10-601 Introduction to Machine Learning

Machine Learning Department
School of Computer Science
Carnegie Mellon University

Optimization for ML

+

Logistic Regression

Matt Gormley
Lecture 8
Feb. 11, 2019

Machine Learning Department
School of Computer Science
Carnegie Mellon University

Optimization for ML

+

~~Logistic Regression~~
Probabilistic Learning

Matt Gormley
Lecture 8
Feb. 11, 2019

Q&A

Reminders

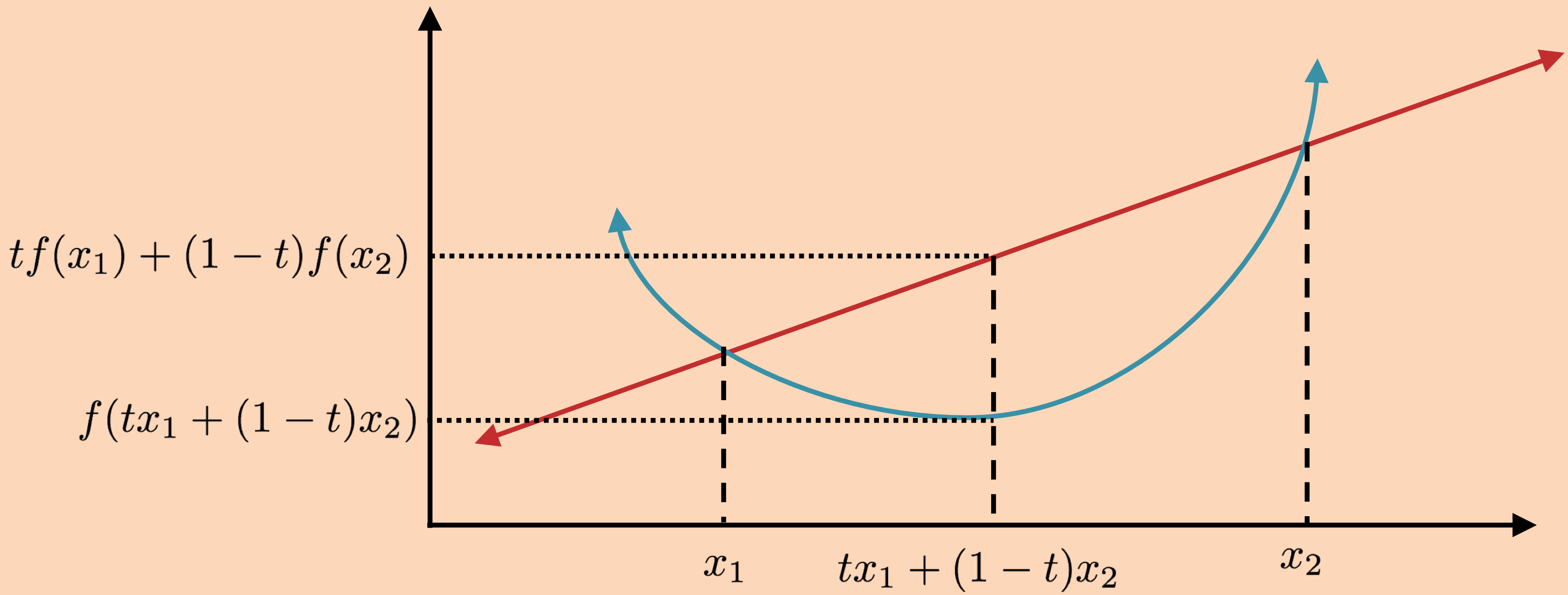
- **Homework 3: KNN, Perceptron, Lin.Reg.**
  - Out: Wed, Feb 6
  - Due: Fri, Feb 15 at 11:59pm
- **Today's In-Class Poll**
  - http://p8.mlcourse.org

CONVEXITY

Convexity

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

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

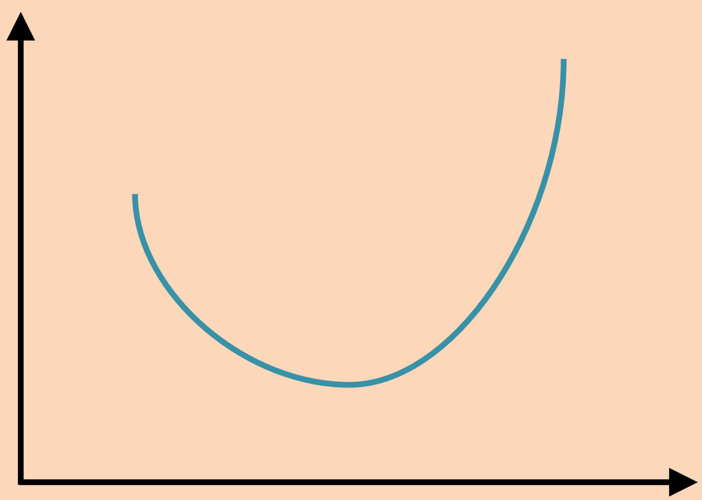


Convexity

Suppose we have a function $f(x) : \mathcal{X} \to \mathcal{Y}$.

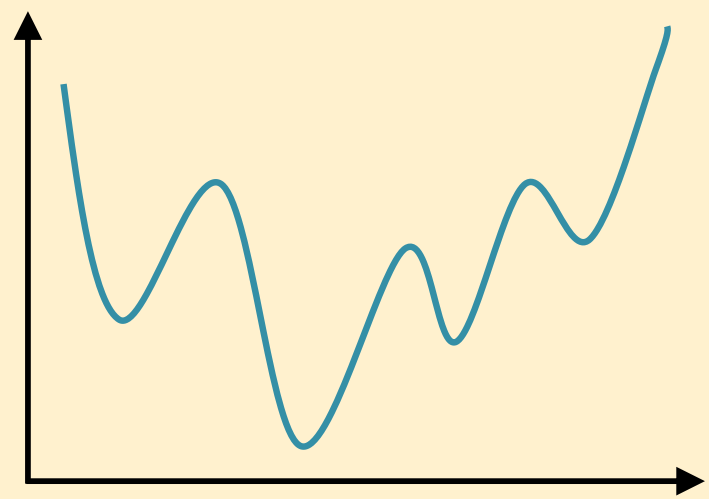
- The value x^* is a **global minimum** of f iff $f(x^*) \leq f(x), \forall x \in \mathcal{X}$.
- The value x^* is a **local minimum** of f iff $\exists \epsilon$ s.t. $f(x^*) \leq f(x), \forall x \in [x^* - \epsilon, x^* + \epsilon]$.

Convex Function



- Each **local minimum** is a **global minimum**

Nonconvex Function

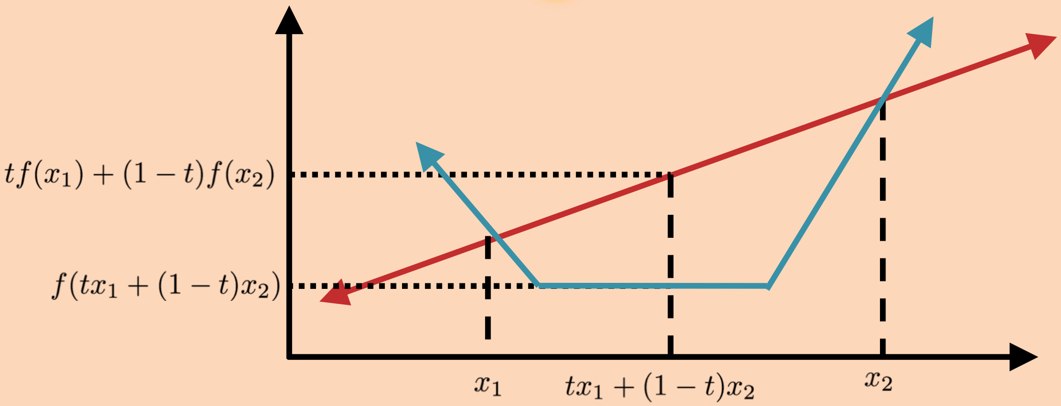


- A *nonconvex* function is **not convex**
- Each **local minimum** is **not** necessarily a **global minimum**

Convexity

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

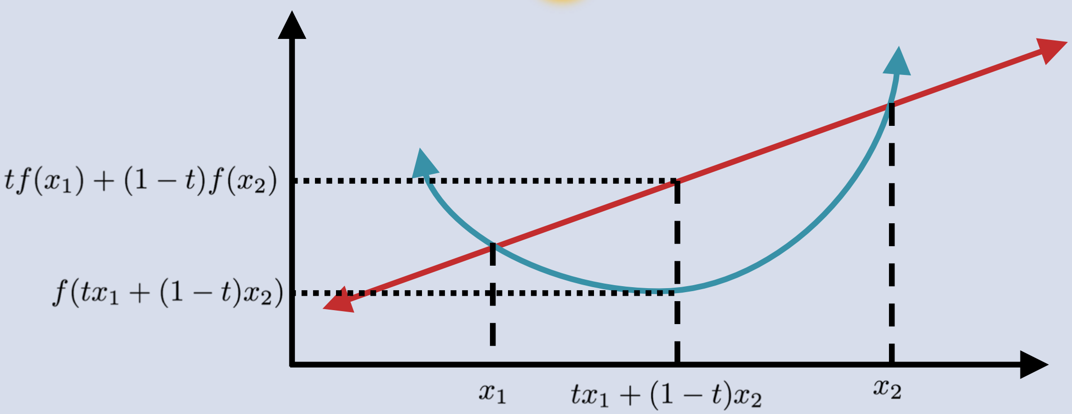
$$f(t\mathbf{x}_1 + (1-t)\mathbf{x}_2) \leq t f(\mathbf{x}_1) + (1-t) f(\mathbf{x}_2)$$



Each **local minimum** of a **convex** function is also a **global minimum.**

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

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



A **strictly convex** function has a **unique global minimum.**

Convexity

The **Mean Squared Error** function, which we minimize for learning the parameters of Linear Regression, **is convex!**

Regression Loss Functions

In-Class Exercise:

Which of the following could be used as loss functions for training a linear regression model?

Select all that apply.

A. $\ell(\hat{y}, y) = ||\hat{y} - y||_2$

B. $\ell(\hat{y}, y) = |\hat{y} - y|$

C. $\ell(\hat{y}, y) = \frac{1}{2}(\hat{y} - y)^2$

D. $\ell(\hat{y}, y) = \frac{1}{4}(\hat{y} - y)^4$

E. $\ell(\hat{y}, y) = \begin{cases} \frac{1}{2}(\hat{y} - y)^2 & \text{if } |\hat{y} - y| \leq \delta \\ \delta|\hat{y} - y| - \frac{1}{2}\delta^2 & \text{otherwise} \end{cases}$

F. $\ell(\hat{y}, y) = \log(\cosh(\hat{y} - y))$

Solving Linear Regression

Question:

True or False: If Mean Squared Error (i.e. $\frac{1}{N}\sum_{i=1}^{N}(y^{(i)} - h(\mathbf{x}^{(i)}))^2$) has a unique minimizer (i.e. argmin), then Mean Absolute Error (i.e. $\frac{1}{N}\sum_{i=1}^{N}|y^{(i)} - h(\mathbf{x}^{(i)})|$) must also have a unique minimizer.

Answer:

The Big Picture

OPTIMIZATION FOR ML

Function Approximation

Chalkboard

- The Big Picture

GRADIENT DESCENT

Motivation: Gradient Descent

To solve the Ordinary Least Squares problem we compute:

$$\hat{\boldsymbol{\theta}} = \underset{\boldsymbol{\theta}}{\text{argmin}} = \frac{1}{N}\sum_{i=1}^{N}\frac{1}{2}(y^{(i)} - (\boldsymbol{\theta}^T\mathbf{x}^{(i)}))^2$$
$$= (\mathbf{X}^T\mathbf{X})^{-1}(\mathbf{X}^T\mathbf{Y})$$

The resulting shape of the matrices:

$$\underbrace{(\underbrace{\mathbf{X}^T}_{M\times N}\ \underbrace{\mathbf{X}}_{N\times M})^{-1}}_{M\times M}\underbrace{(\underbrace{\mathbf{X}^T}_{M\times N}\ \underbrace{\mathbf{Y}}_{N\times 1})}_{M\times 1}$$

Background: Matrix Multiplication Given matrices $\mathbf{A}$ and $\mathbf{B}$

- If $\mathbf{A}$ is $q \times r$ and $\mathbf{B}$ is $r \times s$, computing $\mathbf{AB}$ takes $O(qrs)$
- If $\mathbf{A}$ and $\mathbf{B}$ are $q \times q$, computing $\mathbf{AB}$ takes $O(q^{2.373})$
- If $\mathbf{A}$ is $q \times q$, computing A^{-1} takes $O(q^{2.373})$.

Computational Complexity of OLS:

$\mathbf{X}^T\mathbf{X}$	$O(M^2N)$
$(\quad)^{-1}$	$O(M^{2.373})$
$\mathbf{X}^T\mathbf{Y}$	$O(MN)$
$(\quad)^{-1}(\quad)$	$O(M^2)$
total	$O(M^2N + M^{2.373})$

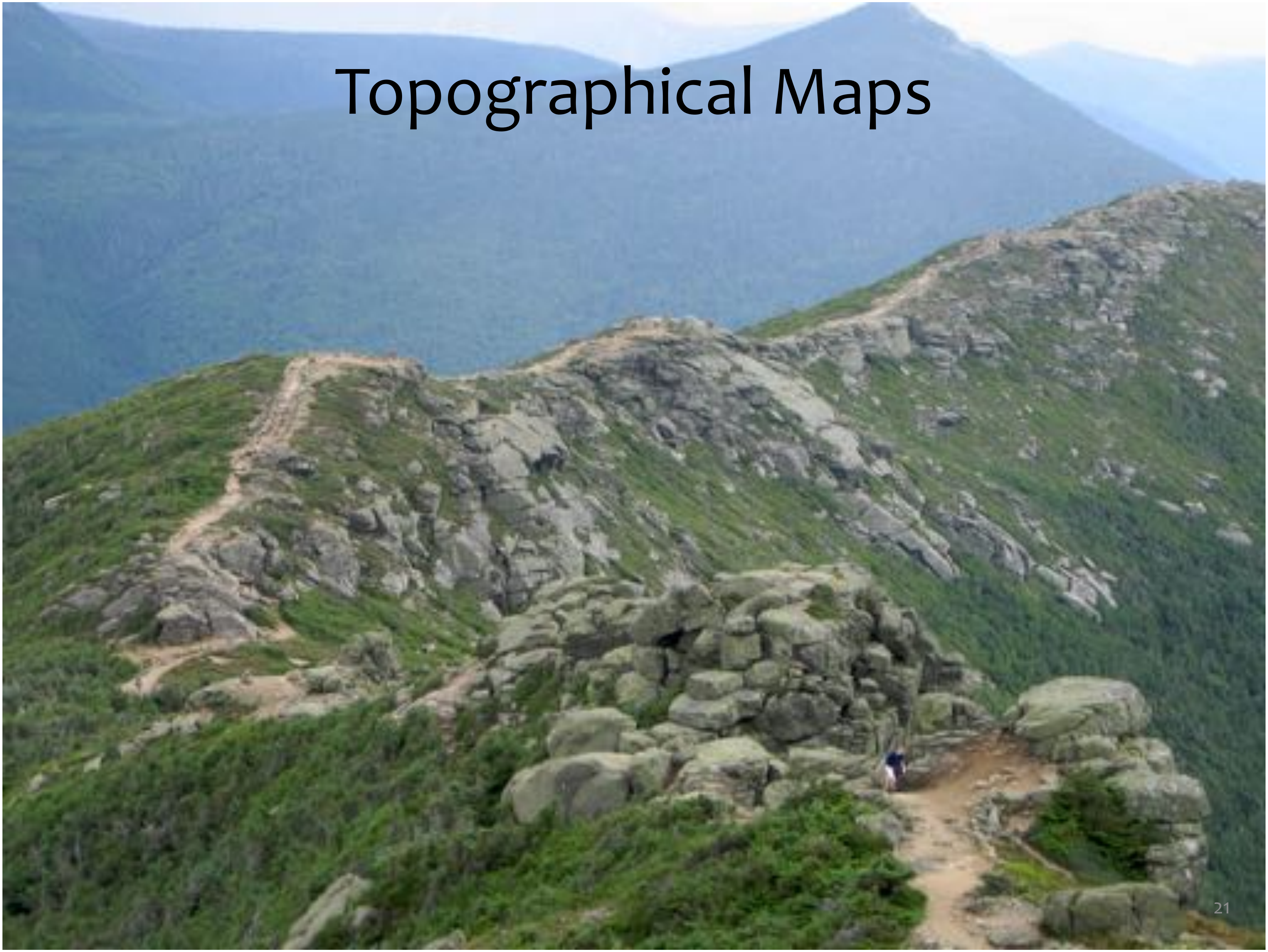
Linear in # of examples, N
Polynomial in # of features, M

Motivation: Gradient Descent

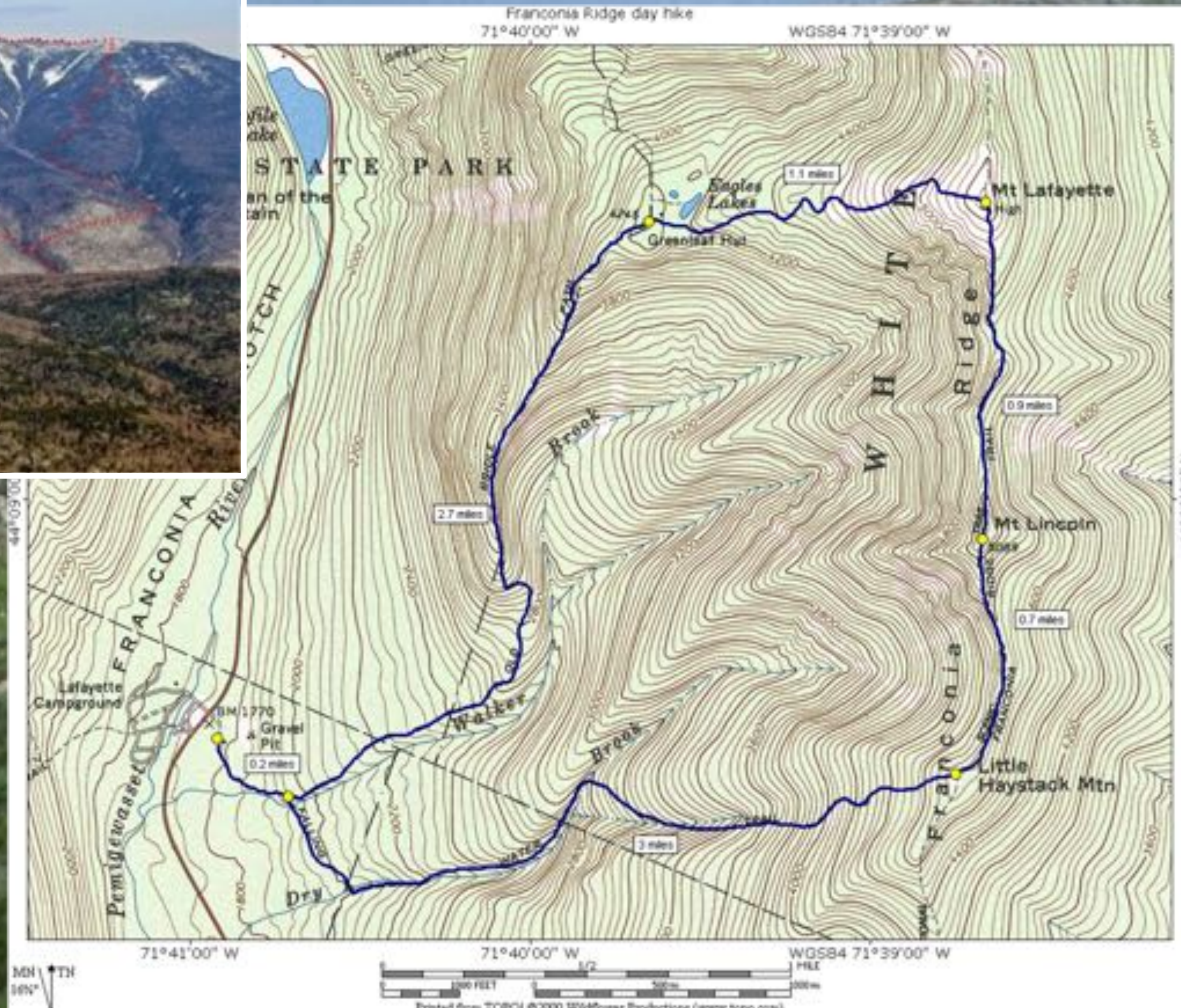
Cases to consider gradient descent:

1. What if we **can not** find a closed-form solution?
2. What if we **can**, but it's inefficient to compute?
3. What if we **can**, but it's numerically unstable to compute?

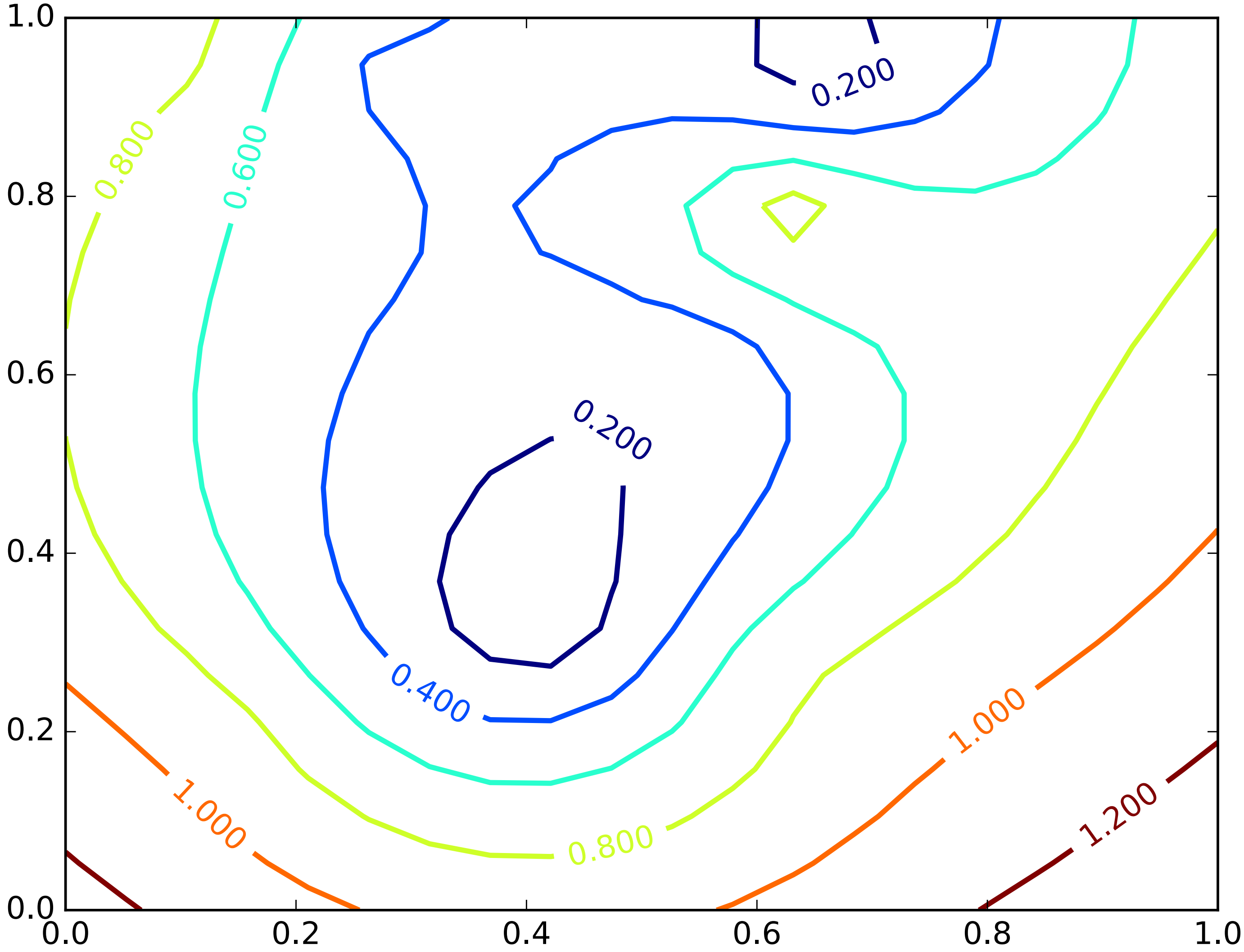
Topographical Maps



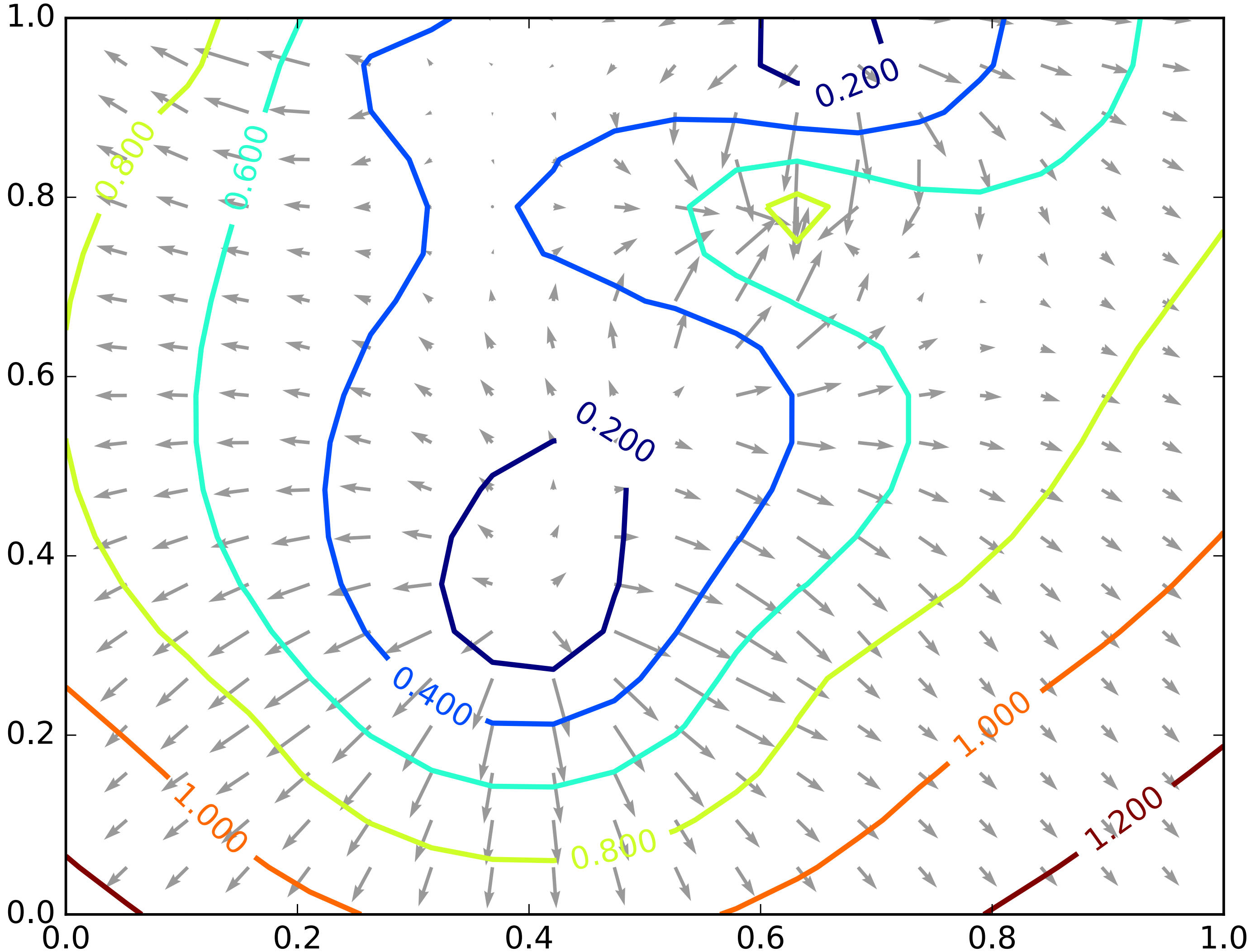
Topographical Maps



Gradients

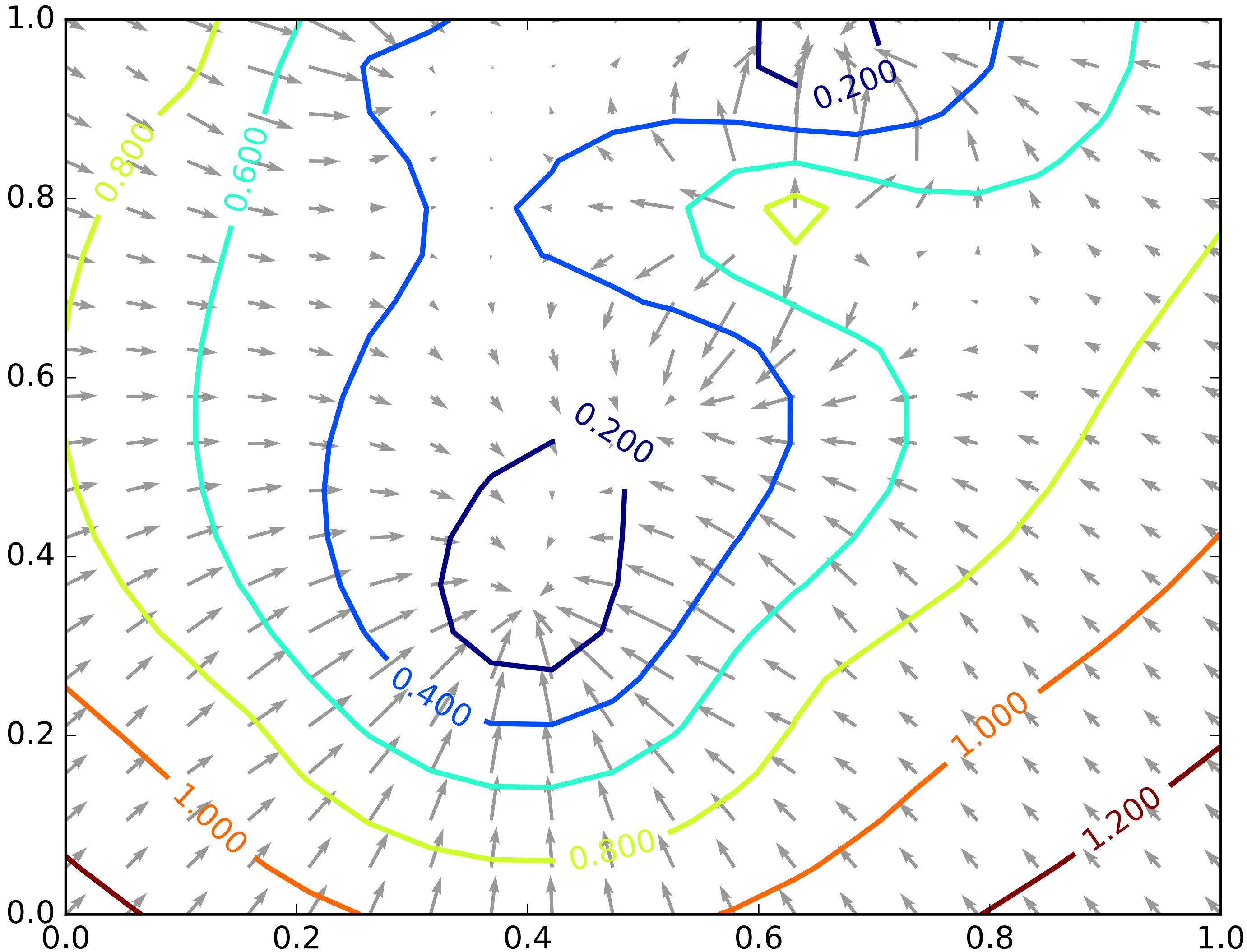


Gradients



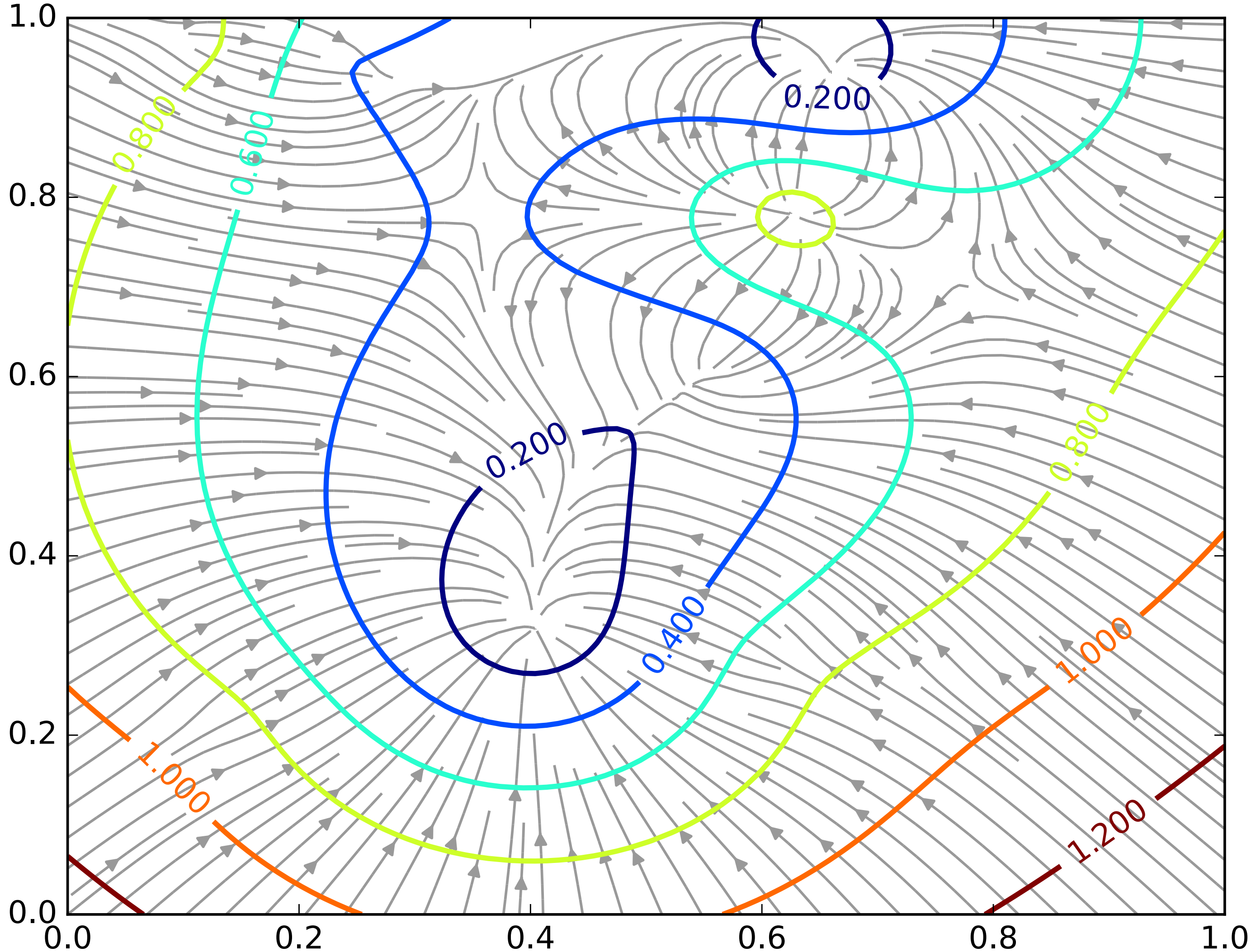
These are the **gradients** that
Gradient **Ascent** would follow.

(Negative) Gradients



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

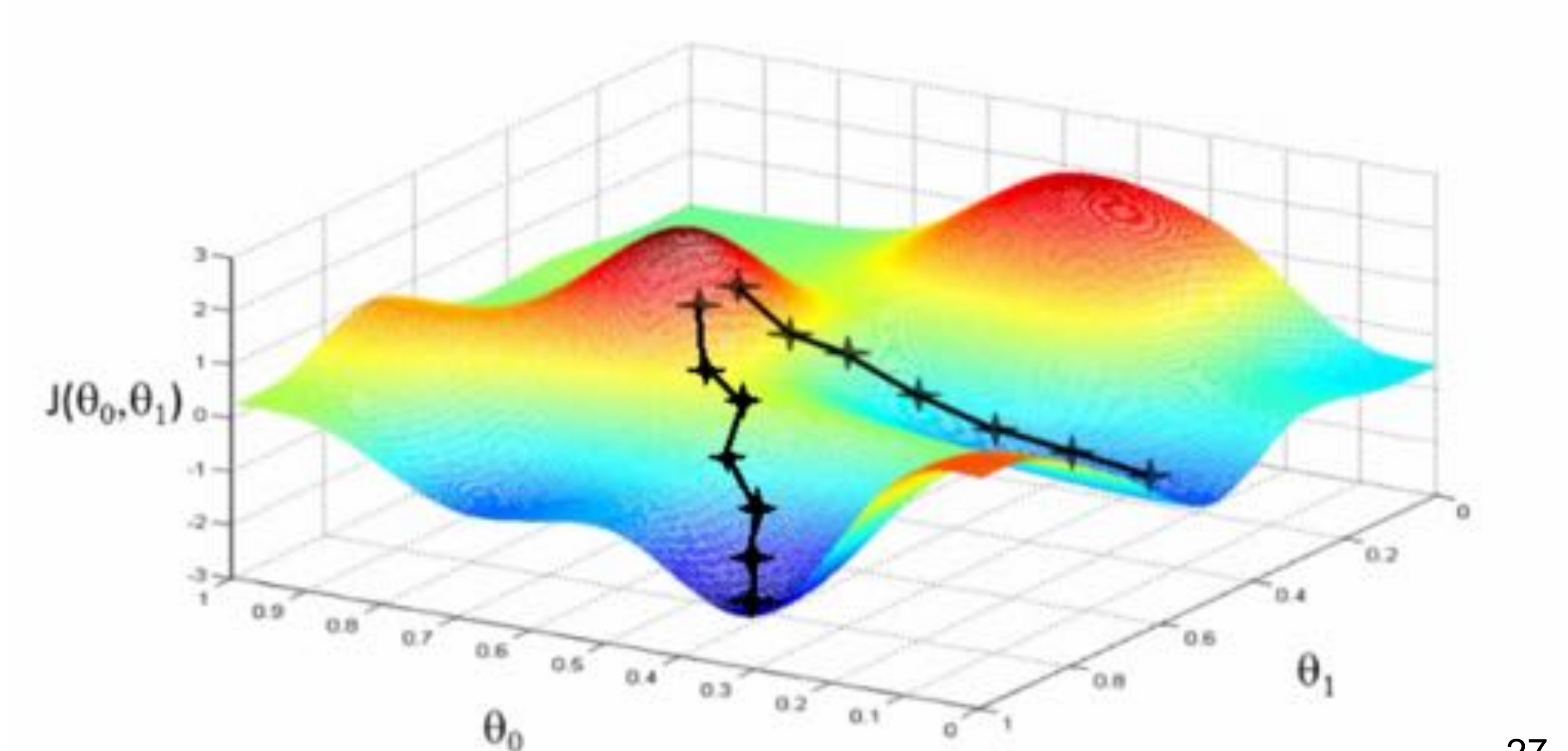
(Negative) Gradient *Paths*



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

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

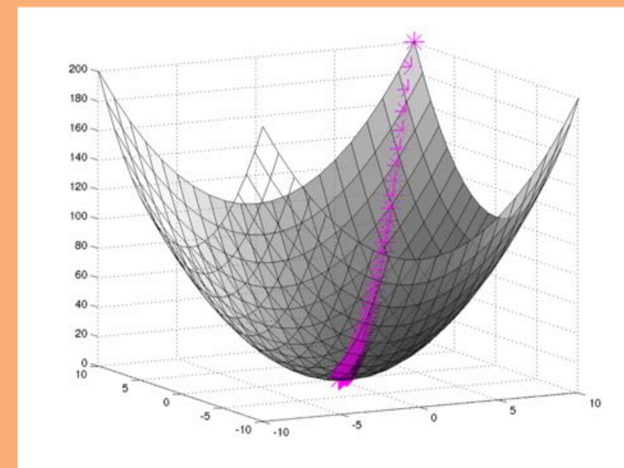
Chalkboard

- Gradient Descent Algorithm
- Details: starting point, stopping criterion, line search

Gradient Descent

Algorithm 1 Gradient Descent

$$
\begin{array}{l}
\textbf{procedure } \mathrm{GD}(\mathcal{D}, \boldsymbol{\theta}^{(0)}) \\
\quad \boldsymbol{\theta} \leftarrow \boldsymbol{\theta}^{(0)} \\
\quad \textbf{while } \text{not converged } \textbf{do} \\
\quad\quad \boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \lambda \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) \\
\quad \textbf{return } \boldsymbol{\theta}
\end{array}
$$



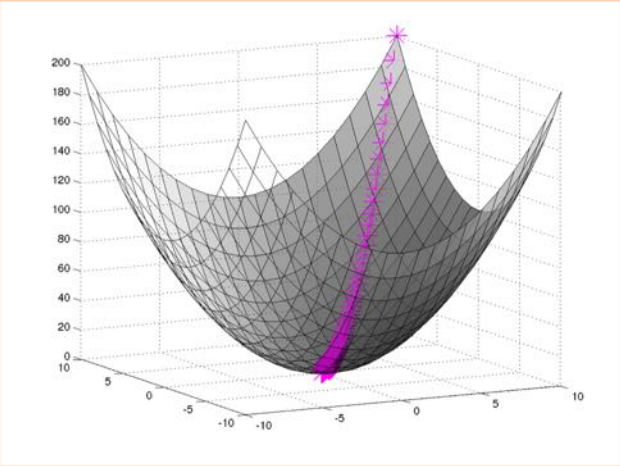
In order to apply GD to Linear Regression all we need is the **gradient** of the objective function (i.e. vector of partial derivatives).

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

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

GRADIENT DESCENT FOR LINEAR REGRESSION

Optimization for Linear Regression

Chalkboard

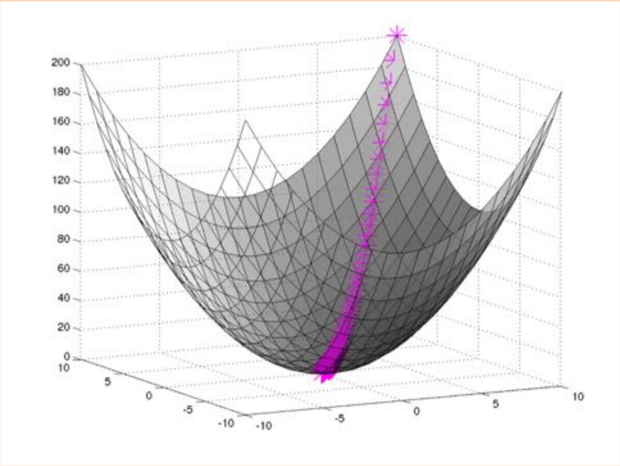
- Computing the gradient for Linear Regression
- Gradient Descent for Linear Regression
- 2D Example in Three Parts:
  1. Line over time
  2. Parameters space over time
  3. Train / test error over time

STOCHASTIC GRADIENT DESCENT

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** $\boldsymbol{\theta}$

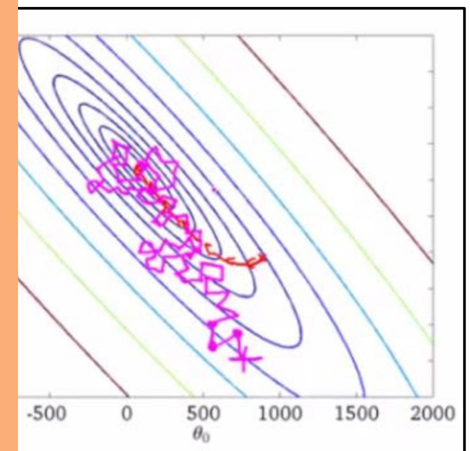


M

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 $i \sim$ Uniform($\{1, 2, \ldots, N\}$) **do**
            $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \lambda \nabla_{\boldsymbol{\theta}} J^{(i)}(\boldsymbol{\theta})$
    return $\boldsymbol{\theta}$



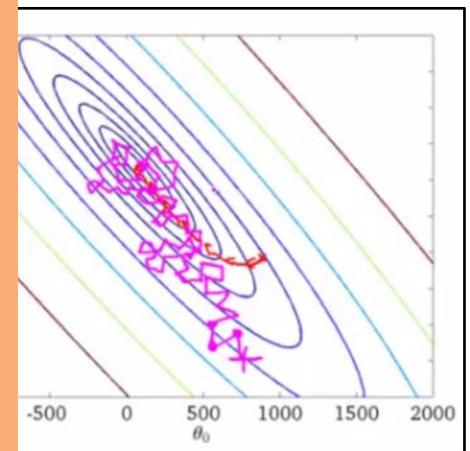
We need a per-example objective:

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

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 $i \sim$ Uniform($\{1, 2, \ldots, N\}$) **do**
            $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \lambda \nabla_{\boldsymbol{\theta}} J^{(i)}(\boldsymbol{\theta})$
    return $\boldsymbol{\theta}$



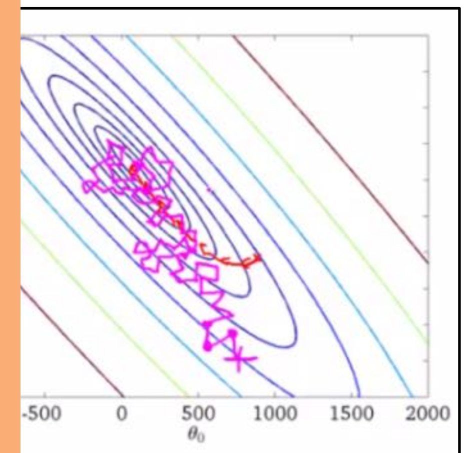
We need a per-example objective:

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

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 $i \in$ shuffle($\{1, 2, \ldots, N\}$) **do**
            $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \lambda \nabla_{\boldsymbol{\theta}} J^{(i)}(\boldsymbol{\theta})$
    return $\boldsymbol{\theta}$

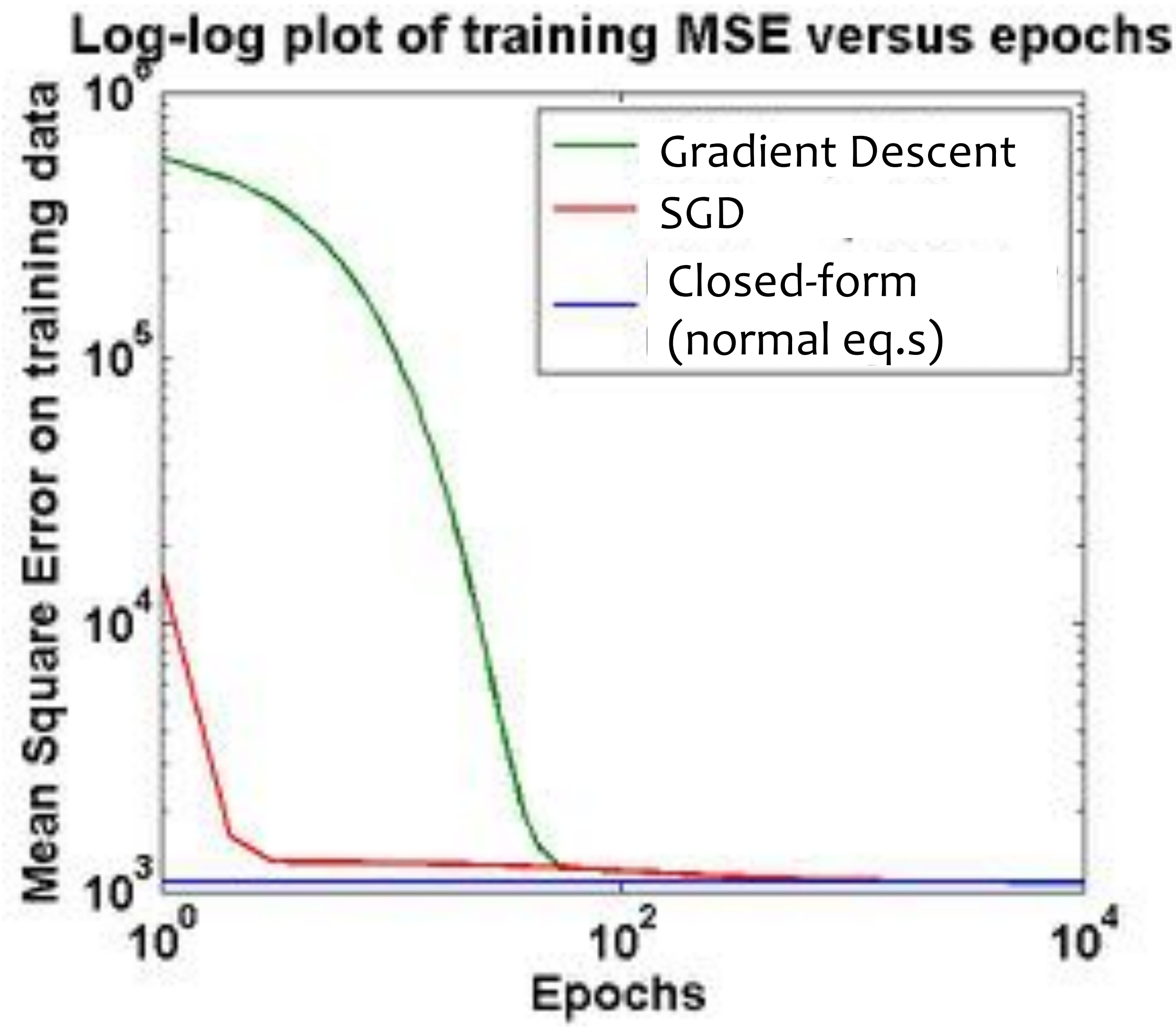


In practice, it is common to implement SGD using sampling **without** replacement (i.e. shuffle({1,2,… N}), even though most of the theory is for sampling **with** replacement (i.e. Uniform({1,2,… N}).

We need a per-example objective:

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

Convergence Curves

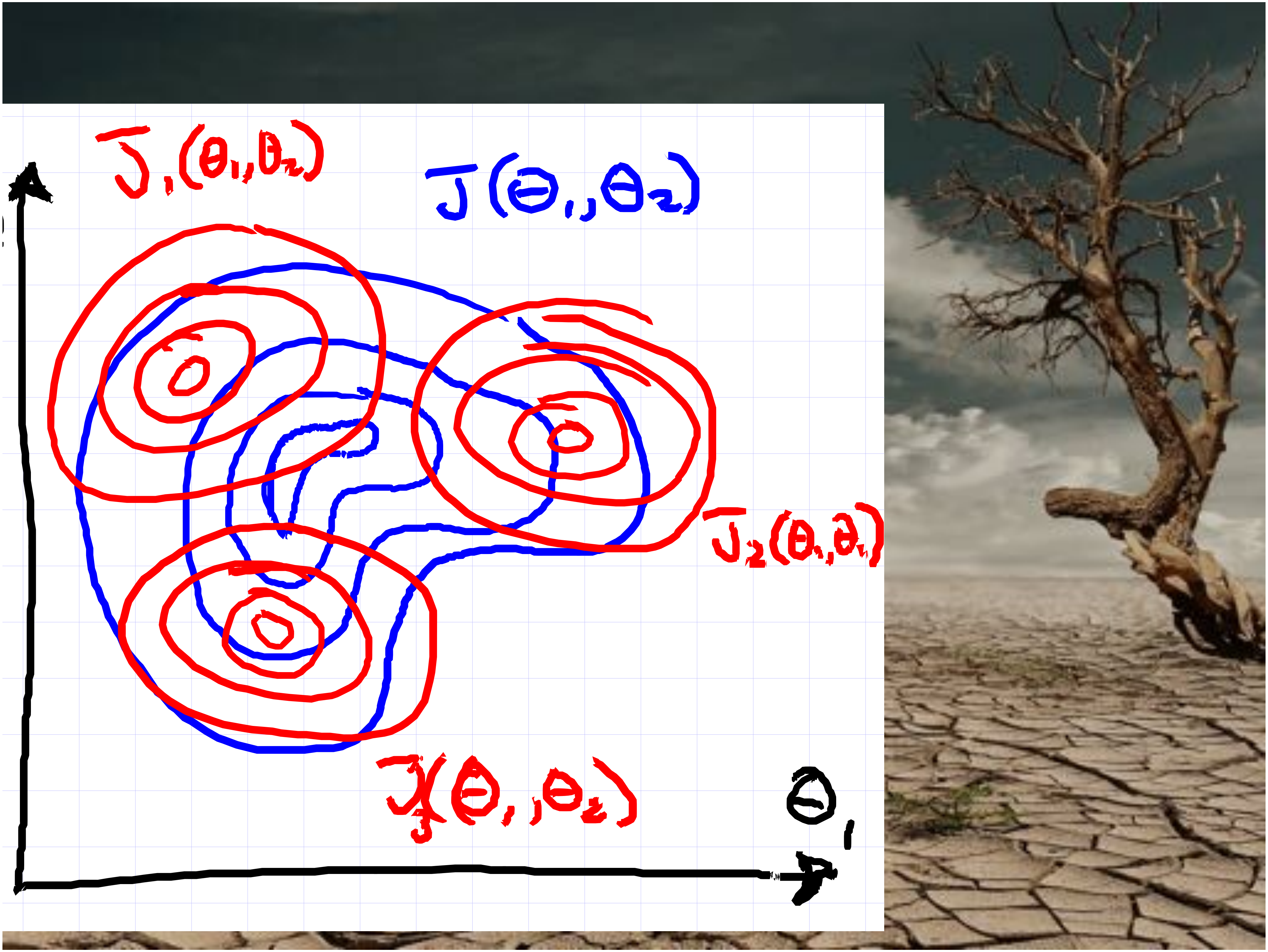


- *Def*: an **epoch** is a single pass through the training data

1. For GD, only **one update** per epoch
2. For SGD, ***N* updates** per epoch
   N = (# train examples)

- SGD reduces MSE much more rapidly than GD
- For GD / SGD, training MSE is initially large due to uninformed initialization

Figure adapted from Eric P. Xing



$J_1(\theta_1,\theta_2)$
$J(\theta_1,\theta_2)$
$J_2(\theta_1,\theta_2)$
$J_3(\theta_1,\theta_2)$
θ_1

Expectations of Gradients

$$\frac{dJ(\vec{\theta})}{d\theta_j} = \frac{1}{N}\sum_{i=1}^{N}\frac{d}{d\theta_j}\left(J_i(\vec{\theta})\right)$$

$$\nabla J(\vec{\theta}) = \begin{array}{cc} & \begin{bmatrix} \vdots \\ \hline \vdots \end{bmatrix} \\ j^{th} & \end{array} = \frac{1}{N}\sum_{i=1}^{N}\nabla J_i(\vec{\theta})$$

Recall: for any discrete r.v. X

$$E_X[f(X)] \triangleq \sum_x P(X=x)f(x)$$

Q: What is the expected value of a randomly chosen $\nabla J_i(\theta)$?

Let $I \sim \text{Uniform}(\{1,\dots,N\})$

$\Rightarrow P(I=i) = \frac{1}{N}$ if $i \in \{1 \dots N\}$

$$E_I[\nabla J_I(\vec{\theta})] = \sum_{i=1}^{N} P(I=i)\nabla J_i(\vec{\theta})$$

$$= \frac{1}{N}\sum_{i=1}^{N}\nabla J_i(\vec{\theta})$$

$$= \nabla J(\vec{\theta})$$

Convergence of Optimizers

Convergence Analysis:

Def: convergence is when $J(\vec{\theta}) - J(\vec{\theta}^*) < \epsilon$

true unknown min

not covered

Methods	Steps to Converge	Computation per iteration	
Newton's Method	$O(\ln \ln 1/\epsilon)$	$\nabla J(\theta)$	$\nabla^2 J(\theta) \leftarrow O(NM^2)$
GD	$O(\ln 1/\epsilon)$	$\nabla J(\theta) \leftarrow O(NM)$	
SGD	$O(1/\epsilon)$	$\nabla J_i(\theta) \leftarrow O(M)$	

"almost sure" convergence

lots of caveats and conditions

way less computation

Takeaway: SGD has much slower asymptotic convergence, but is often faster in practice.

Optimization Objectives

You should be able to…

- Apply gradient descent to optimize a function
- Apply stochastic gradient descent (SGD) to optimize a function
- Apply knowledge of zero derivatives to identify a closed-form solution (if one exists) to an optimization problem
- Distinguish between convex, concave, and nonconvex functions
- Obtain the gradient (and Hessian) of a (twice) differentiable function

Linear Regression Objectives

You should be able to…

- Design k-NN Regression and Decision Tree Regression
- Implement learning for Linear Regression using three optimization techniques: (1) closed form, (2) gradient descent, (3) stochastic gradient descent
- Choose a Linear Regression optimization technique that is appropriate for a particular dataset by analyzing the tradeoff of computational complexity vs. convergence speed
- Distinguish the three sources of error identified by the bias-variance decomposition: bias, variance, and irreducible error.