Machine Learning Department
School of Computer Science
Carnegie Mellon University

Stochastic Gradient Descent

+

Binary Logistic Regression

Matt Gormley
Lecture 9
Feb. 10, 2025

Reminders

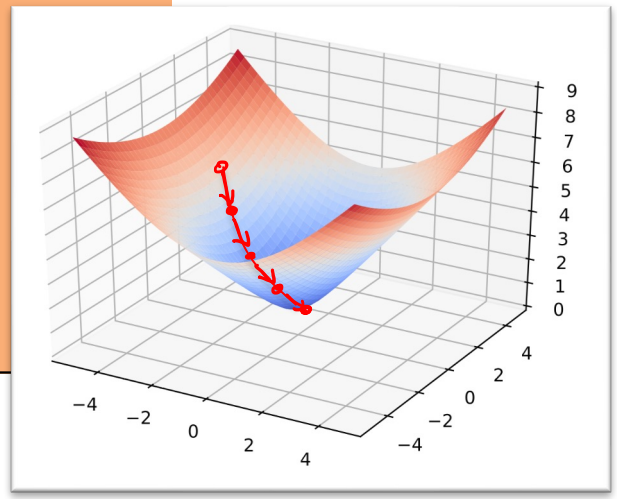
- Homework 3: KNN, Perceptron, Lin.Reg.
  - Out: Mon, Feb. 3
  - Due: Mon, Feb. 10 at 11:59pm
  - (only two grace/late days permitted)
- Practice Problems 1
  - released on course website
- Exam 1: Mon, Feb. 17
  - Time: 7:00 – 9:00pm
  - Location: Your room/seat assignment will be announced on Piazza

OPTIMIZATION METHOD #4: 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: $\quad\quad\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \gamma \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$
5: $\quad$**return** $\boldsymbol{\theta}$



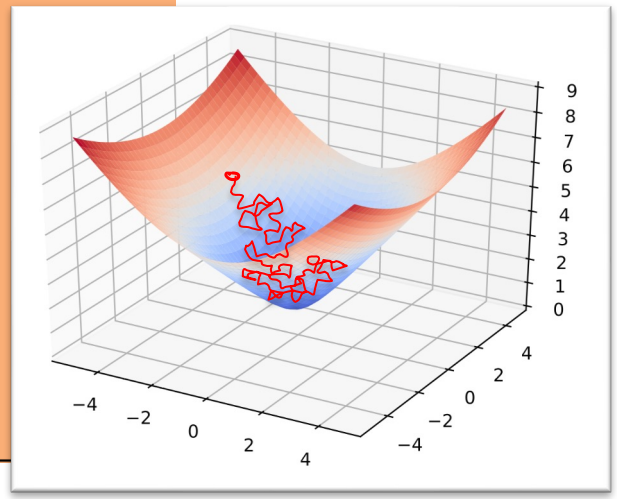
$$\nabla J(\theta) = \frac{1}{N} \sum_{i=1}^{N} \nabla J^{(i)}(\theta)$$

M

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: $\quad\quad i \sim \text{Uniform}(\{1, 2, \ldots, N\})$
5: $\quad\quad \boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \gamma \nabla_{\boldsymbol{\theta}} J^{(i)}(\boldsymbol{\theta})$
6: $\quad$ **return** $\boldsymbol{\theta}$



per-example objective:

$$J^{(i)}(\boldsymbol{\theta})$$

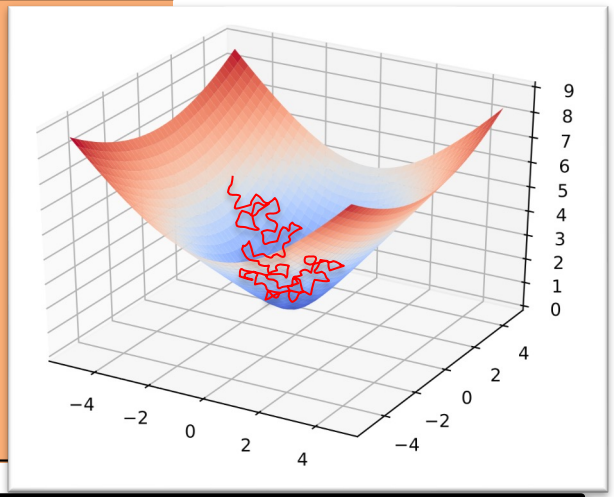
original objective:

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

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: $\quad\quad\quad \boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \gamma \nabla_{\boldsymbol{\theta}} J^{(i)}(\boldsymbol{\theta})$
6: $\quad$ **return** $\boldsymbol{\theta}$



per-example objective:

$$J^{(i)}(\boldsymbol{\theta})$$

original objective:

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

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}).

Why does SGD work?

Background: *Expectation of a function of a random variable*

For any discrete random variable X

$$E_X[f(X)] = \sum_{x \in X} P(X = x) f(x)$$

Objective Function for SGD

We assume the form to be:

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

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

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

Expectation of a Stochastic Gradient:

- If the example is sampled uniformly at random, the expected value of the pointwise gradient is the same as the full gradient!

$$E\left[\nabla_{\boldsymbol{\theta}} J^{(i)}(\boldsymbol{\theta})\right] = \sum_{i=1}^{N} \left(\text{probability of selecting } \boldsymbol{x}^{(i)}, y^{(i)}\right) \nabla_{\boldsymbol{\theta}} J^{(i)}(\boldsymbol{\theta})$$

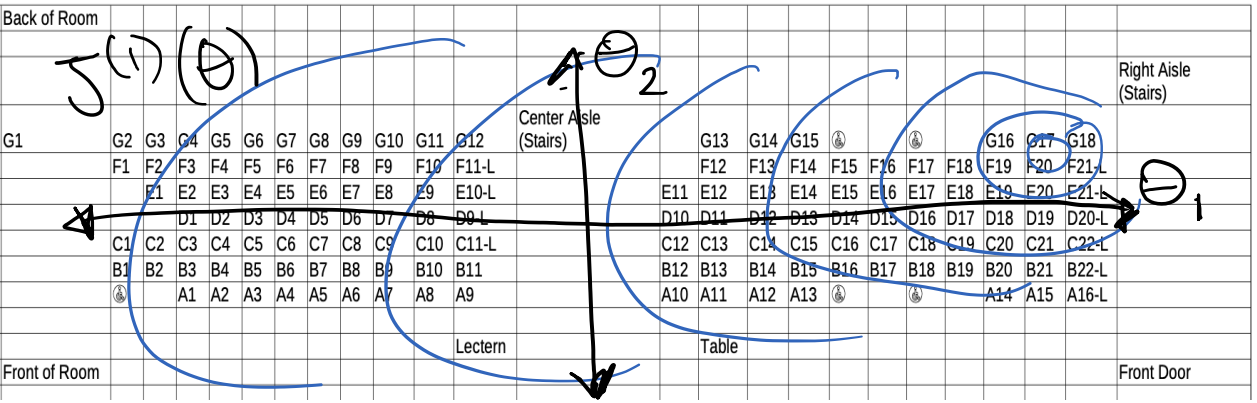
$$= \sum_{i=1}^{N} \left(\frac{1}{N}\right) \nabla_{\boldsymbol{\theta}} J^{(i)}(\boldsymbol{\theta})$$

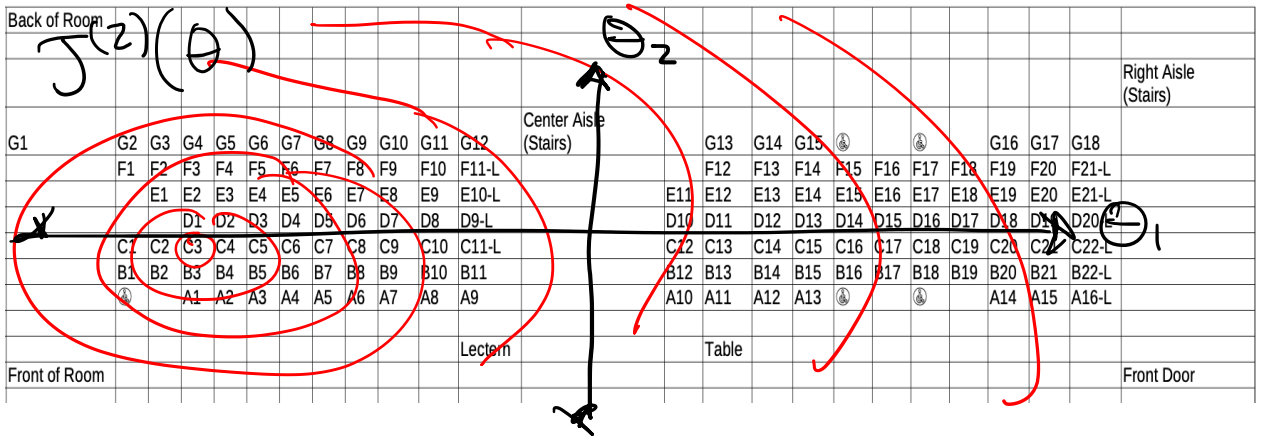
$$= \frac{1}{N} \sum_{i=1}^{N} \nabla_{\boldsymbol{\theta}} J^{(i)}(\boldsymbol{\theta})$$

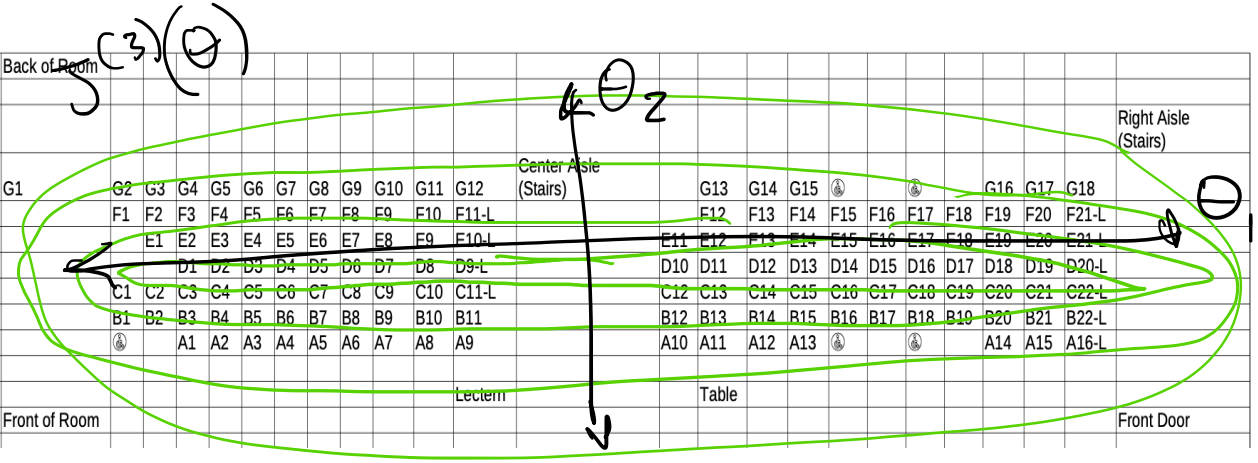
$$= \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$$

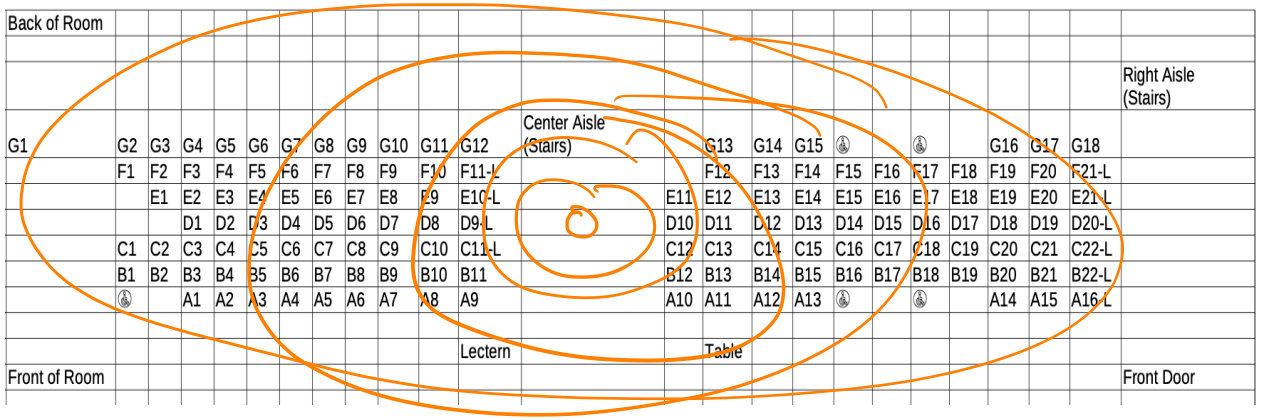
- In practice, the data set is randomly shuffled then looped through so that each data point is used equally often

Why does SGD work?

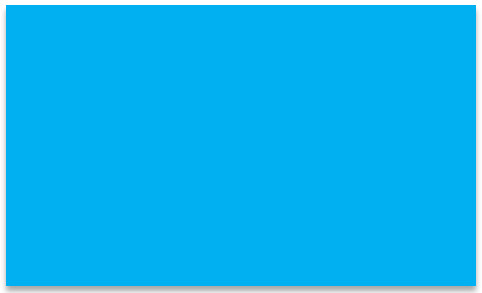






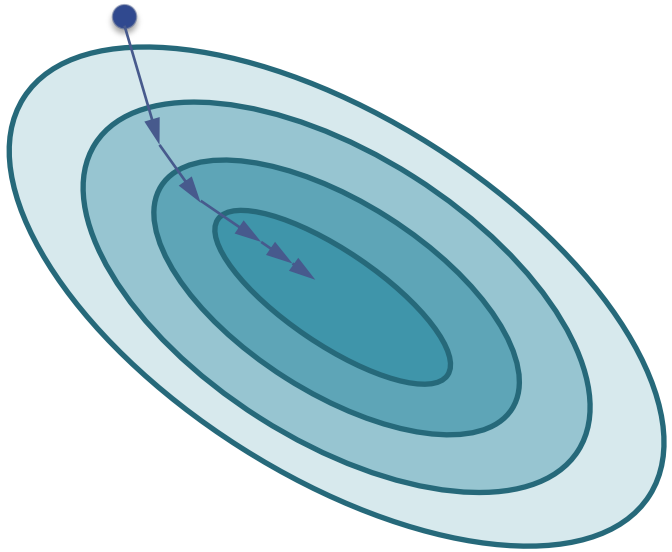


$$J(\theta) = J^{(1)}(\theta) + J^{(2)}(\theta) + J^{(3)}(\theta)$$

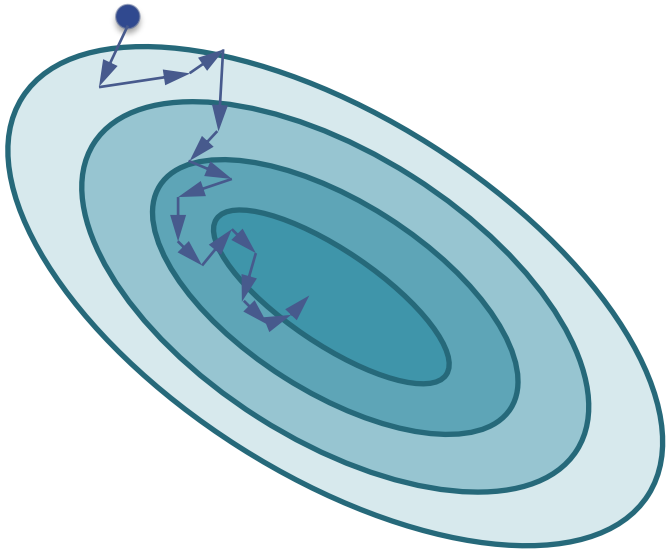


SGD VS. GRADIENT DESCENT

SGD vs. Gradient Descent



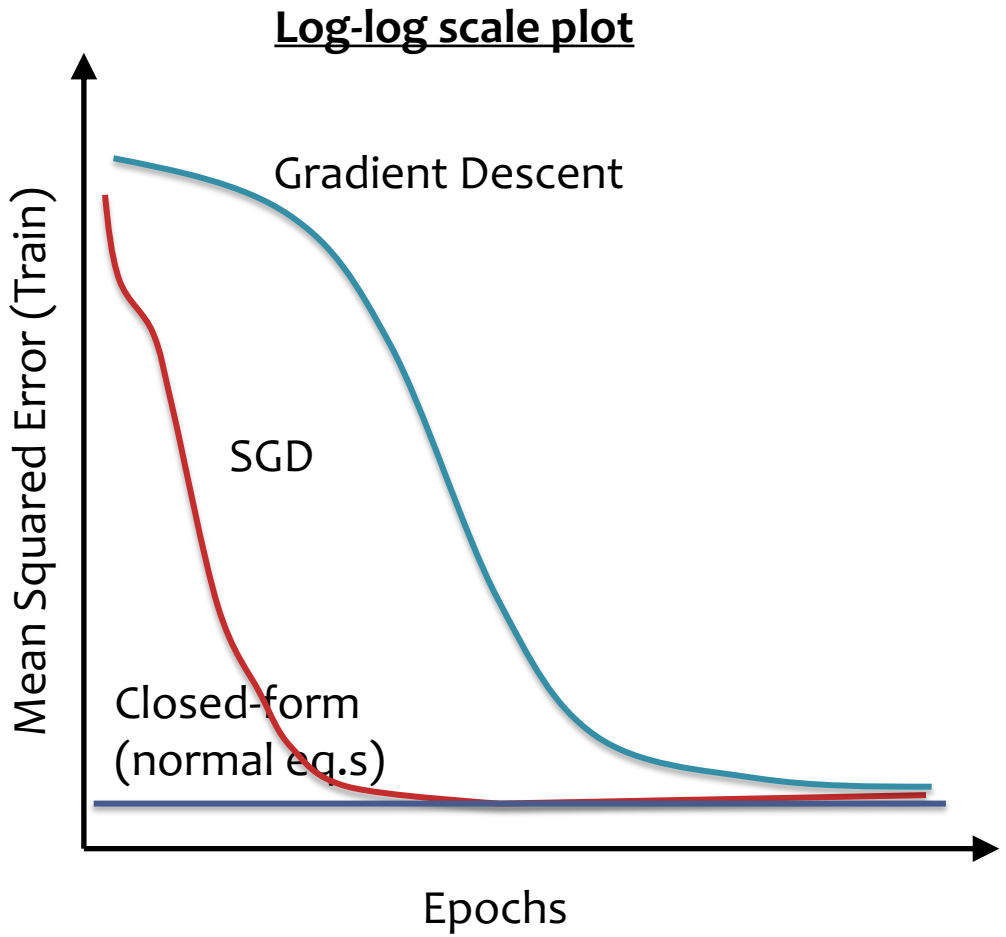
Gradient Descent



Stochastic Gradient Descent

SGD vs. Gradient Descent

- Empirical comparison:



- *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

SGD vs. Gradient Descent

- Theoretical comparison:

Define convergence to be when $J\left(\boldsymbol{\theta}^{(t)}\right) - J(\boldsymbol{\theta}^*) < \epsilon$

Method	Steps to Convergence	Computation per Step
Gradient descent	$O\left(\log {}^{1}/_{\epsilon}\right)$	$O(NM)$
SGD	$O\left({}^{1}/_{\epsilon}\right)$	$O(M)$

(with high probability under certain assumptions)

Main Takeaway: SGD has much slower asymptotic convergence (i.e. it's slower in theory), but is often much faster in practice.

SGD FOR LINEAR REGRESSION

Linear Regression as Function Approximation

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

1. Assume $\mathcal{D}$ generated as:

$$\mathbf{x}^{(i)} \sim p^*(\cdot)$$
$$y^{(i)} = h^*(\mathbf{x}^{(i)})$$

2. Choose hypothesis space, $\mathcal{H}$:
all linear functions in M-dimensional space

$$\mathcal{H} = \{h_{\boldsymbol{\theta}} : h_{\boldsymbol{\theta}}(\mathbf{x}) = \boldsymbol{\theta}^T \mathbf{x}, \boldsymbol{\theta} \in \mathbb{R}^M\}$$

3. Choose an objective function:
mean squared error (MSE)

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

4. Solve the unconstrained optimization problem via favorite method:

- *gradient descent*
- *closed form*
- *stochastic gradient descent*
- . . .

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

5. Test time: given a new $\mathbf{x}$, make prediction $\hat{y}$

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

Gradient Calculation for Linear Regression

Derivative of $J^{(i)}(\boldsymbol{\theta})$:

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

Derivative of $J(\boldsymbol{\theta})$:

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

new

forSGD

Gradient of $J^{(i)}(\boldsymbol{\theta})$ [used by SGD]

$$\begin{aligned}
\nabla_{\boldsymbol{\theta}} J^{(i)}(\boldsymbol{\theta}) &= \begin{bmatrix} \frac{d}{d\theta_1} J^{(i)}(\boldsymbol{\theta}) \\ \frac{d}{d\theta_2} J^{(i)}(\boldsymbol{\theta}) \\ \vdots \\ \frac{d}{d\theta_M} J^{(i)}(\boldsymbol{\theta}) \end{bmatrix} = \begin{bmatrix} (\boldsymbol{\theta}^T \mathbf{x}^{(i)} - y^{(i)}) x_1^{(i)} \\ (\boldsymbol{\theta}^T \mathbf{x}^{(i)} - y^{(i)}) x_2^{(i)} \\ \vdots \\ (\boldsymbol{\theta}^T \mathbf{x}^{(i)} - y^{(i)}) x_N^{(i)} \end{bmatrix} \\
&= (\boldsymbol{\theta}^T \mathbf{x}^{(i)} - y^{(i)}) \mathbf{x}^{(i)}
\end{aligned}$$

Gradient of $J(\boldsymbol{\theta})$ [used by Gradient Descent]

$$\begin{aligned}
\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) &= \begin{bmatrix} \frac{d}{d\theta_1} J(\boldsymbol{\theta}) \\ \frac{d}{d\theta_2} J(\boldsymbol{\theta}) \\ \vdots \\ \frac{d}{d\theta_M} J(\boldsymbol{\theta}) \end{bmatrix} = \begin{bmatrix} \sum_{i=1}^{N} (\boldsymbol{\theta}^T \mathbf{x}^{(i)} - y^{(i)}) x_1^{(i)} \\ \sum_{i=1}^{N} (\boldsymbol{\theta}^T \mathbf{x}^{(i)} - y^{(i)}) x_2^{(i)} \\ \vdots \\ \sum_{i=1}^{N} (\boldsymbol{\theta}^T \mathbf{x}^{(i)} - y^{(i)}) x_N^{(i)} \end{bmatrix} \\
&= \frac{1}{N} \sum_{i=1}^{N} (\boldsymbol{\theta}^T \mathbf{x}^{(i)} - y^{(i)}) \mathbf{x}^{(i)}
\end{aligned}$$

For GD

SGD for Linear Regression

SGD applied to Linear Regression is called the "Least Mean Squares" algorithm

Algorithm 1 Least Mean Squares (LMS)

procedure LMS($\mathcal{D}, \boldsymbol{\theta}^{(0)}$)
  $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta}^{(0)}$ ▷ Initialize parameters
  while not converged **do**
    for $i \in$ shuffle($\{1, 2, \ldots, N\}$) **do**
      $\mathbf{g} \leftarrow (\boldsymbol{\theta}^T \mathbf{x}^{(i)} - y^{(i)})\mathbf{x}^{(i)}$ ▷ Compute gradient
      $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \gamma \mathbf{g}$ ▷ Update parameters
  return $\boldsymbol{\theta}$

GD for Linear Regression

Gradient Descent for Linear Regression repeatedly takes steps opposite the gradient of the objective function

Algorithm 1 GD for Linear Regression

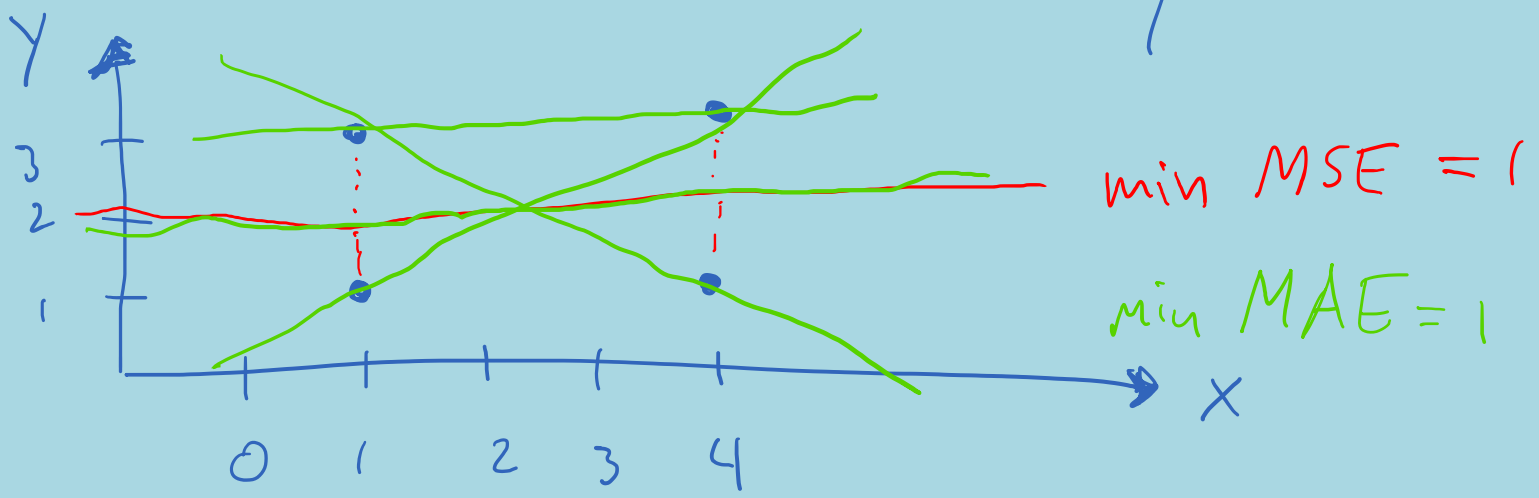
1: **procedure** GDLR($\mathcal{D}$, $\boldsymbol{\theta}^{(0)}$)
2: $\quad\boldsymbol{\theta} \leftarrow \boldsymbol{\theta}^{(0)}$ ▷ Initialize parameters
3: $\quad$**while** not converged **do**
4: $\quad\quad\mathbf{g} \leftarrow \frac{1}{N}\sum_{i=1}^{N}(\boldsymbol{\theta}^T\mathbf{x}^{(i)} - y^{(i)})\mathbf{x}^{(i)}$ ▷ Compute gradient
5: $\quad\quad\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \gamma\mathbf{g}$ ▷ Update parameters
6: $\quad$**return** $\boldsymbol{\theta}$

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:



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

PROBABILISTIC LEARNING

Probabilistic Learning

Function Approximation

Previously, we assumed that our output was generated using a **deterministic target function:**

$$\mathbf{x}^{(i)} \sim p^*(\cdot)$$

$$y^{(i)} = c^*(\mathbf{x}^{(i)})$$

Our goal was to learn a hypothesis h(**x**) that best approximates $c^*(\mathbf{x})$

Probabilistic Learning

Today, we assume that our output is **sampled** from a conditional **probability distribution:**

$$\mathbf{x}^{(i)} \sim p^*(\cdot)$$

$$y^{(i)} \sim p^*(\cdot|\mathbf{x}^{(i)})$$

Our goal is to learn a probability distribution p(y|**x**) that best approximates $p^*(y|\mathbf{x})$

Robotic Farming

	Deterministic	**Probabilistic**
Classification (binary output)	Is this a picture of a wheat kernel?	Is this plant drought resistant?
Regression (continuous output)	How many wheat kernels are in this picture?	What will the yield of this plant be?

MAXIMUM LIKELIHOOD ESTIMATION

Likelihood Function

One R.V.

Given N **independent, identically distributed (iid)** samples D = $\{x^{(1)}, x^{(2)}, \ldots, x^{(N)}\}$ from a discrete **random variable** X with probability mass function (*pmf*) $p(x|\theta)$ …

- Case 1: The **likelihood** function

$$L(\theta) = p(x^{(1)}|\theta)\, p(x^{(2)}|\theta) \ldots\ p(x^{(N)}|\theta)$$

The **likelihood** tells us how likely one sample is relative to another

- Case 2: The **log-likelihood** function is

$$\log L(\theta) = \ell(\theta) = \log p(x^{(1)}|\theta) + \ldots + \log p(x^{(N)}|\theta)$$

Likelihood Function

Two R.V.s

Given N **iid** samples $D = \{(x^{(1)}, y^{(1)}), \ldots, (x^{(N)}, y^{(N)})\}$ from a pair of **random variables** X, Y where Y is **discrete** with probability mass function (pmf) $p(y \mid x, \theta)$

- Case 3: The **conditional likelihood** function:

$$L(\theta) = p(y^{(1)} \mid x^{(1)}, \theta) \ldots p(y^{(N)} \mid x^{(N)}, \theta)$$

- Case 4: The **conditional log-likelihood** function is

$$\ell(\theta) = \log p(y^{(1)} \mid x^{(1)}, \theta) + \ldots + \log p(y^{(N)} \mid x^{(N)}, \theta)$$

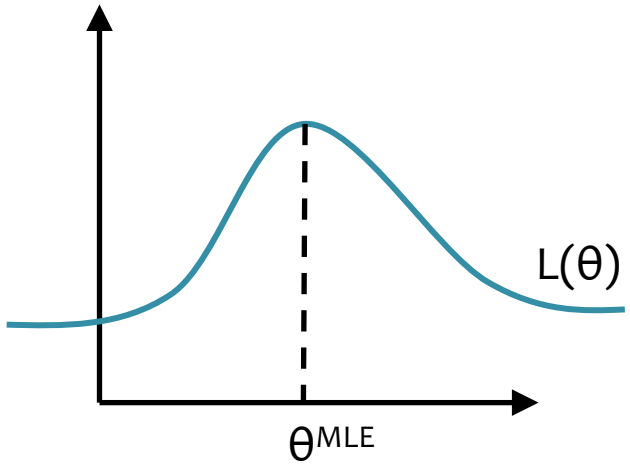
MLE

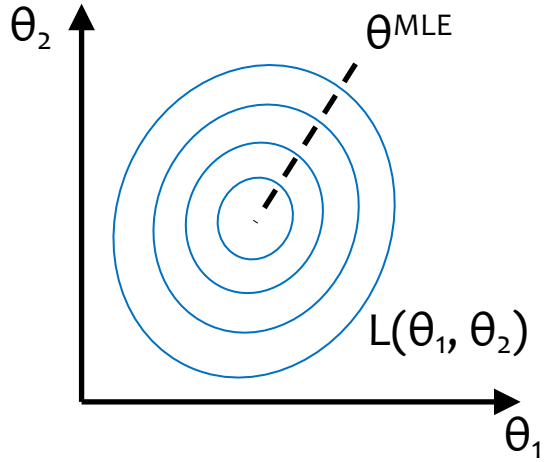
Suppose we have data $\mathcal{D} = \{x^{(i)}\}_{i=1}^{N}$

Principle of Maximum Likelihood Estimation:
Choose the parameters that maximize the likelihood of the data.

$$\boldsymbol{\theta}^{\text{MLE}} = \underset{\boldsymbol{\theta}}{\operatorname{argmax}} \prod_{i=1}^{N} p(\mathbf{x}^{(i)}|\boldsymbol{\theta})$$

Maximum Likelihood Estimate (MLE)





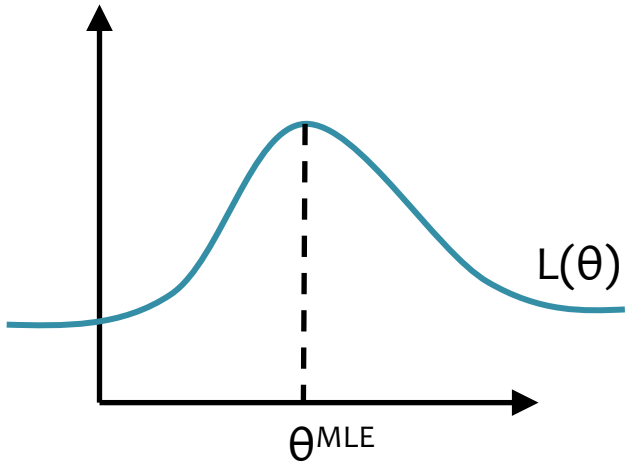
MLE

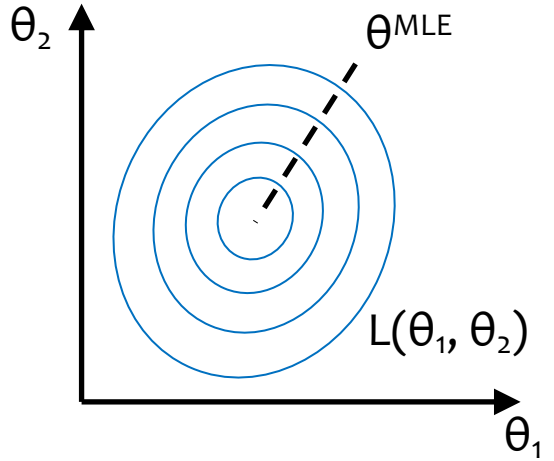
Suppose we have data $\mathcal{D} = \{(y^{(i)}, \mathbf{x}^{(i)})\}_{i=1}^{N}$

Principle of Maximum Likelihood Estimation:
Choose the parameters that maximize the conditional likelihood of the data.

$$\boldsymbol{\theta}^{\text{MLE}} = \underset{\boldsymbol{\theta}}{\operatorname{argmax}} \prod_{i=1}^{N} p(y^{(i)} \mid \mathbf{x}^{(i)}, \boldsymbol{\theta})$$

Maximum Likelihood Estimate (MLE)





MLE

Suppose we have data $\mathcal{D} = \{(y^{(i)}, \mathbf{x}^{(i)})\}_{i=1}^{N}$

Principle of Maximum Likelihood Estimation:

Choose the parameters that maximize the conditional log-likelihood of the data.

$$\boldsymbol{\theta}^{\text{MLE}} = \underset{\boldsymbol{\theta}}{\operatorname{argmax}} \prod_{i=1}^{N} p(y^{(i)} \mid \mathbf{x}^{(i)}, \boldsymbol{\theta})$$

$$= \underset{\boldsymbol{\theta}}{\operatorname{argmax}} \sum_{i=1}^{N} \log p(y^{(i)} \mid \mathbf{x}^{(i)}, \boldsymbol{\theta})$$

MLE

Suppose we have data $\mathcal{D} = \{(y^{(i)}, \mathbf{x}^{(i)})\}_{i=1}^{N}$

Principle of Maximum Likelihood Estimation:

Choose the parameters that minimize the negative conditional log-likelihood of the data.

$$\boldsymbol{\theta}^{\text{MLE}} = \underset{\boldsymbol{\theta}}{\text{argmax}} \prod_{i=1}^{N} p(y^{(i)} \mid \mathbf{x}^{(i)}, \boldsymbol{\theta})$$

$$= \underset{\boldsymbol{\theta}}{\text{argmax}} \sum_{i=1}^{N} \log p(y^{(i)} \mid \mathbf{x}^{(i)}, \boldsymbol{\theta})$$

$$= \underset{\boldsymbol{\theta}}{\text{argmin}} - \sum_{i=1}^{N} \log p(y^{(i)} \mid \mathbf{x}^{(i)}, \boldsymbol{\theta})$$

MLE

What does maximizing likelihood accomplish?

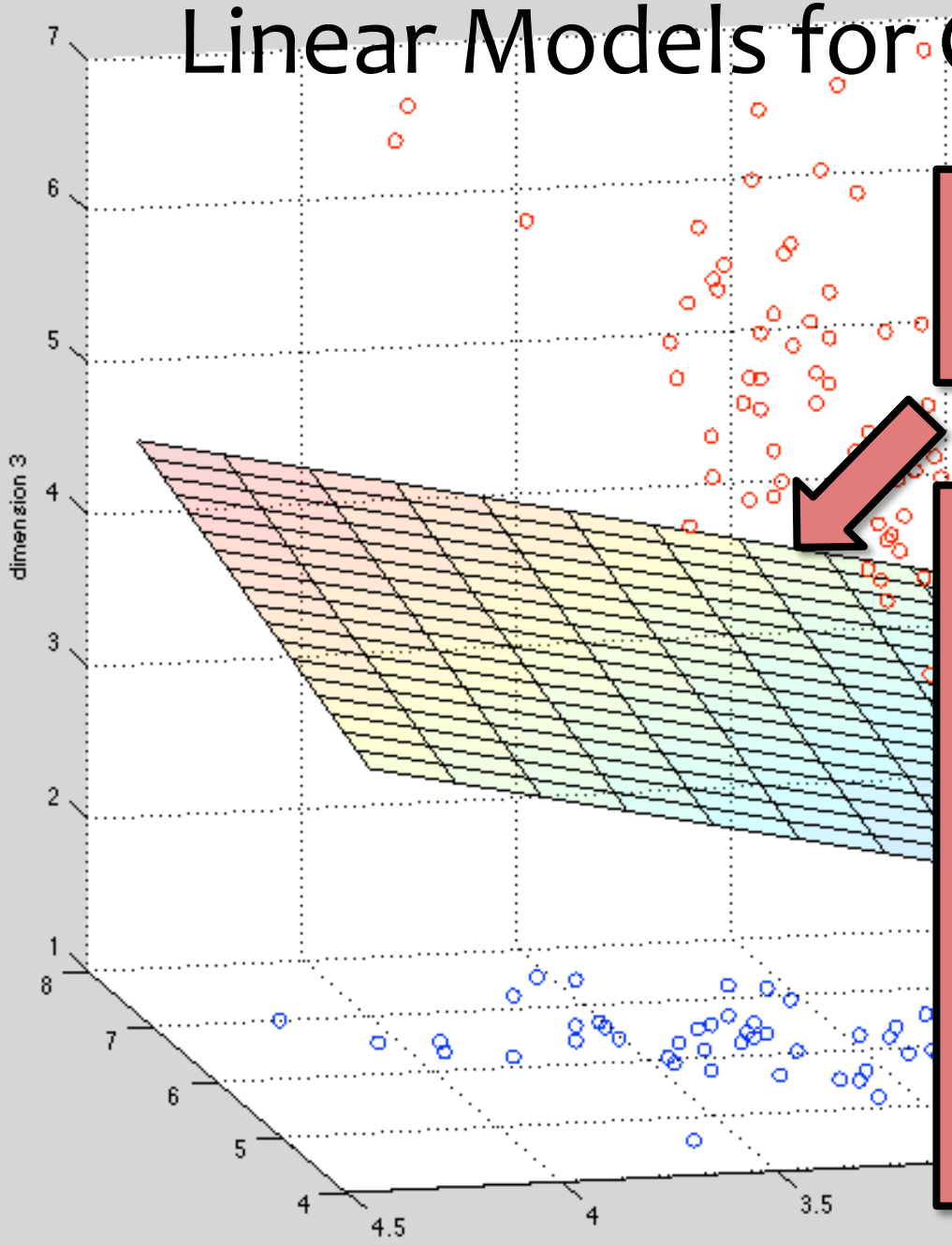
- There is only a finite amount of probability mass (i.e. sum-to-one constraint)
- MLE tries to allocate **as much** probability mass **as possible** to the things we have observed…

  …**at the expense** of the things we have **not** observed

LOGISTIC REGRESSION

Linear Models for Classification

Recall…



Key idea: Try to learn this hyperplane directly

Directly modeling the hyperplane would use a decision function:

$$h(\mathbf{x}) = \text{sign}(\boldsymbol{\theta}^T \mathbf{x})$$

for:

$$y \in \{-1, +1\}$$

Background: Hyperplanes

Recall…

Notation Trick: fold the bias *b* and the weights **w** into a single vector **θ** by prepending a constant to **x** and increasing dimensionality by one to get **x'**!

Hyperplane (Definition 1):

$$\mathcal{H} = \{\mathbf{x} : \mathbf{w}^T\mathbf{x} + b = 0\}$$

Hyperplane (Definition 2):

$$\mathcal{H} = \{\mathbf{x}' : \boldsymbol{\theta}^T\mathbf{x}' = 0 \text{ and } x_1' = 1\}$$

$$\boldsymbol{\theta} = [b, w_1, \ldots, w_M]^T$$

$$\mathbf{x}' = [1, x_1, \ldots, x_M]^T$$

Half-spaces:

$$\mathcal{H}^+ = \{\mathbf{x} : \boldsymbol{\theta}^T\mathbf{x} > 0 \text{ and } x_0^1 = 1\}$$

$$\mathcal{H}^- = \{\mathbf{x} : \boldsymbol{\theta}^T\mathbf{x} < 0 \text{ and } x_0^1 = 1\}$$

Using gradient descent for linear classifiers

Key idea behind today's lecture:

1. Define a linear classifier (logistic regression)
2. Define an objective function (likelihood)
3. Optimize it with gradient descent to learn parameters
4. Predict the class with highest probability under the model

Logistic Regression

Data: Inputs are continuous vectors of length M. Outputs are discrete.

$$\mathcal{D} = \{\mathbf{x}^{(i)}, y^{(i)}\}_{i=1}^{N} \text{ where } \mathbf{x} \in \mathbb{R}^M \text{ and } y \in \{0, 1\}$$

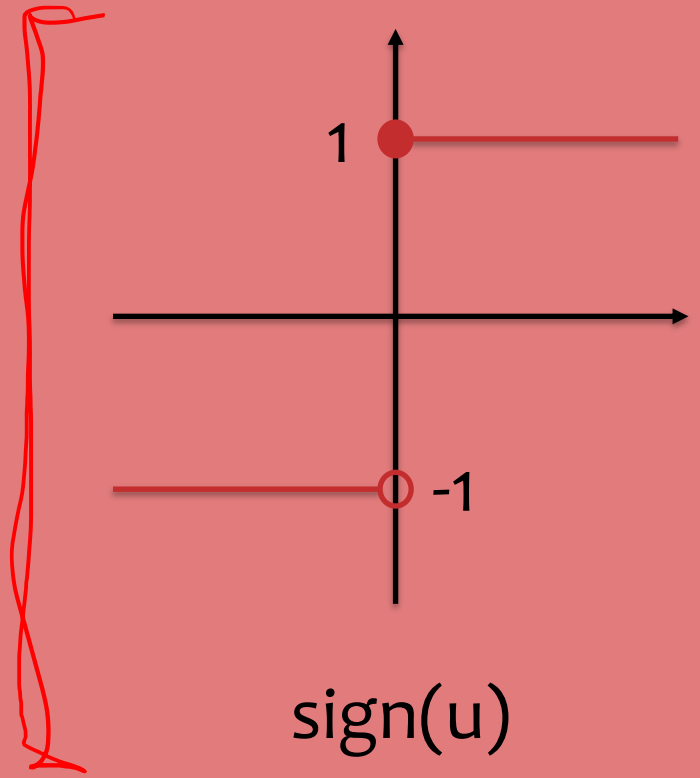
We are back to classification.

Despite the name logistic **regression.**

sign(·) vs. sigmoid(·)

Suppose we wanted to learn a linear classifier, but instead of predicting y ∈ {-1,+1} we wanted to predict y ∈ {0,1}

$$h(\mathbf{x}) = \text{sign}(\boldsymbol{\theta}^T \mathbf{x})$$

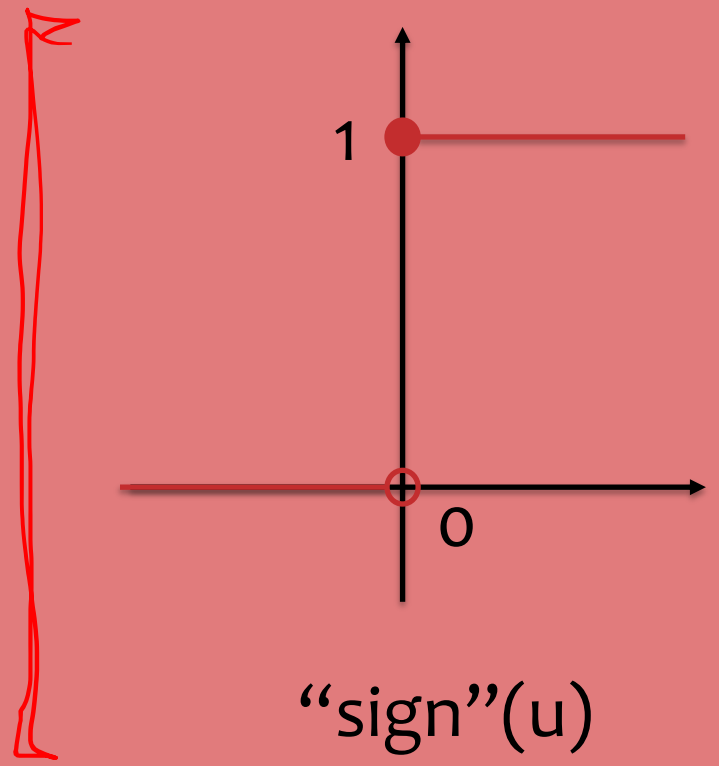


sign(u)

sign(·) vs. sigmoid(·)

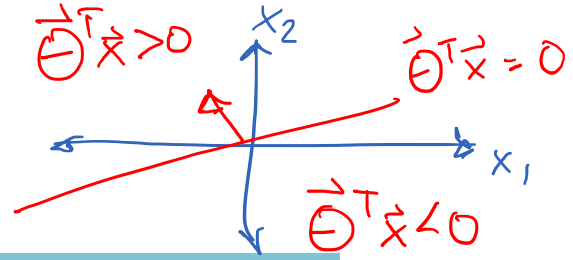
Suppose we wanted to learn a linear classifier, but instead of predicting y ∈ {-1,+1} we wanted to predict y ∈ {0,1}

$$h(\mathbf{x}) = \text{“sign”}(\boldsymbol{\theta}^T \mathbf{x})$$



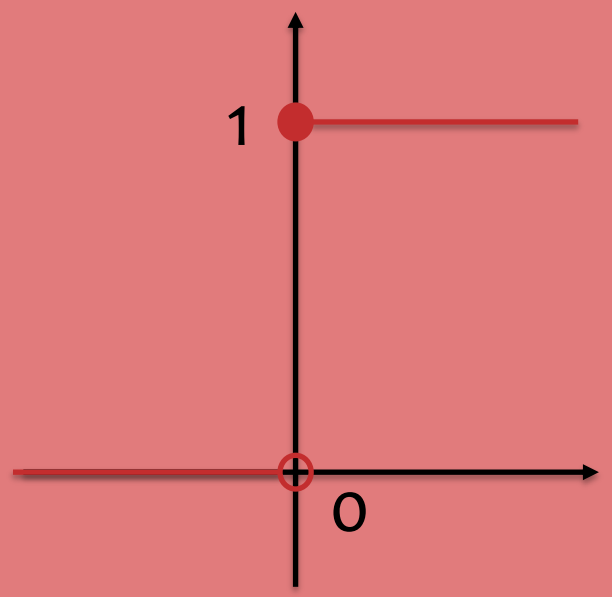
Goal: Learn a linear classifier with Gradient Descent

sign(·) vs. sigmoid(·)



But this decision function isn't differentiable...

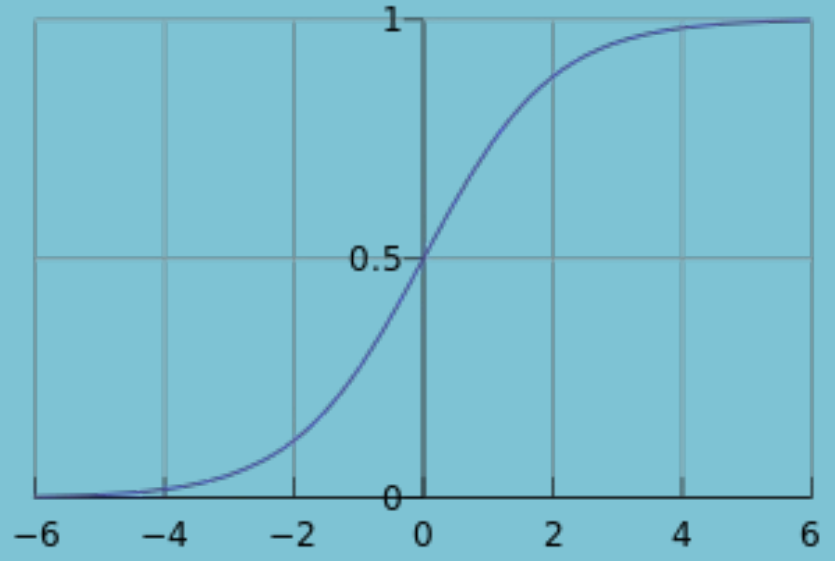
$$h(\mathbf{x}) = \text{“sign”}(\boldsymbol{\theta}^T\mathbf{x})$$



“sign”(u)

Use a differentiable function instead!

$$p_{\boldsymbol{\theta}}(y=1|\mathbf{x}) = \frac{1}{1+\exp(-\boldsymbol{\theta}^T\mathbf{x})} = \text{logistic}(\vec{\theta}^T\vec{x})$$



$$\text{logistic}(u) \equiv \frac{1}{1+e^{-u}}$$

The *logistic* function is also called the *sigmoid* function.

Logistic Regression

Data: Inputs are continuous vectors of length M. Outputs are discrete.

$$\mathcal{D} = \{\mathbf{x}^{(i)}, y^{(i)}\}_{i=1}^{N} \text{ where } \mathbf{x} \in \mathbb{R}^M \text{ and } y \in \{0, 1\}$$

Model: Logistic function applied to dot product of parameters with input vector.

$$p_{\boldsymbol{\theta}}(y = 1|\mathbf{x}) = \frac{1}{1 + \exp(-\boldsymbol{\theta}^T \mathbf{x})}$$

Learning: finds the parameters that minimize some objective function. $\boldsymbol{\theta}^* = \underset{\boldsymbol{\theta}}{\operatorname{argmin}}\, J(\boldsymbol{\theta})$

Prediction: Output is the most probable class.

$$\hat{y} = \underset{y \in \{0,1\}}{\operatorname{argmax}}\, p_{\boldsymbol{\theta}}(y|\mathbf{x})$$

Learning Logistic Regression

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

Approach 0: Random Search
(horridly slow because it lacks gradient information)

Approach 1: Gradient Descent
(take large confident steps opposite the gradient)

Approach 2: Stochastic Gradient Descent (SGD)
(take many small steps roughly opposite the gradient)

~~**Approach 3:** Closed Form
(set derivatives equal to zero and solve for parameters)~~

Logistic Regression does not have a closed form solution for MLE parameters.

Logistic Regression

1. Model

$y \sim \text{Bernoulli}(\phi)$

$\phi = \sigma(\vec{\theta}^T \vec{x})$ where $\sigma(v) = \frac{1}{1+\exp(-v)}$

$$p(y \mid \vec{x}, \theta) = \begin{cases} \sigma(\vec{\theta}^T \vec{x}) & \text{if } y = 1 \\ 1 - \sigma(\vec{\theta}^T \vec{x}) & \text{if } y = 0 \end{cases}$$

2. Objective

$\ell(\theta) =$ cond. log. like.

$$J(\theta) = -\frac{1}{N} \ell(\theta)$$

$$= \frac{1}{N} \sum_{i=1}^{N} \underbrace{-\log p(y^{(i)} \mid \vec{x}^{(i)}, \vec{\theta})}_{J^{(i)}(\vec{\theta})}$$

Logistic Regression

3A. Derivatives

$$\frac{\partial J^{(i)}(\theta)}{\partial \theta_m} = \frac{\partial}{\partial \theta_m}\left(-\log p(y^{(i)} \mid \vec{x}^{(i)}, \theta)\right)$$

$$= \begin{cases} \frac{\partial}{\partial \theta_m}\left(-\log \sigma(\vec{\theta}^T \vec{x})\right) & \text{if } y^{(i)} = 1 \\ \frac{\partial}{\partial \theta_m}\left(-\log\left(1 - \sigma(\vec{\theta}^T \vec{x})\right)\right) & \text{if } y^{(i)} = 0 \end{cases}$$

$= \cdots$

$= \cdots$ } HW

$= \cdots$

$= -\left(y^{(i)} - \sigma(\vec{\theta}^T \vec{x}^{(i)})\right) x_m^{(i)}$

3B. Gradients

Logistic Regression

4. Optimization

5. Prediction