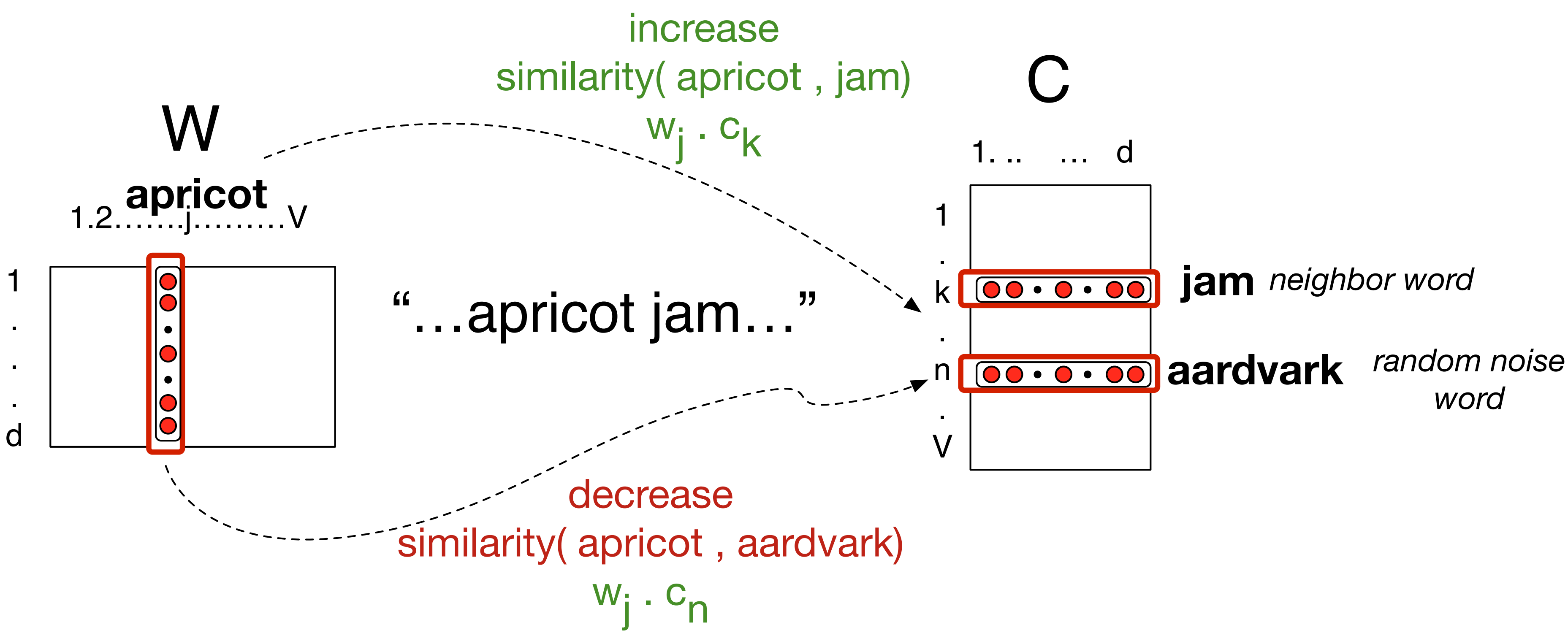
Logistic Regression

JURAFSKY AND MARTIN CHAPTER 5


increase
similarity(apricot , jam)
$w_j \cdot c_k$
W
apricot
1.2…….j………V
1
.
.
.
d
C
1. ..   … d
1
.
k
.
n
.
V
jam
neighbor word
aardvark
random noise word
"…apricot jam…"
decrease
similarity(apricot , aardvark)
$w_j \cdot c_n$

Recap: How to learn word2vec (skip-gram) embeddings

Start with V random 300-dimensional vectors as initial embeddings

Use **logistic regression** to

- Take a corpus and take pairs of words that co-occur as positive examples
- Take pairs of words that don't co-occur as negative examples
- Train the classifier to distinguish these by slowly adjusting all the embeddings to improve the classifier performance
- Throw away the classifier code and keep the embeddings.

Classifier components

Machine learning classifiers require a training corpus of M observations input/output pairs $(x^{(i)}, y^{(i)})$.

1. A **feature representation** of the input. For each input observation $x^{(i)}$, this will be a vector of features $[x_1, x_2, \ldots, x_n]$.
2. A **classification function** that computes the estimated class $\hat{y}$ via $p(y|x)$.
3. An **objective function** for learning, usually involving minimizing error on training examples.
4. An algorithm for **optimizing** the objective function.

Sentiment classifier

For sentiment classification, consider an input observation *x*, represented by a vector of features $[x_1, x_2, ..., x_n]$. The classifier output *y* can be 1 (positive sentiment) or 0 (negative sentiment). We want to estimate ***P*(*y* = 1|*x*)**.

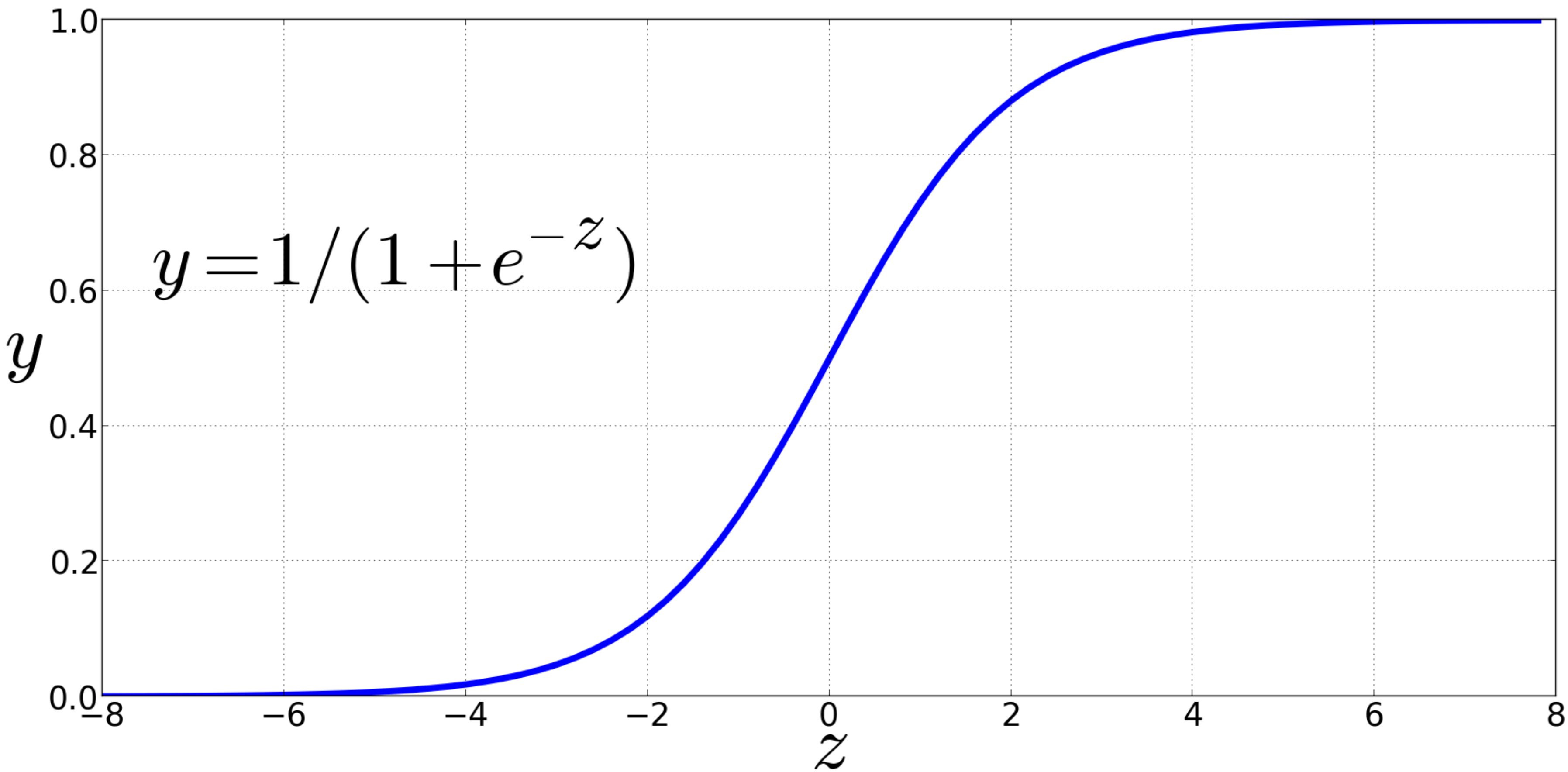
Logistic regression solves this task by learning, from a training set, a vector of **weights** and a **bias term**.

$$z = \left(\sum_{i=1}^{n} w_i x_i\right) + b$$

We can also write this as a dot product:

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

Sigmoid function



Probabilities

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

Decision boundary

Now we have an algorithm that given an instance x computes the probability $P(y = 1|x)$. How do we make a decision?

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

For a test instance x, we say yes if the probability $P(y = 1|x)$ is more than .5, and no otherwise. We call .5 the decision boundary

Extracting Features

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 of positive lexicon words	
x_2	Count of negative lexicon words	
x_3	Does no appear? (binary feature)	
x_4	Number of 1^{st} and 2nd person pronouns	
x_5	Does ! appear? (binary feature)	
x_6	Log of the word count for the document	

Extracting Features

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 of positive lexicon words	3
x_2	Count of negative lexicon words	
x_3	Does no appear? (binary feature)	
x_4	Number of 1^{st} and 2nd person pronouns	
x_5	Does ! appear? (binary feature)	
x_6	Log of the word count for the document	

Extracting Features

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 of positive lexicon words	3
x_2	Count of negative lexicon words	2
x_3	Does no appear? (binary feature)	
x_4	Number of 1^{st} and 2nd person pronouns	
x_5	Does ! appear? (binary feature)	
x_6	Log of the word count for the document	

Extracting Features

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 of positive lexicon words	3
x_2	Count of negative lexicon words	2
x_3	Does no appear? (binary feature)	1
x_4	Number of 1^{st} and 2nd person pronouns	
x_5	Does ! appear? (binary feature)	
x_6	Log of the word count for the document	

Extracting Features

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 of positive lexicon words	3
x_2	Count of negative lexicon words	2
x_3	Does no appear? (binary feature)	1
x_4	Number of 1^{st} and 2nd person pronouns	3
x_5	Does ! appear? (binary feature)	
x_6	Log of the word count for the document	

Extracting Features

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 .

Word count = 64, ln(64) = 4.15

Var	Definition	Value
x_1	Count of positive lexicon words	3
x_2	Count of negative lexicon words	2
x_3	Does no appear? (binary feature)	1
x_4	Number of 1st and 2nd person pronouns	3
x_5	Does ! appear? (binary feature)	0
x_6	Log of the word count for the document	4.15

Var	Definition	Value	Weight	Product
x_1	Count of positive lexicon words	3	2.5	
x_2	Count of negative lexicon words	2	-5.0	
x_3	Does no appear? (binary feature)	1	-1.2	
x_4	Num 1st and 2nd person pronouns	3	0.5	
x_5	Does ! appear? (binary feature)	0	2.0	
x_6	Log of the word count for the doc	4.15	0.7	
b	bias	1	0.1	

$$z = \left(\sum_{i=1}^{n} w_i x_i\right) + b$$

Computing Z

Var	Definition	Value	Weight	Product
x_1	Count of positive lexicon words	3	2.5	7.5
x_2	Count of negative lexicon words	2	-5.0	-10
x_3	Does no appear? (binary feature)	1	-1.2	-1.2
x_4	Num 1^{st} and 2nd person pronouns	3	0.5	1.5
x_5	Does ! appear? (binary feature)	0	2.0	0
x_6	Log of the word count for the doc	4.15	0.7	2.905
b	bias	1	0.1	.1

$$z = \sum_{i=1}^{n} w_i x_i \Big) + b$$

Z=0.805

Sigmoid(Z)

Var	Definition	Value	Weight	Product
x_1	Count of positive lexicon words	3	2.5	7.5
x_2	Count of negative lexicon words	2	-5.0	-10
x_3	Does no appear? (binary feature)	1	-1.2	-1.2
x_4	Num 1st and 2nd person pronouns	3	0.5	1.5
x_5	Does ! appear? (binary feature)	0	2.0	0
x_6	Log of the		0.7	2.905
b	bias		0.1	.1

$\sigma(0.805)$
$= 0.69$

Learning in logistic regression

How do we get the weights of the model? We learn the parameters (weights + bias) via learning. This requires 2 components:

1. An objective function or **loss function** that tells us *distance* between the system output and the gold output. We will use **cross-entropy loss**.
2. An algorithm for optimizing the objective function. We will use stochastic gradient descent to **minimize** the **loss function.**

Loss functions

We need to determine for some observation x how close the classifier output ($\hat{y}$= σ ($w \cdot x + b$)) is to the correct output (y, which is 0 or 1).

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

One example is mean squared error

$$L_{MSE}(\hat{y}, y) = \frac{1}{2}(\hat{y} - y)^2$$

Loss functions for probabilistic classification

We use a loss function that prefers the correct class labels of the training example to be more likely.

Conditional maximum likelihood estimation: Choose parameters *w, b* that maximize the (log) probabilities of the true labels in the training data.

The resulting loss function is the negative log likelihood loss, more commonly called the **cross entropy loss.**

Loss functions for probabilistic classification

For one observation x, let's **maximize** the probability of the correct label p(y|x).

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

If y = 1, then $\mathrm{p(y|x)} = \hat{y}$.

If y = 0, then $\mathrm{p(y|x)} = 1 - \hat{y}$.

Loss functions for probabilistic classification

Change to logs (still maximizing)

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

$$= y \log \hat{y} + (1-y)\log(1-\hat{y})$$

This tells us what log likelihood should be maximized. But for loss functions, we want to minimize things, so we'll flip the sign.

Cross-entropy loss

The result is cross-entropy loss:

$$L_{CE}(\hat{y}, y) = -\log p(y|x) = -[y \log \hat{y} + (1-y)\log(1-\hat{y})]$$

Finally, plug in the definition for $\hat{\boldsymbol{y}}$= σ ($w \cdot x$) + b

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

Cross-entropy loss

Why does minimizing this negative log probability do what we want?

A perfect classifier would assign probability 1 to the correct outcome (y=1 or y=0) and probability 0 to the incorrect outcome.

That means the higher $\hat{\boldsymbol{y}}$ (the closer it is to 1), the better the classifier; the lower $\hat{\boldsymbol{y}}$ is (the closer it is to 0), the worse the classifier.

The negative log of this probability is a convenient loss metric since it goes from 0 (negative log of 1, no loss) to infinity (negative log of 0, infinite loss).

Loss on all training examples

$$\begin{aligned}
\log p(\text{training labels}) &= \log \prod_{i=1}^{m} p(y^{(i)}|x^{(i)}) \\
&= \sum_{i=1}^{m} \log p(y^{(i)}|x^{(i)}) \\
&= -\sum_{i=1}^{m} L_{CE}(\hat{y}^{(i)}, y^{(i)})
\end{aligned}$$

Average Loss

$$
\begin{aligned}
Cost(w,b) &= \frac{1}{m}\sum_{i=1}^{m} L_{CE}(\hat{y}^{(i)}, y^{(i)}) \\
&= -\frac{1}{m}\sum_{i=1}^{m} y^{(i)} \log \sigma(w \cdot x^{(i)} + b) + (1 - y^{(i)}) \log \left(1 - \sigma(w \cdot x^{(i)} + b)\right)
\end{aligned}
$$

This is what we want to minimize!!

Finding good parameters

We use gradient descent to find good settings for our weights and bias by minimizing the loss function.

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

Gradient descent is a method that finds a minimum of a function by figuring out in which direction (in the space of the parameters θ) the function's slope is rising the most steeply, and moving in the opposite direction.

Gradient descent

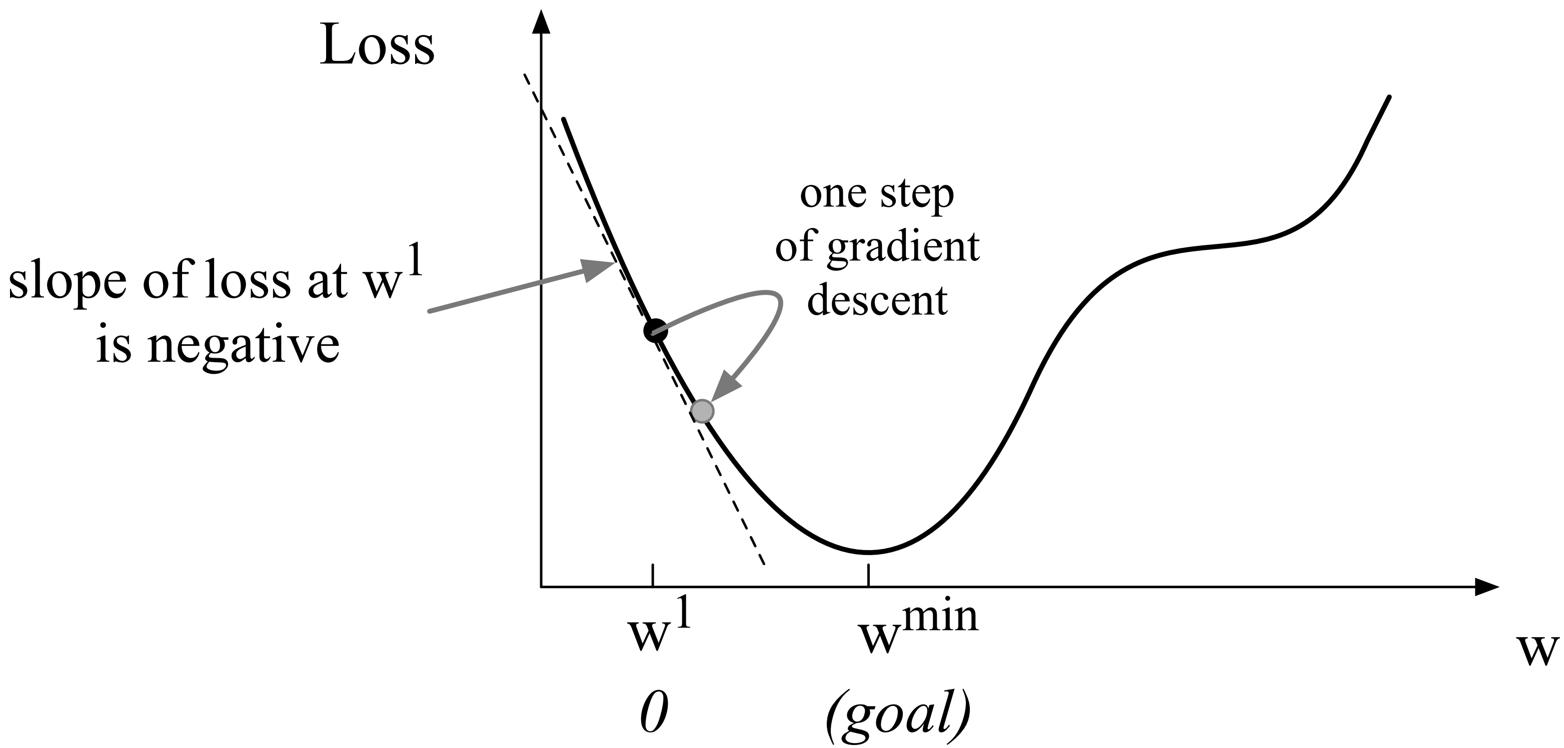
Global v. Local Minimums

For logistic regression, this loss function is conveniently convex.

A convex function has just one minimum, so there are no local minima to get stuck in.

So gradient descent starting from any point is guaranteed to find the minimum.

Iteratively find minimum



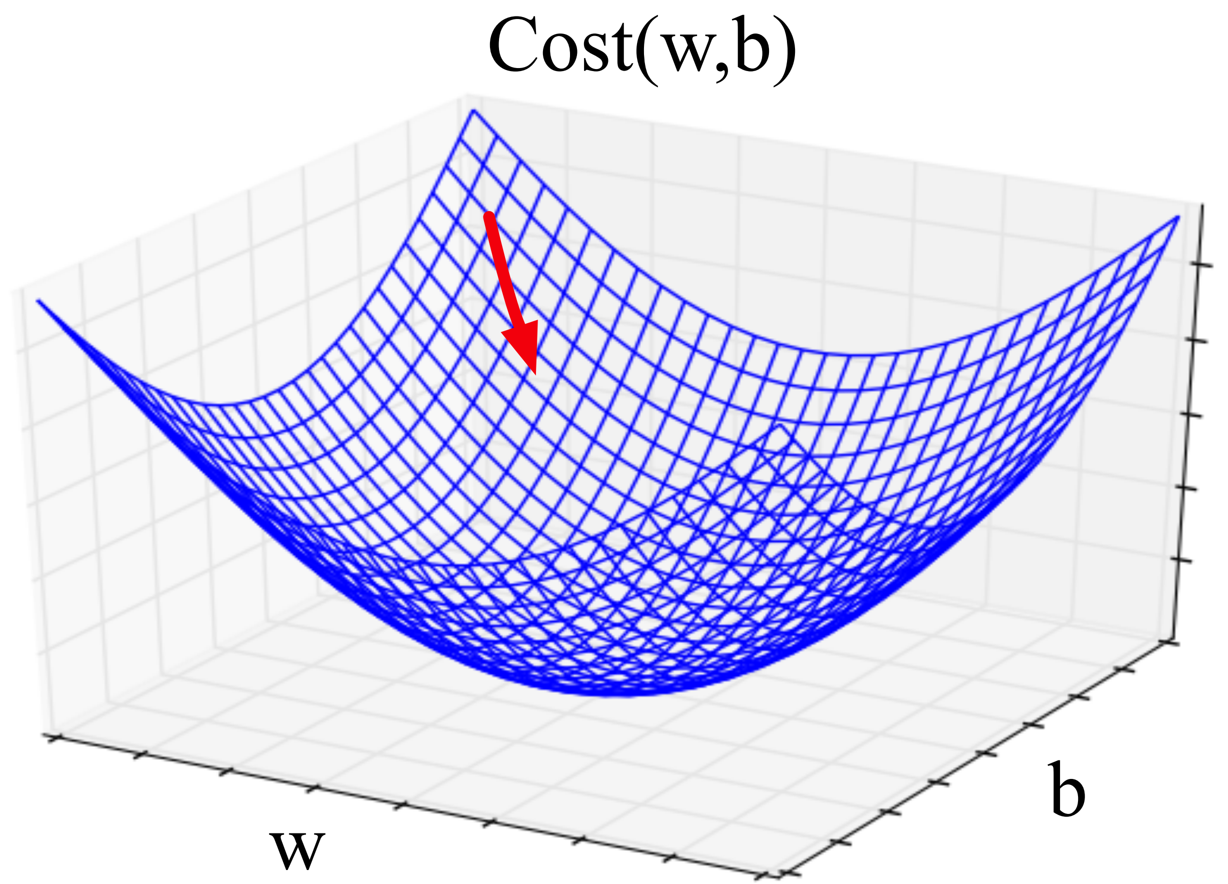
How much should we update the parameter by?

The magnitude of the amount to move in gradient descent is the value of the slope weighted by a learning rate η.

A higher/faster learning rate means that we should move *w* more on each step.

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

Many dimensions



Updating each dimension w_i

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

The final equation for updating θ based on the gradient is

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

The Gradient

To update θ, we need a definition for the gradient $\nabla L(f(x; \theta), y)$.

For logistic regression the cross-entropy loss function is:

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

The derivative of this function for one observation vector *x* for a single weight w^j is

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

The gradient is a very intuitive value: the difference between the true *y* and our estimate for x, multiplied by the corresponding input value x_j .

The Gradient

The loss for a batch of data or an entire dataset is just the average loss over the *m* examples

$$Cost(w,b) \;=\; -\frac{1}{m}\sum_{i=1}^{m} y^{(i)}\log\sigma(w\cdot x^{(i)}+b)+(1-y^{(i)})\log\left(1-\sigma(w\cdot x^{(i)}+b)\right)$$

The gradient for multiple data points is the sum of the indiviidual gradients:

$$\frac{\partial Cost(w,b)}{\partial w_j} \;=\; \sum_{i=1}^{m}\left[\sigma(w\cdot x^{(i)}+b)-y^{(i)}\right]x_j^{(i)}$$

Stochastic gradient descent algorithm

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 $x^{(1)}, x^{(2)}, \ldots, x^{(n)}$
y is the set of training outputs (labels) $y^{(1)}, y^{(2)}, \ldots, y^{(n)}$

$\theta \leftarrow 0$

repeat T times

For each training tuple $(x^{(i)}, y^{(i)})$ (in random order)

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 $y^{(i)}$?

$g \leftarrow \nabla_\theta L(f(x^{(i)}; \theta), y^{(i)})$ # How should we move θ to maximize loss ?

$\theta \leftarrow \theta - \eta\, g$ # go the other way instead

return θ

Worked example

Neural Networks

The building block of a neural network is a single computational unit. A unit takes a set of real valued numbers as input, performs some computation.