



# Stochastic Gradient Descent + Binary Logistic Regression

Geoff Gordon

Lecture 9

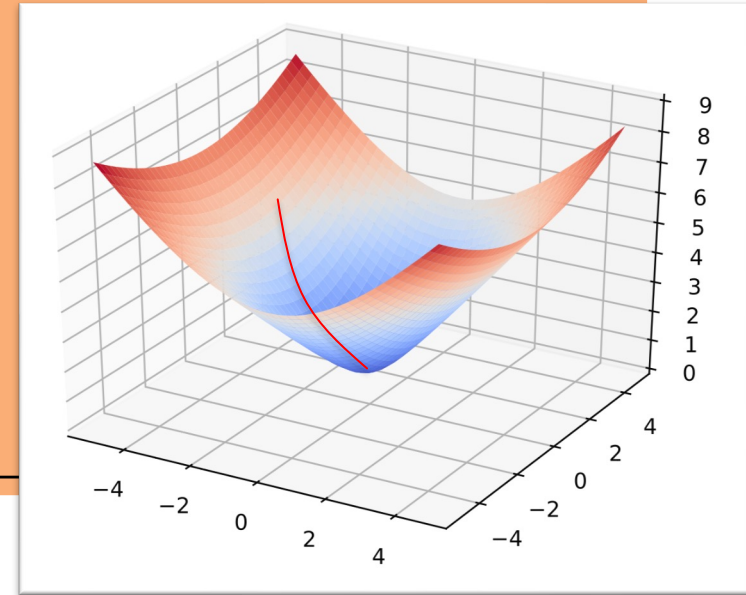
with thanks to Matt Gormley

# **OPTIMIZATION METHOD #3: STOCHASTIC GRADIENT DESCENT**

# Gradient Descent

## Algorithm 1 Gradient Descent

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



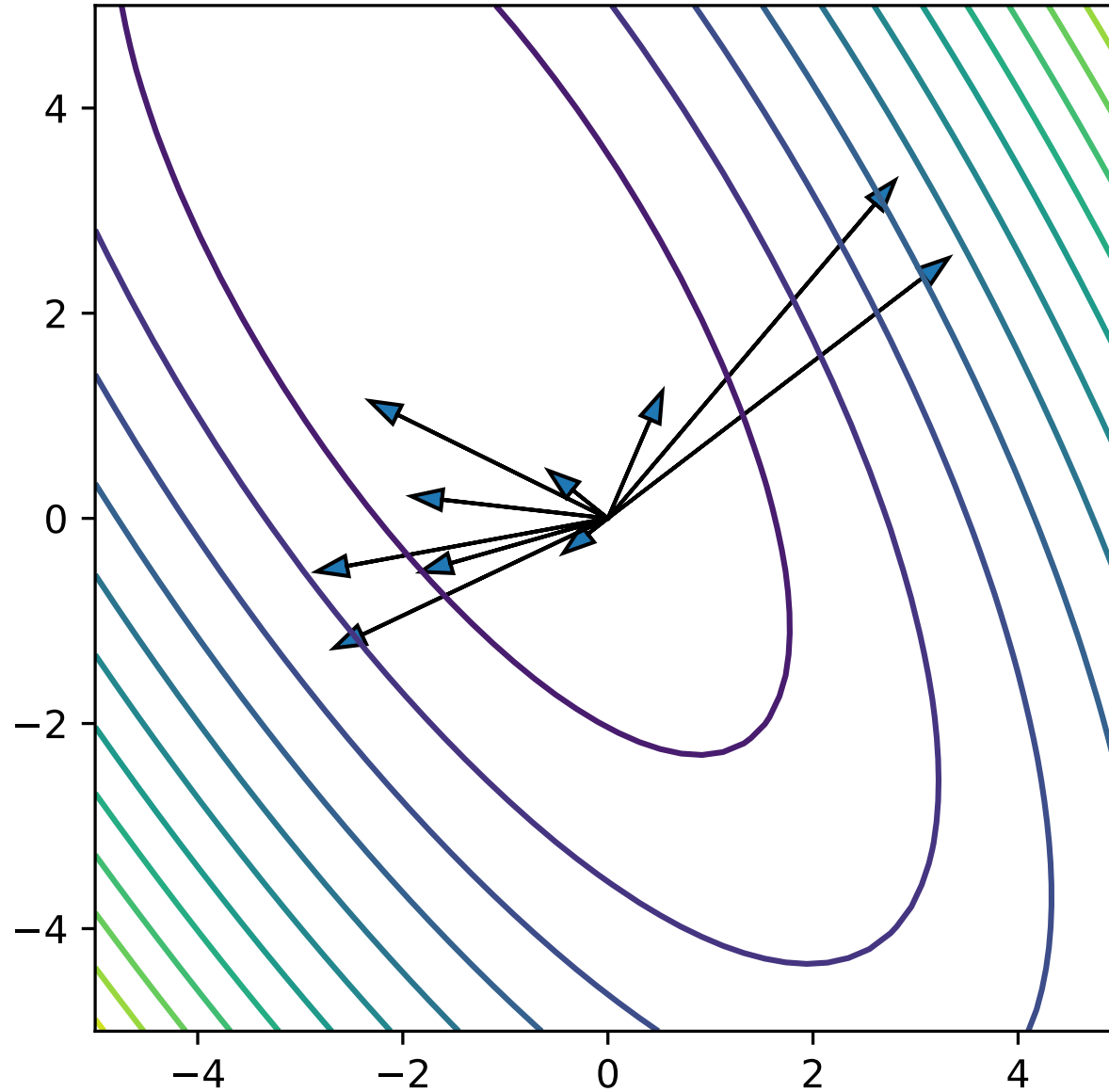
per-example objective:

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

full objective:

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

# Gradient is an average

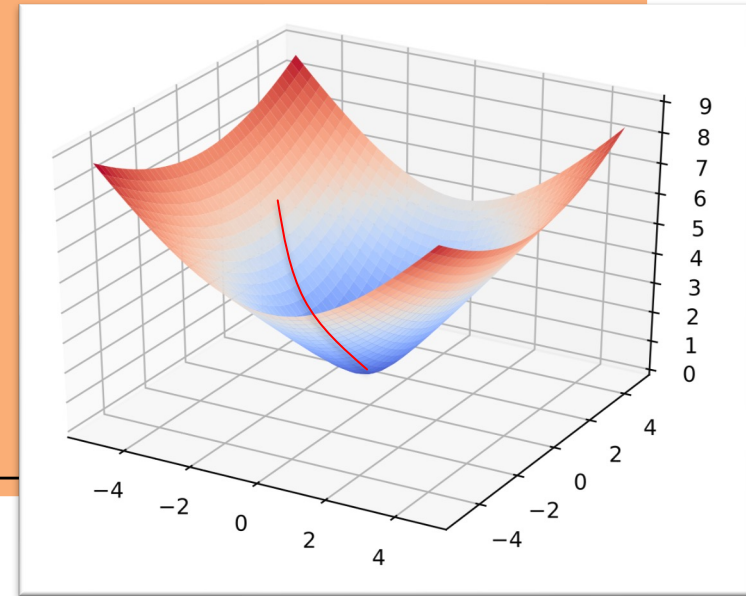


Contours: loss  
Arrows: individual  
negative gradient terms

# Gradient Descent

## Algorithm 1 Gradient Descent

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



per-example objective:

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

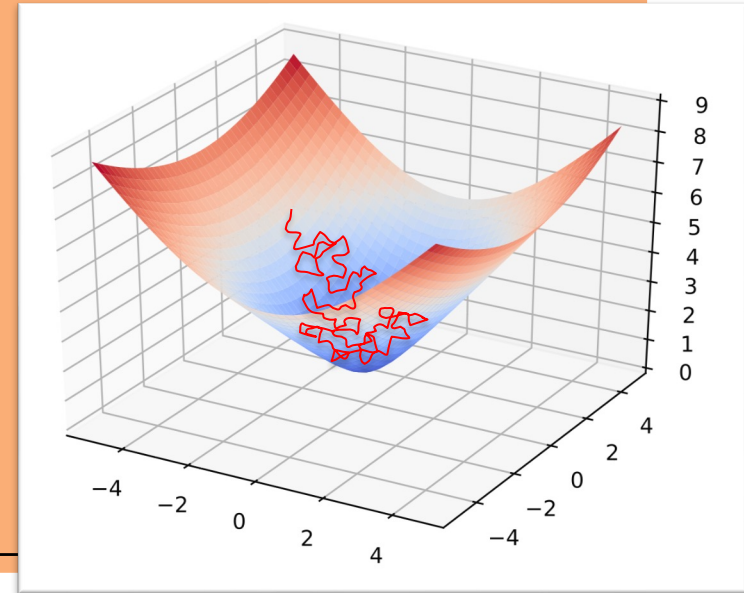
full objective:

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

# Stochastic Gradient Descent (SGD)

## Algorithm 2 Stochastic Gradient Descent (SGD)

```
1: procedure SGD( $\mathcal{D}, \theta^{(0)}$ )  
2:    $\theta \leftarrow \theta^{(0)}$   
3:   while not converged do  
4:      $i \sim \text{Uniform}(\{1, 2, \dots, N\})$   
5:      $\theta \leftarrow \theta - \gamma \nabla_{\theta} J^{(i)}(\theta)$   
6:   return  $\theta$ 
```



per-example objective:

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

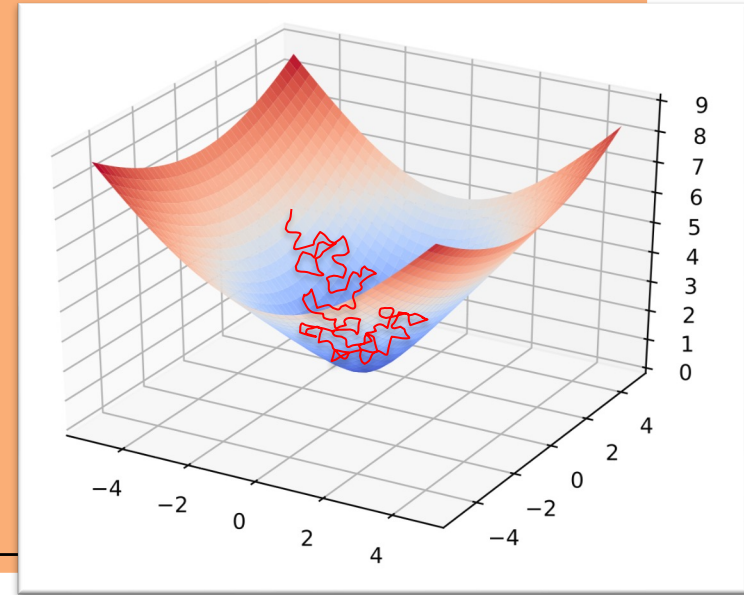
full objective:

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

# Stochastic Gradient Descent (SGD)

## Algorithm 2 Stochastic Gradient Descent (SGD)

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



per-example objective:

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

full objective:

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

In practice, it is common to use sampling **without** replacement (i.e.,  $\text{shuffle}(\{1, 2, \dots, N\})$ ), even though most of the theory is for sampling **with** replacement (i.e.,  $\text{Uniform}(\{1, 2, \dots, N\})$ )

# Why does SGD work?

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

For any discrete random variable  $X$

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

## Objective Function for SGD

We assume the form to be:

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

## Expectation of a Stochastic Gradient:

- If we sample examples uniformly at random, the expected value of the pointwise gradient is the same as the full gradient!

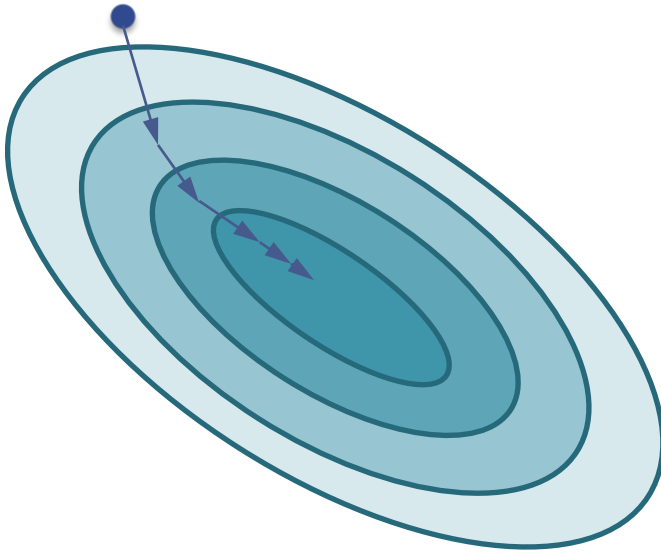
$$\begin{aligned} E[\nabla_{\boldsymbol{\theta}} J^{(i)}(\boldsymbol{\theta})] &= \sum_{i=1}^N (\text{probability of selecting } \mathbf{x}^{(i)}, y^{(i)}) \nabla_{\boldsymbol{\theta}} J^{(i)}(\boldsymbol{\theta}) \\ &= \sum_{i=1}^N \left(\frac{1}{N}\right) \nabla_{\boldsymbol{\theta}} J^{(i)}(\boldsymbol{\theta})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

- If we randomly shuffle and loop through, each data point is used **exactly** equally often

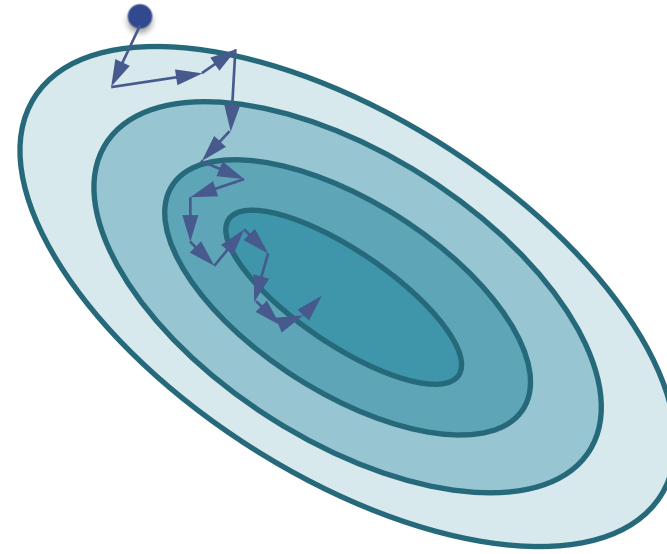


# **SGD VS. GRADIENT DESCENT**

# SGD vs. Gradient Descent



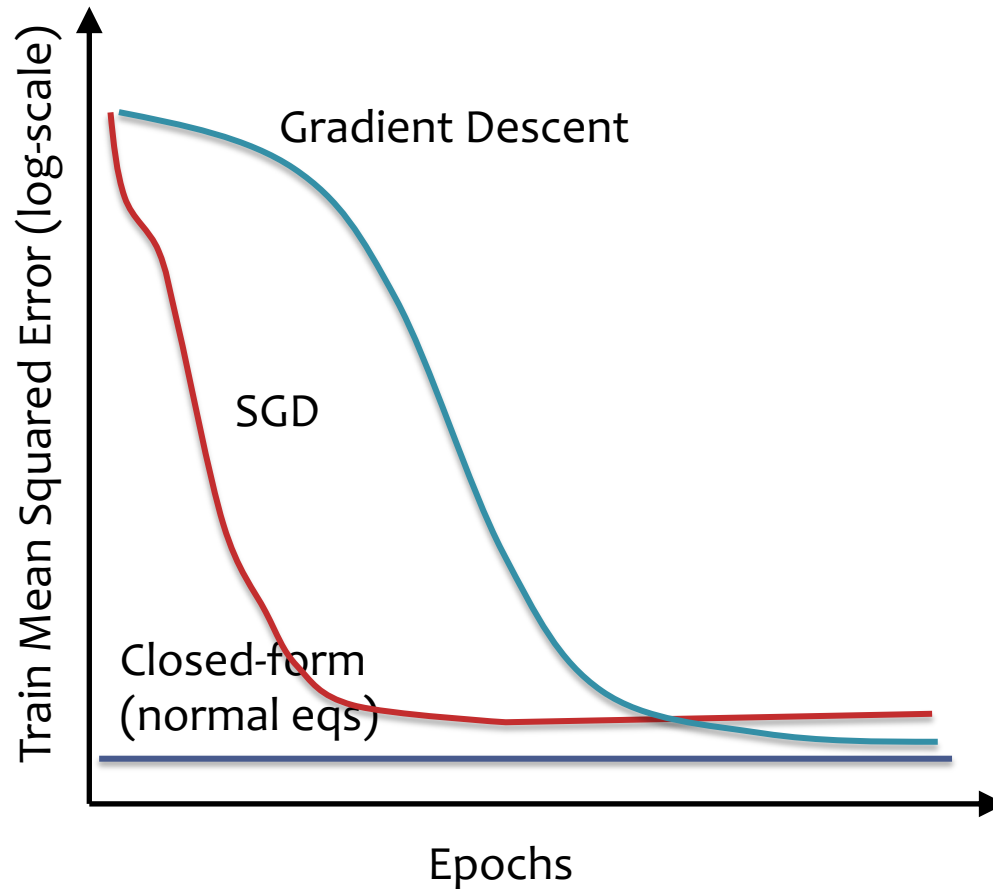
Gradient Descent



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  
 $N = (\# \text{ train examples})$

- 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  $J(\boldsymbol{\theta}^{(t)}) - J(\boldsymbol{\theta}^*) < \epsilon$

Method	Steps to Convergence	Computation per Step
Gradient descent	$O(\log 1/\epsilon)$	$O(NM)$
SGD	$O(1/\epsilon)$	$O(M)$

(with high probability under certain assumptions)

**Main Takeaway:** SGD takes many more steps, especially for small  $\epsilon$ ; SGD takes much better time per step;  
... at  $\epsilon$  we care about, tradeoff is that SGD can be much faster

# **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:

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

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)*

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

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

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

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

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

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

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

Gradient of  $J(\boldsymbol{\theta})$

[used by Gradient Descent]

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

# SGD for Linear Regression

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

---

**Algorithm 1** Least Mean Squares (LMS)

---

```
1: procedure LMS( $\mathcal{D}, \theta^{(0)}$ )  
2:    $\theta \leftarrow \theta^{(0)}$  ▷ Initialize parameters  
3:   while not converged do  
4:     for  $i \in \text{shuffle}(\{1, 2, \dots, N\})$  do  
5:        $\mathbf{g} \leftarrow (\theta^T \mathbf{x}^{(i)} - y^{(i)}) \mathbf{x}^{(i)}$  ▷ Compute gradient  
6:        $\theta \leftarrow \theta - \gamma \mathbf{g}$  ▷ Update parameters  
7:   return  $\theta$ 
```

---



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

---

# Optimization (Learning) Objectives

*You should be able to...*

- Apply gradient descent to optimize a function
- Apply stochastic gradient descent (SGD) to optimize a function
- Set gradient to zero to identify a closed-form solution (if one exists) to an optimization problem
- Distinguish between (strictly/non-strictly) 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  $h(\mathbf{x})$  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 conditional probability distributions  $p(y|\mathbf{x})$  that best approximate  $p^*(y|\mathbf{x})$

# **MAXIMUM LIKELIHOOD ESTIMATION**

# Likelihood Function

One R.V. per example

Given  $N$  **independent, identically distributed (iid)** samples

$D = \{x^{(1)}, x^{(2)}, \dots, x^{(N)}\}$  from a discrete **random variable**  $X$  with probability mass function (pmf)  $p(x|\theta)$  ...

- Case 1: The **likelihood** function

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

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

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

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

# Likelihood Function

Two R.V.s per example

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

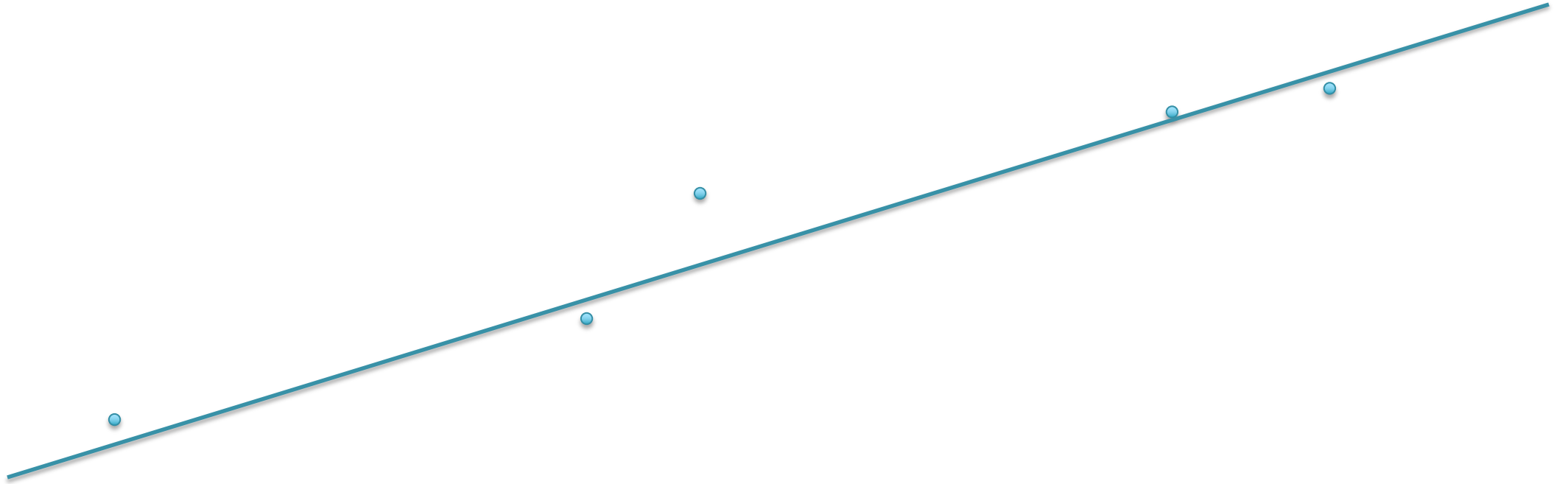
- Case 3: The **conditional likelihood** function:

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

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

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

# E.g., linear regression



We've been thinking of linear regression as just function optimization  
Could give it a conditional probability interpretation too:  $p(y \mid \mathbf{x})$  is high near the line, low away from the line  
(we'll see later that the two interpretations can be equivalent)



# MLE

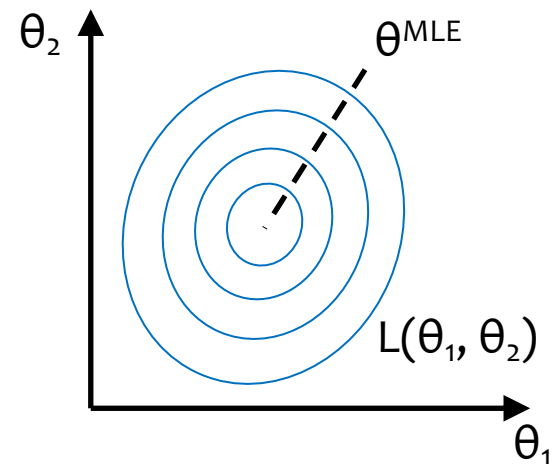
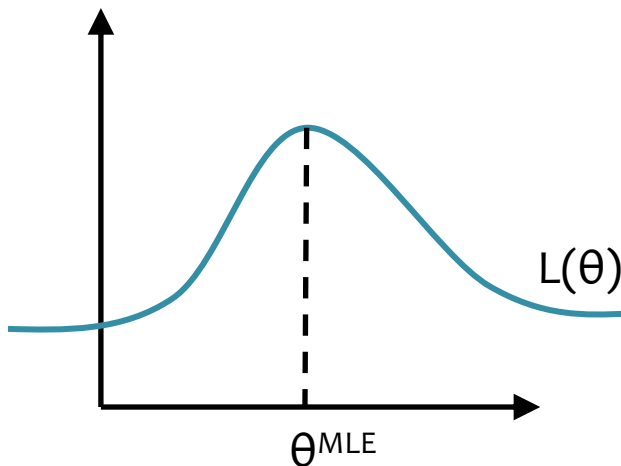
Suppose we have data  $\mathcal{D} = \{x^{(i)}\}_{i=1}^N$

## 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

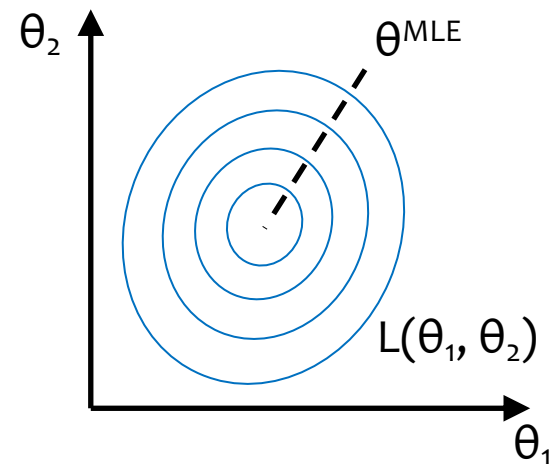
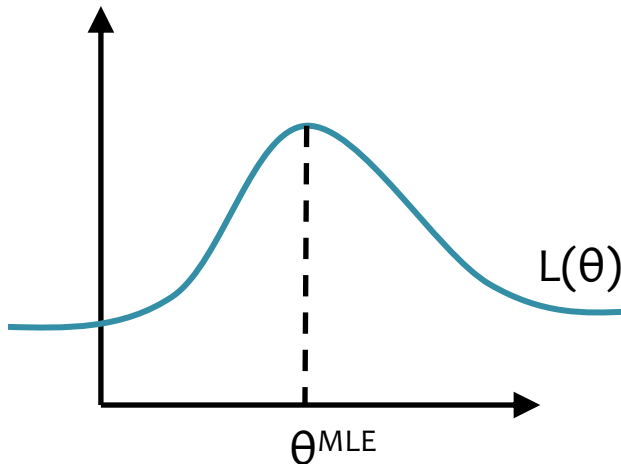
Suppose we have data  $\mathcal{D} = \{(y^{(i)}, \mathbf{x}^{(i)})\}_{i=1}^N$

## 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.

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

# 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.

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

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

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

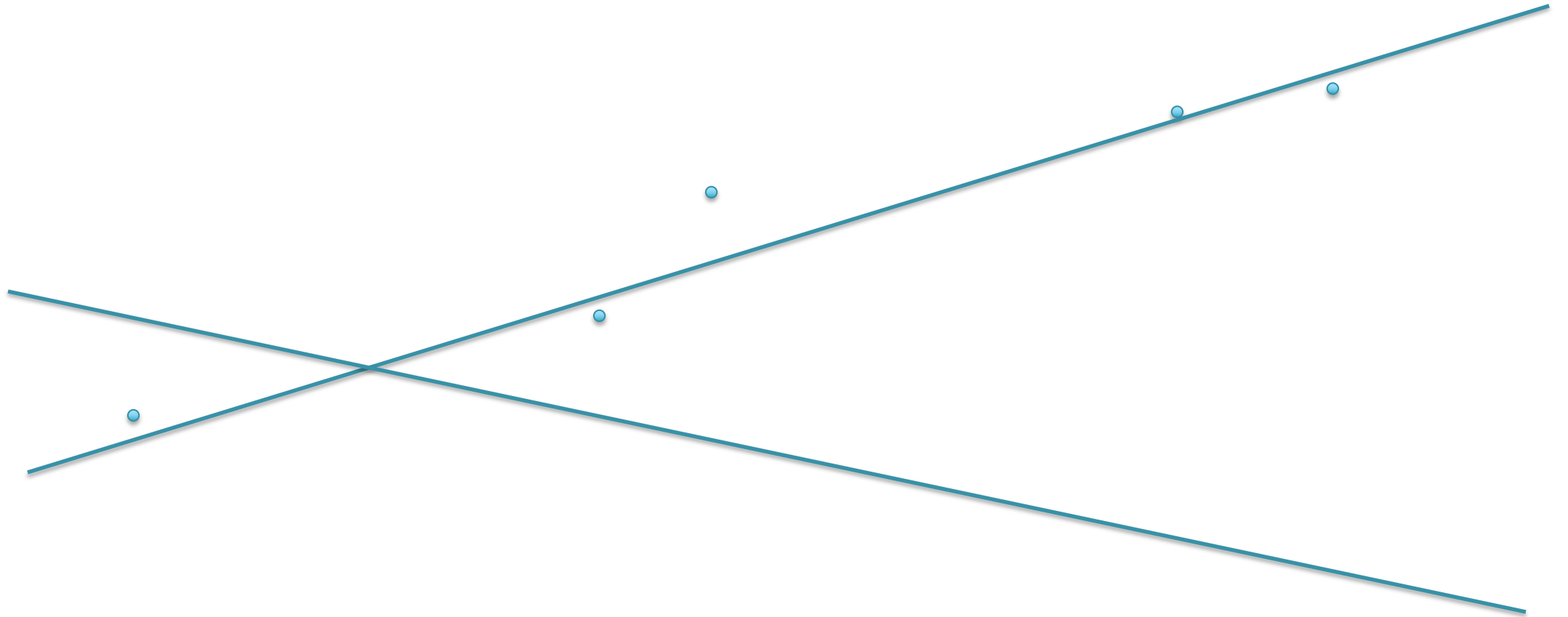
# MLE

What does maximizing likelihood accomplish?

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

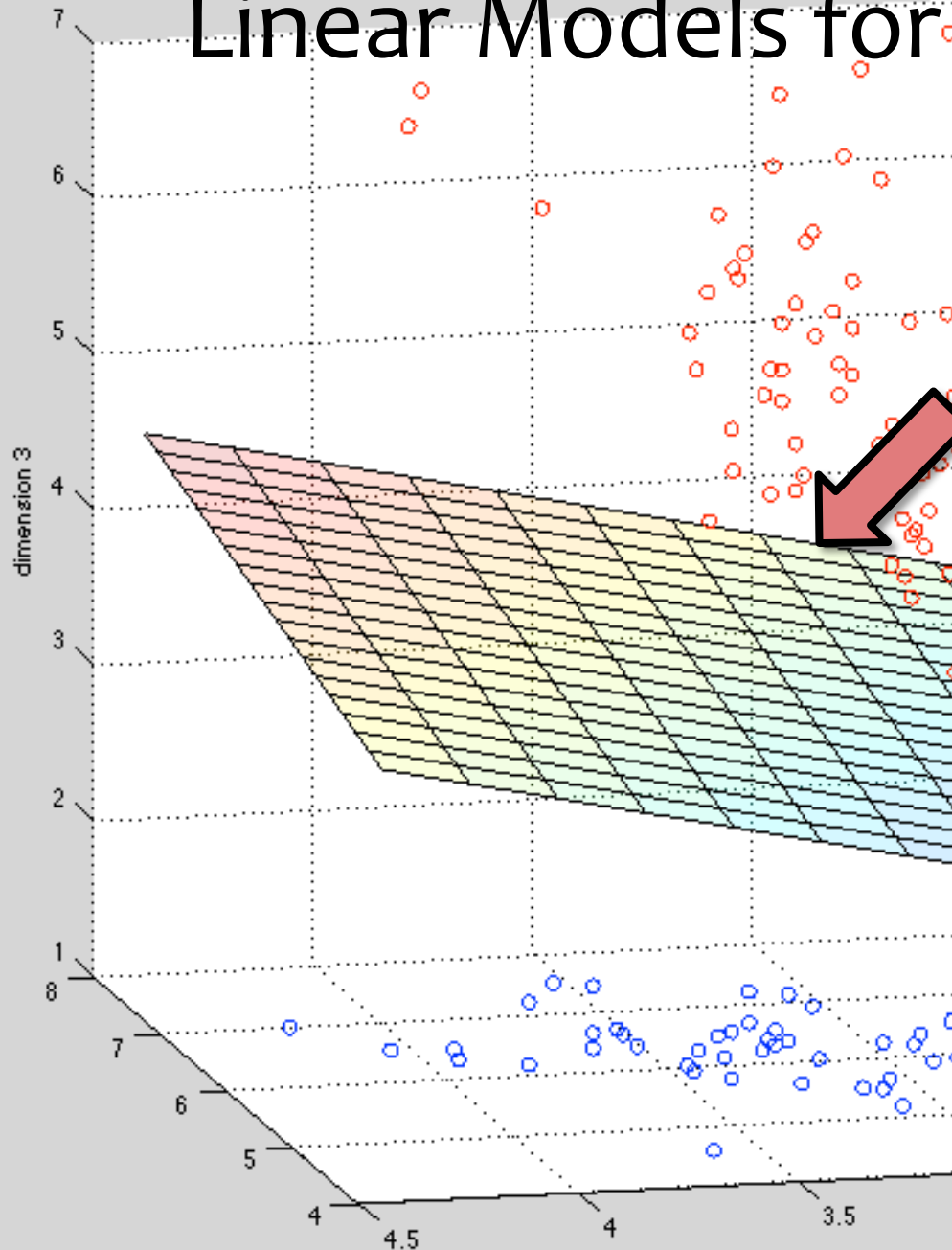
**...at the expense** of the things we have **not** observed

For example, linear regression MLE



# **LOGISTIC REGRESSION**

# Linear Models for Classification



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:

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



Recall...

# Background: Hyperplanes

*Notation Trick:* fold the bias  $b$  and the weights  $\mathbf{w}$  into a single vector  $\boldsymbol{\theta}$  by prepending a constant to  $\mathbf{x}$  and increasing dimensionality by one to get  $\mathbf{x}'$ !

Hyperplane (Definition 1):

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

Hyperplane (Definition 2):

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

$$\text{and } x'_1 = 1\}$$

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

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

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.

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



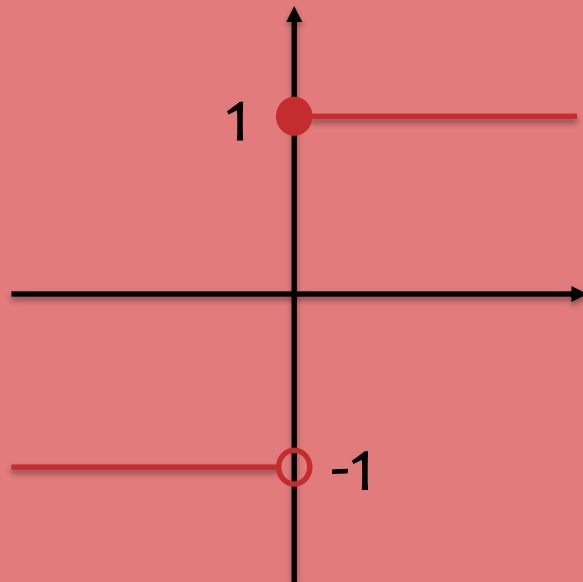
We are back to  
classification.

Despite the name  
logistic **regression**.

# sign( $\cdot$ ) vs. sigmoid( $\cdot$ )

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

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

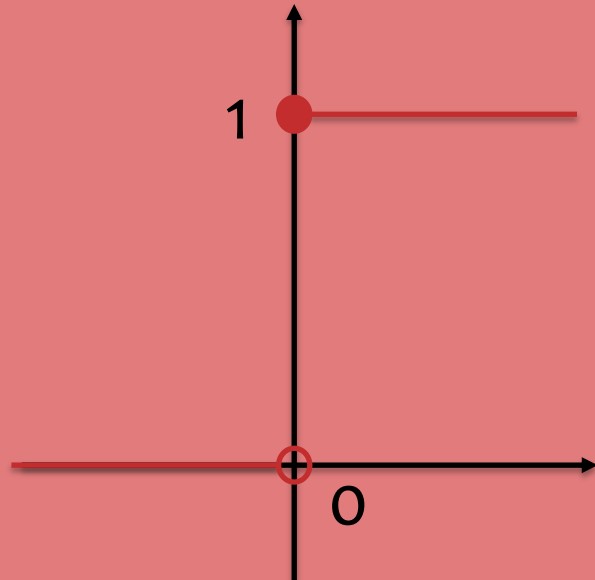


$\text{sign}(u)$

# sign( $\cdot$ ) vs. sigmoid( $\cdot$ )

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

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

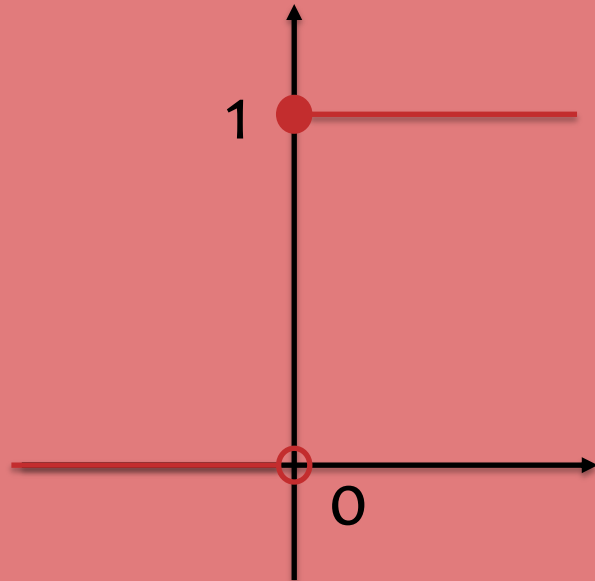


**Goal:** Learn a linear classifier with Gradient Descent

# sign( $\cdot$ ) vs. sigmoid( $\cdot$ )

But this decision function has derivative 0 (mostly)...

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



“sign”(u)

Use a smoother function instead!

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



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

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

**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**

**2. Objective**

# Logistic Regression

**3A. Derivatives**

**3B. Gradients**

# Logistic Regression

**4. Optimization**

**5. Prediction**