

Logistic
Regression

The Task of Text Classification

Is this spam?

Dear Sir,

Please find attached the forms you requested.

Special Note: Student needs to provide the hand signed forms on the very first day of registration- (Forms attached)

IMPORTANT NOTE:

We will be needing additional bank statements and proof for justification if students have completed education outside their home state. (Legitimate documents- electricity bill, Rental Agreement, Bank Statement, College/University ID)

Please feel free to reach us in case any queries arise.

Study Abroad Partners

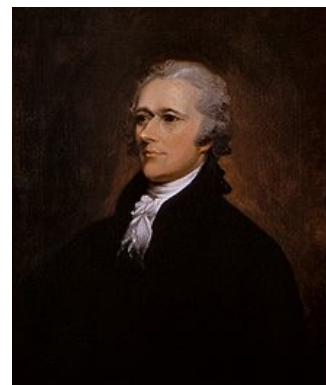
Who wrote which *Federalist Papers*?

1787-8: essays anonymously written by:

Alexander Hamilton, James Madison, and John Jay

to convince New York to ratify U.S Constitution

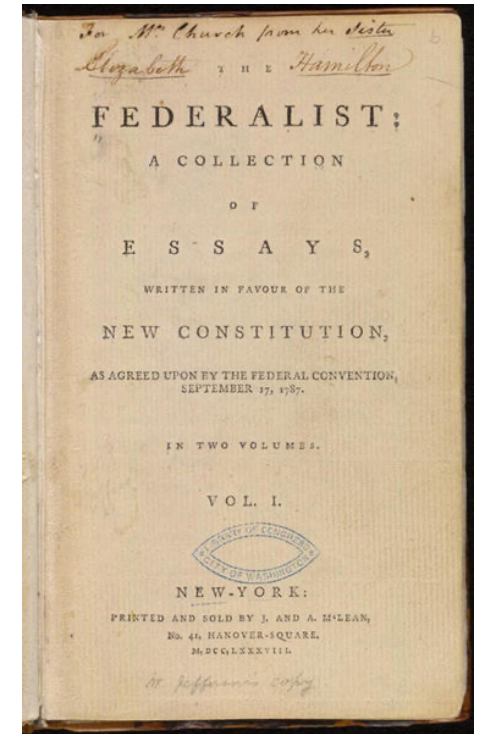
Authorship of 12 of the letters unclear between:



Alexander Hamilton



James Madison



1963: solved by Mosteller and Wallace using Bayesian methods

Positive or negative movie review?



unbelievably disappointing



Full of zany characters and richly applied satire, and some great plot twists



this is the greatest screwball comedy ever filmed



It was pathetic. The worst part about it was the boxing scenes.

Even language modeling can be viewed as classification!

- Let the set of classes be the words (vocabulary V)
- Predicting the next word is classifying
 - the context-so-far
 - into a class for each possible next word

Text Classification

Spam detection

Authorship identification (who wrote this?)

Language Identification (is this Portuguese?)

Sentiment analysis

Language modeling (what next word does this context expect)

Many kinds of classifiers!

- Logistic regression (this lecture)
- Naive Bayes
- Neural networks
- k -nearest neighbors
- LLMs
 - Fine-tuned as classifiers
 - Prompted to give a classification

Probabilistic Classifiers

A subset of classifiers that return a **probability** for each possible output class.

Probabilistic Text Classification: definition

Input:

- a "document" (any text) d
- a fixed set of output classes $Y = \{y_1, y_2, \dots, y_J\}$

Output: the probability (for each possible class y_i) that the true class y is y_i :

$$P(y=y_i/d) \quad \forall y_i \in Y$$

These probabilities sum to one: $\sum_i P(y=y_i/d)=1$

So for binary classification $Y = \{0,1\}$ and

$$P(y=1/d) = 1 - P(y=0/d)$$

Probabilistic Text Classification is usually a kind of Supervised Machine Learning

Training

Input:

- a fixed set of classes $Y = \{y_1, y_2, \dots, y_J\}$
- a training set of m hand-labeled inputs $(x^{(1)}, y^{(1)}), \dots, (x^{(m)}, y^{(m)})$

Learning algorithm

Output:

- a learned classifier that maps from $d \rightarrow P(y=y_i|d) \quad \forall y_i \in Y$

Inference or Test

Input: a document d

Output: a distribution over the output classes $P(y=y_i|d) \quad \forall y_i \in Y$

So for binary classification that means $P(y=1|d)$ and $P(y=0|d)$

Logistic
Regression

The Task of Text Classification

Logistic Regression Classification

Logistic
Regression

Logistic Regression

Important analytic tool in natural and social sciences

Baseline supervised machine learning tool for classification

Is also the foundation of neural networks

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 j for input $x^{(i)}$ is x_j , more completely $x_j^{(i)}$, or sometimes $f_j(x)$.
2. A **classification function** that computes the estimated probability $p(y|x)$, 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 (next lecture):

- We learn weights w and b using **stochastic gradient descent** and **cross-entropy loss**.

Test (also called inference) (this lecture):

- Given a test example x we compute $p(y/x)$ using learned weights w and b
- Then return whichever label ($y = 1$ or $y = 0$) has higher probability

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 probability $\hat{y}^{(i)} = P(y^{(i)} = 1)$

Features in logistic regression

- For feature x_i , weight w_i tells is how important is x_i
 - x_i = "review contains 'awesome'": $w_i = +10$
 - x_j = "review contains 'abysmal'": $w_j = -10$
 - x_k = "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 probability $P(y=1 | x, \theta)$
and then $P(y=0 | x, \theta) = 1 - P(y=1 | x, \theta)$

We sometimes introduce the term $\hat{y} = P(y=1 | x, \theta)$

So $\hat{y}$ (pronounced y-hat) means

"our estimated probability that x is a positive example"

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 and can tell us

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

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

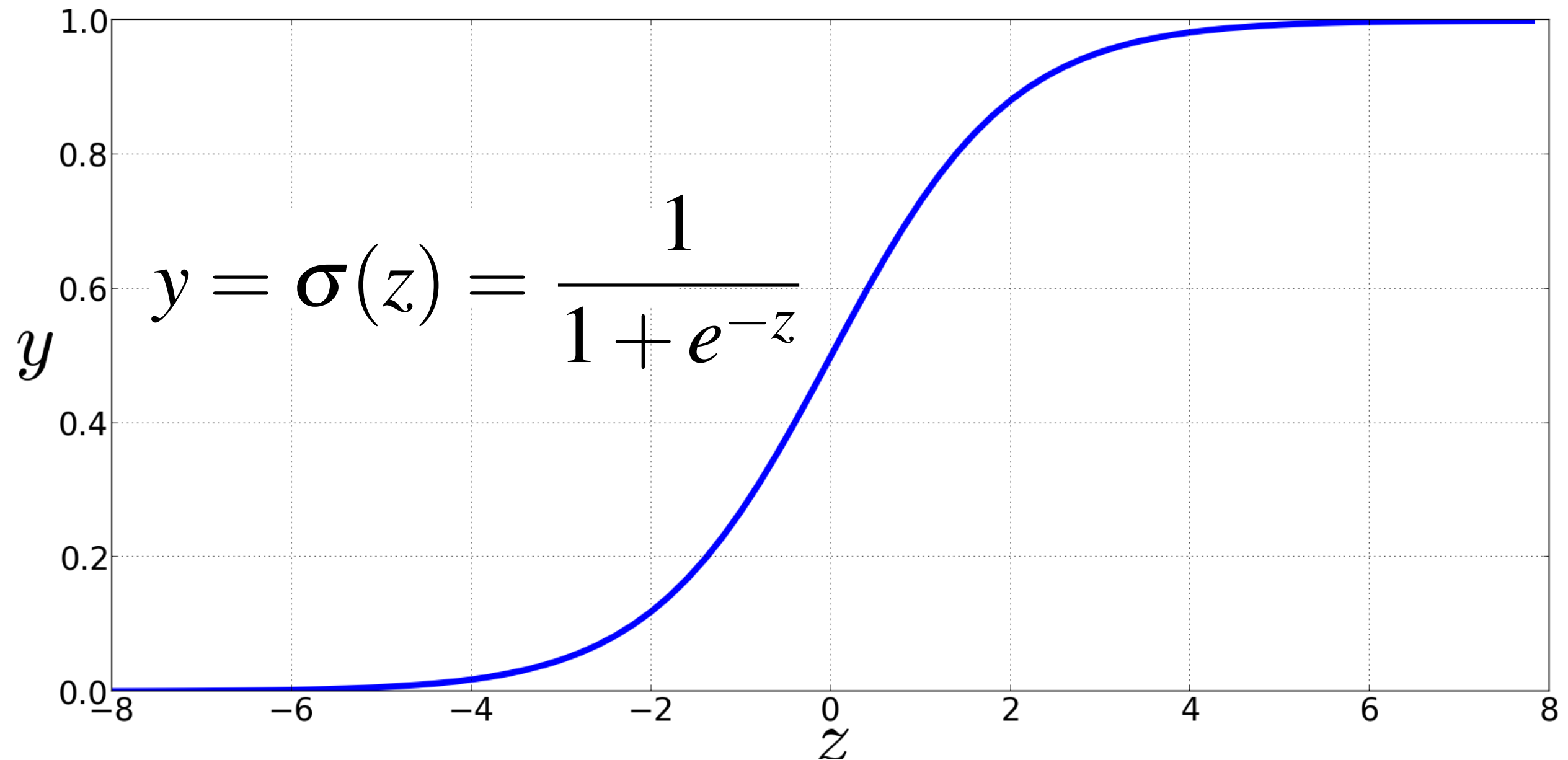
The problem: z isn't a probability, it's just a number!

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

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:

$$P(y = 0) = 1 - \sigma(w \cdot x + b) = \sigma(-(w \cdot x + b))$$

$$= 1 - \frac{1}{1 + \exp(-(w \cdot x + b))}$$

$$= \frac{\exp(-(w \cdot x + b))}{1 + \exp(-(w \cdot x + b))}$$

Because

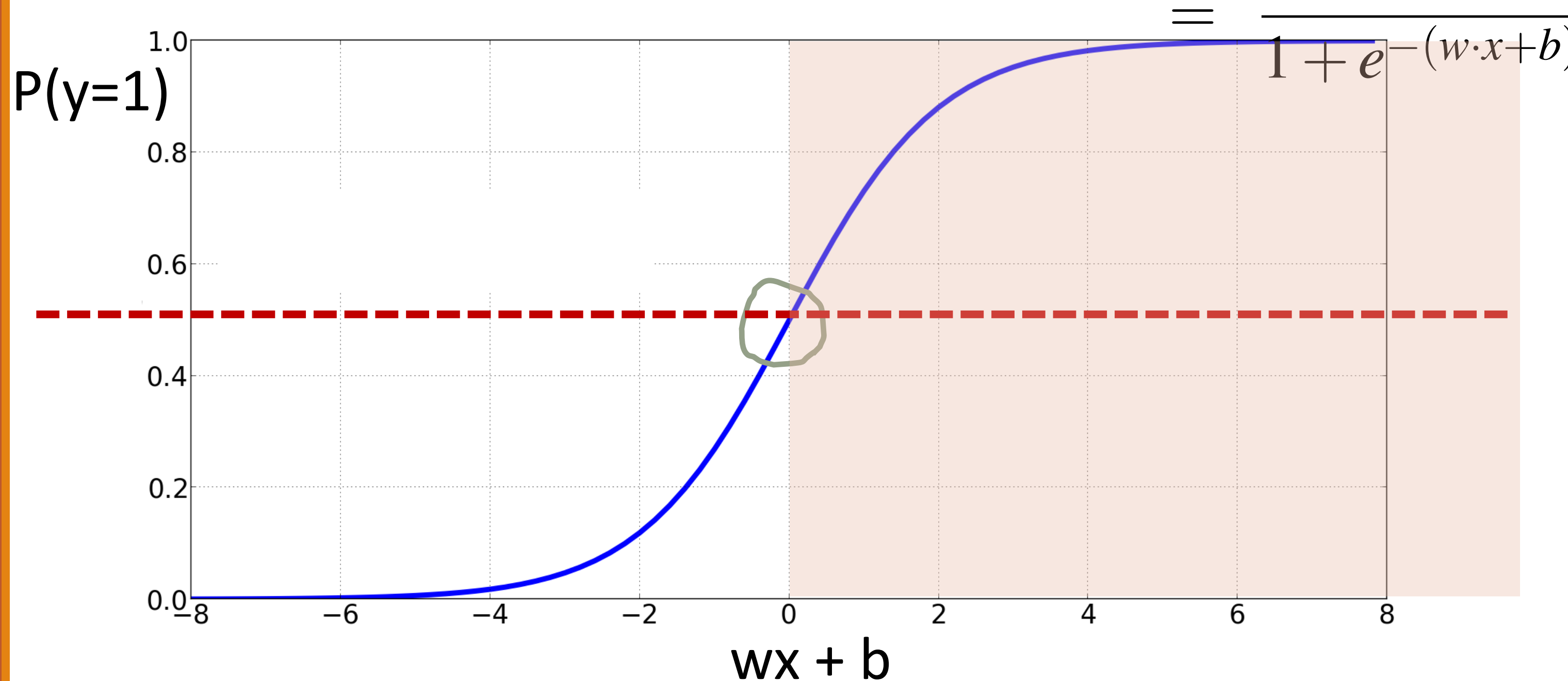
$$1 - \sigma(x) = \sigma(-x)$$

Turning a probability into a classifier

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

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

$$\text{decision}(x) = \begin{cases} 1 & \text{if } P(y = 1|x) > 0.5 \\ 0 & \text{otherwise} \end{cases} \quad \begin{array}{l} \text{if } w \cdot x + b > 0 \\ \text{if } w \cdot x + b \leq 0 \end{array}$$

Classification in Logistic Regression

Logistic
Regression

Logistic Regression

Logistic Regression: a text example
on sentiment classification

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**.

$x_2=2$
 $x_3=1$
 $x_1=3$
 $x_5=0$
 $x_6=4.19$
 $x_4=3$

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

End of sentence

This ends in a period.
The house at 465 Main St. is new.

Not end

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

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

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

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
on sentiment classification

Learning: Cross-Entropy Loss

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 .
- But 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**

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:

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

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

if $y=0$, 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:
$$\begin{aligned} \log p(y|x) &= \log [\hat{y}^y (1 - \hat{y})^{1-y}] \\ &= y \log \hat{y} + (1 - y) \log(1 - \hat{y}) \end{aligned}$$

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!

Cross-Entropy Loss

Logistic
Regression

Stochastic Gradient Descent

Logistic
Regression

Our goal: minimize the loss

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

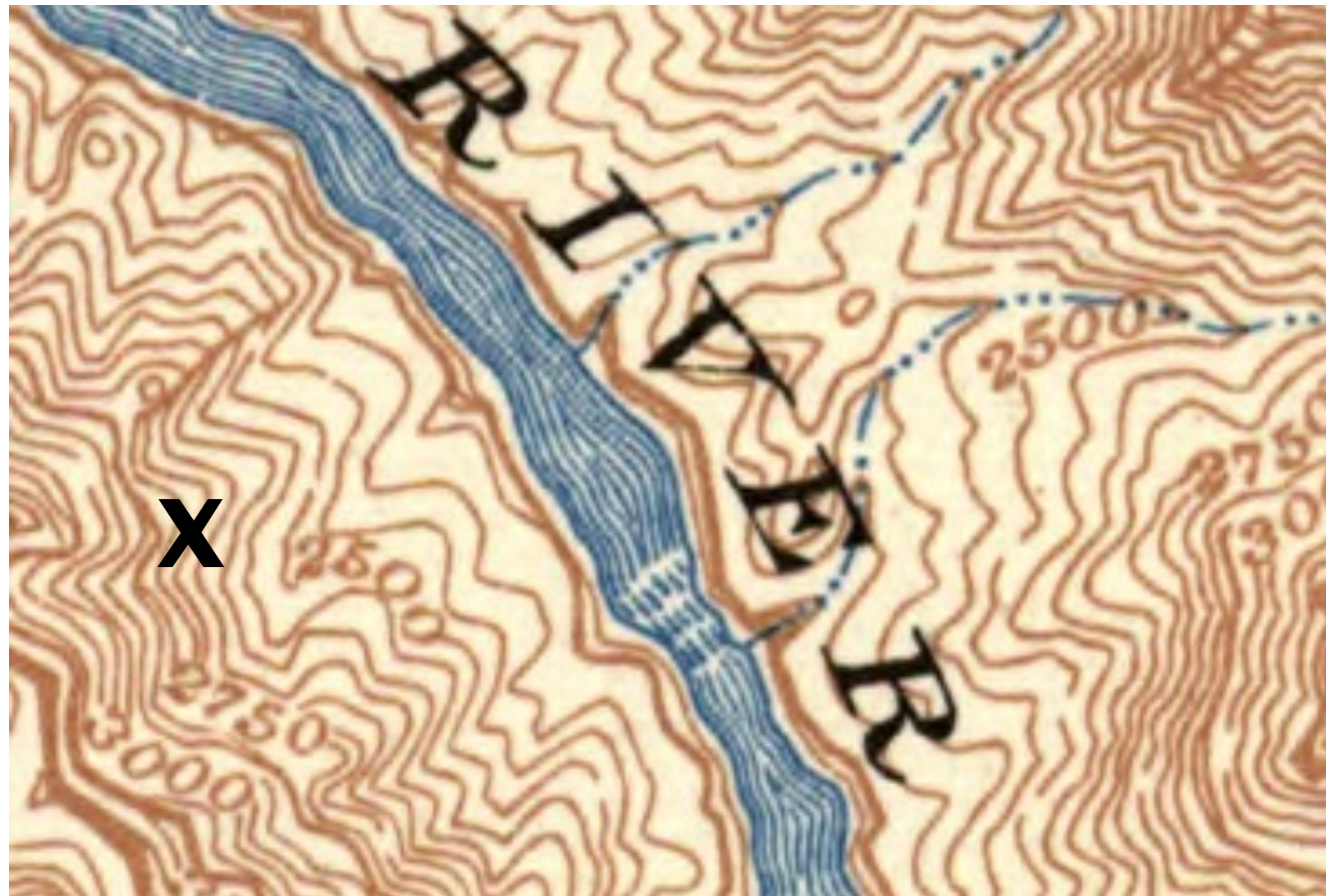
- And we'll represent $\hat{y}$ as $f(x; \theta)$ to make the dependence on θ more obvious

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 down

Go that way

Our goal: minimize the loss

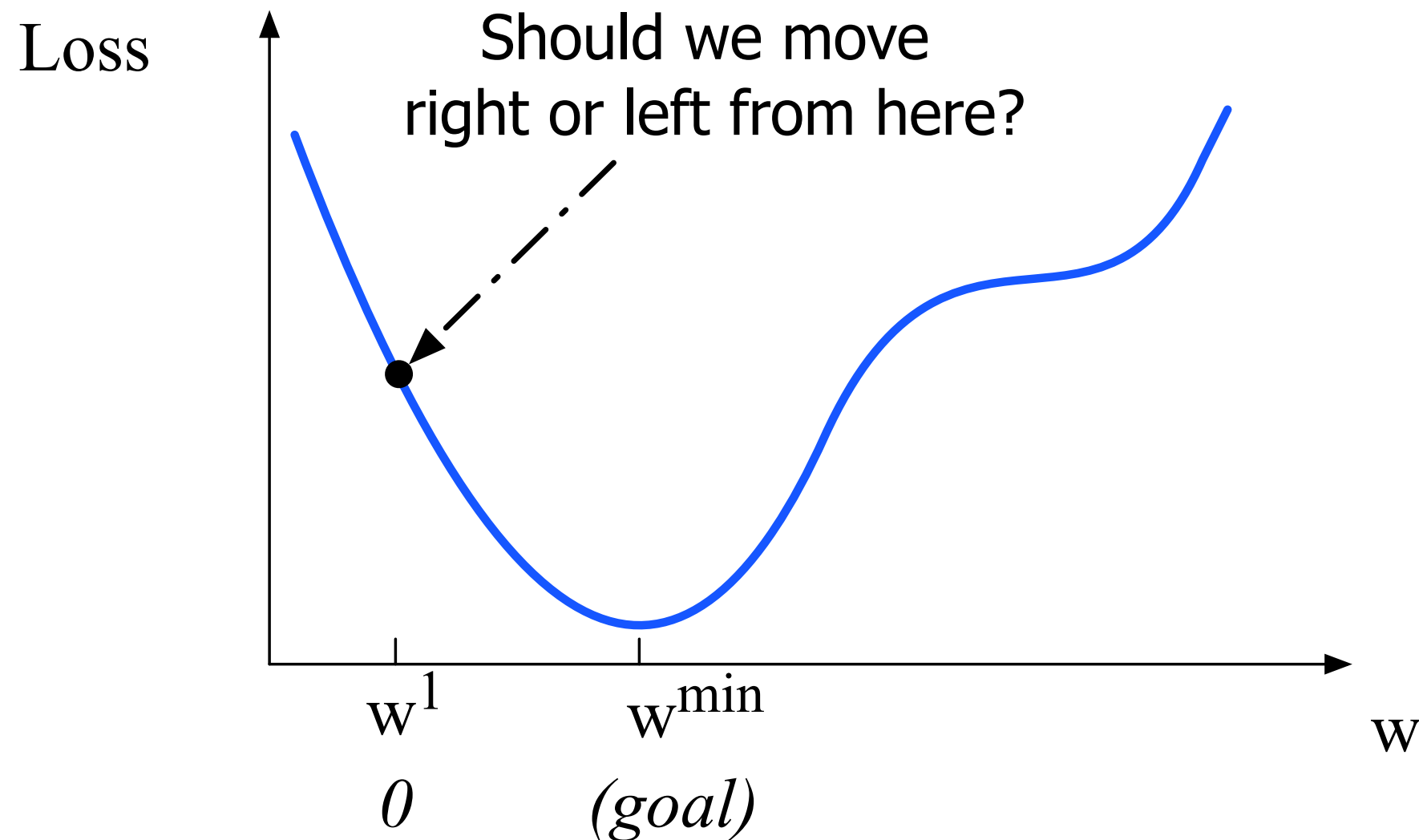
For logistic regression, loss function is **convex**

- 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?

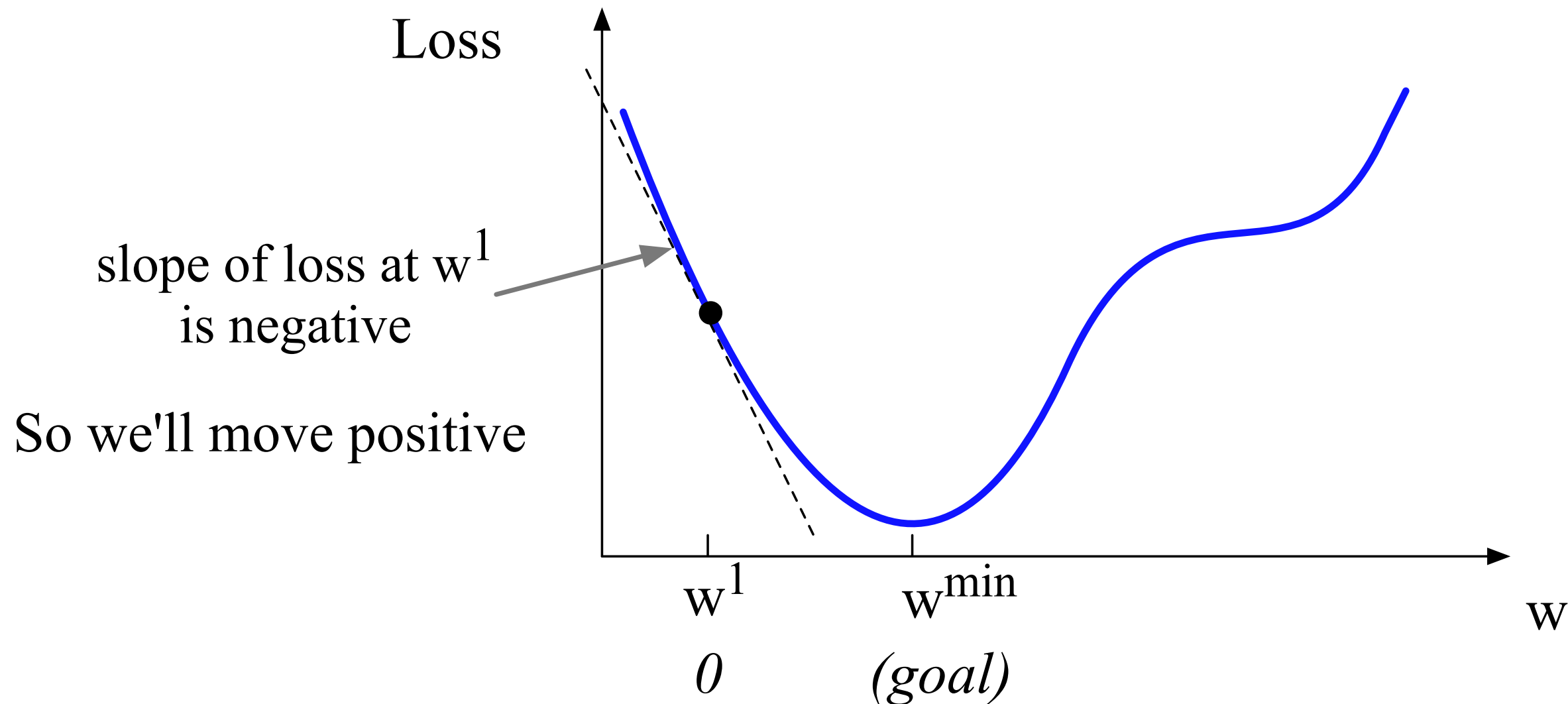
A: Move w in the reverse direction from the slope of the function



Let's first visualize for a single scalar w

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

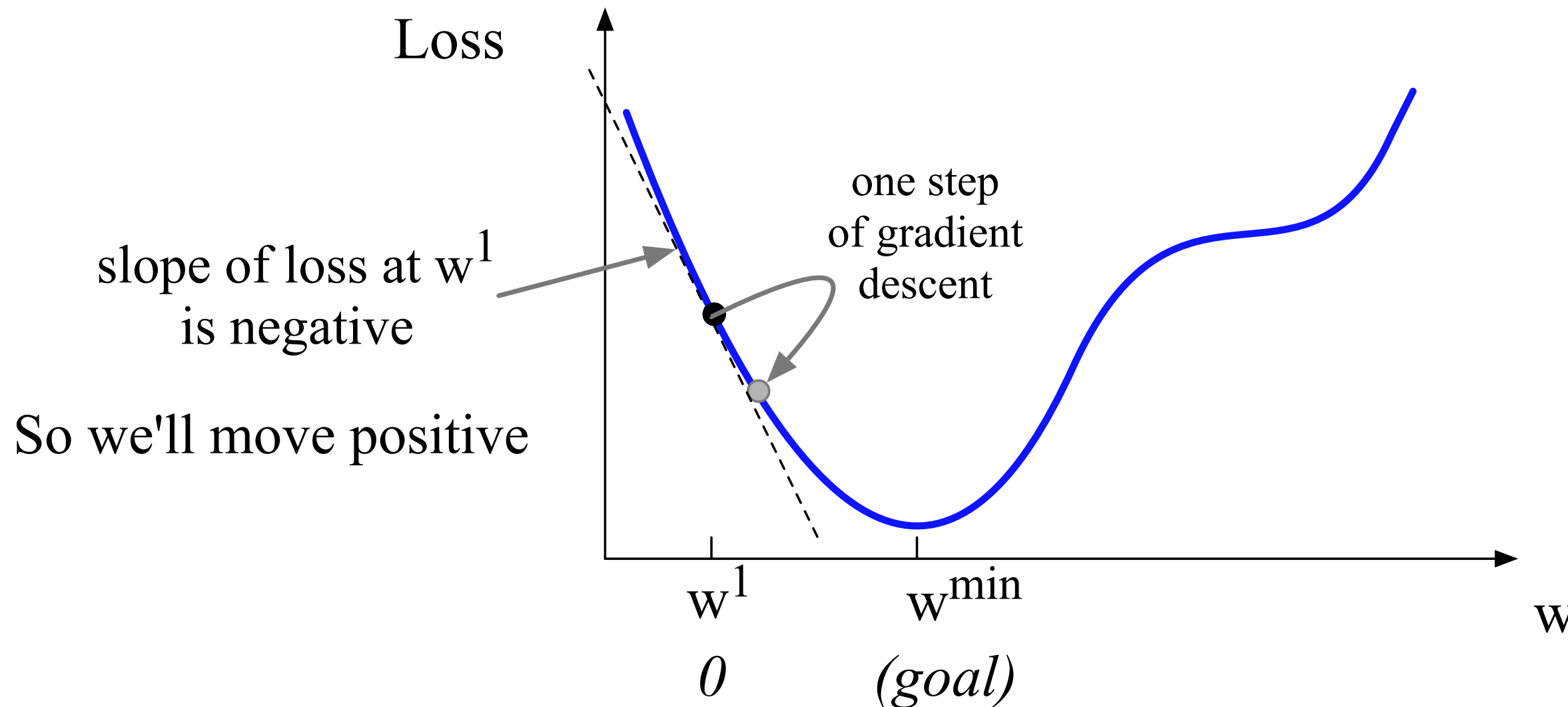
A: Move w in the reverse direction from the slope of the function



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) $\frac{d}{dw} L(f(x; w), y)$ weighted by a **learning rate** η
- Higher learning rate means move w faster

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

Now let's consider N dimensions

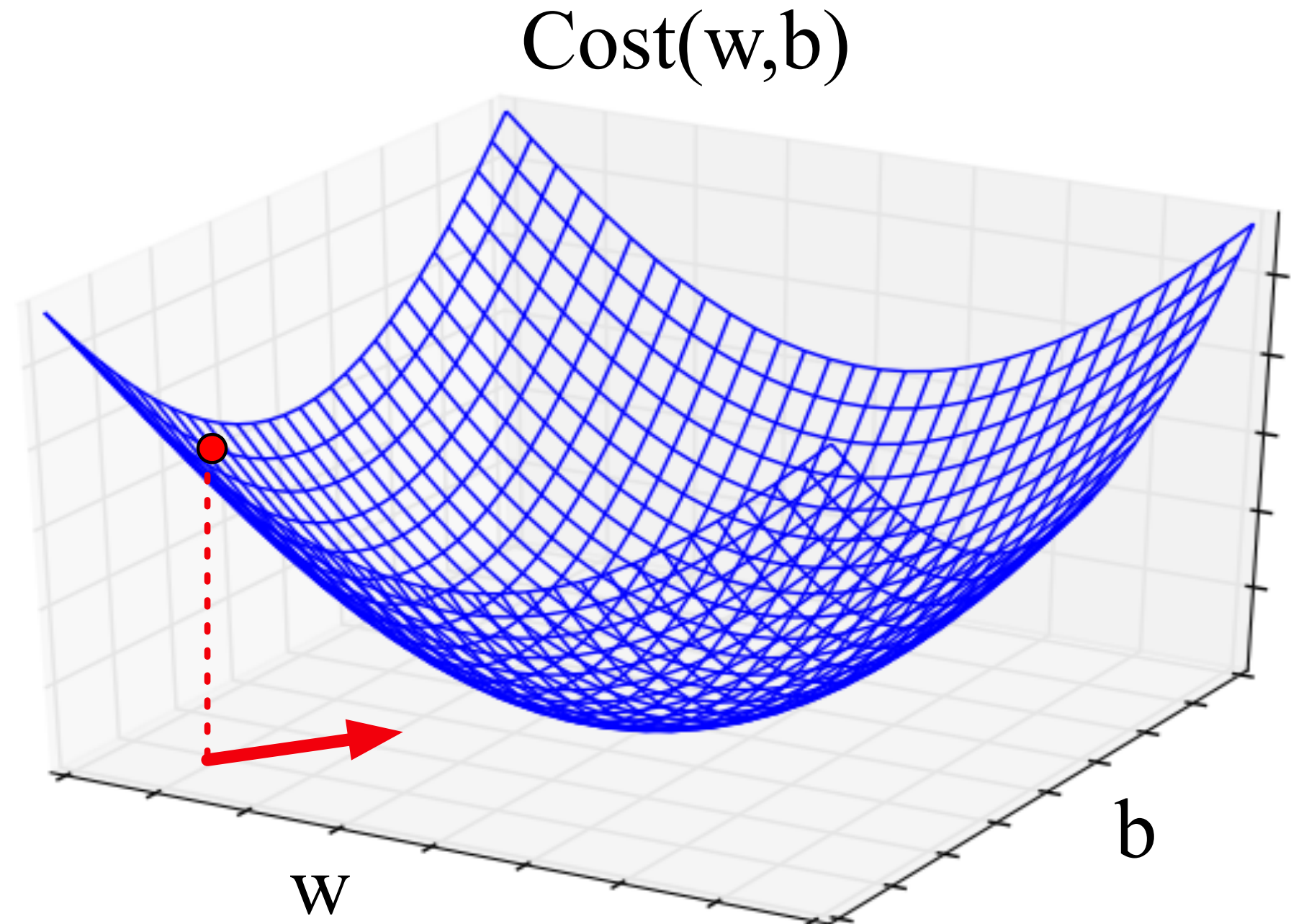
We want to know where in the N -dimensional space (of the N parameters that make up θ) we should move.

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



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 ∂ of the loss ∂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 θ 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 θ 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** θ

where: L is the loss function

f is a function parameterized by θ

x is the set of training inputs $x^{(1)}, x^{(2)}, \dots, x^{(m)}$

y is the set of training outputs (labels) $y^{(1)}, y^{(2)}, \dots, y^{(m)}$

$\theta \leftarrow 0$

repeat til done

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 θ to maximize loss?

3. $\theta \leftarrow \theta - \eta g$ # Go the other way instead

return θ

Hyperparameters

The learning rate η is a **hyperparameter**

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

Hyperparameters:

- 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.

Stochastic Gradient Descent

Logistic
Regression

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 Θ^0 are zero:

$$w_1 = w_2 = b = 0$$

$$\eta = 0.1$$

Example of gradient descent

Update step for update θ is:

$$w_1 = w_2 = b = 0;$$

$$x_1 = 3; \quad x_2 = 2$$

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

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:

$$\nabla_{w,b} = \begin{bmatrix} \frac{\partial L_{\text{CE}}(\hat{y}, y)}{\partial w_1} \\ \frac{\partial L_{\text{CE}}(\hat{y}, y)}{\partial w_2} \\ \frac{\partial L_{\text{CE}}(\hat{y}, y)}{\partial b} \end{bmatrix}$$

Example of gradient descent

Update step for update θ is:

$$w_1 = w_2 = b = 0;$$

$$x_1 = 3; \quad x_2 = 2$$

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

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:

$$\nabla_{w,b} = \begin{bmatrix} \frac{\partial L_{\text{CE}}(\hat{y}, y)}{\partial w_1} \\ \frac{\partial L_{\text{CE}}(\hat{y}, y)}{\partial w_2} \\ \frac{\partial L_{\text{CE}}(\hat{y}, y)}{\partial b} \end{bmatrix} = \begin{bmatrix} \phantom{\frac{\partial L_{\text{CE}}(\hat{y}, y)}{\partial w_1}} \\ \phantom{\frac{\partial L_{\text{CE}}(\hat{y}, y)}{\partial w_2}} \\ \phantom{\frac{\partial L_{\text{CE}}(\hat{y}, y)}{\partial b}} \end{bmatrix}$$

Example of gradient descent

Update step for update θ is:

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

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

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:

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

Example of gradient descent

Update step for update θ is:

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

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

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:

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

Example of gradient descent

Update step for update θ is:

$$w_1 = w_2 = b = 0;$$

$$x_1 = 3; \quad x_2 = 2$$

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

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 θ^1 by moving θ^0 in the opposite direction from the gradient:

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

$$\theta^1 =$$

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 θ^1 by moving θ^0 in the opposite direction from the gradient:

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

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

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 θ^1 by moving θ^0 in the opposite direction from the gradient:

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

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

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 θ^1 by moving θ^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

Scale: Inference over many
examples at once

Processing many binary examples

Suppose our test set has m examples to classify as 1 or 0.

Each test example $x^{(i)}$, $1 \leq i \leq m$, has a feature vector $\mathbf{x}^{(i)}$

For each $x^{(i)}$ we want to compute $\hat{y}^{(i)} = P(y^{(i)} = 1 | x^{(i)})$

We could just use a for-loop,

```
foreach            $x^{(i)}$            in input  $[x^{(1)}, x^{(2)}, \dots, x^{(m)}]$   
     $\hat{y}^{(i)} = P(y^{(i)} = 1) = \sigma(\mathbf{w} \cdot \mathbf{x}^{(i)} + b)$ 
```

Processing many examples

$$\text{foreach } x^{(i)} \text{ in input } [x^{(1)}, x^{(2)}, \dots, x^{(m)}]$$
$$\hat{y}^{(i)} = P(y^{(i)} = 1) = \sigma(\mathbf{w} \cdot \mathbf{x}^{(i)} + b)$$

The first three test examples would be:

$$\hat{y}^{(1)} = P(y^{(1)} = 1 | x^{(1)}) = \sigma(\mathbf{w} \cdot \mathbf{x}^{(1)} + b)$$

$$\hat{y}^{(2)} = P(y^{(2)} = 1 | x^{(2)}) = \sigma(\mathbf{w} \cdot \mathbf{x}^{(2)} + b)$$

$$\hat{y}^{(3)} = P(y^{(3)} = 1 | x^{(3)}) = \sigma(\mathbf{w} \cdot \mathbf{x}^{(3)} + b)$$

Processing many examples

We can do this efficiently with one matrix operation!!!

First, let's pack the input into a single matrix **X**

- each row i is the feature vector for $x^{(i)}$ ($= \mathbf{x}^{(i)}$)
- each example has f features/weights, so **X** has shape $[m \times f]$

$$\mathbf{X} = \begin{bmatrix} x_1^{(1)} & x_2^{(1)} & \dots & x_f^{(1)} \\ x_1^{(2)} & x_2^{(2)} & \dots & x_f^{(2)} \\ x_1^{(3)} & x_2^{(3)} & \dots & x_f^{(3)} \\ \dots & & & \end{bmatrix}$$

Let

$\mathbf{b} = [b, b, \dots, b]$, is a vector of shape $[m \times 1]$

$\hat{\mathbf{y}} = [\hat{y}^{(1)}, \hat{y}^{(2)}, \dots, \hat{y}^{(m)}]$ is the output vector, $[m \times 1]$

$\mathbf{w} = [w_1, w_2, \dots, w_f]$ is a weight vector $[f \times 1]$

$$\mathbf{X} = \begin{bmatrix} x_1^{(1)} & x_2^{(1)} & \dots & x_f^{(1)} \\ x_1^{(2)} & x_2^{(2)} & \dots & x_f^{(2)} \\ x_1^{(3)} & x_2^{(3)} & \dots & x_f^{(3)} \\ \dots & & & \end{bmatrix}$$

Now:

$$\hat{\mathbf{y}} = \sigma(\mathbf{X}\mathbf{w} + \mathbf{b})$$

$\mathbf{X}$ shape is $[m \times f]$

$$\mathbf{X} = \begin{bmatrix} x_1^{(1)} & x_2^{(1)} & \dots & x_f^{(1)} \\ x_1^{(2)} & x_2^{(2)} & \dots & x_f^{(2)} \\ x_1^{(3)} & x_2^{(3)} & \dots & x_f^{(3)} \\ \dots & & & \end{bmatrix} \quad \mathbf{w} = \begin{bmatrix} w_1 \\ w_2 \\ \dots \\ w_f \end{bmatrix} \quad \mathbf{b} = \begin{bmatrix} b \\ b \\ \dots \\ b \end{bmatrix}$$

$$\hat{\mathbf{y}} = \sigma(\mathbf{X} \mathbf{w} + \mathbf{b})$$

$(m \times 1) \qquad (m \times f) \quad (f \times 1) \quad (m \times 1)$

$$\hat{y}^{(1)} = \sigma \left([x_1^{(1)}, x_2^{(1)}, \dots, x_f^{(1)}] \cdot [w_1, w_2, \dots, w_f] + b \right)$$

Ordering

Note that new efficient matrix version is reordered

$$\hat{\mathbf{y}} = \sigma(\mathbf{X}\mathbf{w} + \mathbf{b})$$

Vector version for one example we saw above:

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

We could have kept the old order by defining X differently
But it's convenient to have each row be one example

Logistic Regression

Scale: Inference over many
examples at once

Precision, Recall, and F1

Logistic
Regression

Evaluating Classifiers: How well does our classifier work?

Let's first address binary classifiers:

- Is this email spam?
spam (+) or not spam (-)
- Is this post about Delicious Pie Company?
about Del. Pie Co (+) or not about Del. Pie Co(-)

We'll need to know

1. What did our classifier say about each email or post?
2. What should our classifier have said, i.e., the correct answer, usually as defined by humans ("gold label")

First step in evaluation: The confusion matrix

		<i>gold standard labels</i>	
		gold positive	gold negative
<i>system output labels</i>	system positive	true positive	false positive
	system negative	false negative	true negative

Accuracy on the confusion matrix

		<i>gold standard labels</i>	
		gold positive	gold negative
<i>system output labels</i>	system positive	true positive	false positive
	system negative	false negative	true negative

$$\text{accuracy} = \frac{\text{tp} + \text{tn}}{\text{tp} + \text{fp} + \text{tn} + \text{fn}}$$

Why don't we use accuracy?

Accuracy doesn't work well when we're dealing with uncommon or imbalanced classes

Suppose we look at 1,000,000 social media posts to find Delicious Pie-lovers (or haters)

- 100 of them talk about our pie
- 999,900 are posts about something unrelated

Imagine the following simple classifier

Every post is "not about pie"

Accuracy re: pie posts

100 posts are about pie; 999,900 aren't

gold standard labels

		gold positive	gold negative
<i>system output labels</i>	system positive	true positive	false positive
	system negative	false negative	true negative

$$\text{accuracy} = \frac{\text{tp} + \text{tn}}{\text{tp} + \text{fp} + \text{tn} + \text{fn}}$$

Why don't we use accuracy?

Accuracy of our "nothing is pie" classifier

999,900 true negatives and 100 false negatives

Accuracy is $999,900/1,000,000 = 99.99\%$!

But useless at finding pie-lovers (or haters)!!

Which was our goal!

Accuracy doesn't work well for unbalanced classes

Most tweets are not about pie!

Instead of accuracy we use precision and recall

		<i>gold standard labels</i>		
		gold positive	gold negative	
<i>system output labels</i>	system positive	true positive	false positive	precision = $\frac{tp}{tp+fp}$
	system negative	false negative	true negative	
		recall = $\frac{tp}{tp+fn}$		accuracy = $\frac{tp+tn}{tp+fp+tn+fn}$

Precision: % of selected items that are correct

Recall: % of correct items that are selected

Precision/Recall aren't fooled by the "just call everything negative" classifier!

Stupid classifier: Just say no: every tweet is "not about pie"

- 100 tweets talk about pie, 999,900 tweets don't
- Accuracy = $999,900/1,000,000 = 99.99\%$

But the Recall and Precision for this classifier are terrible:

$$\mathbf{Recall} = \frac{\text{true positives}}{\text{true positives} + \text{false negatives}}$$

$$\mathbf{Precision} = \frac{\text{true positives}}{\text{true positives} + \text{false positives}}$$

A combined measure: F1

F1 is a combination of precision and recall.

$$F_1 = \frac{2PR}{P + R}$$

F1 is a special case of the general "F-measure"

F-measure is the (weighted) harmonic mean of precision and recall

$$\text{HarmonicMean}(a_1, a_2, a_3, a_4, \dots, a_n) = \frac{n}{\frac{1}{a_1} + \frac{1}{a_2} + \frac{1}{a_3} + \dots + \frac{1}{a_n}}$$

$$F = \frac{1}{\alpha \frac{1}{P} + (1 - \alpha) \frac{1}{R}} \quad \text{or} \left(\text{with } \beta^2 = \frac{1 - \alpha}{\alpha} \right) \quad F = \frac{(\beta^2 + 1)PR}{\beta^2 P + R}$$

F1 is a special case of F-measure with $\beta=1$, $\alpha=\frac{1}{2}$

Suppose we have more than 2 classes?

Lots of text classification tasks have more than two classes.

- Sentiment analysis (positive, negative, neutral) , named entities (person, location, organization)

We can define precision and recall for multiple classes like this 3-way email task:

		<i>gold labels</i>			
		urgent	normal	spam	
<i>system output</i>	urgent	8	10	1	precision_u = $\frac{8}{8+10+1}$
	normal	5	60	50	precision_n = $\frac{60}{5+60+50}$
	spam	3	30	200	precision_s = $\frac{200}{3+30+200}$
		recall_u = $\frac{8}{8+5+3}$	recall_n = $\frac{60}{10+60+30}$	recall_s = $\frac{200}{1+50+200}$	

How to combine P/R values for different classes: Microaveraging vs Macroaveraging

Class 1: Urgent

	true urgent	true not
system urgent	8	11
system not	8	340

$$\text{precision} = \frac{8}{8+11} = .42$$

Class 2: Normal

	true normal	true not
system normal	60	55
system not	40	212

$$\text{precision} = \frac{60}{60+55} = .52$$

Class 3: Spam

	true spam	true not
system spam	200	33
system not	51	83

$$\text{precision} = \frac{200}{200+33} = .86$$

Pooled

	true yes	true no
system yes	268	99
system no	99	635

$$\text{microaverage precision} = \frac{268}{268+99} = .73$$

$$\text{macroaverage precision} = \frac{.42+.52+.86}{3} = .60$$

Precision, Recall, and F1

Logistic
Regression

Avoiding Harms in Classification

Logistic
Regression

Harms of classification

Classifiers, like any NLP algorithm, can cause harms

This is true for any classifier, whether Naive Bayes or other algorithms

Representational Harms

- Harms caused by a system that demeans a social group
 - Such as by perpetuating negative stereotypes about them.
- Kiritchenko and Mohammad 2018 study
 - Examined 200 **sentiment analysis** systems on pairs of sentences
 - **Identical** except for names:
 - common African American (Shaniqua) or European American (Stephanie).
 - Like "I talked to Shaniqua yesterday" vs "I talked to Stephanie yesterday"
- Result: systems assigned **lower sentiment** and more negative emotion to sentences with **African American names**
- Downstream harm:
 - Perpetuates stereotypes about African Americans
 - African Americans treated differently by NLP tools like sentiment (widely used in marketing research, mental health studies, etc.)

Harms of Censorship

- **Toxicity detection** is the text classification task of detecting hate speech, abuse, harassment, or other kinds of toxic language.
 - Widely used in online content moderation
- Toxicity classifiers incorrectly flag non-toxic sentences that simply mention minority identities (like the words "blind" or "gay")
 - women (Park et al., 2018),
 - disabled people (Hutchinson et al., 2020)
 - gay people (Dixon et al., 2018; Oliva et al., 2021)
- Downstream harms:
 - Censorship of speech by disabled people and other groups
 - Speech by these groups becomes less visible online
 - Writers might be nudged by these algorithms to avoid these words making people less likely to write about themselves or these groups.

Performance Disparities

1. Text classifiers perform worse on many **languages** of the world due to lack of data or labels
2. Text classifiers perform worse on **varieties** of even high-resource languages like English
 - Example task: **language identification**, a first step in NLP pipeline ("Is this post in English or not?")
 - English language detection performance worse for writers who are African American (Blodgett and O'Connor 2017) or from India (Jurgens et al., 2017)

Harms in text classification

- **Causes:**
 - Issues in the data; NLP systems amplify biases in training data
 - Problems in the labels
 - Problems in the algorithms (like what the model is trained to optimize)
- **Prevalence:** The same problems occur throughout NLP (including large language models)
- **Solutions:** There are no general mitigations or solutions
 - But harm mitigation is an active area of research
 - And there are standard benchmarks and tools that we can use for measuring some of the harms

Avoiding Harms in Classification

Logistic
Regression