10-601 Introduction to Machine Learning

Machine Learning Department
School of Computer Science
Carnegie Mellon University

Optimization for ML

+

Linear Regression

Optimization Readings:

Lecture notes from 10-600 (see Piazza note)

"Convex Optimization" Boyd and Vandenberghe (2009) [See Chapter 9. This advanced reading is entirely optional.]

Linear Regression Readings:

Murphy 7.1 – 7.3

Bishop 3.1

HTF 3.1 – 3.4

Mitchell 4.1-4.3

Matt Gormley

Lecture 7

February 8, 2016

Reminders

- **Homework 2: Naive Bayes**
  - Release: Wed, Feb. 1
  - Due: Mon, Feb. 13 at 5:30pm
- **Homework 3: Linear / Logistic Regression**
  - Release: Mon, Feb. 13
  - Due: Wed, Feb. 22 at 5:30pm

Optimization Outline

- **Optimization for ML**
  - Differences
  - Types of optimization problems
  - Unconstrained optimization
  - Convex, concave, nonconvex
- **Optimization: Closed form solutions**
  - Example: 1-D function
  - Example: higher dimensions
  - Gradient and Hessian
- **Gradient Descent**
  - Example: 2D gradients
  - Algorithm
  - Details: starting point, stopping criterion, line search
- **Stochastic Gradient Descent (SGD)**
  - Expectations of gradients
  - Algorithm
  - Mini-batches
  - Details: mini-batches, step size, stopping criterion
  - Problematic cases for SGD
- **Convergence**
  - Comparison of Newton's method, Gradient Descent, SGD
  - Asymptotic convergence
  - Convergence in practice

Optimization for ML

Not quite the same setting as other fields…

- Function we are optimizing might not be the true goal
  (e.g. likelihood vs generalization error)
- Precision might not matter
  (e.g. data is noisy, so optimal up to 1e-16 might not help)
- Stopping early can help generalization error
  (i.e. "early stopping" is a technique for regularization – discussed more next time)

Optimization for ML

Whiteboard

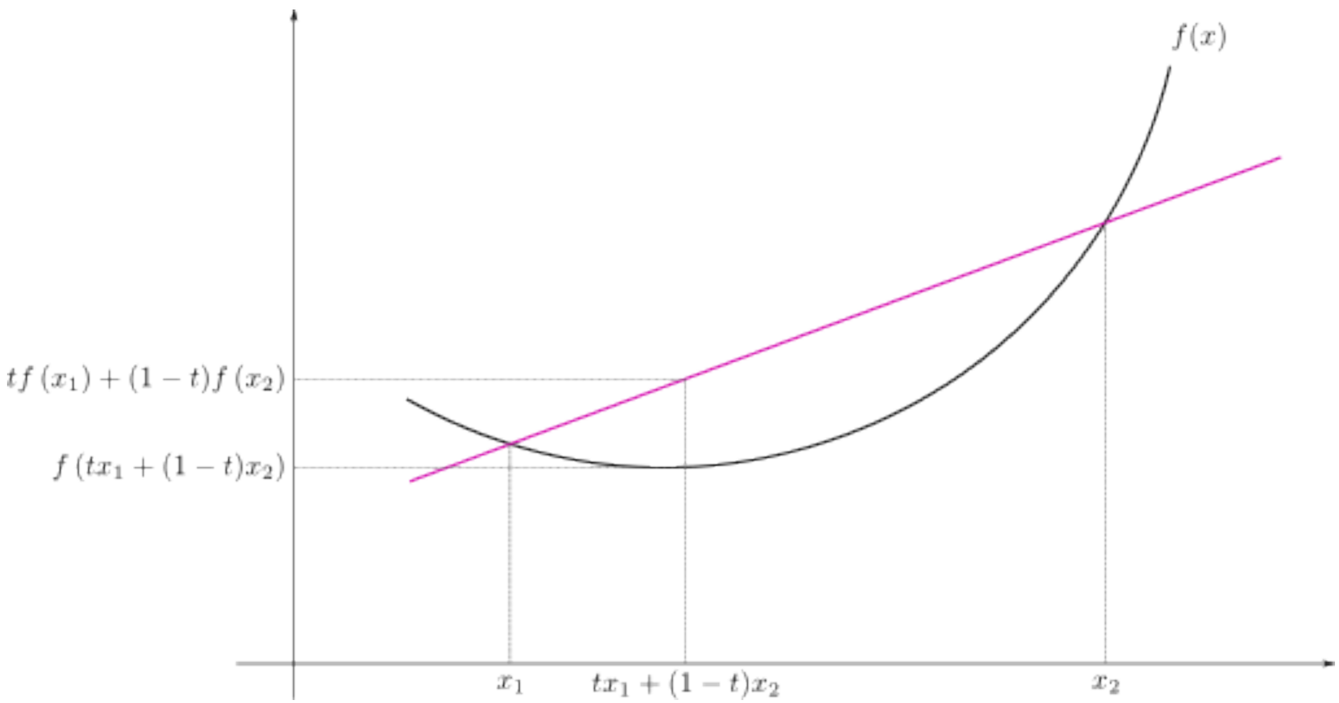
- Differences
- Types of optimization problems
- Unconstrained optimization
- Convex, concave, nonconvex

Convexity

Function $f : \mathbb{R}^M \rightarrow \mathbb{R}$ is **convex**
if $\forall\ \mathbf{x}_1 \in \mathbb{R}^M, \mathbf{x}_2 \in \mathbb{R}^M, 0 \leq t \leq 1$:

$$f(t\mathbf{x}_1 + (1-t)\mathbf{x}_2) \leq tf(\mathbf{x}_1) + (1-t)f(\mathbf{x}_2)$$

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



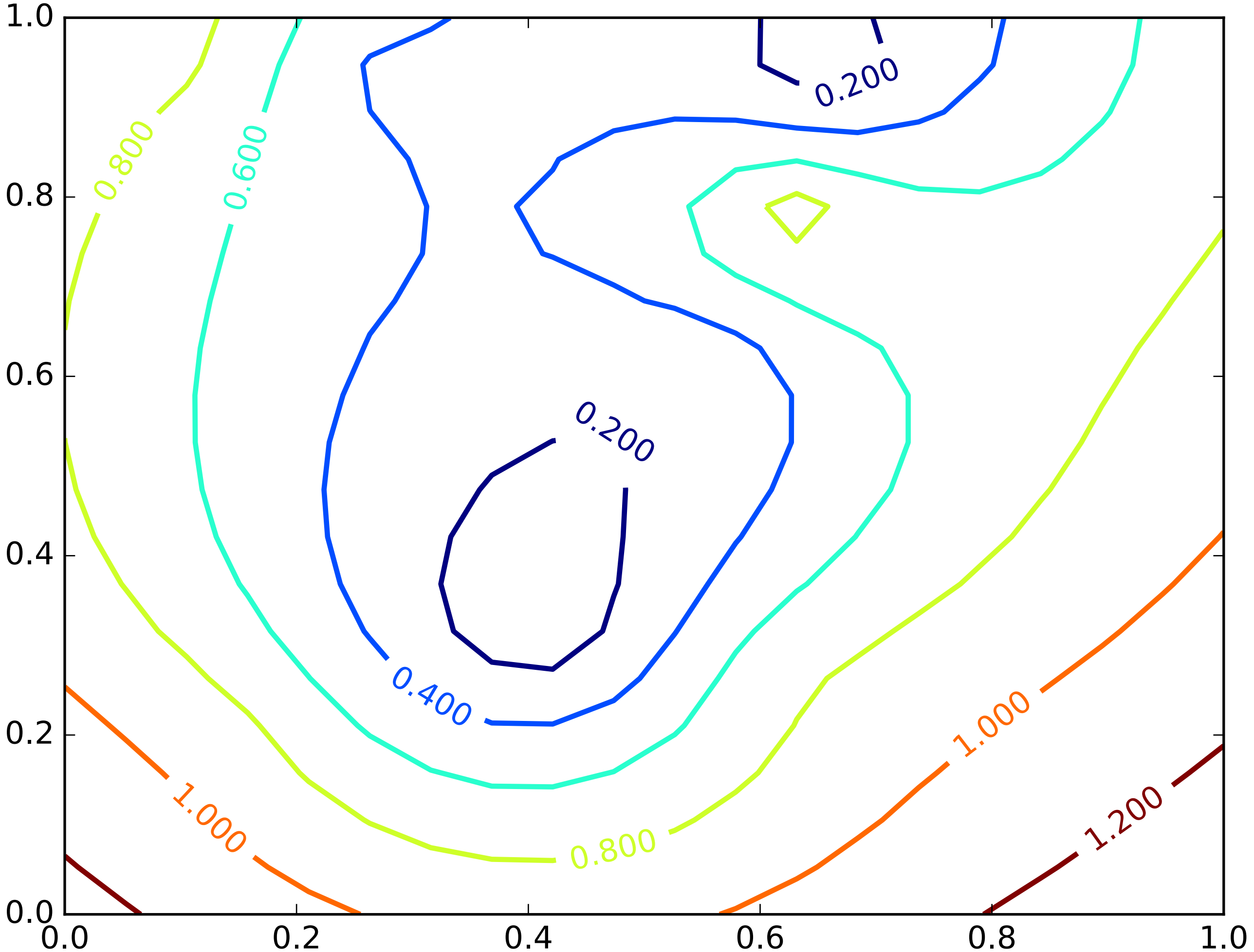
Slide adapted from William Cohen

Optimization: Closed form solutions

Whiteboard

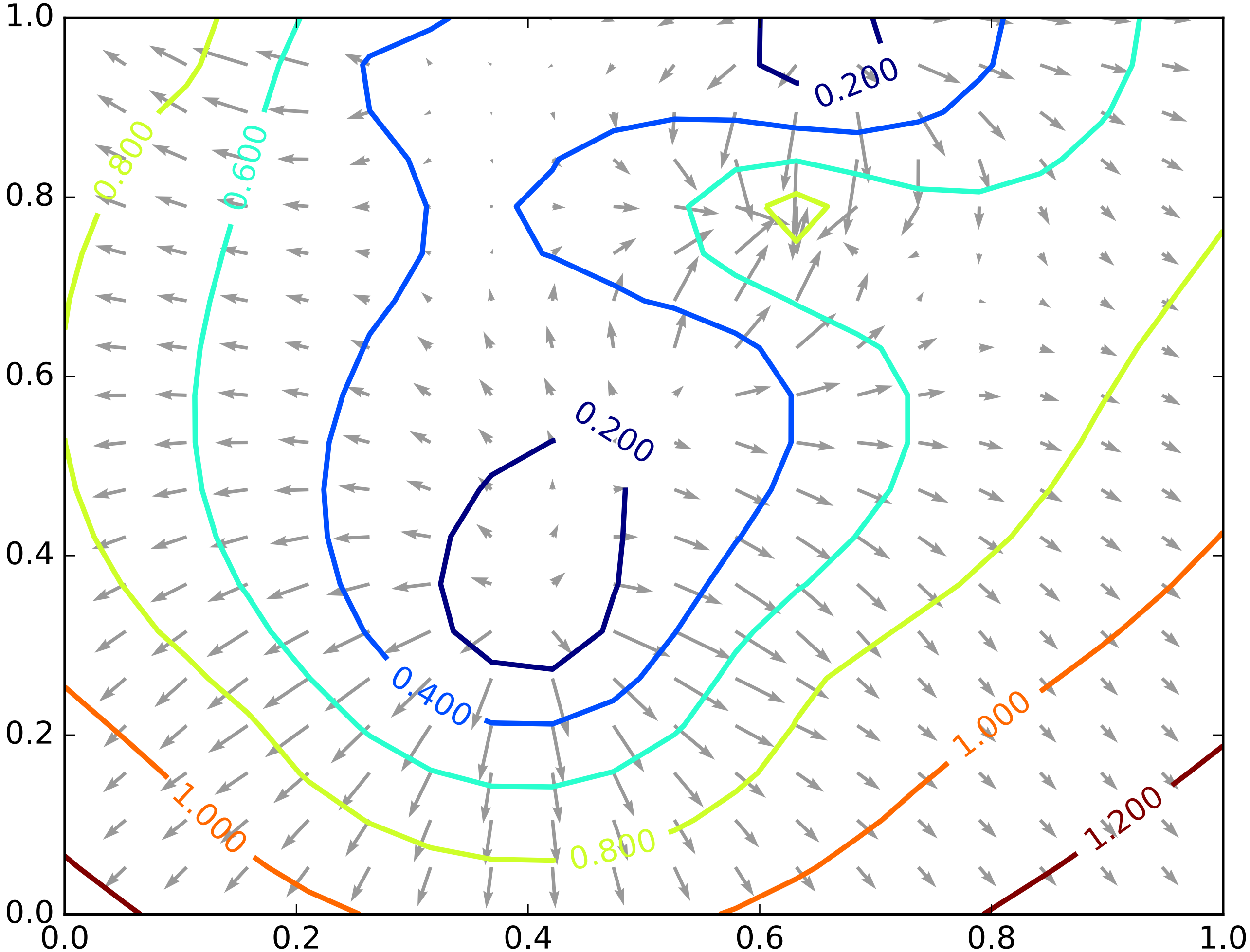
- Example: 1-D function
- Example: higher dimensions
- Gradient and Hessian

Gradients



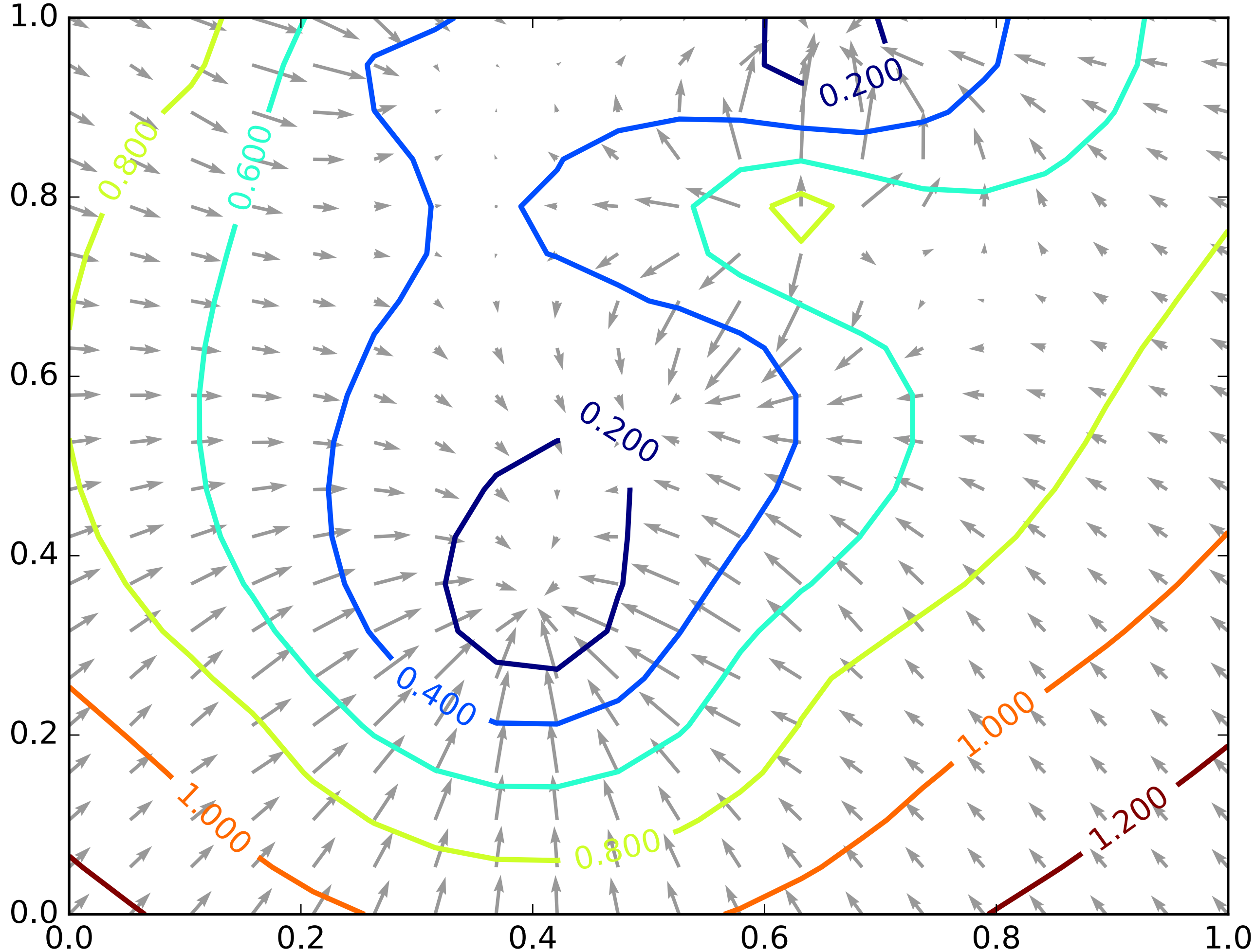
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0
0.200
0.200
0.400
0.600
0.800
0.800
1.000
1.000
1.200

Gradients



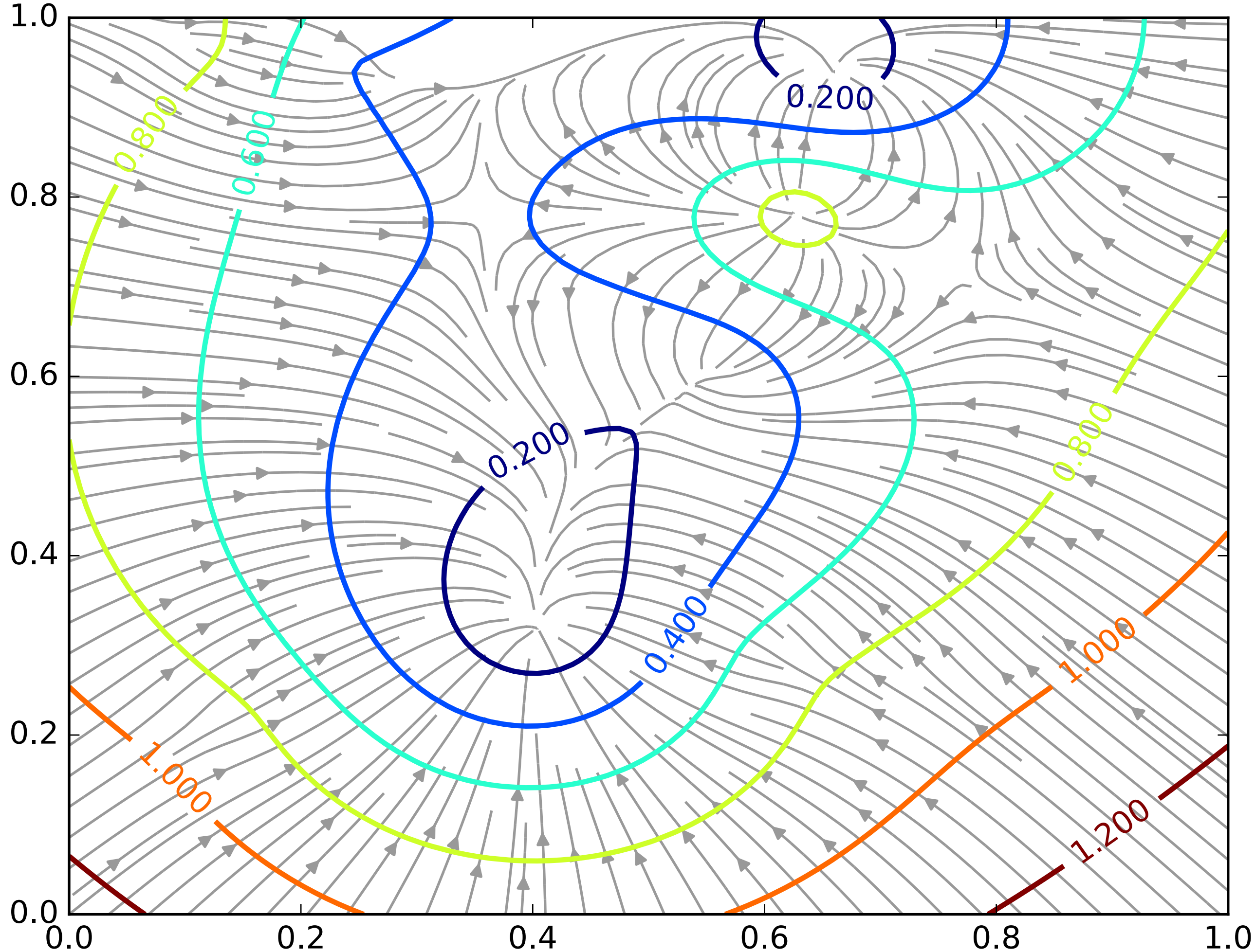
These are the **gradients** that
Gradient **Ascent** would follow.

Negative Gradients



These are the **negative** gradients that
Gradient **Descent** would follow.

Negative Gradient Paths



Shown are the paths that Gradient Descent would follow if it were making infinitesimally small steps.

Gradient Descent

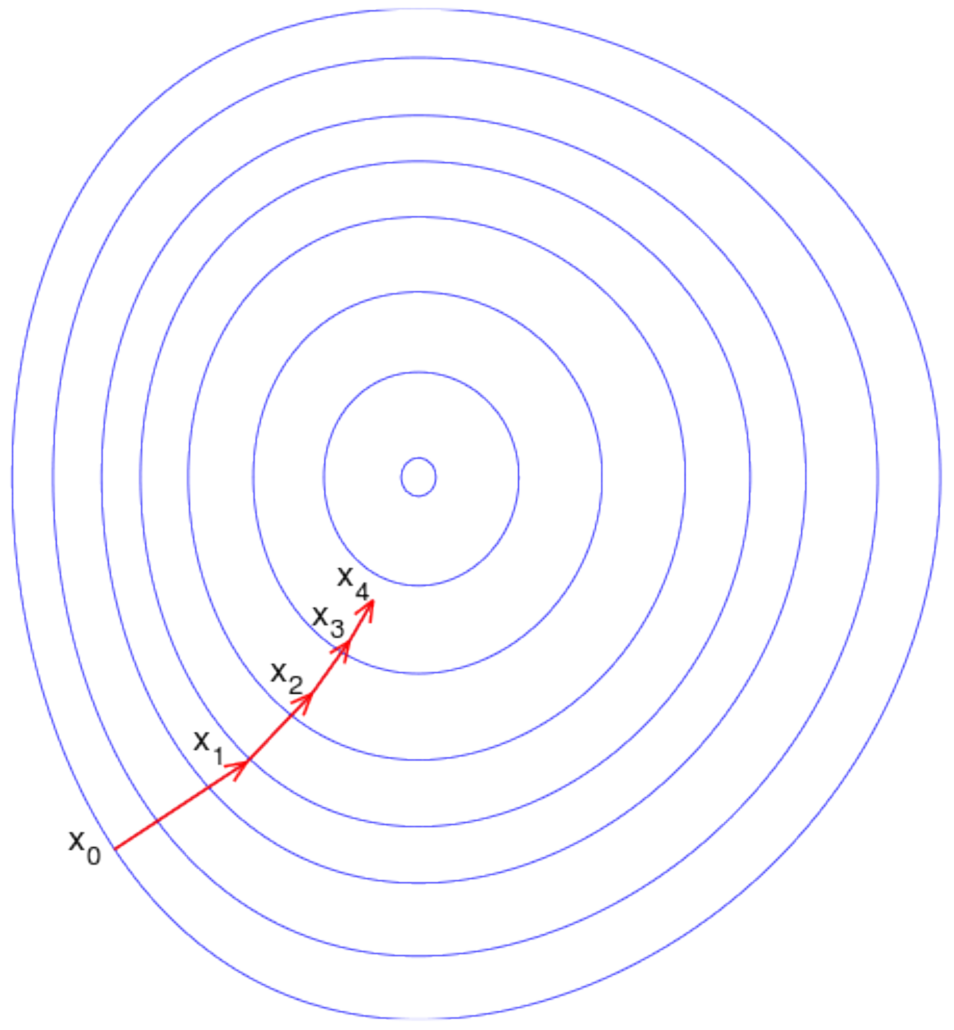
Whiteboard

- Example: 2D gradients
- Algorithm
- Details: starting point, stopping criterion, line search

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

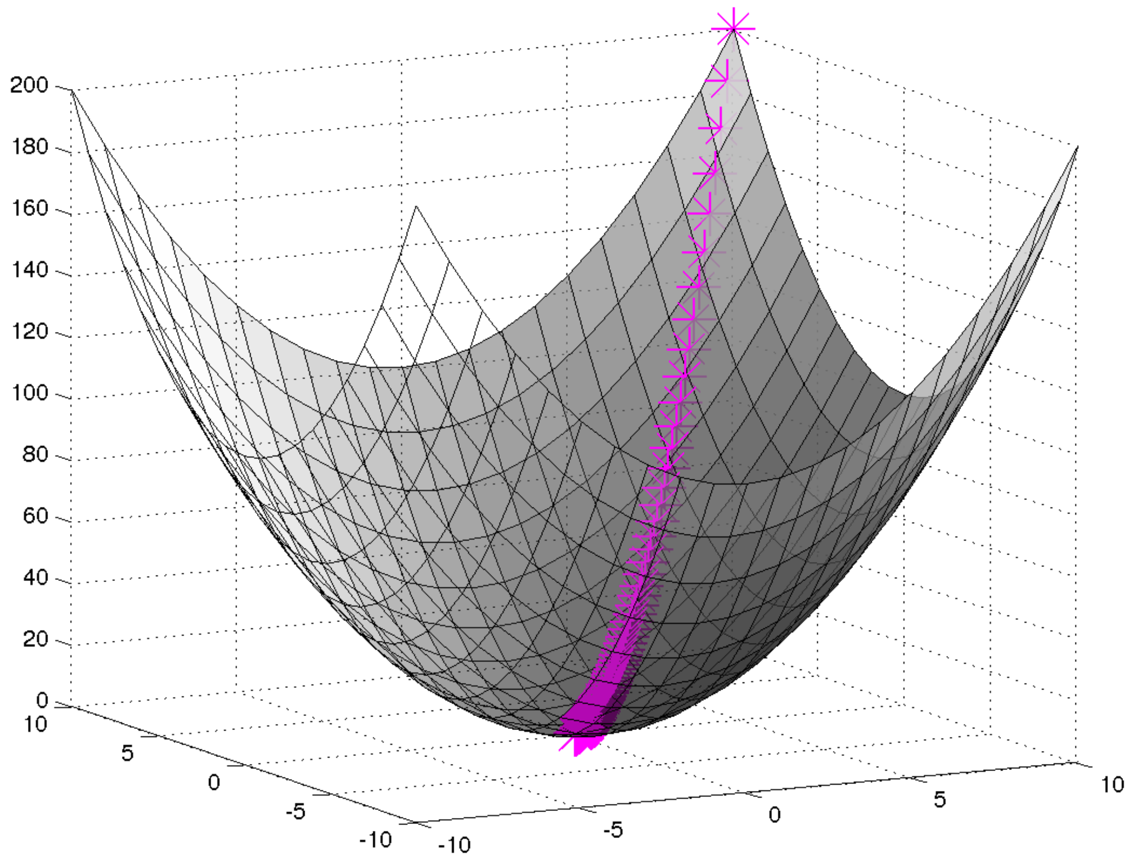


Slide courtesy of William Cohen

Gradient descent

Likelihood: ascent

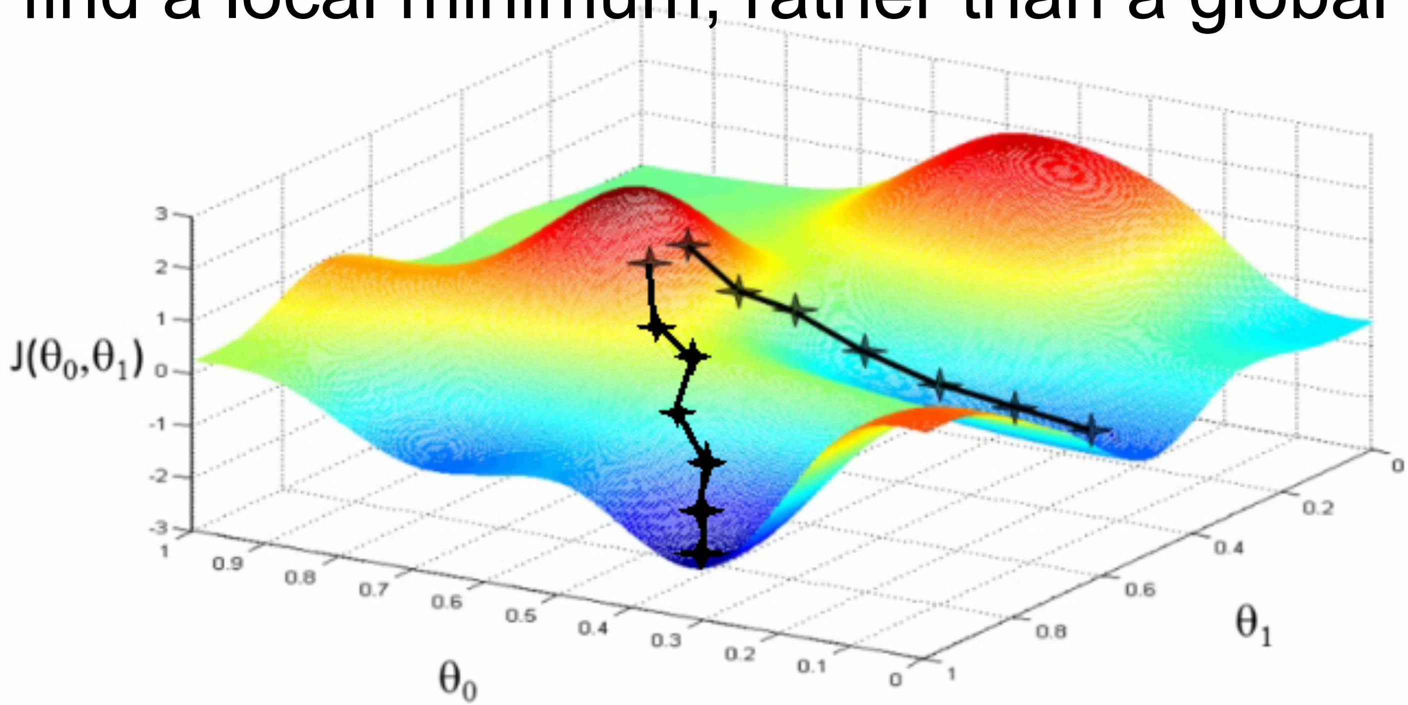
Loss: descent



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

Gradient Descent

Algorithm 1 Gradient Descent

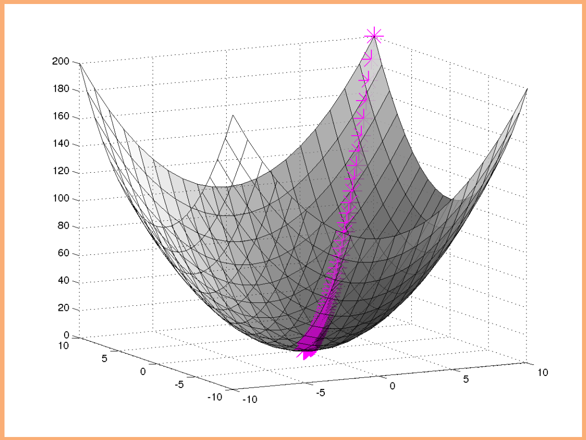
procedure GD($\mathcal{D}$, $\boldsymbol{\theta}^{(0)}$)

  $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta}^{(0)}$

  while not converged **do**

    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \lambda \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$

  return $\boldsymbol{\theta}$



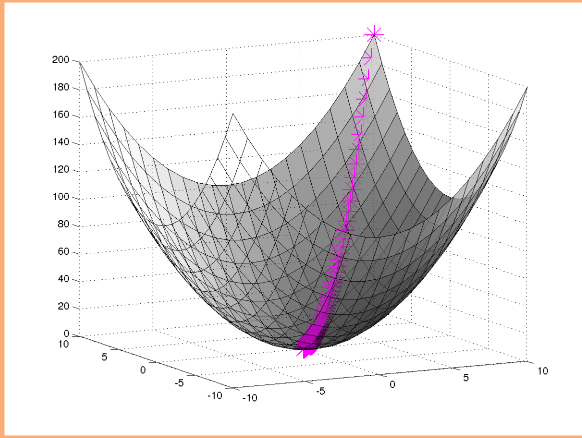
In order to apply GD to Linear Regression all we need is the **gradient** of the objective function (i.e. vector of partial derivatives).

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

Gradient Descent

Algorithm 1 Gradient Descent

procedure GD($\mathcal{D}$, $\boldsymbol{\theta}^{(0)}$)
  $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta}^{(0)}$
  while not converged **do**
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \lambda \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$
  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)

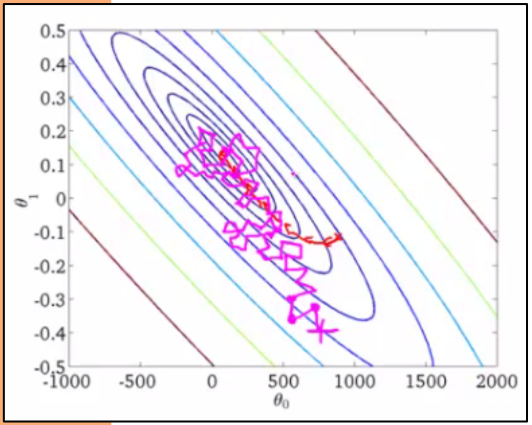
Whiteboard

- Expectations of gradients
- Algorithm
- Mini-batches
- Details: mini-batches, step size, stopping criterion
- Problematic cases for SGD

Stochastic Gradient Descent (SGD)

Algorithm 2 Stochastic Gradient Descent (SGD)

procedure SGD($\mathcal{D}$, $\boldsymbol{\theta}^{(0)}$)
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta}^{(0)}$
    while not converged **do**
        for $i \in \text{shuffle}(\{1, 2, \ldots, N\})$ **do**
            $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \lambda \nabla_{\boldsymbol{\theta}} J^{(i)}(\boldsymbol{\theta})$
    return $\boldsymbol{\theta}$



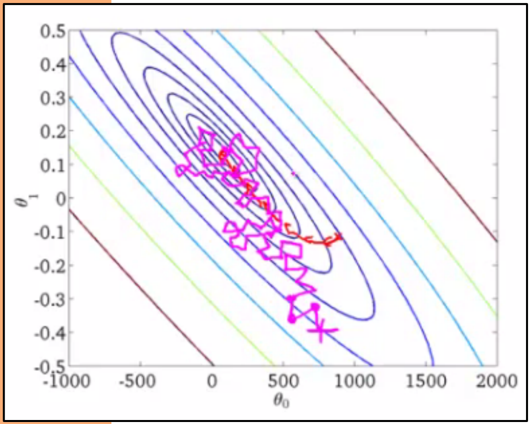
We need a per-example objective:

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

Stochastic Gradient Descent (SGD)

Algorithm 2 Stochastic Gradient Descent (SGD)

procedure SGD($\mathcal{D}$, $\boldsymbol{\theta}^{(0)}$)
  $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta}^{(0)}$
  while not converged **do**
    for $i \in \text{shuffle}(\{1, 2, \ldots, N\})$ **do**
      for $k \in \{1, 2, \ldots, K\}$ **do**
        $\theta_k \leftarrow \theta_k - \lambda \frac{d}{d\theta_k} J^{(i)}(\boldsymbol{\theta})$
  return $\boldsymbol{\theta}$



We need a per-example objective:

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

Convergence

Whiteboard

- Comparison of Newton's method, Gradient Descent, SGD
- Asymptotic convergence
- Convergence in practice

Linear Regression Outline

- **Regression Problems**
  - Definition
  - Linear functions
  - Residuals
  - Notation trick: fold in the intercept
- **Linear Regression as Function Approximation**
  - Objective function: Mean squared error
  - Hypothesis space: Linear Functions
- **Optimization for Linear Regression**
  - Normal Equations (Closed-form solution)
    - Computational complexity
    - Stability
  - SGD for Linear Regression
    - Partial derivatives
    - Update rule
  - Gradient Descent for Linear Regression
- **Probabilistic Interpretation of Linear Regression**
  - Generative vs. Discriminative
  - Conditional Likelihood
  - Background: Gaussian Distribution
  - Case #1: 1D Linear Regression
  - Case #2: Multiple Linear Regression

Regression Problems

Whiteboard

- Definition
- Linear functions
- Residuals
- Notation trick: fold in the intercept

Linear Regression as Function Approximation

Whiteboard

- Objective function: Mean squared error
- Hypothesis space: Linear Functions

Optimization for Linear Regression

Whiteboard

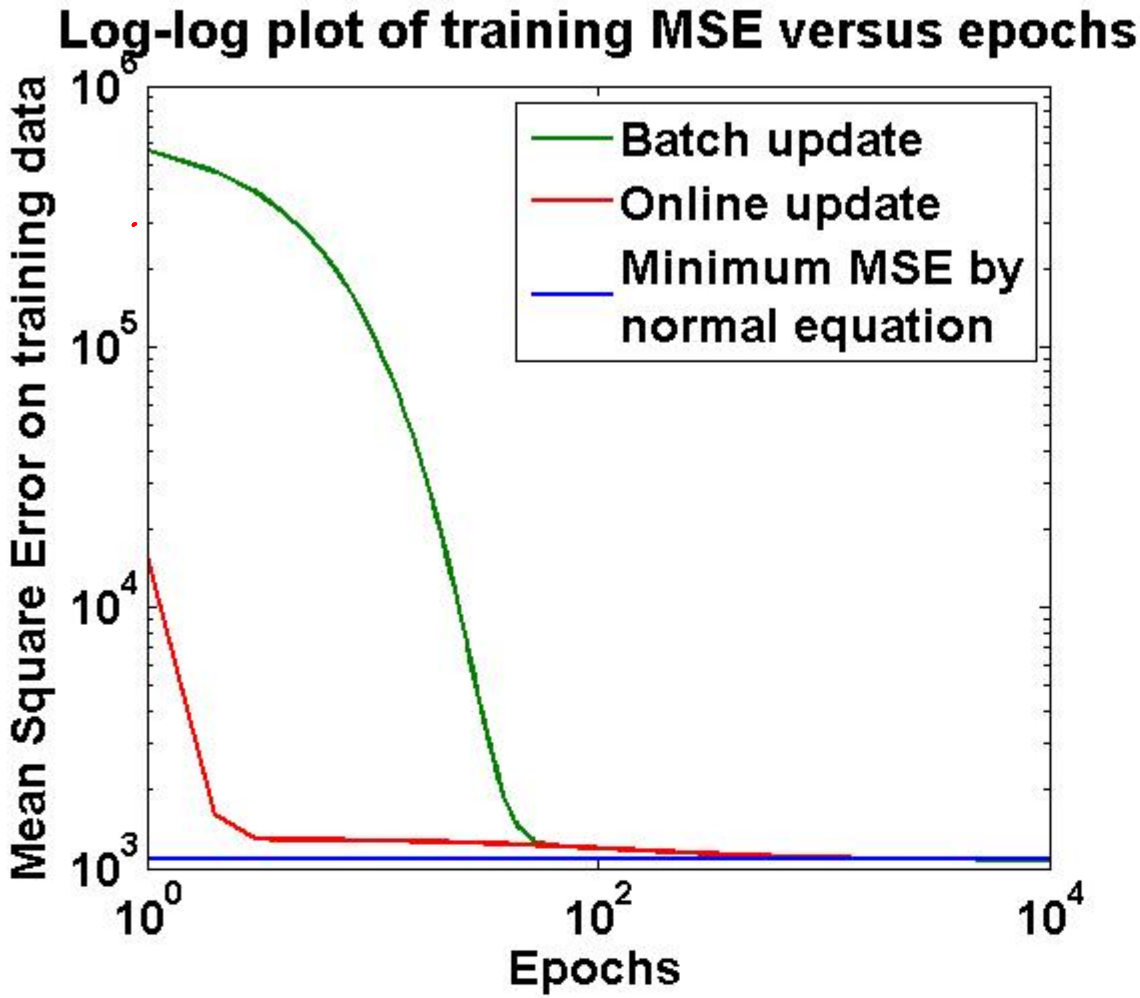
- Normal Equations (Closed-form solution)
  - Computational complexity
  - Stability
- SGD for Linear Regression
  - Partial derivatives
  - Update rule
- Gradient Descent for Linear Regression

Probabilistic Interpretation of Linear Regression

Whiteboard

- Generative vs. Discriminative
- Conditional Likelihood
- Background: Gaussian Distribution
- Case #1: 1D Linear Regression
- Case #2: Multiple Linear Regression

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.