

CS 1671 / CS 2071 / ISSP 2071

Human Language Technologies

Session 10: Logistic regression part 2

Michael Miller Yoder

February 16, 2026



University of
Pittsburgh

School of Computing and Information

Course logistics: project

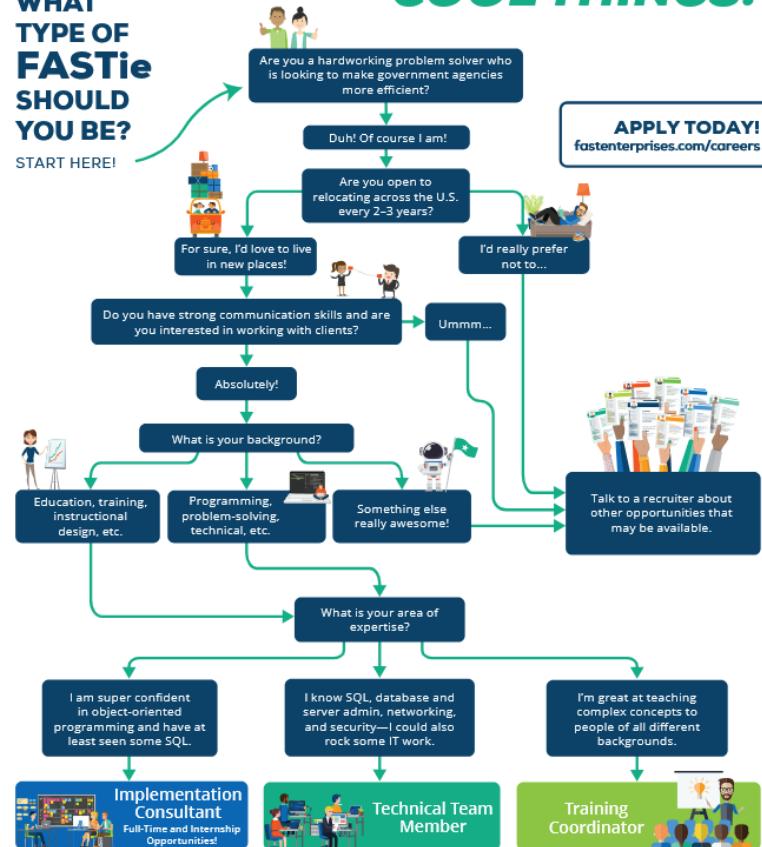
- Michael will send some additional tips and info about your project
 - Load in and examine your dataset, or look for suitable datasets if you don't have any
 - Think about what exactly the input and output for your task will be
- Project proposal due **next Thu Feb 27** is the next deliverable

WE KNOW, GOVERNMENT SOFTWARE SOUNDS BORING BUT REALLY, **WE DO COOL THINGS!**

- FAST Enterprises at Career Fair Day 2: Computing, Information, and Analytics
- Feb 18 from 11:00am – 3:00pm in the William Pitt Union

WHAT TYPE OF FASTie SHOULD YOU BE?

START HERE!



Lecture overview: logistic regression part 2

- Learning the weights for features in logistic regression
- Cross-entropy loss function
- Stochastic gradient descent
- Batch and mini-batch training

Review: classification with logistic regression

1. What is the necessary format for the input to logistic regression? What will the output format be?
2. What is the equation for calculating \hat{y} , the predicted class from an input vector x ?

Logistic regression: learning the weights

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:

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

We'll call this difference:

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

Cross-entropy loss for binary classification

Cross-entropy between Bernoulli distributions of the predicted, where \hat{y} is the predicted label and y is the true label

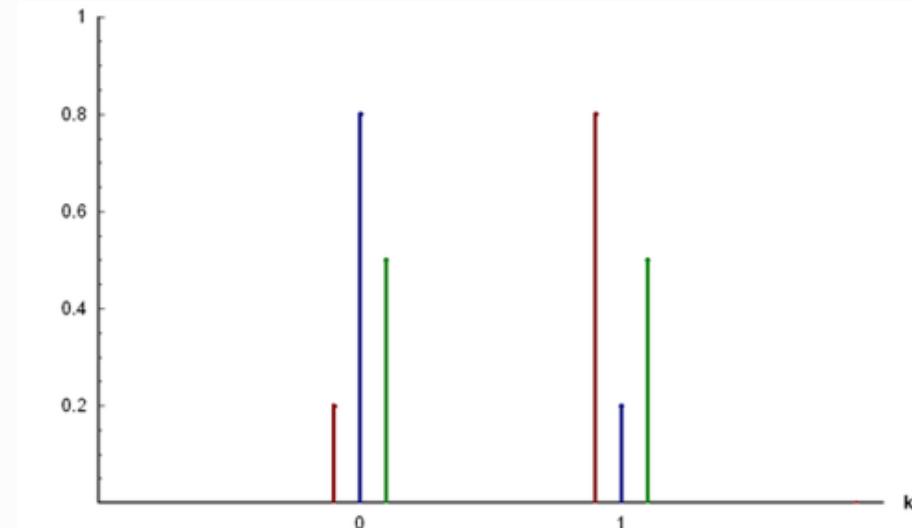
Minimize: $L_{\text{CE}}(\hat{y}, y) = -[y \log \hat{y} + (1 - y) \log(1 - \hat{y})]$



Claude Shannon

Cross-entropy loss for binary classification

- Cross-entropy loss: measure of distance between true distribution and predicted probability distribution of labels
- Logistic regression predicts $p(y=0)$ and $p(y=1)$ in a Bernoulli distribution. The true labels can also be considered a Bernoulli distribution over possible labels. If $y=1$, $p(y=1) = 1$ and $p(y=0) = 0$.



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?

$$\begin{aligned} P(+|X) &= P(Y = 1|X) \\ &= \sigma(w \cdot x + b) = \sigma(\sum_{i=1}^n w_i x_i + b) \\ &= \sigma((2.5*3) + (-5.0*2) + (-1.2*1) + (0.5*3) + (2.0*0) + (0.7*4.19) + b) \\ &= \sigma(0.733 + 0.1) \\ &= \sigma(0.833) = 0.7 \end{aligned}$$

Pretty well! What's the loss?

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

Let's see if this works for our sentiment example

Suppose true value instead was $y=0$.

$$\begin{aligned} p(y=0|x) &= 1 - p(y=1|x) \\ &= 1 - 0.7 \\ &= 0.3 \end{aligned}$$

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 in the case where the model was wrong!

Stochastic gradient descent

Our Goal: Minimize the Loss

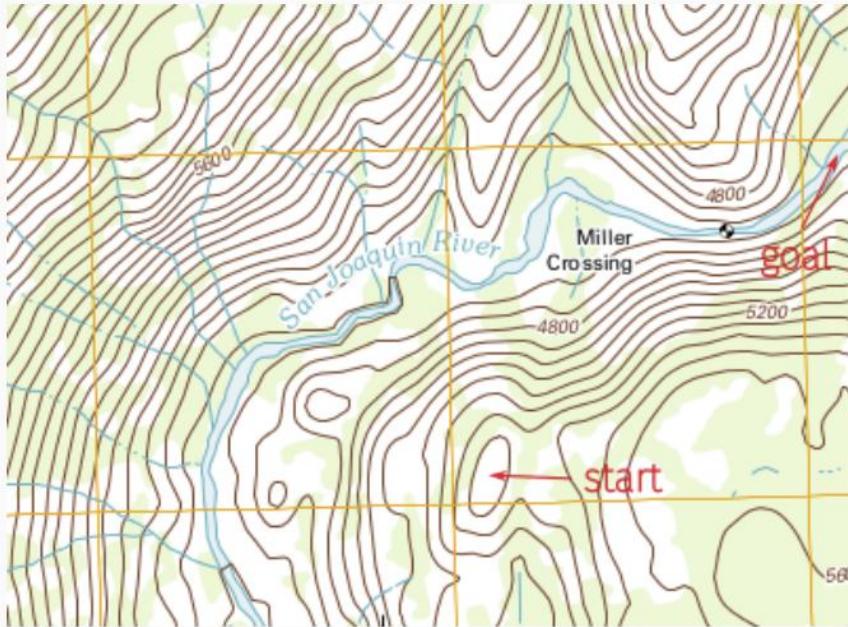
Let's make it explicit that the loss function is parameterized by weights $\theta = (w, b)$.

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 (L_{CE}), averaged over all examples:

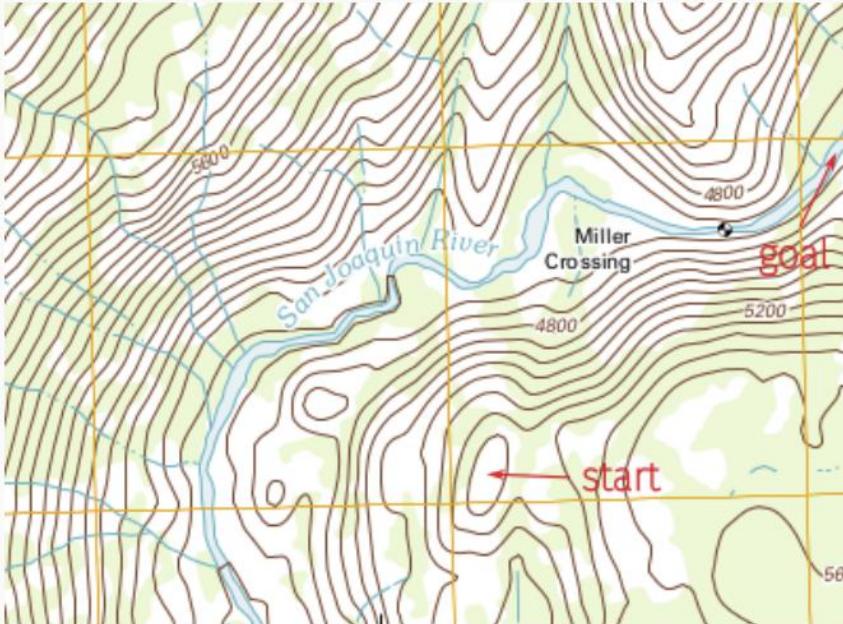
$$\hat{\theta} = \operatorname{argmin}_{\theta} \frac{1}{m} \sum_{i=1}^m L_{CE}(f(x^{(i)}; \theta), y^{(i)})$$

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?

The Intuition of Gradient Descent

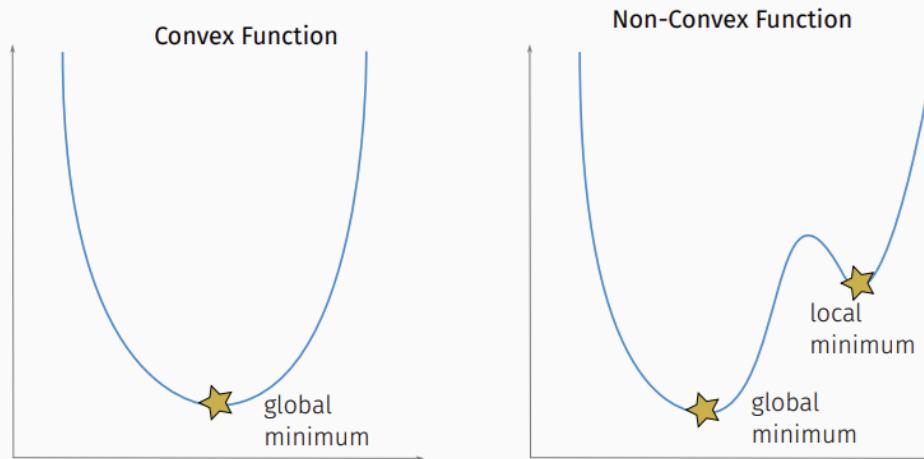


- You are on a hill
- It is your mission to reach the river at the bottom of the canyon (as quickly as possible)
- What is your strategy?
 1. Determine in which direction the steepest downhill slope lies
 2. Take a step in that direction
 3. Repeat until a step in any direction will take you up hill

Our Goal: Minimize the Loss

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

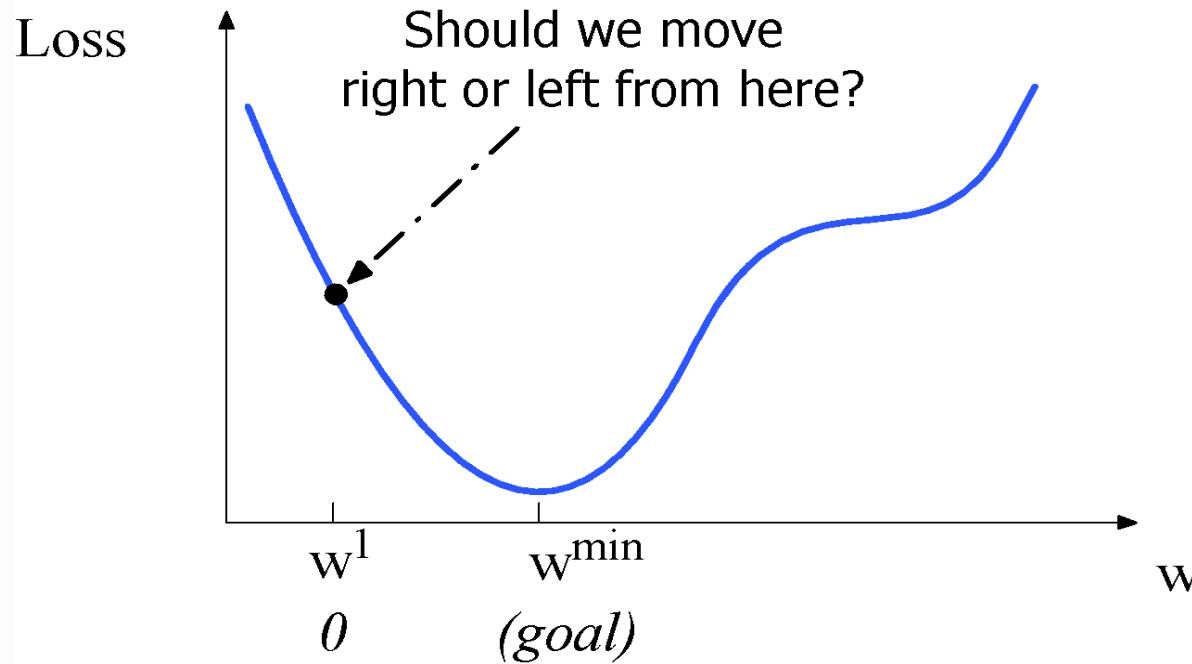
- Just one minimum
- Gradient descent is guaranteed to find the minimum, no matter where you start



Let's first visualize for a single scalar w

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

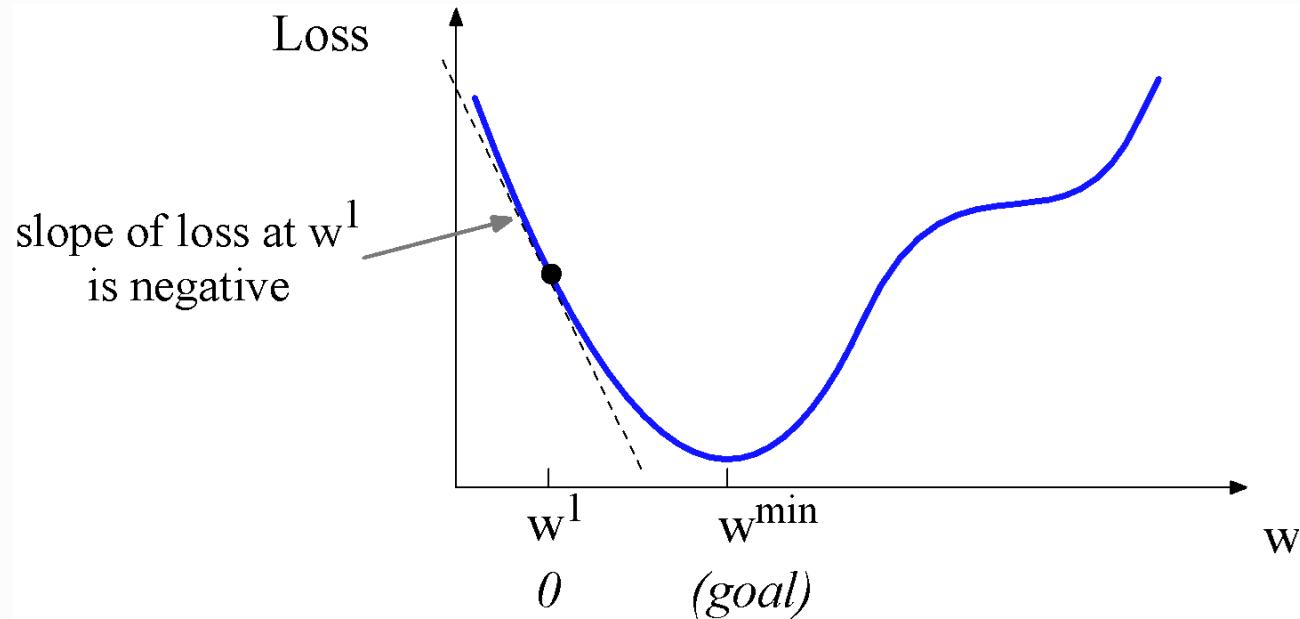
A: Move w in the reverse direction from the slope of the function



Let's first visualize for a single scalar w

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

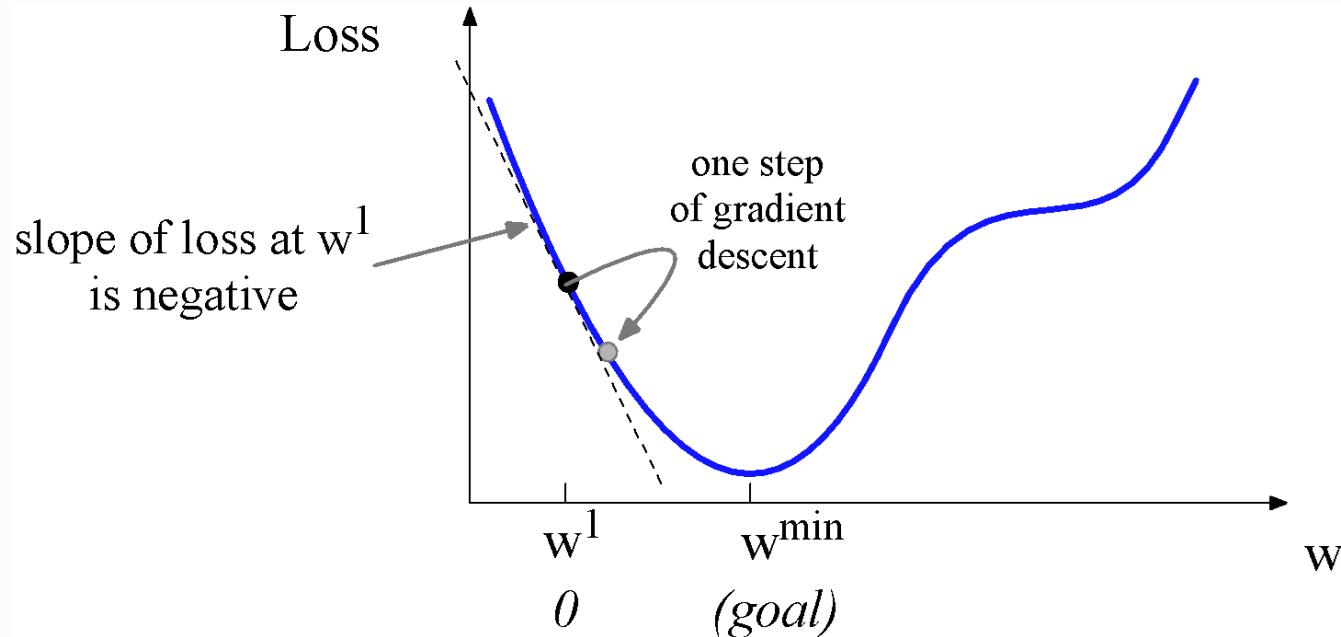
A: Move w in the reverse direction from the slope of the function



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



So we'll move
positive (to the right)

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)

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

weighted by the LEARNING RATE η

- The higher the learning rate, the faster w changes:

$$w_{t+1} = w_t - \eta \frac{d}{dw} L_{CE}(f(x; w), y)$$

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.

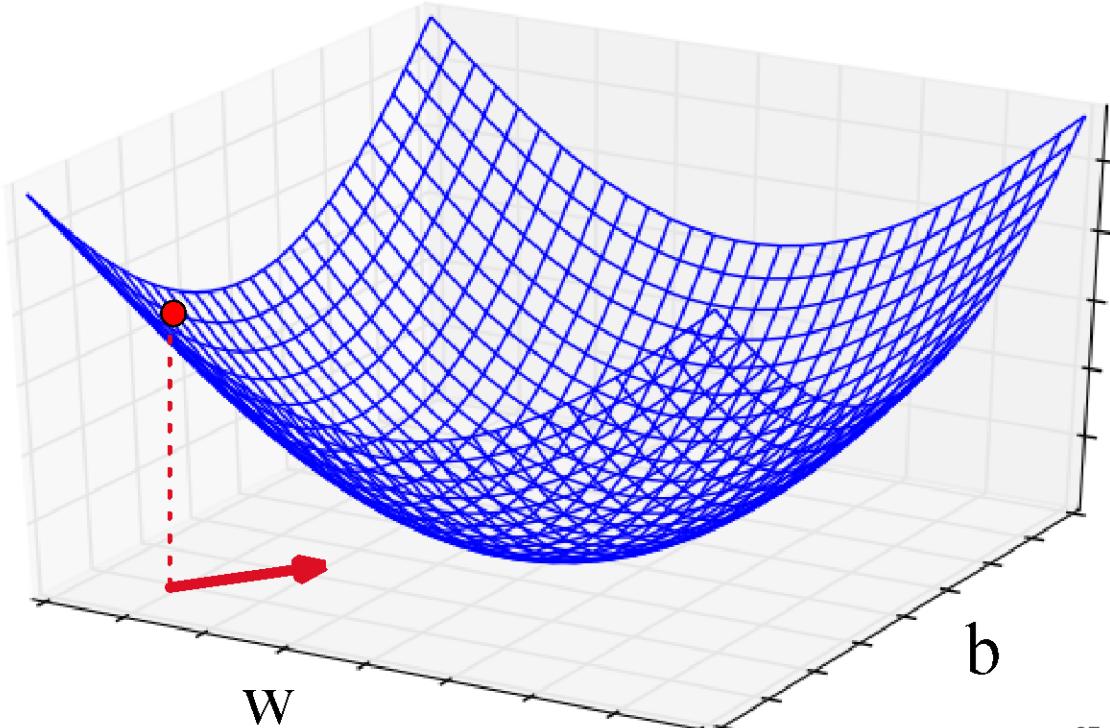
The **gradient is just such a vector**; it expresses the directional components of the sharpest slope along each of the N dimensions.

Imagine 2 dimensions, w and b

Visualizing the gradient vector at the red point

It has two dimensions shown in the x-y plane

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



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 $f(\mathbf{x}; \theta)$ to make things clearer:

$$\nabla_{\theta} L(f(\mathbf{x}; \theta), y) = \begin{bmatrix} \frac{\partial}{\partial w_0} L(f(\mathbf{x}; \theta), y) \\ \frac{\partial}{\partial w_1} L(f(\mathbf{x}; \theta), y) \\ \frac{\partial}{\partial w_2} L(f(\mathbf{x}; \theta), y) \\ \vdots \\ \frac{\partial}{\partial w_n} L(f(\mathbf{x}; \theta), y) \end{bmatrix}$$

Note that, since we are representing the bias b as w_0 , θ is more-or-less equivalent to \mathbf{w} .

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

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

(For us, L is the cross-entropy loss L_{CE}).

So What Are These Partial Derivatives Used in Logistic Regression?

The textbook lays out the derivation in §4.15 but here's the basic idea:

Here is the cross-entropy loss function (for binary classification):

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

The derivative of this function is:

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

which is 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 keep hearing

- **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 again?

- 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

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:

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

$$\begin{aligned} w_1 &= w_2 = b = 0; \\ x_1 &= 3; \quad x_2 = 2; \\ y &= 1 \end{aligned}$$

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:

$$\nabla_{w,b} = \left[\begin{array}{c} \frac{\partial L_{\text{CE}}(\hat{y}, y)}{\partial w_1} \\ \frac{\partial L_{\text{CE}}(\hat{y}, y)}{\partial w_2} \\ \frac{\partial L_{\text{CE}}(\hat{y}, y)}{\partial b} \end{array} \right] = \left[\begin{array}{c} \text{?} \\ \text{?} \\ \text{?} \end{array} \right]$$

Example of gradient descent

Update step for update θ is:

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

$$w_1 = w_2 = b = 0; \\ x_1 = 3; \quad x_2 = 2; \\ y = 1$$

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

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:

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

$$\theta^1 =$$

Example of gradient descent

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:

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

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

Batch and mini-batch training

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**