Natural Language Processing
CS 6320
Lecture 5
Dense Embeddings

Instructor: Sanda Harabagiu

Tf-idf and PPMI are sparse representations

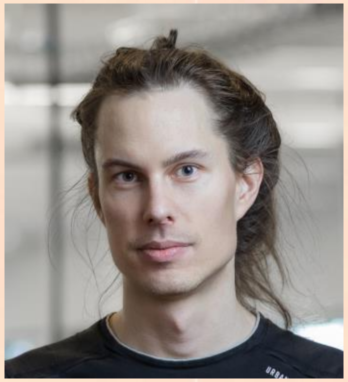
- *tf-idf and PPMI vectors are*
  - ***long** (length |V|= 20,000 to 50,000)*
  - ***sparse** (most elements are zero)*
- *Alternative: vectors which are*
  - ***short** (length 50-1000)*
  - ***dense** (most elements are non-zero)*

Sparse versus dense vectors

- *Why dense vectors?*
  - *Short vectors may be easier to use as **features** in machine learning (less weights to tune)*
  - *Dense vectors may **generalize** better than storing explicit counts*
  - *They may do better at capturing synonymy:*
    - *car and automobile are synonyms; but are distinct dimensions*
      - *a word with car as a neighbor and a word with automobile as a neighbor should be similar, but aren't*
  - ***In practice, they work better***

Dense embeddings you can download!

- ***Word2vec*** *(Mikolov et al.)*
- *https://code.google.com/archive/p/word2vec/*
- ***Fasttext*** *http://www.fasttext.cc/*
- ***Glove*** *(Pennington, Socher, Manning)*
- *http://nlp.stanford.edu/projects/glove/*



Word2vec

- *Popular embedding method*
- *Very fast to train*
- *Code available on the web*
- *Idea: **predict** rather than **count***

Word2vec

- *Instead of* ***counting*** *how often each word w occurs near "apricot"*
- *Train a classifier on a binary* ***prediction*** *task:*
  - *Is w likely to show up near "apricot"?*
- *We don’t actually care about this task*
  - *But we'll take the learned classifier weights as the word embeddings*

Brilliant insight: Use running text as implicitly supervised training data!

- *A word s near apricot*
  - *Acts as gold 'correct answer' to the question:*

  "Is word w likely to show up near apricot?"
- *No need for hand-labeled supervision*
- *The idea comes from **neural language modeling***
  - *Bengio et al. (2003)*
  - *Collobert et al. (2011)*

Word2Vec: Skip-Gram Task

- *Word2vec provides a variety of options. Let's do*
  - *"skip-gram with negative sampling" (SGNS)*

Word2Vec with Skip-gram algorithm

Word2vec provides a variety of options. Let's do "skip-gram with negative sampling" (SGNS)

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

Skip-Gram Training Data

Training sentence:

- *... lemon, a tablespoon of* ***apricot*** *jam a pinch ...*

c1 c2 target c3 c4

Asssume context words are those in +/- 2 word window

Skip-Gram Goal

- *Given a tuple (t,c) = target, context*
  - *(apricot, jam)*
  - *(apricot, aardvark)*
- *Return probability that c is a real context word:*

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

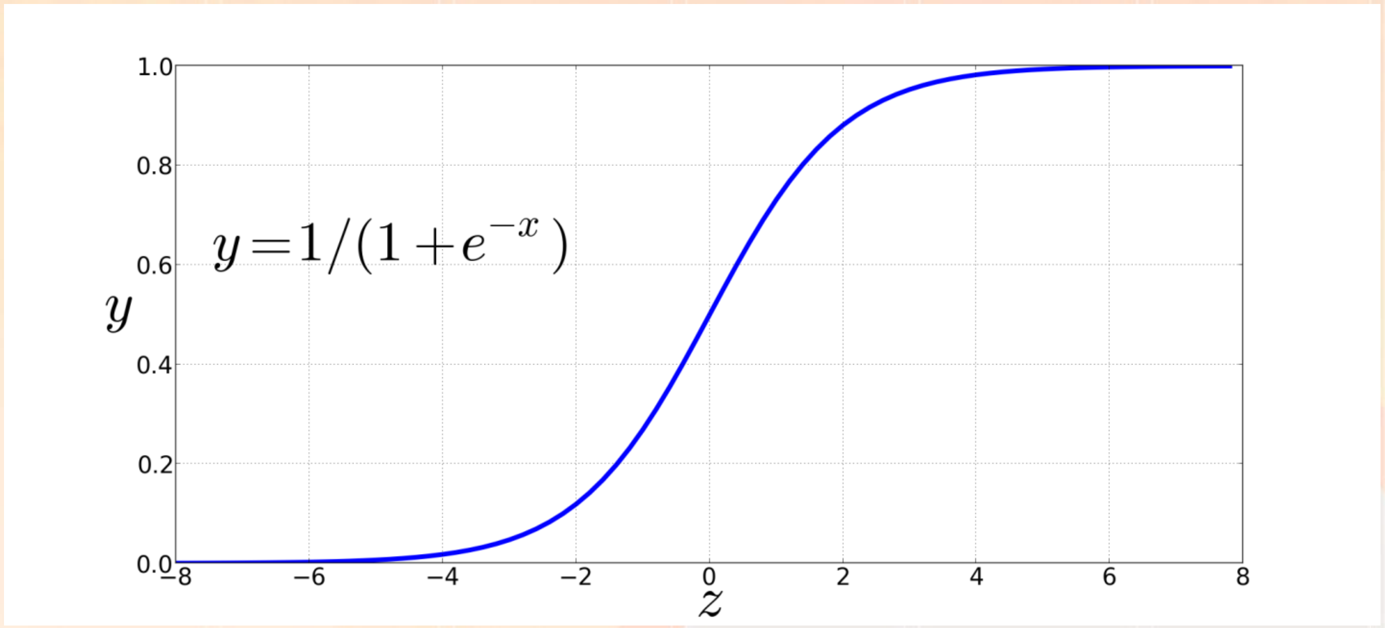
How to compute p(+|t,c)?

- *Intuition:*
  - *Words are likely to appear near similar words*
  - *Model similarity with dot-product!*
  - *Similarity(t,c)* $\propto t \cdot c$
- *Problem:*
  - *Dot product is not a probability!*
    - *(Neither is cosine)*

Turning dot product into **a probability**

- *The sigmoid lies between 0 and 1:*

$$\sigma(x) = \frac{1}{1+e^{-x}}$$



Turning dot product into a probability

$$P(+|t,c) = \frac{1}{1+e^{-t\cdot c}}$$

$$\begin{aligned} P(-|t,c) &= 1 - P(+|t,c) \\ &= \frac{e^{-t\cdot c}}{1+e^{-t\cdot c}} \end{aligned}$$

For all the context words:

- *Assume all context words are independent*

$$P(+|t,c_{1:k}) = \prod_{i=1}^{k} \frac{1}{1+e^{-t\cdot c_i}}$$

$$\log P(+|t,c_{1:k}) = \sum_{i=1}^{k} \log \frac{1}{1+e^{-t\cdot c_i}}$$

Skip-Gram Training Data

- *Training sentence:*

*... lemon, a tablespoon of **apricot** jam a pinch ...*

c1 c2 t c3 c4

- *Training data: input/output pairs centering on apricot*
- *Asssume a +/- 2 word window*

Skip-Gram Training

Training sentence:

... lemon, a tablespoon of ***apricot*** *jam a pinch ...*

c1 c2 t c3 c4

positive examples +

t	c
apricot	tablespoon
apricot	of
apricot	preserves
apricot	or

- For each positive example, we'll create *k* negative examples.
  - ➢ Using *noise* words
  - ➢ Any random word that isn't *t*

Skip-Gram Training

Training sentence:

*... lemon, a tablespoon of **apricot** jam a pinch ...*

c1 c2 t c3 c4

positive examples +

t	c
apricot	tablespoon
apricot	of
apricot	preserves
apricot	or

negative examples -

t	c	t	c
apricot	aardvark	apricot	twelve
apricot	puddle	apricot	hello
apricot	where	apricot	dear
apricot	coaxial	apricot	forever

Choosing noise words

- *Could pick w according to their unigram frequency P(w)*
- *More common to chosen then according to $p_\alpha(w)$*

$$P_\alpha(w) = \frac{count(w)^\alpha}{\sum_w count(w)^\alpha}$$

- *α= ¾ works well because it gives rare noise words slightly higher probability*

➢ *To show this, imagine two events p(a)=.99 and p(b) = .01:*

$$\begin{aligned} P_\alpha(a) &= \frac{.99^{.75}}{.99^{.75} + .01^{.75}} = .97 \\ P_\alpha(b) &= \frac{.01^{.75}}{.99^{.75} + .01^{.75}} = .03 \end{aligned}$$

Setup

- *Let's represent words as vectors of some length (say 300), randomly initialized.*
- *So we start with 300 * V random parameters*
- *Over the entire training set, we'd like to adjust those word vectors such that we*
  - *Maximize the similarity of the target word, context word pairs (t,c) drawn from the positive data*
  - *Minimize the similarity of the (t,c) pairs drawn from the negative data.*

Learning the classifier

- *Iterative process.*
  - *We'll start with 0 or random weights*
  - *Then adjust the word weights to*
    - *make the positive pairs more likely and*
    - *the negative pairs less likely*
  - *over the entire training set:*

HOW?

HOW???

Objective Criteria

HOW?

- *We want to maximize…*

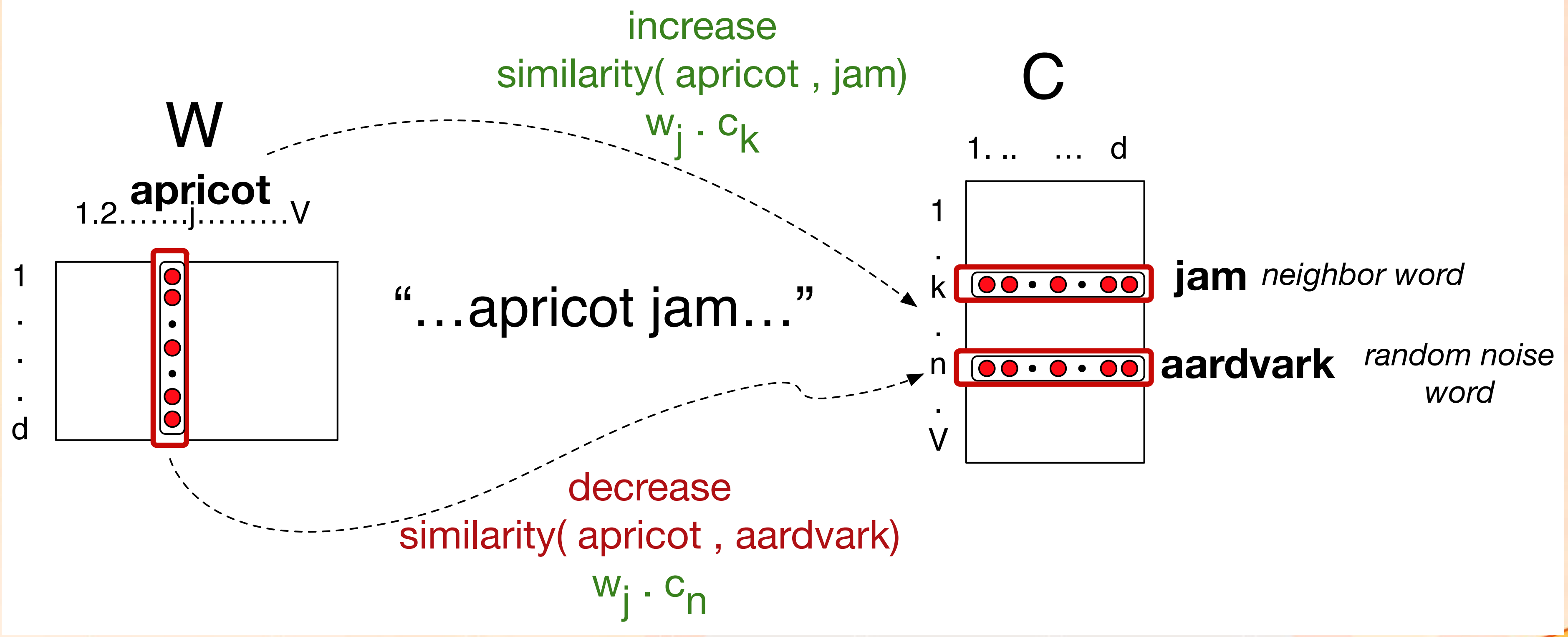
$$\sum_{(t,c)\in +} logP(+|t,c) + \sum_{(t,c)\in -} logP(-|t,c)$$

- *Maximize the + label for the pairs from the positive training data, and the – label for the pairs sample from the negative data.*

Focusing on one target word t:

$$
\begin{aligned}
L(\theta) &= \log P(+|t,c) + \sum_{i=1} \log P(-|t,n_i) \\
&= \log \sigma(c \cdot t) + \sum_{i=1}^{k} \log \sigma(-n_i \cdot t) \\
&= \log \frac{1}{1+e^{-c \cdot t}} + \sum_{i=1}^{k} \log \frac{1}{1+e^{n_i \cdot t}}
\end{aligned}
$$

Weight Matrixes



Skip-Gram Word2Vec Network Architecture

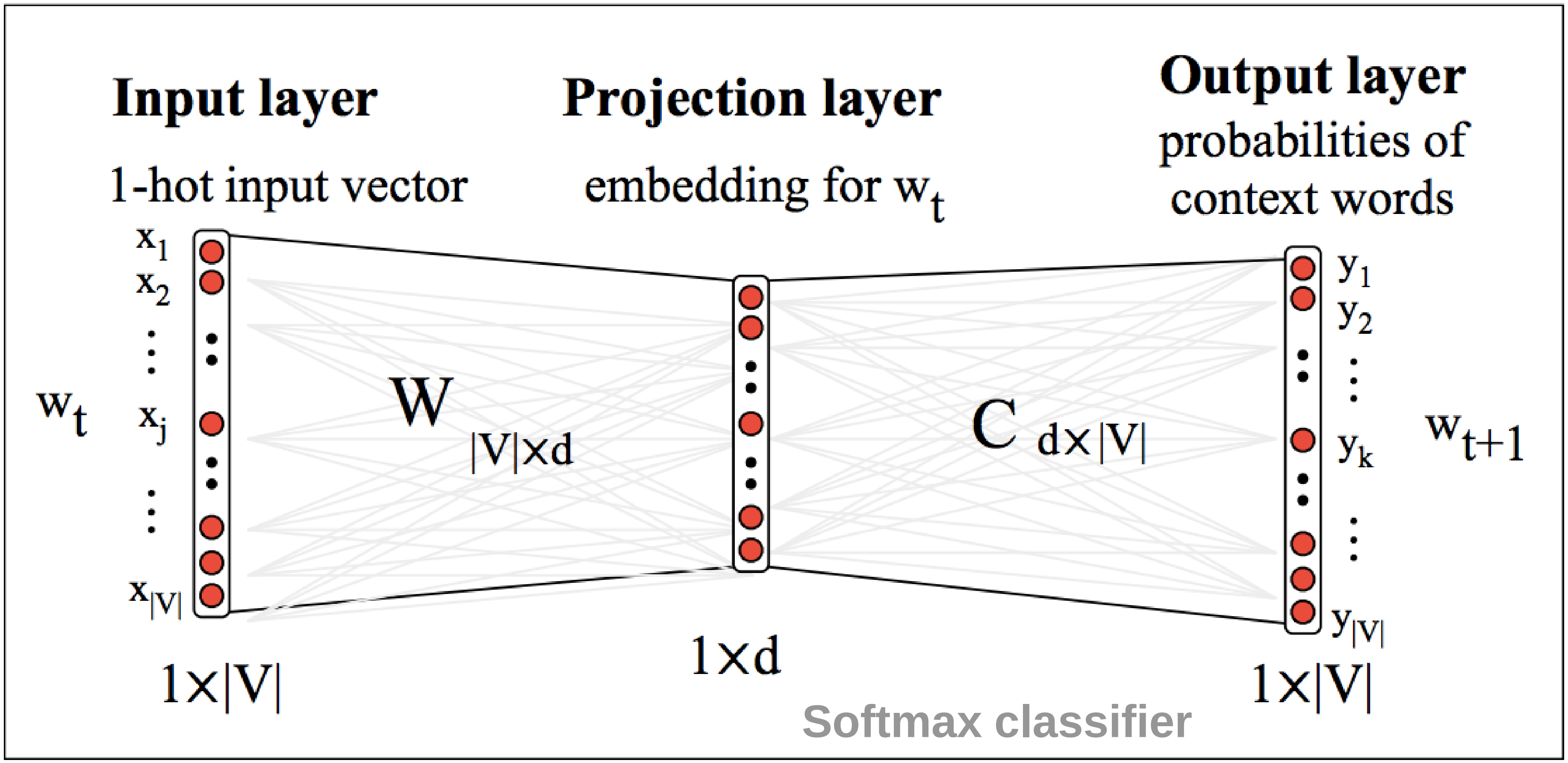


Figure 16.5 The skip-gram model viewed as a network (Mikolov et al. 2013, Mikolov et al. 2013a).

How to learn word2vec (skip-gram) embeddings

- *Start with **V** random 300-dimensional vectors as initial embeddings*
- *Use logistic regression (later discussed):*
  - *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*
  - ✓ *<u>Train the classifier</u> to distinguish these examples by slowly adjusting all the embeddings to improve the classifier performance*
  - *Throw away the classifier code and keep the embeddings.*

Train using gradient descent

- *Actually learns two separate embedding matrices W and C*
- *Can use W and throw away C, or merge them somehow*
- *But <u>how did we actually learn</u>????*

HOW???

Logistic Regression

- *In natural language processing, logistic regression is the baseline supervised machine learning algorithm for classification, and also has a very close relationship with neural networks.*
- *A neural network can be viewed as a series of logistic regression classifiers stacked on top of each other.*
- *Logistic regression can be used to classify an observation into one of two classes (like 'positive sentiment' and 'negative sentiment'), or into one of many classes*

Components of a probabilistic machine learning classifier:

A machine learning system for classification has four components:

1. ***A feature representation of the input**. For each input observation x(i), this will be a vector of features $x=[x_1,x_2,\ldots,x_n]$.*
2. *A **classification function** that computes $\hat{y}$, the estimated class, via p(y|x). Examples of classification functions: **sigmoid** or **softmax** functions;*
3. *An **objective function** for learning, usually involving minimizing errors on training examples. Examples of objective functions: **cross-entropy loss** function.*
4. *An **algorithm for optimizing the objective function**. We introduce the stochastic gradient descent algorithm.*

Logistic regression for classification

Logistic regression computes the estimated class by learning, <u>from a training set</u>, a vector of weights and a bias term.

- ➢ *Each **weight** w_i is a real number, and is associated with one of the input features x_i. The weight w_i represents how important that input feature is to the classification decision, and can be positive (meaning the feature is associated with the class) or negative (meaning the feature is not associated with the class).*
- ➢ *The **bias term** is another real number that's added to the weighted inputs.*

Classification with logistic regression

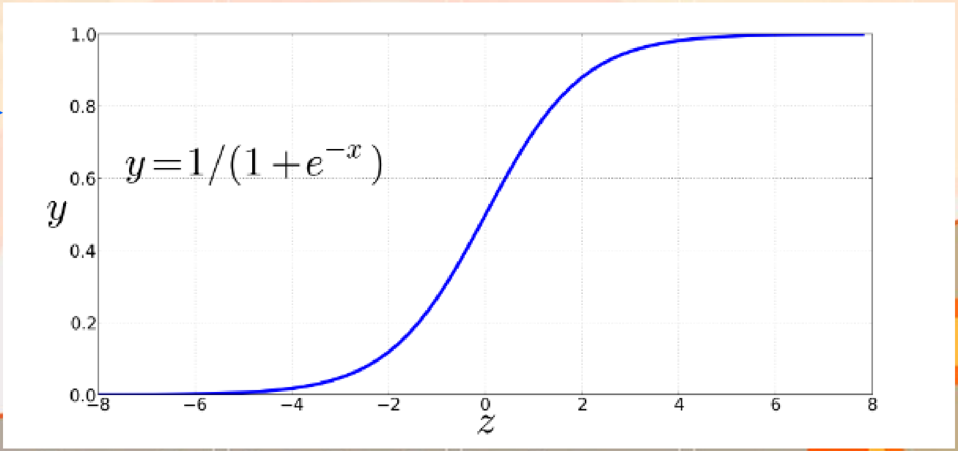
To make a classification decision on a **test instance**— after we've learned the weights in training— the classifier first multiplies each x_i by its weight w_i, sums up the weighted features, and adds the bias term b. The resulting single number z expresses the weighted sum of the evidence for the class.

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

Not a probability!!!

To create a probability, we'll pass z through the sigmoid function, $\sigma(z)$. The sigmoid function (named because it looks like an s) is also called the logistic function, and gives **logistic regression** its name!

Sigmoid function



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

What else is needed?

If we apply the sigmoid to the sum of the weighted features, we get a number between 0 and 1. To make it a probability, we just need to make sure that the two cases, p(y=1) and p(y=0), sum to 1.

$$P(y=1) = \sigma(w \times x + b) = \frac{1}{1 + e^{-(w \times x + +b)}}$$

$$P(y=0) = 1 - \sigma(w \times x + b) = \frac{e^{-(w \times x + +b)}}{1 + e^{-(w \times x + +b)}}$$

Now we have a classification function that given an instance *x* computes the probability *P(y=1|x)*. How do we make **a decision**? For a test instance *x*, we say yes if the Probability *P(y=1|x)* is more than .5, and no otherwise.
We call 0.5 the **decision boundary**:

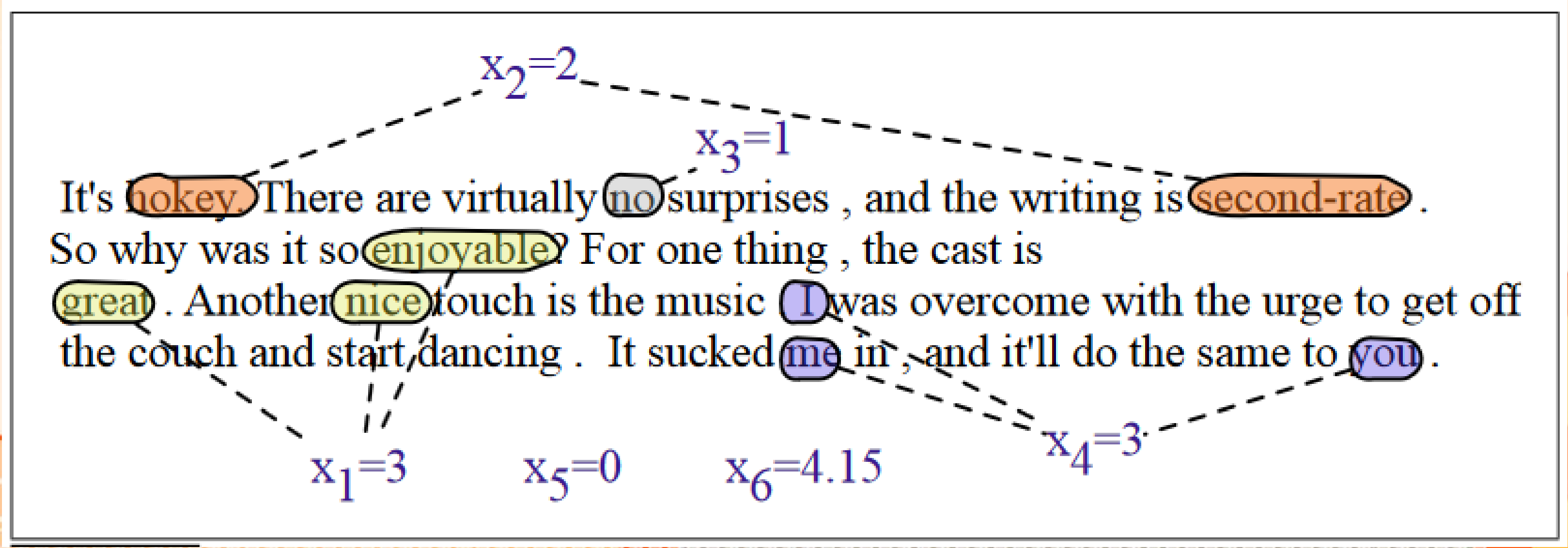
$$\hat{y} = \begin{cases} 1, & if\ P(y=1|x) > 0.5 \\ 0, & otherwise \end{cases}$$

Example

- Use logistic regression for sentiment classification.

STEP 1: extract features for the input vector *x*

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)	$\ln(64) = 4.15$


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 .

Example (cont)

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)	$\ln(64) = 4.15$

Assume we learned real-valued weight for each of these features, and that the 6 weights corresponding to the 6 features are [2.5,−5.0,−1.2,0.5, 2.0,0.7], while b= 0.1.

$$\begin{aligned}
p(+|x) = P(Y=1|x) &= \sigma(w\cdot x+b) \\
&= \sigma([2.5,-5.0,-1.2,0.5,2.0,0.7]\cdot[3,2,1,3,0,4.15]+0.1) \\
&= \sigma(1.805) \\
&= 0.86 \\
p(-|x) = P(Y=0|x) &= 1-\sigma(w\cdot x+b) \\
&= 0.14
\end{aligned}$$

But how are the parameters learned?

The *weights* and the *bias* of the logistic regression need to be learned from the training examples.
How are they learned?

We need two components:

1. The first is a metric for how close the current label ($\hat{y}$) is to the true gold label y.

➢ Rather than measure similarity, we usually talk about the opposite of this:

- The distance between the system output and the gold output, and we call this distance the **loss function** or the cost function.

2. The second thing we need is an **optimization algorithm** for iteratively updating the weights so as to minimize this loss function. The standard algorithm for this is gradient descent ; we'll introduce the **stochastic gradient descent** algorithm.

The cross-entropy loss function

What do we need? We need a loss function that expresses, for an observation *x*, **how close** the classifier output ($\hat{y}$ = $\sigma(w \times x + b)$) is to the correct output (y, which is 0 or 1).

➢ This is a loss function $L = (\hat{y}, y)$ which measures how much $\hat{y}$ differs from y

Possible loss function?

- A function that just takes the mean squared error between $\hat{y}$ and y:

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

It turns out that this MSE loss, which is very useful for some algorithms like linear regression, becomes harder to optimize (technically, non-convex), when it's applied to probabilistic classification.

Instead, we use a loss function that prefers the correct class labels of the training example **to be more likely**.

This is called ***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. The resulting loss function is the negative log likelihood loss, generally called the **cross entropy loss**.

The cross-entropy loss function - 2

Let's derive cross-entropy loss function, applied to a single observation x !!!

➢ We'd like to learn weights that maximize the probability of the correct label *p(y|x)*. Since there are only two discrete outcomes (1 or 0), this is a **Bernoulli distribution**, and we can express the probability *p(y|x)* that our classifier produces for one observation as:

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

❑ Now we take the log of both sides:

whatever values maximize a probability will also maximize the log of the probability

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

➢ In order to turn this into loss function (something that we need to minimize), we'll just flip the sign:

$$L_{CE} = -\log p(y|x) = -y log\hat{y} - (1-\mathrm{y})\log(1-\hat{y})$$

➢ Finally, we can plug in the definition of $\hat{y} = \sigma(w \cdot x + b)$ in the loss function:

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

Why is it called cross-entropy?

- 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 $\hat{y}$ (the closer it is to 1), the better the classifier;

  the lower $\hat{y}$ is (the closer it is to 0), the worse the classifier.

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

- 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). This loss function also insures that as probability of the correct answer is maximized, the probability of the incorrect answer is minimized; since the two sum to one, any increase in the probability of the correct answer is coming at the expense of the incorrect answer!

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

- It's called the cross-entropy loss, because Equation is also the formula for the cross-entropy between the true probability distribution y and our estimated distribution $\hat{y}$

Extend to the entire training set

Going from one example to the whole training set: we'll consider the notation $x^{(i)}$ and $y^{(i)}$ to refer to the *i*-th training features and training label, respectively.

❑ We make the assumption that the training examples are independent:

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

➢ We'll define the cost function for the whole dataset as the average loss for each example:

$$Cost(w,b) = \frac{1}{m}\sum_{i=1}^{m} L_{CE}(\hat{y}^{(i)}, y^{(i)}) =$$

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

$\uparrow$

This is what we want to minimize! BUT HOW?????

Gradient Descent

GOAL with gradient descent is to find the optimal weights &&& minimize the loss function we've defined for the model.

➢ Represent the fact that the loss function L is parameterized by the weights, which we'll refer to in machine learning in general as θ (in the case of logistic regression θ={w,b}). *Gradient Descent will:*

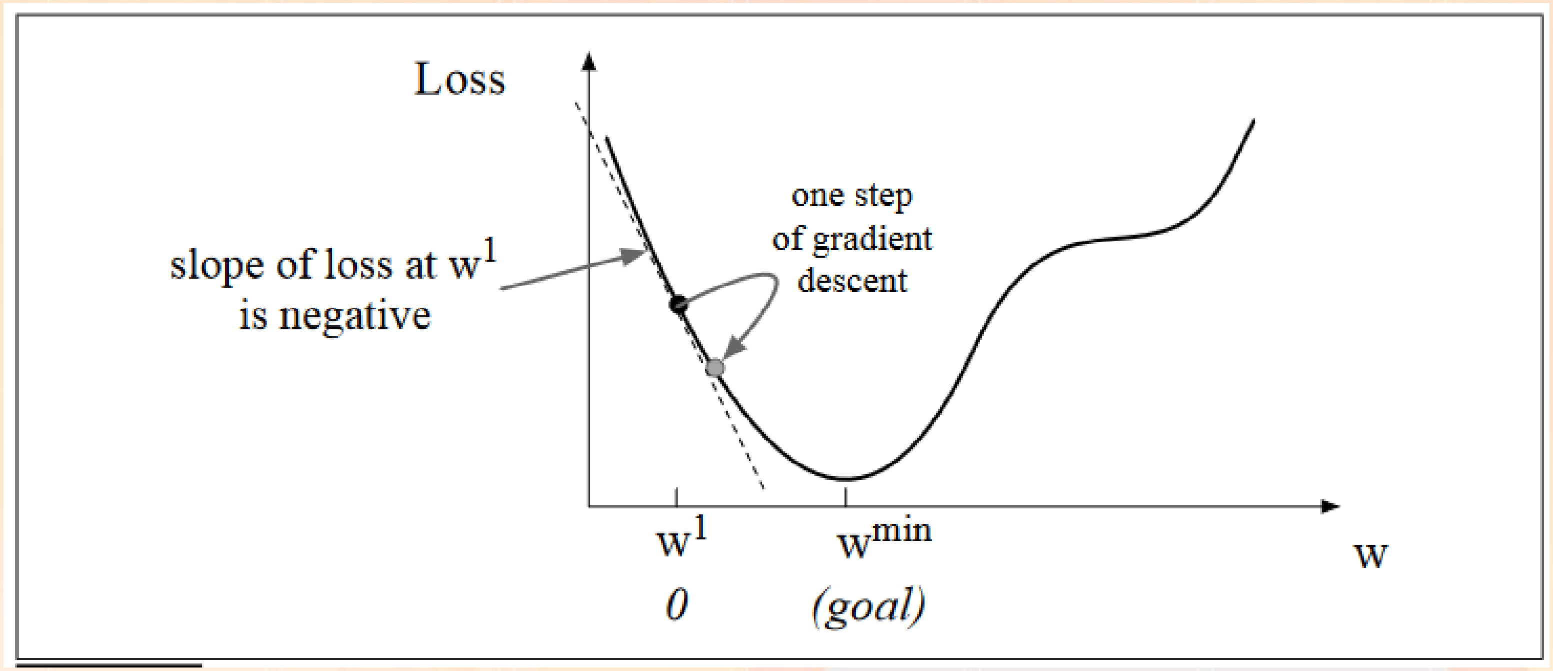
$$\hat{\theta} = \underset{\theta}{argmin}\frac{1}{m}\sum_{i=1}^{m} L_{CE}\left(y^{(i)}, x^{(i)}, \theta\right)$$

How shall we find the minimum of this (or any) loss function?
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 thus moving in the opposite direction.

HOW? ??

Intuition

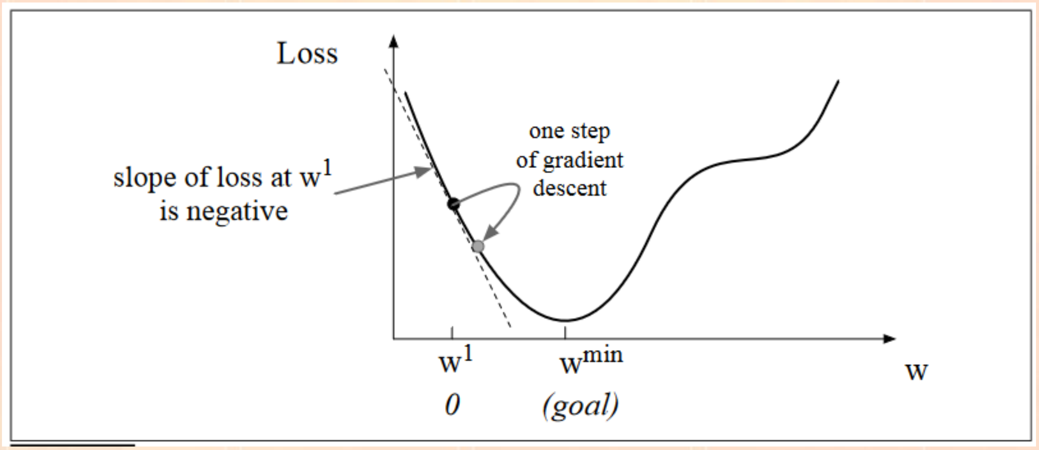
The intuition is that if you are hiking in a canyon and trying to descend most quickly down to the river at the bottom, you might look around yourself 360 degrees, find the direction where the ground is sloping the steepest, and walk downhill in that direction.


Loss
slope of loss at w^1 is negative
one step of gradient descent
w^1
0
w^{min}
(goal)
w

The first step in iteratively finding the minimum of this loss function, by moving *w* in the reverse direction from the slope of the function. Since the slope is negative, we need to move *w* in a positive direction, to the right.

Gradient

QUESTION: Given a random initialization of w at some value w_1, and assuming the loss function L happened to have the shape in Fig, we need the algorithm to tell us whether at the next iteration, we should move left (making w_2 smaller than w_1) or right (making w_2 bigger than w_1) to reach the minimum?

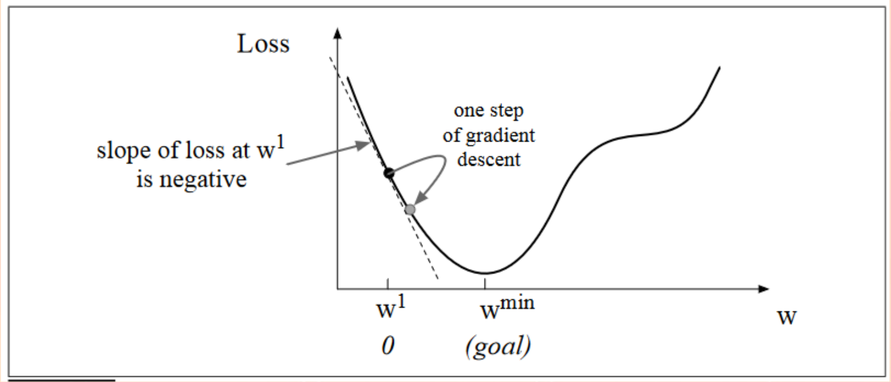


The gradient descent algorithm answers this question by finding the Gradient of the loss function at the current point and moving in the opposite direction. The gradient of a function of many variables is a vector pointing in the direction the greatest increase in a function. The gradient is a multi-variable generalization of the slope, so for a function of one variable like the one in Figure We can informally think of the gradient as the slope. The dotted line in Fig. shows the slope of this hypothetical loss function at point $w= w_1$ You can see that the slope of this dotted line is negative. Thus to find the minimum, gradient descent tells us to go in the opposite direction: Moving w in a positive direction.

Learning Rate

The magnitude of the amount to move in gradient descent is the value of the slope

$\frac{d}{dw} f(w,x)$ *weighted by a learning rate* η


Loss
slope of loss at w^1 is negative
one step of gradient descent
w^1
0
w^{min}
(goal)
w

A higher (faster) learning rate means that learning rate we should move w more on each step. The change we make in our parameter is the learning rate times the gradient (or the slope, in our single-variable example):

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

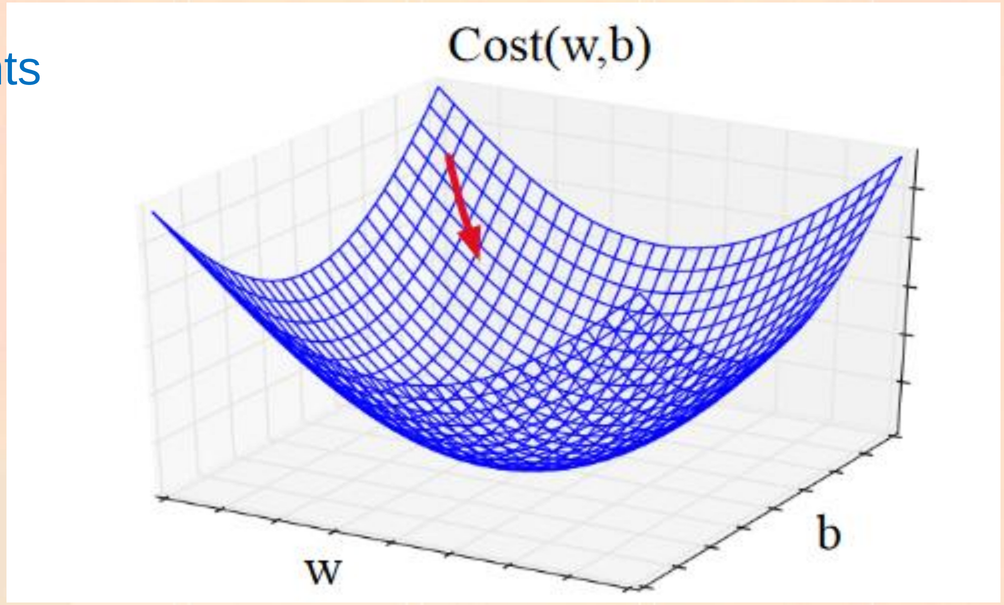
Let's extend the intuition from a loss function of one scalar variable *w* to many variables, because we don't just want to move left or right, we want to know where in the *N*-dimensional space (of the N parameters that make up *θ*) we should move.
NOTE: *The gradient is just such a vector; it expresses the directional components of the sharpest slope along each of those N dimensions.*

Gradient for Logistic Regression

Remember, in logistic regression, the parameter vector *w* has as many elements as the input feature vector *x* (we have a w_i for each x_i

- For each dimension/variable w_i in *w* (plus the bias *b*), the gradient will have a component that tells us the slope with respect to that variable.

QUESTION: "*How much would a small change in that variable w_i influence the total loss function L?*"



If for each dimension w_i we express the slope as a partial derivative:

$$\nabla_{\theta} L(f(x;\theta),y)) = \begin{bmatrix} \frac{\partial}{\partial w_1} L(f(x;\theta),y) \\ \frac{\partial}{\partial w_2} L(f(x;\theta),y) \\ \vdots \\ \frac{\partial}{\partial w_n} L(f(x;\theta),y) \end{bmatrix} \Rightarrow \theta_{t+1} = \theta_t - \eta \nabla L(f(x;\theta),y)$$

Gradient of Cross-Entropy Loss

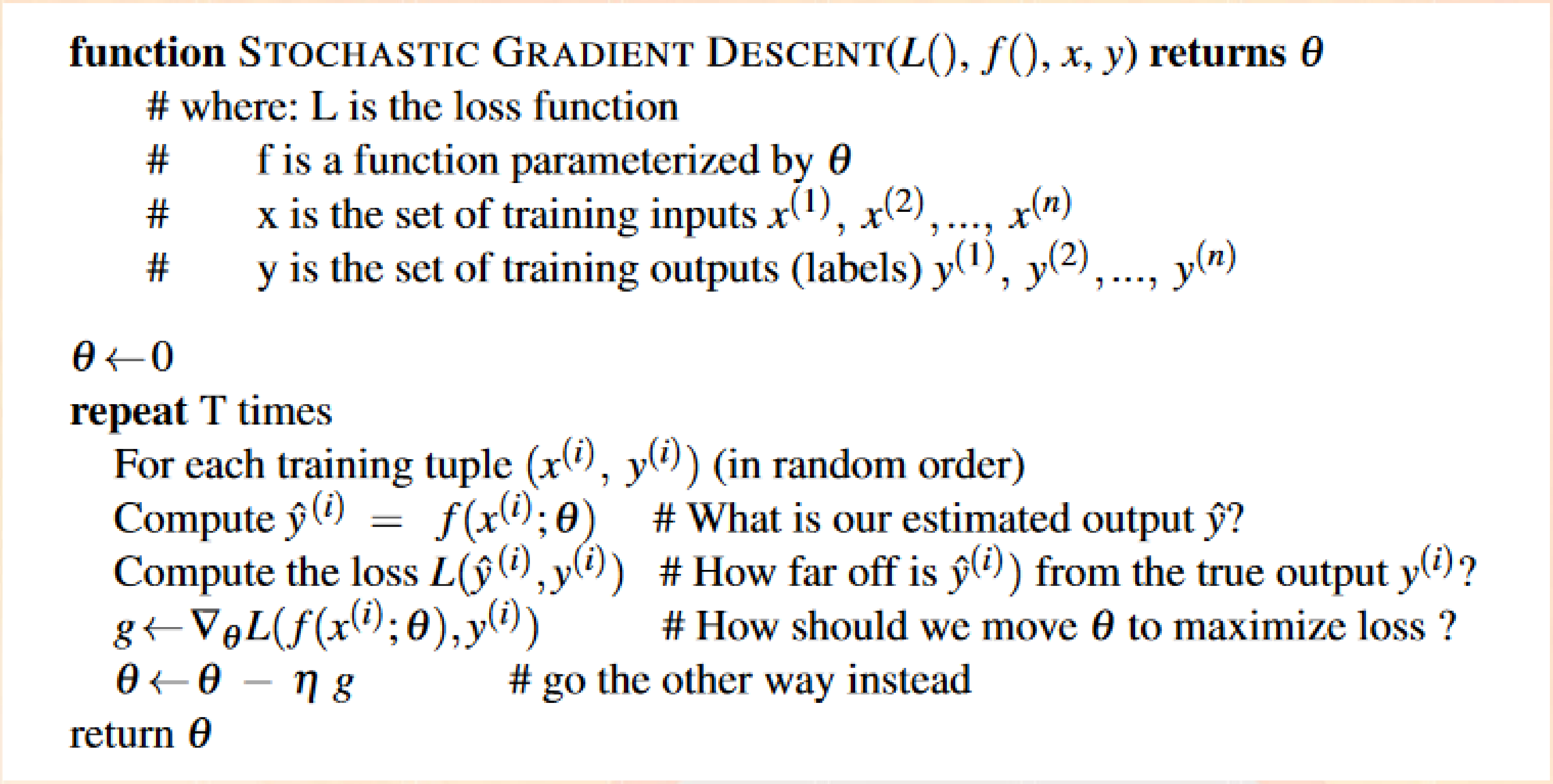
In order to update *θ:* $\theta_{t+1} = \theta_t - \eta \nabla L(f(x;\theta),y)$ we need to compute the gradient of L_{CE}

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

Which is:

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

The Stochastic Gradient Descent Algorithm



```
function STOCHASTIC GRADIENT DESCENT($L()$, $f()$, $x$, $y$) returns $\theta$
     # where: L is the loss function
     #       f is a function parameterized by $\theta$
     #       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 $\theta$ to maximize loss ?
   $\theta \leftarrow \theta - \eta g$          # go the other way instead
return $\theta$
```

Evaluate embeddings

- *Have humans rate the semantic similarity of a large set of word pairs.*
  - *(dog, canine): 10; (dog, cat): 7; (dog, carrot): 3; (dog, knife): 1*
- ➢ *Compute vector-space similarity of each pair of embeddings.*
- ❑ *Compute correlation coefficient (Pearson or Spearman) between human and machine ratings.*

Evaluating embeddings

- *Compare to human scores on word similarity-type tasks:*
  - *WordSim-353 (Finkelstein et al., 2002)*
  - *SimLex-999 (Hill et al., 2015)*
  - *Stanford Contextual Word Similarity (SCWS) dataset (Huang et al., 2012)*
  - *TOEFL dataset: Levied is closest in meaning to: imposed, believed, requested, correlated*

Properties of embeddings

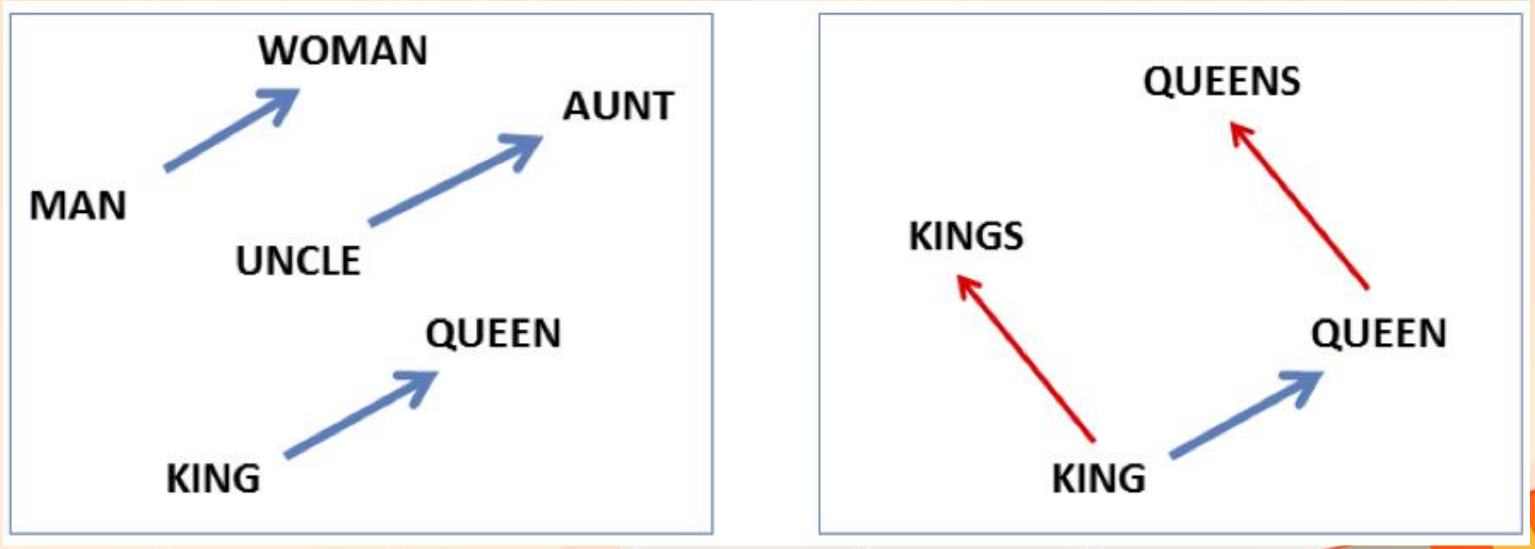
Similarity depends on window size C

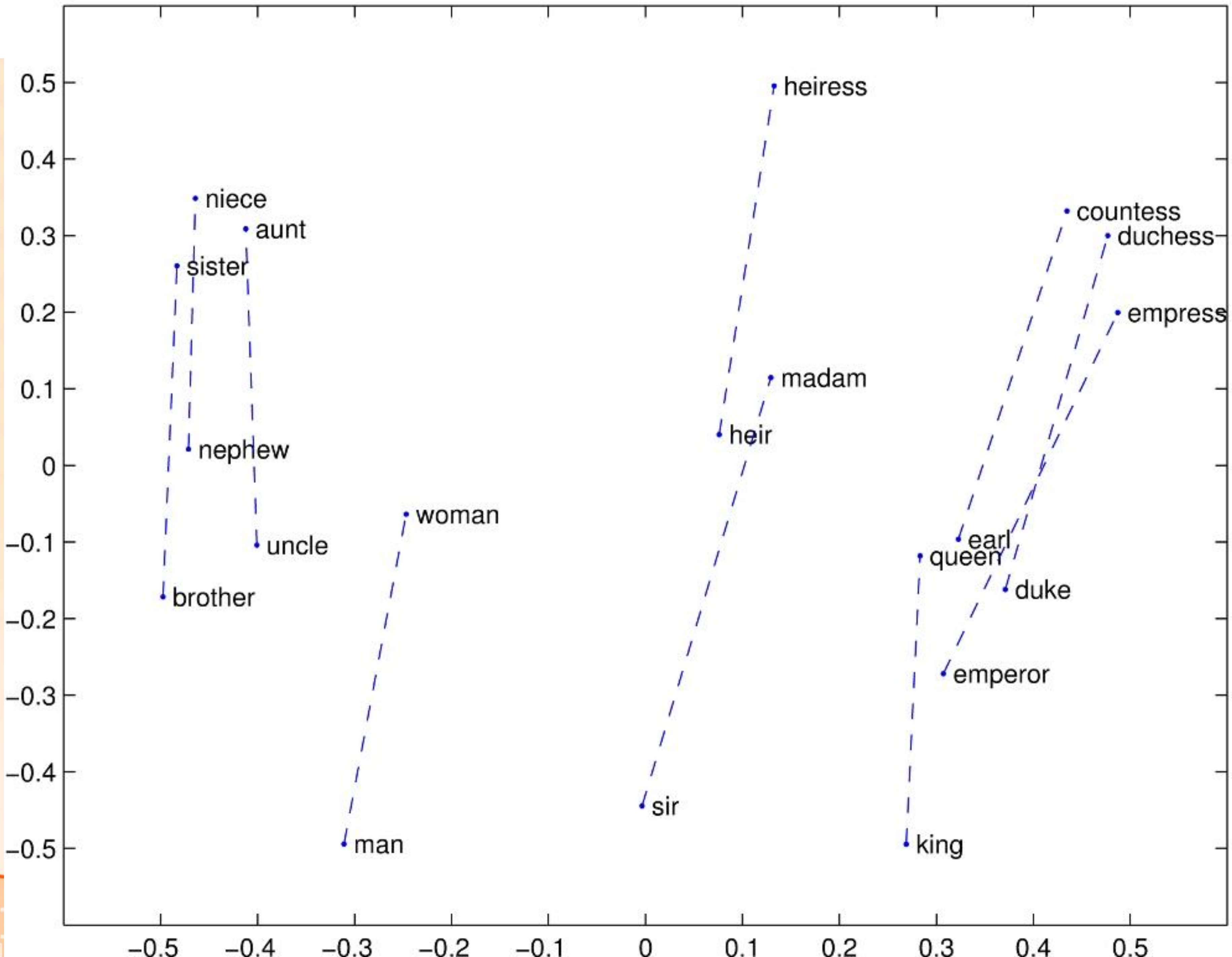
- *C = ±2 The nearest words to Hogwarts:*
  - *Sunnydale*
  - *Evernight*
- *C = ±5 The nearest words to Hogwarts:*
  - *Dumbledore*
  - *Malfoy*
  - *halfblood*

Analogy: Embeddings capture relational meaning!

vector('king') - vector('man') + vector('woman') ≈ vector('queen')

vector('Paris') - vector('France') + vector('Italy') ≈ vector('Rome')





heiress
niece
aunt
sister
countess
duchess
empress
madam
heir
nephew
woman
uncle
earl
queen
duke
brother
emperor
sir
man
king
0.5
0.4
0.3
0.2
0.1
0
−0.1
−0.2
−0.3
−0.4
−0.5
−0.5
−0.4
−0.3
−0.2
−0.1
0
0.1
0.2
0.3
0.4
0.5

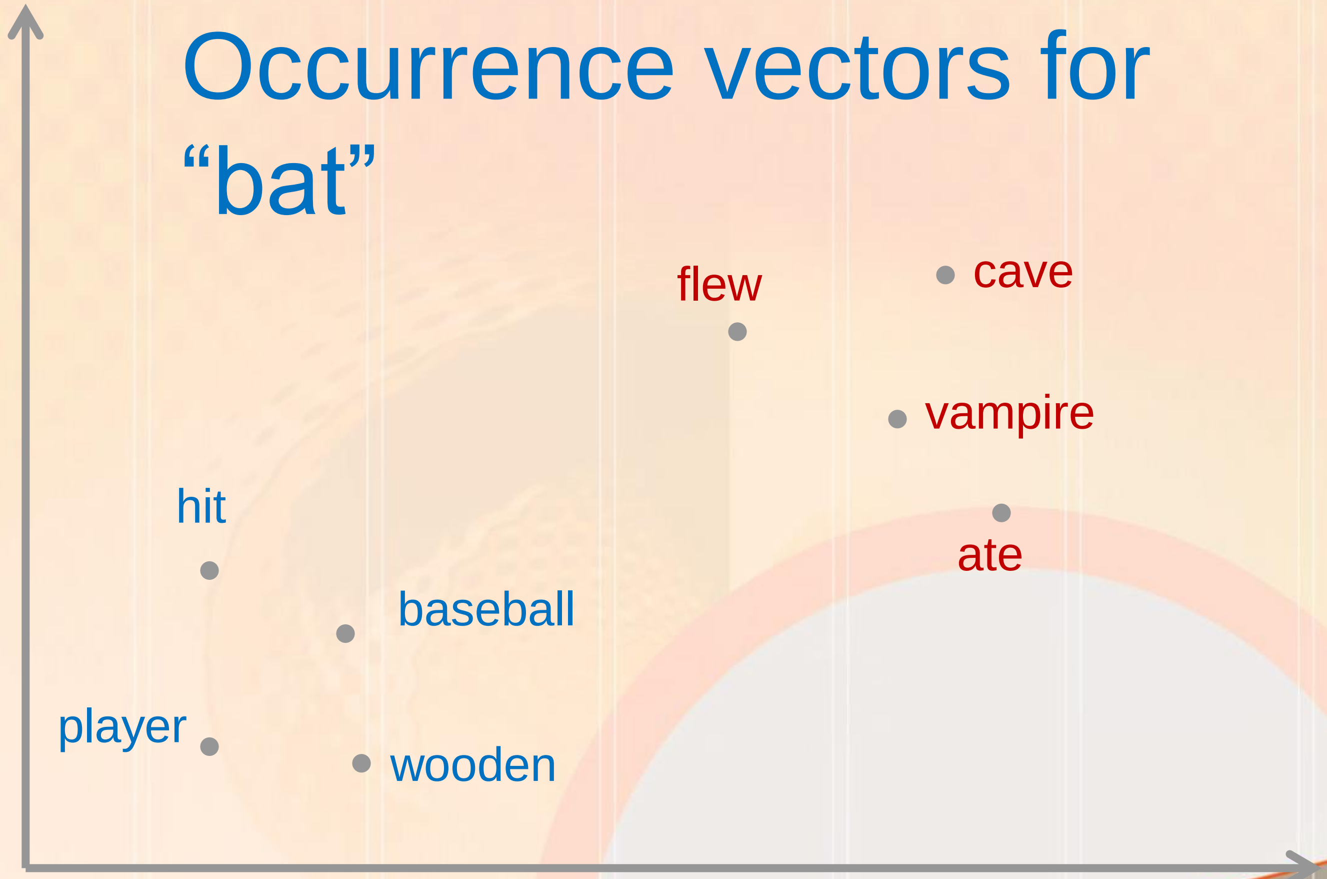
Vector-Space Word Sense Induction (WSI)

- *Create a context-vector for* ***each individual occurrence*** *of the target word, w.*

- *Cluster these vectors into k groups.*

- *Assume each group represents a "sense" of the word and compute a vector for this sense by taking the mean of each cluster.*

Sample Word Sense Induction

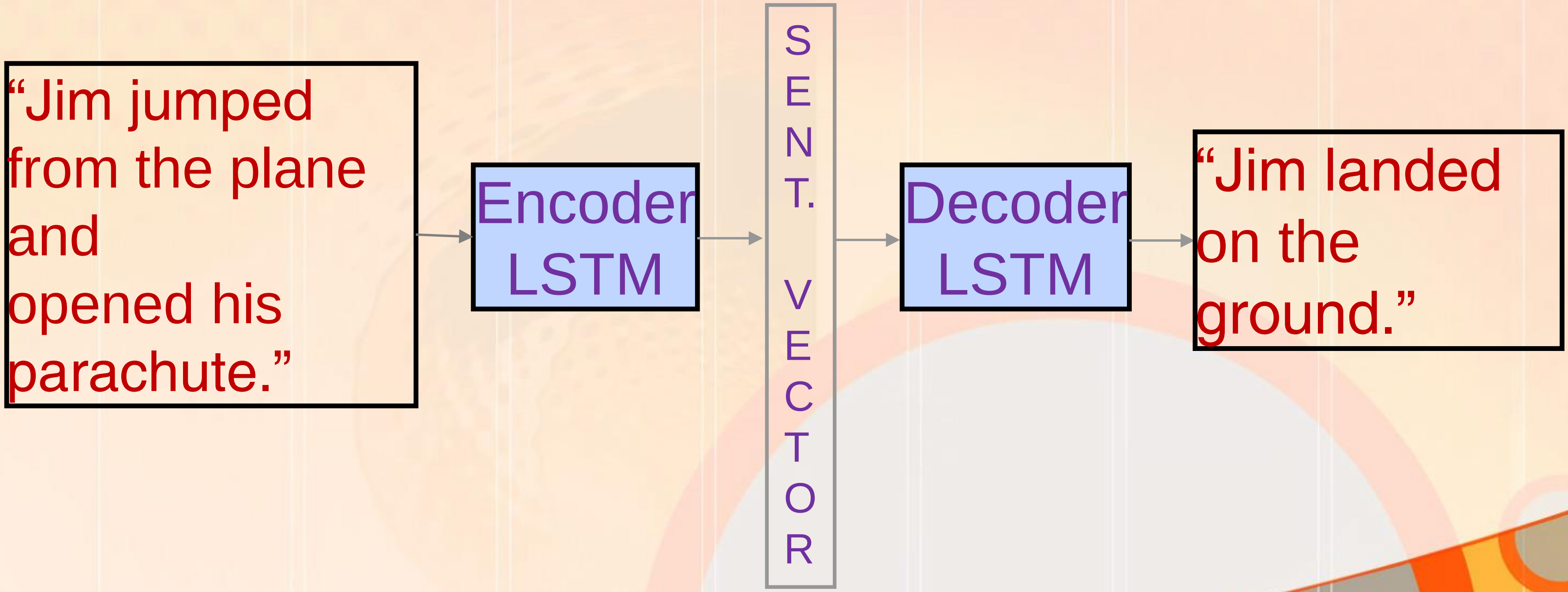


Compositional Vector Semantics

- *Compute vector meanings of phrases and sentences by combining (composing) the vector meanings of its words.*
- *Simplest approach is to use vector addition or component-wise multiplication to combine word vectors.*
- *Evaluate on human judgements of sentence-level semantic similarity (semantic textual similarity, STS, SemEval competition).*

Sentence-Level Neural Language Models

- *"Skip-Thought Vectors"* *(Kiros et al., NIPS 2015)*
  - *Use LSTMs to encode whole sentences into lower-dimensional vectors.*
  - *Vectors trained to predict previous and next sentences.*

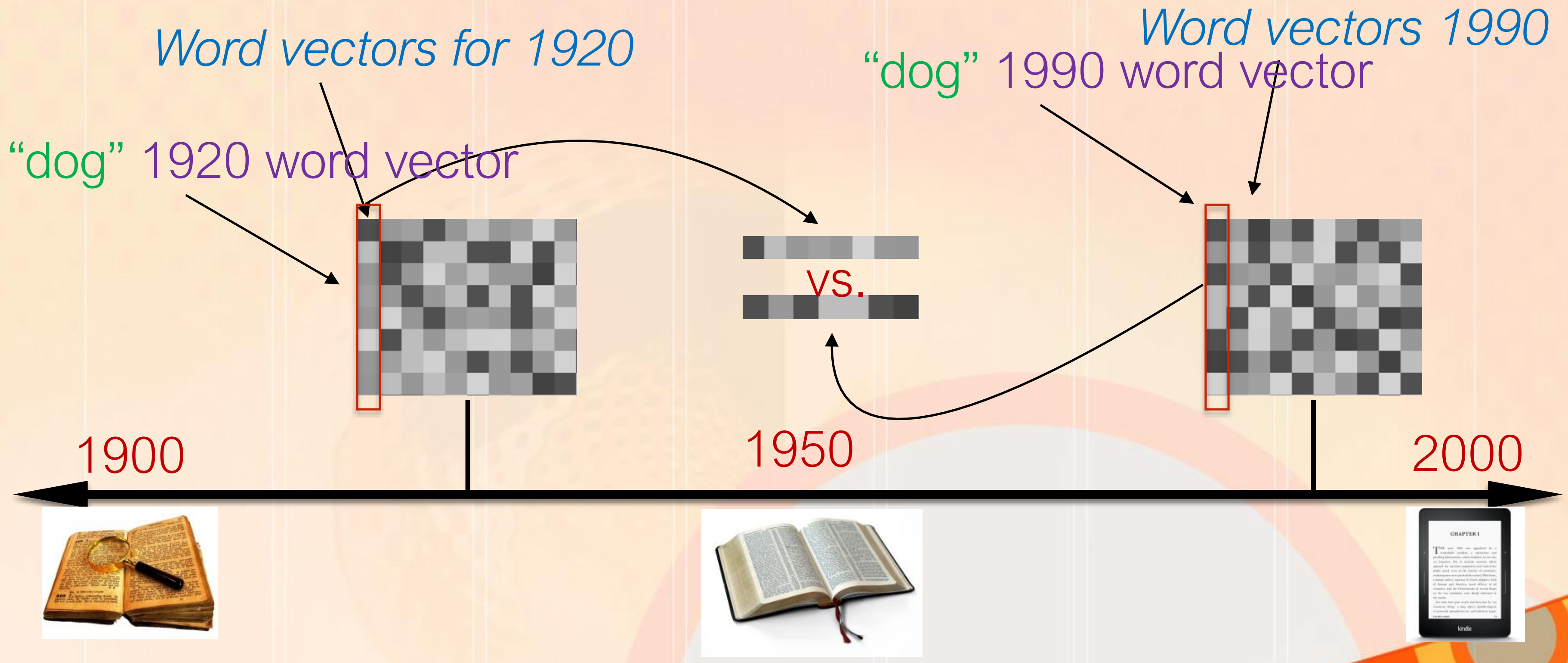

"Jim jumped from the plane and opened his parachute."
Encoder LSTM
SENT. VECTOR
Decoder LSTM
"Jim landed on the ground."

Embeddings can help study word history!

- *Train embeddings on old books to study changes in word meaning!!*

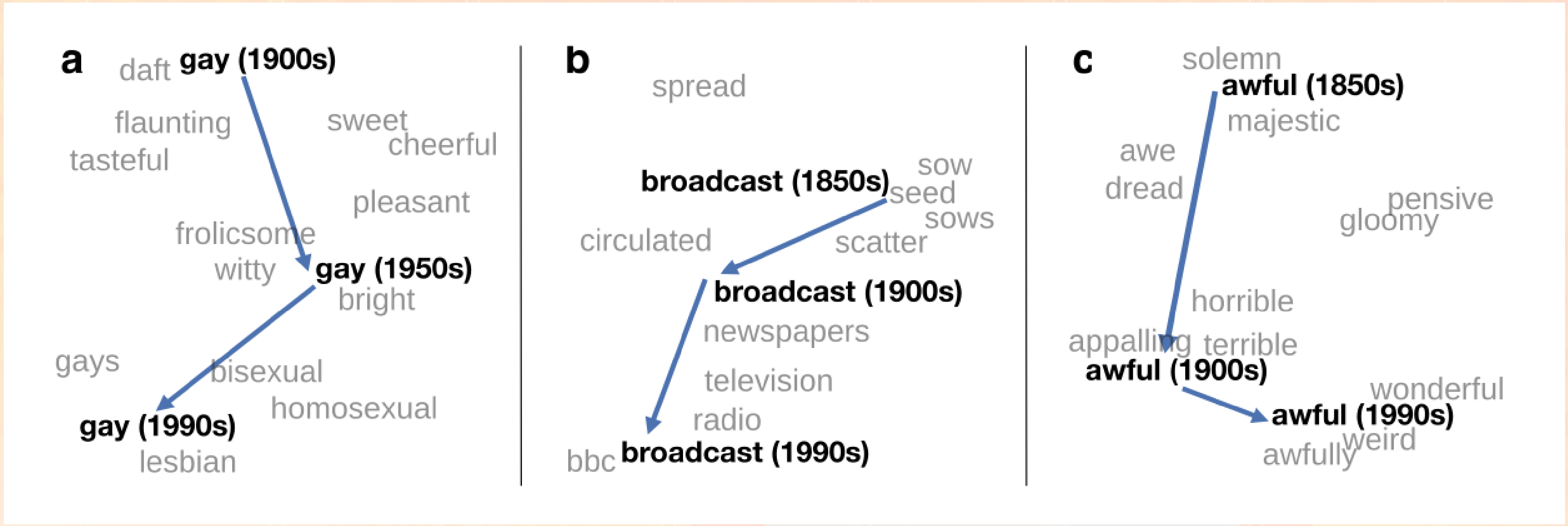
Will Hamilton

Diachronic word embeddings for studying language change!



Visualizing changes

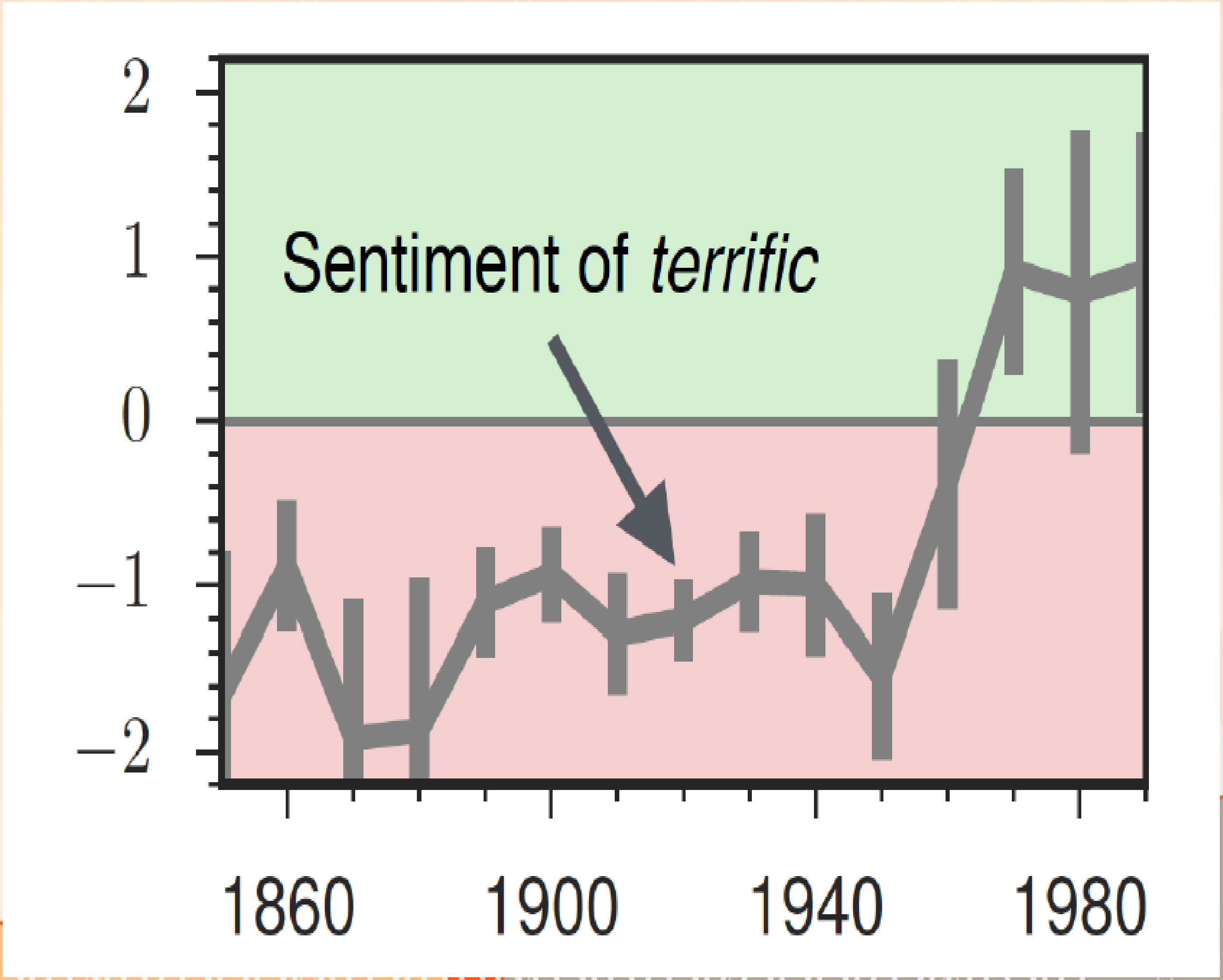
Project 300 dimensions down into 2



~30 million books, 1850-1990, Google Books data

The evolution of sentiment words

Negative words change faster than positive words



Embeddings reflect cultural bias

Bolukbasi, Tolga, Kai-Wei Chang, James Y. Zou, Venkatesh Saligrama, and Adam T. Kalai. "Man is to computer programmer as woman is to homemaker? debiasing word embeddings." In *Advances in Neural Information Processing Systems*, pp. 4349-4357. 2016.

- *Ask "Paris : France :: Tokyo : x"*
  - *x = Japan*
- *Ask "father : doctor :: mother : x"*
  - *x = nurse*
- *Ask "man : computer programmer :: woman : x"*
  - *x = homemaker*

Embeddings reflect cultural bias

Caliskan, Aylin, Joanna J. Bruson and Arvind Narayanan. 2017. Semantics derived automatically from language corpora contain human-like biases. Science 356:6334, 183-186.

- *Implicit Association test (Greenwald et al 1998): How associated are*
  - *concepts (flowers, insects) & attributes (pleasantness, unpleasantness)?*
  - *Studied by measuring timing latencies for categorization.*
- *Psychological findings on US participants:*
  - *African-American names are associated with unpleasant words (more than European-American names)*
  - *Male names associated more with math, female names with arts*
  - *Old people's names with unpleasant words, young people with pleasant words.*
- *Caliskan et al. replication with embeddings:*
  - *African-American names (Leroy, Shaniqua) had a higher GloVe cosine with unpleasant words (abuse, stink, ugly)*
  - *European American names (Brad, Greg, Courtney) had a higher cosine with pleasant words (love, peace, miracle)*
- *Embeddings reflect and replicate all sorts of pernicious biases.*

Directions

- *Debiasing algorithms for embeddings*
  - *Bolukbasi, Tolga, Chang, Kai-Wei, Zou, James Y., Saligrama, Venkatesh, and Kalai, Adam T. (2016). Man is to computer programmer as woman is to homemaker? debiasing word embeddings. In Advances in Neural Information Processing Systems, pp. 4349–4357.*
- *Use embeddings as a historical tool to study bias*