Natural Language Processing
Text classification

Sofia Serrano
sofias6@cs.washington.edu

Credit to Yulia Tsvetkov and Noah Smith for slides

Announcements

https://courses.cs.washington.edu/courses/cse447/23wi/

- Quiz 1: Released as soon as lecture ends today (2:20pm)
  - 6 multiple-choice questions
  - Will be open for ~~12 hours~~ 24 hours, until 2:20pm Thursday 1/19
  - 10-min time limit once you start the quiz
  - Materials from weeks 1 and 2 (anything we talked about up through the end of class on Friday)
    - Introduction to NLP, introduction to text classification
    - Instructions for HW 1
  - We'll release the answers by the start of next week

Logistic regression

- Supervised text classification

- Rule-based
- Probabilistic

- Generative models
- Discriminative models

- Naïve Bayes
- Linear models
  - Multinomial logistic regression (aka MaxEnt)
- Non-linear models
  - Multilayer perceptron

Binary logistic regression

Logistic regression classifier

- Important analytic tool in natural and social sciences
- Baseline supervised machine learning tool for classification
- Is also the foundation of neural networks

Text classification

Input:

- a document d (e.g., a movie review)
- a fixed set of classes $C = \{c_1, c_2, \dots c_j\}$ (e.g., positive, negative, neutral)

Output

- a predicted class $\hat{y} \in C$

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 $\{x_1, x_2, \ldots, x_n\}$
  - We compute an output: a predicted class $\hat{y}^{(i)} \in \{0,1\}$

Features in logistic regression

- For feature $x_i \in \{x_1, x_2, \ldots, x_n\}$, weight $w_i \in \{w_1, w_2, \ldots, w_n\}$ tells us how important is x_i
    - x_i = "review contains 'awesome'": $w_i = +10$
    - x_j = "review contains horrible": $w_j = -10$
    - x_k = "review contains 'mediocre'": $w_k = -2$

Binary Logistic Regression for one observation x

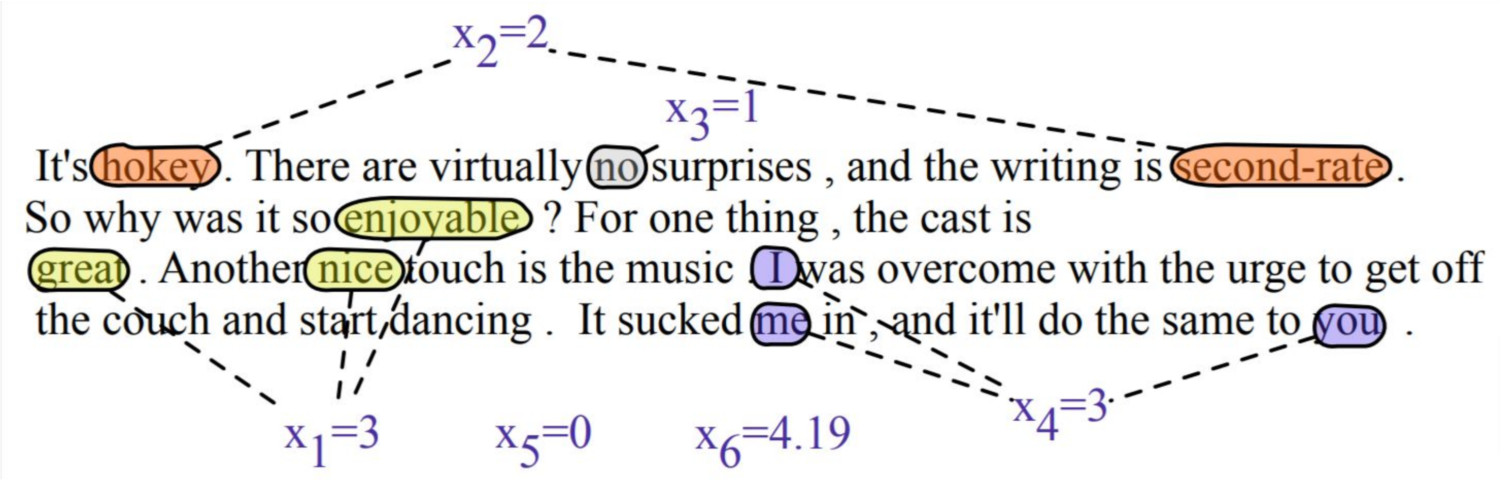
- Input observation: vector $x^{(i)} = \{x_1, x_2, \ldots, x_n\}$

- Weights: one per feature: $W = [w_1, w_2, \ldots, w_n]$
  - Sometimes we call the weights $\theta = [\theta_1, \theta_2, \ldots, \theta_n]$
  - We'll talk about how to extend this to multinomial logistic regression on Friday
    - *Hint: taking potential classes into account too*

- Output: a predicted class $\hat{y}^{(i)} \in \{0,1\}$

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 .



Var	Definition	Value
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	Value
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

How to do classification

- For each feature x_i, (the absolute value of) 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 ranges from $-\infty$ to ∞

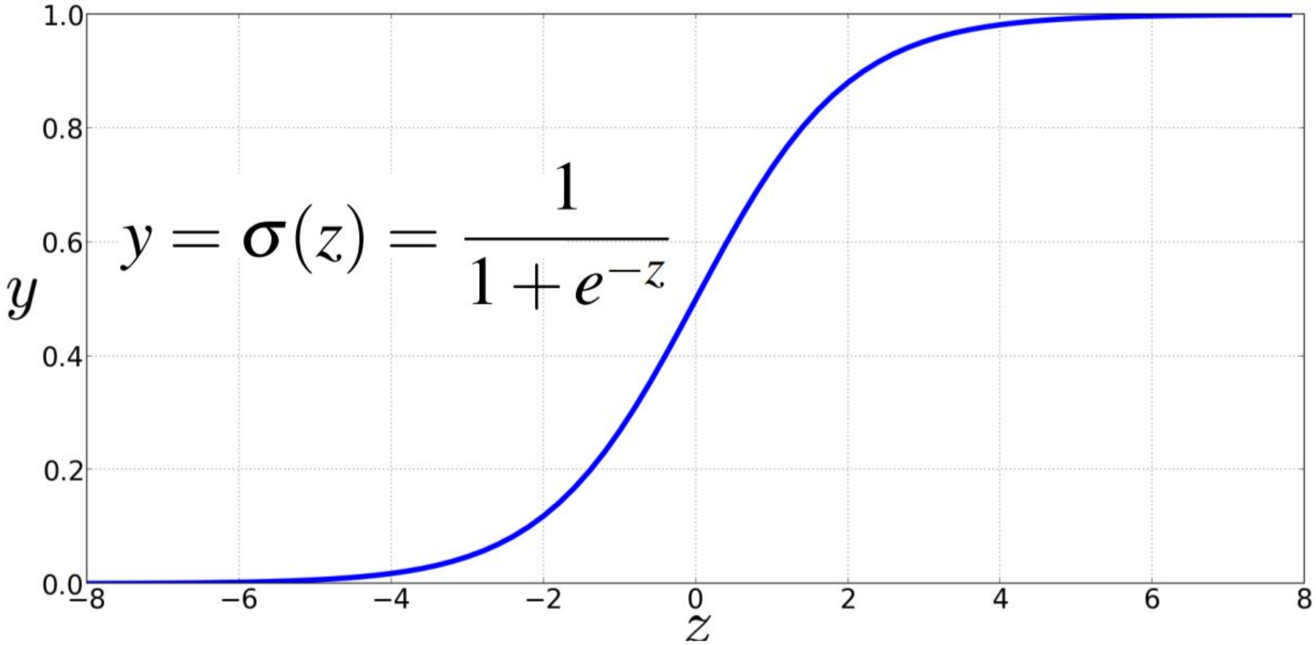
$$z = w \cdot x + b$$

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

"sigmoid" or
"logistic" function

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

The very useful sigmoid or logistic function



Idea of logistic regression

- We'll compute w·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

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

Making probabilities with sigmoids

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

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

By the way:

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

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

Because

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

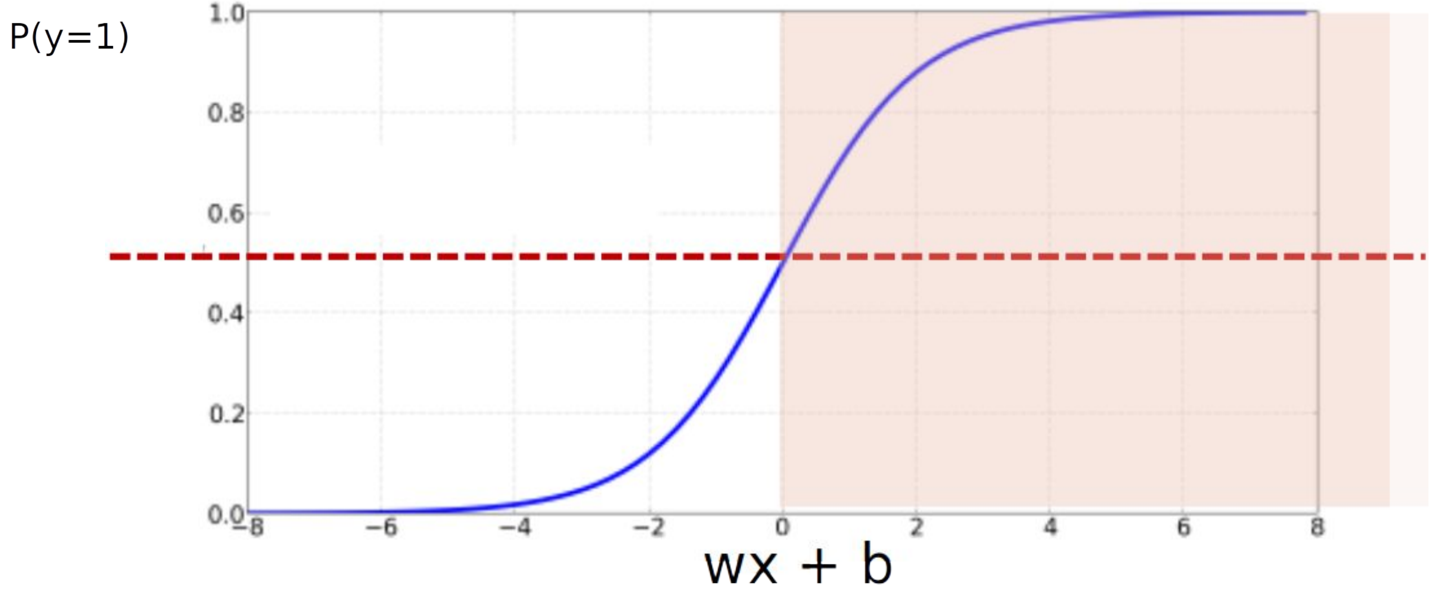
Turning a probability into a classifier

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

- 0.5 here is called the **decision boundary**

The probabilistic classifier

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

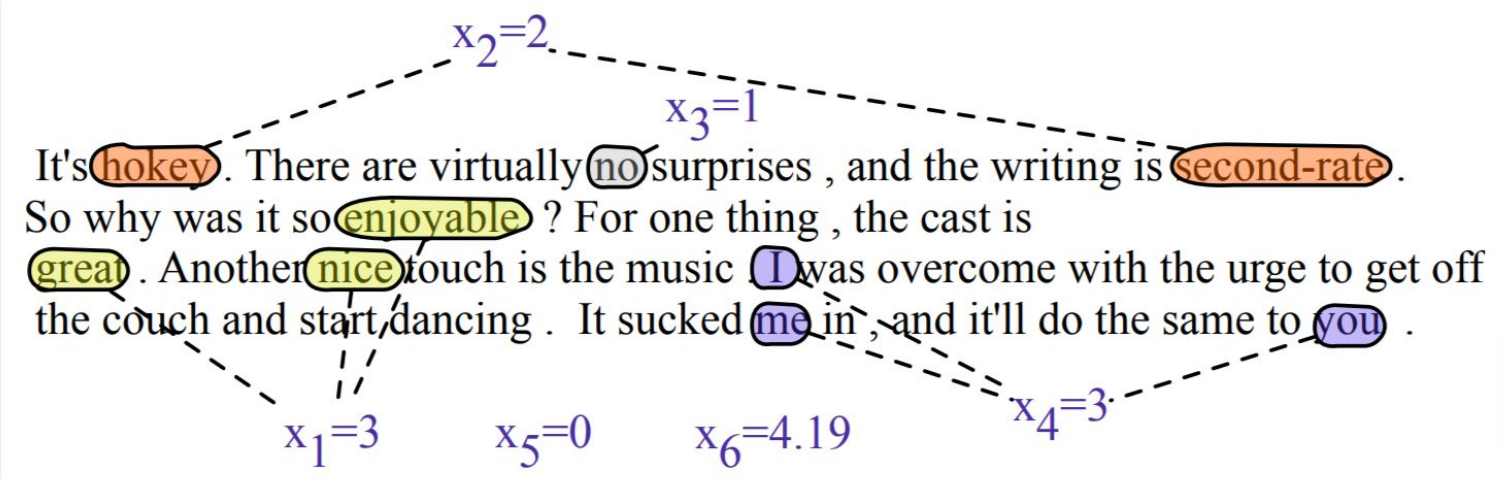


Turning a probability into a classifier

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

if $w \cdot x + b > 0$

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



Var	Definition	Value
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	Value
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]

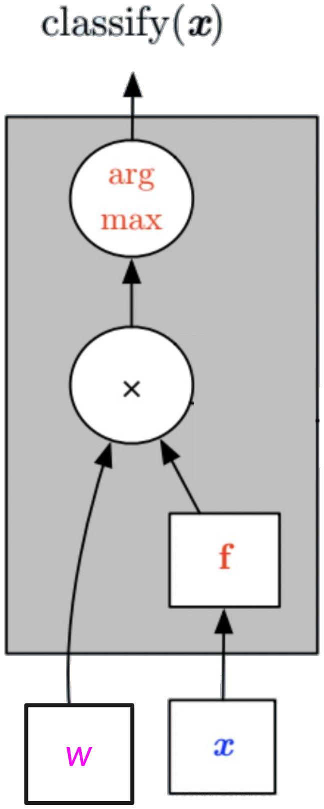
b = 0.1

Classifying sentiment for input x

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

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

A computation graph view of logistic regression



Wait, where did the W's come from?

- Supervised classification:
  - A training time we know the correct label y (either 0 or 1) for each x.
  - But what the system produces at inference time is an estimate $\hat{y}$

Wait, where did the W's come from?

- Supervised classification:
  - A training time we know the correct label y (either 0 or 1) for each x.
  - But what the system produces at inference time 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 an **optimization algorithm** to update w and b to minimize the loss
  - We need a distance estimator: a **loss function** or a cost function

Learning components in LR

An optimization algorithm:

- **stochastic gradient descent**

Gradient descent

Stochastic Gradient Descent

- Stochastic Gradient Descent algorithm
  - is used to optimize the weights
  - for logistic regression
  - also for neural networks
- We'll talk about the distinction between Stochastic Gradient Descent (SGD) and vanilla Gradient Descent (GD) tomorrow
  - *Hint: has to do with how often you adjust your function's weights*

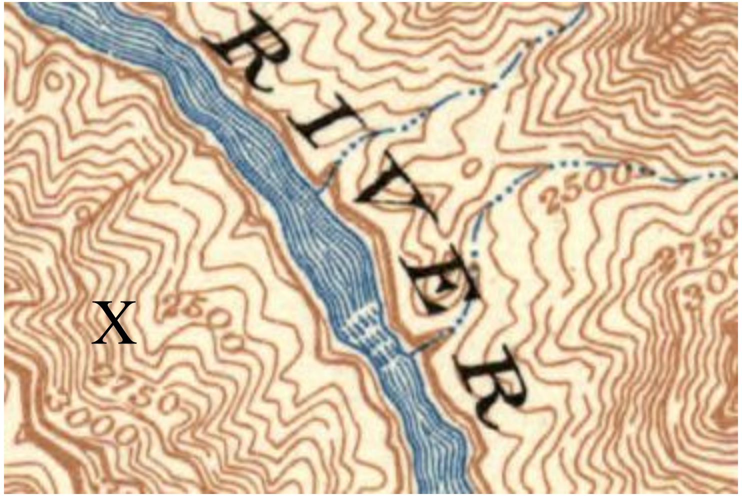
Intuition of gradient descent

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

Look around me 360∘

Find the direction of steepest slope up

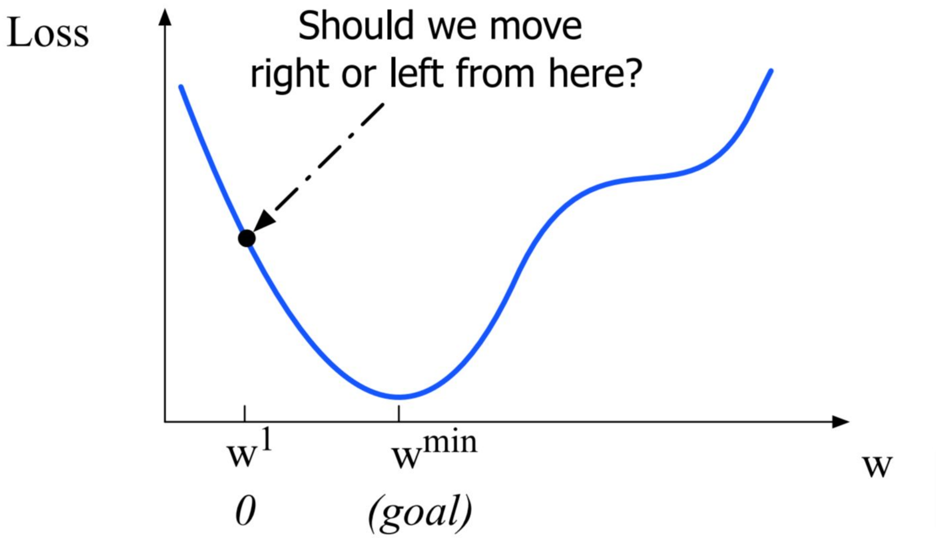
Go the opposite direction



Let's first visualize for a single scalar w

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

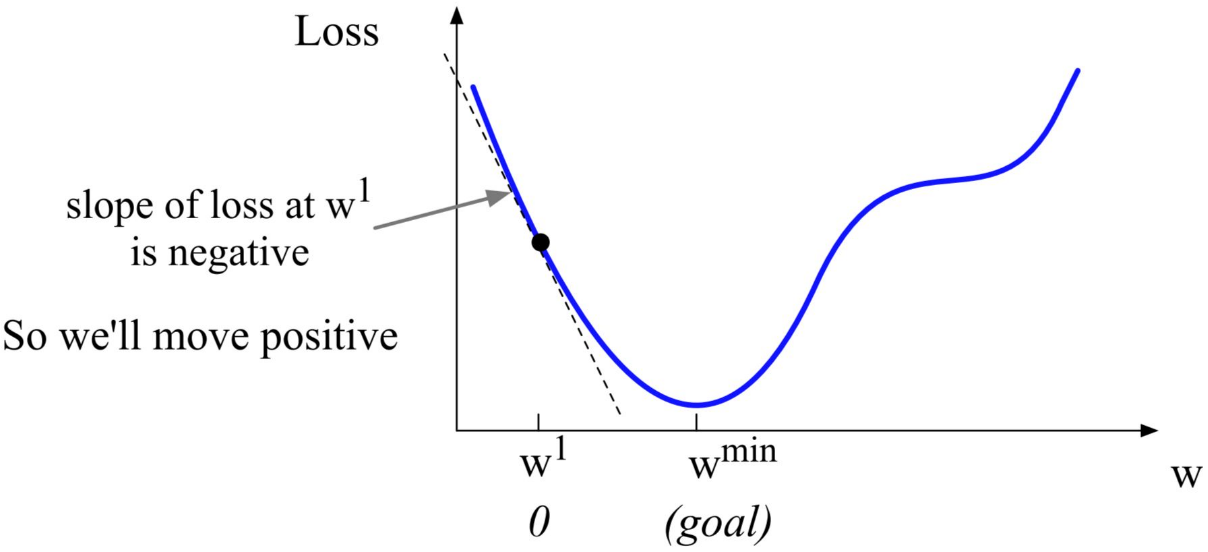
A: Move w in the reverse direction from the slope of the function



Let's first visualize for a single scalar w

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

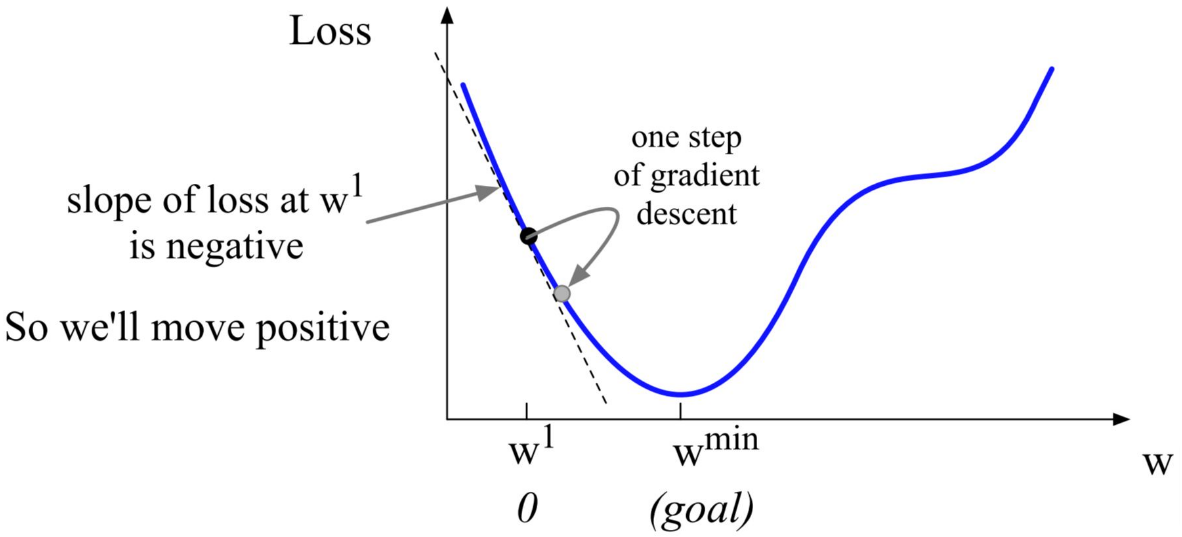
A: Move w in the reverse direction from the slope of the function



Let's first visualize for a single scalar w

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

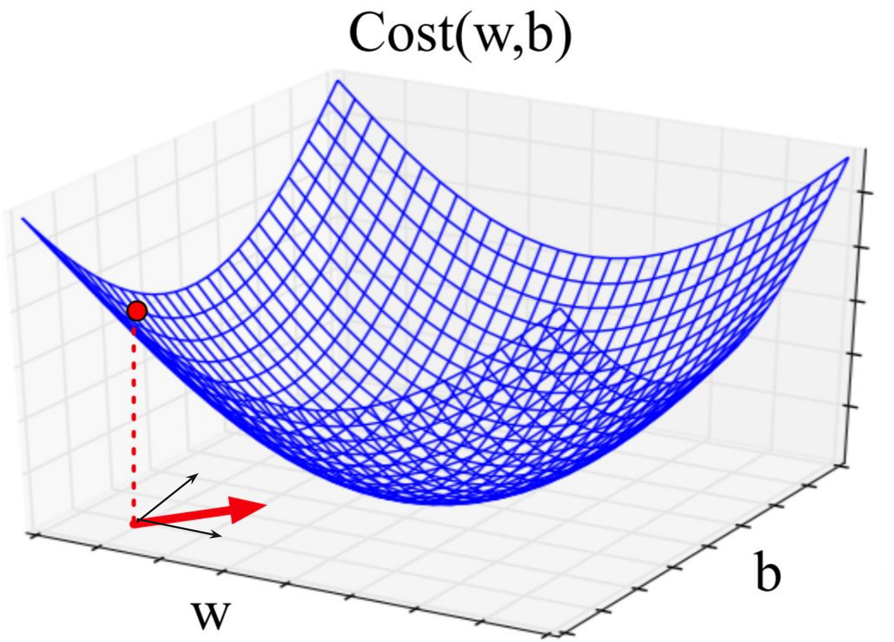
A: Move w in the reverse direction from the slope of the function



Now let's imagine 2 dimensions, w and b

Visualizing the (negative) gradient vector at the red point

It has two dimensions shown in the x-y plane



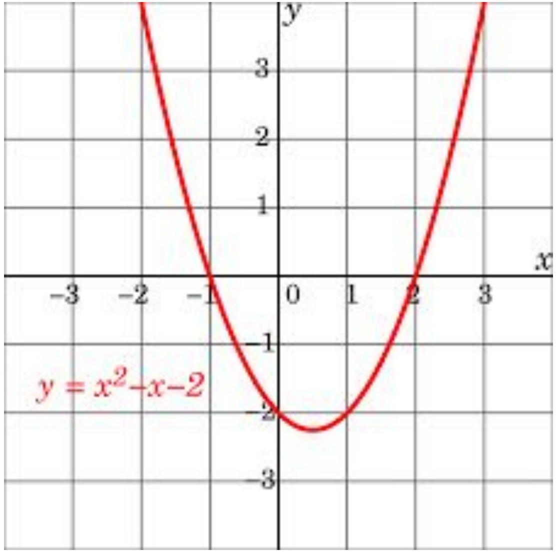
Now let's consider N dimensions

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" at your current point when you're considering all N dimensions at once.

What gives with highlighting the “current point” bit?

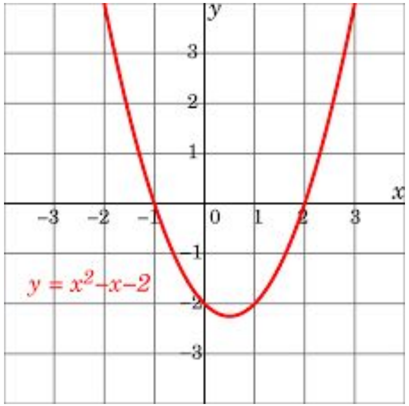
Think back to calculus I.



We’re used to finding a formula for the gradient (in this case, y’ = 2x - 1). Why am I not telling you to do that and *then* plug in the current point?

In some cases you *can*...

... like in that example,



or, as it turns out, for the loss function we'll construct for logistic regression!

Nice closed expression :))

… but in other cases, computing/storing that formula can get pretty awful.

Consider this (hypothetical) function:

$$w^T M_1(\boldsymbol{\sigma}(M_2 w) + M_3 w)$$

We begin full of optimism:

- Gradient of $M_2 w$ is M_2^T, of $M_3 w$ is M_3^T, of $M_1 z$ is M_1^T
- Gradient of $\boldsymbol{\sigma}(z)$ is $\boldsymbol{\sigma}(z)(1 - \boldsymbol{\sigma}(z))$

… and then we run into the chain rule combined with the product rule.

… but in other cases, computing/storing that formula can get pretty awful.

Chain rule: $$\frac{d}{dx}\left[f\Big(g(x)\Big)\right] = f'\Big(g(x)\Big)g'(x)$$

Product rule: $$\frac{d}{dx}[f(x) \cdot g(x)] = \frac{d}{dx}[f(x)] \cdot g(x) + f(x) \cdot \frac{d}{dx}[g(x)]$$

$$w^T M_1(\boldsymbol{\sigma}(M_2 w) + M_3 w)$$

d/dw (w^T $M_1(\boldsymbol{\sigma}(M_2 w) + M_3 w)$)

… which will involve d/dw(M_1 $(\boldsymbol{\sigma}(M_2 w) + M_3 w)$) …

… which will involve d/dw($\boldsymbol{\sigma}(M_2 w)$) …

Good news: we don't have to compute that formula!

(believe it or not) chain rule to the rescue!!

$$w^T M_1(\sigma(M_2 w) + M_3 w)$$

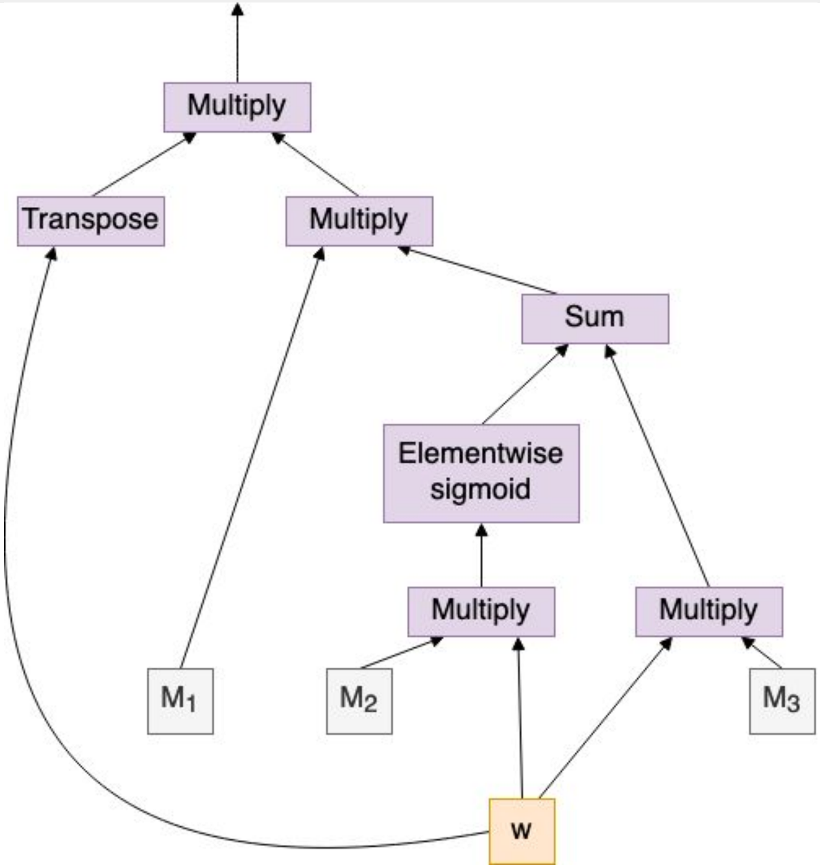
We can represent that function like this →

We want d/dw of that function.

Remember:

$$\frac{d}{dx}\left[f\Big(g(x)\Big)\right] = f'\Big(g(x)\Big)g'(x)$$

http://colah.github.io/posts/2015-08-Backprop/



The lesson:

If you have a function that is end-to-end differentiable,

you get the **same result** from backpropagating from the output back through a computation graph representation of that function

than you would by calculating the formula for that function’s gradient and plugging in the input you used to get that output.

- And since this formula calculation is often pretty horrible to compute/store, we generally compute gradients through backpropagation.

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.

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)

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 $\frac{\partial}{\partial w_i}$

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

Loss function: the distance between ŷ and y

We want to know how far is the classifier output $\hat{y} = \sigma(w \cdot x + b)$

from the true output: y [= either 0 or 1]

We'll call this difference: $L(\hat{y}, y)$ = how much ŷ differs from the true y

Our goal: minimize the loss

Let's make explicit that the loss function is parameterized by weights θ=(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:

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

$$L_{CE}(\hat{y}, y)$$

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 equation for updating θ based on the gradient is thus:

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

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

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

(Beginning to) construct our cross entropy loss

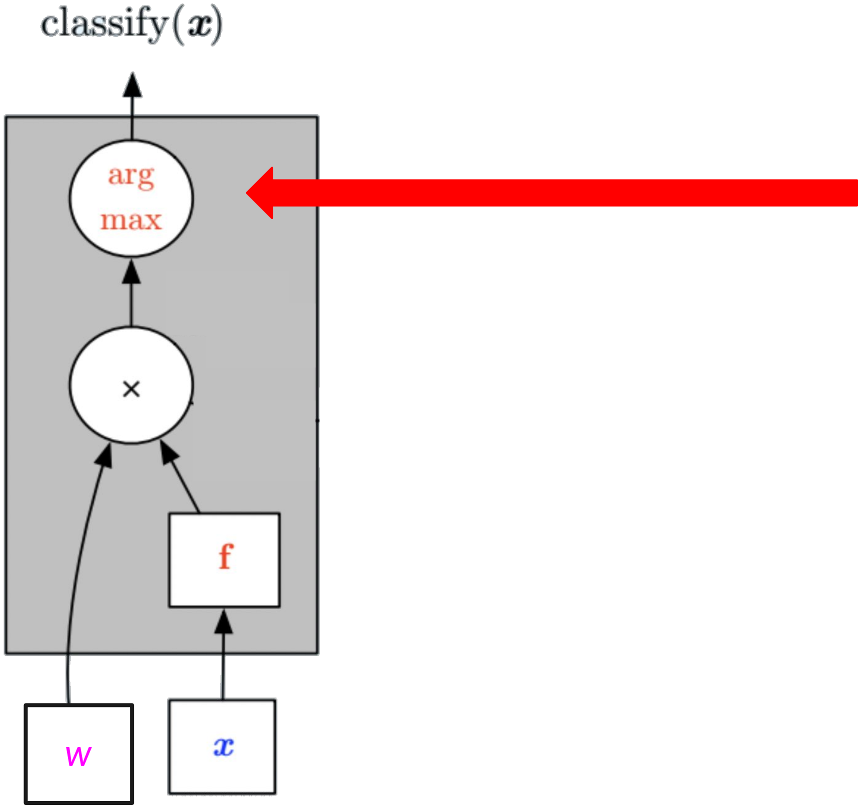
Intuition of negative log likelihood loss = cross-entropy loss

A case of conditional maximum likelihood estimation

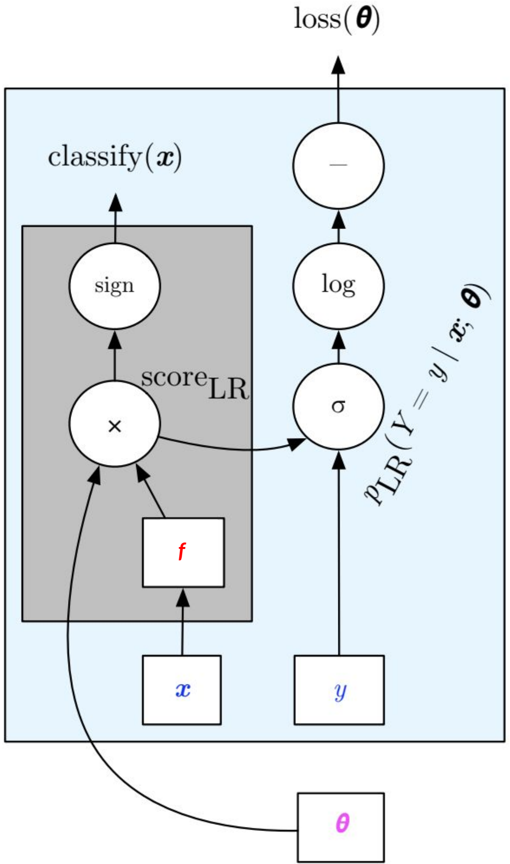
We choose the parameters w, b that maximize

- the log probability
- of the true y labels in the training data
- given the observations x

A computation graph view of logistic regression



Previewing the construction of our loss function



Next class:

- Deriving cross-entropy loss
- Moving from binary to multinomial logistic regression
- Tying up loose ends (picking a step size, regularization, etc.)