Lecture 3:
Smoothing n-grams and Logistic Regression

Instructor: Swabha Swayamdipta
USC CSCI 444 NLP
Sep 3, 2025

Some slides adapted from Dan Jurafsky and Chris Manning

Announcements

- HW1 released on 9/1; due 9/17
- Assignments and projects will now be awarded differently
- HW Grading scheme
  - Peer anonymous graders and gradees, randomly assigned
  - Will check a few random assignments and their gradings to ensure fairness
- Next Mon:
  - Project Pitches
    - Every student pitches a 5-minute project idea for which all the other students vote. The pitch should outline the problem being solved and why should we care about it. There should be a clear connection to language models. Clear description of what the inputs and the outputs are, ideally with real-world examples. Name the project idea. See website for examples of projects from previous iterations of the class
  - Quiz 1: 5-10 multiple choice questions from everything taught in class till today

Example Projects

- Goal: MixRx classifies drug combination interactions as Additive, Synergistic, or Antagonistic, given a multi-drug patient history using language models
- Motivation: Evaluates if LLMs can be used for medical / biological prediction tasks in high-stakes domains
- Compares the classification accuracy of 4 models, GPT-2, Mistral Instruct 2.0, and their fine-tuned counterparts

MixRx

Risha Surana, Cameron Saidock, Hugo Chacon
University of Southern California

Example Input Prompt

Given the following set of drugs, decide if the synergy of the drug combination is synergistic or antagonistic: Sepantronium Bromide, Crizotinib, Pictilisib.

"Pictilisib and Sepantronium Bromide have a Loewe score of: -0.3191, HSA score of: -0.4446, and ZIP score of: -0.7671. Crizotinib and Sepantronium Bromide have a Loewe score of... Generate a prediction on whether the overall drug combination is synergistic or antagonistic. Provide a quantitative synergy score based on the weighted average of Loewe, HSA, and ZIP scores, qualitative reasoning supporting your prediction, and a confidence level in your prediction. Format your answer as specified below: { \"Prediction\": \"Antagonistic\", \"Qualitative Reasoning\": \"This...", }

MotivationGPT

Avinash Gala agala@usc.edu
Albert Tan ahtan@usc.edu
Erel Papo papo@usc.edu

- Motivation: to bring motivation and inspiration into the daily lives of people through the impersonation of renowned fitness influencer David Goggins.
- Input: Struggles with motivation. Output: advice
- Evaluation: Through human surveys to verify if models actually helped!

Lecture Outline

- Announcements
- n-grams and Smoothing
- Basics of Supervised Machine Learning
  - I. Data: Preprocessing and Feature Extraction
  - II. Model:
    - I. Logistic Regression
  - III. Loss
  - IV. Optimization Algorithm
  - V. Inference

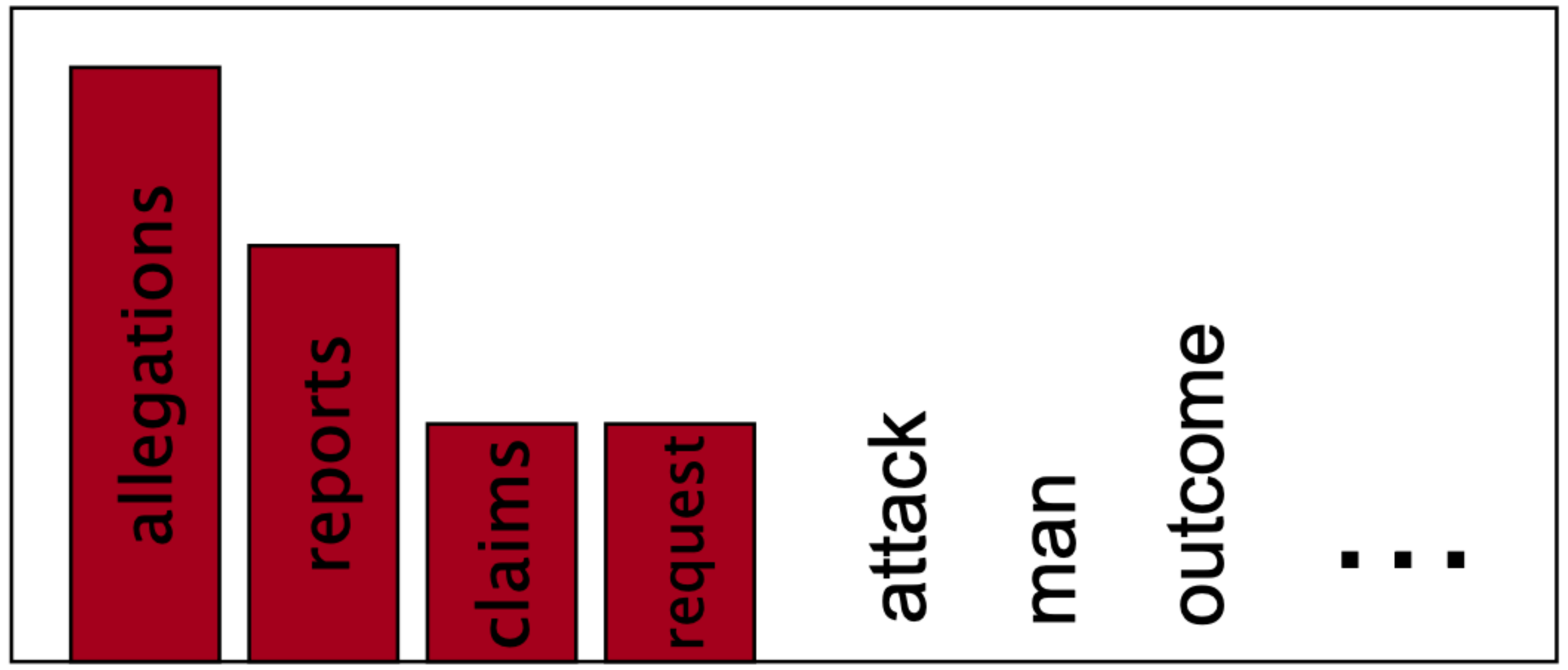
USC Viterbi

Smoothing for n-gram language models

Smoothing ~ Massaging Probability Masses

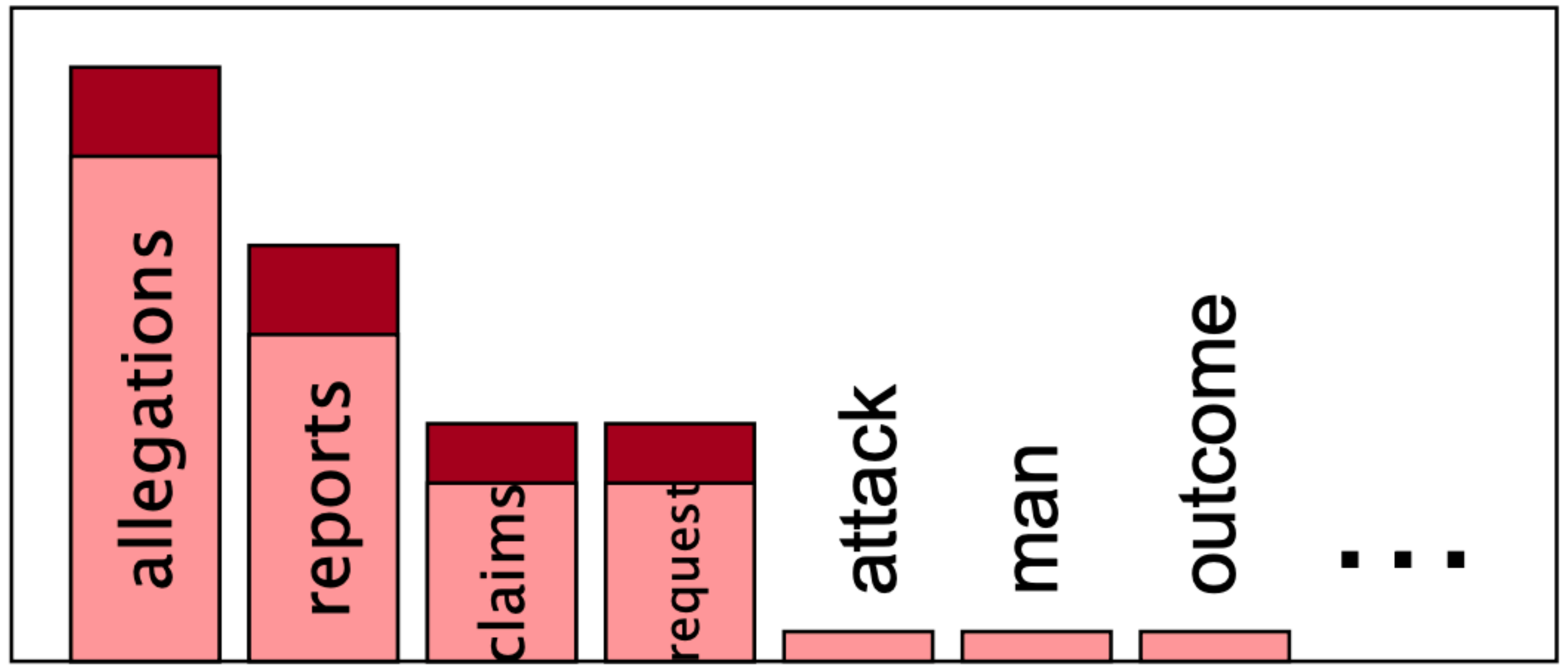
When we have sparse statistics: $Count(w \mid \text{denied the})$

3 allegations
2 reports
1 claims
1 request
7 total



Steal probability mass to generalize better: $Count(w \mid \text{denied the})$

2.5 allegations
1.5 reports
0.5 claims
0.5 request
2 other
7 total



Add-One Estimation

Laplace smoothing

1. Pretend we saw each n-gram one more time than we did
2. Just add one to all the n-gram counts!
3. All the counts that used to be zero will now have a count of 1…

Add-1 estimate for Unigrams

$$P_{Add-1}(w_i) = \frac{c(w_i) + 1}{\sum_w (c(w) + 1)} = \frac{c(w_i) + 1}{V + \sum_w c(w)}$$

Add-1 estimate for Bigrams

$$P_{Add-1}(w_i \mid w_{i-1}) = \frac{c(w_{i-1}w_i) + 1}{c(w_{i-1}) + V}$$

Original vs Add-1-smoothed bigram counts

Original, Raw

	i	want	to	eat	chinese	food	lunch	spend
i	5	827	0	9	0	0	0	2
want	2	0	608	1	6	6	5	1
to	2	0	4	686	2	0	6	211
eat	0	0	2	0	16	2	42	0
chinese	1	0	0	0	0	82	1	0
food	15	0	15	0	1	4	0	0
lunch	2	0	0	0	0	1	0	0
spend	1	0	1	0	0	0	0	0

Reconstructed

	i	want	to	eat	chinese	food	lunch	spend
i	3.8	527	0.64	6.4	0.64	0.64	0.64	1.9
want	1.2	0.39	238	0.78	2.7	2.7	2.3	0.78
to	1.9	0.63	3.1	430	1.9	0.63	4.4	133
eat	0.34	0.34	1	0.34	5.8	1	15	0.34
chinese	0.2	0.098	0.098	0.098	0.098	8.2	0.2	0.098
food	6.9	0.43	6.9	0.43	0.86	2.2	0.43	0.43
lunch	0.57	0.19	0.19	0.19	0.19	0.38	0.19	0.19
spend	0.32	0.16	0.32	0.16	0.16	0.16	0.16	0.16

Big change to the counts!

Perhaps 1 is too much, add a fraction?

Add-k smoothing

k is a hyperparameter

Interpolation

Perhaps use some pre-existing evidence

- Condition on less context for contexts you haven't learned much about

Interpolation

- mix unigram, bigram, trigram probabilities for a trigram LM
- mix n-gram, $(n-1)$-gram, … unigram probabilities for an n-gram LM

Interpolation works better than Add-1 / Laplace smoothing

Linear Interpolation

Simple Interpolation

$$\hat{P}(w_i | w_{i-2}w_{i-1}) = \lambda_1 P(w_i) + \lambda_2 P(w_i | w_{i-1}) + \lambda_3 P(w_i | w_{i-2}w_{i-1})$$

$$\sum_k \lambda_k = 1$$

Hyperparameters!

Context-Conditional Interpolation

$$\hat{P}(w_i | w_{i-2}w_{i-1}) = \lambda_3(w_{i-2}^{i-1}) P(w_i | w_{i-2}w_{i-1}) + \lambda_2(w_{i-2}^{i-1}) P(w_i | w_{i-1}) + \lambda_1(w_{i-2}^{i-1}) P(w_i)$$

Different for different bigrams!
Serve as Reconstituted Counts

Different for every unique context

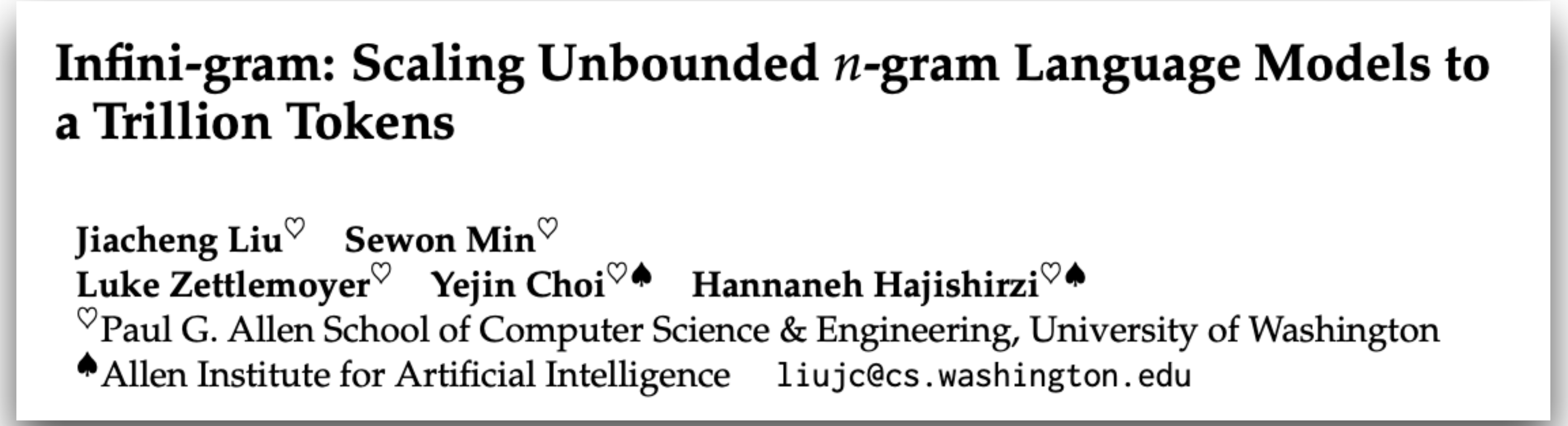
How to set the λs?

Choose λs to maximize the probability of held-out data:

- Fix the *n*-gram probabilities (on the training data)
- Then search for λs that give largest probability to held-out set:

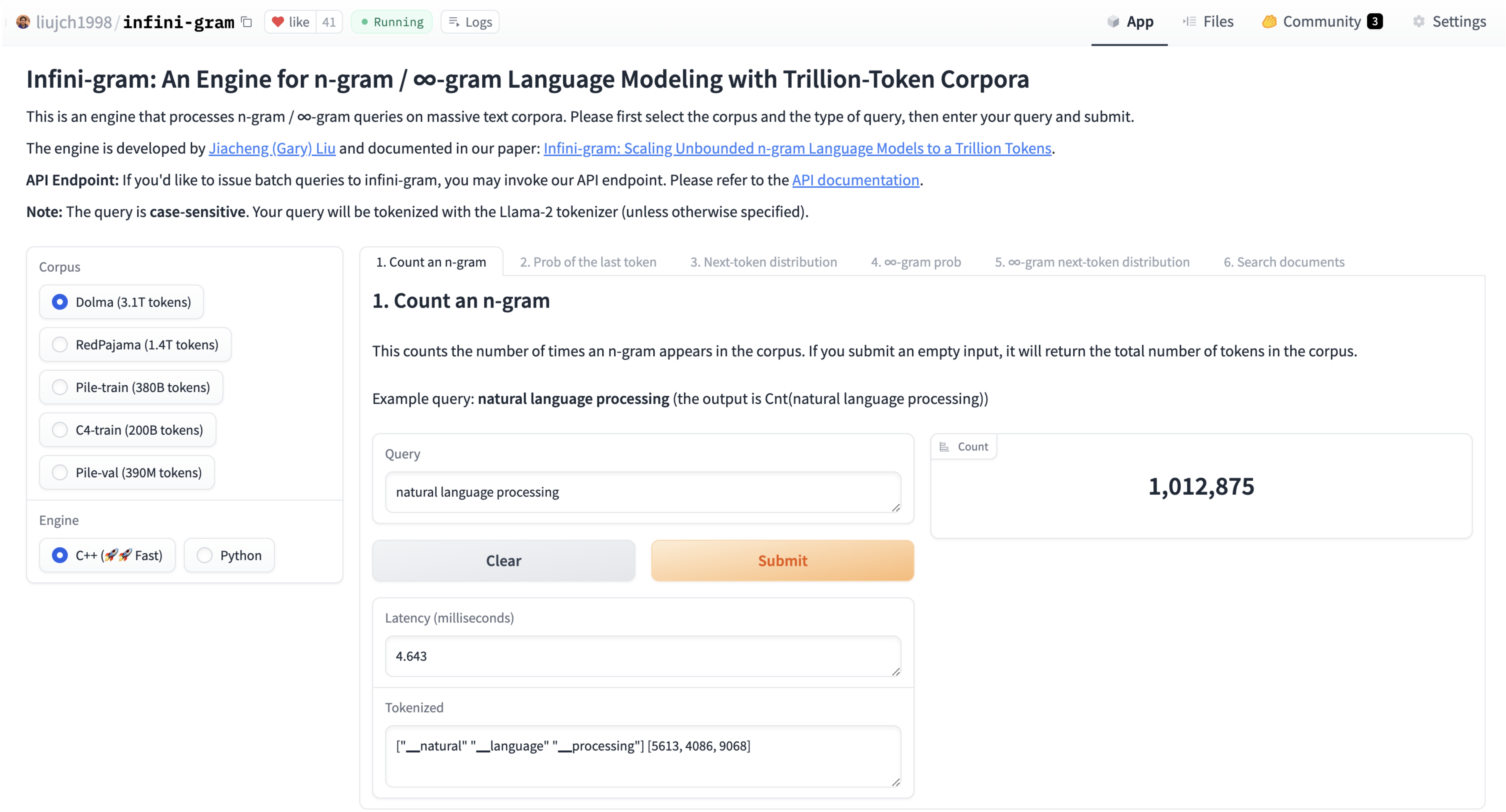
$$logP(w_1 \dots w_n | M(\lambda_1 \dots \lambda_k)) = \sum_i logP_{M(\lambda_1 \dots \lambda_k)}(w_i | w_{i-1})$$

n-grams Today



Infini-gram: Scaling Unbounded n-gram Language Models to a Trillion Tokens

Jiacheng Liu[♡] Sewon Min[♡]
Luke Zettlemoyer[♡] Yejin Choi[♡♠] Hannaneh Hajishirzi[♡♠]
[♡]Paul G. Allen School of Computer Science & Engineering, University of Washington
[♠]Allen Institute for Artificial Intelligence liujc@cs.washington.edu



- Clever use of smoothing to create n-gram LMs where $n = \infty$, at least in principle
- Trained on several open text corpora: Dolma, RedPajama, Pile, and C4
  - Same corpora are used to train LLMs
- Several applications: search for n-grams, their counts and their probabilities, use them for generation, etc.

Lecture Outline

- Announcements
- n-grams and Smoothing
- Basics of Supervised Machine Learning
  I. Data: Preprocessing and Feature Extraction
  II. Model:
    I. Logistic Regression
  III. Loss
  IV. Optimization Algorithm
  V. Inference

Basics of Supervised Machine Learning

Ingredients of Supervised Machine Learning

I. **Data** as pairs $(\mathbf{x}^{(i)}, y^{(i)})$ s.t $i \in \{1 \dots N\}$
  - The input is usually represented by a feature vector $\mathbf{x}^{(i)} = [x_1, x_2, \dots, x_d]$,
    - e.g. word embeddings

II. **Model**
  - A classification function that computes $\hat{y}$, the estimated class, via $p(y \mid \mathbf{x})$
    - e.g. logistic regression, naïve Bayes, neural nets, Transformers

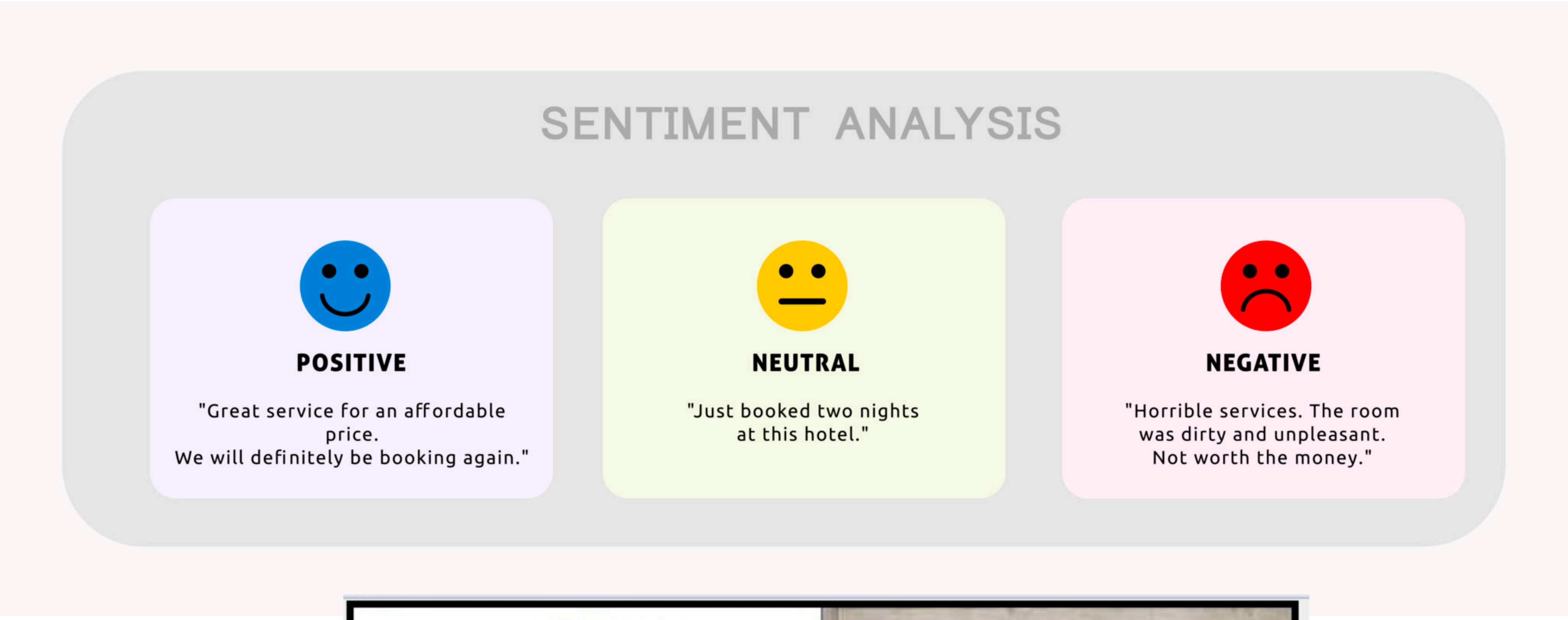
III. **Loss**
  - An objective function for learning
    - e.g. cross-entropy loss, L_{CE}

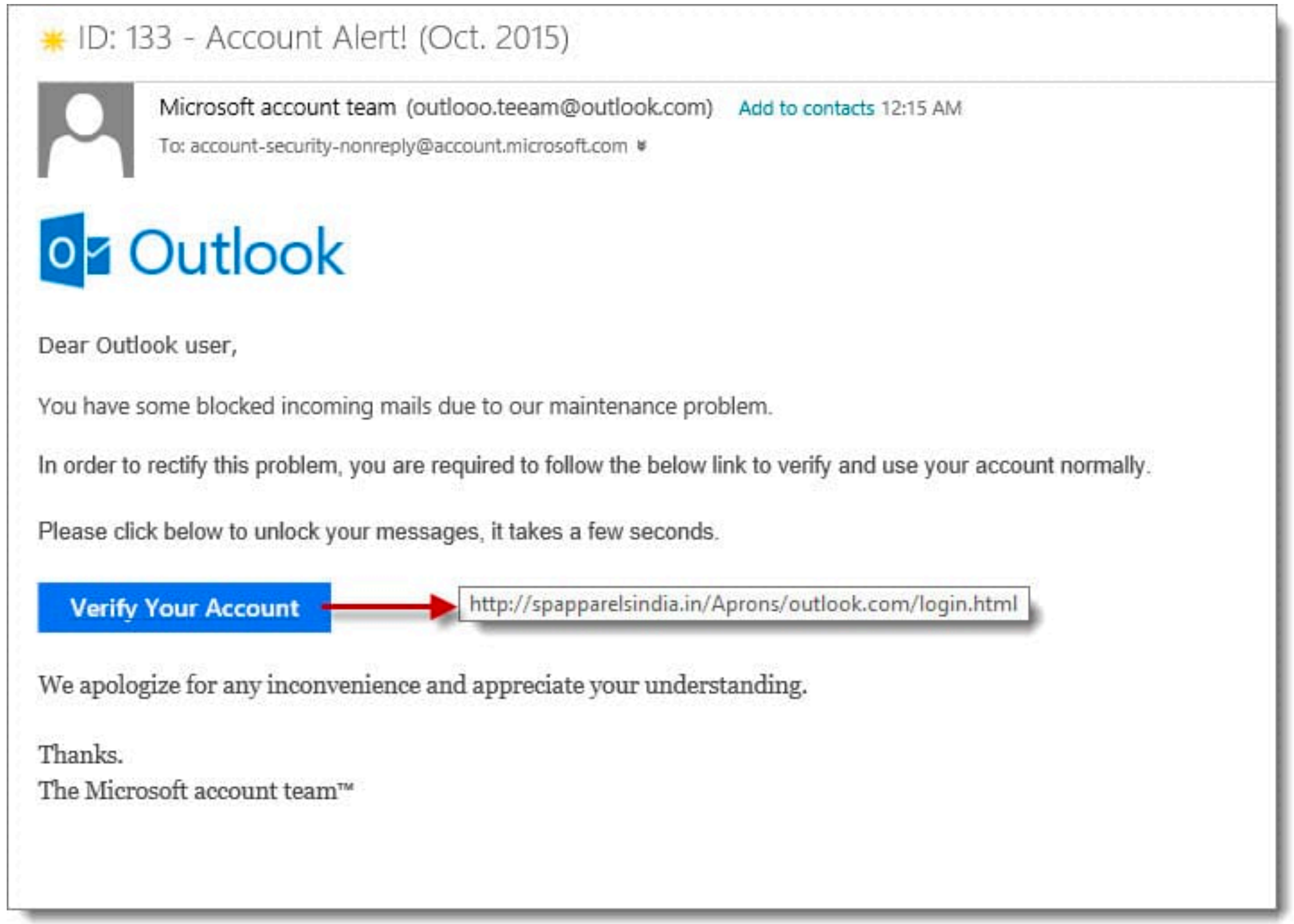
IV. **Optimization**
  - An algorithm for optimizing the objective function
    - e.g. stochastic gradient descent

V. **Inference** / Evaluation

Learning Phase

Text Classification Tasks





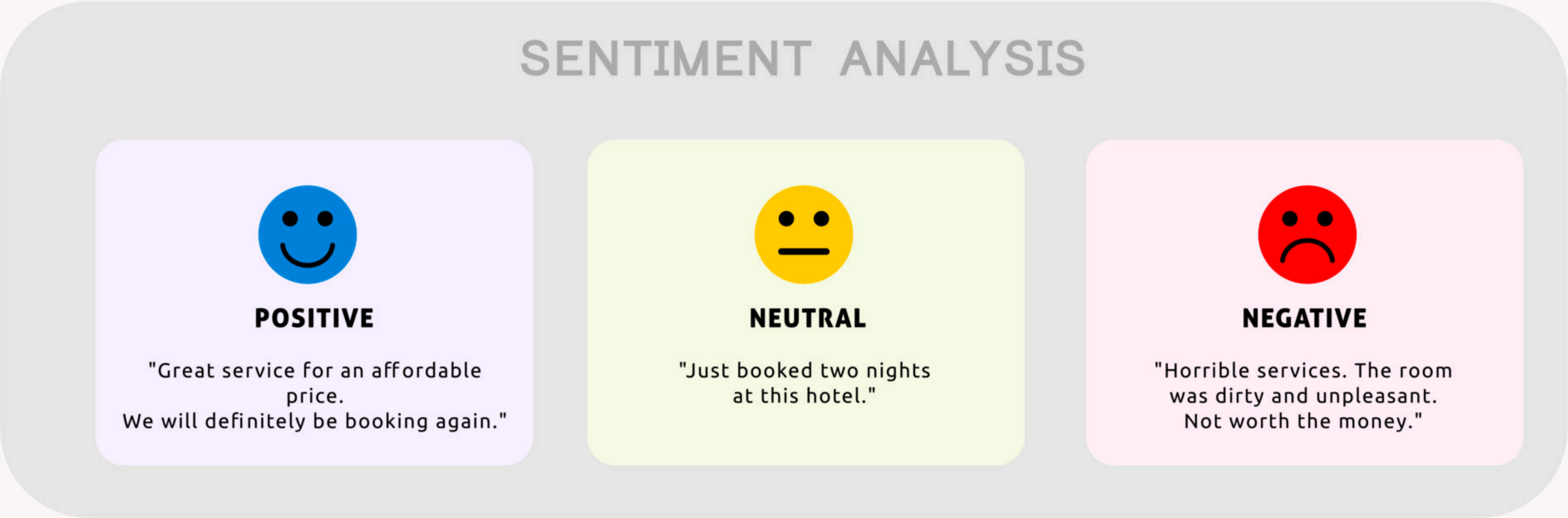
Not just NLP, classification is a general ML technique often applied across a wide variety of prediction tasks!

Is language modeling a classification task?

I. Data:
Preprocessing and Feature Extraction

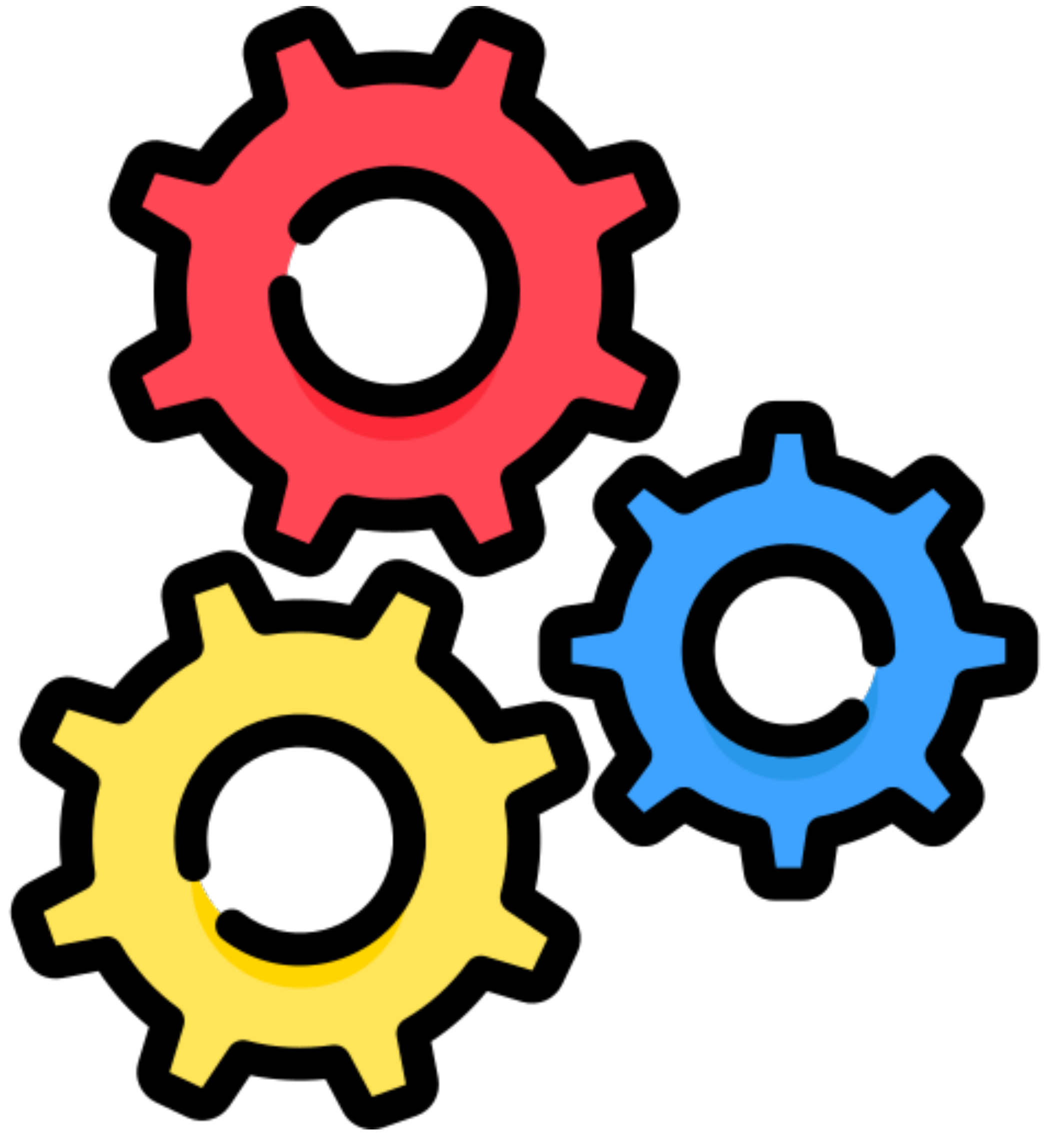
Features in Classification

- The input, $\mathbf{x}$ could be the entire review text, or (manually) broken down into pieces of relevant information
  - These pieces: features
- Examples of feature x_i
  - $x_i =$"review contains 'awesome'"; $w_i = +10$
  - $x_j =$"review contains 'abysmal'"; $w_j = -10$
  - $x_k =$"review contains 'mediocre'"; $w_k = -2$
- Each x_i is associated with a weight w_i which determines how important x_i is
  - (For predicting the positive class)
- May be
  - manually configured or
  - automatically inferred, as in neural nets



Data Pre-processing in Language Models

- Documents containing raw texts must be preprocessed before feature extraction
  - Tokenization: splitting the text into units for processing
    - Removing extra spaces and unhelpful characters, e.g., non-alphabetical characters
    - Removing unhelpful tokens, e.g., external URL links
  - Optional Steps
    - Removing unsafe data, e.g., Personal Identifiable Information and / or hate speech
    - Deduplication of content (only relevant for large LMs)



Still relevant, especially for you to understand what current LMs can automate!

USC Viterbi

II. Model: Logistic Regression

Ingredients of Supervised Machine Learning

I. **Data** as pairs $(\mathbf{x}^{(i)}, y^{(i)})$ s.t $i \in \{1...N\}$
   - The input is usually represented by a feature vector $\mathbf{x}^{(i)} = [x_1, x_2, ..., x_d]$,
     - e.g. word embeddings

II. **Model**
   - A classification function that computes $\hat{y}$, the estimated class, via $p(y \mid \mathbf{x})$
     - e.g. logistic regression, naïve Bayes, neural nets, Transformers

III. **Loss**
   - An objective function for learning
     - e.g. cross-entropy loss, L_{CE}

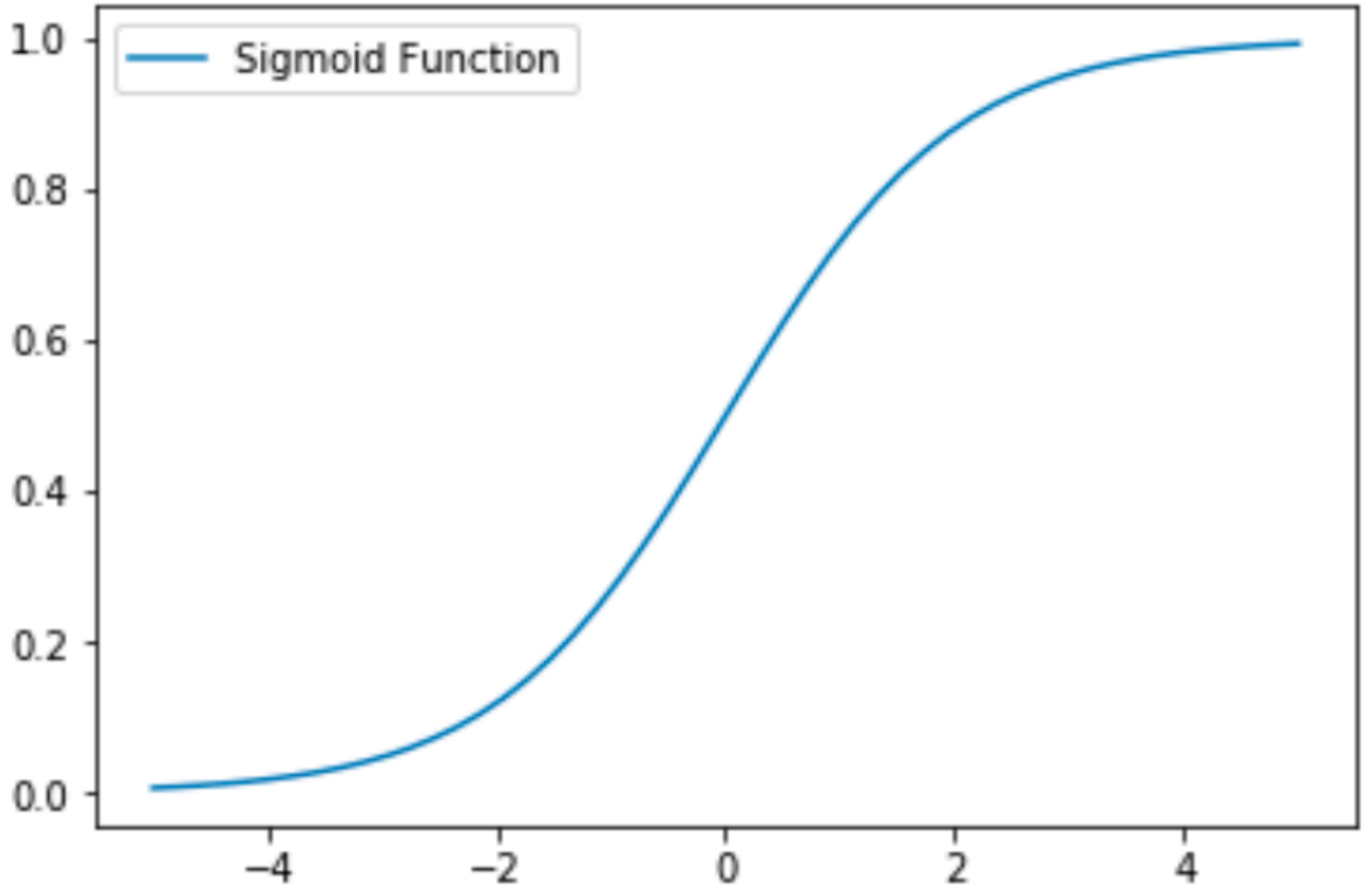
IV. **Optimization**
   - An algorithm for optimizing the objective function
     - e.g. stochastic gradient descent

V. **Inference** / Evaluation

Learning Phase

Example: Logistic Regression

- Important analytic tool in natural and social sciences
- Baseline supervised machine learning tool for classification
- Is also the foundation of neural networks
- Logistic regression is a discriminative classifier
    - Learn a model that can (given the input) distinguish between different classes
- Other classification algorithms: Naïve Bayes, K-Nearest Neighbors, Decision Trees, SVMs

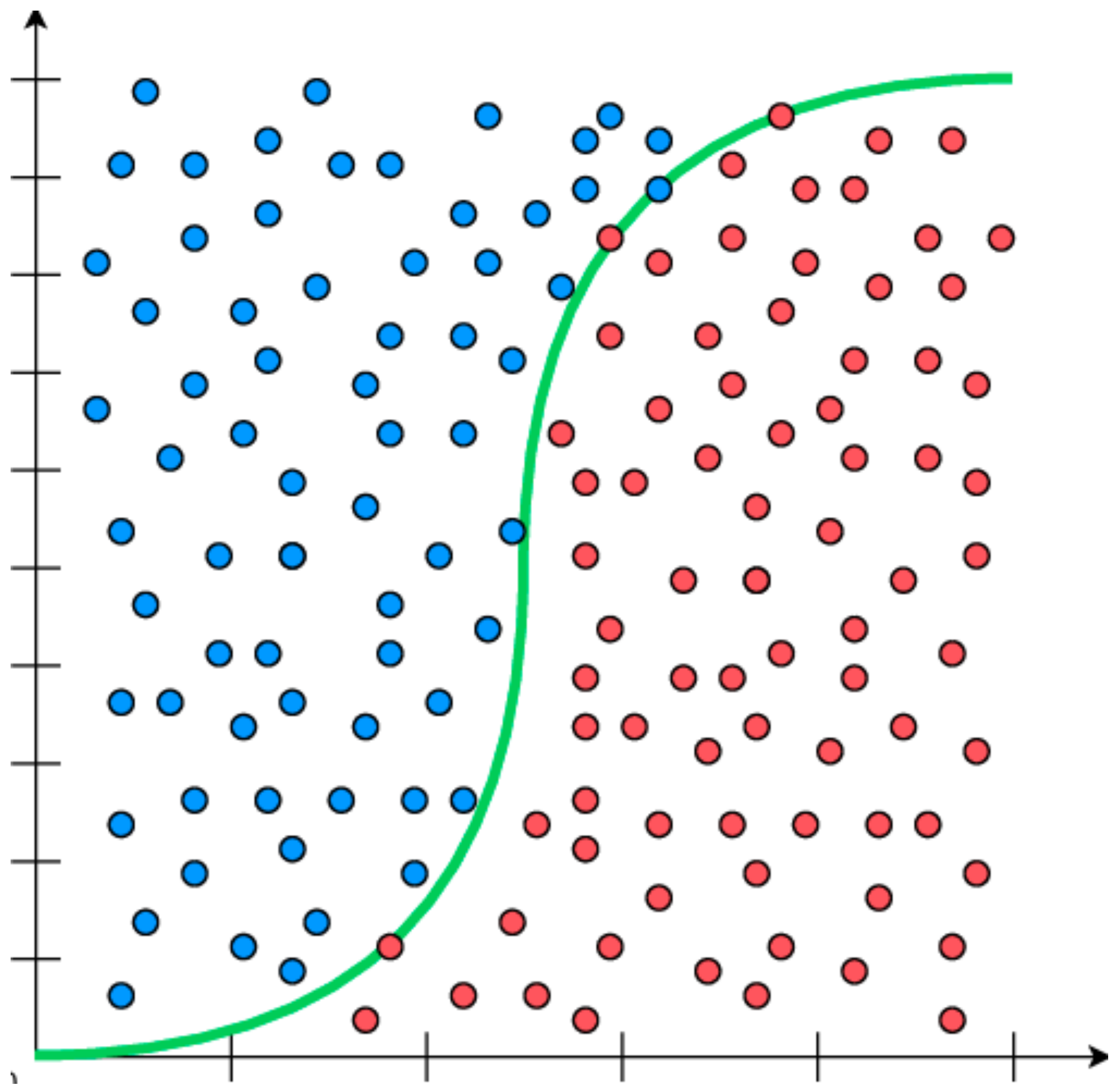


Classification: Single Observation

- Input observation: vector of features, $\mathbf{x} = [x_1, x_2, \ldots, x_n]$
- Output: a predicted class
  - Binary logistic regression $\hat{y} \in \{0,1\}$
  - Multinomial logistic regression (e.g. 5 classes): $\hat{y} \in \{0,1,2,3,4\}$

Text Classification Setup

- Input:
  - The i-th document $\mathbf{x}^{(i)}$
    - Each observation is represented by a feature vector $\mathbf{x}^{(i)} = [x_1^{(i)}, x_2^{(i)}, \ldots, x_d^{(i)}]$
  - The corresponding label $y^{(i)}$ from a fixed set of classes $C = c_1, c_2, \ldots, c_J$
- Output: a predicted class $\hat{y} \in C$
- Binary Classification:
  - given a series of input / output pairs during training:
    - $(\mathbf{x}^{(i)}, y^{(i)})$ where label $y^{(i)} \in C = \{0,1\}$
  - predict, at test time,
    - for input $\mathbf{x}^{test}$, an output $\hat{y}^{test} \in \{0,1\}$



How to get the right y?

- For each feature x_i, introduce a weight w_i, which determines the importance of x_i
  - Sometimes we have a bias term, b or w_0, which is just another weight not associated to any feature
  - Together, all parameters can be termed as $\theta = [w; b]$
- We consider the weighted sum of all features and the bias

$$z = \left(\sum_d w_d x_d + b \right)$$

$$= \mathbf{w} \cdot \mathbf{x} + b$$

If high, $\hat{y} = 1$

If low, $\hat{y} = 0$

But how to determine the threshold?

We need probabilistic models!

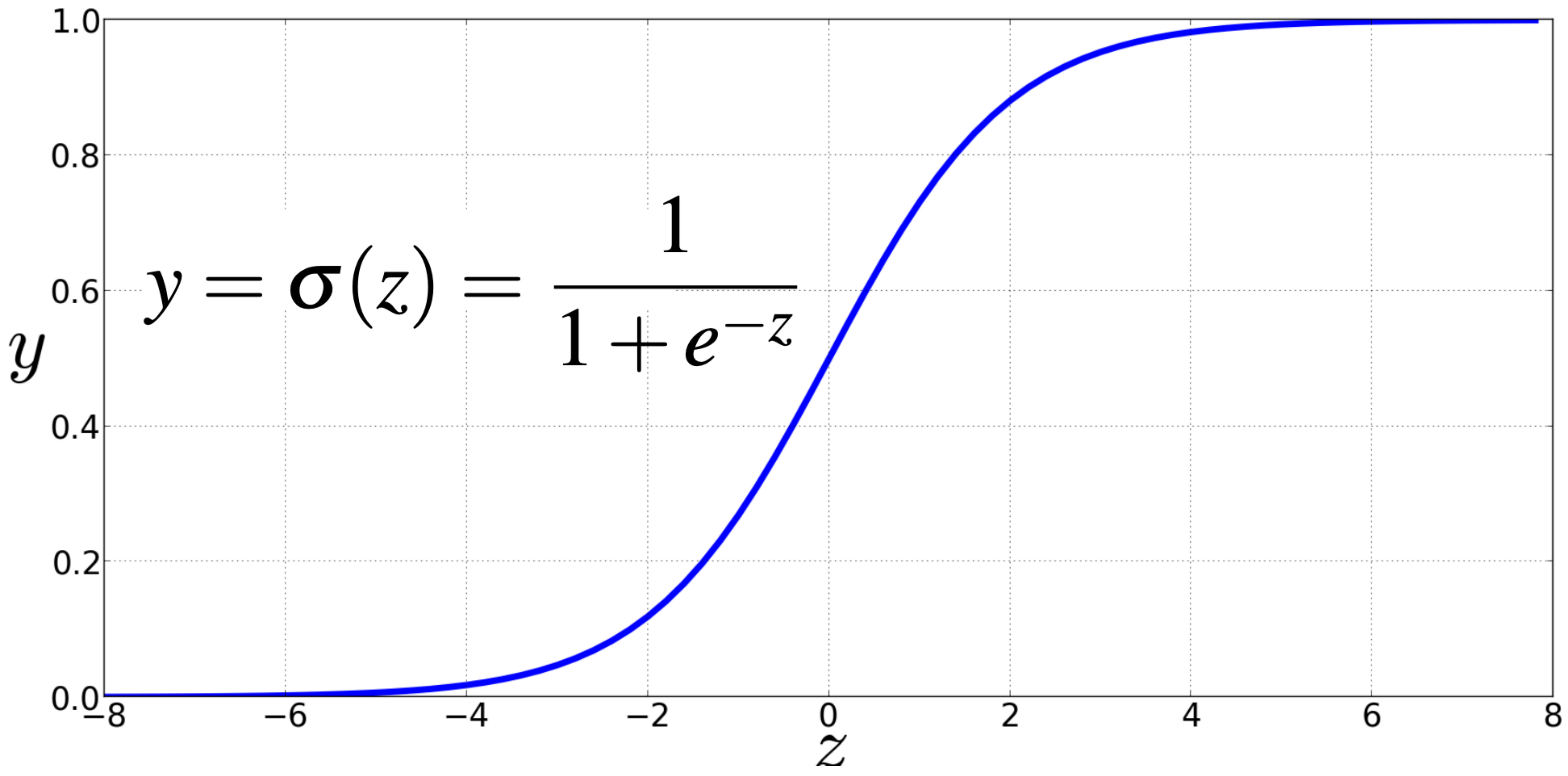
$$P(y = 1 \mid \mathbf{x}; \theta)$$

$$P(y = 0 \mid \mathbf{x}; \theta)$$

Solution: Squish it into the 0-1 range

$$z = \mathbf{w} \cdot \mathbf{x} + b \qquad z \in \mathbb{R}$$

- Sigmoid Function, $\sigma(\cdot)$
  - Non-linear!
- Compute z and then pass it through the sigmoid function
- Treat it as a probability!
- Also, a differentiable function, which makes it a good candidate for optimization (more on this later!)

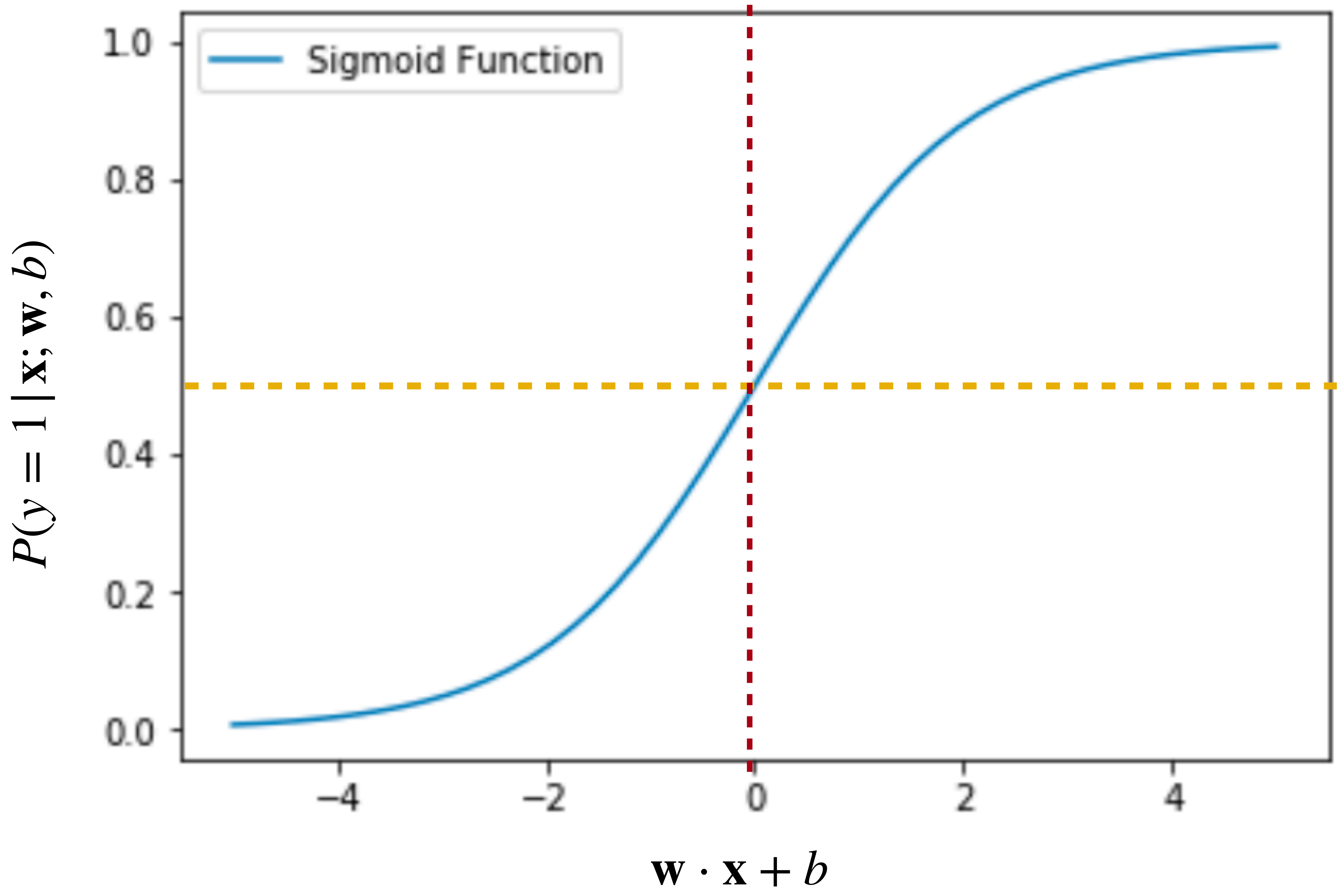


Classification Decision

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

Decision Boundary

$$\hat{y} = \begin{cases} 1 & \text{if } \mathbf{w} \cdot \mathbf{x} + b > 0 \\ 0 & \text{if } \mathbf{w} \cdot \mathbf{x} + b \leq 0 \end{cases}$$

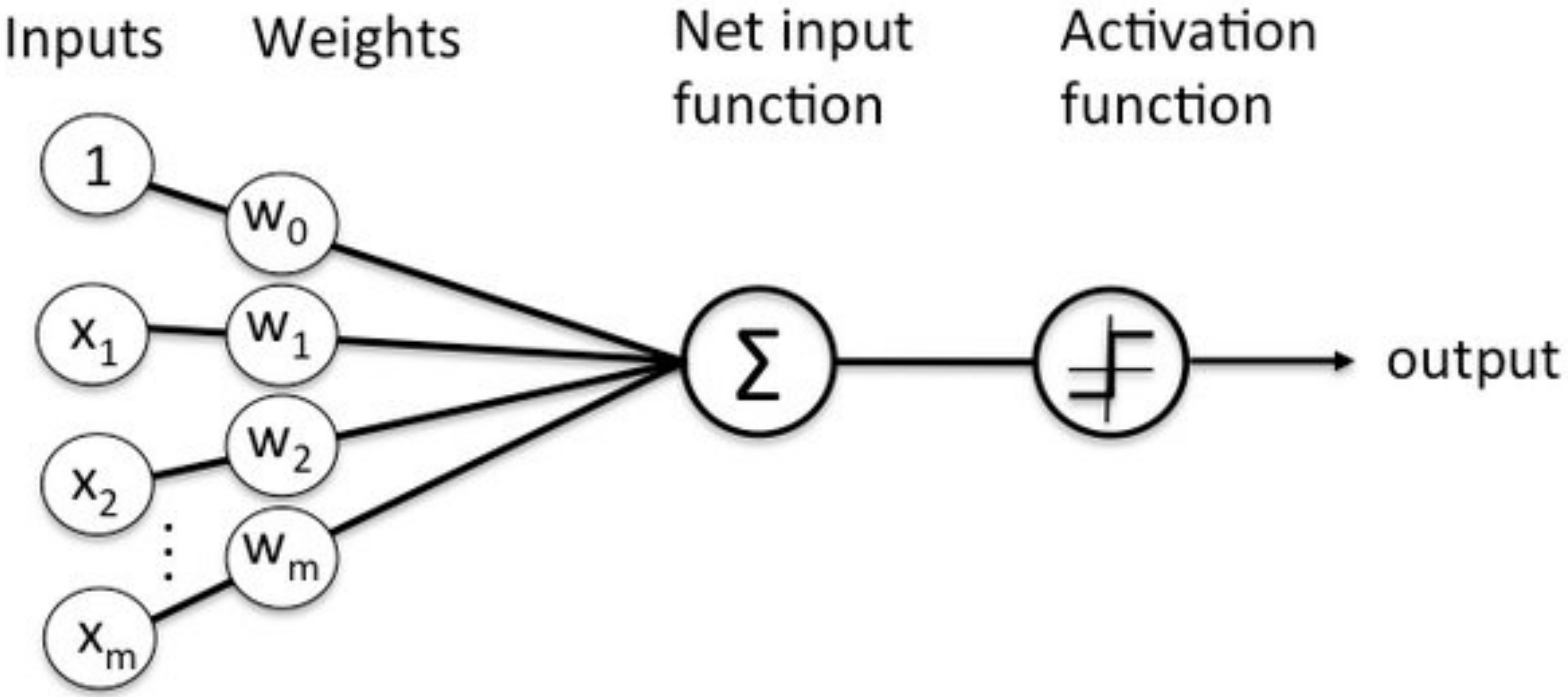


Sigmoids and Probabilities

$$P(y=1\,|\,\mathbf{x};\theta) = \sigma(\mathbf{w}\cdot\mathbf{x}+b)$$
$$= \frac{1}{1+\exp(-(\mathbf{w}\cdot\mathbf{x}+b))}$$

$$P(y=0\,|\,\mathbf{x};\theta) = 1-\sigma(\mathbf{w}\cdot\mathbf{x}+b)$$
$$= 1-\frac{1}{1+\exp(-(\mathbf{w}\cdot\mathbf{x}+b))}$$
$$= \frac{\exp(-(\mathbf{w}\cdot\mathbf{x}+b))}{1+\exp(-(\mathbf{w}\cdot\mathbf{x}+b))}$$
$$= \frac{1}{1+\exp(\mathbf{w}\cdot\mathbf{x}+b)}$$
$$= \sigma(-(\mathbf{w}\cdot\mathbf{x}+b))$$

Another notation



But where do the $\mathbf{w}$'s and the b'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, $\hat{y}$
- Set $\mathbf{w}$ 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 $\mathbf{w}$ and b to minimize the loss.

Loss function

Optimization Algorithm

Lecture Outline

- Announcements
- n-grams and Smoothing
- Basics of Supervised Machine Learning
  - I. Data: Preprocessing and Feature Extraction
  - II. Model:
    - I. Logistic Regression
  - III. Loss
  - IV. Optimization Algorithm
  - V. Inference

USC Viterbi

III. Loss:
Cross-Entropy

USC Viterbi

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

- We want to know how far is the classifier output:
  - $\hat{y} = \sigma(\mathbf{w} \cdot \mathbf{x} + b)$
- From the true (ground truth / gold standard) label:
  - $y \in \{0,1\}$
- This difference is called the loss or cost
  - $L(\hat{y}, y)$ = how much $\hat{y}$ differs from y
  - In other words, how much would you lose if you mispredicted
  - Or how much would it cost you to mispredict

Remember maximum likelihood?

- Here: **conditional** maximum likelihood estimation
- We choose the parameters $\mathbf{w}, b$ that maximize
  - the log probability
    - of the true y labels in the training data
    - **given** the observations x

$$\max \log p(y \mid x)$$

Suppose we flip the coin four times and see (H, H, H, T).

What is p?

$p = 3/4 = 0.75$ maximizes the probability of data sequence (H,H,H,T)

maximum likelihood estimate

Maximizing conditional likelihood

For a single observation

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 $p(y|\mathbf{x})$ from our classifier (the thing we want to maximize) as

Data Likelihood

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

	$\hat{y}=0$	$\hat{y}=.3$	$\hat{y}=.5$	$\hat{y}=.7$	$\hat{y}=1$
$y=0$	1	0.7	0.5	0.3	0
$y=1$	0	0.3	0.5	0.7	1

← **Output estimates** →

	$\hat{y}=0$	$\hat{y}=1$
$y=0$	1	0
$y=1$	0	1

Maximizing conditional likelihood

Goal: maximize probability of the correct label $p(y \mid \mathbf{x})$

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

Now take the log of both sides

$$\log p(y \mid x) = \log(\hat{y}^{y}(1 - \hat{y})^{1-y})$$
$$= y \log \hat{y} + (1 - y)\log(1 - \hat{y})$$

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

Why does this work?

Minimizing negative log likelihood

Goal: maximize probability of the correct label $p(y \mid \mathbf{x})$

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

Measures how well the training data matches the proposed model distribution and how good the model distribution is

Now flip the sign for something to minimize (we minimize the loss / cost)

Minimize: $$L_{CE}(y, \hat{y}) = -\log p(y \mid x) = -[y \log \hat{y} + (1-y)\log(1-\hat{y})]$$
$$= -[y \log \sigma(\mathbf{w} \cdot \mathbf{x} + b) + (1-y)\log \sigma(-(\mathbf{w} \cdot \mathbf{x} + b))]$$

Cross-Entropy Loss

Loss for sentiment classification

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.

Sentiment Example

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

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

Pretty well! What's the loss?

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

USC Viterbi

Sentiment Example: Contd

Now, suppose true value is $y = 0$. How well is our model doing?

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

What's the loss?

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

Sentiment Example: Summary

The loss when the model is right (if true $y = 1$):

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

Loss is bigger when the model is wrong!

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

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

Next: an **optimization algorithm** to update $\mathbf{w}$ and b to minimize the loss

Lecture Outline

- Announcements
- n-grams and Smoothing
- Basics of Supervised Machine Learning
  I. Data: Preprocessing and Feature Extraction
  II. Model:
    I. Logistic Regression
  III. Loss
  IV. Optimization Algorithm
  V. Inference

IV. Optimization: Stochastic Gradient Descent

USC Viterbi

Our goal: minimize the loss

- Loss function is parameterized by weights: $\theta = [\mathbf{w}; b]$
- We will represent $\hat{y}$ as $f(x; \theta)$ to make the dependence on θ more obvious

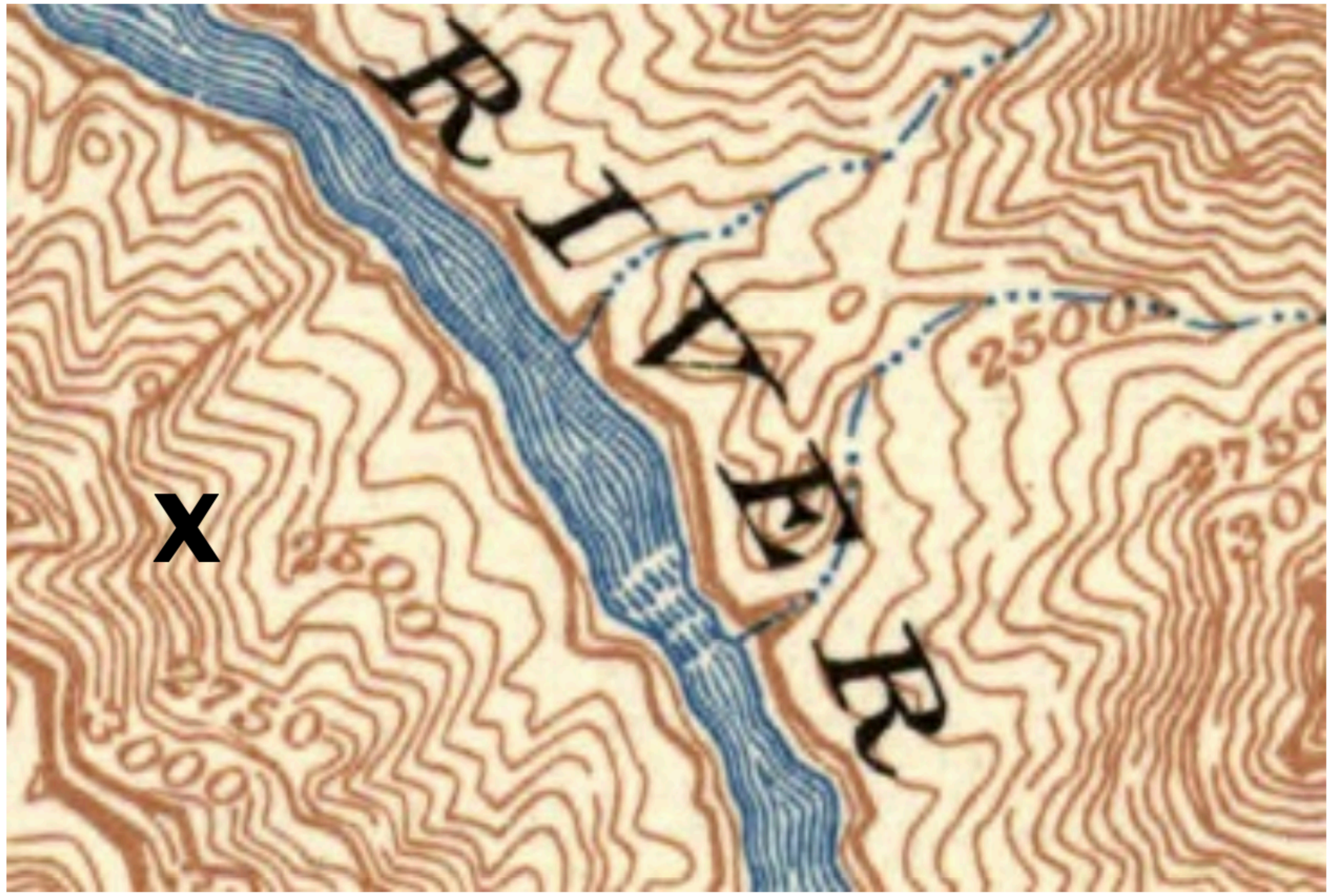
We want the weights that minimize the loss, averaged over all examples:

$$L_{CE}(f(\mathbf{x}^{(i)}; \theta), y^{(i)})$$

Intuition for gradient descent

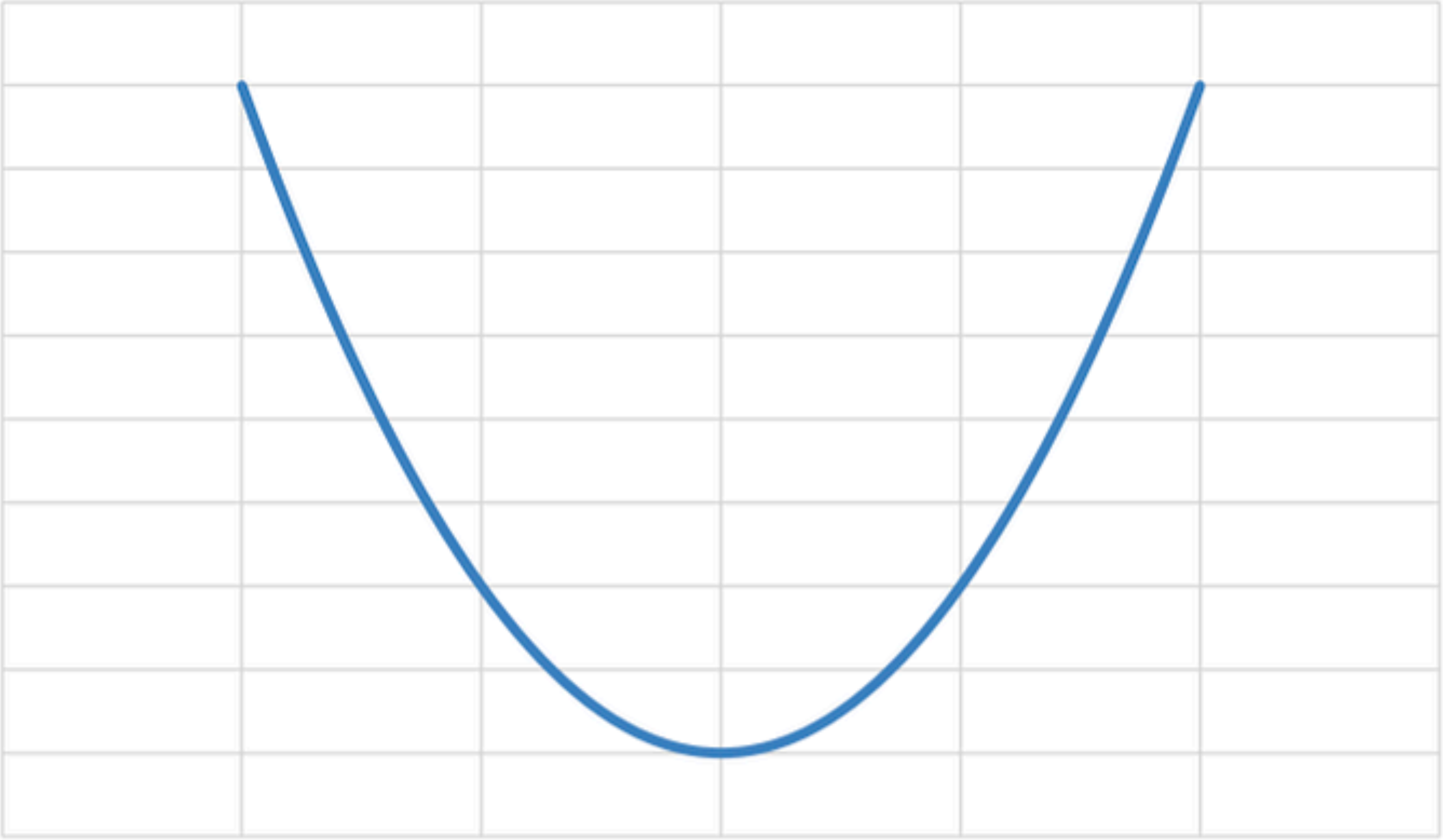
How to get to the bottom of the river canyon?

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



What if multiple equally good alternatives?

Logistic Regression: Loss



Convex function

- Has only one option for steepest gradient
  - Or one minimum
- Gradient descent starting from any point is guaranteed to find the minimum

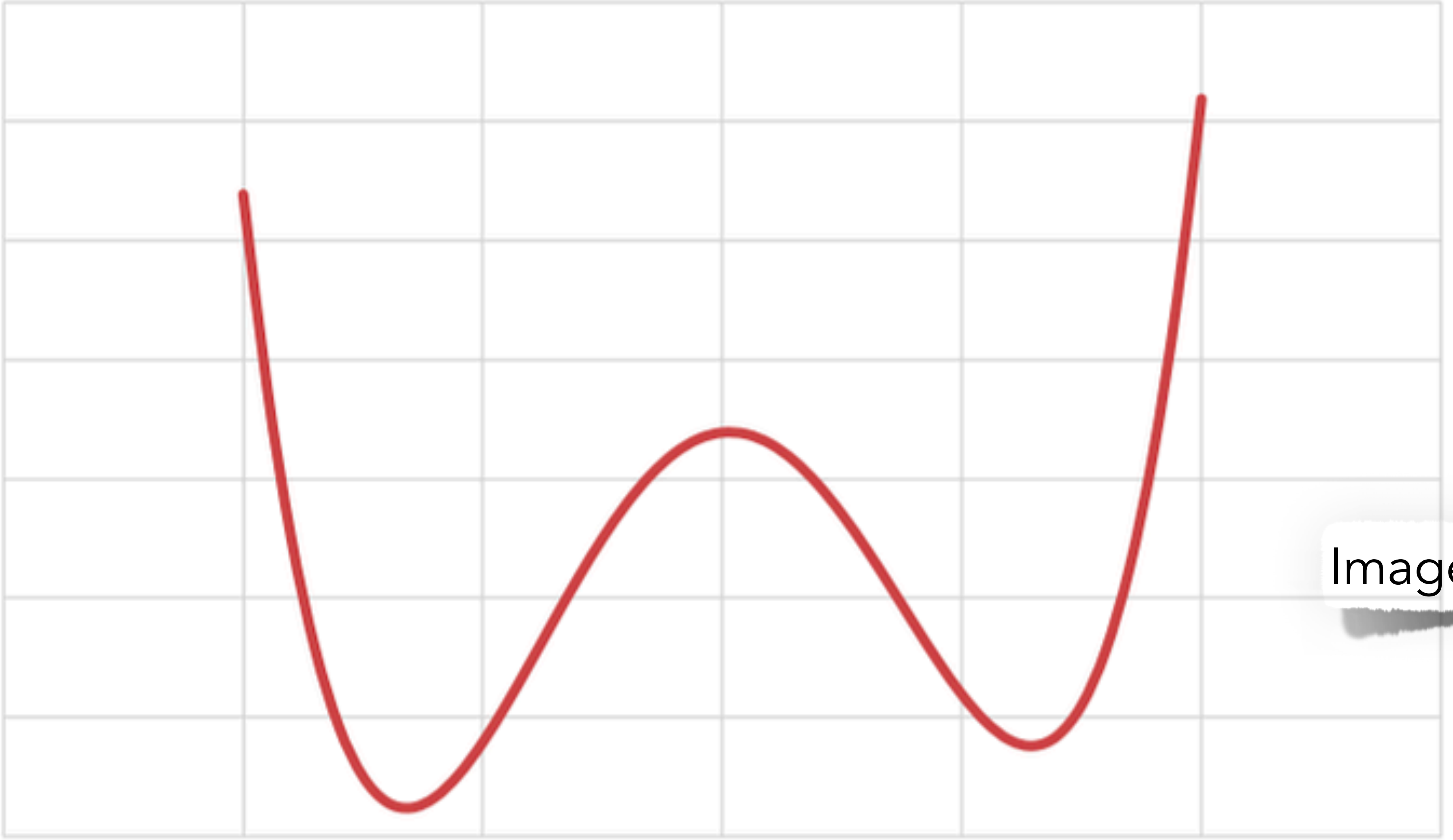


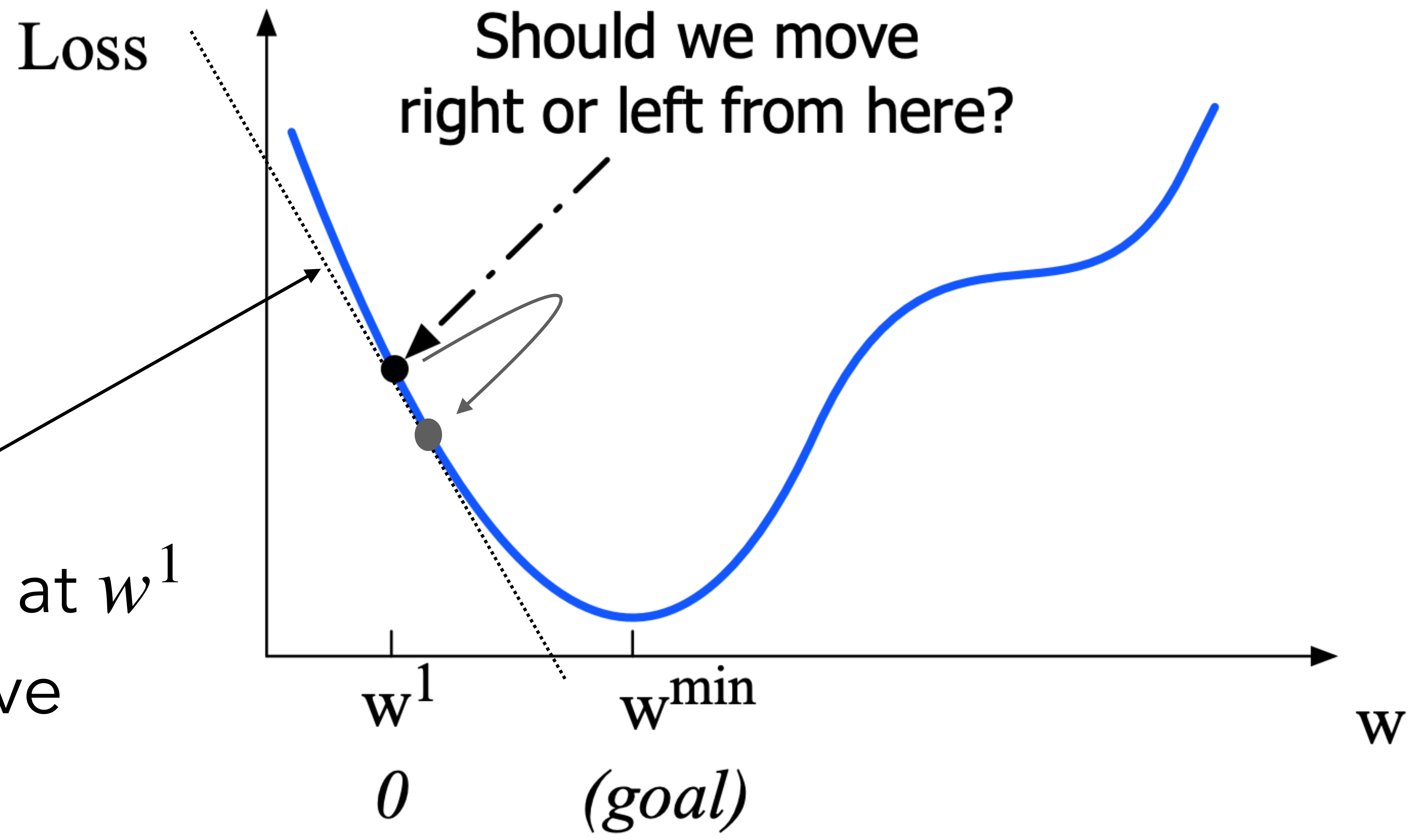
Image Credit: Medium

Non-convex function

Neural Networks - multiple alternatives

Consider: a single scalar w

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



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

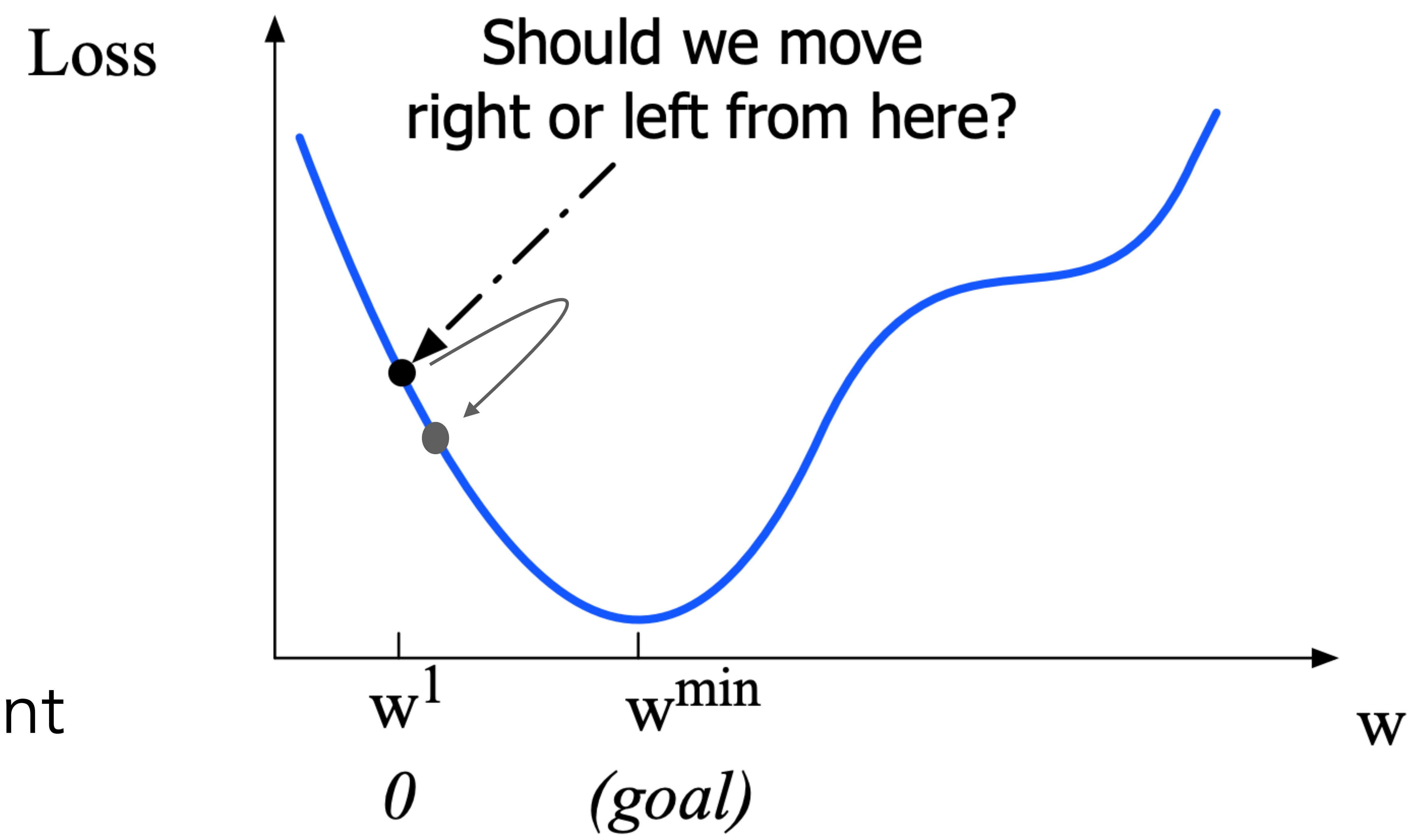
slope of loss at w^1 is negative

need to move positive

Gradients

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

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



Gradient Descent

But by how much?

Gradient Updates

- Move w by the value of the gradient $\frac{\partial}{\partial w} L(f(x; w), y^*)$, weighted by a learning rate η
- Higher learning rate means move w faster

η Too high: the learner will take big steps and overshoot

$$w_{t+1} = w_t - \eta \frac{\partial}{\partial w} L(f(x; w), y^*)$$

η Too low: the learner will take too long

If parameter θ is a vector of d dimensions:

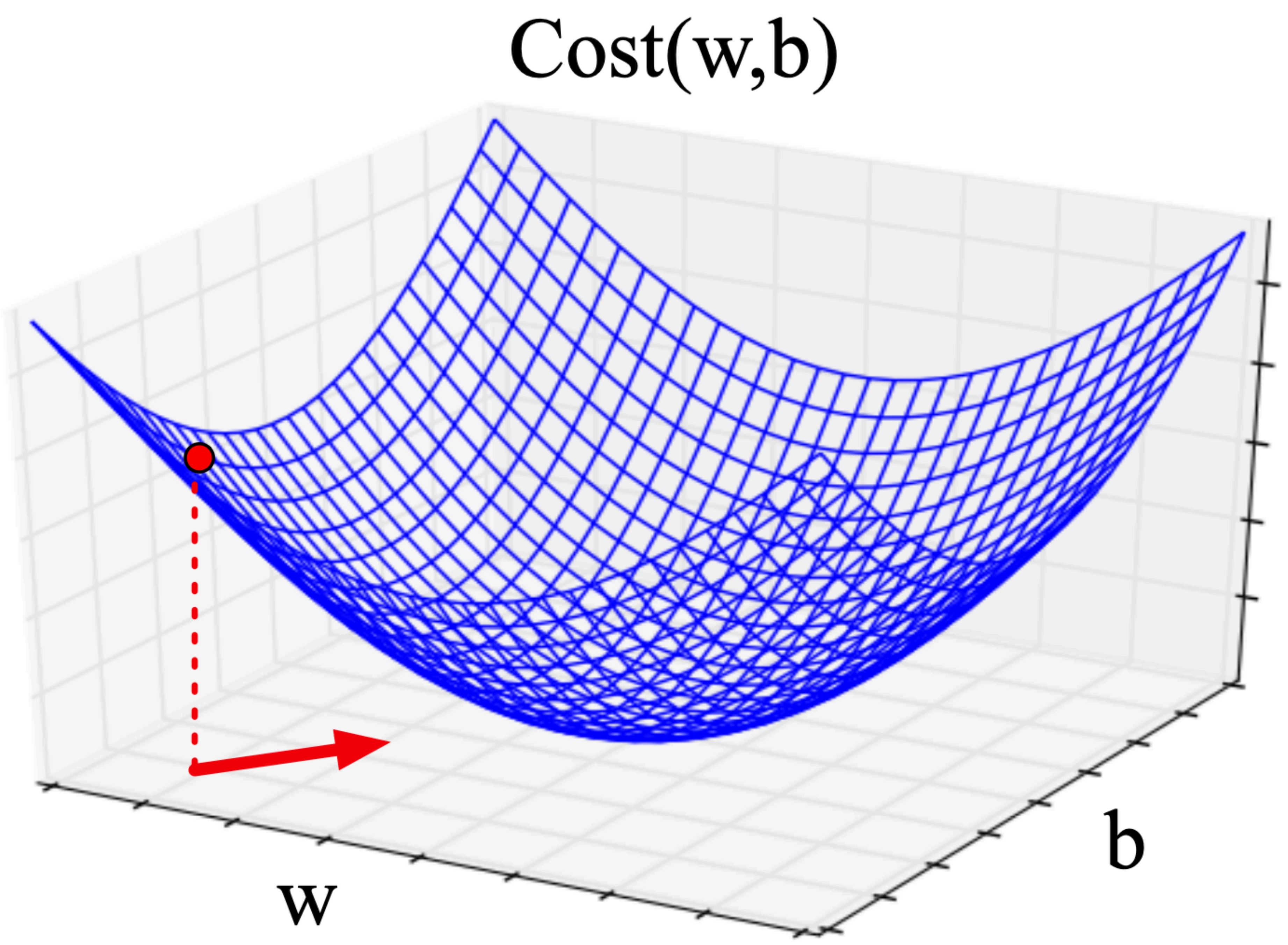
The gradient is just such a vector; it expresses the directional components of the sharpest slope along each of the d dimensions.

Under 2 dimensions

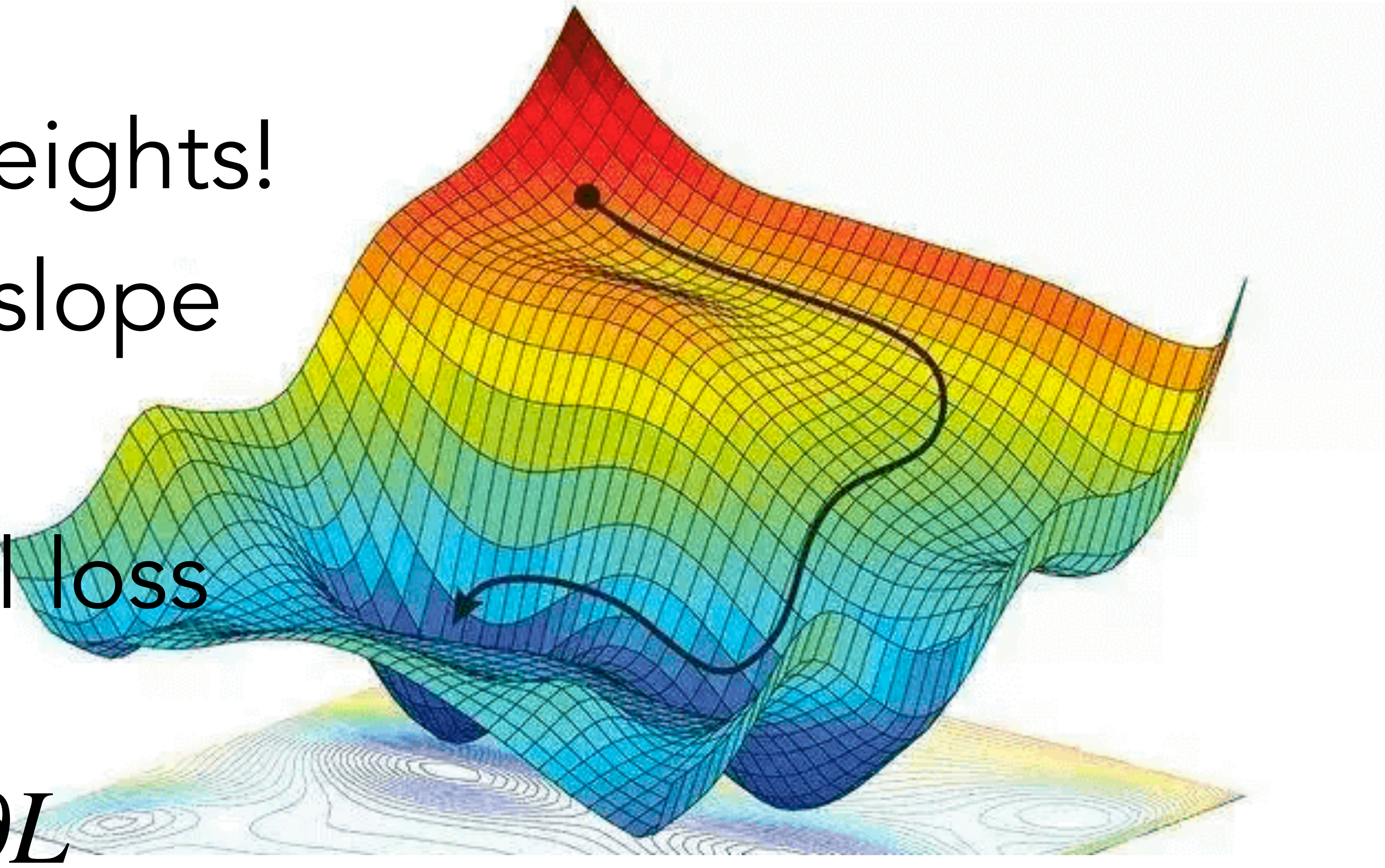
Consider 2 dimensions, w and b:

Visualizing the gradient vector at the red point

It has two dimensions shown in the $x - y$ plane



Real-life gradients, however…



- …are much longer; models usually contain lots and lots of weights!
- For each dimension θ_i the gradient component i tells us the slope with respect to that variable
  - "How much would a small change in θ_i influence the total loss function L?"
- We express the slope as a partial derivative $\frac{\partial}{\partial \theta_i}$ of the loss, $\frac{\partial L}{\partial \theta_i}$
  - The gradient is then defined as a vector of these partials

Real-life gradients

We will 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 at time step $t+1$ based on the gradient is thus:

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