Carnegie Mellon University
Language
Technologies
Institute

11-411/11-611 Natural Language Processing

Logistic Regression

David R. Mortensen (after Dan Jurafsky)

September 20, 2022

Language Technologies Institute

Learning Objectives

At the end of this lecture, students will be able to:

- Explain the difference between generative and discriminative classifiers
- Describe the basic requirements of an ML classifier and how these are satisfied by logistic regression classifiers
- Implement classification with logistic regression works (be able to calculate it)
- Explain why the sigmoid function is used in logistic regression
- Define GRADIENT in the context of logistic regression and explain how it is calculated
- Explain at a high level how gradient descent works
- Work through an example of training a logistic regression classifier using gradient descent
- Define and diagnose overfitting
- Implement L2 and L1 regularization as a solution to overfitting in logistic regression

Generative and Discriminative Classifiers

1. **Generative**—builds a model of each of the categories, so it could **generate** instances of them
2. **Discriminative**—weights heavily the features that best discriminate between categories

An Example: Classifying Orchids

Cymbidium

Phalaenopsis

Features: terrestrial/epiphitic, stem shape (sympodial, monopodial, keikis), leaves (shape, pigments), roots (thickness, tuberous), flowers (dorsal sepal, other sepals, petals, labelum, pollinia)

- **Generative Classifier:** Builds of a model of each genus of orchids (*cymbidiums*, *phalaenopsis*, etc.) such that it could generate reasonable instances of each class (genus).
- **Discriminative Classifiers:** attach weights to each of the features according to their utility in distinguishing (discriminating), e.g., *cymbidiums* from *phalaenopsis*.

Finding the Correct Class c from a Document d

Naive Bayes

$$\hat{c} = \operatorname*{argmax}_{c \in C} P(d|c)P(c) \tag{1}$$

(compute the likelihood times the prior)

Logistic Regression

$$\hat{c} = \operatorname*{argmax}_{c \in C} P(c|d) \tag{2}$$

(compute the posterior directly)

What Goes into an ML Classifier?

1. A feature representation
2. A classification function
3. An objective function
4. An algorithm for optimizing the objective function

What Goes into Logistic Regression?

GENERAL	IN LOGISTIC REGRESSION
feature representation	represent each observation $x^{(i)}$ as a **vector of features** $[x_1, x_2, \ldots x_n]$
classification function	**sigmoid function** (logistic function)
objective function	**cross-entropy loss**
optimization function	**stochastic gradient descent**

The Two Phases of Logistic Regression

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

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

Classification with Logistic Regression

Features in Logistic Regression

For feature x_i, weight w_i tells us how import x_i is

- $x_i =$ "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 feature vector $x = [x_1, x_2, \ldots, x_n]$

weights, one per feature $W = [w_1, w_2, \ldots, w_n]$ (which can also be called $\Theta = [\theta_1, \theta_2, \ldots, \theta_n]$

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

How to Do Classification

For each feature x_i, weight w_i tells us the importance of x_i (and we also have the 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

What does “sum is high” even mean?

Can’t our classifier be like Naive Bayes and give us a probability?

What we really want:

- $p(y = 1|x; \theta)$
- $p(y = 0|x; \theta)$

The Problem: z Isn't a Probability!

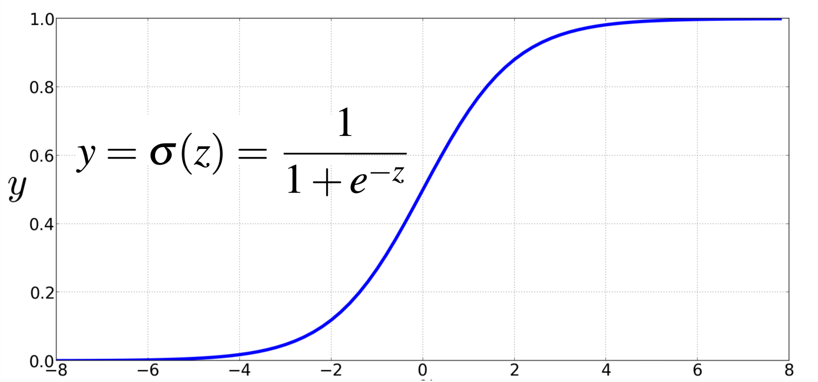
z is just a number:

$$z = w \cdot x + b \tag{3}$$

Solution: use a function of z that goes from 0 to 1, like the **logistic function** or **sigmoid function**:

$$\hat{y} = \sigma(z) = \frac{1}{1 + e^{-z}} = \frac{1}{1 + \exp(-z)} \tag{4}$$

The Sigmoid Function



Logistic Regression in Three Easy Steps

1. Compute $w \cdot x + b$
2. Pass it through the sigmoid function: $\sigma(w \cdot x + b)$
3. Treat the result as a probability

Making Probabilities with Sigmoids

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

Wait, what?

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

because

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

a very convenient property!

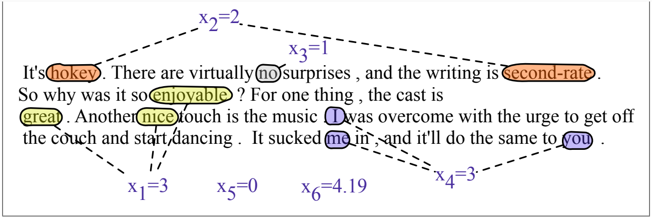
From Probability to Classification

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

0.5 here is called the **decision boundary**

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 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	$\log(\text{word count of doc})$	$\ln(66) = 4.19$

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

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

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

Learning to Classify with Logistic Regression

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, $\hat{y}$

Our Goal: Minimize the Loss

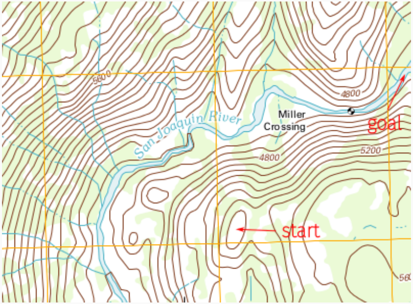
Let's make it explicit that the loss function is parameterized by weights $\theta = (w, b)$.

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

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

$$\hat{\theta} = \operatorname*{argmin}_{\theta} \frac{1}{m} \sum_{i=1}^{m} L_{CE}(f(x^{(i)}; \theta), y^{(i)}) \tag{5}$$

The Intuition of Gradient Descent



- You are on a hill
- It is your mission to reach the river at the bottom of the canyon (as quickly as possible)
- What is your strategy?
    1. Determine in which direction the steepest downhill slope lies
    2. Take a step in that direction
    3. Repeat until a step in any direction will take you up hill

Our Goal: Minimize the Loss

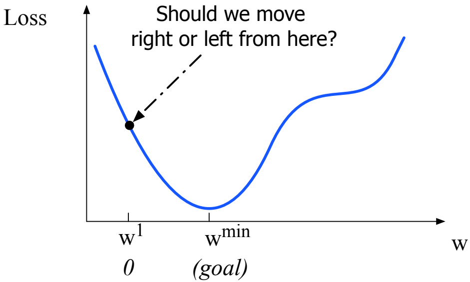
For logistic regression, the loss function is convex

- Just one minimum
- Gradient descent is guaranteed to find the minimum, no matter where you start

(Loss for neural networks is non-convex; we'll talk more about that later)

Let's first visualize for a single scalar w

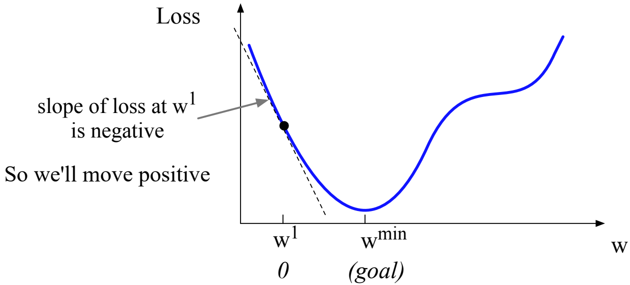
Q: Given current w, should we make it bigger or smaller?
A: Move *w* in the reverse direction from the slope of the function



Let's first visualize for a single scalar w

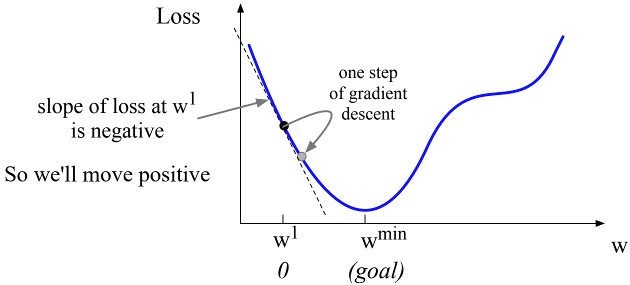
Q: Given current w, should we make it bigger or smaller?

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



A Gradient is a Vector Pointing in the Direction of Greatest Increase

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 a Step?

- We move by the value of the gradient (in our example, the slope)

$$\frac{d}{dw}L(f(x;w),y)$$

  weighted by the LEARNING RATE η
- The higher the learning rate, the faster w moves:

$$w^{t+1} = w^t - \eta\frac{d}{dw}L(f(x;w),y) \tag{6}$$

How Do We Do Gradient Descent in N Dimensions?

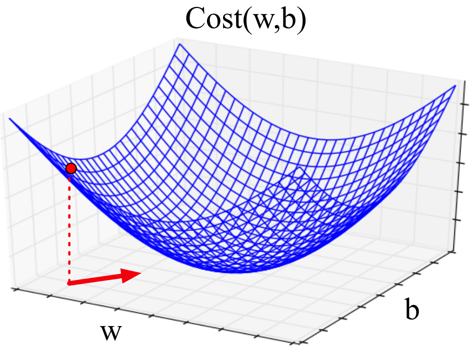
We want to know where in the N-dimensional space (of the N parameters that make up θ) we should move.

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



But Real Gradients Have More than Two Dimensions

- They are much longer
- They have lots of weights
- For each dimension w_i, the gradient component i tells us the slope w.r.t. that variable
  - "How much would a small change in w_i influence the total loss function L?"
  - The slope is expressed as the partial derivative ∂ of the loss ∂w_i
- We can then define the gradient as **a vector of these partials**

Computing the Gradient

Let's represent $\hat{y}$ as $f(x;\theta)$ to make things more clear:

$$\nabla_\theta L(f(x;\theta),y) = \begin{bmatrix} \frac{\partial}{\partial w_1} L(f(x;\theta),y) \\ \frac{\partial}{\partial w_2} L(f(x;\theta),y) \\ \vdots \\ \frac{\partial}{\partial w_n} L(f(x;\theta),y) \end{bmatrix} \tag{7}$$

What is the final equation for updating θ based on the gradient?

$$\theta_{t+1} = \theta_t - \eta \nabla L(f(x;\theta),y) \tag{8}$$

So What Are These Partial Derivatives for Logistic Regression?

The textbook lays out the derivation in §5.8, but here's the basic idea:

Here is the loss function:

$$L_{CE}(\hat{y}, y) = -[y \log \sigma(w \cdot x + b) + (1 - y) \log(1 - \sigma(w \cdot x + b))] \tag{9}$$

The derivative of this function is:

$$\frac{\partial L_{CE}(\hat{y}, y)}{\partial w_j} = [\sigma(w \cdot x + b) - y]x_j \tag{10}$$

which looks very manageable!

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 $x^{(1)}, x^{(2)}, ..., x^{(m)}$
    #   y is the set of training outputs (labels) $y^{(1)}, y^{(2)}, ..., y^{(m)}$

$\theta \leftarrow 0$

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

A Sidenote: Hyperparameters

The learning rate (our η) is a **hyperparameter**, a term you will hear a lot

- Set it too high? The learner will catapult itself across the minimum and may not converge
- Set it too low? The learner will take a long time to get to the minimum, and may not converge in our lifetime

But what are hyperparameters?

- Hyperparameters are parameters in a machine learning model that are not learned empirically
- They have to be set by the human who is designing the algorithm

Now let’s work through an example of gradient descent borrowed directly from the textbook authors.

Example of gradient descent

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

Update step for update θ is:

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

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

Gradient vector has 3 dimensions:

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

Example of gradient descent

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

Update step for update θ is:

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

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

Gradient vector has 3 dimensions:

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

Example of gradient descent

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

Update step for update θ is:

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

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

Gradient vector has 3 dimensions:

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

Example of gradient descent

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

Update step for update θ is:

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

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

Gradient vector has 3 dimensions:

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

Example of gradient descent

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

Update step for update θ is:

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

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

Gradient vector has 3 dimensions:

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

Example of gradient descent

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

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:

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

$\eta = 0.1$;

$$\theta^1 =$$

Example of gradient descent

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

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:

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

$\eta = 0.1;$

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

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:

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

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

Example of gradient descent

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

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:

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

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

Note that enough negative examples would eventually make w_2 negative

Mini-batching

- In stochastic gradient descent, the algorithm chooses one random example at each iteration
- The result? Sometimes movements are choppy and abrupt
- In practice, instead, we usually compute the gradient over **batches** of training instances
- Entire dataset: BATCH TRAINING
- *m* examples (e.g., 512 or 1024): MINI-BATCH TRAINING

Regularization

Don't Fight Yesterday's War

In fact a 4-gram model trained on small data can and will memorize the data, achieving perfect accuracy on the training set. Sounds great, right?

But what happens when it encounters 4-grams that were not in the training set? It will get low accuracy on the test set. The combination of a too-powerful model and small data can be disastrous.

This is called overfitting

OVERFITTING is when the model fits the details of the training set **so exactly** that it cannot generalize to a test set. How to avoid overfitting?

- REGULARIZATION (logistic regression)
- DROPOUT (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)

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

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 θ to the origin.

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

L2 regularized objective function:

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

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:

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

Questions?