

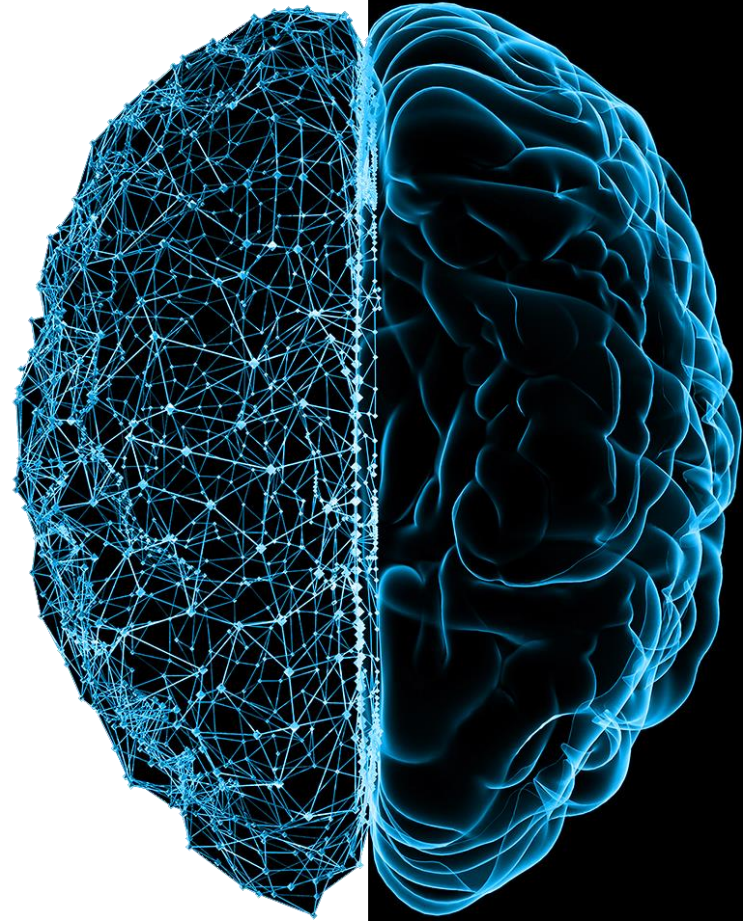
Procesamiento de Lenguaje Natural

Clase 11 – Semántica Vectorial (I)

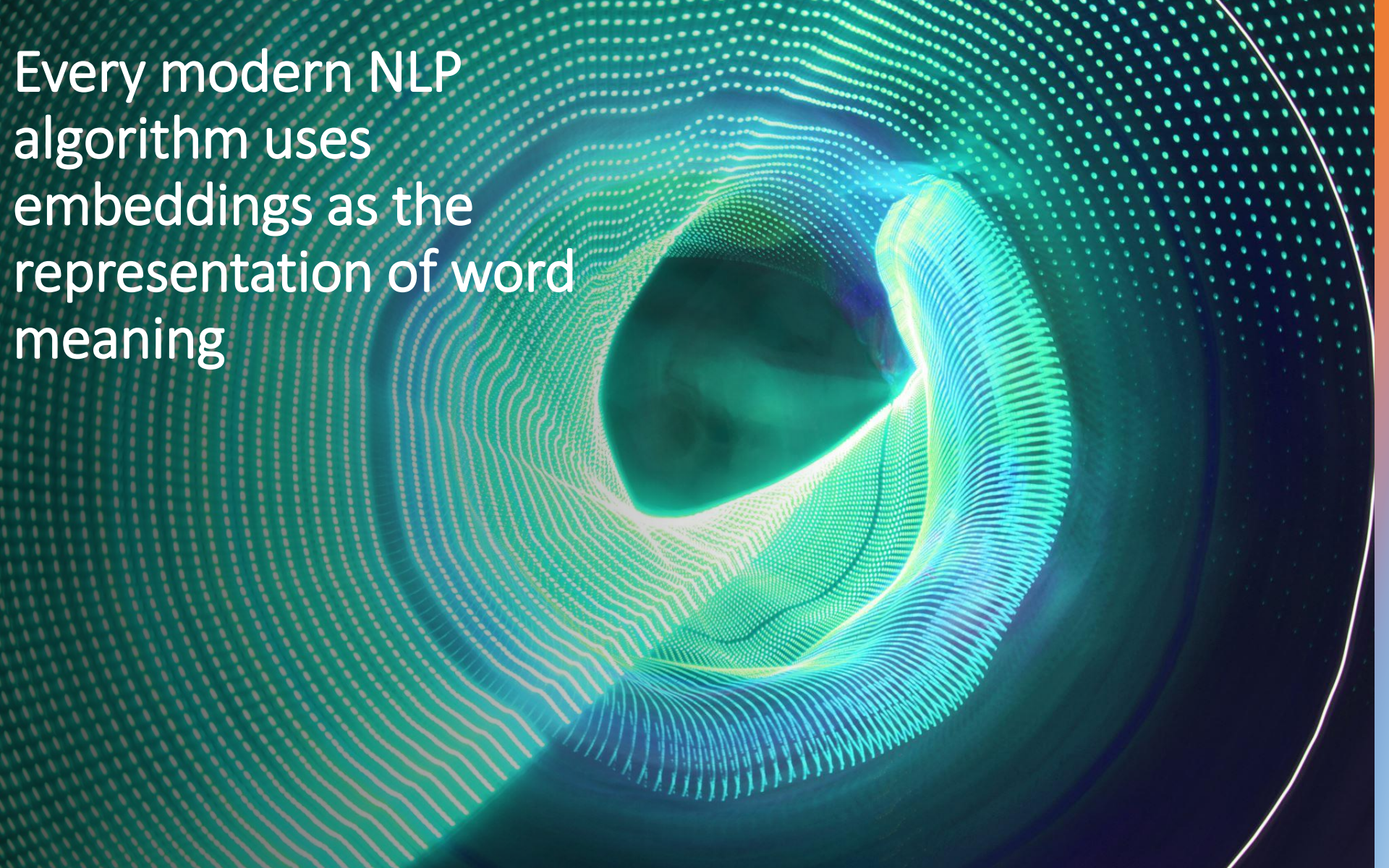
Ph.D. Rubén Manrique

rf.manrique@uniandes.edu.co

Maestría en Ingeniería de Sistemas
y Computación



Every modern NLP
algorithm uses
embeddings as the
representation of word
meaning



Defining meaning as a point in space based on distribution

Each word = a vector (not just "good" or " w_{45} ")

Similar words are "**nearby in semantic space**"

We build this space automatically by seeing which words are **nearby in text**



We'll discuss 2 kinds of embeddings

tf-idf

- Information Retrieval workhorse!
- A common baseline model
- **Sparse** vectors
- Words are represented by (a simple function of) the **counts** of nearby words

Word2vec

- **Dense** vectors
- Representation is created by training a classifier to **predict** whether a word is likely to appear nearby
- Later we'll discuss extensions called **contextual embeddings**

Sparse versus dense vectors

tf-idf (or PMI) vectors are

- **long** (length $|V| = 20,000$ to $50,000$)
- **sparse** (most elements are zero)

Alternative: learn 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 (fewer weights to tune)
- Dense vectors may **generalize** better than explicit counts
- Dense vectors 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**

Common methods for getting short dense vectors

“Neural Language Model”-inspired models

- Word2vec (skipgram, CBOW), GloVe

Singular Value Decomposition (SVD)

- A special case of this is called LSA – Latent Semantic Analysis

Alternative to these "static embeddings":

- Contextual Embeddings (ELMo, BERT)
- Compute distinct embeddings for a word in its context
- Separate embeddings for each token of a word

Capa de salida representación

- La capa de salida puede o no estar con el parámetro bias, pero si tiene pesos. Denotaremos los pesos de la capa de salida como la matriz $\mathbf{U} \in \mathbb{R}^{n_2 \times n_1}$, la salida $\mathbf{z} \in \mathbb{R}^{n_2}$ se calcula como:

$$\mathbf{z} = \mathbf{U}\mathbf{h}$$

- Sin embargo, \mathbf{z} no puede ser la salida del clasificador, ya que es un vector de valores reales números, mientras que lo que necesitamos para la clasificación es un vector de probabilidades.
- **Softmax:** codifica una distribución de probabilidad.

$$\text{softmax}(z_i) = \frac{e^{z_i}}{\sum_{j=1}^K e^{z_j}}$$

Para el ejemplo $K = n_2$?

Simple static embeddings you can download!

Word2vec (Mikolov et al)

<https://code.google.com/archive/p/word2vec/>

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 provides various options. We'll do:

skip-gram with negative sampling (SGNS)

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*"?

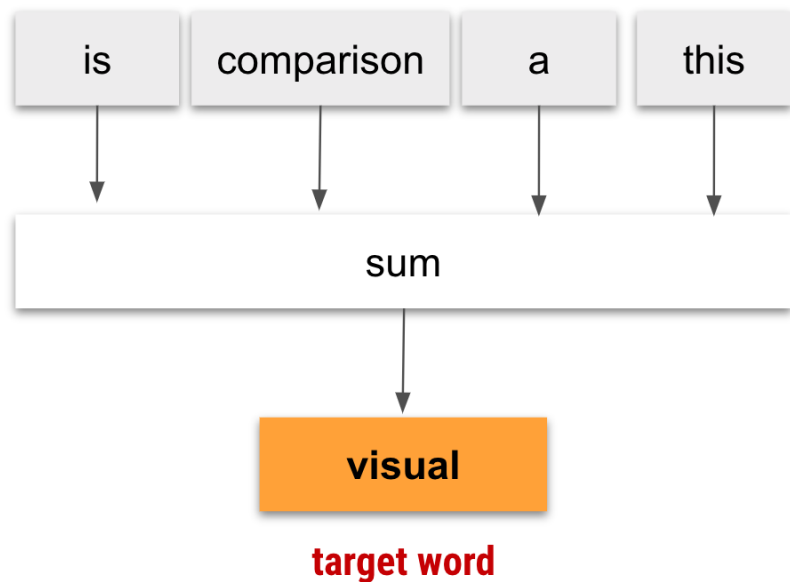
We don't actually care about this task

- But we'll take the learned classifier weights as the word embeddings

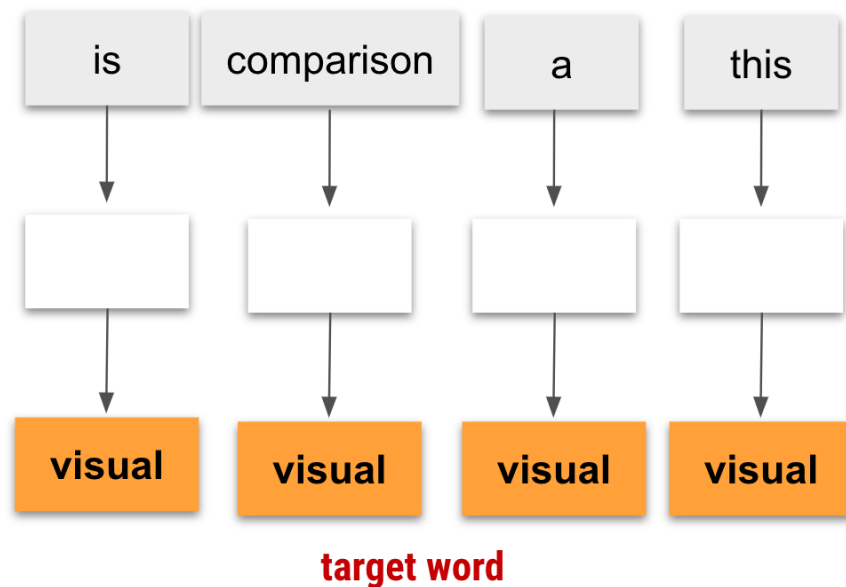
Big idea: **self-supervision**:

- A word c that occurs near *apricot* in the corpus acts as the gold "correct answer" for supervised learning
- No need for human labels
- Bengio et al. (2003); Collobert et al. (2011)

CBOW



SkipGram



By: Kavita Ganesan

This is a visual comparison

Approach: predict if candidate word c is a "neighbor"

1. Treat the target word t 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 (or neural network) to train a classifier to distinguish those two cases
4. Use the learned weights as the embeddings

Skip-Gram Training Data

Assume a ± 2 word window, given training sentence:

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

c1 c2 c3 c4

[target]

Skip-Gram Classifier

(assuming a +/- 2 word window)

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

c1 c2 [target] c3 c4

Goal: train a classifier that is given a candidate (**w**ord, **c**ontext) pair

(apricot, jam)

(apricot, aardvark)

...

And assigns each pair a probability:

$$P(+ | w, c)$$

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

Similarity is computed from dot product

Remember: two vectors are similar if they have a high dot product

- Cosine is just a normalized dot product

So:

- $\text{Similarity}(w, c) \propto w \cdot c$

We'll need to normalize to get a probability

- (cosine isn't a probability either)

Turning dot products into probabilities

$$\text{Sim}(w, c) \approx w \cdot c$$

To turn this into a probability

We'll use the sigmoid from logistic regression:

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

$$\begin{aligned} P(-|w, c) &= 1 - P(+|w, c) \\ &= \sigma(-c \cdot w) = \frac{1}{1 + \exp(c \cdot w)} \end{aligned}$$

How Skip-Gram Classifier in **testing** computes $P(+ | w, c)$

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

This is for one context word, but we have lots of context words.
We'll assume independence and just multiply them:

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

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

Skip-gram classifier: summary

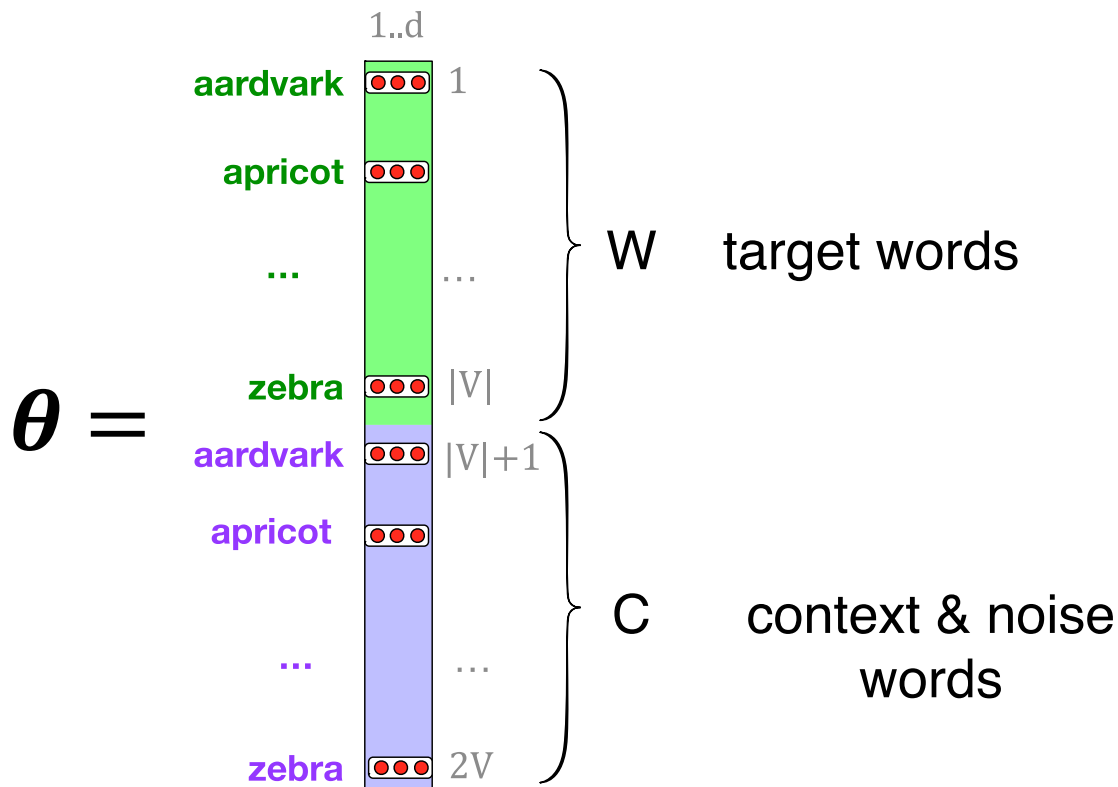
A probabilistic classifier, given

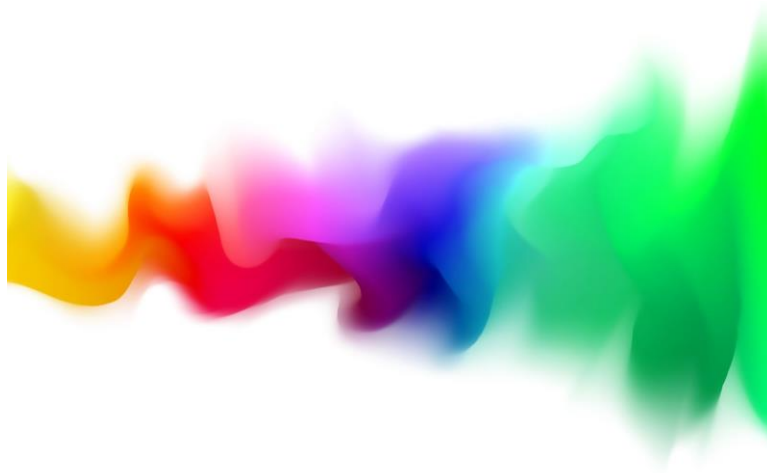
- a test target word w
- its context window of L words $c_{1:L}$

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.

These embeddings we'll need: a set for w , a set for c





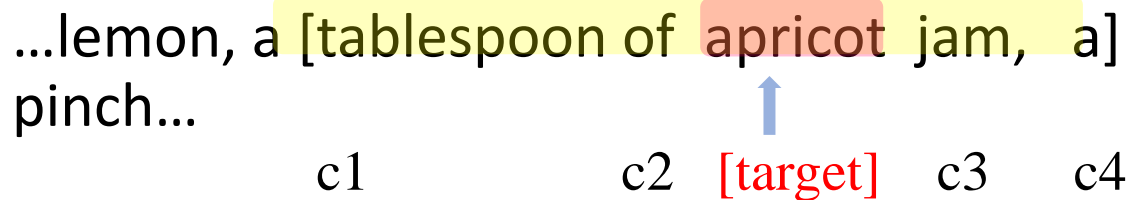
Training



Skip-Gram Training data

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

c1 c2 [target] c3 c4



positive examples +

t

c

apricot tablespoon

apricot of

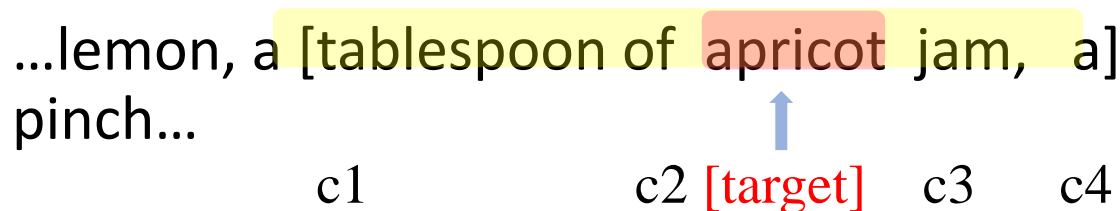
apricot jam

apricot a

Skip-Gram Training data

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

c1 c2 [target] c3 c4



positive examples +

t

c

apricot tablespoon

apricot of

apricot jam

apricot a

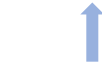
For each positive
example we'll grab k
negative examples.

Skip-Gram Training data

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

c1

c2



[target]

c3

c4

positive examples +

t	c
apricot	tablespoon
apricot	of
apricot	jam
apricot	a

negative examples -

t	c	t	c
apricot	aardvark	apricot	seven
apricot	my	apricot	forever
apricot	where	apricot	dear
apricot	coaxial	apricot	if

Word2vec: how to learn vectors

Given the set of positive and negative training instances, and an initial set of embedding vectors

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_{pos}) drawn from the positive data
- **Minimize** the similarity of the (w, c_{neg}) pairs drawn from the negative data.

Loss function for one w with c_{pos} , $c_{neg1} \dots c_{negk}$

Maximize the similarity of the target with the actual context words, and minimize the similarity of the target with the k negative sampled non-neighbor words.

$$\begin{aligned} L_{CE} &= -\log \left[P(+|w, c_{pos}) \prod_{i=1}^k P(-|w, c_{neg_i}) \right] \\ &= - \left[\log P(+|w, c_{pos}) + \sum_{i=1}^k \log P(-|w, c_{neg_i}) \right] \\ &= - \left[\log P(+|w, c_{pos}) + \sum_{i=1}^k \log (1 - P(+|w, c_{neg_i})) \right] \\ &= - \left[\log \sigma(c_{pos} \cdot w) + \sum_{i=1}^k \log \sigma(-c_{neg_i} \cdot w) \right] \end{aligned}$$

Learning the classifier

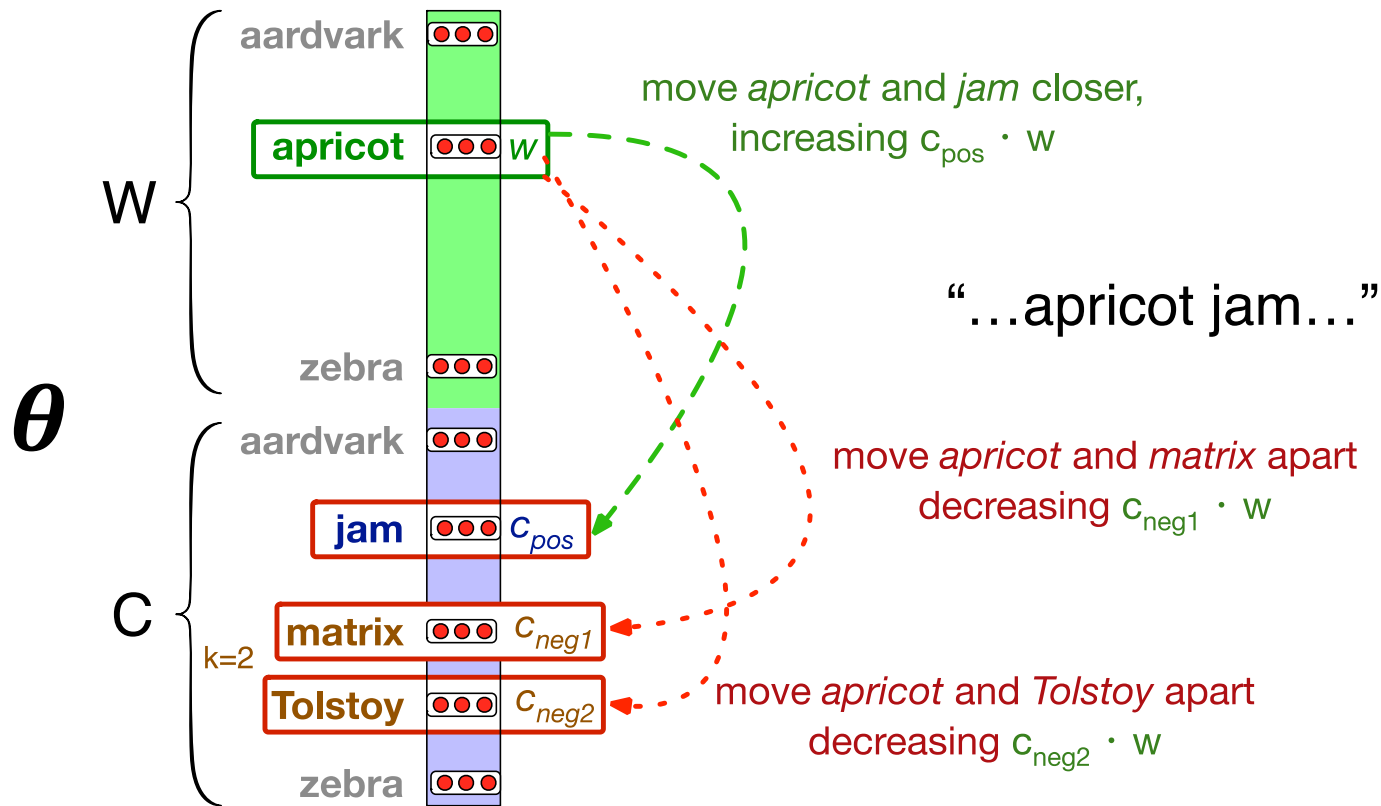
How to learn?

- Stochastic gradient descent!

We'll adjust the word weights to

- make the positive pairs more likely
- and the negative pairs less likely,
- over the entire training set.

Intuition of one step of gradient descent



The derivatives of the loss function

$$L_{CE} = - \left[\log \sigma(c_{pos} \cdot w) + \sum_{i=1}^k \log \sigma(-c_{neg_i} \cdot w) \right]$$

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

$$\frac{\partial L_{CE}}{\partial c_{neg}} = [\sigma(c_{neg} \cdot w)]w$$

$$\frac{\partial L_{CE}}{\partial w} = [\sigma(c_{pos} \cdot w) - 1]c_{pos} + \sum_{i=1}^k [\sigma(c_{neg_i} \cdot w)]c_{neg_i}$$

Update equation in SGD

Start with randomly initialized C and W matrices, then incrementally do updates

$$c_{pos}^{t+1} = c_{pos}^t - \eta [\sigma(c_{pos}^t \cdot w^t) - 1] w^t$$

$$c_{neg}^{t+1} = c_{neg}^t - \eta [\sigma(c_{neg}^t \cdot w^t)] w^t$$

$$w^{t+1} = w^t - \eta \left[[\sigma(c_{pos} \cdot w^t) - 1] c_{pos} + \sum_{i=1}^k [\sigma(c_{neg_i} \cdot w^t)] c_{neg_i} \right]$$

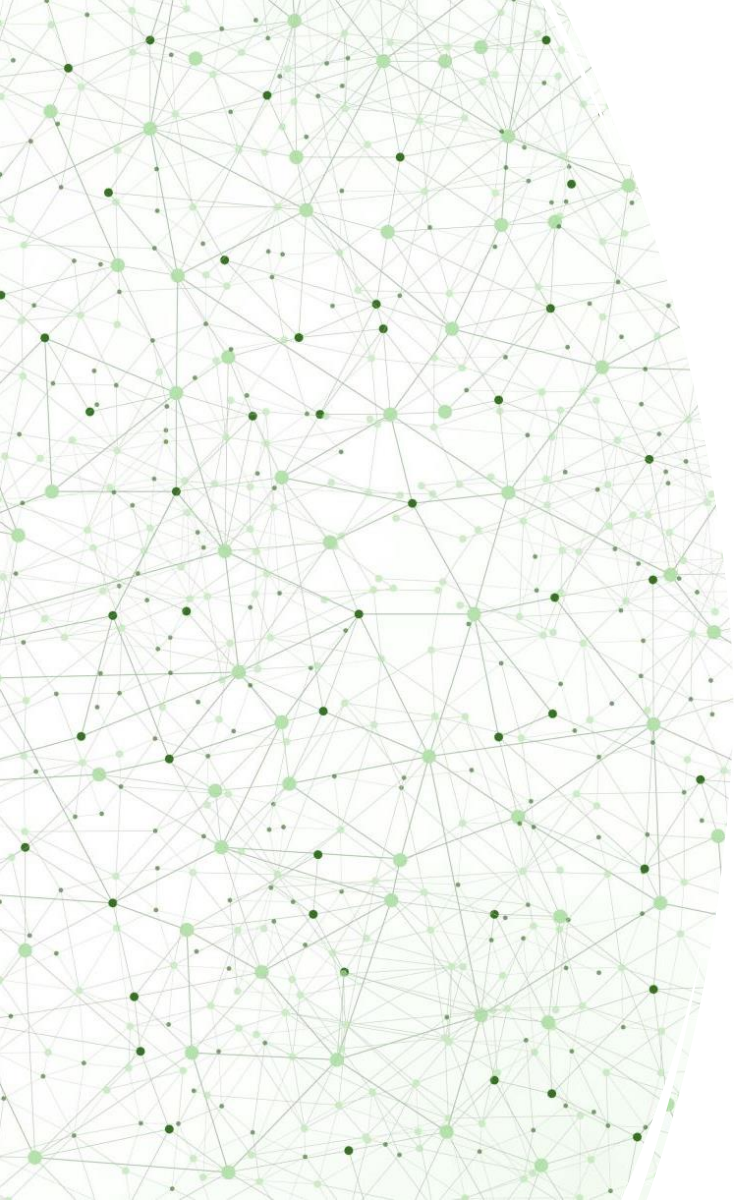
Two sets of embeddings

SGNS learns two sets of embeddings

Target embeddings matrix W

Context embedding matrix C

It's common to just add them together,
representing word i as the vector $w_i + c_i$



Properties of Embeddings

The kinds of neighbors depend on window size

Small windows ($C = \pm 2$) : nearest words are syntactically similar words in same taxonomy

- *Hogwarts* nearest neighbors are other fictional schools
- *Sunnydale, Evernight, Blandings*

Large windows ($C = \pm 5$) : nearest words are related words in same semantic field

- *Hogwarts* nearest neighbors are Harry Potter world:
- *Dumbledore, half-blood, Malfoy*

Analogical relations via parallelogram

The parallelogram method can solve analogies with both sparse and dense embeddings (Turney and Littman 2005, Mikolov et al. 2013b)

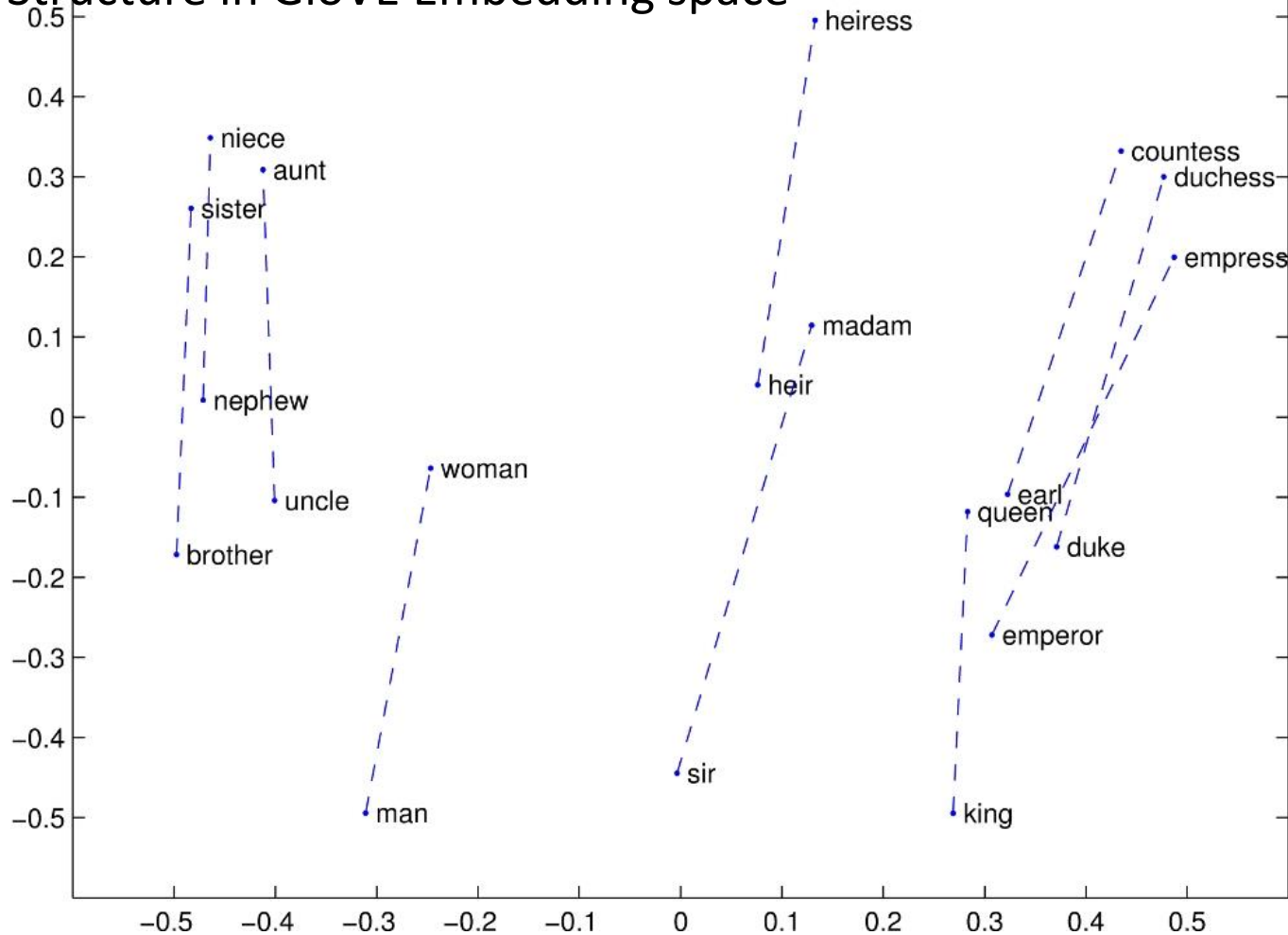
$\overrightarrow{\text{king}} - \overrightarrow{\text{man}} + \overrightarrow{\text{woman}}$ is close to $\overrightarrow{\text{queen}}$

$\overrightarrow{\text{Paris}} - \overrightarrow{\text{France}} + \overrightarrow{\text{Italy}}$ is close to $\overrightarrow{\text{Rome}}$

For a problem $a:a^*:b:b^*$, the parallelogram method is:

$$\hat{b}^* = \operatorname{argmax}_x \text{distance}(x, a^* - a + b)$$

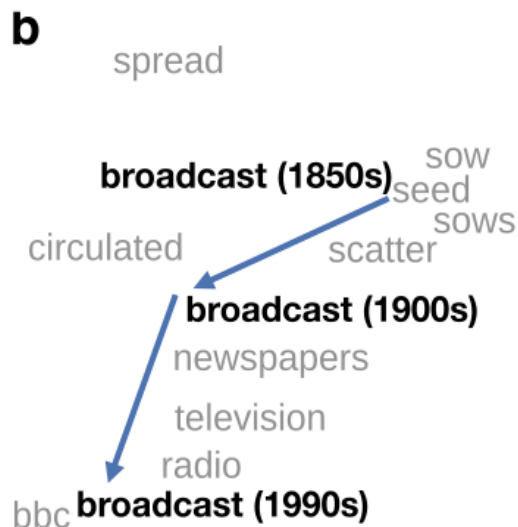
Structure in GloVe Embedding space



Embeddings as a window onto historical semantics

Train embeddings on different decades of historical text to see meanings shift

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



William L. Hamilton, Jure Leskovec, and Dan Jurafsky. 2016. Diachronic Word Embeddings Reveal Statistical Laws of Semantic Change. Proceedings of ACL.

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 *NeurIPS*, 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

Algorithms that use embeddings as part of e.g., hiring searches for programmers, might lead to bias in hiring

Historical embedding as a tool to study cultural biases

Garg, N., Schiebinger, L., Jurafsky, D., and Zou, J. (2018). Word embeddings quantify 100 years of gender and ethnic stereotypes. Proceedings of the National Academy of Sciences 115(16), E3635–E3644.

- Compute a **gender or ethnic bias** for each adjective: e.g., how much closer the adjective is to "woman" synonyms than "man" synonyms, or names of particular ethnicities
- Embeddings for **competence** adjective (*smart, wise, brilliant, resourceful, thoughtful, logical*) are biased toward men, a bias slowly decreasing 1960-1990
- Embeddings for **dehumanizing** adjectives (barbaric, monstrous, bizarre) were biased toward Asians in the 1930s, bias decreasing over the 20th century.
- These match the results of old surveys done in the 1930s

Gracias por la atención

¿Tiene alguna pregunta?

