdots & \vdots \\ \cdots & \mathbf{x}_n & \cdots \end{bmatrix}$$

$$\bar{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}$$

$$\Rightarrow \boxed{X^T X\theta = X^T \bar{y}}$$

**The normal equations**

$$\theta^* = \boxed{\downarrow} (X^T X)^{-1} X^T \bar{y}$$



# Comments on the normal equation

- In most situations of practical interest, the number of data points  $N$  is larger than the dimensionality  $k$  of the input space and the matrix  $\mathbf{X}$  is of full column rank. If this condition holds, then it is easy to verify that  $\mathbf{X}^T\mathbf{X}$  is necessarily invertible.
- The assumption that  $\mathbf{X}^T\mathbf{X}$  is invertible implies that it is positive definite, thus the critical point we have found is a minimum.
- What if  $\mathbf{X}$  has less than full column rank?  $\rightarrow$  regularization (later).



# Direct and Iterative methods

- Direct methods: we can achieve the solution in a single step by solving the normal equation
  - Using Gaussian elimination or QR decomposition, we converge in a finite number of steps
  - It can be infeasible when data are streaming in in real time, or of very large amount
- Iterative methods: stochastic or steepest gradient
  - Converging in a limiting sense
  - But more attractive in large practical problems
  - Caution is needed for deciding the learning rate  $\alpha$



# Convergence rate

- **Theorem:** the steepest descent equation algorithm converge to the minimum of the cost characterized by normal equation:

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

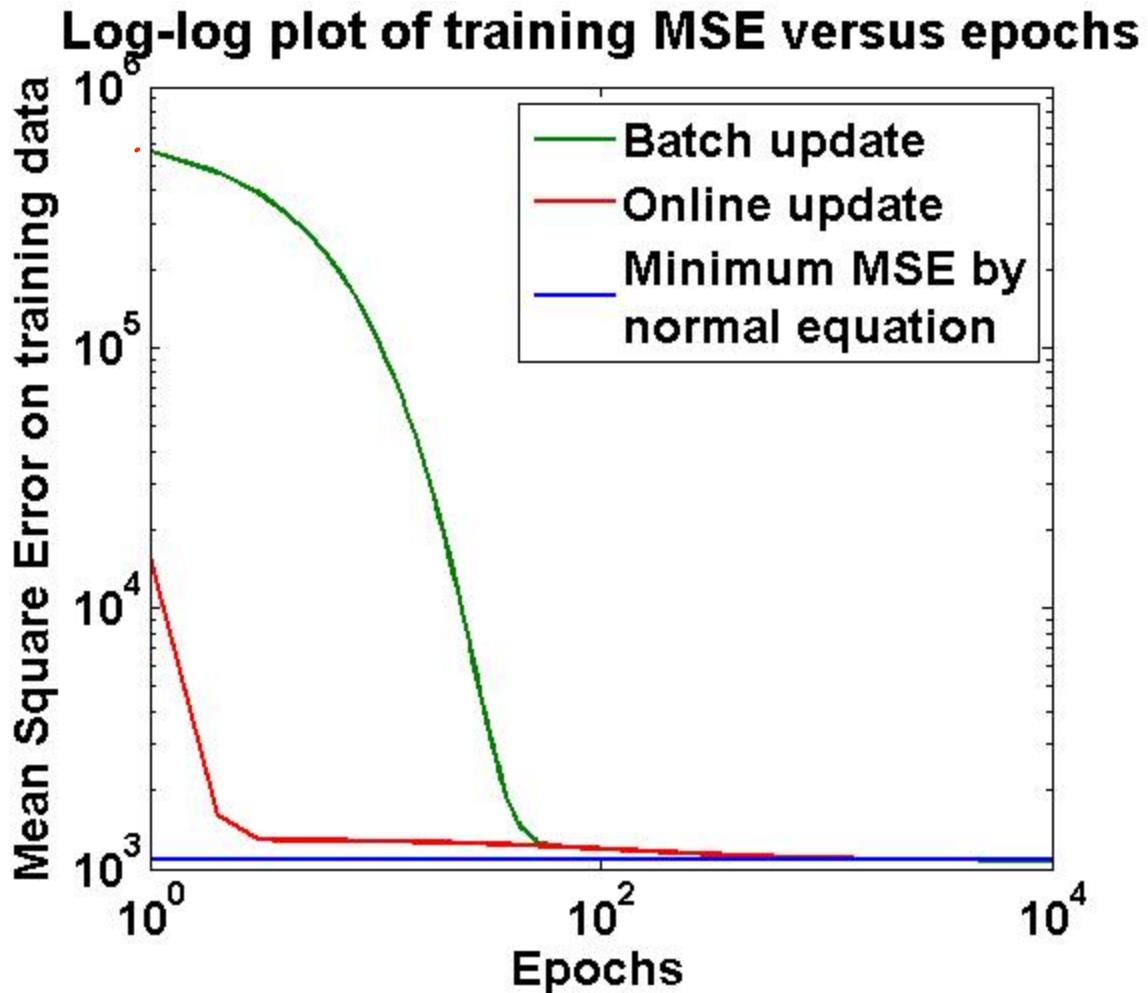
If

$$0 < \alpha < 2/\lambda_{\max}[X^T X]$$

- A formal analysis of LMS needs more math; in practice, one can use a small  $\alpha$ , or gradually decrease  $\alpha$ .



# Convergence Curves



- For the batch method, the training MSE is initially large due to uninformed initialization
- In the online update,  $N$  updates for every epoch reduces MSE to a much smaller value.

# Least Squares

**Learning:** Three approaches to solving  $\theta^* = \operatorname{argmin}_{\theta} J(\theta)$

## **Approach 1:** Gradient Descent

(take larger – more certain – steps opposite the gradient)

- pros: conceptually simple, guaranteed convergence
- cons: batch, often slow to converge

## **Approach 2:** Stochastic Gradient Descent (SGD)

(take many small steps opposite the gradient)

- pros: memory efficient, fast convergence, less prone to local optima
- cons: convergence in practice requires tuning and fancier variants

## **Approach 3:** Closed Form

(set derivatives equal to zero and solve for parameters)

- pros: one shot algorithm!
- cons: does not scale to large datasets (matrix inverse is bottleneck)

# Geometric Interpretation of LMS

- The predictions on the training data are:

$$\hat{\vec{y}} = X\theta^* = X(X^T X)^{-1} X^T \vec{y}$$

- Note that

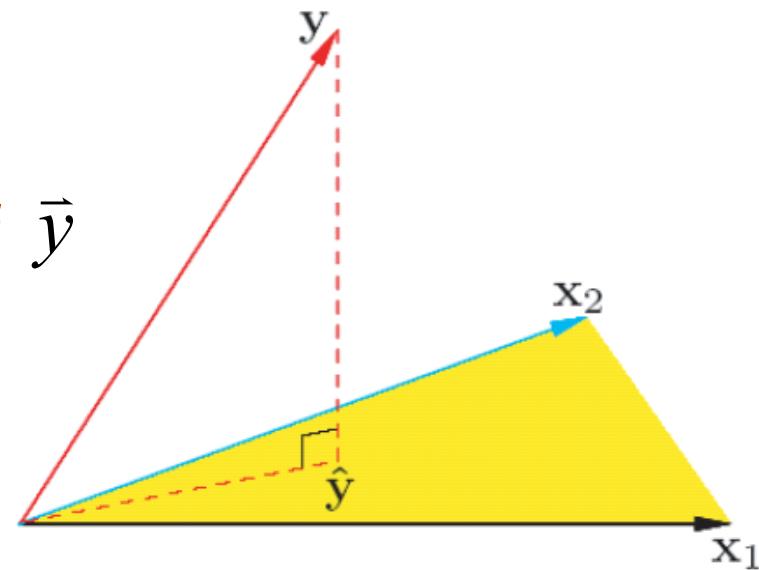
$$\hat{\vec{y}} - \vec{y} = (X(X^T X)^{-1} X^T - I)\vec{y}$$

and

$$\begin{aligned} X^T(\hat{\vec{y}} - \vec{y}) &= X^T(X(X^T X)^{-1} X^T - I)\vec{y} \\ &= (X^T X(X^T X)^{-1} X^T - X^T)\vec{y} \\ &= 0 !! \end{aligned}$$

$\hat{\vec{y}}$  is the orthogonal projection of  $\vec{y}$  into the space spanned by the columns of  $X$

$$\vec{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} \quad \mathbf{X} = \begin{bmatrix} \cdots & \mathbf{x}_1 & \cdots \\ \cdots & \mathbf{x}_2 & \cdots \\ \vdots & \vdots & \vdots \\ \cdots & \mathbf{x}_n & \cdots \end{bmatrix}$$

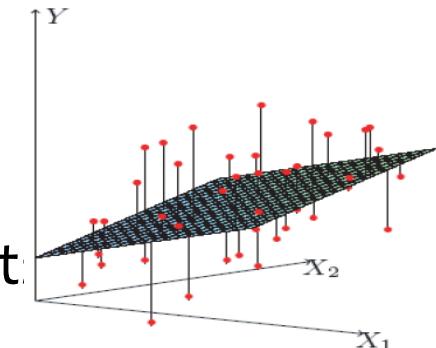


# Probabilistic Interpretation of LMS

- Let us assume that the target variable and the inputs are related by the equation:

$$y_i = \theta^T \mathbf{x}_i + \varepsilon_i$$

where  $\varepsilon$  is an error term of unmodeled effect noise



- Now assume that  $\varepsilon$  follows a Gaussian  $N(0, \sigma^2)$ , then we have:

$$p(y_i | x_i; \theta) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(y_i - \theta^T \mathbf{x}_i)^2}{2\sigma^2}\right)$$

- By independence assumption:

$$L(\theta) = \prod_{i=1}^n p(y_i | x_i; \theta) = \left(\frac{1}{\sqrt{2\pi\sigma^2}}\right)^n \exp\left(-\frac{\sum_{i=1}^n (y_i - \theta^T \mathbf{x}_i)^2}{2\sigma^2}\right)$$

# Probabilistic Interpretation of LMS

- Hence the log-likelihood is:

$$l(\theta) = n \log \frac{1}{\sqrt{2\pi\sigma^2}} - \frac{1}{\sigma^2} \frac{1}{2} \sum_{i=1}^n (y_i - \theta^T \mathbf{x}_i)^2$$

- Do you recognize the last term?

Yes it is:

$$J(\theta) = \frac{1}{2} \sum_{i=1}^n (\mathbf{x}_i^T \theta - y_i)^2$$

- Thus under independence assumption, LMS is equivalent to MLE of  $\theta$  !

# Non-Linear basis function

- So far we only used the observed values  $x_1, x_2, \dots$
- However, linear regression can be applied in the same way to **functions** of these values
  - Eg: to add a term  $w x_1 x_2$  add a new variable  $z = x_1 x_2$  so each example becomes:  $x_1, x_2, \dots, z$
- As long as these functions can be directly computed from the observed values the parameters are still linear in the data and the problem remains a multi-variate linear regression problem

$$y = w_0 + w_1 x_1^2 + \dots + w_k x_k^2 + \varepsilon$$

# Non-linear basis functions

- What type of functions can we use?
- A few common examples:

- Polynomial:  $\phi_j(x) = x^j$  for  $j=0 \dots n$

- Gaussian:  $\phi_j(x) = \frac{(x - \mu_j)}{2\sigma_j^2}$

- Sigmoid:  $\phi_j(x) = \frac{1}{1 + \exp(-s_j x)}$

- Logs:  $\phi_j(x) = \log(x + 1)$

Any function of the input values can be used. The solution for the parameters of the regression remains the same.

# General linear regression problem

- Using our new notations for the basis function linear regression can be written as

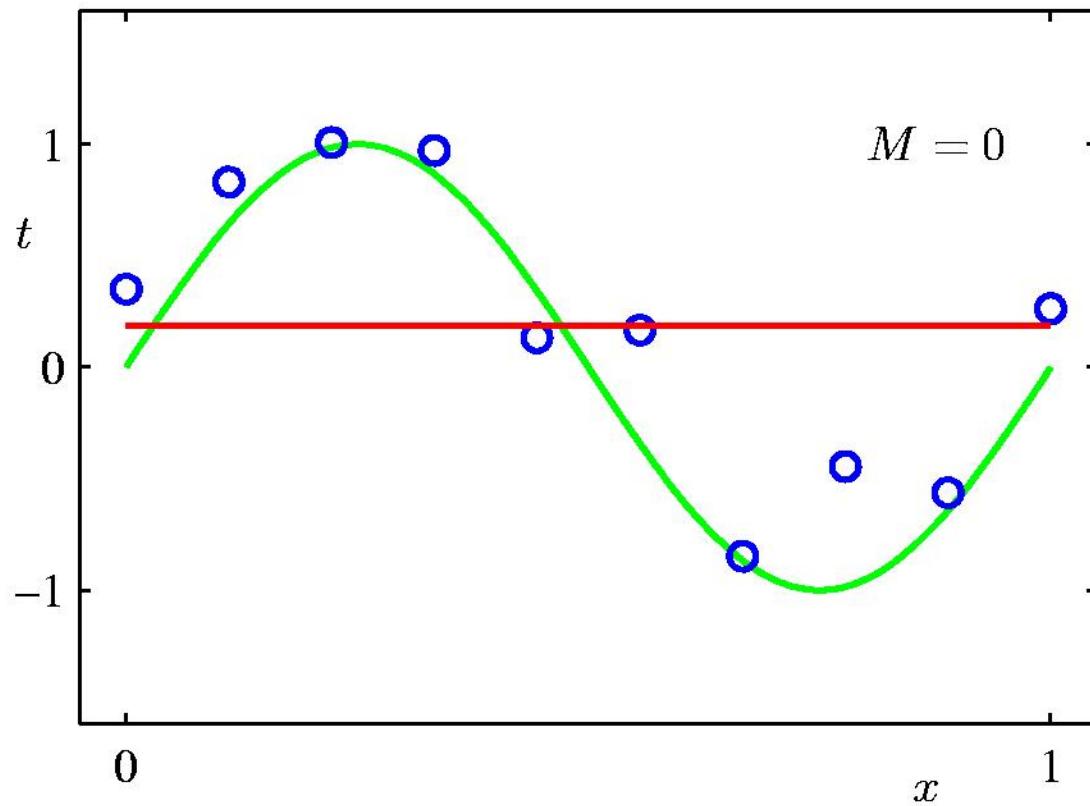
$$y = \sum_{j=0}^n w_j \phi_j(x)$$

- Where  $\phi_j(\mathbf{x})$  can be either  $x_j$  for multivariate regression or one of the non-linear basis functions we defined
- ... and  $\phi_0(\mathbf{x})=1$  for the intercept term

# An example: polynomial basis vectors on a small dataset

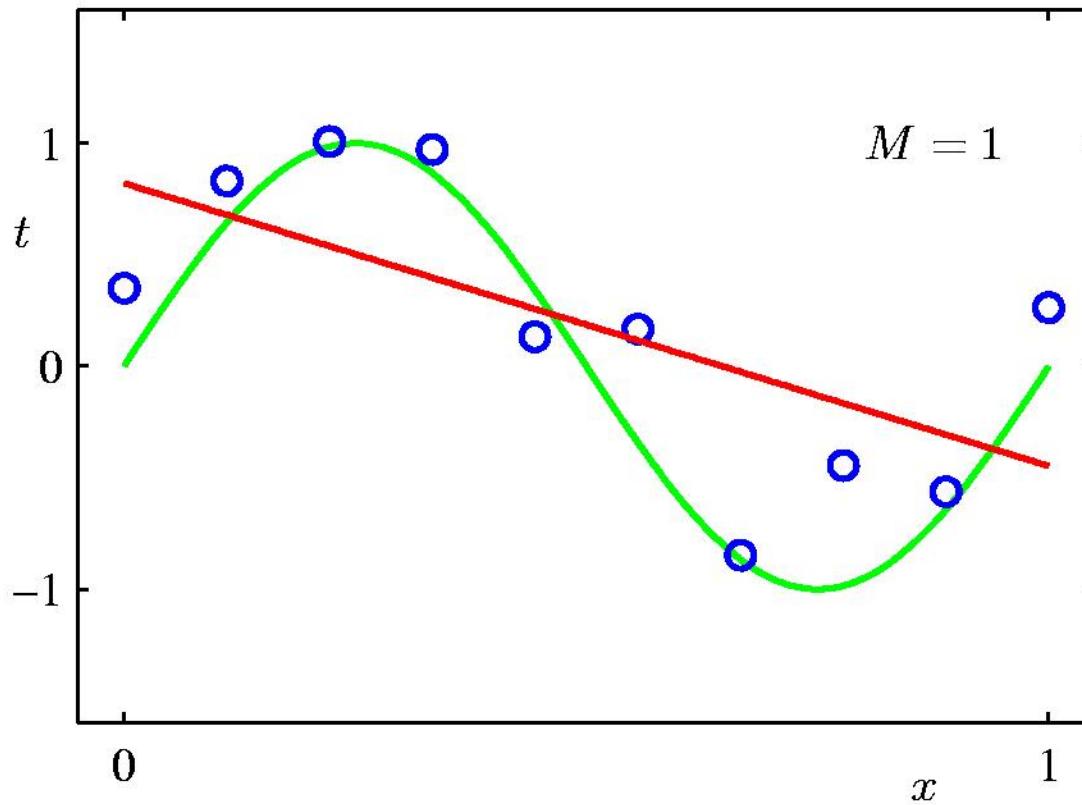
– From Bishop Ch 1

# 0<sup>th</sup> Order Polynomial

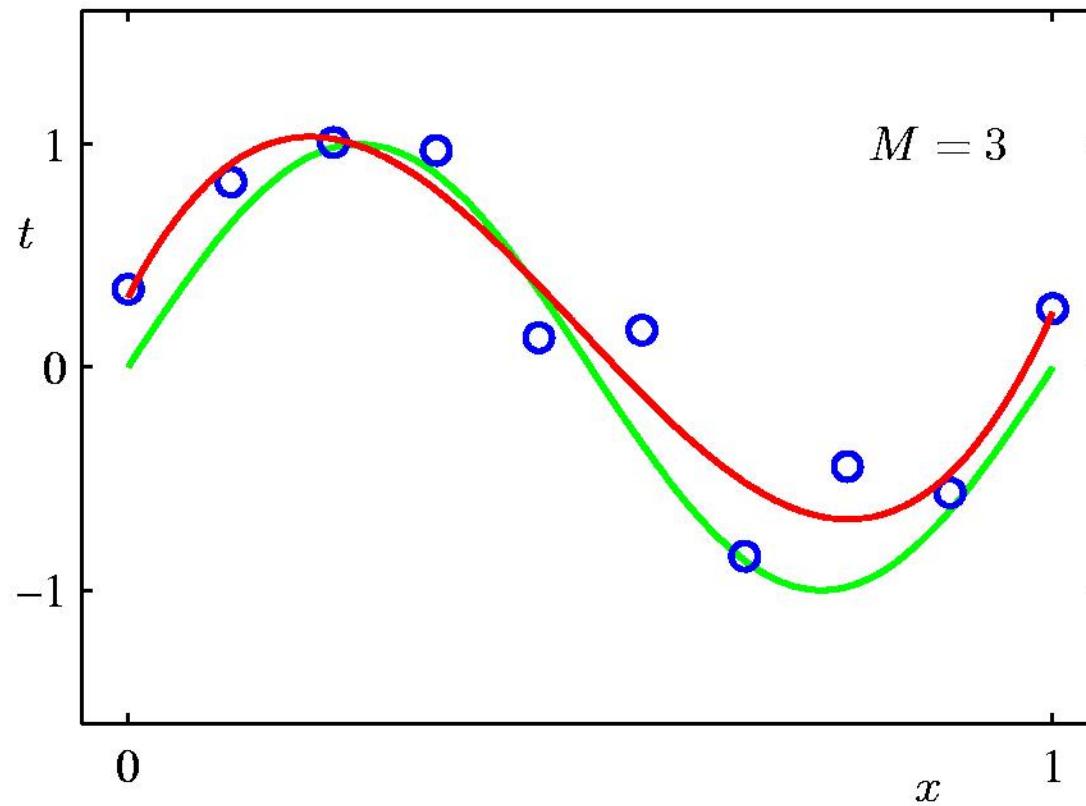


$n=10$

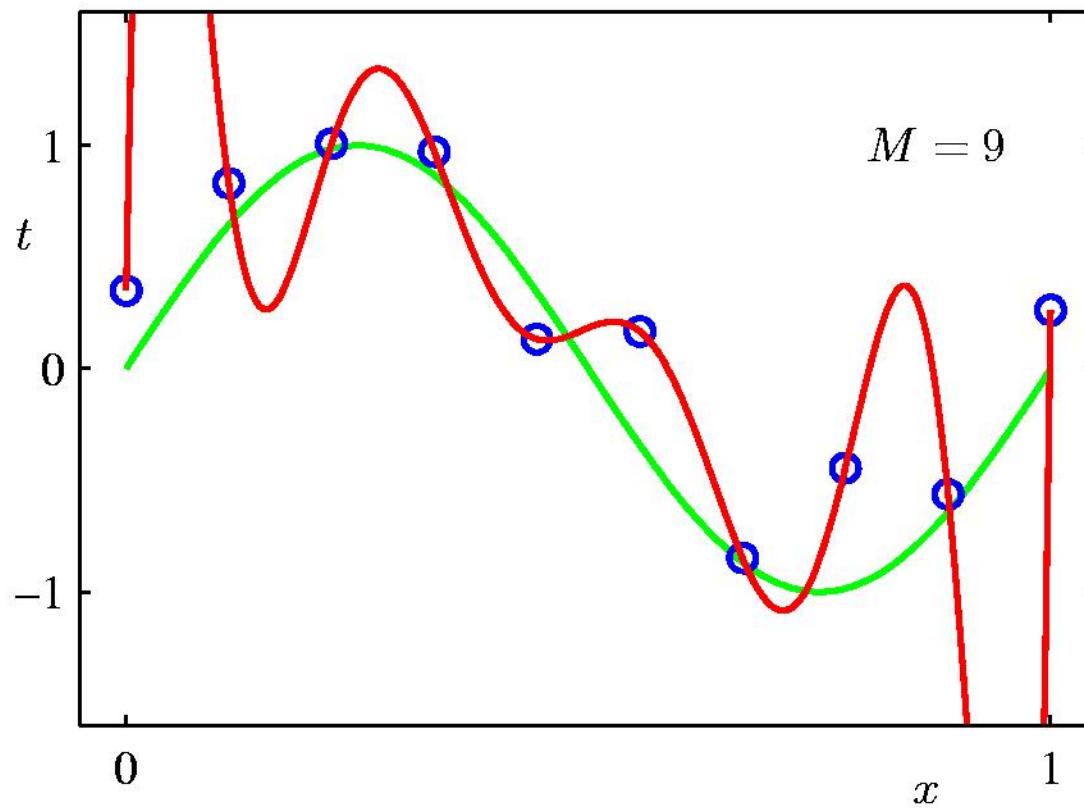
# 1<sup>st</sup> Order Polynomial



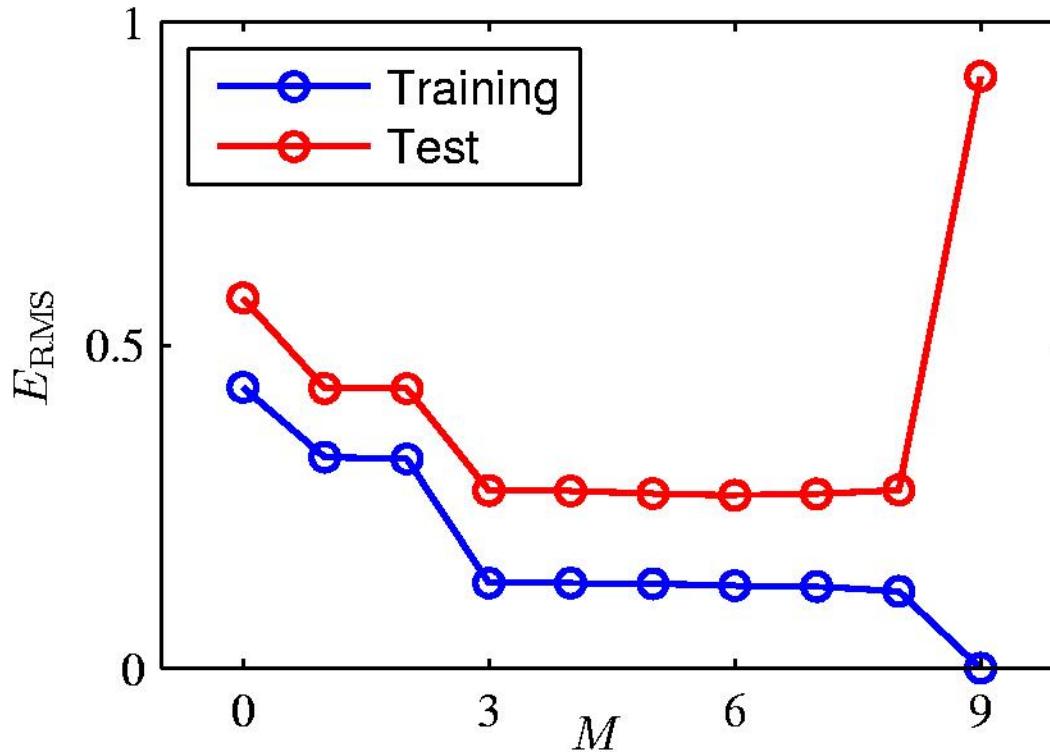
# 3<sup>rd</sup> Order Polynomial



# 9<sup>th</sup> Order Polynomial



# Over-fitting



Root-Mean-Square (RMS) Error:  $E_{\text{RMS}} = \sqrt{2E(\mathbf{w}^*)/N}$

# Polynomial Coefficients

	$M = 0$	$M = 1$	$M = 3$	$M = 9$
$w_0^*$	0.19	0.82	0.31	0.35
$w_1^*$		-1.27	7.99	232.37
$w_2^*$			-25.43	-5321.83
$w_3^*$			17.37	48568.31
$w_4^*$				-231639.30
$w_5^*$				640042.26
$w_6^*$				-1061800.52
$w_7^*$				1042400.18
$w_8^*$				-557682.99
$w_9^*$				125201.43

# Regularization

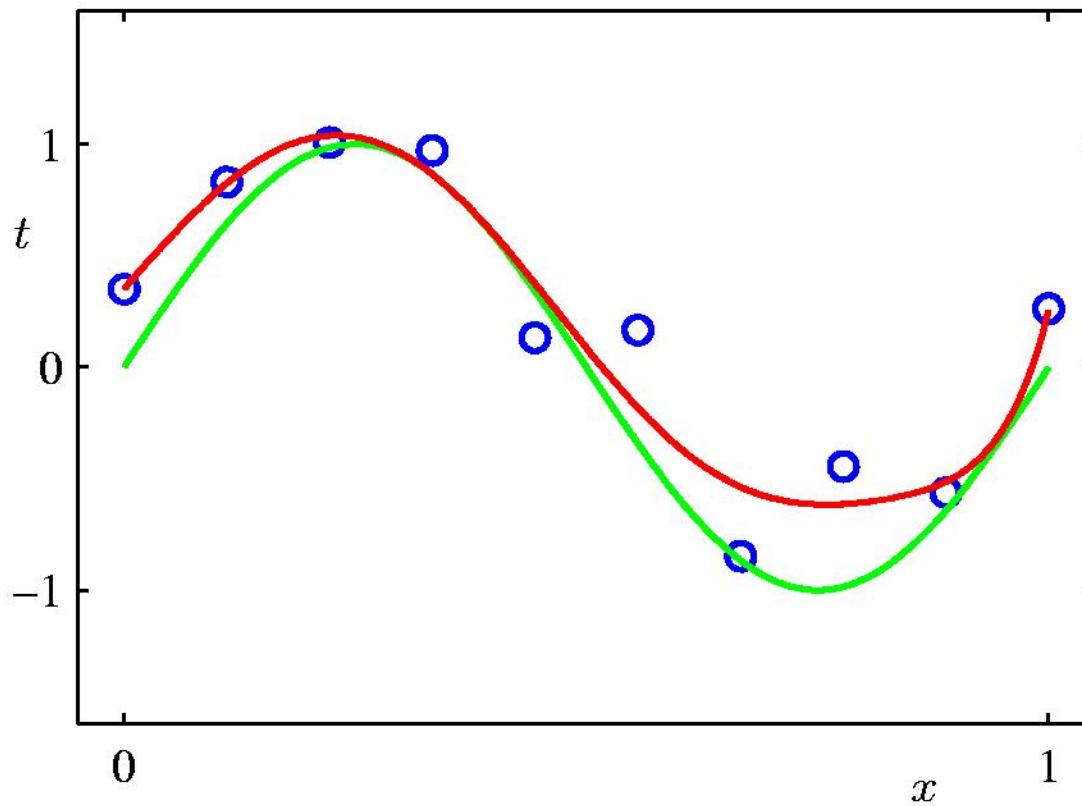
Penalize large coefficient values

$$J_{\mathbf{x}, \mathbf{y}}(\mathbf{w}) = \frac{1}{2} \sum_i \left( y^i - \sum_j w_j \phi_j(\mathbf{x}^i) \right)^2 - \frac{\lambda}{2} \|\mathbf{w}\|^2$$

---

# Regularization:

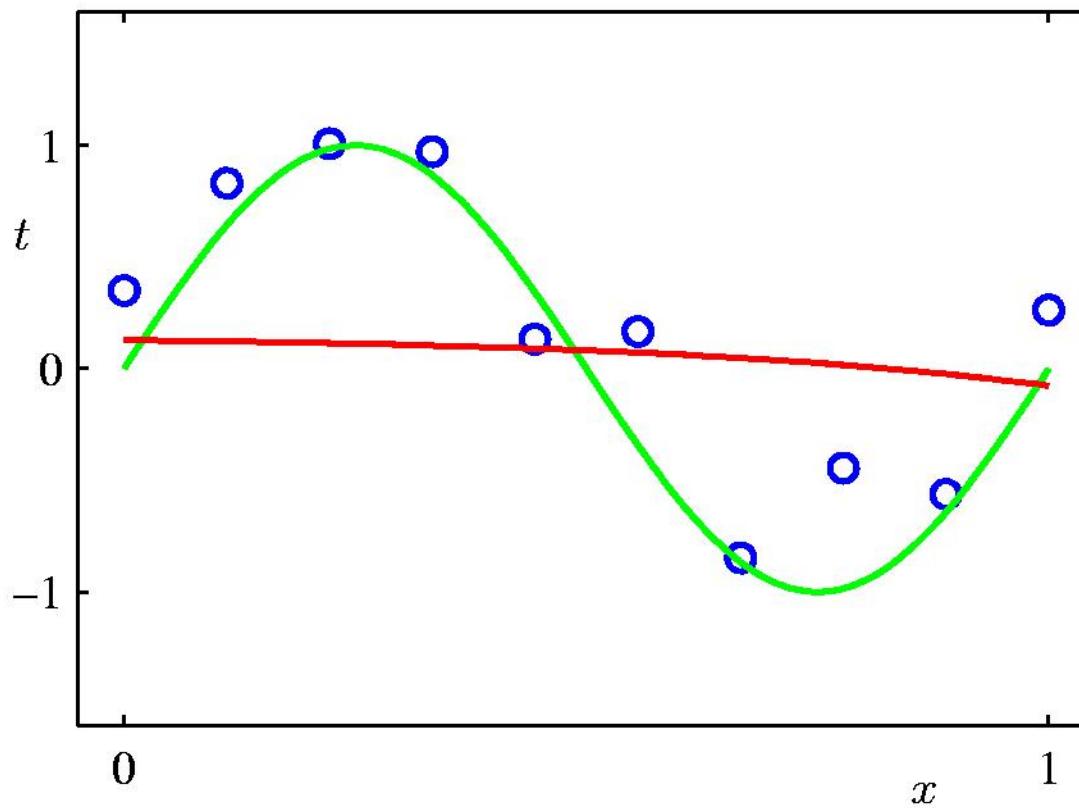
$$\ln \lambda = +.18$$



# Polynomial Coefficients

	none	exp(18)	huge
$w_0^*$	0.35	0.35	0.13
$w_1^*$	232.37	4.74	-0.05
$w_2^*$	-5321.83	-0.77	-0.06
$w_3^*$	48568.31	-31.97	-0.05
$w_4^*$	-231639.30	-3.89	-0.03
$w_5^*$	640042.26	55.28	-0.02
$w_6^*$	-1061800.52	41.32	-0.01
$w_7^*$	1042400.18	-45.95	-0.00
$w_8^*$	-557682.99	-91.53	0.00
$w_9^*$	125201.43	72.68	0.01

# Over Regularization:



# Example: Stock Prices

- Suppose we wish to predict Google's stock price at time  $t+1$
- **What features should we use?**  
(putting all computational concerns aside)
  - Stock prices of all other stocks at times  $t$ ,  $t-1$ ,  $t-2$ , ...,  $t-k$
  - Mentions of Google with positive / negative sentiment words in all newspapers and social media outlets
- Do we believe that **all** of these features are going to be useful?

# Ridge Regression

- Adds an **L2 regularizer** to Linear Regression

$$J_{\text{RR}}(\boldsymbol{\theta}) = J(\boldsymbol{\theta}) + \boxed{\lambda \|\boldsymbol{\theta}\|_2^2}$$

$$= \frac{1}{2} \sum_{i=1}^N (\boldsymbol{\theta}^T \mathbf{x}^{(i)} - y^{(i)})^2 + \boxed{\lambda \sum_{k=1}^K \theta_k^2}$$

prefers  
parameters  
close to zero

- Bayesian interpretation: MAP estimation with a **Gaussian prior** on the parameters

$$\begin{aligned} \boldsymbol{\theta}^{MAP} &= \operatorname{argmax}_{\boldsymbol{\theta}} \sum_{i=1}^N \log p_{\boldsymbol{\theta}}(y^{(i)} | \mathbf{x}^{(i)}) + \log p(\boldsymbol{\theta}) \\ &= \operatorname{argmax}_{\boldsymbol{\theta}} J_{\text{RR}}(\boldsymbol{\theta}) \end{aligned}$$

where  
 $p(\boldsymbol{\theta}) \sim \mathcal{N}(0, \frac{1}{\lambda})$

# LASSO

- Adds an **L1 regularizer** to Linear Regression

$$J_{\text{LASSO}}(\boldsymbol{\theta}) = J(\boldsymbol{\theta}) + \boxed{\lambda \|\boldsymbol{\theta}\|_1}$$

$$= \frac{1}{2} \sum_{i=1}^N (\boldsymbol{\theta}^T \mathbf{x}^{(i)} - y^{(i)})^2 + \boxed{\lambda \sum_{k=1}^K |\theta_k|}$$

yields sparse  
parameters  
(exact zeros)

- Bayesian interpretation: MAP estimation with a **Laplace prior** on the parameters

$$\boldsymbol{\theta}^{MAP} = \operatorname{argmax}_{\boldsymbol{\theta}} \sum_{i=1}^N \log p_{\boldsymbol{\theta}}(y^{(i)} | \mathbf{x}^{(i)}) + \log p(\boldsymbol{\theta})$$

$$= \operatorname{argmax}_{\boldsymbol{\theta}} J_{\text{LASSO}}(\boldsymbol{\theta})$$

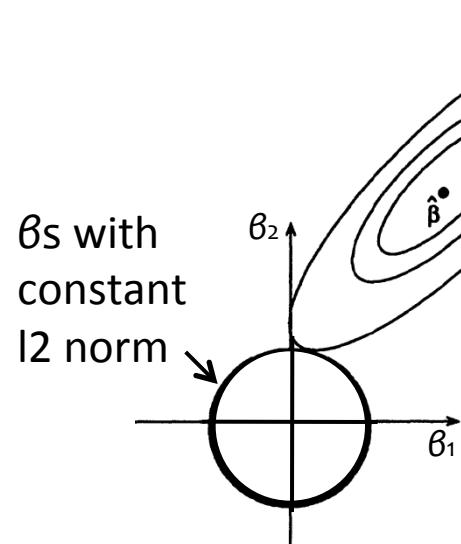
where  
 $p(\boldsymbol{\theta}) \sim \text{Laplace}(0, f(\lambda))$

# Ridge Regression vs Lasso

$$\min_{\beta} (\mathbf{X}\beta - \mathbf{Y})^T (\mathbf{X}\beta - \mathbf{Y}) + \lambda \text{pen}(\beta) = \min_{\beta} J(\beta) + \lambda \text{pen}(\beta)$$

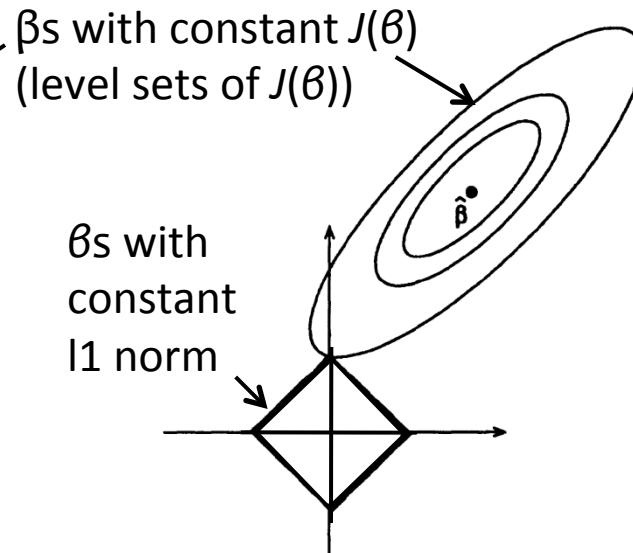
Ridge Regression:

$$\text{pen}(\beta) = \|\beta\|_2^2$$



Lasso:

$$\text{pen}(\beta) = \|\beta\|_1$$



**Lasso (l1 penalty) results in sparse solutions – vector with more zero coordinates  
Good for high-dimensional problems – don't have to store all coordinates!**

# Optimization for LASSO

Can we apply SGD to the LASSO learning problem?

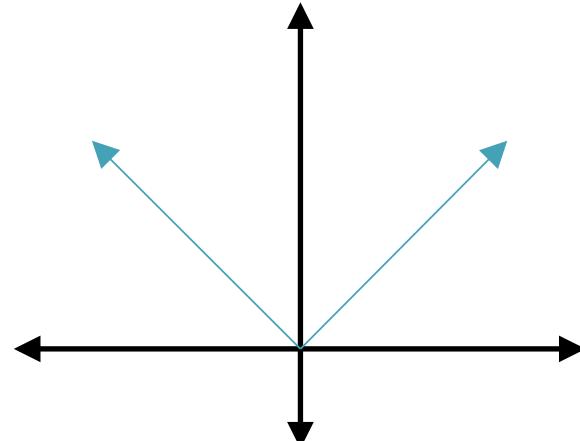
$$\operatorname*{argmax}_{\boldsymbol{\theta}} J_{\text{LASSO}}(\boldsymbol{\theta})$$

$$\begin{aligned} J_{\text{LASSO}}(\boldsymbol{\theta}) &= J(\boldsymbol{\theta}) + \boxed{\lambda \|\boldsymbol{\theta}\|_1} \\ &= \frac{1}{2} \sum_{i=1}^N (\boldsymbol{\theta}^T \mathbf{x}^{(i)} - y^{(i)})^2 + \boxed{\lambda \sum_{k=1}^K |\theta_k|} \end{aligned}$$

# Optimization for LASSO

- Consider the absolute value function:

$$r(\theta) = \lambda \sum_{k=1}^K |\theta_k|$$



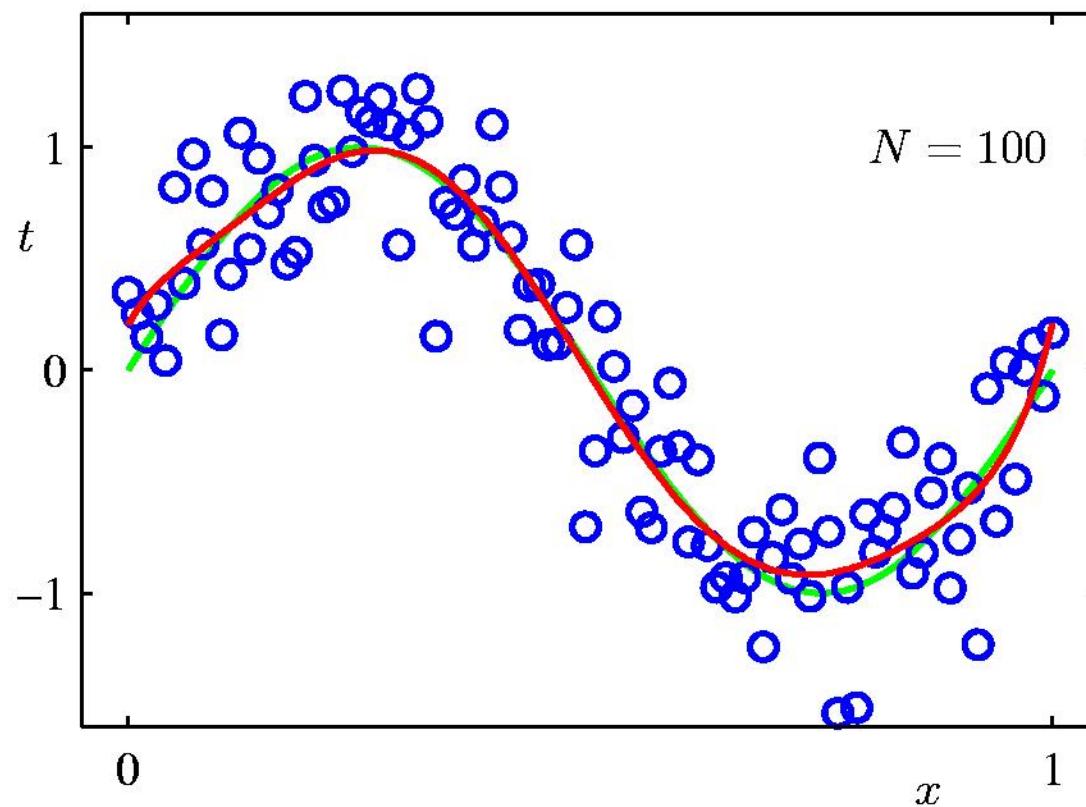
- The L1 penalty is subdifferentiable (i.e. not differentiable at 0)
- Where does a small step opposite the gradient accomplish...
  - ... far from zero?
  - ... close to zero?

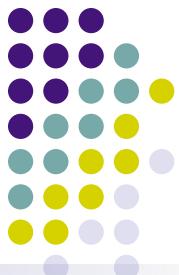
# Optimization for LASSO

- The L1 penalty is subdifferentiable (i.e. not differentiable at 0)
- An array of optimization algorithms exist to handle this issue:
  - Coordinate Descent
  - Orthant-Wise Limited memory Quasi-Newton (OWL-QN) (Andrew & Gao, 2007) and provably convergent variants
  - Block coordinate Descent (Tseng & Yun, 2009)
  - Sparse Reconstruction by Separable Approximation (SpaRSA) (Wright et al., 2009)
  - Fast Iterative Shrinkage Thresholding Algorithm (FISTA) (Beck & Teboulle, 2009)

# Data Set Size:

9<sup>th</sup> Order Polynomial  $N = 100$





# Locally weighted linear regression

- The algorithm:

Instead of minimizing

$$J(\theta) = \frac{1}{2} \sum_{i=1}^n (\mathbf{x}_i^T \theta - y_i)^2$$

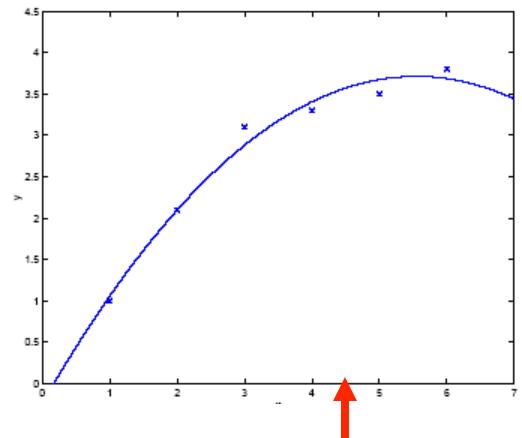
now we fit  $\theta$  to minimize

$$J(\theta) = \frac{1}{2} \sum_{i=1}^n w_i (\mathbf{x}_i^T \theta - y_i)^2$$

Where do  $w_i$ 's come from?

$$w_i = \exp\left(-\frac{(\mathbf{x}_i - \mathbf{x})^2}{2\tau^2}\right)$$

- where  $\mathbf{x}$  is the query point for which we'd like to know its corresponding  $\mathbf{y}$

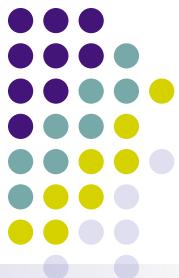


→ Essentially we put higher weights on (errors on) training examples that are close to the query point (than those that are further away from the query)



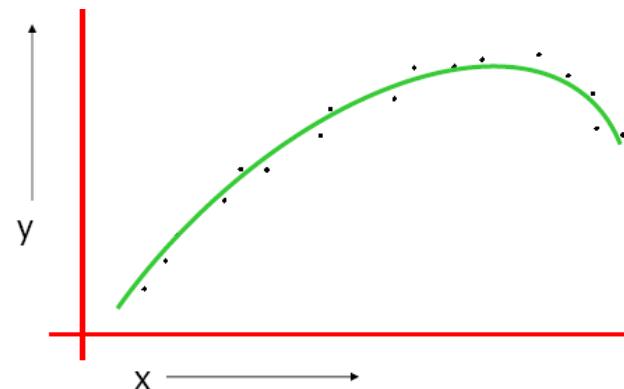
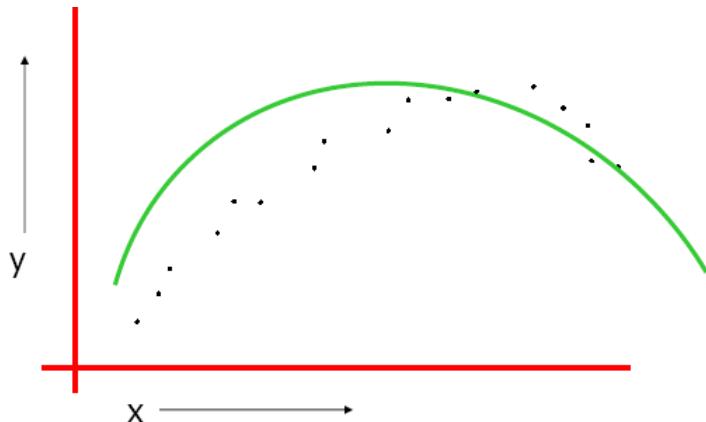
# Parametric vs. non-parametric

- Locally weighted linear regression is the second example we are running into of a **non-parametric** algorithm. (what is the first?)
- The (unweighted) linear regression algorithm that we saw earlier is known as a **parametric** learning algorithm
  - because it has a fixed, finite number of parameters (the  $\theta$ ), which are fit to the data;
  - Once we've fit the  $\theta$  and stored them away, we no longer need to keep the training data around to make future predictions.
  - In contrast, to make predictions using locally weighted linear regression, we need to keep the entire training set around.
- The term "**non-parametric**" (roughly) refers to the fact that the amount of stuff we need to keep in order to represent the hypothesis grows linearly with the size of the training set.



# Robust Regression

- The best fit from a quadratic regression
- But this is probably better ...

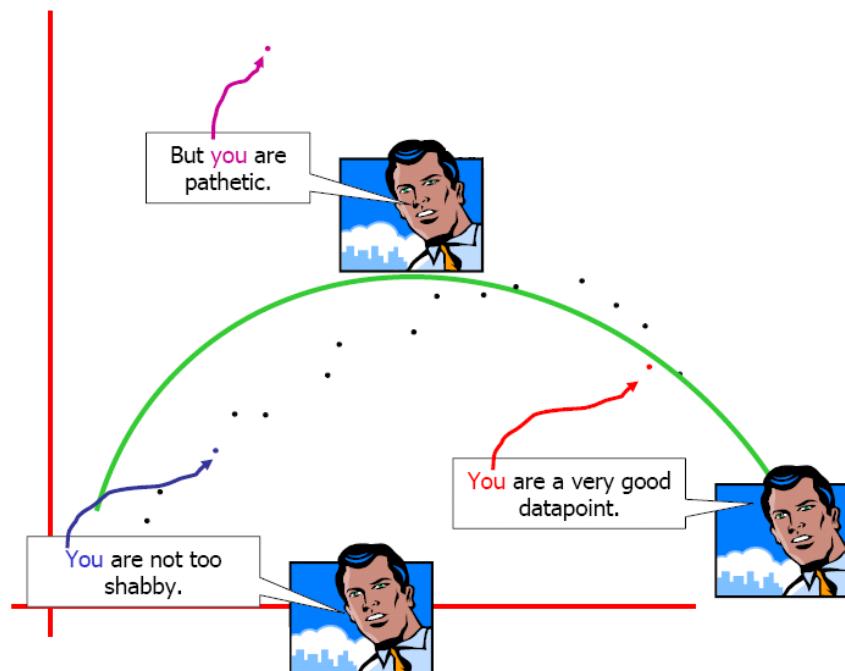


How can we do this?

# LOESS-based Robust Regression



- Remember what we do in "locally weighted linear regression"?  
→ we "score" each point for its impotence
- Now we score each point according to its "fitness"



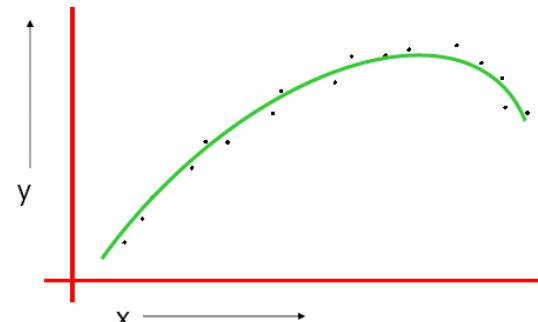
(Courtesy to Andrew Moore)

# Robust regression



- For  $k = 1$  to  $R$ ...
  - Let  $(x_k, y_k)$  be the  $k$ th datapoint
  - Let  $y_k^{\text{est}}$  be predicted value of  $y_k$
  - Let  $w_k$  be a weight for data point  $k$  that is large if the data point fits well and small if it fits badly:

$$w_k = \phi((y_k - y_k^{\text{est}})^2)$$



- Then redo the regression using weighted data points.
- Repeat whole thing until converged!

# Robust regression—probabilistic interpretation



- **What regular regression does:**

Assume  $y_k$  was originally generated using the following recipe:

$$y_k = \theta^T \mathbf{x}_k + \mathcal{N}(0, \sigma^2)$$

Computational task is to find the Maximum Likelihood estimation of  $\theta$

# Robust regression—probabilistic interpretation



- What LOESS robust regression does:

Assume  $y_k$  was originally generated using the following recipe:

with probability  $p$ :  $y_k = \theta^T \mathbf{x}_k + \mathcal{N}(0, \sigma^2)$

but otherwise  $y_k \sim \mathcal{N}(\mu, \sigma_{\text{huge}}^2)$

Computational task is to find the Maximum Likelihood estimates of  $\theta$ ,  $p$ ,  $\mu$  and  $\sigma_{\text{huge}}$ .

- The algorithm you saw with iterative **reweighting/refitting** does this computation for us. Later you will find that it is an instance of the famous **E.M. algorithm**

# Summary

- Linear regression **predicts** its output as a **linear function** of its inputs
- Learning **optimizes** a function (equivalently likelihood or mean squared error) using **standard techniques** (gradient descent, SGD, closed form)
- Regularization enables **shrinkage** and **model selection**