

# Logistic Regression

Background: Generative and  
Discriminative Classifiers

# Logistic Regression

Important analytic tool in natural and social sciences

Baseline supervised machine learning tool for classification

Is also the foundation of neural networks

# Generative and Discriminative Classifiers

Naïve Bayes is a **generative classifier**

by contrast:

Logistic regression is a **discriminative classifier**

# Generative and Discriminative Classifiers

Suppose we're distinguishing cat from dog images



imagenet



imagenet

# Generative Classifier:

- Build a model of what's in a cat image
  - Knows about whiskers, ears, eyes
  - Assigns a probability to any image:
    - how cat-y is this image?



Also build a model for dog images

Now given a new image:

**Run both models and see which one fits better**

# Discriminative Classifier

Just try to distinguish dogs from cats



Oh look, dogs have collars!  
Let's ignore everything else

# Finding the correct class $c$ from a document $d$ in Generative vs Discriminative Classifiers

## Naive Bayes

$$\hat{c} = \operatorname{argmax}_{c \in C} \underbrace{\widehat{P(d|c)}}_{\text{likelihood}} \underbrace{\widehat{P(c)}}_{\text{prior}}$$

## Logistic Regression

$$\hat{c} = \operatorname{argmax}_{c \in C} \underbrace{P(c|d)}_{\text{posterior}}$$

# Components of a probabilistic machine learning classifier

Given  $m$  input/output pairs  $(x^{(i)}, y^{(i)})$ :

1. A **feature representation** of the input. For each input observation  $x(i)$ , a vector of features  $[x_1, x_2, \dots, x_n]$ . Feature  $i$  for input  $x^{(j)}$  is  $x_i$ , more completely  $x_i^{(j)}$ , or sometimes  $f_i(x)$ .
2. A **classification function** that computes  $\hat{y}$ , the estimated class, via  $p(y|x)$ , like the **sigmoid** or **softmax** functions.
3. An objective function for learning, like **cross-entropy loss**.
4. An algorithm for optimizing the objective function: **stochastic gradient descent**.

# The two phases of Logistic Regression

**Training:** we learn weights  $w$  and  $b$  using stochastic gradient descent and cross-entropy loss.

**Test:** Given a test example  $x$  we compute  $p(y|x)$  using learned weights  $w$  and  $b$ , and return whichever label ( $y = 1$  or  $y = 0$ ) is higher probability

# Logistic Regression

Background: Generative and  
Discriminative Classifiers

# Logistic Regression

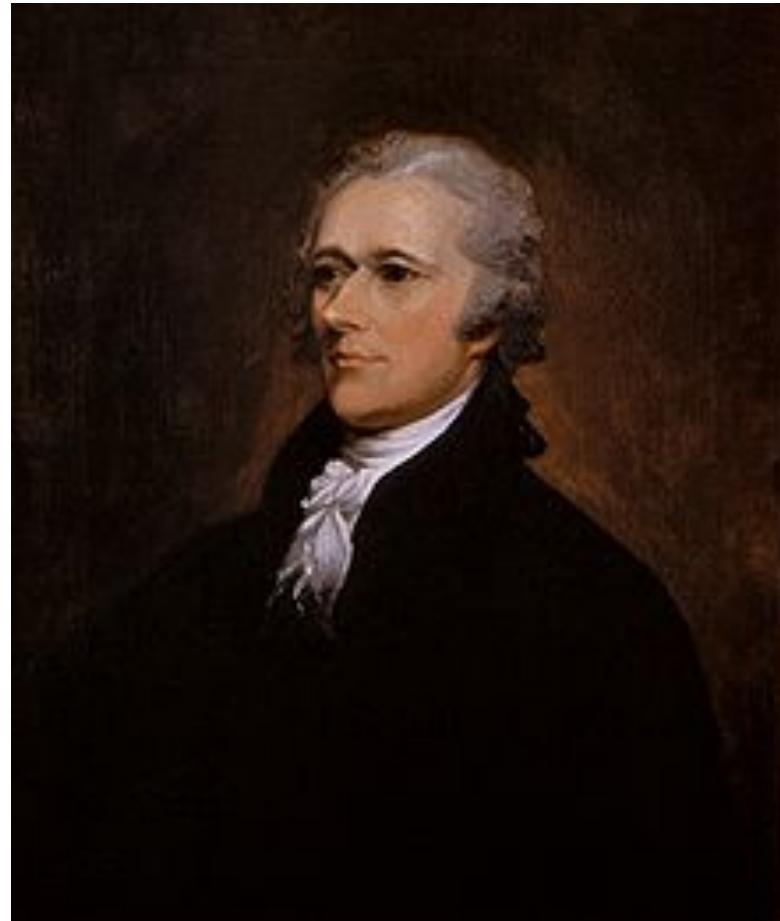
## Classification in Logistic Regression

# Classification Reminder

Positive/negative sentiment

Spam/not spam

Authorship attribution  
(Hamilton or Madison?)



Alexander Hamilton

# Text Classification: definition

*Input:*

- a document  $x$
- a fixed set of classes  $C = \{c_1, c_2, \dots, c_J\}$

*Output:* a predicted class  $y \in C$

# Binary Classification in Logistic Regression

Given a series of input/output pairs:

- $(x^{(i)}, y^{(i)})$

For each observation  $x^{(i)}$

- We represent  $x^{(i)}$  by a **feature vector**  $[x_1, x_2, \dots, x_n]$
- We compute an output: a predicted class  $\hat{y}^{(i)} \in \{0,1\}$

# Features in logistic regression

- For feature  $x_i$ , weight  $w_i$  tells us how important is  $x_i$ 
  - $x_i = \text{"review contains 'awesome'"}: w_i = +10$
  - $x_j = \text{"review contains 'abysmal'"}: w_j = -10$
  - $x_k = \text{"review contains 'mediocre'"}: w_k = -2$

# Logistic Regression for one observation $x$

Input observation: vector  $x = [x_1, x_2, \dots, x_n]$

Weights: one per feature:  $W = [w_1, w_2, \dots, w_n]$

- Sometimes we call the weights  $\theta = [\theta_1, \theta_2, \dots, \theta_n]$

Output: a predicted class  $\hat{y} \in \{0, 1\}$

(multinomial logistic regression:  $y \in \{0, 1, 2, 3, 4\}$ )

# How to do classification

For each feature  $x_i$ , weight  $w_i$  tells us importance of  $x_i$

- (Plus we'll have a bias  $b$ )

We'll sum up all the weighted features and the bias

$$z = \left( \sum_{i=1}^n w_i x_i \right) + b$$
$$z = w \cdot x + b$$

If this sum is high, we say  $y=1$ ; if low, then  $y=0$

But we want a probabilistic classifier

We need to formalize “sum is high”.

We'd like a principled classifier that gives us a probability, just like Naive Bayes did

We want a model that can tell us:

$$p(y=1|x; \theta)$$

$$p(y=0|x; \theta)$$

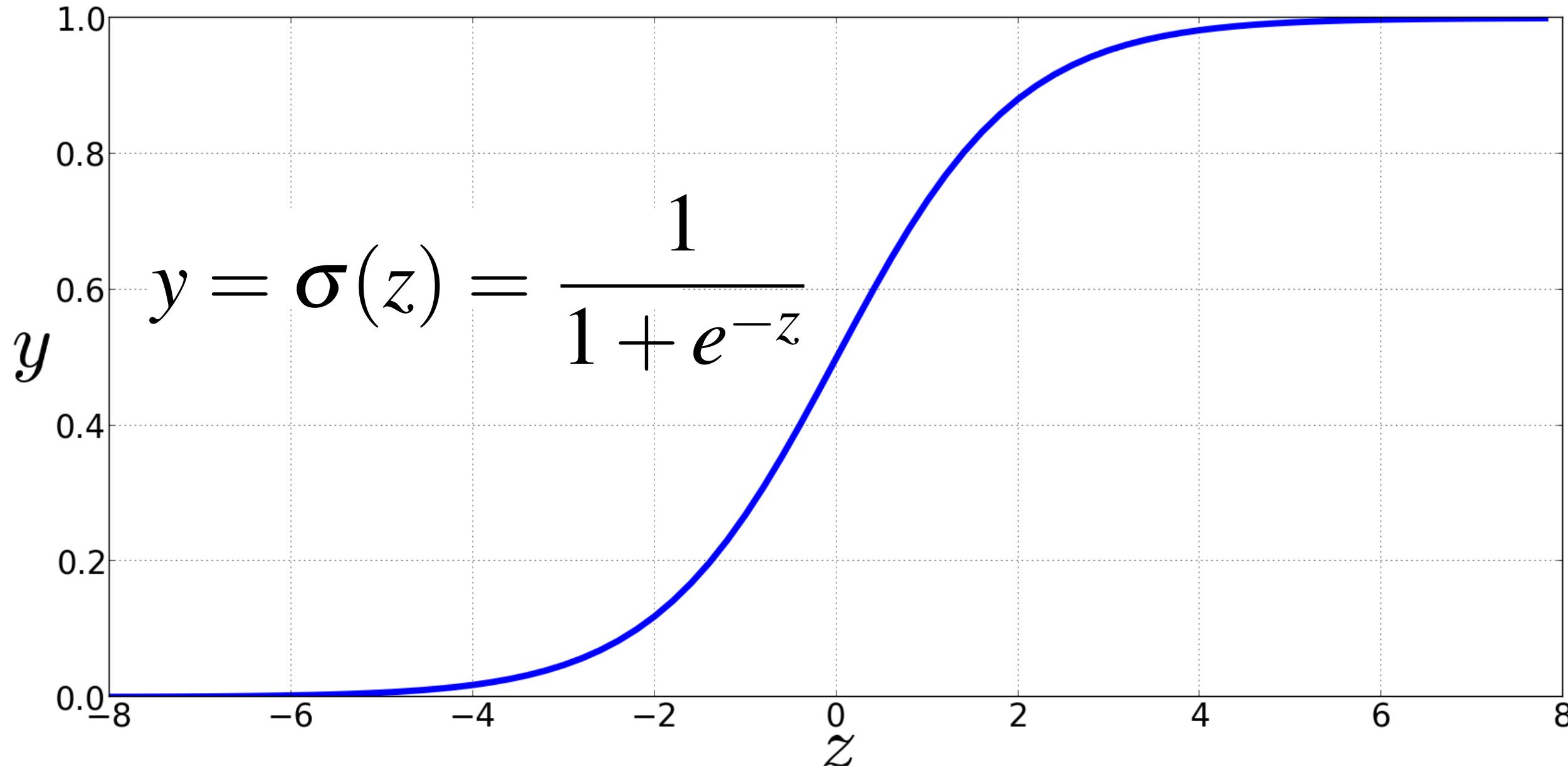
The problem:  $z$  isn't a probability, it's just a number!

$$z = w \cdot x + b$$

Solution: use a function of  $z$  that goes from 0 to 1

$$y = \sigma(z) = \frac{1}{1 + e^{-z}} = \frac{1}{1 + \exp(-z)}$$

# The very useful sigmoid or logistic function



# Idea of logistic regression

We'll compute  $w \cdot x + b$

And then we'll pass it through the sigmoid function:

$$\sigma(w \cdot x + b)$$

And we'll just treat it as a probability

# Making probabilities with sigmoids

$$\begin{aligned} P(y = 1) &= \sigma(w \cdot x + b) \\ &= \frac{1}{1 + \exp(-(w \cdot x + b))} \end{aligned}$$

$$\begin{aligned} P(y = 0) &= 1 - \sigma(w \cdot x + b) \\ &= 1 - \frac{1}{1 + \exp(-(w \cdot x + b))} \\ &= \frac{\exp(-(w \cdot x + b))}{1 + \exp(-(w \cdot x + b))} \end{aligned}$$

By the way:

$$\begin{aligned} P(y=0) &= 1 - \sigma(w \cdot x + b) & = \sigma(-(w \cdot x + b)) \\ &= 1 - \frac{1}{1 + \exp(-(w \cdot x + b))} \\ &= \frac{\exp(-(w \cdot x + b))}{1 + \exp(-(w \cdot x + b))} \end{aligned}$$

Because

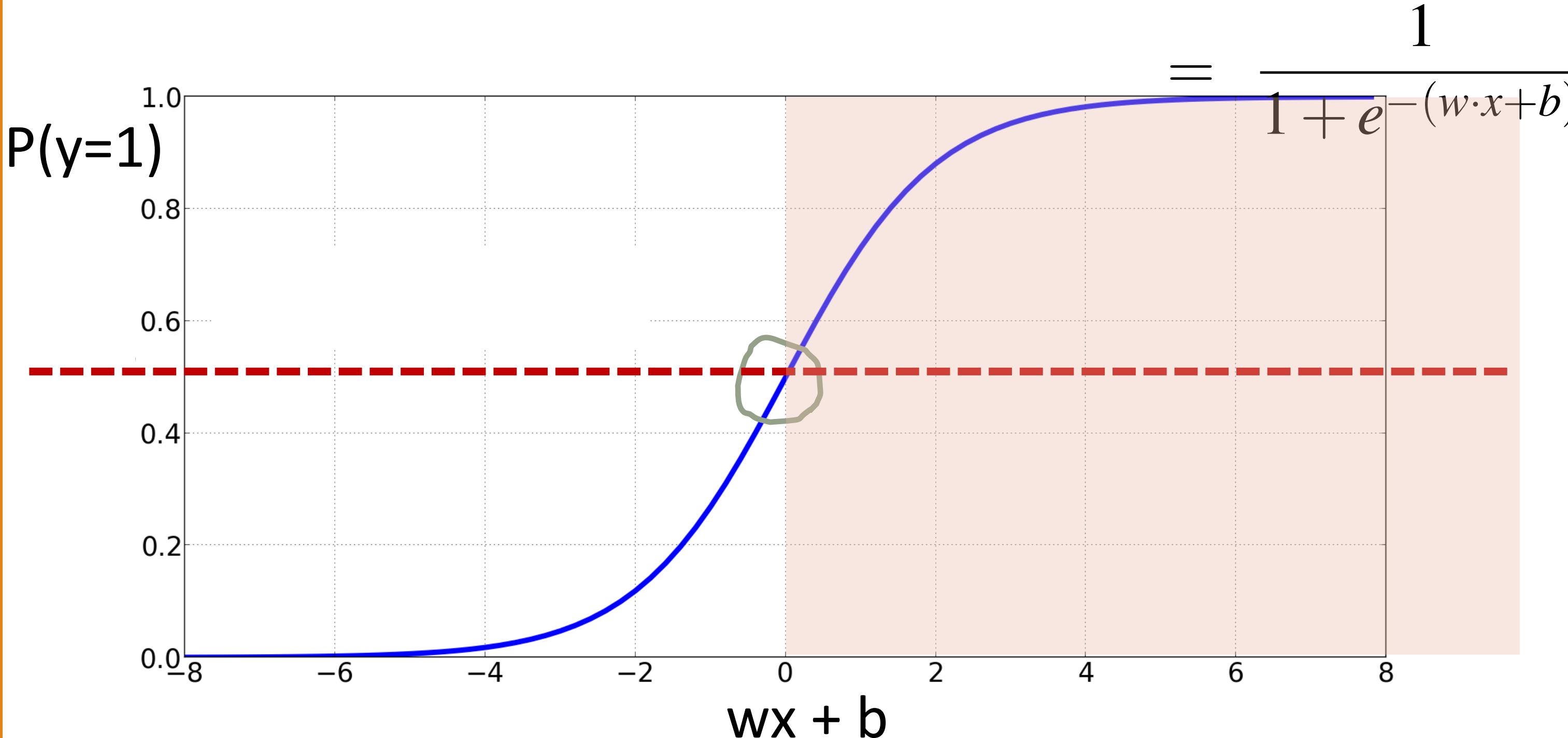
$$1 - \sigma(x) = \sigma(-x)$$

# Turning a probability into a classifier

$$\hat{y} = \begin{cases} 1 & \text{if } P(y=1|x) > 0.5 \\ 0 & \text{otherwise} \end{cases}$$

0.5 here is called the **decision boundary**

The probabilistic classifier  $P(y = 1) = \sigma(w \cdot x + b)$



# Turning a probability into a classifier

$$\hat{y} = \begin{cases} 1 & \text{if } P(y=1|x) > 0.5 \\ 0 & \text{otherwise} \end{cases}$$

if  $w \cdot x + b > 0$   
if  $w \cdot x + b \leq 0$

# Logistic Regression

## Classification in Logistic Regression

# Logistic Regression

Logistic Regression: A text  
(sentiment classification)  
example

# Sentiment example: does $y=1$ or $y=0$ ?

It's hokey . There are virtually no surprises , and the writing is second-rate . So why was it so enjoyable ? For one thing , the cast is great . Another nice touch is the music . I was overcome with the urge to get off the couch and start dancing . It sucked me in , and it'll do the same to you .

It's **hokey**. There are virtually **no** surprises , and the writing is **second-rate**.  
 So why was it so **enjoyable**? For one thing , the cast is  
**great**. Another **nice** touch is the music **I** was overcome with the urge to get off  
 the couch and start dancing . It sucked **me** in , and it'll do the same to **you** .

$$x_1=3$$

$$x_5=0$$

$$x_6=4.19$$

$$x_4=3$$

$$x_3=1$$

$$x_2=2$$

Var	Definition	Value in Fig. 5.2
$x_1$	count(positive lexicon) $\in$ doc)	3
$x_2$	count(negative lexicon) $\in$ doc)	2
$x_3$	$\begin{cases} 1 & \text{if “no”} \in \text{doc} \\ 0 & \text{otherwise} \end{cases}$	1
$x_4$	count(1st and 2nd pronouns $\in$ doc)	3
$x_5$	$\begin{cases} 1 & \text{if “!”} \in \text{doc} \\ 0 & \text{otherwise} \end{cases}$	0
$x_6$	log(word count of doc)	$\ln(66) = 4.19$

# Classifying sentiment for input x

Var	Definition	Val	5.2
$x_1$	count(positive lexicon) $\in$ doc)	3	
$x_2$	count(negative lexicon) $\in$ doc)	2	
$x_3$	$\begin{cases} 1 & \text{if “no”} \in \text{doc} \\ 0 & \text{otherwise} \end{cases}$	1	
$x_4$	count(1st and 2nd pronouns $\in$ doc)	3	
$x_5$	$\begin{cases} 1 & \text{if “!”} \in \text{doc} \\ 0 & \text{otherwise} \end{cases}$	0	
$x_6$	log(word count of doc)	$\ln(66) = 4.19$	

Suppose  $w = [2.5, -5.0, -1.2, 0.5, 2.0, 0.7]$

$$b = 0.1$$

# Classifying sentiment for input $x$

$$\begin{aligned} p(+|x) = P(Y = 1|x) &= \sigma(w \cdot x + b) \\ &= \sigma([2.5, -5.0, -1.2, 0.5, 2.0, 0.7] \cdot [3, 2, 1, 3, 0, 4.19] + 0.1) \\ &= \sigma(.833) \\ &= 0.70 \end{aligned}$$

$$\begin{aligned} p(-|x) = P(Y = 0|x) &= 1 - \sigma(w \cdot x + b) \\ &= 0.30 \end{aligned}$$

We can build features for logistic regression for any classification task: period disambiguation

This ends in a period.

The house at 465 Main St. is new.

End of sentence

Not end

The diagram illustrates the task of period disambiguation. It shows two sentences: "This ends in a period." and "The house at 465 Main St. is new.". The period at the end of the first sentence is circled in red, with a red arrow pointing to the text "End of sentence". The period at the end of the second sentence is also circled in red, with a red arrow pointing to the text "Not end".

$$x_1 = \begin{cases} 1 & \text{if } \text{"Case}(w_i) = \text{Lower"} \\ 0 & \text{otherwise} \end{cases}$$

$$x_2 = \begin{cases} 1 & \text{if } w_i \in \text{AcronymDict} \\ 0 & \text{otherwise} \end{cases}$$

$$x_3 = \begin{cases} 1 & \text{if } w_i = \text{St.} \& \text{Case}(w_{i-1}) = \text{Cap} \\ 0 & \text{otherwise} \end{cases}$$

# Classification in (binary) logistic regression: summary

Given:

- a set of classes: (+ sentiment, - sentiment)
- a vector  $\mathbf{x}$  of features [  $x_1, x_2, \dots, x_n$  ]
  - $x_1 = \text{count}(\text{"awesome"})$
  - $x_2 = \log(\text{number of words in review})$
- A vector  $\mathbf{w}$  of weights [  $w_1, w_2, \dots, w_n$  ]
  - $w_i$  for each feature  $f_i$

$$\begin{aligned} P(y = 1) &= \sigma(w \cdot x + b) \\ &= \frac{1}{1 + e^{-(w \cdot x + b)}} \end{aligned}$$

# Logistic Regression

Logistic Regression: A text example

# Logistic Regression

## Learning in Logistic Regression

# Wait, where did the W's come from?

Supervised classification: we know the correct label  $y$  (either 0 or 1) for each  $x$ .

What the system produces is an estimate,  $\hat{y}$

We want to set  $w$  and  $b$  to minimize the **distance** between our estimate  $\hat{y}^{(i)}$  and the true  $y^{(i)}$ .

- We need a distance estimator: a **loss function** or a **cost function**
- We need an optimization algorithm to update  $w$  and  $b$  to minimize the loss.

# Learning components

A loss function:

- **cross-entropy loss**

An optimization algorithm:

- **stochastic gradient descent**

# Logistic Regression

## Learning in Logistic Regression

# Logistic Regression

## Cross-Entropy Loss

The distance between  $\hat{y}$  and  $y$

We want to know how far is the classifier output:

$$\hat{y} = \sigma(w \cdot x + b)$$

from the true output:

$$y \quad [= \text{either 0 or 1}]$$

We'll call this difference:

$$L(\hat{y}, y) = \text{how much } \hat{y} \text{ differs from the true } y$$

# Intuition of negative log likelihood loss = cross-entropy loss

A case of conditional maximum likelihood estimation

We choose the parameters  $w, b$

that maximize the log probability of the true  $y$  labels in the training data

given the observations  $x$

# Deriving cross-entropy loss for a single observation $x$

**Goal:** maximize probability of the correct label  $p(y|x)$

Since there are only 2 discrete outcomes (0 or 1) we can express the probability  $p(y|x)$  from our classifier (the thing we want to maximize) as

$$p(y|x) = \hat{y}^y (1 - \hat{y})^{1-y}$$

noting:

if  $y=0$ , this simplifies to  $\hat{y}$

if  $y=1$ , this simplifies to  $1 - \hat{y}$

Deriving cross-entropy loss for a single observation  $x$

**Goal:** maximize probability of the correct label  $p(y|x)$

Maximize:  $p(y|x) = \hat{y}^y (1 - \hat{y})^{1-y}$

Now take the log of both sides (mathematically handy)

Maximize:  $\log p(y|x) = \log [\hat{y}^y (1 - \hat{y})^{1-y}]$   
 $= y \log \hat{y} + (1 - y) \log (1 - \hat{y})$

Whatever values maximize  $\log p(y|x)$  will also maximize  $p(y|x)$

Deriving cross-entropy loss for a single observation  $x$

**Goal:** maximize probability of the correct label  $p(y|x)$

**Maximize:**

$$\begin{aligned}\log p(y|x) &= \log [\hat{y}^y (1 - \hat{y})^{1-y}] \\ &= y \log \hat{y} + (1 - y) \log(1 - \hat{y})\end{aligned}$$

Now flip sign to turn this into a loss: something to minimize

**Cross-entropy loss (because is formula for cross-entropy( $y, \hat{y}$ ))**

**Minimize:**

$$L_{\text{CE}}(\hat{y}, y) = -\log p(y|x) = -[y \log \hat{y} + (1 - y) \log(1 - \hat{y})]$$

Or, plugging in definition of  $\hat{y}$ :

$$L_{\text{CE}}(\hat{y}, y) = -[y \log \sigma(w \cdot x + b) + (1 - y) \log(1 - \sigma(w \cdot x + b))]$$

# Let's see if this works for our sentiment example

We want loss to be:

- smaller if the model estimate is close to correct
- bigger if model is confused

Let's first suppose the true label of this is  $y=1$  (positive)

It's hokey . There are virtually no surprises , and the writing is second-rate . So why was it so enjoyable ? For one thing , the cast is great . Another nice touch is the music . I was overcome with the urge to get off the couch and start dancing . It sucked me in , and it'll do the same to you .

Let's see if this works for our sentiment example

True value is  $y=1$ . How well is our model doing?

$$\begin{aligned} p(+|x) = P(Y = 1|x) &= \sigma(w \cdot x + b) \\ &= \sigma([2.5, -5.0, -1.2, 0.5, 2.0, 0.7] \cdot [3, 2, 1, 3, 0, 4.19] + 0.1) \\ &= \sigma(.833) \\ &= 0.70 \end{aligned} \tag{5.6}$$

Pretty well! What's the loss?

$$\begin{aligned} L_{\text{CE}}(\hat{y}, y) &= -[y \log \sigma(w \cdot x + b) + (1 - y) \log(1 - \sigma(w \cdot x + b))] \\ &= -[\log \sigma(w \cdot x + b)] \\ &= -\log(.70) \\ &= .36 \end{aligned}$$

Let's see if this works for our sentiment example

Suppose true value instead was  $y=0$ .

$$\begin{aligned} p(-|x) = P(Y = 0|x) &= 1 - \sigma(w \cdot x + b) \\ &= 0.30 \end{aligned}$$

What's the loss?

$$\begin{aligned} L_{\text{CE}}(\hat{y}, y) &= -[y \log \sigma(w \cdot x + b) + (1 - y) \log (1 - \sigma(w \cdot x + b))] \\ &= -[\log (1 - \sigma(w \cdot x + b))] \\ &= -\log (.30) \\ &= 1.2 \end{aligned}$$

Let's see if this works for our sentiment example

The loss when model was right (if true  $y=1$ )

$$\begin{aligned} L_{\text{CE}}(\hat{y}, y) &= -[y \log \sigma(w \cdot x + b) + (1 - y) \log(1 - \sigma(w \cdot x + b))] \\ &= -[\log \sigma(w \cdot x + b)] \\ &= -\log(.70) \\ &= .36 \end{aligned}$$

Is lower than the loss when model was wrong (if true  $y=0$ ):

$$\begin{aligned} L_{\text{CE}}(\hat{y}, y) &= -[y \log \sigma(w \cdot x + b) + (1 - y) \log(1 - \sigma(w \cdot x + b))] \\ &= -[\log(1 - \sigma(w \cdot x + b))] \\ &= -\log(.30) \\ &= 1.2 \end{aligned}$$

Sure enough, loss was bigger when model was wrong!

# Logistic Regression

## Cross-Entropy Loss

# Logistic Regression

Stochastic Gradient Descent

# Our goal: minimize the loss

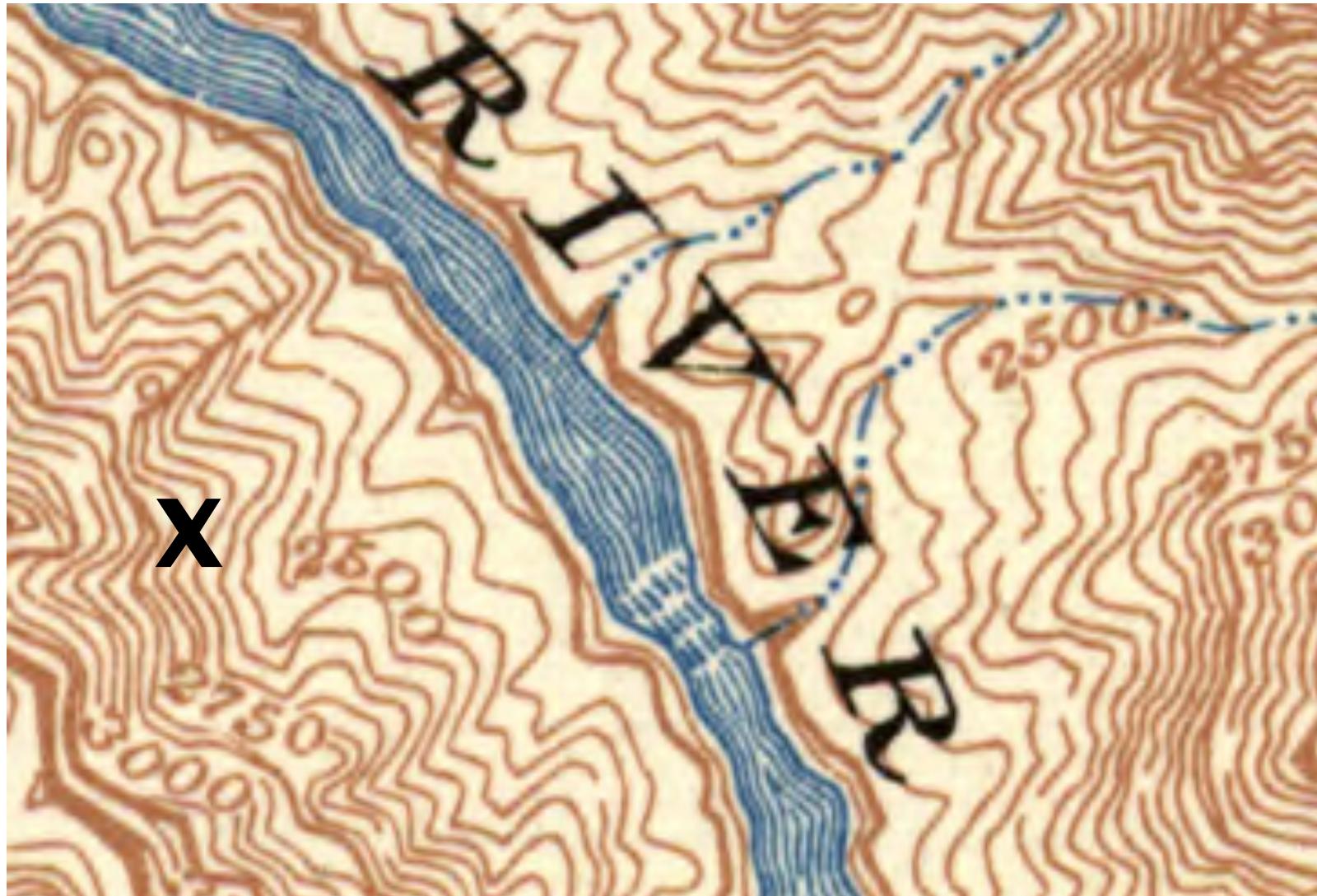
Let's make explicit that the loss function is parameterized by weights  $\theta=(w,b)$

We want the weights that minimize the loss, averaged over all examples:

$$\hat{\theta} = \operatorname{argmin}_{\theta} \frac{1}{m} \sum_{i=1}^m L_{\text{CE}}(f(x^{(i)}; \theta), y^{(i)})$$

# Intuition of gradient descent

How do I get to the bottom of this river canyon?



Look around me 360°  
Find the direction of  
steepest slope  
Go that way

# Our goal: minimize the loss

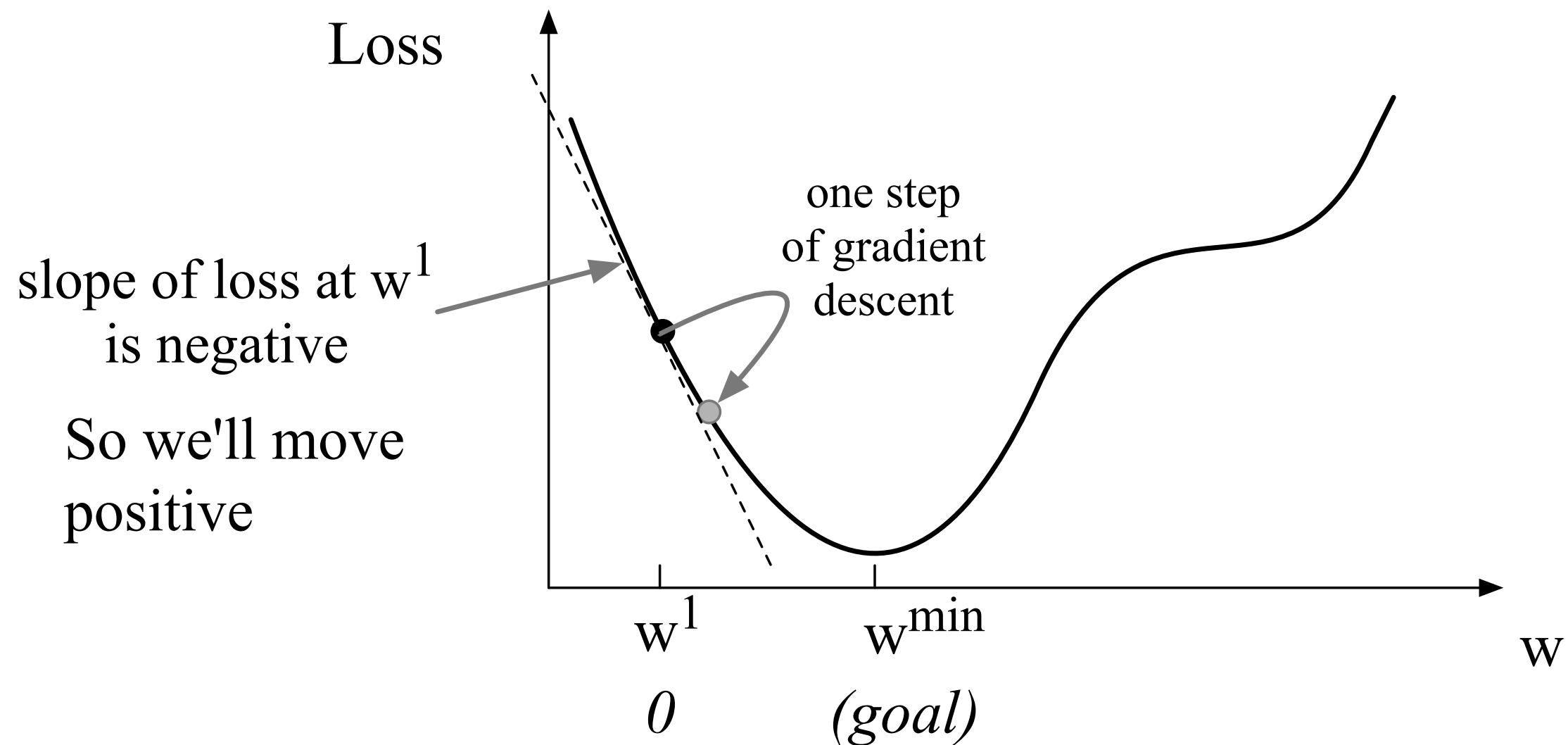
For logistic regression, loss function is **convex**

- A convex function has just one minimum
- Gradient descent starting from any point is guaranteed to find the minimum
  - (Loss for neural networks is non-convex)

# Let's first visualize for a single scalar $w$

Q: Given current  $w$ , should we make it bigger or smaller?

A: Move  $w$  in the reverse direction from the slope of the function



# Gradients

The **gradient** of a function of many variables is a vector pointing in the direction of the greatest increase in a function.

**Gradient Descent:** Find the gradient of the loss function at the current point and move in the **opposite** direction.

How much do we move in that direction ?

- The value of the gradient (slope in our example)  $\frac{d}{dw} f(x; w)$  weighted by a **learning rate**  $\eta$
- Higher learning rate means move  $w$  faster

$$w^{t+1} = w^t - \eta \frac{d}{dw} f(x; w)$$

# Now let's consider $N$ dimensions

We want to know where in the  $N$ -dimensional space (of the  $N$  parameters that make up  $\theta$ ) we should move.

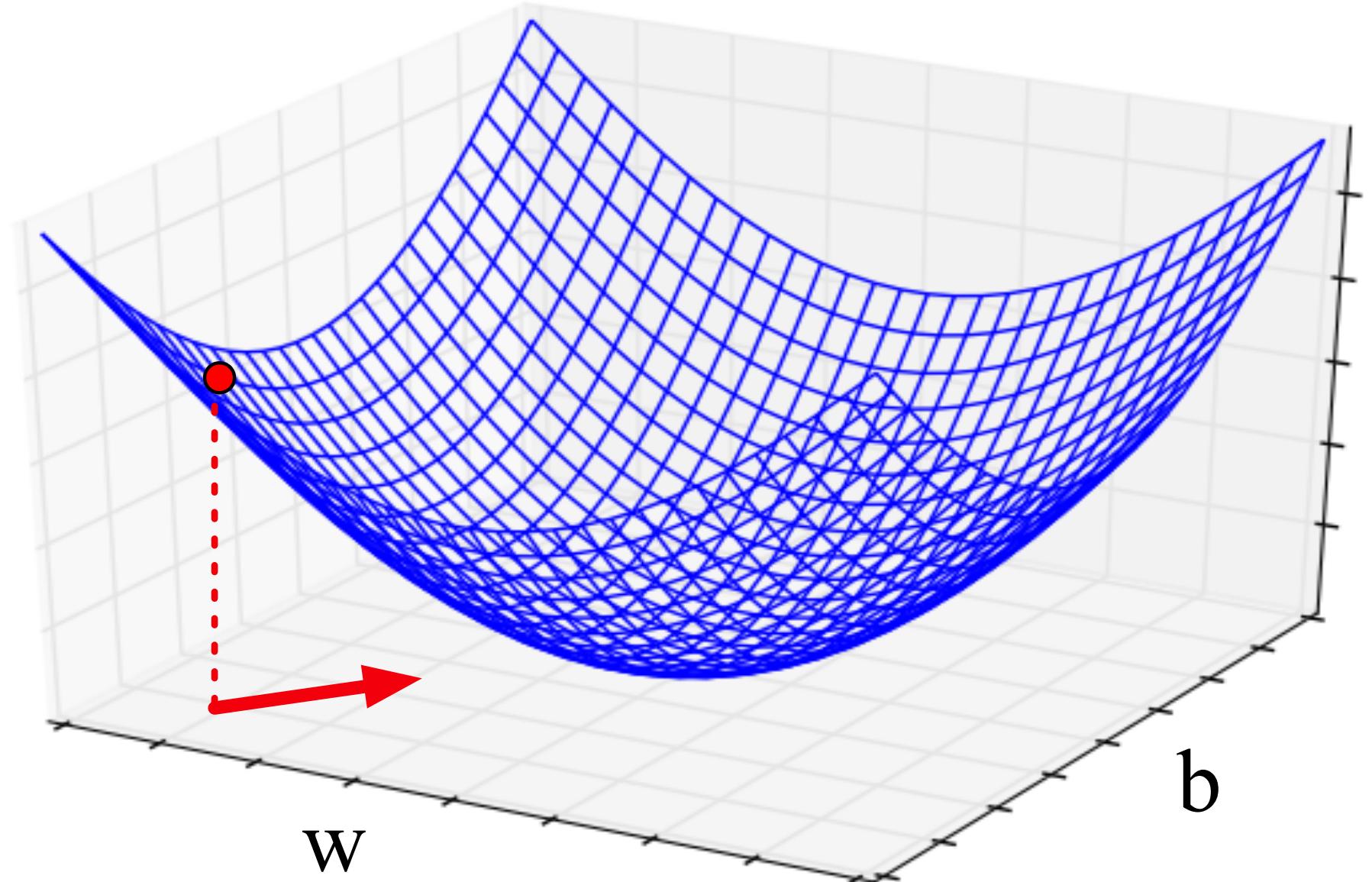
The gradient is just such a vector; it expresses the directional components of the sharpest slope along each of the  $N$  dimensions.

# Imagine 2 dimensions, w and b

Visualizing the gradient vector at the red point

It has two dimensions shown in the x-y plane

$\text{Cost}(w,b)$



# Real gradients

Are much longer; lots and lots of weights

For each dimension  $w_i$ , the gradient component  $i$  tells us the slope with respect to that variable.

- “How much would a small change in  $w_i$  influence the total loss function  $L$ ? ”
- We express the slope as a partial derivative  $\partial$  of the loss  $\partial w_i$

The gradient is then defined as a vector of these partials.

# The gradient

We'll represent  $\hat{y}$  as  $f(x; \theta)$  to make the dependence on  $\theta$  more obvious:

$$\nabla_{\theta} L(f(x; \theta), y) = \begin{bmatrix} \frac{\partial}{\partial w_1} L(f(x; \theta), y) \\ \frac{\partial}{\partial w_2} L(f(x; \theta), y) \\ \vdots \\ \frac{\partial}{\partial w_n} L(f(x; \theta), y) \end{bmatrix}$$

The final equation for updating  $\theta$  based on the gradient is thus

$$\theta_{t+1} = \theta_t - \eta \nabla L(f(x; \theta), y)$$

What are these partial derivatives for logistic regression?

The loss function

$$L_{\text{CE}}(\hat{y}, y) = -[y \log \sigma(w \cdot x + b) + (1 - y) \log (1 - \sigma(w \cdot x + b))]$$

The elegant derivative of this function (see textbook 5.8 for derivation)

$$\frac{\partial L_{\text{CE}}(\hat{y}, y)}{\partial w_j} = [\sigma(w \cdot x + b) - y]x_j$$

```

function STOCHASTIC GRADIENT DESCENT( $L()$ ,  $f()$ ,  $x$ ,  $y$ ) returns  $\theta$ 
  # where: L is the loss function
  #       f is a function parameterized by  $\theta$ 
  #       x is the set of training inputs  $x^{(1)}$ ,  $x^{(2)}$ , ...,  $x^{(m)}$ 
  #       y is the set of training outputs (labels)  $y^{(1)}$ ,  $y^{(2)}$ , ...,  $y^{(m)}$ 

   $\theta \leftarrow 0$ 
  repeat til done    # see caption
    For each training tuple  $(x^{(i)}, y^{(i)})$  (in random order)
      1. Optional (for reporting):          # How are we doing on this tuple?
        Compute  $\hat{y}^{(i)} = f(x^{(i)}; \theta)$     # What is our estimated output  $\hat{y}$ ?
        Compute the loss  $L(\hat{y}^{(i)}, y^{(i)})$     # How far off is  $\hat{y}^{(i)}$  from the true output  $y^{(i)}$ ?
      2.  $g \leftarrow \nabla_{\theta} L(f(x^{(i)}; \theta), y^{(i)})$     # How should we move  $\theta$  to maximize loss?
      3.  $\theta \leftarrow \theta - \eta g$                       # Go the other way instead

  return  $\theta$ 

```

# Hyperparameters

The learning rate  $\eta$  is a **hyperparameter**

- too high: the learner will take big steps and overshoot
- too low: the learner will take too long

More on hyperparameters in Chapter 7

- Briefly, a special kind of parameter for an ML model
- Instead of being learned by algorithm from supervision (like regular parameters), they are chosen by algorithm designer.

# Logistic Regression

Stochastic Gradient Descent

# Logistic Regression

Stochastic Gradient Descent:  
An example and more details

# Working through an example

One step of gradient descent

A mini-sentiment example, where the true  $y=1$  (positive)

Two features:

$x_1 = 3$  (count of positive lexicon words)

$x_2 = 2$  (count of negative lexicon words)

Assume 3 parameters (2 weights and 1 bias) in  $\Theta^0$  are zero:

$$w_1 = w_2 = b = 0$$

$$\eta = 0.1$$

# Example of gradient descent

$$w_1 = w_2 = b = 0; \\ x_1 = 3; \quad x_2 = 2$$

Update step for update  $\theta$  is:

$$\theta_{t+1} = \theta_t - \eta \nabla L(f(x; \theta), y)$$

where

$$\frac{\partial L_{\text{CE}}(\hat{y}, y)}{\partial w_j} = [\sigma(w \cdot x + b) - y]x_j$$

Gradient vector has 3 dimensions:

$$\nabla_{w,b} = \begin{bmatrix} \frac{\partial L_{\text{CE}}(\hat{y}, y)}{\partial w_1} \\ \frac{\partial L_{\text{CE}}(\hat{y}, y)}{\partial w_2} \\ \frac{\partial L_{\text{CE}}(\hat{y}, y)}{\partial b} \end{bmatrix} = \begin{bmatrix} (\sigma(w \cdot x + b) - y)x_1 \\ (\sigma(w \cdot x + b) - y)x_2 \\ \sigma(w \cdot x + b) - y \end{bmatrix} = \begin{bmatrix} (\sigma(0) - 1)x_1 \\ (\sigma(0) - 1)x_2 \\ \sigma(0) - 1 \end{bmatrix} = \begin{bmatrix} -0.5x_1 \\ -0.5x_2 \\ -0.5 \end{bmatrix} = \begin{bmatrix} -1.5 \\ -1.0 \\ -0.5 \end{bmatrix}$$

# Example of gradient descent

$$\nabla_{w,b} = \begin{bmatrix} \frac{\partial L_{\text{CE}}(\hat{y}, y)}{\partial w_1} \\ \frac{\partial L_{\text{CE}}(\hat{y}, y)}{\partial w_2} \\ \frac{\partial L_{\text{CE}}(\hat{y}, y)}{\partial b} \end{bmatrix} = \begin{bmatrix} (\sigma(w \cdot x + b) - y)x_1 \\ (\sigma(w \cdot x + b) - y)x_2 \\ \sigma(w \cdot x + b) - y \end{bmatrix} = \begin{bmatrix} (\sigma(0) - 1)x_1 \\ (\sigma(0) - 1)x_2 \\ \sigma(0) - 1 \end{bmatrix} = \begin{bmatrix} -0.5x_1 \\ -0.5x_2 \\ -0.5 \end{bmatrix} = \begin{bmatrix} -1.5 \\ -1.0 \\ -0.5 \end{bmatrix}$$

Now that we have a gradient, we compute the new parameter vector  $\theta^1$  by moving  $\theta^0$  in the opposite direction from the gradient:

$$\theta_{t+1} = \theta_t - \eta \nabla L(f(x; \theta), y) \quad \eta = 0.1;$$

$$\theta^1 = \begin{bmatrix} w_1 \\ w_2 \\ b \end{bmatrix} - \eta \begin{bmatrix} -1.5 \\ -1.0 \\ -0.5 \end{bmatrix} = \begin{bmatrix} .15 \\ .1 \\ .05 \end{bmatrix}$$

Note that enough negative examples would eventually make  $w_2$  negative

# Mini-batch training

Stochastic gradient descent chooses a single random example at a time.

That can result in choppy movements

More common to compute gradient over batches of training instances.

**Batch training:** entire dataset

**Mini-batch training:**  $m$  examples (512, or 1024)

# Logistic Regression

Stochastic Gradient Descent:  
An example and more details

# Logistic Regression

## Regularization

# Overfitting

A model that perfectly match the training data has a problem.

It will also **overfit** to the data, modeling noise

- A random word that perfectly predicts  $y$  (it happens to only occur in one class) will get a very high weight.
- Failing to generalize to a test set without this word.

A good model should be able to **generalize**

# Overfitting

+

This movie drew me in, and it'll  
do the same to you.

-

I can't tell you how much I  
hated this movie. It sucked.

Useful or harmless features

X1 = "this"

X2 = "movie"

X3 = "hated"

X4 = "drew me in"

4gram features that just  
"memorize" training set and  
might cause problems

X5 = "the same to you"

X7 = "tell you how much"

# Overfitting

4-gram model on tiny data will just memorize the data

- 100% accuracy on the training set

But it will be surprised by the novel 4-grams in the test data

- Low accuracy on test set

Models that are too powerful can **overfit** the data

- Fitting the details of the training data so exactly that the model doesn't generalize well to the test set
- How to avoid overfitting?
  - Regularization in logistic regression
  - Dropout in neural networks

# Regularization

A solution for overfitting

Add a regularization term  $R(\theta)$  to the loss function  
(for now written as maximizing logprob rather than minimizing loss)

$$\hat{\theta} = \operatorname{argmax}_{\theta} \sum_{i=1}^m \log P(y^{(i)} | x^{(i)}) - \alpha R(\theta)$$

Idea: choose an  $R(\theta)$  that penalizes large weights

- fitting the data well with lots of big weights not as good as fitting the data a little less well, with small weights

# L2 Regularization (= ridge regression)

The sum of the squares of the weights

The name is because this is the (square of the)  
**L2 norm**  $\|\theta\|_2$ , = Euclidean distance of  $\theta$  to the origin.

$$R(\theta) = \|\theta\|_2^2 = \sum_{j=1}^n \theta_j^2$$

L2 regularized objective function:

$$\hat{\theta} = \operatorname{argmax}_{\theta} \left[ \sum_{i=1}^m \log P(y^{(i)} | x^{(i)}) \right] - \alpha \sum_{j=1}^n \theta_j^2$$

# L1 Regularization (= lasso regression)

The sum of the (absolute value of the) weights

Named after the **L1 norm**  $\|W\|_1$ , = sum of the absolute values of the weights, = **Manhattan distance**

$$R(\theta) = \|\theta\|_1 = \sum_{i=1}^n |\theta_i|$$

L1 regularized objective function:

$$\hat{\theta} = \operatorname{argmax}_{\theta} \left[ \sum_{i=1}^m \log P(y^{(i)} | x^{(i)}) \right] - \alpha \sum_{j=1}^n |\theta_j|$$

# Logistic Regression

## Regularization

# Logistic Regression

# Multinomial Logistic Regression

# Multinomial Logistic Regression

Often we need more than 2 classes

- Positive/negative/neutral
- Parts of speech (noun, verb, adjective, adverb, preposition, etc.)
- Classify emergency SMSs into different actionable classes

If >2 classes we use **multinomial logistic regression**

- = Softmax regression
- = Multinomial logit
- = (defunct names : Maximum entropy modeling or MaxEnt

So "logistic regression" will just mean binary (2 output classes)

# Multinomial Logistic Regression

The probability of everything must still sum to 1

$$P(\text{positive} \mid \text{doc}) + P(\text{negative} \mid \text{doc}) + P(\text{neutral} \mid \text{doc}) = 1$$

Need a generalization of the sigmoid called the **softmax**

- Takes a vector  $z = [z_1, z_2, \dots, z_k]$  of  $k$  arbitrary values
- Outputs a probability distribution
  - each value in the range  $[0,1]$
  - all the values summing to 1

# The softmax function

Turns a vector  $z = [z_1, z_2, \dots, z_k]$  of  $k$  arbitrary values into probabilities

$$\text{softmax}(z_i) = \frac{\exp(z_i)}{\sum_{j=1}^k \exp(z_j)} \quad 1 \leq i \leq k$$

The denominator  $\sum_{i=1}^k e^{z_i}$  is used to normalize all the values into probabilities.

$$\text{softmax}(z) = \left[ \frac{\exp(z_1)}{\sum_{i=1}^k \exp(z_i)}, \frac{\exp(z_2)}{\sum_{i=1}^k \exp(z_i)}, \dots, \frac{\exp(z_k)}{\sum_{i=1}^k \exp(z_i)} \right]$$

# The softmax function

- Turns a vector  $z = [z_1, z_2, \dots, z_k]$  of  $k$  arbitrary values into probabilities

$$z = [0.6, 1.1, -1.5, 1.2, 3.2, -1.1]$$

$$\text{softmax}(z) = \left[ \frac{\exp(z_1)}{\sum_{i=1}^k \exp(z_i)}, \frac{\exp(z_2)}{\sum_{i=1}^k \exp(z_i)}, \dots, \frac{\exp(z_k)}{\sum_{i=1}^k \exp(z_i)} \right]$$

$$[0.055, 0.090, 0.0067, 0.10, 0.74, 0.010]$$

# Softmax in multinomial logistic regression

$$p(y = c|x) = \frac{\exp(w_c \cdot x + b_c)}{\sum_{j=1}^k \exp(w_j \cdot x + b_j)}$$

Input is still the dot product between weight vector  $w$  and input vector  $x$

But now we'll need separate weight vectors for each of the  $K$  classes.

# Features in binary versus multinomial logistic regression

Binary: positive weight  $\rightarrow y=1$  neg weight  $\rightarrow y=0$

$$x_5 = \begin{cases} 1 & \text{if “!”} \in \text{doc} \\ 0 & \text{otherwise} \end{cases} \quad w_5 = 3.0$$

Multinomial: separate weights for each class:

Feature	Definition	$w_{5,+}$	$w_{5,-}$	$w_{5,0}$
$f_5(x)$	$\begin{cases} 1 & \text{if “!”} \in \text{doc} \\ 0 & \text{otherwise} \end{cases}$	3.5	3.1	-5.3

# Logistic Regression

# Multinomial Logistic Regression