Lecture 5: Word2Vec (cont.) & Logistic Regression

Instructor: Xiang Ren

USC CSCI 444 NLP

2026 Spring

Slides mostly adapted from Dan Jurafsky, some from Mohit Iyyer

Announcements

Announcements + Logistics

- Project team finalized by end of Jan → submit your team by 11:59pm today and no more adjustments!
- HW1 is due by Feb 4, 11:59 PM PT
- Project proposal is due by Feb 11, 11:59 PM PT

word2vec

word2vec : Intuition

is traditionally followed by	**cherry**	pie, a traditional dessert
often mixed, such as	**strawberry**	rhubarb pie. Apple pie
computer peripherals and personal	**digital**	assistants. These devices usually
a computer. This includes	**information**	available on the internet

Instead of counting how often each word W occurs near another, e.g. "cherry"

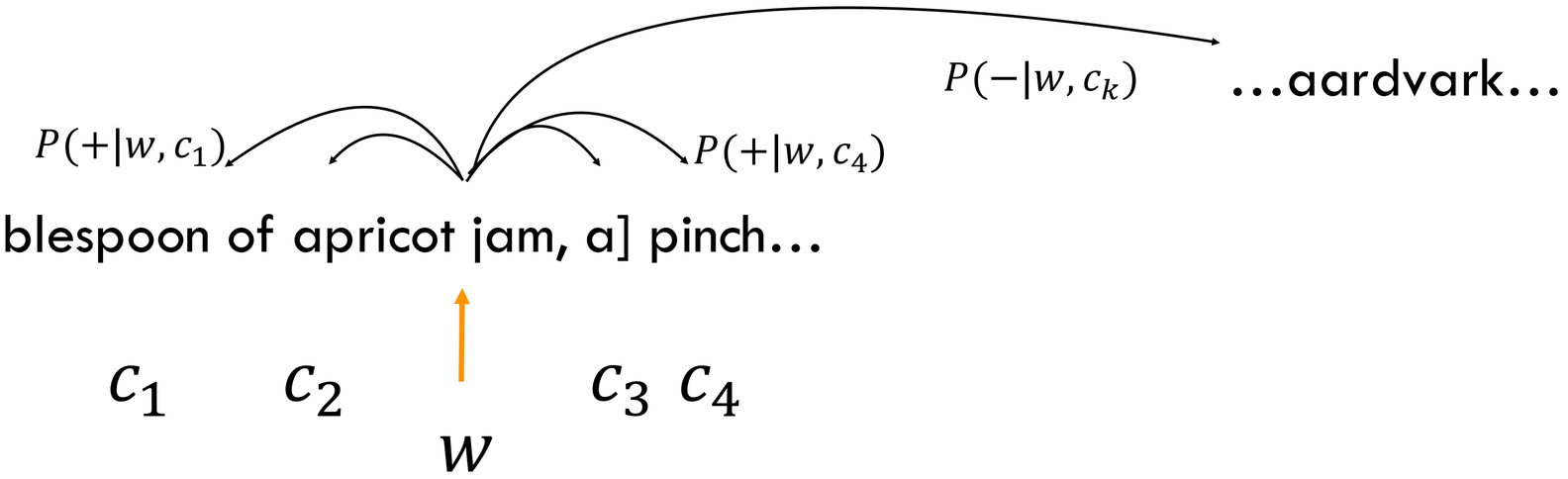
- Train a classifier on a binary prediction task:
  - Is W likely to show up near "cherry"?
- We don't actually care about this task!!!
  - But we'll take the learned **classifier weights** as the word embeddings

x? y?

Word embedding itself is the learned parameter!

word2vec: Goal

Assume a +/- 2 word window, given training sentence:



Goal: train a classifier that is given a candidate (word, context) pair:

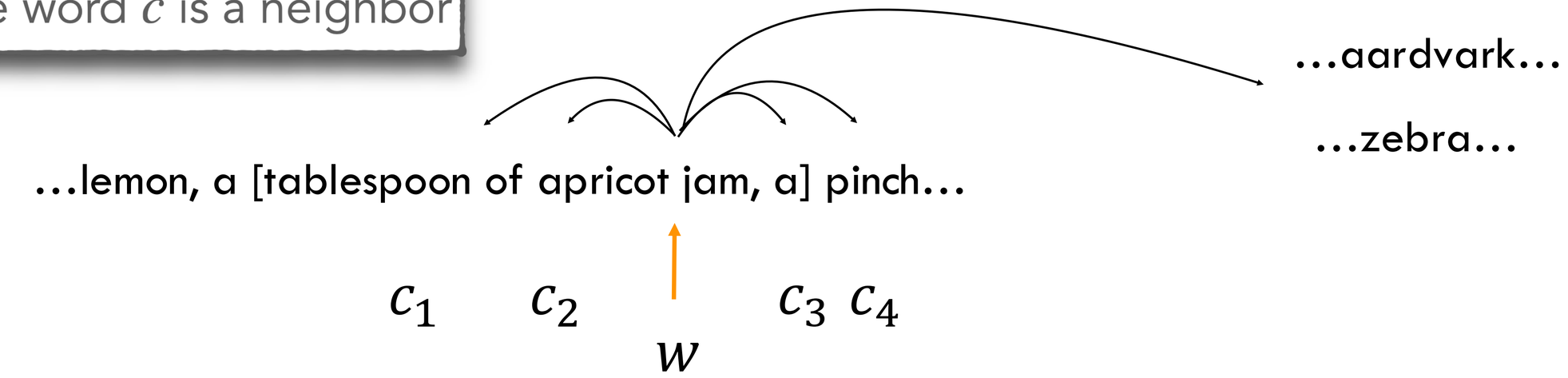
- (apricot, jam)
- (apricot, aardvark)
- …

And assigns each pair a probability:

$$P(+|w, c)$$
$$P(-|w, c) = 1 - P(+|w, c)$$

word2vec: Pseudocode

Predict if candidate word c is a neighbor



1. Treat the target word w and a neighboring context word c as positive examples.
2. Randomly sample other words in the lexicon to get negative examples
3. Use logistic regression to train a classifier to distinguish those two cases
4. Use the learned weights as the embeddings

word2vec: Probability Estimates

$$P(+|w,c)$$
$$P(-|w,c) = 1 - P(+|w,c)$$

- Central intuition: Base this probability on embedding similarity!
- Remember: two vectors are similar if they have a high dot product
  - Cosine similarity is just a normalized dot product
- So:

$$sim(w,c) \propto \mathbf{w} \cdot \mathbf{c}$$

Vectors, not scalars!

- Still not a probability!
  - We'll need to normalize to get a probability

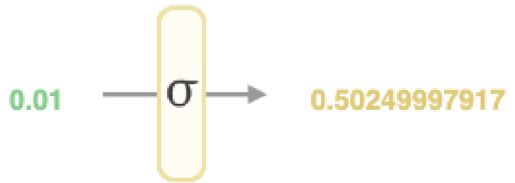
Can we just use cosine?

Turning dot products into probabilities

Sigmoid

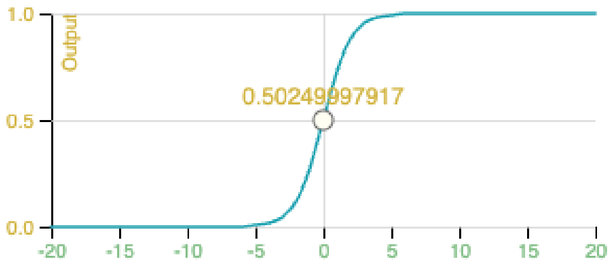
Similarity:

$$sim(w, c) \approx \mathbf{w} \cdot \mathbf{c}$$



Turn into a probability using the sigmoid function:

$$f(0.01) = \frac{1}{1 + e^{-(0.01)}} = 0.50249997917$$



$$P(+|w, c) = \sigma(\mathbf{c} \cdot \mathbf{w}) = \frac{1}{1 + \exp(-\mathbf{c} \cdot \mathbf{w})}$$

$$P(-|w, c) = 1 - P(+|w, c)$$

$$= \sigma(-\mathbf{c} \cdot \mathbf{w}) = \frac{1}{1 + \exp(\mathbf{c} \cdot \mathbf{w})}$$

Logistic Regression!

Accounting for a context window

$$P(+|w, c) = \sigma(\mathbf{c} \cdot \mathbf{w}) = \frac{1}{1 + \exp(-\mathbf{c} \cdot \mathbf{w})}$$

Single Context Word

But we have lots of context words

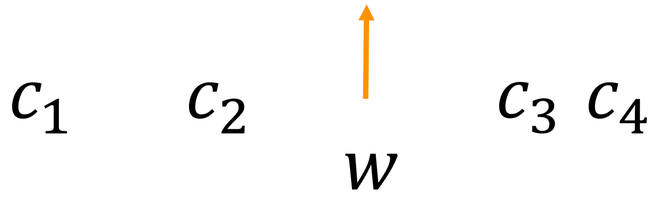
- Depends on window size
- We'll assume independence and just multiply them

Same with negative context words!

c_{neg} } ...aardvark...
...zebra...

$$\log P(-|w, c_{neg}) = \sum_{c' \in c_{neg}} \log \sigma(-\mathbf{c}' \cdot \mathbf{w})$$

...lemon, a [tablespoon of apricot jam, a] pinch...

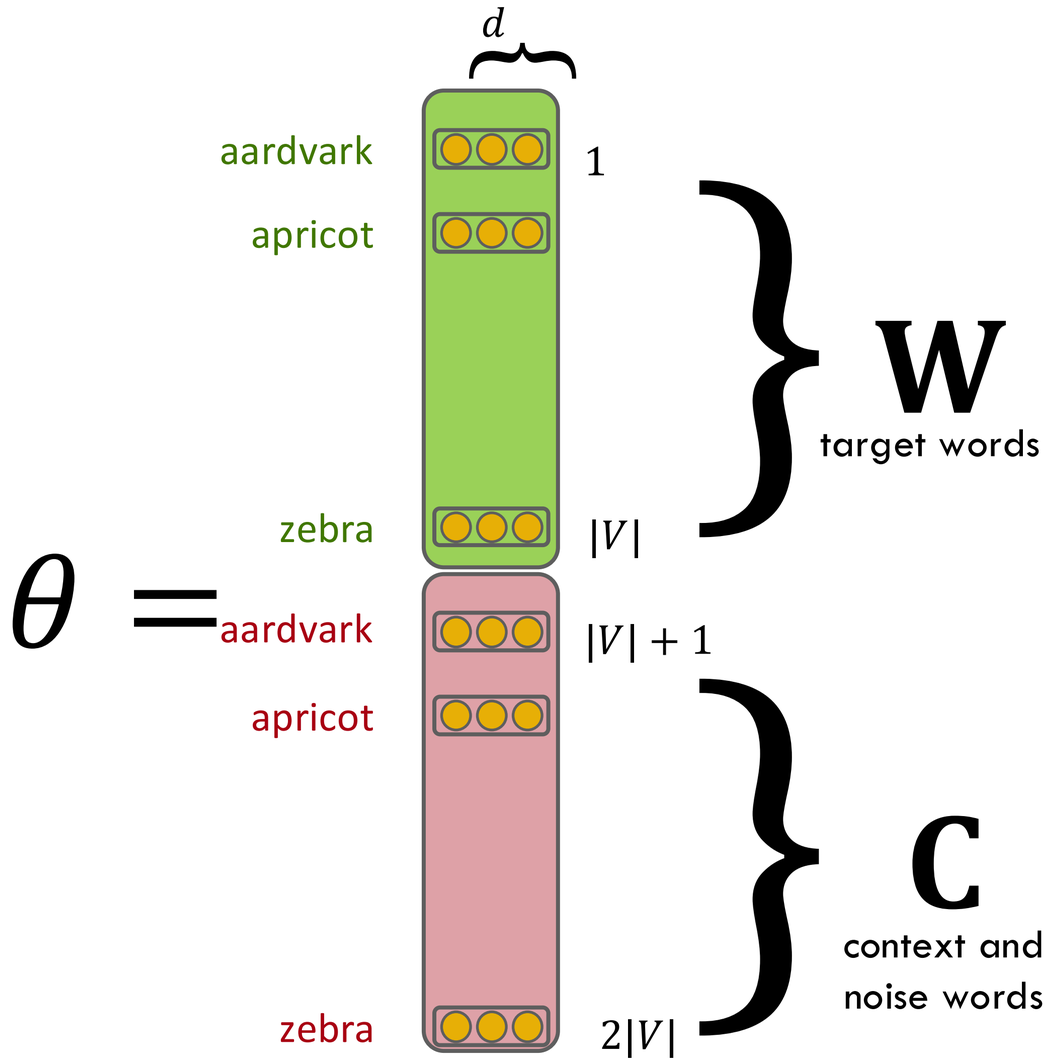


$$P(+|w, c_{1:L}) = \prod_{i=1}^{L} \sigma(\mathbf{c}_i \cdot \mathbf{w})$$

$$\log P(+|w, c_{1:L}) = \sum_{i=1}^{L} \log \sigma(\mathbf{c}_i \cdot \mathbf{w})$$

word2vec classifier: Summary

- A probabilistic classifier, given
  - a test target word w
  - its context window of L words $c_{1:L}$
- Estimates probability that w occurs in this window based on similarity of w (embeddings) to $c_{1:L}$ (embeddings)
- To compute this, we just need embeddings for all the words
  - Separate representations for targets and contexts
  - Same as the parameters we need to estimate!



Learning word2vec embeddings

Word2vec: Training Data

For each positive example we'll grab a set of negative examples, sampling by weighted unigram frequency

c_{neg} } ...aardvark...
...zebra...

...lemon, a [tablespoon of apricot jam, a] pinch...

c_1 c_2 w c_3 c_4

Negative examples

w	c_{neg}
apricot	aardvark
apricot	zebra
apricot	where
apricot	adversarial

Positive examples

w	c
apricot	tablespoon
apricot	of
apricot	jam
apricot	a

Word2vec: Learning Problem

c_{neg} } …aardvark…
…zebra…

…lemon, a [tablespoon of apricot jam, a] pinch…

c_1 c_2 w c_3 c_4

Given

- the set of positive and negative training instances, and
- a set of randomly initialized embedding vectors of size $2|V|$,

the goal of learning is to adjust those word vectors such that we:

- Maximize the similarity of the target word, context word pairs $(w, c_{1:L})$ drawn from the positive data
- Minimize the similarity of the (w, c_{neg}) pairs drawn from the negative data

Loss function

Maximize the similarity of the target with the actual context words in a window of size L, and minimize the similarity of the target with the $K > L$ negative sampled non-neighbor words

For every word, context pair…

Cross Entropy

$$L_{CE} = -\log[P(+|\mathbf{w}, \mathbf{c}_{pos})P(-|\mathbf{w}, \mathbf{c}_{neg})]$$

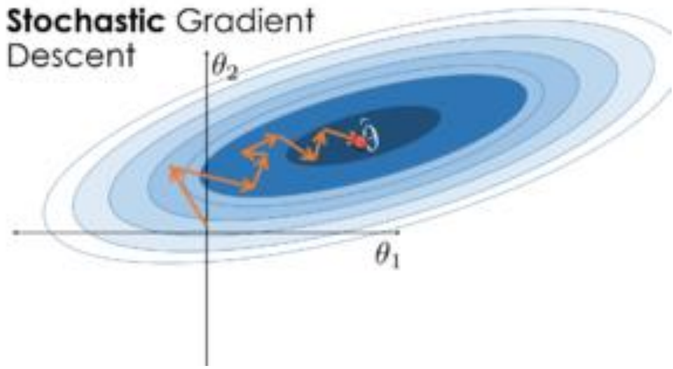
$$= -[\log P(+|\mathbf{w}, \mathbf{c}_{pos}) + \sum_{j=1}^{K} \log P(-|\mathbf{w}, \mathbf{c}_{neg_j})]$$

$$= -[\log P(+|\mathbf{w}, \mathbf{c}_{pos}) + \sum_{j=1}^{K} \log(1 - P(+|\mathbf{w}, \mathbf{c}_{neg_j})]$$

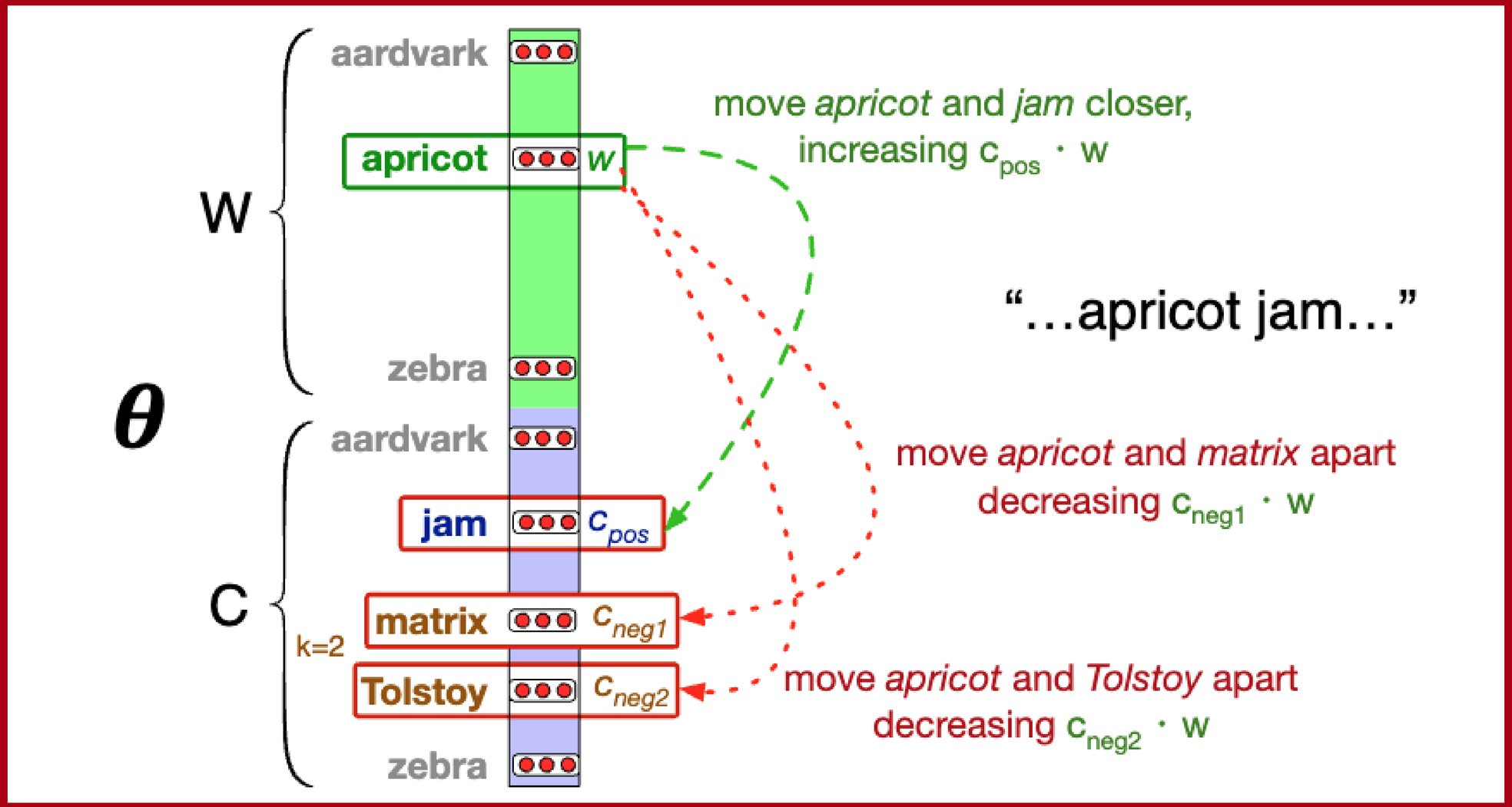
$$= -[\log \sigma(\mathbf{w} \cdot \mathbf{c}_{pos}) + \sum_{j=1}^{K} \log \sigma(-\mathbf{w} \cdot \mathbf{c}_{neg_j})]$$

Learning the classifier

- How to learn?
  - Stochastic gradient descent!
  - Iterative process
  - Start with randomly initialized weights
    - Update the parameters (coming up)
  - Stop when the parameters do not change much…



- We'll adjust the word weights to
  - make the positive pairs more likely
  - and the negative pairs less likely,
  - over the entire training set.


θ
W
aardvark
apricot
w
zebra
C
aardvark
jam
c_{pos}
matrix
c_{neg1}
k=2
Tolstoy
c_{neg2}
zebra
move *apricot* and *jam* closer,
increasing $c_{pos} \cdot w$
"…apricot jam…"
move *apricot* and *matrix* apart
decreasing $c_{neg1} \cdot w$
move *apricot* and *Tolstoy* apart
decreasing $c_{neg2} \cdot w$

Reminder: Gradient Descent

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

At each step of gradient descent, we update the parameter w

- Direction: We move in the reverse direction from the gradient of the loss function
- Magnitude: we move the value of this gradient $\frac{\partial}{\partial w} L(f(x; w), y^*)$, weighted by a learning rate η
- Higher learning rate means move w faster

SGD: Derivates

$$L_{CE} = -[\log\sigma(\mathbf{w} \cdot \mathbf{c}_{pos}) + \sum_{j=1}^{K} \log\sigma(-\mathbf{w} \cdot \mathbf{c}_{neg_j})]$$

3 different parameters

$$\frac{\partial L_{CE}}{\partial \mathbf{c}_{pos}} = [\sigma(\mathbf{c}_{pos} \cdot \mathbf{w}) - 1]\mathbf{w}$$

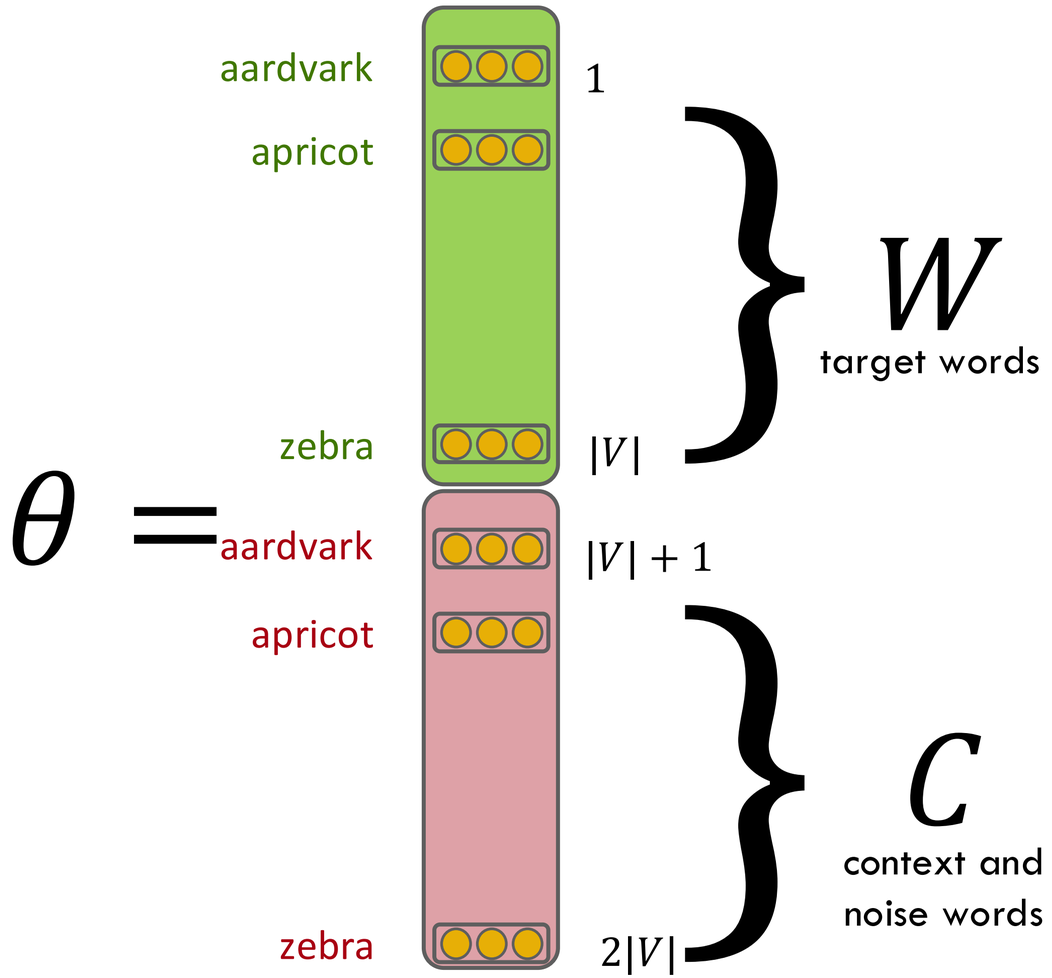
$$\frac{\partial L_{CE}}{\partial \mathbf{c}_{neg_j}} = [\sigma(\mathbf{c}_{neg_j} \cdot \mathbf{w})]\mathbf{w}$$

Update the parameters by subtracting respective η-weighted gradients

$$\frac{\partial L_{CE}}{\partial w} = [\sigma(\mathbf{c}_{pos} \cdot \mathbf{w}) - 1]\mathbf{c}_{pos} + \sum_{j=1}^{K} [\sigma(\mathbf{c}_{neg_j} \cdot \mathbf{w})]\mathbf{c}_{neg_j}$$

word2vec: Learned Embeddings

- SGNS learns two sets of embeddings:
  - Target embeddings matrix $\mathbf{W}$
  - Context embedding matrix $\mathbf{C}$
- It's common to just add them together, representing word i as the vector $\mathbf{w}_i + \mathbf{c}_i$



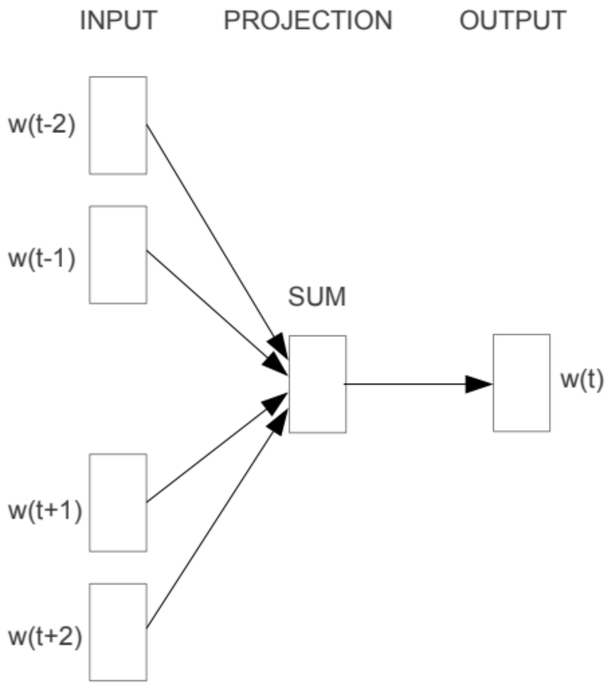
CBOW and Skipgram

- CBOW: continuous bag of words - given context, predict which word might be in the target position

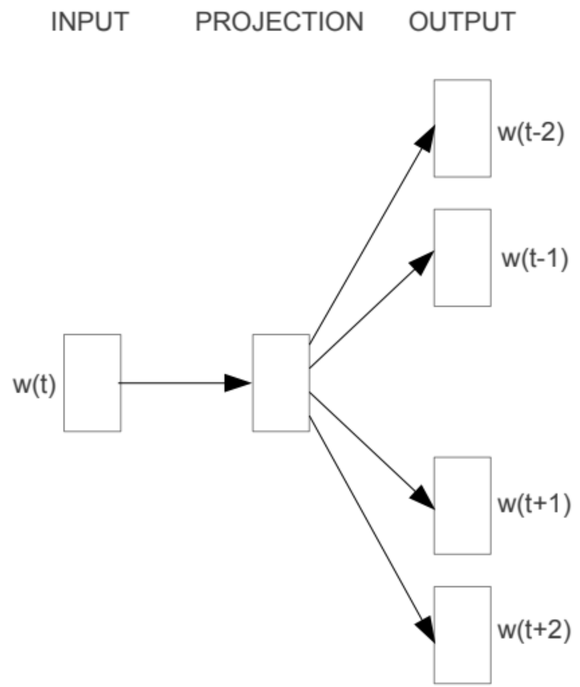
- Skip-gram: given word, predict which words make the best context

- CBOW is faster than Skip-gram. Why?

- Skip-gram generally works better



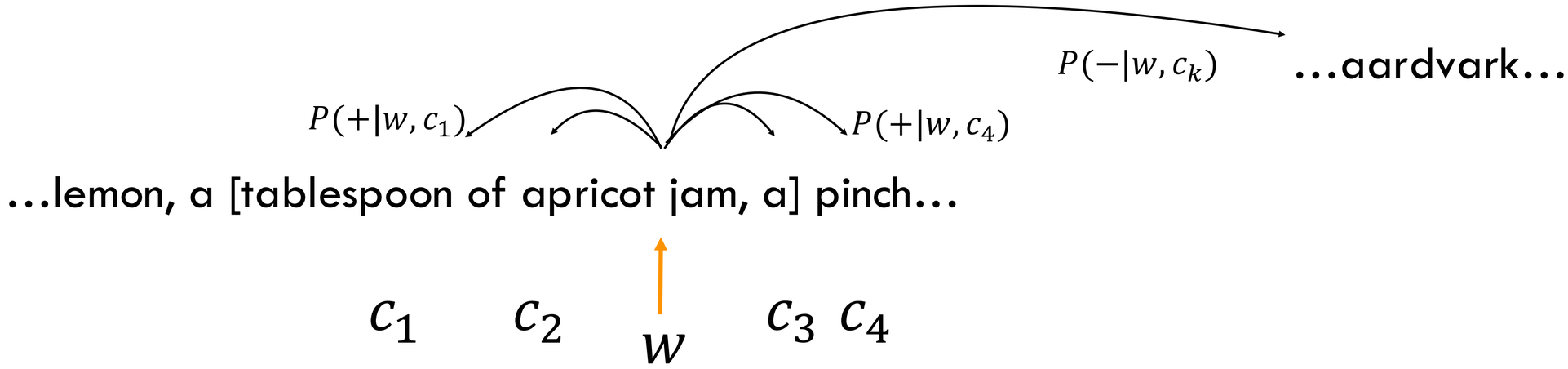
CBOW



Skip-gram

Mikolov et al., 2013. Exploiting Similarities among Languages for Machine Translation.

word2vec: Summary



- Start with $2|V|$ random d-dimensional vectors as initial embeddings
- Train a classifier based on embedding similarity
  - 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.

GloVe: Global Vectors

- Another very widely used static embedding model
  - model is based on capturing global corpus statistics
    - based on ratios of probabilities from the word-word co-occurrence matrix,
    - intuitions of count-based models like PPMI
- Builds on matrix factorization
  - Idea: store most of the important information in a fixed, small number of dimensions: a dense vector
  - Goal: Create a low-dimensional matrix for the embedding while minimizing reconstruction loss
- Fast training, scalable to huge corpora

Pennington et al., 2014

Supervised Machine Learning 101

Ingredients of Supervised Machine Learning

I. Data as pairs $(x^{(i)}, y^{(i)}) \quad s.t \;\; i \in \{1 \ldots N\}$

- $x^{(i)}$ usually represented by a feature vector $\mathbf{x}^{(i)} = [x_1, x_2, \ldots, x_d]$,
  - e.g. word embeddings

II. Model

- A classification function that computes $\hat{y}$, the estimated class, via $p(y|x)$
  - e.g. sigmoid function: $\sigma(z) = 1/(1 + \exp(-z))$

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

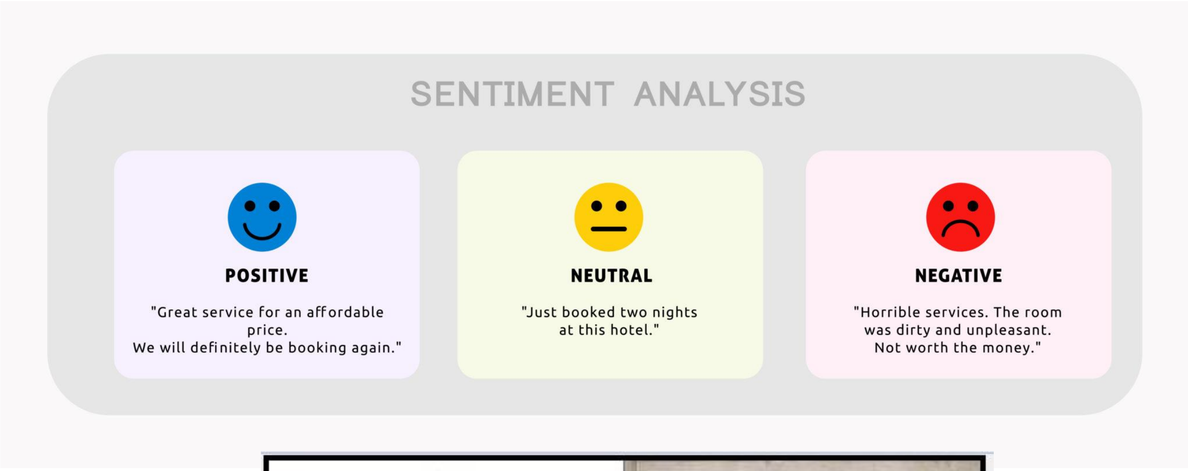
Learning Phase

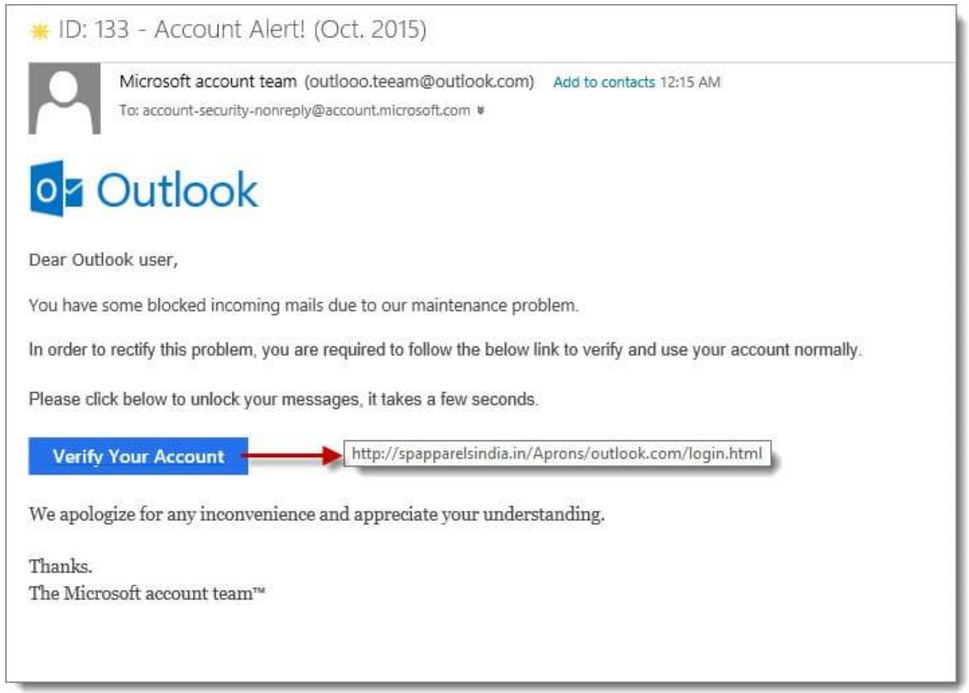
Learning vs. Inference

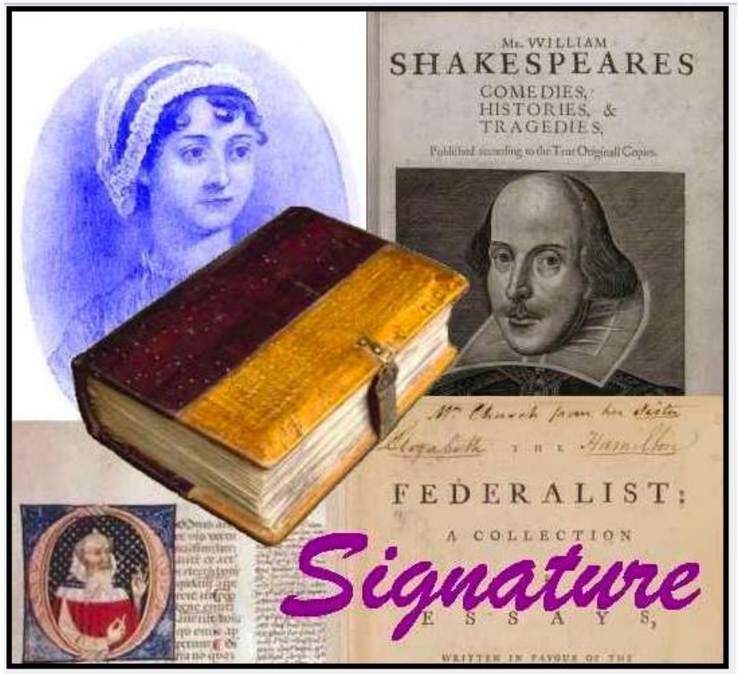
- Learning: we learn parameter weights by minimizing the loss function using an optimization algorithm
- Inference: Given a test example x_{test} we compute $p(y|x)$ using learned weights and return whichever label receives higher probability
- Distinct from training and evaluation
  - Evaluation only contains inference; no parameters are updated
  - Training contains both learning and inference

I. Data

Examples of Classification Tasks







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

Text Classification Setup

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

Features in Classification

- Examples of feature x_i
  - x_i ="review contains 'awesome'"
  - x_j ="review contains 'abysmal'"
  - x_k ="review contains 'mediocre'"

$$w_i = +10$$
$$w_j = -10$$
$$w_k = -2$$

- Each x_i is associated with a weight w_i which determines how important x_i is
  - (For predi

Can you guess the w for x_l = "review contains 'restaurant'"?

II. Model: Logistic Regression

How to get the right y?

- For each feature x_i, introduce a weight w_i, which determines the importance of x_i
  - Plus we will have a bias term b
  - 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?

Probabilistic Models!

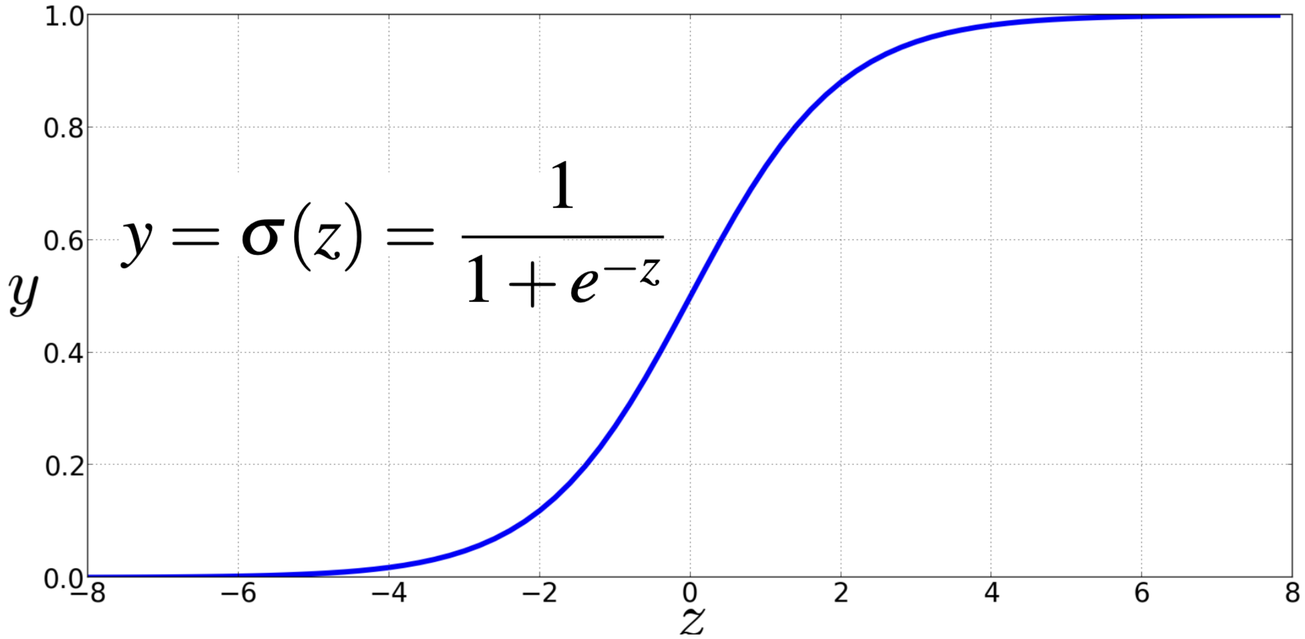
$$P(y = 1|\mathbf{x}; \theta)$$

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

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!



Sigmoids and Probabilities

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

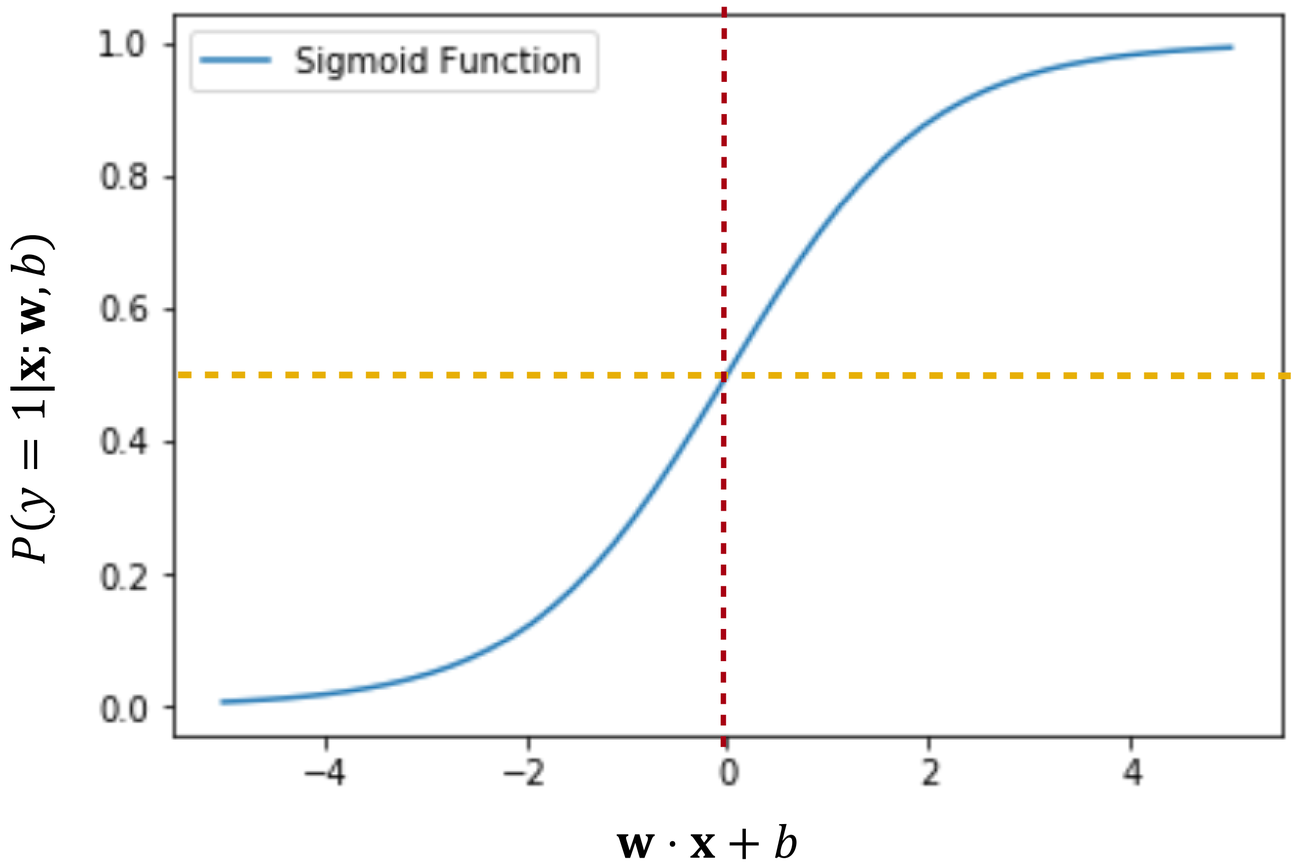
$$P(y = 0|\mathbf{x};\theta) = 1 - \sigma(\mathbf{w}\cdot\mathbf{x} + b)$$
$$= 1 - \frac{1}{1 + \exp(-(\mathbf{w}\cdot\mathbf{x} + b))}$$
$$= \frac{\exp(-(\mathbf{w}\cdot\mathbf{x} + b))}{1 + \exp(-(\mathbf{w}\cdot\mathbf{x} + b))}$$
$$= \frac{1}{1 + \exp(\mathbf{w}\cdot\mathbf{x} + b)}$$
$$= \sigma(-(\mathbf{w}\cdot\mathbf{x}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

Classification Decision

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

Decision Boundary

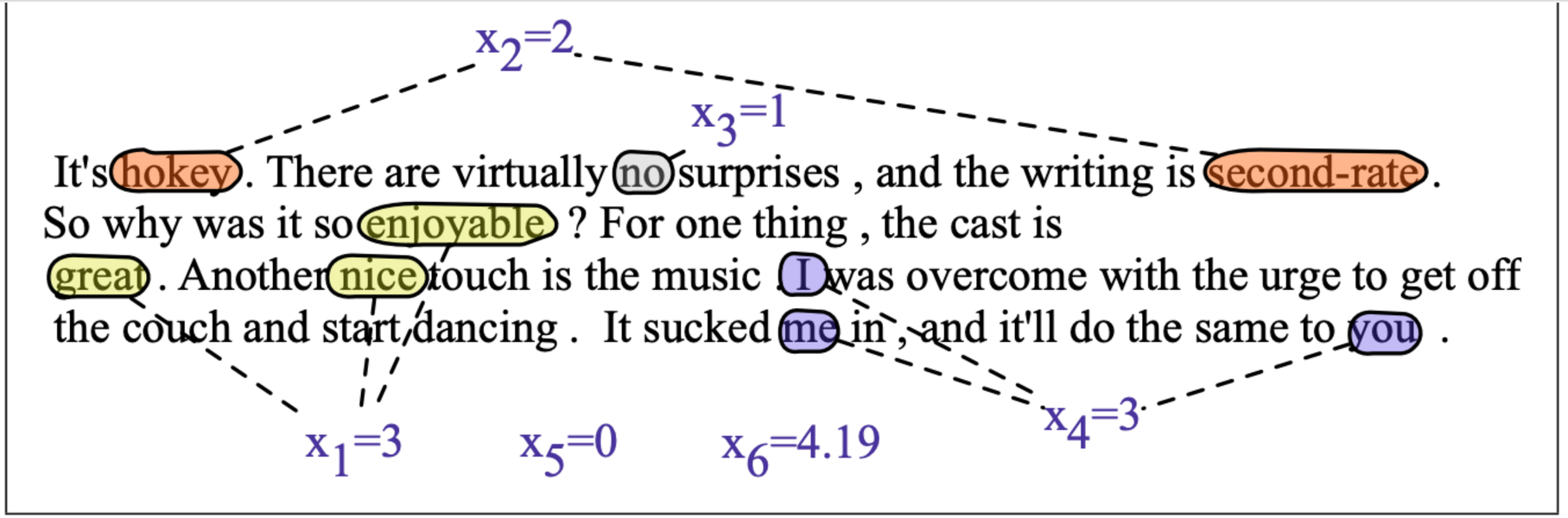
$$\hat{y} = \begin{cases} 1 & if \quad \mathbf{w} \cdot \mathbf{x} + b > 0 \\ 0 & if \quad \mathbf{w} \cdot \mathbf{x} + b \leq 0 \end{cases}$$



Example: Sentiment Classification

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.

Is $y = 1$ or $y = 0$?



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(word count of doc)	$\ln(66) = 4.19$

Example: Classifying Sentiment

Var	Definition	Val
x_1	count(positive lexicon) $\in$ doc)	3
x_2	count(negative lexicon) $\in$ doc)	2
x_3	$\begin{cases} 1 & \text{if } \text{“no”} \in \text{doc} \\ 0 & \text{otherwise} \end{cases}$	1
x_4	count(1st and 2nd pronouns $\in$ doc)	3
x_5	$\begin{cases} 1 & \text{if } \text{“!”} \in \text{doc} \\ 0 & \text{otherwise} \end{cases}$	0
x_6	log(word count of doc)	$\ln(66) = 4.19$

5.2

Suppose $w = [2.5, -5.0, -1.2, 0.5, 2.0, 0.7]$

$b = 0.1$

Example: Classifying Sentiment

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

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

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.

Applying LR to other tasks

- Example: Period Disambiguation: Does a period correspond to the end of sentence?
  - "We saw many algorithms in class, e.g. word2vec."

$$x_1 = \begin{cases} 1 & \text{if } \text{“}Case(w_i) = \text{Lower”} \\ 0 & \text{otherwise} \end{cases}$$

$$x_2 = \begin{cases} 1 & \text{if } \text{“}w_i \in \text{AcronymDict”} \\ 0 & \text{otherwise} \end{cases}$$

$$x_3 = \begin{cases} 1 & \text{if } \text{“}w_i = \text{St. \& } Case(w_{i-1}) = \text{Cap”} \\ 0 & \text{otherwise} \end{cases}$$

Different tasks need different features; manually designed features must be task specific!

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

III. Loss: Cross-Entropy

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

- We want to know how far is the classifier output:
  - $\hat{y} = \sigma(\mathbf{w} \cdot \mathbf{x} + b)$
  - (Not exactly, but it's reducible to $\hat{y}$)
- 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

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

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

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

$$p(y|x;\mathbf{w},b) = \hat{y}^{y}(1-\hat{y})^{1-y}$$

$$\hat{y} = \sigma(\mathbf{w}\cdot\mathbf{x}+b)$$

Bernoulli Distribution

Only extremes!

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

Maximizing conditional likelihood

- Goal: maximize probability of the correct label $p(y|\mathbf{x})$
- Maximize: $p(y|x) = \hat{y}^y(1-\hat{y})^{1-y}$
- Now take the log of both sides… to avoid underflow issues…

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

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

Minimizing negative log likelihood

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

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

- Now flip the sign for something to minimize (we minimize the loss / cost)
- Minimize: $L_{CE}(y,\hat{y}) = -\log p(y|x) = -[y\log\hat{y} + (1-y)\log(1-\hat{y})]$

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

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

Cross-Entropy Loss

Loss for sentiment classification

Case 1: Sentiment Analysis

- 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?

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

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 $y = 1$):

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

Loss is bigger when the model is wrong!

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

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

Loss in word2vec

Case 2: Word2Vec

c_{neg} } …aardvark…
…zebra…

…lemon, a [tablespoon of apricot jam, a] pinch…

c_1 c_2 w c_3 c_4

- Given
  - the set of positive and negative training instances, and
  - a set of randomly initialized embedding vectors of size $2|V|$,
  - the goal of learning is to adjust those word vectors such that we:
    - Maximize the similarity of the target word, context word pairs $(w, c_{1:L})$ drawn from the positive data
    - Minimize the similarity of the (w, c_{neg}) pairs drawn from the negative data

Loss function

Case 2: Word2Vec

Maximize the similarity of the target with the actual context words in a window of size L, and minimize the similarity of the target with the $K > L$ negative sampled non-neighbor words

For every word, context pair…

Cross Entropy

$$L_{CE} = -\log[P(+|\mathbf{w}, \mathbf{c}_{pos})P(-|\mathbf{w}, \mathbf{c}_{neg})]$$

$$= -[\log P(+|\mathbf{w}, \mathbf{c}_{pos}) + \sum_{j=1}^{K} \log P(-|\mathbf{w}, \mathbf{c}_{neg_j})]$$

$$= -[\log P(+|\mathbf{w}, \mathbf{c}_{pos}) + \sum_{j=1}^{K} \log(1 - P(+|\mathbf{w}, \mathbf{c}_{neg_j})]$$

$$= -[\log \sigma(\mathbf{w} \cdot \mathbf{c}_{pos}) + \sum_{j=1}^{K} \log \sigma(-\mathbf{w} \cdot \mathbf{c}_{neg_j})]$$

Regularization

Overfitting

- A model that perfectly match the training data has a problem.
- It will also overfit to the data, modeling noise
  - A random word that perfectly predicts y (it happens to only occur in one class) will get a very high weight.
  - Failing to generalize to a test set without this word.

Why?

A good model should be able to generalize

What happens when a feature only occurs with one class?

e.g. word "wow" for positive reviews

Overfitting: Features

This movie drew me in, and it'll do the same to you.

Useful or harmless features

x_1 = "this"
x_2 = "movie
x_3 = “hated”
x_4 = "drew me in"

I can't tell you how much I hated this movie. It sucked.

4gram features that just "memorize" training set and might cause problems

x_5 = "the same to you"
x_6 = "tell you how much”

Overfitting

- 4-gram model on tiny data will just memorize the data
  - 100% accuracy on the training set
- But it will be surprised by the novel 4-grams in the test data
  - Low accuracy on test set
- Models that are too powerful can overfit the data
  - Fitting the details of the training data so exactly that the model doesn't generalize well to the test set

How to avoid overfitting?

Regularization in logistic regression

Dropout in 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)

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

- 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 / Ridge Regularization

- The sum of the squares of the weights
- The name is because this is the (square of the) L2 norm $\| \theta \|_2^2$, = Euclidean distance of θ to the origin.

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

L2 regularized objective function:

$$\hat{\theta} = \underset{\theta}{\text{argmax}} \sum_{i=1}^{n} \log P(y^{(i)}|x^{(i)}) - \alpha \sum_{j=1}^{d} \theta_j^2$$

L1 / Lasso Regularization

- The sum of the (absolute value of the) weights
- Named after the L1 norm $\| \theta \|_1$ = sum of the absolute values of the weights = Manhattan distance

$$R(\theta) = \| \theta \|_1 = \sum_{j=1}^{d} |\theta_j|$$

L1 regularized objective function:

$$\hat{\theta} = \underset{\theta}{\operatorname{argmax}} \sum_{i=1}^{n} \log P(y^{(i)}|x^{(i)}) - \alpha \sum_{j=1}^{d} |\theta_j|$$

IV. Optimization: Stochastic Gradient Descent

Our goal: minimize the loss

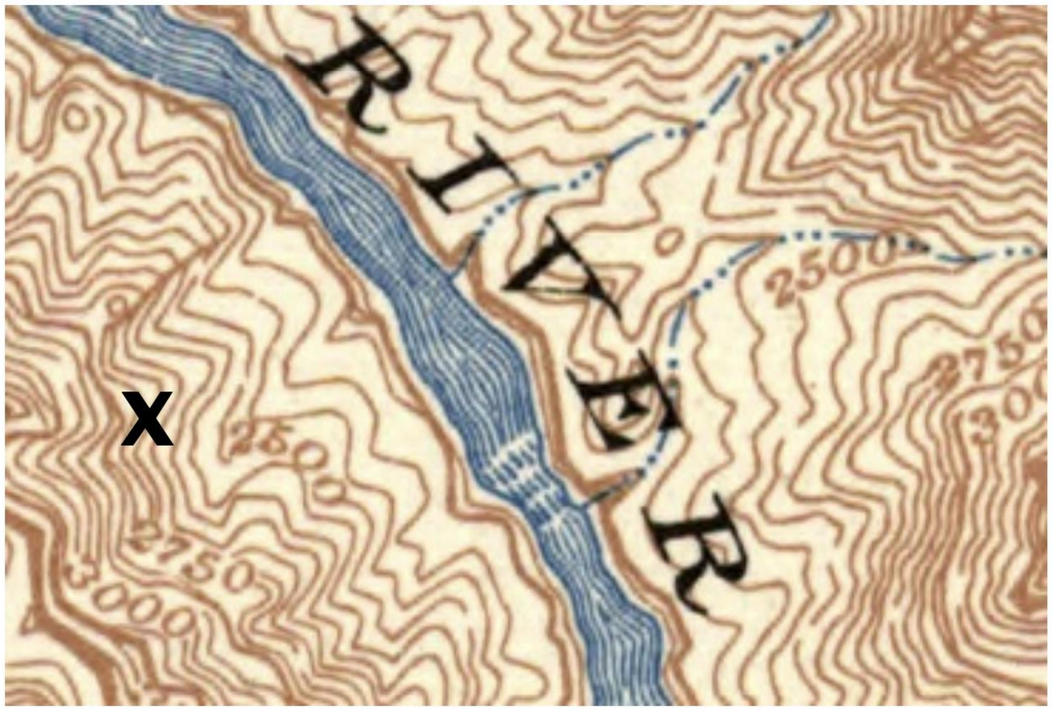
- Loss function is parameterized by weights: $\theta = [\mathbf{w}; b]$
- We will represent $\hat{y}$ as $f(x; \theta)$ to make the dependence on θ more obvious
- We want the weights that minimize the loss, averaged over all examples:

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

Intuition for gradient descent

How to get to the bottom of the river canyon?

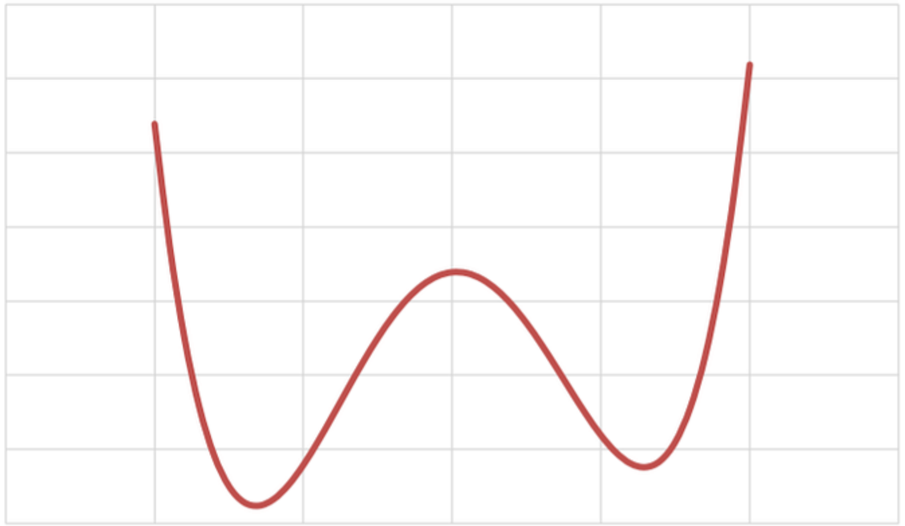
- Look around 360°
- Find the direction of steepest slope down
- Go that way



What if multiple equally good alternatives?

Logistic Regression: Loss

Convex function



Non-convex function

Neural Networks - multiple alternatives

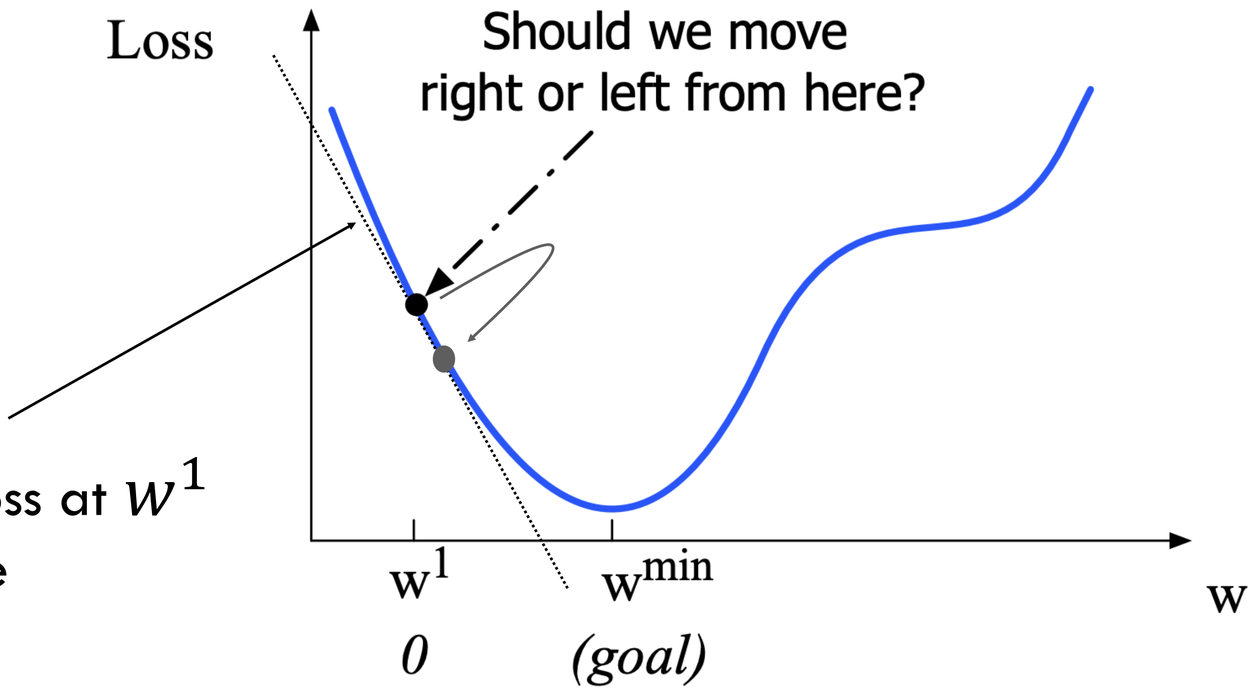
- Has only one option for steepest gradient
  - Or one minimum
- Gradient descent starting from any point is guaranteed to find the minimum

Image Credit: Medium

Consider: a single scalar w

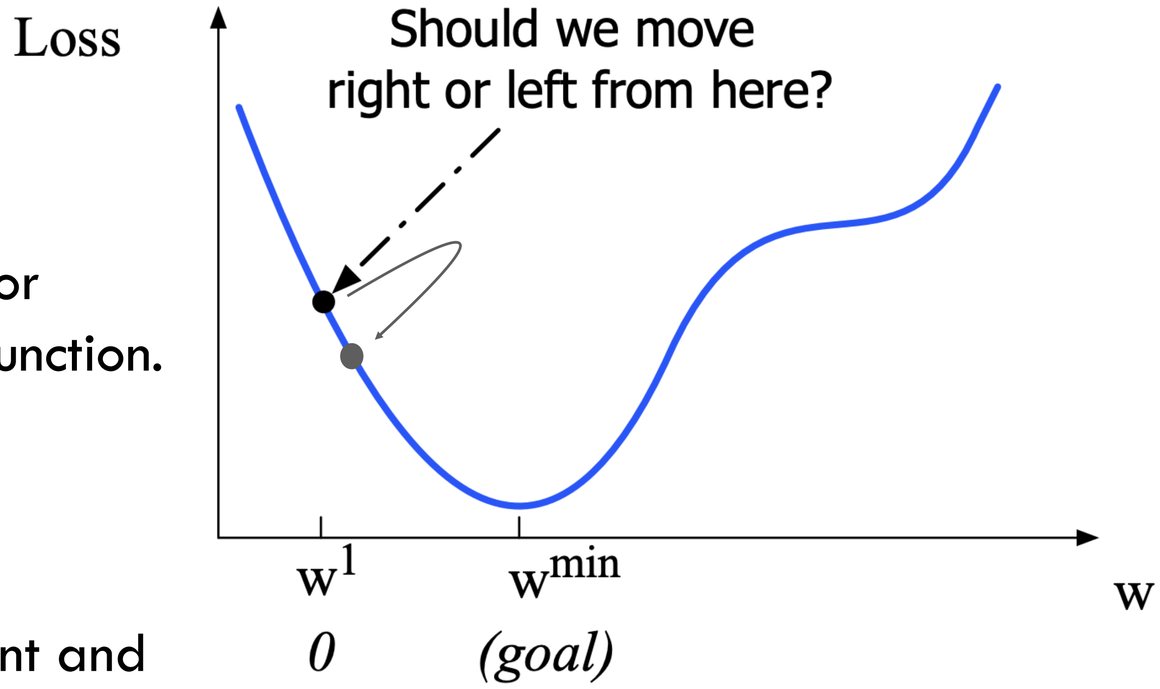
Given current w, should we make it bigger or smaller?

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



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.

Gradient Descent

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

But by how much?