School of Computer Science
Carnegie Mellon

10-701 Introduction to Machine Learning

Linear Regression

Readings:
Bishop, 3.1
Murphy, 7

Matt Gormley
Lecture 4
September 19, 2016

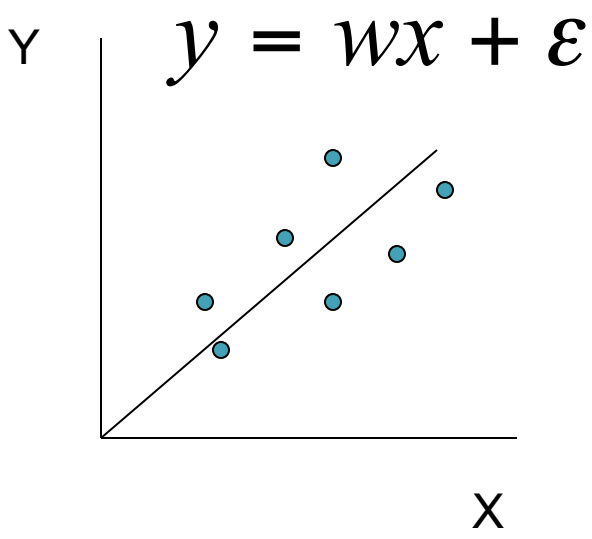
Outline

- **Linear Regression**
  - Simple example
  - Model
- **Learning**
  - Gradient Descent
  - SGD
  - Closed Form
- **Advanced Topics**
  - Geometric and Probabilistic Interpretation of LMS
  - L2 Regularization
  - L1 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
  - L2 Regularization (aka. Ridge Regression)
  - L1 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 $<x_i, y_i>$ 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

Slide courtesy of William Cohen

Solving Linear Regression in 1D

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

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

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

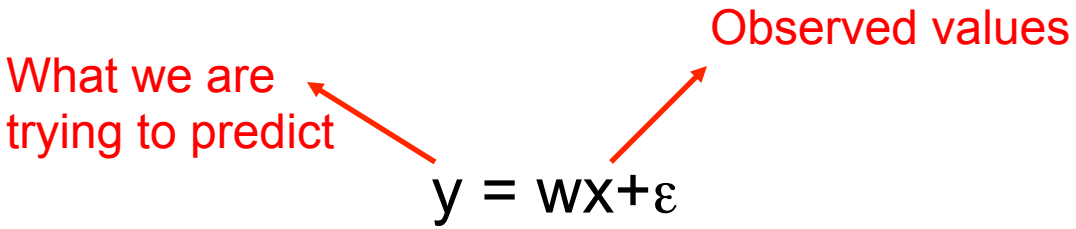
$$\sum_i x_i y_i = \sum_i wx_i^2 \quad \Rightarrow$$

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

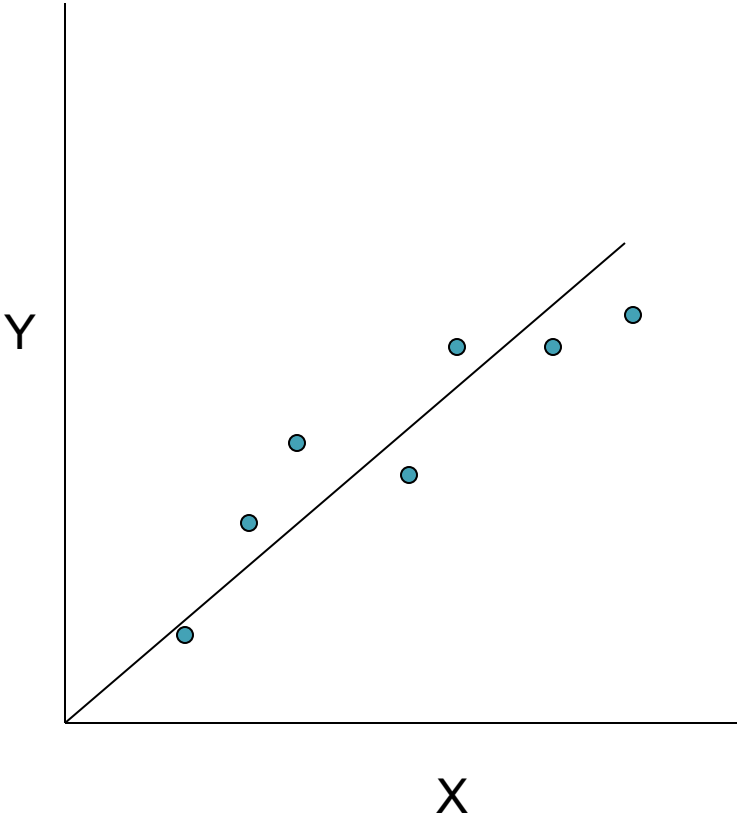
Slide courtesy of William Cohen

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:



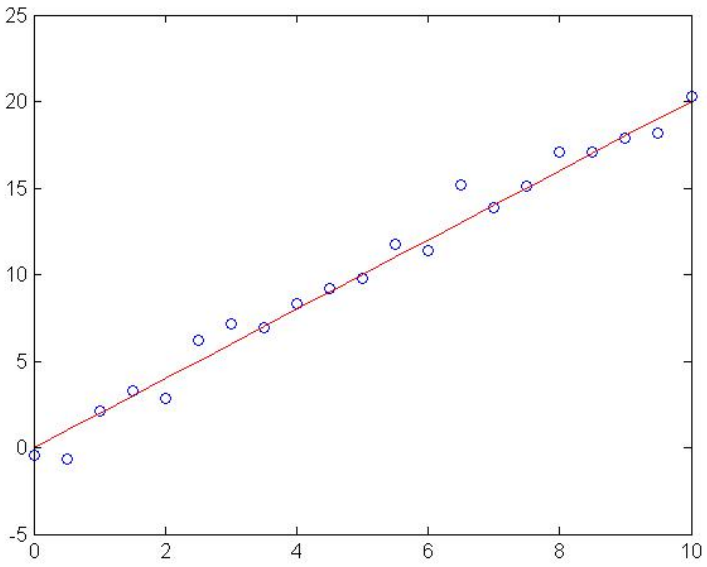
where w is a parameter and ε represents measurement error or other noise



Slide courtesy of William Cohen

Regression example

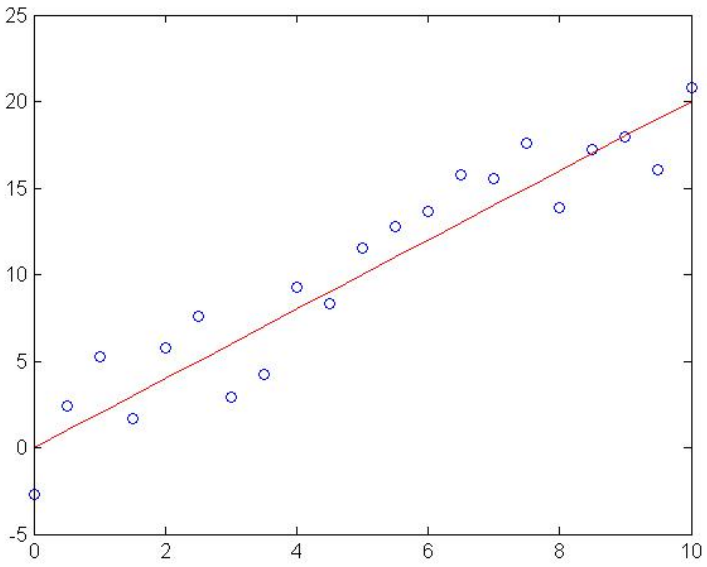
- Generated: w=2
- Recovered: w=2.03
- Noise: std=1



Slide courtesy of William Cohen

Regression example

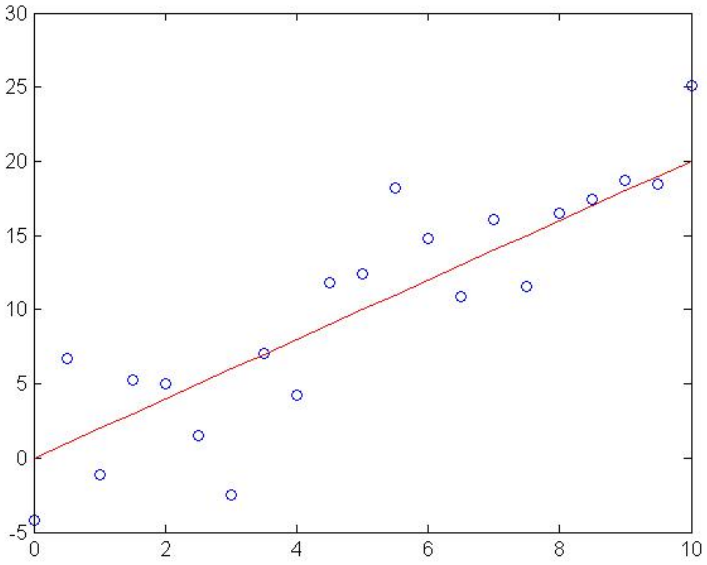
- Generated: w=2
- Recovered: w=2.05
- Noise: std=2



Slide courtesy of William Cohen

Regression example

- Generated: w=2
- Recovered: w=2.08
- Noise: std=4



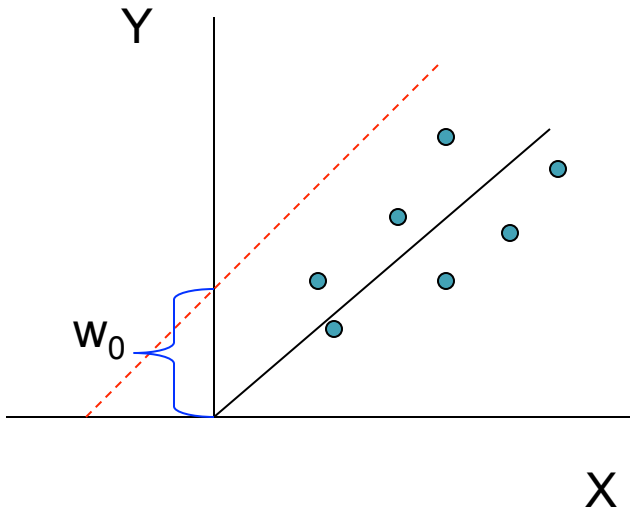
Slide courtesy of William Cohen

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



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

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

Slide courtesy of William Cohen

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.

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

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

(We assume x_1 is 1)

Learning: finds the parameters that minimize some objective function.

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

Least Squares

Learning: finds the parameters that minimize some objective function.

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

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.

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

We minimize the sum of the squares:

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

W

veen true measurements and

line in 1D)

interpretation

This is a very general optimization setup.

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

Least Squares

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

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 $f_{\mathbf{w}}(\mathbf{x}) = \text{sign}(\mathbf{x} \cdot \mathbf{w})$
    - we want to find the classifier **w** that "fits the data best"
  - Formalize as an *optimization problem*
    - Pick a loss function $J(\boldsymbol{w},\boldsymbol{D})$, and find

      $$\text{argmin}_{\mathbf{w}}\, J(\boldsymbol{w})$$

    - OR: Pick a "goodness" function, often $\Pr(D|\mathbf{w})$, and find the $\text{argmax}_{\mathbf{w}}\, f_D(\boldsymbol{w})$

Slide courtesy of William Cohen

Learning as optimization: warmup

Find: optimal (MLE) parameter θ of a binomial

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

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

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

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

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

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

Slide courtesy of William Cohen

Learning as optimization: warmup

Goal: Find the best parameter θ of a binomial

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

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

$\theta = 0$

$\theta = 1$

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

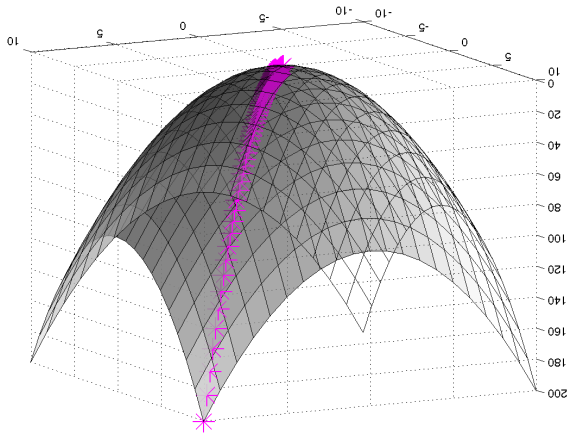
$\rightarrow n\theta = k$

$\rightarrow \theta = k/n$

Slide courtesy of William Cohen

Learning as optimization with gradient ascent

- Goal: Learn the parameter **w** of …
- Dataset: $D=\{(x_1,y_1)\ldots,(x_n, y_n)\}$
- Use your model to define
  - Pr(D|**w**) = ….
- 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



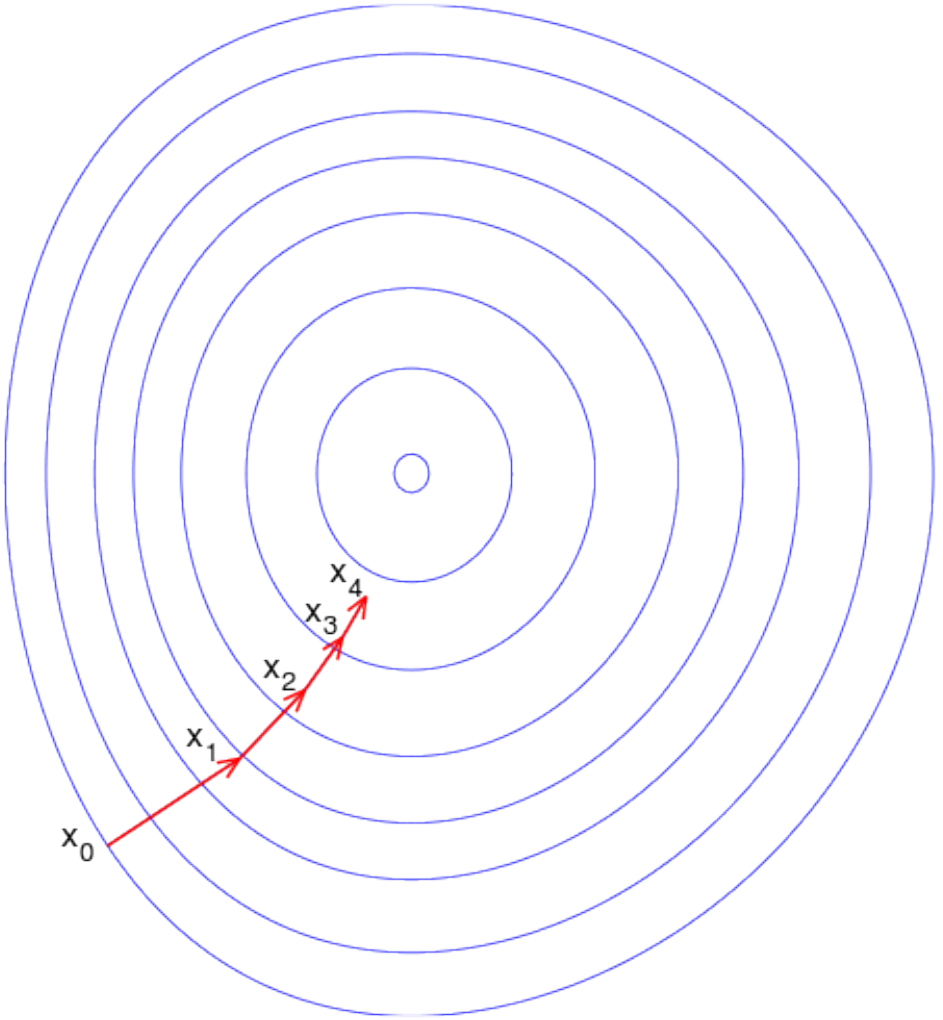
Difference between ascent and descent is only the sign: I'm going to be sloppy here about that.

Slide courtesy of William Cohen

Gradient ascent

To find $\text{argmin}_{\mathbf{x}} f(\mathbf{x})$:

- Start with $\mathbf{x}_0$
- For t=1….
  - $\mathbf{x}_{t+1} = \mathbf{x}_t + \lambda f'(\mathbf{x}_t)$ where λ is small

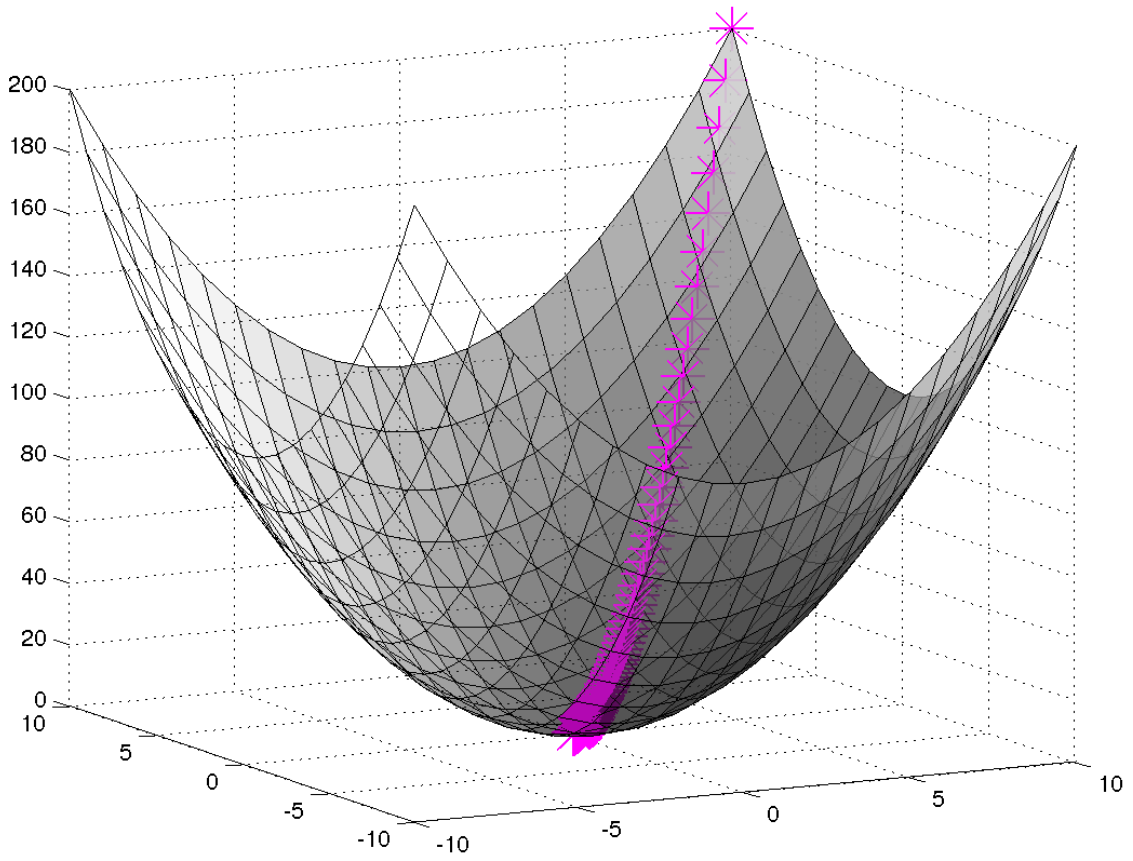


Slide courtesy of William Cohen

Gradient descent

Likelihood: ascent

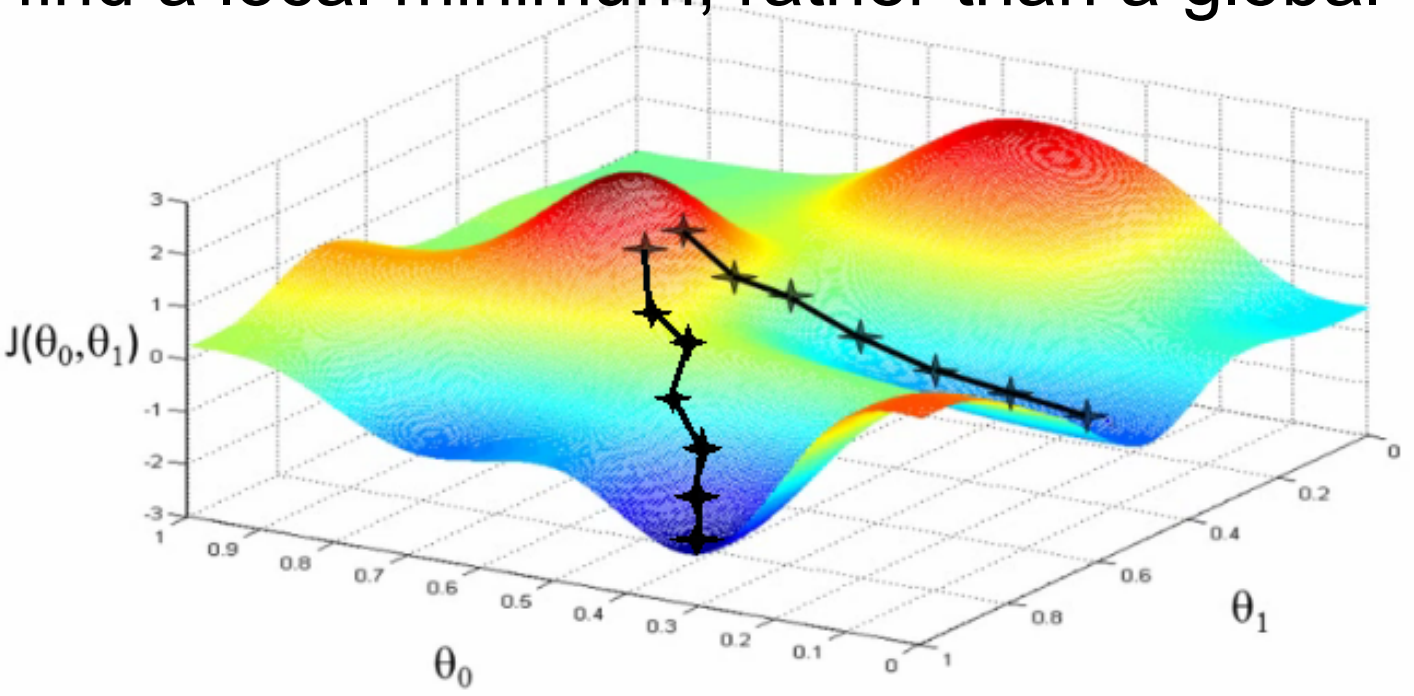
Loss: descent



Slide courtesy of William Cohen

Pros and cons of gradient descent

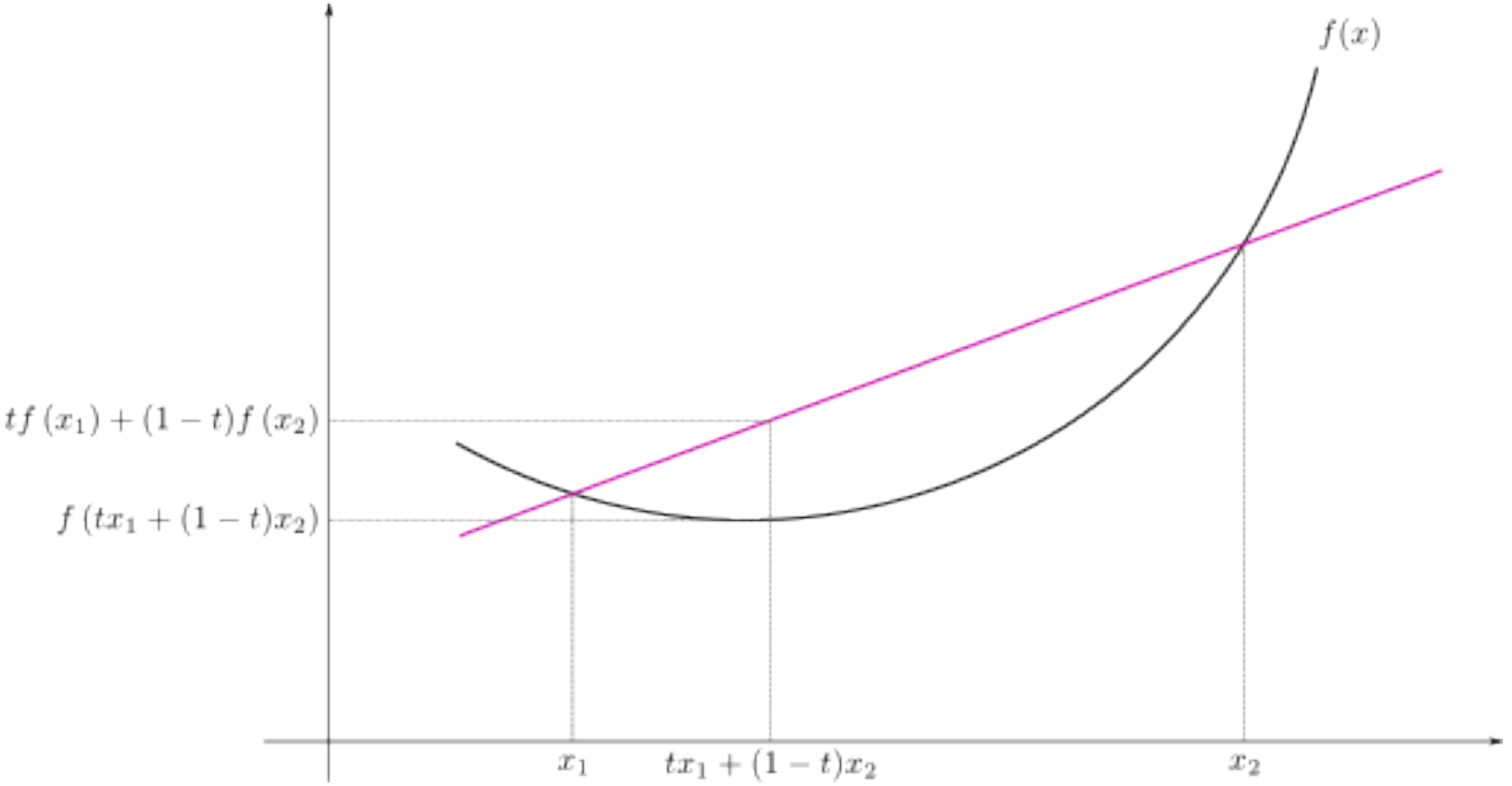
- Simple and often quite effective on ML tasks
- Often very scalable
- Only applies to smooth functions (differentiable)
- Might find a local minimum, rather than a global one



Slide courtesy of William Cohen

Pros and cons of gradient descent

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

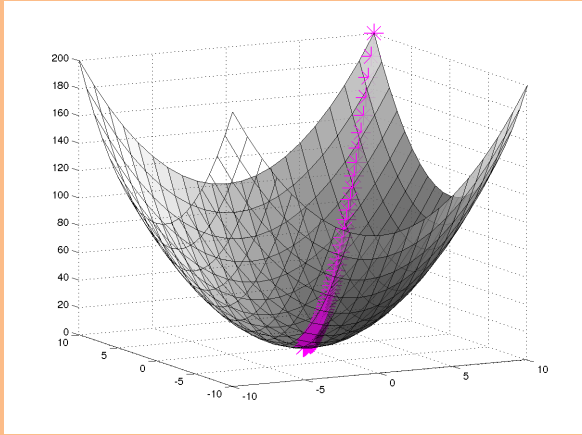


Slide courtesy of William Cohen

Gradient Descent

Algorithm 1 Gradient Descent

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



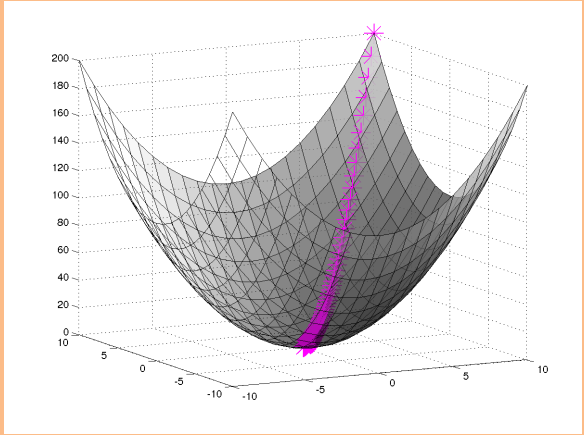
In order to apply GD to Linear Regression all we need is the **gradient** of the objective function (i.e. vector of partial derivatives).

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

Gradient Descent

Algorithm 1 Gradient Descent

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



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.

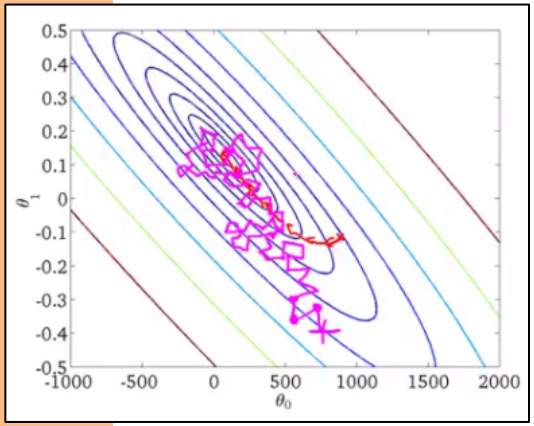
$$||\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})||_2 \leq \epsilon$$

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($\mathcal{D}$, $\boldsymbol{\theta}^{(0)}$)
2: $\quad\boldsymbol{\theta} \leftarrow \boldsymbol{\theta}^{(0)}$
3: $\quad$**while** not converged **do**
4: $\quad\quad$**for** $i \in$ shuffle($\{1, 2, \ldots, N\}$) **do**
5: $\quad\quad\quad\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \lambda \nabla_{\boldsymbol{\theta}} J^{(i)}(\boldsymbol{\theta})$
6: $\quad$**return** $\boldsymbol{\theta}$



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

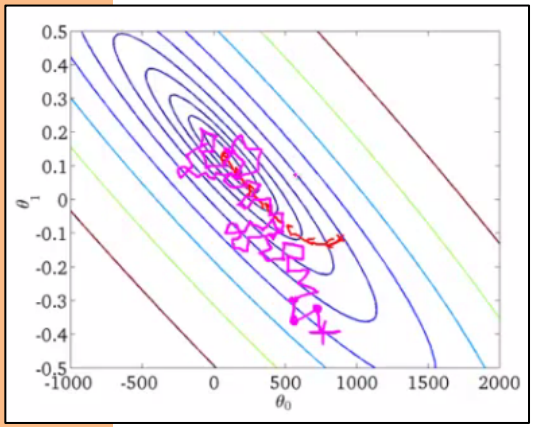
We need a per-example objective:

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

Stochastic Gradient Descent (SGD)

Algorithm 2 Stochastic Gradient Descent (SGD)

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



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

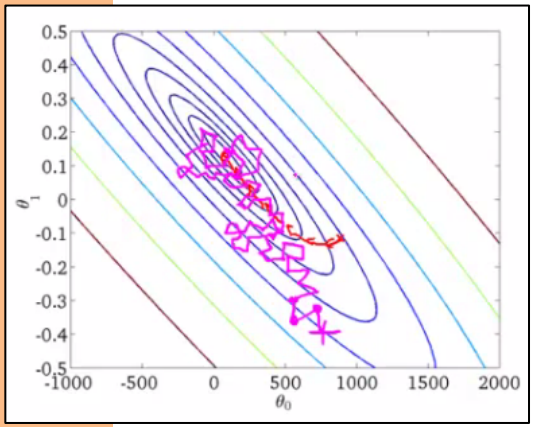
We need a per-example objective:

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

Stochastic Gradient Descent (SGD)

Algorithm 2 Stochastic Gradient Descent (SGD)

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



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

We need a per-example objective:

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

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

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

Partial Derivatives for Linear Reg.

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

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

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

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)

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

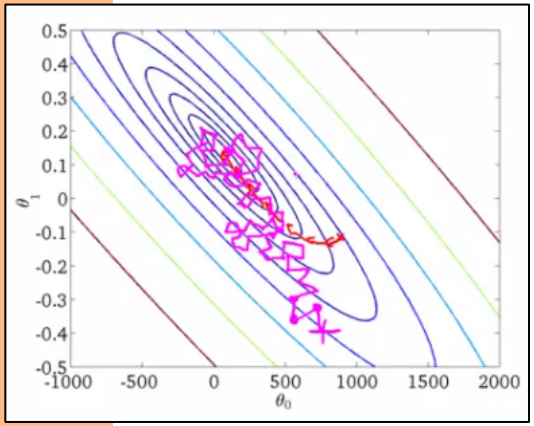
$$= \sum_{i=1}^{N} (\boldsymbol{\theta}^T \mathbf{x}^{(i)} - y^{(i)}) x_k^{(i)}$$

Used by Gradient Descent

Least Mean Squares (LMS)

Algorithm 3 Least Mean Squares (LMS)

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



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

Least Squares

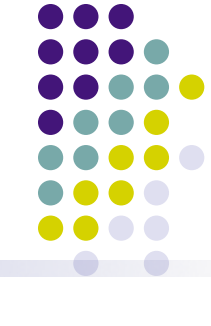
Learning: Three approaches to solving $\boldsymbol{\theta}^* = \underset{\boldsymbol{\theta}}{\operatorname{argmin}}\, J(\boldsymbol{\theta})$

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 $f : \mathrm{R}^{m \times n} \mapsto \mathrm{R}$, define:

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

- Trace:

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

- Some fact of matrix derivatives (without proof)

$$\nabla_A \mathrm{tr}AB = B^T \ , \qquad \nabla_A \mathrm{tr}ABA^T C = CAB + C^T AB^T \ , \qquad \nabla_A |A| = |A| \left(A^{-1}\right)^T$$

The normal equations

- Write the cost function in matrix form:

$$J(\theta) = \frac{1}{2}\sum_{i=1}^{n}(\mathbf{x}_i^T\theta - y_i)^2$$
$$= \frac{1}{2}(X\theta - \vec{y})^T(X\theta - \vec{y})$$
$$= \frac{1}{2}\left(\theta^T X^T X\theta - \theta^T X^T\vec{y} - \vec{y}^T X\theta + \vec{y}^T\vec{y}\right)$$

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

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

$$\nabla_\theta J = \frac{1}{2}\nabla_\theta \mathrm{tr}\left(\theta^T X^T X\theta - \theta^T X^T\vec{y} - \vec{y}^T X\theta + \vec{y}^T\vec{y}\right)$$
$$= \frac{1}{2}\left(\nabla_\theta \mathrm{tr}\,\theta^T X^T X\theta - 2\nabla_\theta \mathrm{tr}\,\vec{y}^T X\theta + \nabla_\theta \mathrm{tr}\,\vec{y}^T\vec{y}\right)$$
$$= \frac{1}{2}\left(X^T X\theta + X^T X\theta - 2X^T\vec{y}\right)$$
$$= X^T X\theta - X^T\vec{y} = 0$$

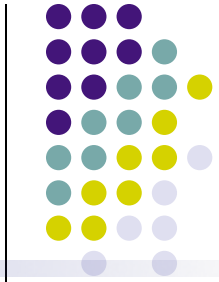
$$\Rightarrow \boxed{X^T X\theta = X^T\vec{y}}$$

The normal equations

$$\Downarrow$$

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

Comments on the normal equation

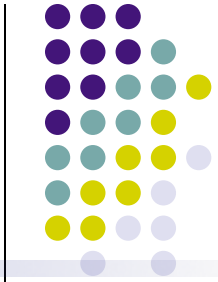


- In most situations of practical interest, the number of data points N is larger than the dimensionality k of the input space and the matrix $\mathbf{X}$ is of full column rank. If this condition holds, then it is easy to verify that $X^T X$ is necessarily invertible.

- The assumption that $X^T X$ 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? → regularization (later).

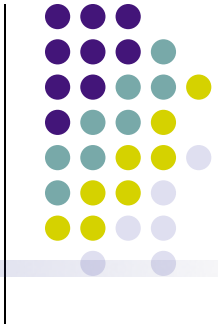
Direct and Iterative methods



- 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 α

Convergence rate



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

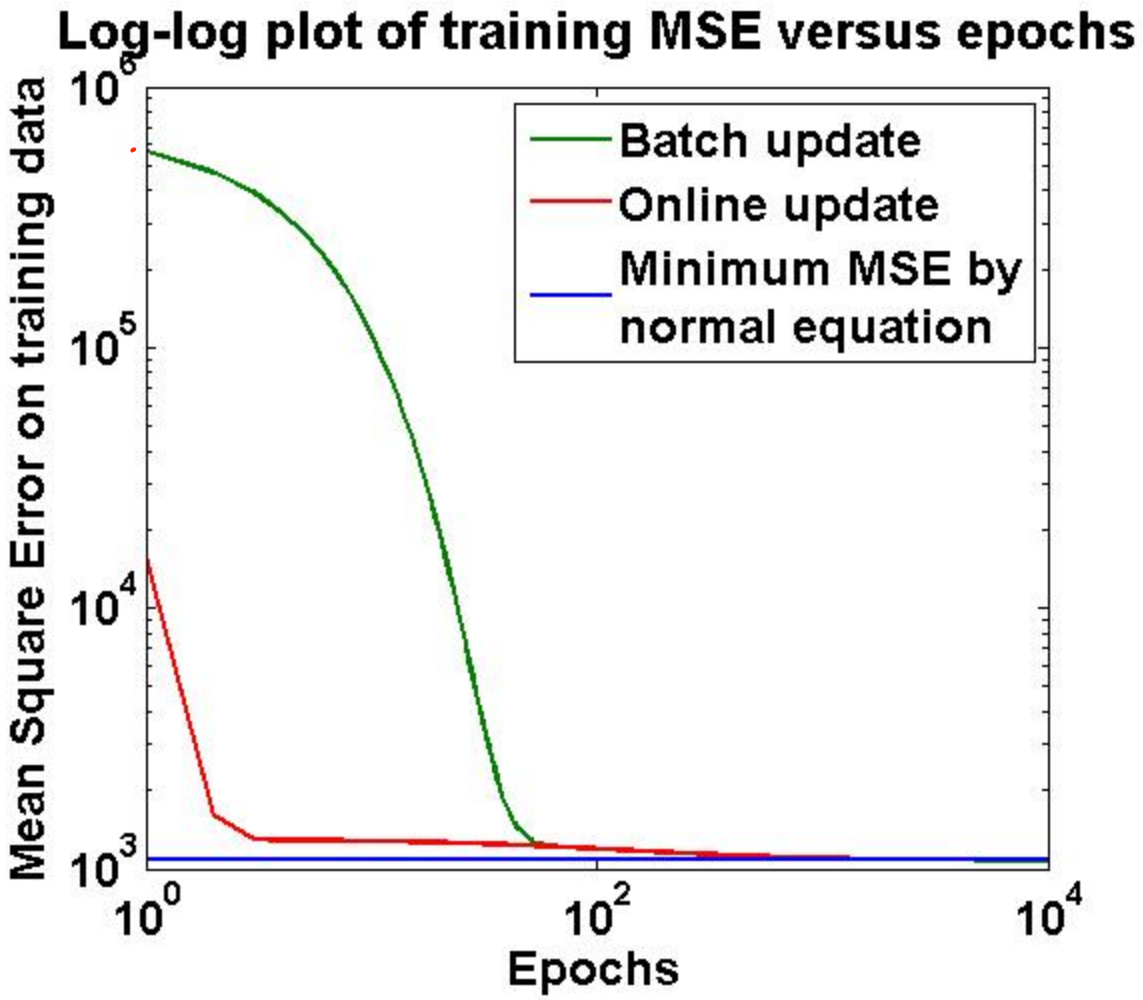
$$\theta^{(\infty)} = (X^T X)^{-1} X^T y$$

If

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

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

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

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:

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

- Note that

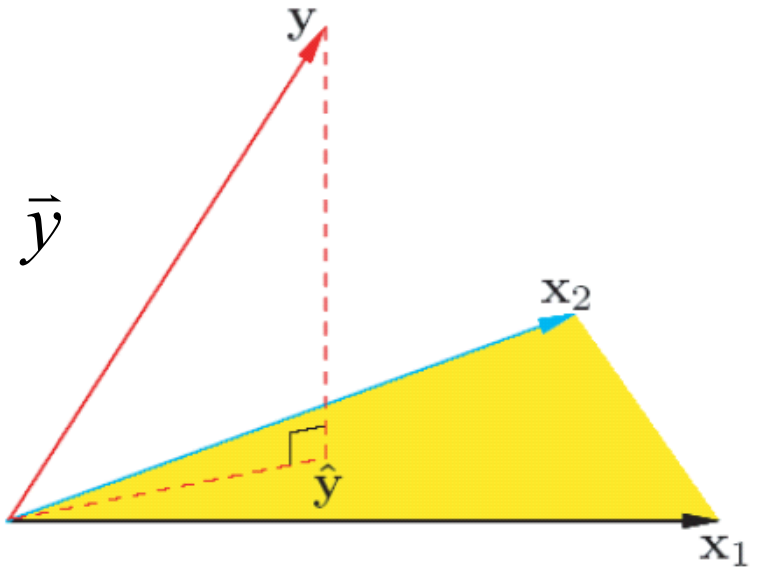
$$\hat{\vec{y}} - \vec{y} = \left(X\left(X^T X\right)^{-1} X^T - I\right)\vec{y}$$

and

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

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

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

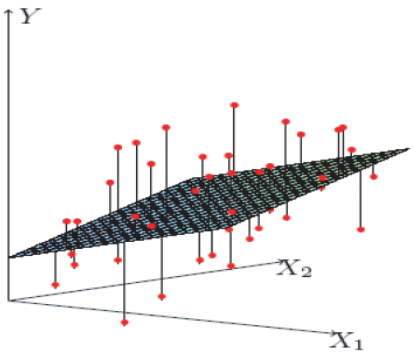


Probabilistic Interpretation of LMS

- 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 ε is an error term of unmodeled effect noise



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

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

- By independence assumption:

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

Probabilistic Interpretation of LMS

- Hence the log-likelihood is:

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

- 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 θ !

Non-Linear basis function

- So far we only used the observed values $x_1, x_2, \ldots$
- 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, …. 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

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

Slide courtesy of William Cohen

Non-linear basis functions

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

- Polynomial: $\phi_j(x) = x^j$ for j=0 … 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.

Slide courtesy of William Cohen

General linear regression problem

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

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

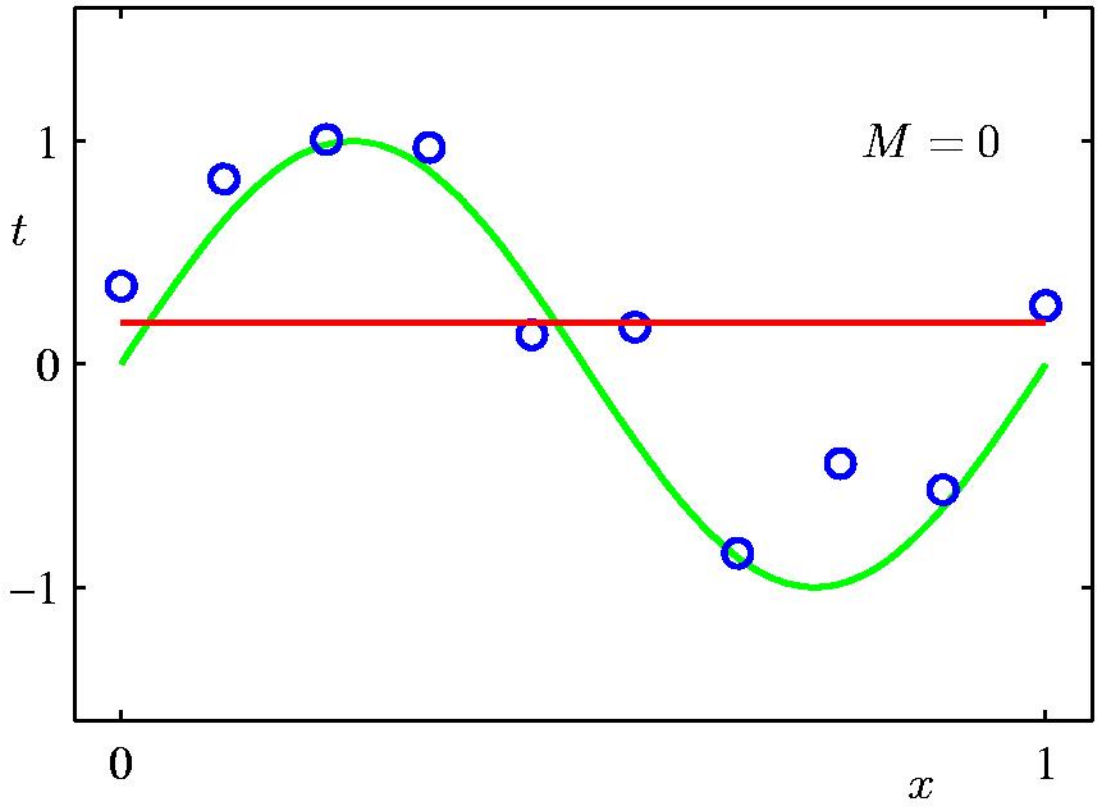
- Where $\phi_j(\mathbf{x})$ can be either x_j for multivariate regression or one of the non-linear basis functions we defined
- … and $\phi_0(\mathbf{x})=1$ for the intercept term

Slide courtesy of William Cohen

An example: polynomial basis vectors on a small dataset

– From Bishop Ch 1

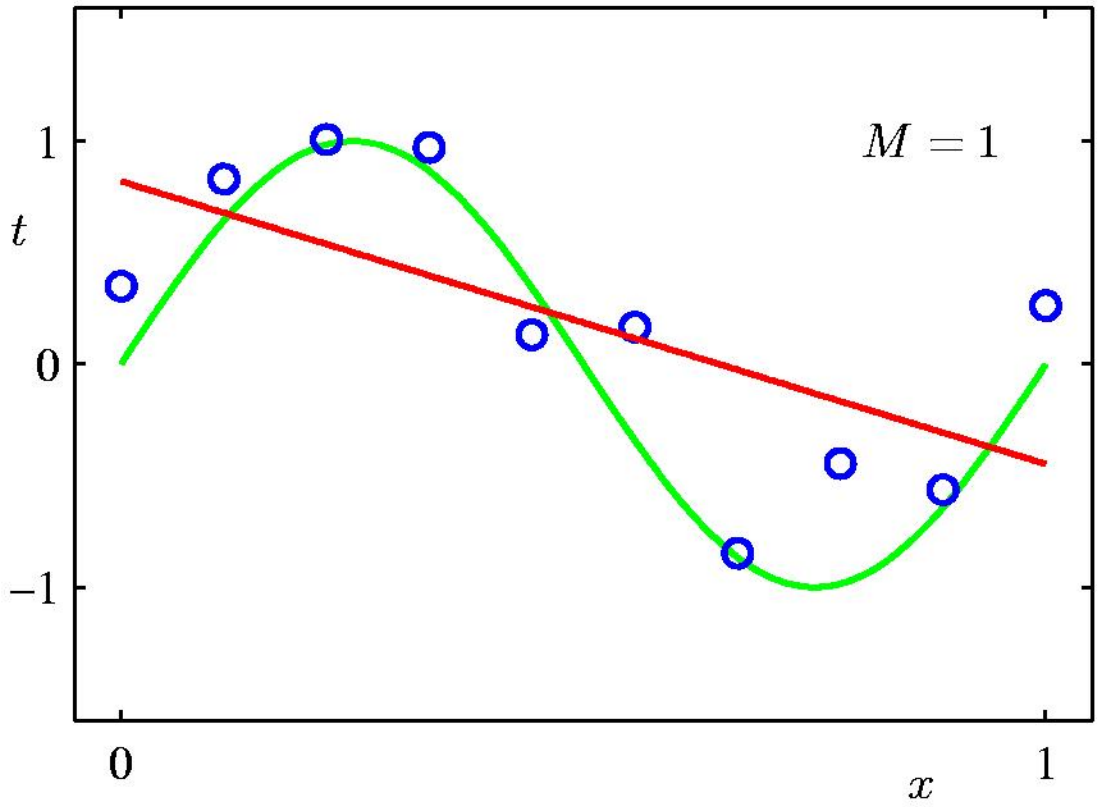
0^{th} Order Polynomial



n=10

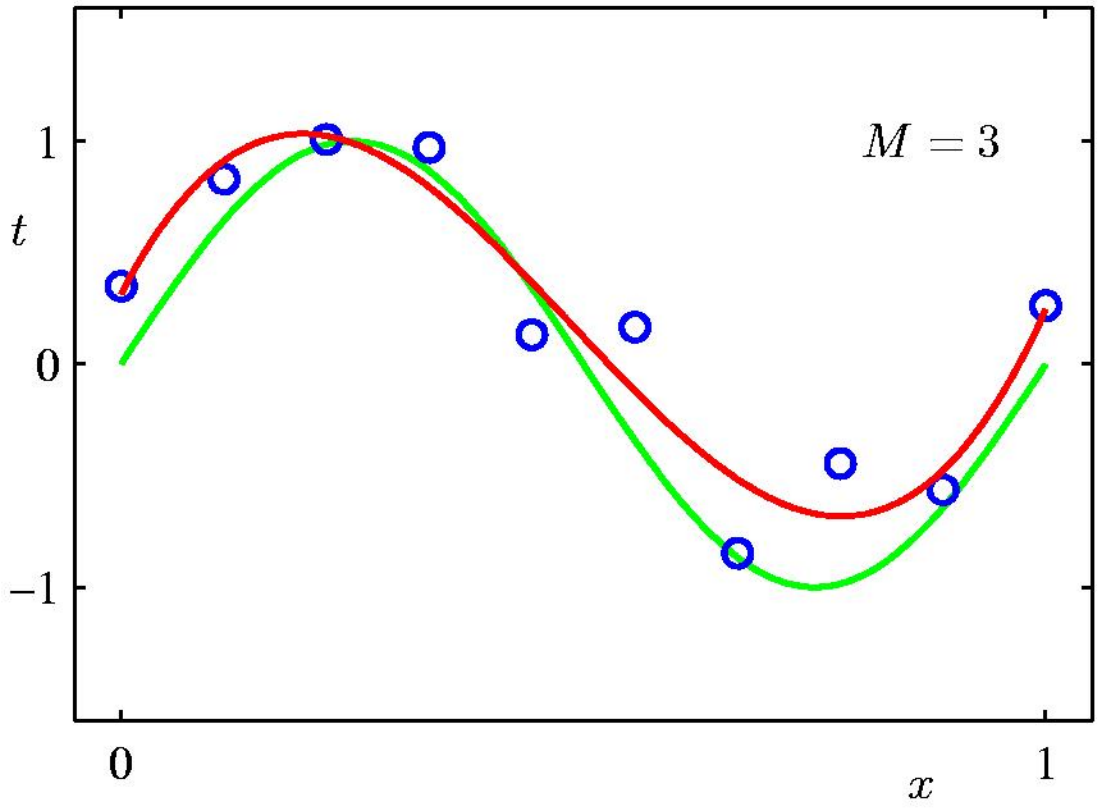
Slide courtesy of William Cohen

1st Order Polynomial



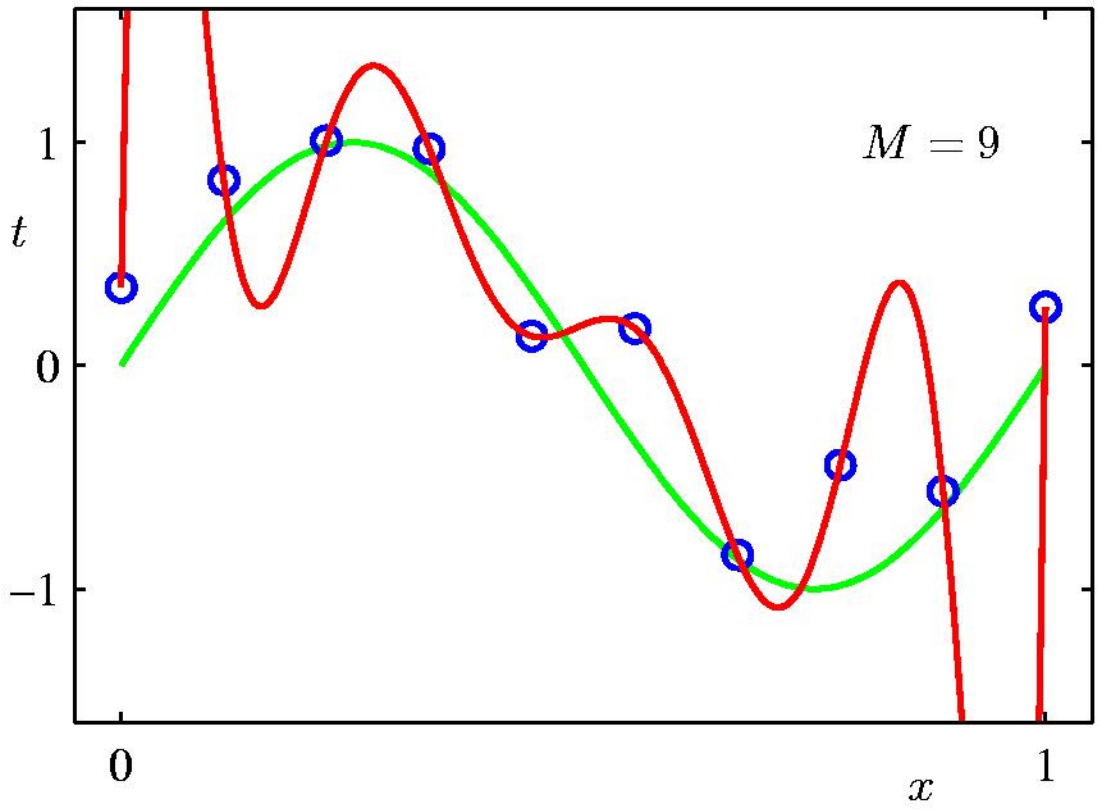
Slide courtesy of William Cohen

3rd Order Polynomial



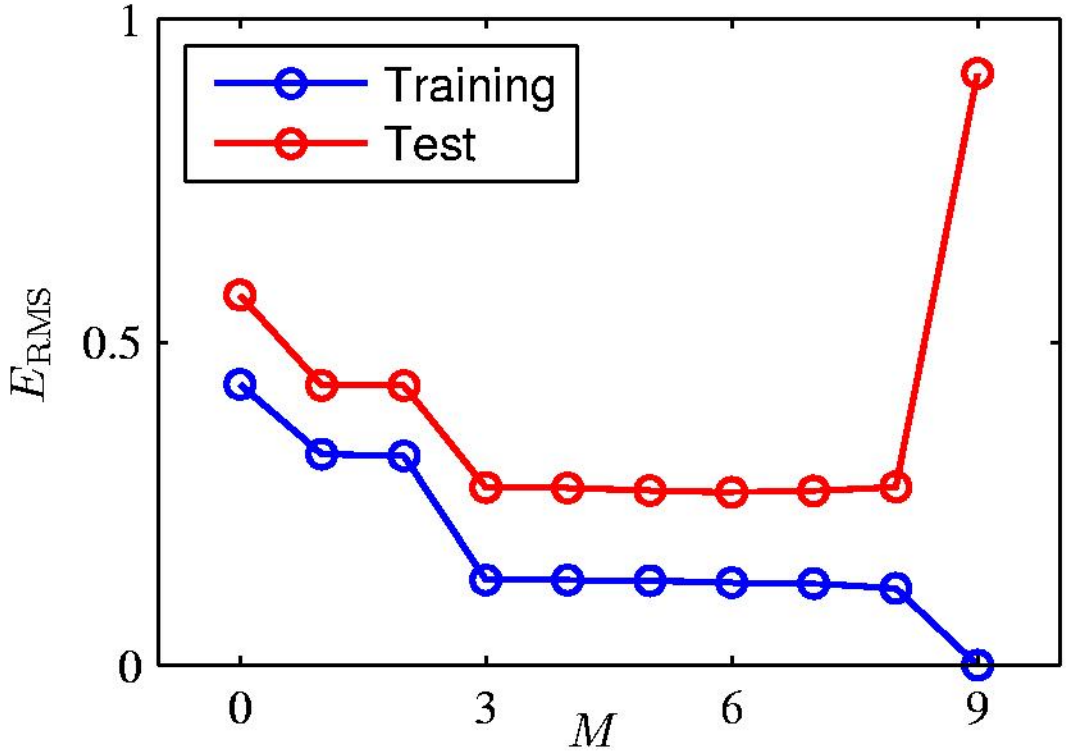
Slide courtesy of William Cohen

9th Order Polynomial



Slide courtesy of William Cohen

Over-fitting



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

Slide courtesy of William Cohen

Polynomial Coefficients

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

Slide courtesy of William Cohen

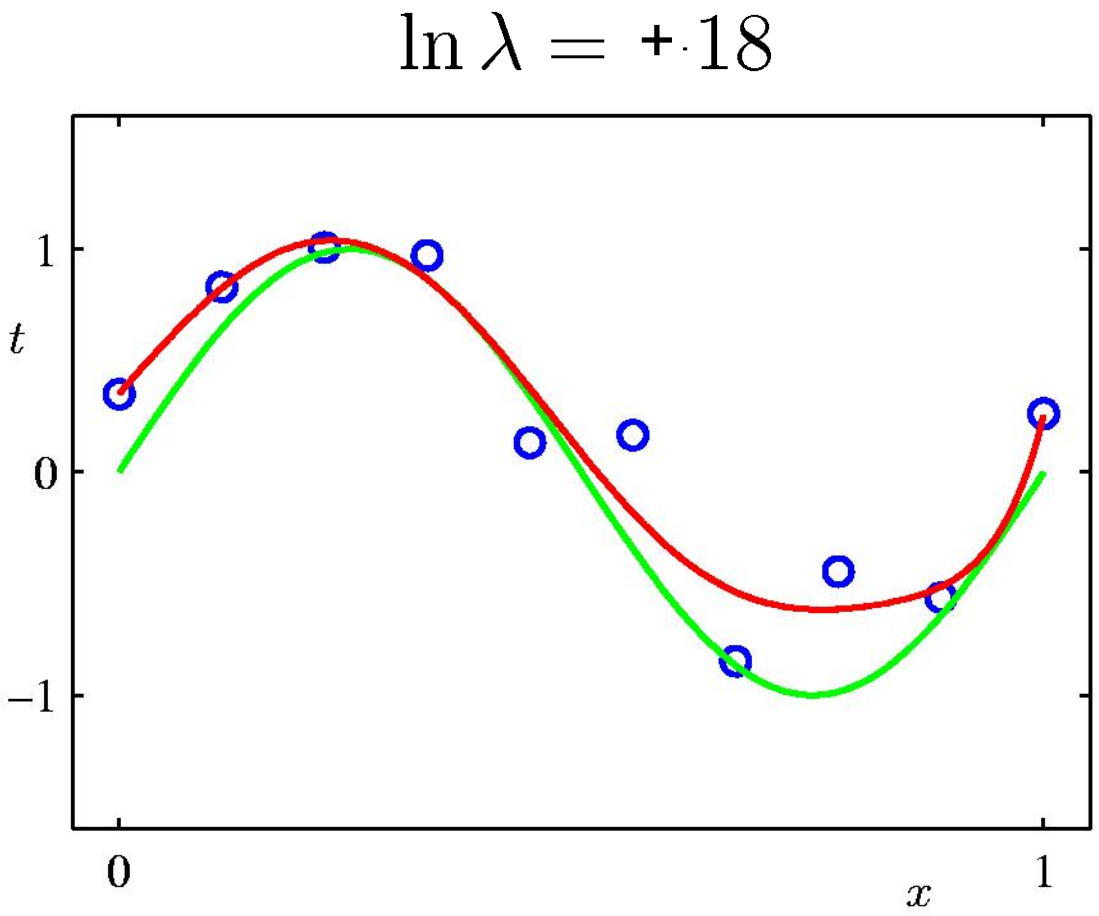
Regularization

Penalize large coefficient values

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

Slide courtesy of William Cohen

Regularization:



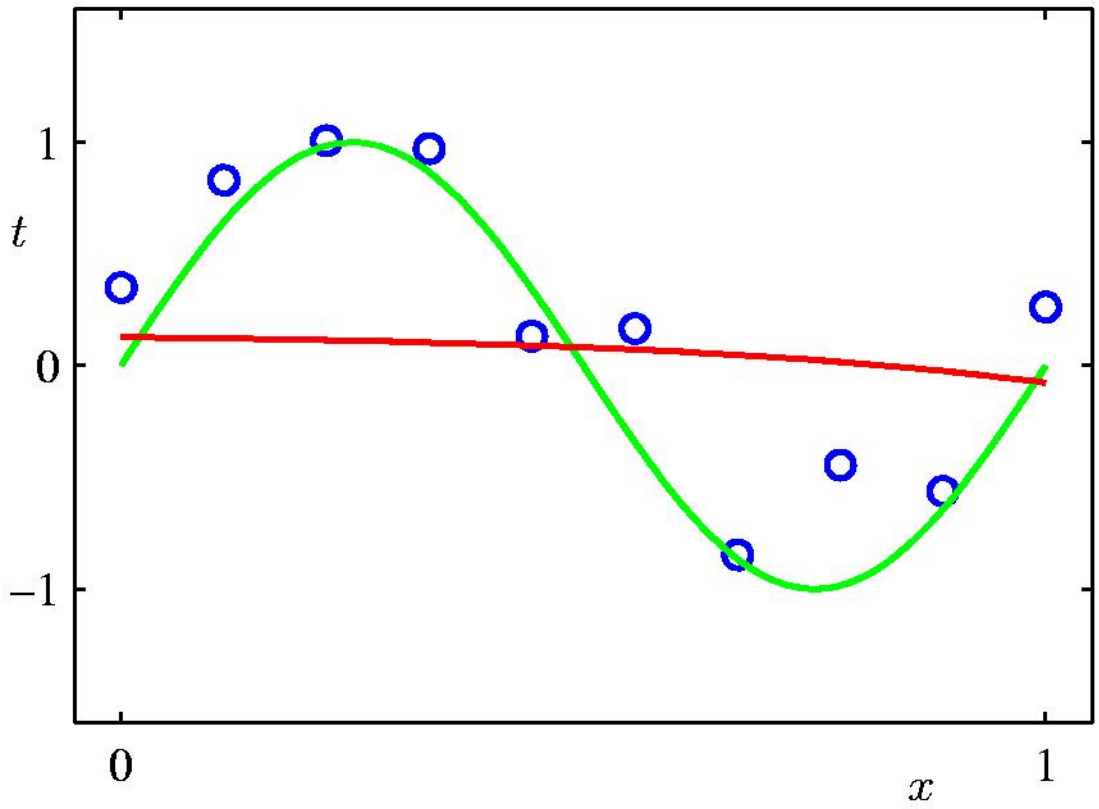
Slide courtesy of William Cohen

Polynomial Coefficients

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

Slide courtesy of William Cohen

Over Regularization:



Slide courtesy of William Cohen

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

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

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

prefers parameters close to zero

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

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

$$= \underset{\boldsymbol{\theta}}{\text{argmax}}\, J_{\text{RR}}(\boldsymbol{\theta})$$

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

LASSO

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

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

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

yields sparse parameters (exact zeros)

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

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

$$= \underset{\boldsymbol{\theta}}{\text{argmax}}\, J_{\text{LASSO}}(\boldsymbol{\theta})$$

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

Ridge Regression vs Lasso

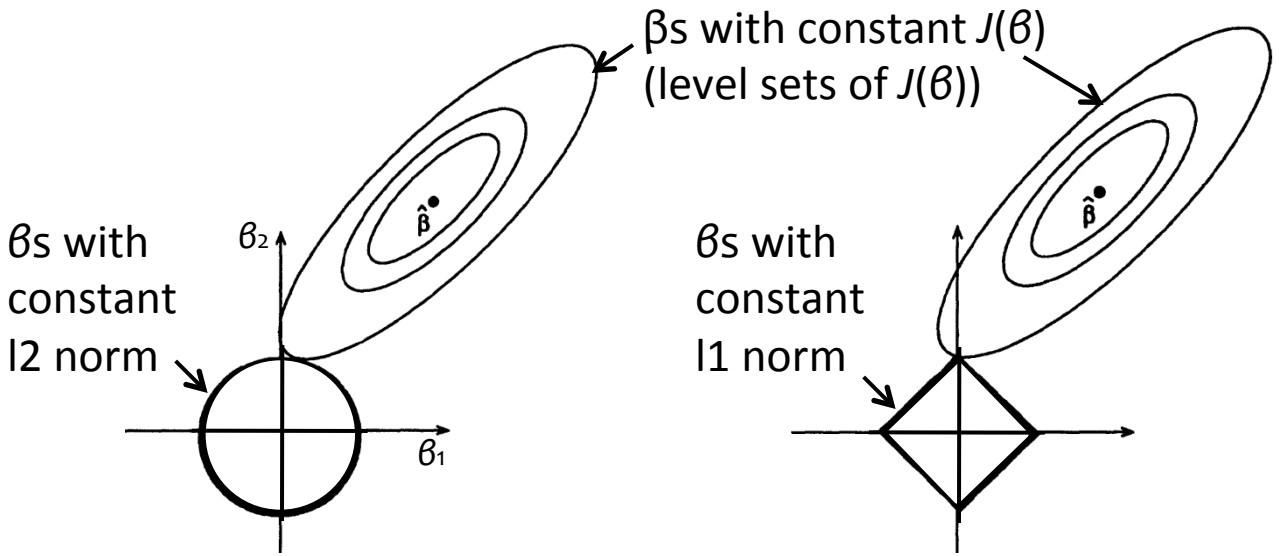
$$\min_{\beta}(\mathbf{X}\beta-\mathbf{Y})^T(\mathbf{X}\beta-\mathbf{Y})+\lambda \text{pen}(\beta)=\min_{\beta} J(\beta)+\lambda \text{pen}(\beta)$$

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?

$$\underset{\boldsymbol{\theta}}{\operatorname{argmax}}\, J_{\text{LASSO}}(\boldsymbol{\theta})$$

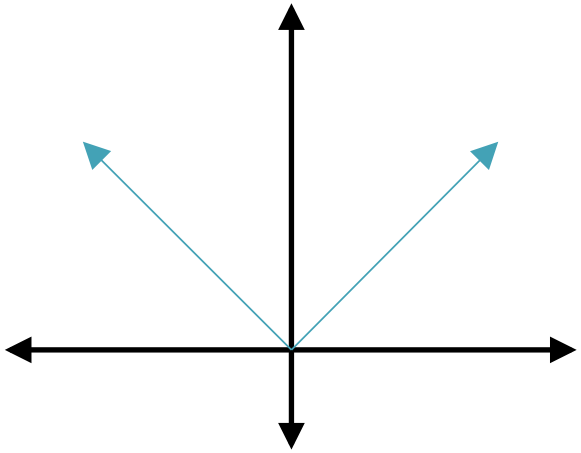
$$J_{\text{LASSO}}(\boldsymbol{\theta}) = J(\boldsymbol{\theta}) + \lambda||\boldsymbol{\theta}||_1$$

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

Optimization for LASSO

- Consider the absolute value function:

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



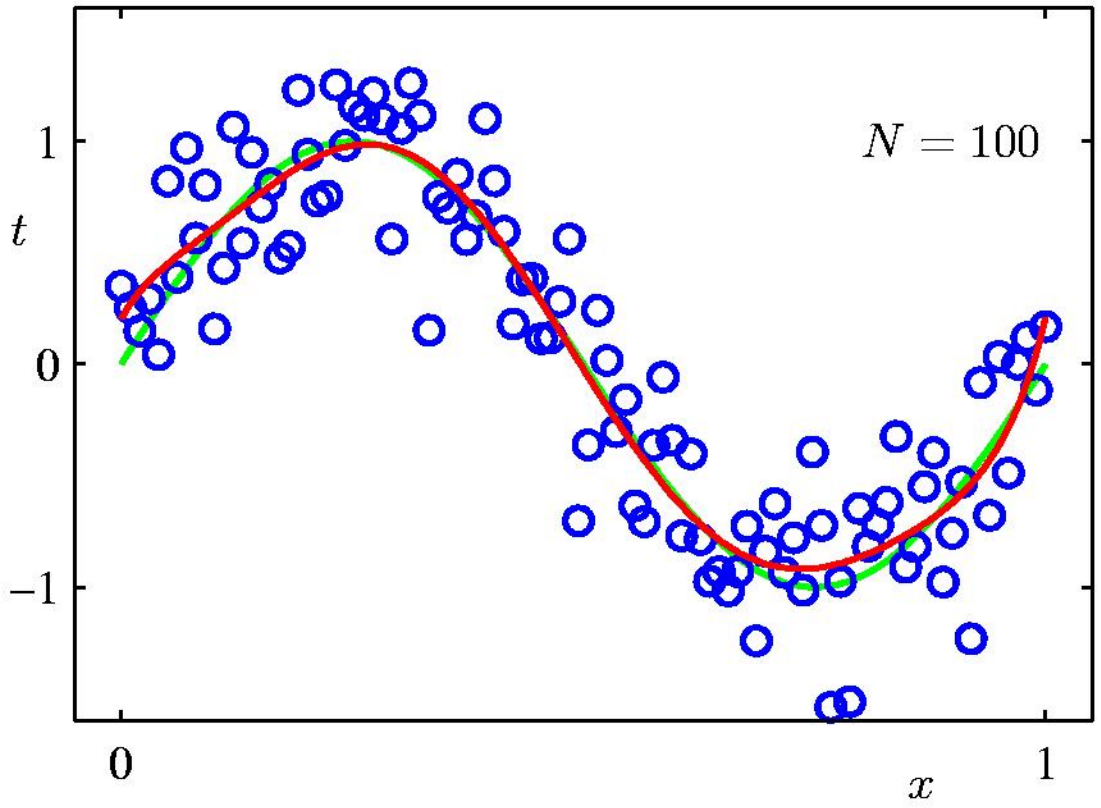
- The L1 penalty is subdifferentiable (i.e. not differentiable at 0)
- Where does a small step opposite the gradient accomplish…
  - … far from zero?
  - … close to zero?

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
  - Othant-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:

9th Order Polynomial $N = 100$



Slide courtesy of William Cohen

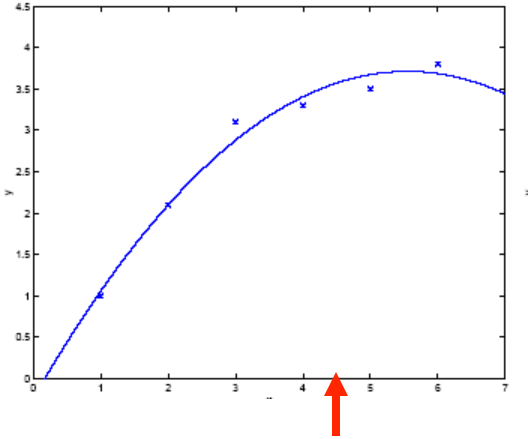
Locally weighted linear regression

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

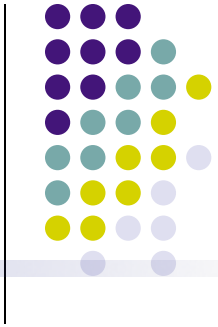
  Where do w_i's come from? $w_i = \exp\left(-\frac{(\mathbf{x}_i - \mathbf{x})^2}{2\tau^2}\right)$



  - 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)

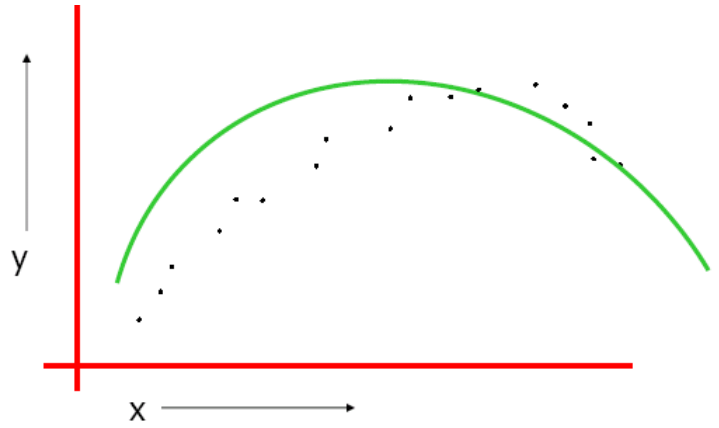
Parametric vs. non-parametric



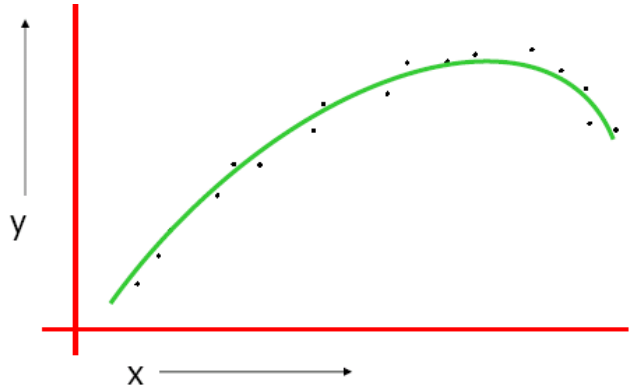
- Locally weighted linear regression is the second example we are running into of a **non-parametric** algorithm. (what is the first?)
- The (unweighted) linear regression algorithm that we saw earlier is known as a **parametric** learning algorithm
  - because it has a fixed, finite number of parameters (the θ), which are fit to the data;
  - Once we've fit the θ 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 …

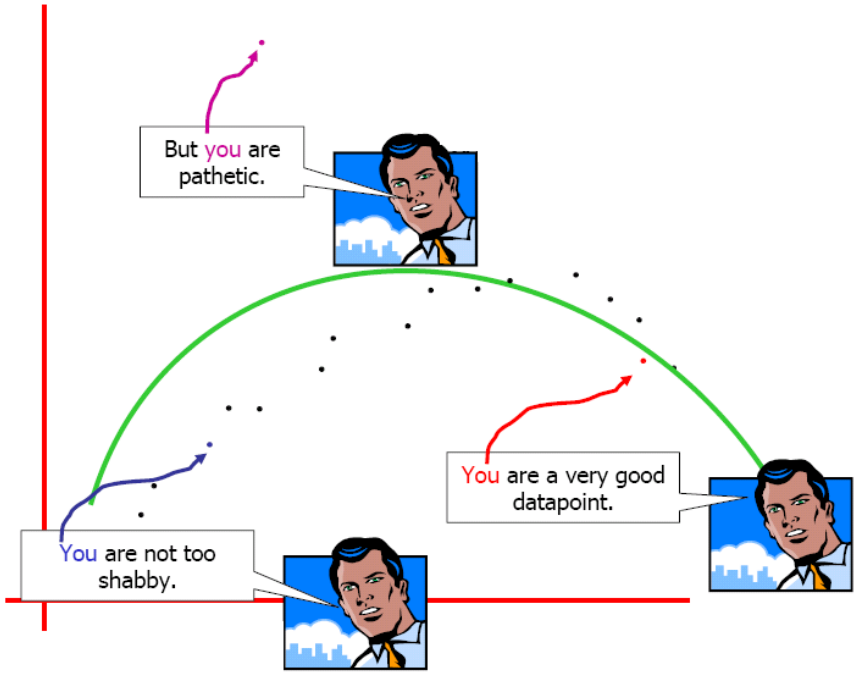


How can we do this?

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"

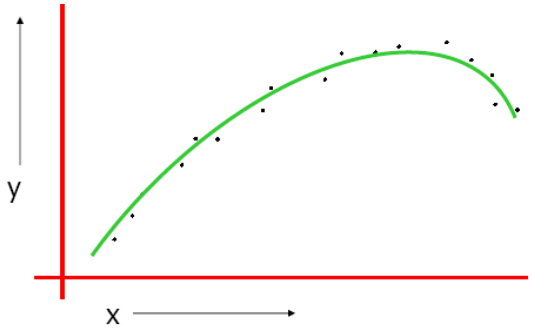


(Courtesy to Andrew Moore)

Robust regression

- For k = 1 to R…
  - Let (x_k, y_k) be the kth datapoint
  - Let y^{est}_k 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:

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



- 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 θ

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 $$y_k \sim \mathcal{N}(\mu, \sigma^2_{\text{huge}})$$

  Computational task is to find the Maximum Likelihood estimates of θ, p, μ and σ_{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**