COS 484: Natural Language Processing

L3: Word Embeddings

Spring 2026

(Some slides are adapted from Dan Jurafsky)

Announcements

- A1 will be released today
  - Deadline in two weeks
- Precepts : Fri 12-1pm, FC 004

How do we represent words in NLP models?

- n-gram models

$$P(w_1, w_2, ...w_n) = \prod_{i=1}^{n} P(w_i|w_{i-1})$$

$$P(w_i|w_{i-1}) = \frac{C(w_{i-1}, w_i) + \alpha}{C(w_{i-1}) + \alpha|V|}$$

Each word is just a string

or indices w_i in the vocabulary list

cat = the 5th word in V

dog = the 10th word in V

cats = the 118th word in V

- Naive Bayes

$$\hat{P}(w_i \mid c_j) = \frac{\text{Count}(w_i, c_j) + \alpha}{\sum_{w \in V} \text{Count}(w, c_j) + \alpha|V|}$$

What are some issues with representing words in these ways?

Need for word meaning in NLP models

- With words, a feature is a word identity (= string)

    - Feature 5: `The previous word was “terrible”’
    - Requires **exact same word** to be in the training and testing set

“terrible” ≠ “horrible”

- If we can represent word meaning in vectors:

    - The previous word was vector [35, 22, 17, …]
    - Now in the test set we might see a similar vector [34, 21, 14, …]
    - We can generalize to **similar but unseen** words!!!

Guess the meaning of “Ongchoi”

- Ongchoi is delicious sautéed with garlic
- Ongchoi is superb over rice
- Ongchoi leaves with salty sauces

Q: What do you think ‘Ongchoi’ means?

(A) a savory snack

(B) a green vegetable.

(C) an alcoholic beverage

(D) a cooking sauce

Guess the meaning of Ongchoi

“Ongchoi”

Ongchoi is a leafty green like spinach, chard or collard greens

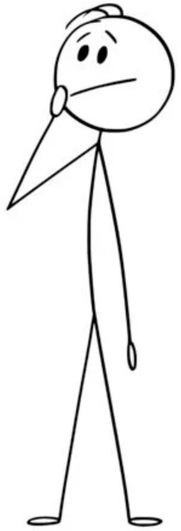
空心菜
kangkong
rau muống
…

How can we do the same thing computationally?

- Count the words in the context of ongchoi
- See what other words occur in those contexts

We can represent a word’s context using vectors!

Distributional hypothesis



- “The meaning of a word is its use in the language”
- “If A and B have almost identical environments we say that they are synonyms.”
- “You shall know a word by the company it keeps”

[Wittgenstein PI 43]

[Harris 1954]

[Firth 1957]

Words and vectors

First solution: Let's use **word-word co-occurrence counts** to represent the meaning of words!

Each word is represented by the corresponding **row vector**

context words:

4 words to the left +

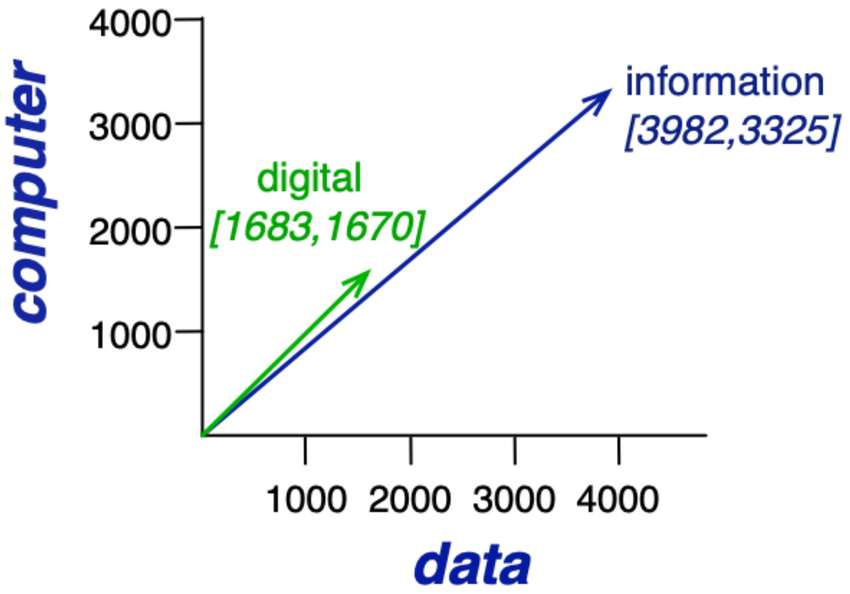
4 words to the right

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

	aardvark	...	computer	data	result	pie	sugar	...
cherry	0	...	2	8	9	442	25	...
strawberry	0	...	0	0	1	60	19	...
digital	0	...	1670	1683	85	5	4	...
information	0	...	3325	3982	378	5	13	...

Most entries are 0s $\Longrightarrow$ sparse vectors

Measuring similarity



A common similarity metric: **cosine** of the angle between the two vectors (the larger, the more similar the two vectors are)

$$\cos(\mathbf{u}, \mathbf{v}) = \frac{\mathbf{u} \cdot \mathbf{v}}{\|\mathbf{u}\|\|\mathbf{v}\|}$$

$$\cos(\mathbf{u}, \mathbf{v}) = \frac{\sum_{i=1}^{|V|} u_i v_i}{\sqrt{\sum_{i=1}^{|V|} u_i^2}\sqrt{\sum_{i=1}^{|V|} v_i^2}}$$

Q: Why cosine similarity instead of dot product $\mathbf{u} \cdot \mathbf{v}$?

Measuring similarity

What is the range of $\cos(u, v)$ if u, v are **count vectors**?

(A) [-1, 1]
(B) [0, 1]
(C) (0, 1)
(D) (-1, 1)
(E) $(-\infty, +\infty)$

$$\cos(\mathbf{u}, \mathbf{v}) = \frac{\sum_{i=1}^{|V|} u_i v_i}{\sqrt{\sum_{i=1}^{|V|} u_i^2} \sqrt{\sum_{i=1}^{|V|} v_i^2}}$$

The answer is (b). Cosine similarity ranges between -1 and 1 in general. In this model, all the values of u_i, v_i are non-negative.

Any issues with this model?

- Raw frequency count is a bad representation!
  - Frequency is clearly useful; if “pie” appears a lot near “cherry”, that's useful information.
  - But overly frequent words like “the”, “it", or “they” also appear a lot near “cherry”. They are not very informative about the context.

- Solution: use a **weighted function** instead of raw counts!

Using a weighted function: PMI

- Solution: use a **weighted function** instead of raw counts!
- Pointwise Mutual Information (PMI):
  - Do events *x* and *y* co-occur more or less than if they were independent?

$$\text{PMI}(x, y) = \log_2 \frac{P(x, y)}{P(x)P(y)}$$

  - As an example,
    - Does “pie” occur more often than we’d expect near “cherry”?

$$\text{PMI}(w = \text{cherry}, c = \text{pie}) = \log_2 \frac{P(w = \text{cherry}, c = \text{pie})}{P(w = \text{cherry})\ P(c = \text{pie})}$$

(compute P using MLE)

Sparse vs dense vectors

- The vectors in the word-word occurrence matrix are
  - Long: vocabulary size
  - Sparse: most are 0's
- Alternative: we want to represent words as short (50-300 dimensional) & dense (real-valued) vectors
  - This is the basis for modern NLP systems!

$$v_{\text{cat}} = \begin{pmatrix} -0.224 \\ 0.130 \\ -0.290 \\ 0.276 \end{pmatrix} \quad v_{\text{dog}} = \begin{pmatrix} -0.124 \\ 0.430 \\ -0.200 \\ 0.329 \end{pmatrix} \quad v_{\text{the}} = \begin{pmatrix} 0.234 \\ 0.266 \\ 0.239 \\ -0.199 \end{pmatrix} \quad v_{\text{language}} = \begin{pmatrix} 0.290 \\ -0.441 \\ 0.762 \\ 0.982 \end{pmatrix}$$

Why dense vectors?

- Short vectors are easier to use as features in ML systems
- Dense vectors may generalize better than explicit counts
- Sparse count vectors can't capture high-order co-occurrence
  - w_1 co-occurs with "car", w_2 co-occurs with "automobile"
  - They should be similar but they aren't because "car" and "automobile" are distinct dimensions
- In practice, dense vectors work better!

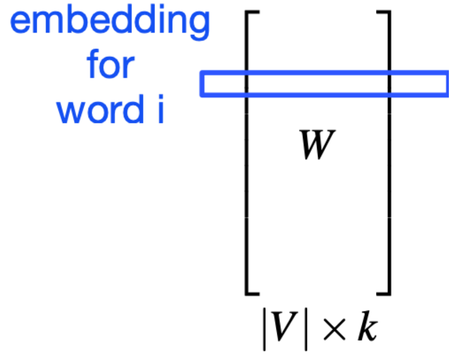
How to get short dense vectors?

- **Count-based methods**: Singular value decomposition (SVD) of count matrix

Singular value decomposition (SVD) of PPMI weighted co-occurrence matrix

$$\underset{|V|\times|V|}{\begin{bmatrix} \\ X \\ \\ \end{bmatrix}} = \underset{|V|\times|V|}{\begin{bmatrix} \\ W \\ \\ \end{bmatrix}} \underset{|V|\times|V|}{\begin{bmatrix} \sigma_1 & 0 & 0 & \dots & 0 \\ 0 & \sigma_2 & 0 & \dots & 0 \\ 0 & 0 & \sigma_3 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & \sigma_V \end{bmatrix}} \underset{|V|\times|V|}{\begin{bmatrix} \\ C \\ \\ \end{bmatrix}}$$

$$\underset{|V|\times|V|}{\begin{bmatrix} \\ X \\ \\ \end{bmatrix}} = \underset{|V|\times k}{\begin{bmatrix} \\ W \\ \\ \end{bmatrix}} \underset{k\times k}{\begin{bmatrix} \sigma_1 & 0 & 0 & \dots & 0 \\ 0 & \sigma_2 & 0 & \dots & 0 \\ 0 & 0 & \sigma_3 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & \sigma_k \end{bmatrix}} \underset{k\times|V|}{\begin{bmatrix} C \end{bmatrix}}$$



We can approximate the full matrix by only keeping the top k (e.g., 100) singular values!

How to get short dense vectors?

- **Prediction-based methods**:
  - Vectors are created by training a classifier to predict whether a word *c* (“pie”) is likely to appear in the context of a word w (“cherry”)
  - Examples: **word2vec** (Mikolov et al., 2013), **Glove** (Pennington et al., 2014), **FastText** (Bojanowski et al., 2017)

Also called word **embeddings!**

Don’t count, predict! **A systematic comparison of context-counting vs. context-predicting semantic vectors**

Marco Baroni and **Georgiana Dinu** and **Germán Kruszewski**
Center for Mind/Brain Sciences (University of Trento, Italy)

(Baroni et al., 2014)

Word2vec and other variants

Word embeddings

- Basic property: similar words have similar vectors

word w^*= “sweden”

$$\arg\max_{w \in V} \cos(e(w), e(w^*))$$

Word	Cosine distance
norway	0.760124
denmark	0.715460
finland	0.620022
switzerland	0.588132
belgium	0.585835
netherlands	0.574631
iceland	0.562368
estonia	0.547621
slovenia	0.531408

$\cos(u, v)$ ranges between -1 and 1

Word embeddings: the learning problem

Learning vectors from text for representing words

- **Input**:
  - a large text corpus,
  - vocabulary V
  - vector dimension d (e.g., 300)
- **Output**: $f\colon V \to \mathbb{R}^d$

$$v_{\text{cat}} = \begin{pmatrix} -0.224 \\ 0.130 \\ -0.290 \\ 0.276 \end{pmatrix} \quad v_{\text{dog}} = \begin{pmatrix} -0.124 \\ 0.430 \\ -0.200 \\ 0.329 \end{pmatrix}$$

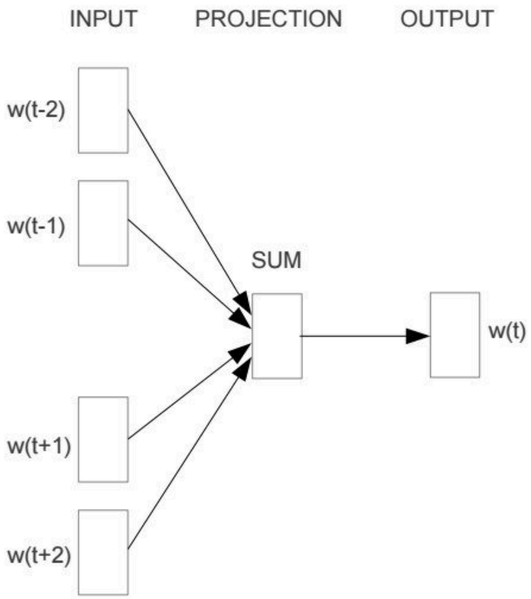
$$v_{\text{the}} = \begin{pmatrix} 0.234 \\ 0.266 \\ 0.239 \\ -0.199 \end{pmatrix} \quad v_{\text{language}} = \begin{pmatrix} 0.290 \\ -0.441 \\ 0.762 \\ 0.982 \end{pmatrix}$$

Note: Each coordinate/dimension of the vector doesn't have a particular interpretation

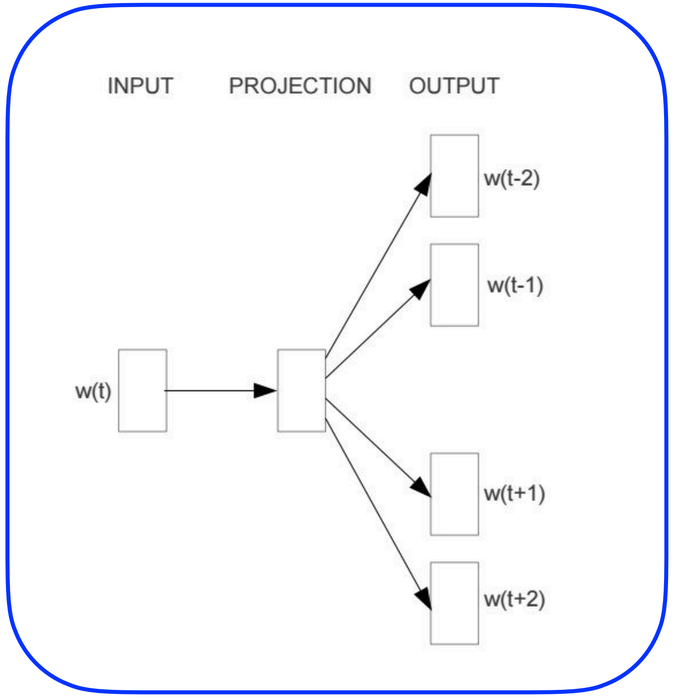
word2vec

- (Mikolov et al 2013a): Efficient Estimation of Word Representations in Vector Space
- (Mikolov et al 2013b): Distributed Representations of Words and Phrases and their Compositionality

Tomáš Mikolov



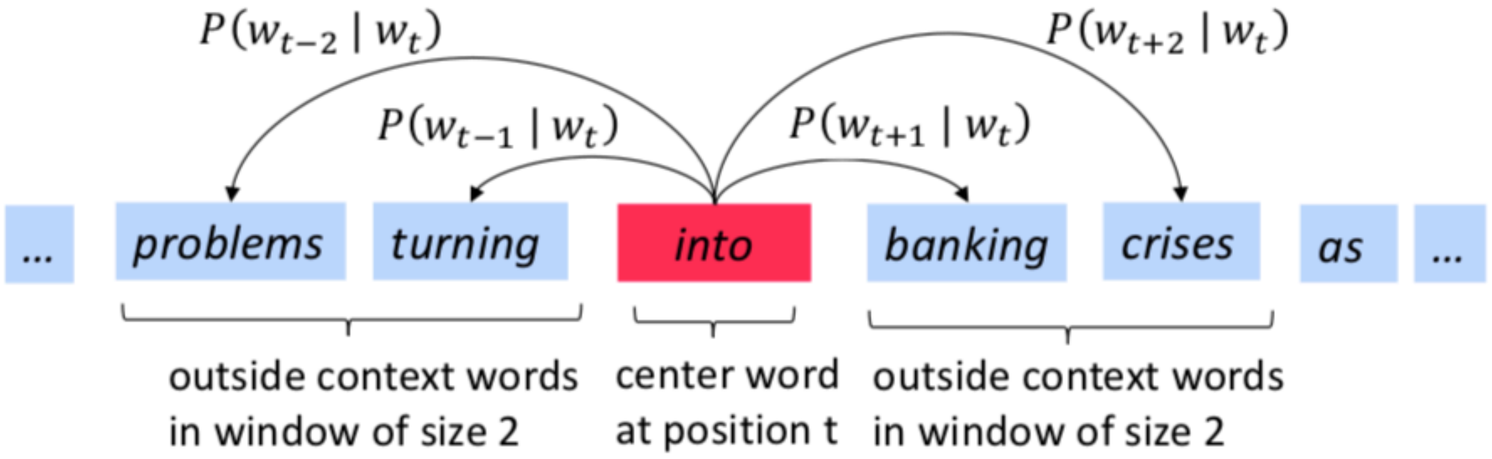
Continuous Bag of Words (CBOW)



Skip-gram

Skip-gram

- Assume that we have a large corpus $w_1, w_2, \ldots, w_T \in V$
- Key idea: Use each word to predict other words in its context
- Context: a fixed window of size 2m (m = 2 in the example)



A classification problem!

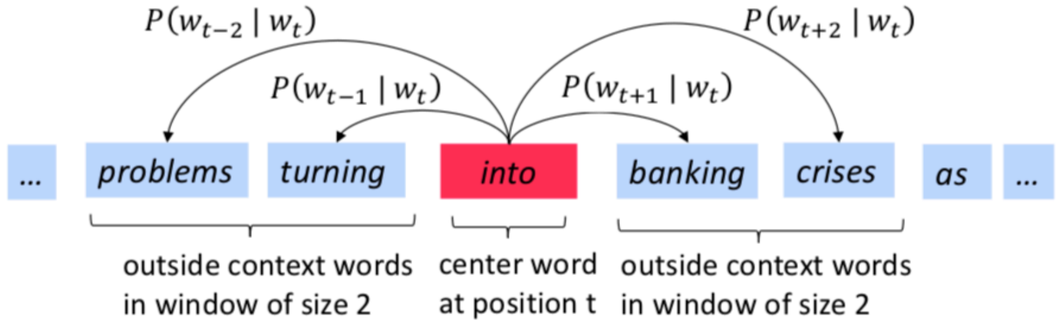
$P(b \mid a)$ = given the center word is a, what is the probability that b is a context word?

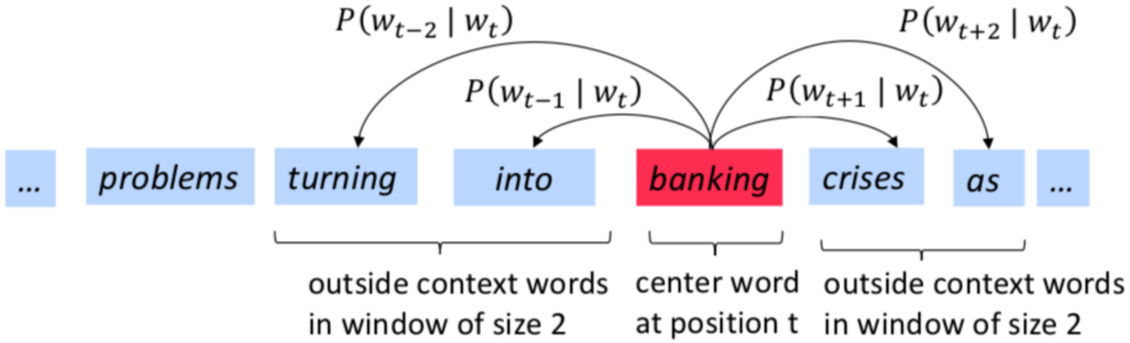
$P(\cdot \mid a)$ is a probability distribution defined over V:

$$\sum_{w \in V} P(w \mid a) = 1$$

We are going to define this distribution soon!

Skip-gram





Convert into training data:
(into, problems)
(into, turning)
(into, banking)
(into, crises)
(banking, turning)
(banking, into)
(banking, crises)
(banking, as)
…

Our goal is to find parameters that can maximize

$P(\text{problems} \mid \text{into}) \times P(\text{turning} \mid \text{into}) \times P(\text{banking} \mid \text{into}) \times P(\text{crises} \mid \text{into}) \times$

$P(\text{turning} \mid \text{banking}) \times P(\text{into} \mid \text{banking}) \times P(\text{crises} \mid \text{banking}) \times P(\text{as} \mid \text{banking}) \ldots$

Skip-gram: objective function

- For each position $t = 1,2,\ldots T$, predict context words within context size m, given center word w_t:

all the parameters to be optimized

$$\mathcal{L}(\theta) = \prod_{t=1}^{T} \prod_{-m \leq j \leq m, j \neq 0} P(w_{t+j} \mid w_t; \theta)$$

- It is equivalent to minimizing the (average) negative log likelihood:

$$J(\theta) = -\frac{1}{T} \log \mathcal{L}(\theta) = -\frac{1}{T} \sum_{t=1}^{T} \sum_{-m \leq j \leq m, j \neq 0} \log P(w_{t+j} \mid w_t; \theta)$$

How to define $P(w_{t+j} \mid w_t; \theta)$?

- Use two sets of vectors for each word in the vocabulary

$$\mathbf{u}_a \in \mathbb{R}^d \quad \text{: vector for center word } a, \forall a \in V$$

$$\mathbf{v}_b \in \mathbb{R}^d \quad \text{: vector for context word } b, \forall b \in V$$

- Use inner product $\mathbf{u}_a \cdot \mathbf{v}_b$ to measure how likely word a appears with context word b

$$P(w_{t+j} \mid w_t) = \frac{\exp\left(\mathbf{u}_{w_t} \cdot \mathbf{v}_{w_{t+j}}\right)}{\sum_{k \in V} \exp\left(\mathbf{u}_{w_t} \cdot \mathbf{v}_k\right)}$$

Does this term seem familiar?

… vs multinominal logistic regression

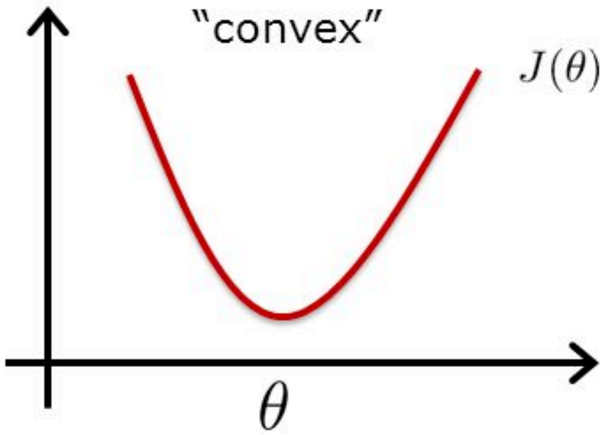
- Essentially a |V|-way classification problem
- Recall: multinomial logistic regression:

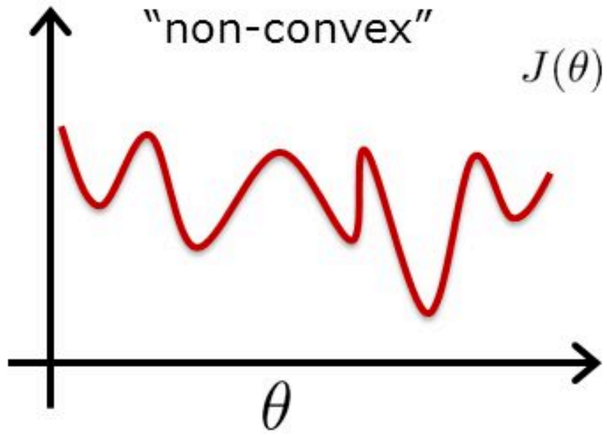
$$P(y = c \,|\, x) = \frac{\exp(\mathbf{w}_c \cdot \mathbf{x} + b_c)}{\sum_{j=1}^{m} \exp(\mathbf{w}_j \cdot \mathbf{x} + b_j)}$$

$$P(w_{t+j} \mid w_t) = \frac{\exp(\mathbf{u}_{w_t} \cdot \mathbf{v}_{w_{t+j}})}{\sum_{k \in V} \exp(\mathbf{u}_{w_t} \cdot \mathbf{v}_k)}$$

- If we fix $\mathbf{u}_{w_t}$, it is reduced to a multinomial logistic regression problem.
- However, since we have to learn both $\mathbf{u}$ and $\mathbf{v}$ together, the training objective is non-convex.

… vs multinominal logistic regression





- It is hard to find a global minimum
- But can still use stochastic gradient descent to optimize θ:

$$\theta^{(t+1)} = \theta^{(t)} - \eta \nabla_\theta J(\theta)$$

Important note

$$J(\theta) = -\frac{1}{T}\sum_{t=1}^{T}\sum_{-m\leq j\leq m, j\neq 0} \log \frac{\exp(\mathbf{u}_{w_t} \cdot \mathbf{v}_{w_{t+j}})}{\sum_{k\in V}\exp(\mathbf{u}_{w_t} \cdot \mathbf{v}_k)}$$

- In this formulation, we don't care about the classification task itself like we do for the logistic regression model we saw previously.
- Instead, the parameters (vectors) that optimize the training objective end up being very good word representations!

How many parameters in this model?

How many parameters does this model have (i.e. what is size of θ)?

$$J(\theta) = -\frac{1}{T}\sum_{t=1}^{T}\sum_{-m\leq j\leq m, j\neq 0}\log\frac{\exp(\mathbf{u}_{w_t}\cdot\mathbf{v}_{w_{t+j}})}{\sum_{k\in V}\exp(\mathbf{u}_{w_t}\cdot\mathbf{v}_k)}$$

(a) $d|V|$
(b) $2d|V|$
(c) $2m|V|$
(d) $2md|V|$

$V :=$ Vocabulary
$d :=$ dimension of embedding
$m :=$ size of context window

The answer is (b).
Each word has two d-dimensional vectors, so it is $2 \times |V| \times d$.

word2vec formulation

$$J(\theta) = -\frac{1}{T}\sum_{t=1}^{T}\sum_{-m\leq j\leq m, j\neq 0} \log \frac{\exp(\mathbf{u}_{w_t}\cdot\mathbf{v}_{w_{t+j}})}{\sum_{k\in V}\exp(\mathbf{u}_{w_t}\cdot\mathbf{v}_k)}$$

Q: Why do we need two vectors for each word?

- Because one word is not likely to appear in its own context window, e.g., $P(\text{dog} \mid \text{dog})$ should be low. If we use one set of vectors only, it essentially needs to minimize $\mathbf{u}_{\text{dog}} \cdot \mathbf{u}_{\text{dog}}$

Q: Which set of vectors are used as word embeddings?

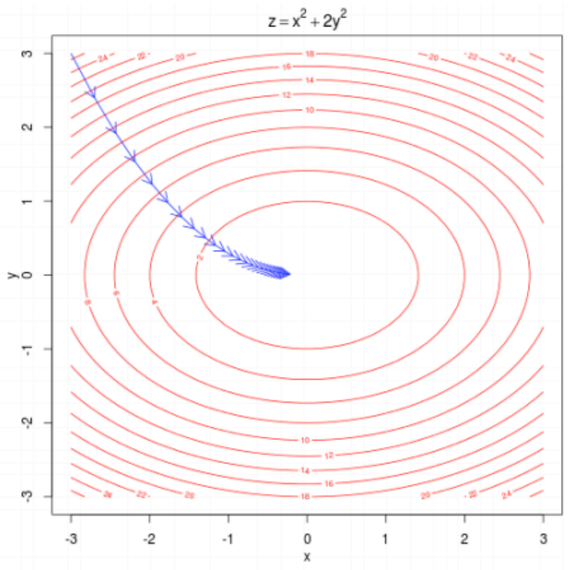
- This is an empirical question. Typically just $\mathbf{u}_w$ but you can also concatenate the two vectors..

How to train this model?

$$J(\theta) = -\frac{1}{T}\sum_{t=1}^{T}\sum_{-m\leq j\leq m, j\neq 0} \log\frac{\exp(\mathbf{u}_{w_t}\cdot\mathbf{v}_{w_{t+j}})}{\sum_{k\in V}\exp(\mathbf{u}_{w_t}\cdot\mathbf{v}_k)}$$

- To train such a model, we need to compute the vector gradient $\nabla_\theta J(\theta) = ?$
- Remember that θ represents all $2d|V|$ model parameters, in one vector.

$$\theta = \begin{bmatrix} v_{aardvark} \\ v_a \\ \vdots \\ v_{zebra} \\ u_{aardvark} \\ u_a \\ \vdots \\ u_{zebra} \end{bmatrix}$$



Vectorized gradients

$$f(\mathbf{x}) = \mathbf{x} \cdot \mathbf{a}$$

$$\mathbf{x}, \mathbf{a} \in \mathbb{R}^n$$

$$\frac{\partial f}{\partial \mathbf{x}} = \mathbf{a}$$

$$f = x_1 a_1 + x_2 a_2 + \ldots + x_n a_n$$

$$\frac{\partial f}{\partial \mathbf{x}} = [\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \ldots, \frac{\partial f}{\partial x_n}]$$

Vectorized gradients: exercises

Let $f = \exp(\mathbf{w} \cdot \mathbf{x})$, what is the value of $\frac{\partial f}{\partial \mathbf{x}}$? (Assume $\mathbf{w}, \mathbf{x} \in \mathbb{R}^n$)

(a) $\mathbf{w}$

(b) $\exp(\mathbf{w} \cdot \mathbf{x})$

(c) $\exp(\mathbf{w} \cdot \mathbf{x})\mathbf{w}$

(d) $\mathbf{x}$

The answer is (c).

$$\frac{\partial}{\partial x_i} = \frac{\exp(\sum_{k=1}^{n} w_i x_i)}{\partial x_i} = \exp(\sum_{k=1}^{n} w_i x_i) w_i$$

Let's compute gradients for word2vec

$$J(\theta) = -\frac{1}{T}\sum_{t=1}^{T}\sum_{-m\leq j\leq m, j\neq 0} \log \frac{\exp(\mathbf{u}_{w_t} \cdot \mathbf{v}_{w_{t+j}})}{\sum_{k\in V}\exp(\mathbf{u}_{w_t} \cdot \mathbf{v}_k)}$$

Consider one pair of center/context words (t, c):

$$y = -\log\left(\frac{\exp(\mathbf{u}_t \cdot \mathbf{v}_c)}{\sum_{k\in V}\exp(\mathbf{u}_t \cdot \mathbf{v}_k)}\right)$$

Here, t = target, c = context

We need to compute the gradient of y with respect to

$$\mathbf{u}_t \text{ and } \mathbf{v}_k, \forall k \in V$$

Let's compute gradients for word2vec

$$y = -\log\left(\frac{\exp(\mathbf{u}_t \cdot \mathbf{v}_c)}{\sum_{k \in V} \exp(\mathbf{u}_t \cdot \mathbf{v}_k)}\right)$$

$$y = -\log(\exp(\mathbf{u}_t \cdot \mathbf{v}_c)) + \log(\sum_{k \in V} \exp(\mathbf{u}_t \cdot \mathbf{v}_k))$$
$$= -\mathbf{u}_t \cdot \mathbf{v}_c + \log(\sum_{k \in V} \exp(\mathbf{u}_t \cdot \mathbf{v}_k))$$

Recall that

$$P(w_{t+j} \mid w_t) = \frac{\exp(\mathbf{u}_{w_t} \cdot \mathbf{v}_{w_{t+j}})}{\sum_{k \in V} \exp(\mathbf{u}_{w_t} \cdot \mathbf{v}_k)}$$

$$\frac{\partial y}{\partial \mathbf{u}_t} = \frac{\partial(-\mathbf{u}_t \cdot \mathbf{v}_c)}{\partial \mathbf{u}_t} + \frac{\partial(\log \sum_{k \in V} \exp(\mathbf{u}_t \cdot \mathbf{v}_k))}{\partial \mathbf{u}_t}$$
$$= -\mathbf{v}_c + \frac{\frac{\partial \sum_{k \in V} \exp(\mathbf{u}_t \cdot \mathbf{v}_k)}{\partial \mathbf{u}_t}}{\sum_{k \in V} \exp(\mathbf{u}_t \cdot \mathbf{v}_k)}$$
$$= -\mathbf{v}_c + \frac{\sum_{k \in V} \exp(\mathbf{u}_t \cdot \mathbf{v}_k) \cdot \mathbf{v}_k}{\sum_{k \in V} \exp(\mathbf{u}_t \cdot \mathbf{v}_k)}$$
$$= -\mathbf{v}_c + \sum_{k \in V} \frac{\exp(\mathbf{u}_t \cdot \mathbf{v}_k)}{\sum_{k' \in V} \exp(\mathbf{u}_t \cdot \mathbf{v}_{k'})} \mathbf{v}_k$$
$$= -\mathbf{v}_c + \sum_{k \in V} P(k \mid t) \mathbf{v}_k$$

Gradients for word2vec

What about context vectors?

$$\frac{\partial y}{\partial \mathbf{v}_k} = \begin{cases} (P(k \mid t) - 1)\,\mathbf{u}_t & k = c \\ P(k \mid t)\mathbf{u}_t & k \neq c \end{cases}$$

$$y = -\log\left(\frac{\exp(\mathbf{u}_t \cdot \mathbf{v}_c)}{\sum_{k \in V} \exp(\mathbf{u}_t \cdot \mathbf{v}_k)}\right)$$

See assignment 2 :)

Overall algorithm

- Input: text corpus, embedding size d, vocabulary V, context size m
- Initialize $\mathbf{u}_i, \mathbf{v}_i$ randomly $\forall i \in V$
- Run through the training corpus and for each training instance (t, c):
  - Update $\mathbf{u}_t \leftarrow \mathbf{u}_t - \eta \frac{\partial y}{\partial \mathbf{u}_t}$; $\quad \frac{\partial y}{\partial \mathbf{u}_t} = -\mathbf{v}_c + \sum_{k \in V} P(k \mid t) \mathbf{v}_k$
  - Update $\mathbf{v}_k \leftarrow \mathbf{v}_k - \eta \frac{\partial y}{\partial \mathbf{v}_k}, \forall k \in V$; $\quad \frac{\partial y}{\partial \mathbf{v}_k} = \begin{cases} (P(k \mid t) - 1)\mathbf{u}_t & k = c \\ P(k \mid t)\mathbf{u}_t & k \neq c \end{cases}$

Q: Can you think of any issues with this algorithm?

Skip-gram with negative sampling (SGNS)

Problem: every time you get one pair of (t, c), you need to update $\mathbf{v}_k$ with all the words in the vocabulary! This is very expensive computationally.

$$\frac{\partial y}{\partial \mathbf{u}_t} = -\mathbf{v}_c + \sum_{k \in V} P(k \mid t)\mathbf{v}_k \quad ; \quad \frac{\partial y}{\partial \mathbf{v}_k} = \begin{cases} (P(k \mid t) - 1)\ \mathbf{u}_t & k = c \\ P(k \mid t)\ \mathbf{u}_t & k \neq c \end{cases}$$

Negative sampling: instead of considering all the words in V, let's randomly sample K (5-20) negative examples.

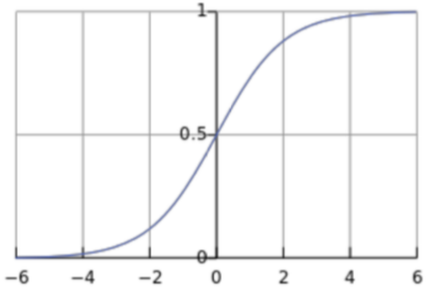
Softmax:

$$y = -\log\left(\frac{\exp\left(\mathbf{u}_t \cdot \mathbf{v}_c\right)}{\sum_{k \in V} \exp\left(\mathbf{u}_t \cdot \mathbf{v}_k\right)}\right)$$

Negative sampling:

$$y = -\log\left(\sigma(\mathbf{u}_t \cdot \mathbf{v}_c)\right) - \sum_{i=1}^{K} \mathbb{E}_{j \sim P(w)} \log\left(\sigma(-\mathbf{u}_t \cdot \mathbf{v}_j)\right)$$

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



Skip-gram with negative sampling (SGNS)

Key idea: Convert the $|V|$-way classification into a set of binary classification tasks.

Every time we get a pair of words (*t*, *c*), we don't predict *c* among all the words in the vocabulary. Instead, we predict (*t*, *c*) is a positive pair, and (*t*, *c'*) is a negative pair for a small number of sampled *c'*.

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

$$y = -\log(\sigma(\mathbf{u}_t \cdot \mathbf{v}_c)) - \sum_{i=1}^{K} \mathbb{E}_{j \sim P(w)} \log(\sigma(-\mathbf{u}_t \cdot \mathbf{v}_j))$$

P(w): sampling according to the frequency of words

Similar to **binary logistic regression**, but we need to optimize $\mathbf{u}$ and $\mathbf{v}$ together.

$$P(y = 1 \mid t, c) = \sigma(\mathbf{u}_t \cdot \mathbf{v}_c) \qquad p(y = 0 \mid t, c') = 1 - \sigma(\mathbf{u}_t \cdot \mathbf{v}_{c'}) = \sigma(-\mathbf{u}_t \cdot \mathbf{v}_{c'})$$

Understanding SGNS

In skip-gram with negative sampling (SGNS), how many parameters need to be updated in θ for every (t, c) pair?

$$y = -\log(\sigma(\mathbf{u}_t \cdot \mathbf{v}_c)) - \sum_{i=1}^{K} \mathbb{E}_{j \sim P(w)} \log(\sigma(-\mathbf{u}_t \cdot \mathbf{v}_j))$$

(a) Kd

(b) $2Kd$

(c) $(K+1)d$

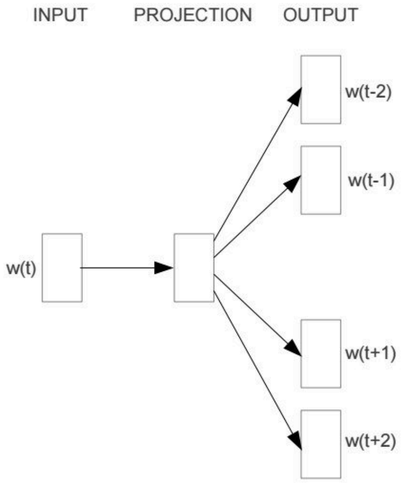
(d) $(K+2)d$

The answer is (d).

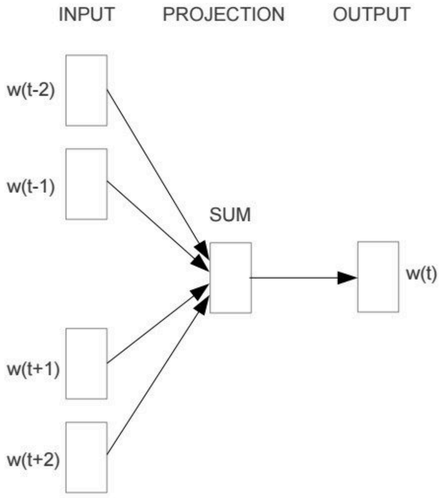
We need to calculate gradients with respect to $\mathbf{u}_t$ and (K + 1) $\mathbf{v}_i$

(one positive and K negatives).

Continuous Bag of Words (CBOW)



Skip-gram



Continuous Bag of Words (CBOW)

$$L(\theta) = \prod_{t=1}^{T} P\left(w_t \mid \{w_{t+j}\}, -m \leq j \leq m, j \neq 0\right)$$

$$\bar{\mathbf{v}}_t = \frac{1}{2m} \sum_{-m \leq j \leq m, j \neq 0} \mathbf{v}_{t+j}$$

$$P(w_t \mid \{w_{t+j}\}) = \frac{\exp(\mathbf{u}_{w_t} \cdot \bar{\mathbf{v}}_t)}{\sum_{k \in V} \exp(\mathbf{u}_k \cdot \bar{\mathbf{v}}_t)}$$

Trained word embeddings available

- word2vec: https://code.google.com/archive/p/word2vec/
- GloVe: https://nlp.stanford.edu/projects/glove/
- FastText: https://fasttext.cc/

Download pre-trained word vectors

- Pre-trained word vectors. This data is made available under the Public Domain Dedication and License v1.0 whose full text can be found at: http://www.opendatacommons.org/licenses/pddl/1.0/.
  - Wikipedia 2014 + Gigaword 5 (6B tokens, 400K vocab, uncased, 50d, 100d, 200d, & 300d vectors, 822 MB download): glove.6B.zip
  - Common Crawl (42B tokens, 1.9M vocab, uncased, 300d vectors, 1.75 GB download): glove.42B.300d.zip
  - Common Crawl (840B tokens, 2.2M vocab, cased, 300d vectors, 2.03 GB download): glove.840B.300d.zip
  - Twitter (2B tweets, 27B tokens, 1.2M vocab, uncased, 25d, 50d, 100d, & 200d vectors, 1.42 GB download): glove.twitter.27B.zip
- Ruby script for preprocessing Twitter data

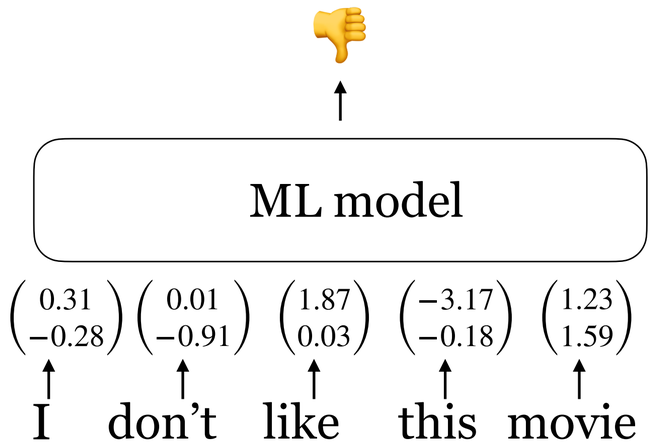
Differ in algorithms, text corpora, dimensions, cased/uncased…

Applied to many other languages

Evaluating word embeddings

Extrinsic vs intrinsic evaluation

- **Extrinsic evaluation**
  - Let's plug these word embeddings into a real NLP system and see whether this improves performance
  - Could take a long time but still the most important evaluation metric
- **Intrinsic evaluation**
  - Evaluate on a specific/intermediate subtask
  - Fast to compute
  - Not clear if it really helps downstream tasks



Intrinsic evaluation: word similarity

Word similarity

Example dataset: wordsim-353: 353 pairs of words with human judgement

http://www.cs.technion.ac.il/~gabr/resources/data/wordsim353/

Word 1	Word 2	Human (mean)
tiger	cat	7.35
tiger	tiger	10
book	paper	7.46
computer	internet	7.58
plane	car	5.77
professor	doctor	6.62
stock	phone	1.62
stock	CD	1.31
stock	jaguar	0.92

$$\cos(\boldsymbol{u}_i, \boldsymbol{u}_j) = \frac{\boldsymbol{u}_i \cdot \boldsymbol{u}_j}{||\boldsymbol{u}_i||_2 \times ||\boldsymbol{u}_j||_2}.$$

Metric: Spearman rank correlation

Intrinsic evaluation: word similarity

Model	Size	WS353	MC	RG	SCWS	RW
SVD	6B	35.3	35.1	42.5	38.3	25.6
SVD-S	6B	56.5	71.5	71.0	53.6	34.7
SVD-L	6B	65.7	72.7	75.1	56.5	37.0
CBOW†	6B	57.2	65.6	68.2	57.0	32.5
SG†	6B	62.8	65.2	69.7	58.1	37.2
GloVe	6B	65.8	72.7	77.8	53.9	38.1
SVD-L	42B	74.0	76.4	74.1	58.3	39.9
GloVe	42B	**75.9**	**83.6**	**82.9**	**59.6**	**47.8**
CBOW*	100B	68.4	79.6	75.4	59.4	45.5

SG: Skip-gram

Intrinsic evaluation: word analogy

Word analogy test: $a : a^* :: b : b^*$

$$b^* = \arg\max_{w \in V} \cos(e(w), e(a^*) - e(a) + e(b))$$

semantic

syntactic

Chicago:Illinois≈Philadelphia: ?

bad:worst ≈ cool: ?

More examples at

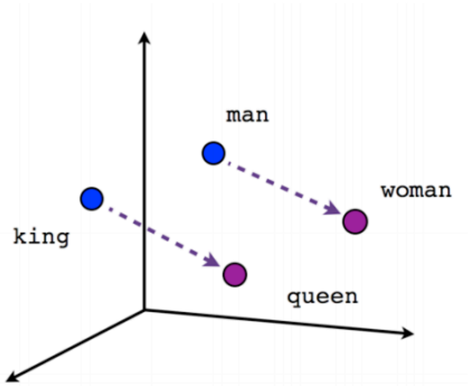
http://download.tensorflow.org/data/questions-words.txt

Metric: accuracy

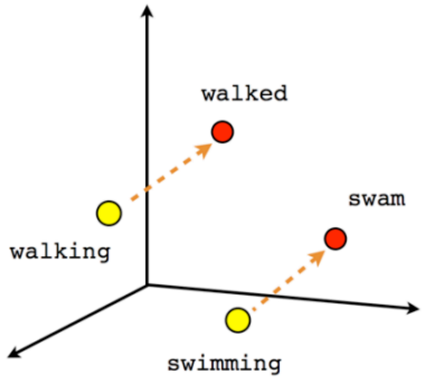
Model	Dim.	Size	Sem.	Syn.	Tot.
ivLBL	100	1.5B	55.9	50.1	53.2
HPCA	100	1.6B	4.2	16.4	10.8
GloVe	100	1.6B	67.5	54.3	60.3
SG	300	1B	61	61	61
CBOW	300	1.6B	16.1	52.6	36.1
vLBL	300	1.5B	54.2	64.8	60.0
ivLBL	300	1.5B	65.2	63.0	64.0
GloVe	300	1.6B	80.8	61.5	70.3
SVD	300	6B	6.3	8.1	7.3
SVD-S	300	6B	36.7	46.6	42.1
SVD-L	300	6B	56.6	63.0	60.1
CBOW†	300	6B	63.6	67.4	65.7
SG†	300	6B	73.0	66.0	69.1
GloVe	300	6B	77.4	67.0	71.7
CBOW	1000	6B	57.3	68.9	63.7
SG	1000	6B	66.1	65.1	65.6
SVD-L	300	42B	38.4	58.2	49.2
GloVe	300	42B	**81.9**	**69.3**	**75.0**

Word embeddings

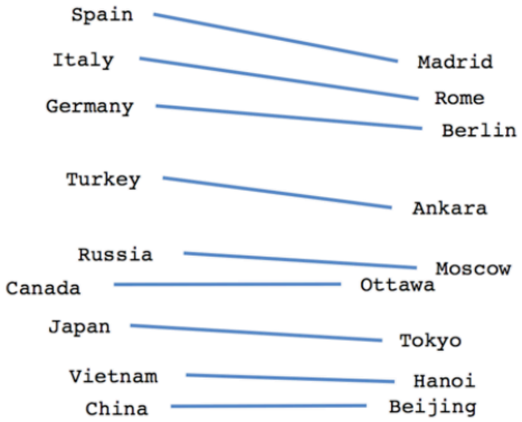
- They have some other nice properties too!



Male-Female



Verb tense



Country-Capital

$$v_{\text{man}} - v_{\text{woman}} \approx v_{\text{king}} - v_{\text{queen}}$$

$$v_{\text{Paris}} - v_{\text{France}} \approx v_{\text{Rome}} - v_{\text{Italy}}$$

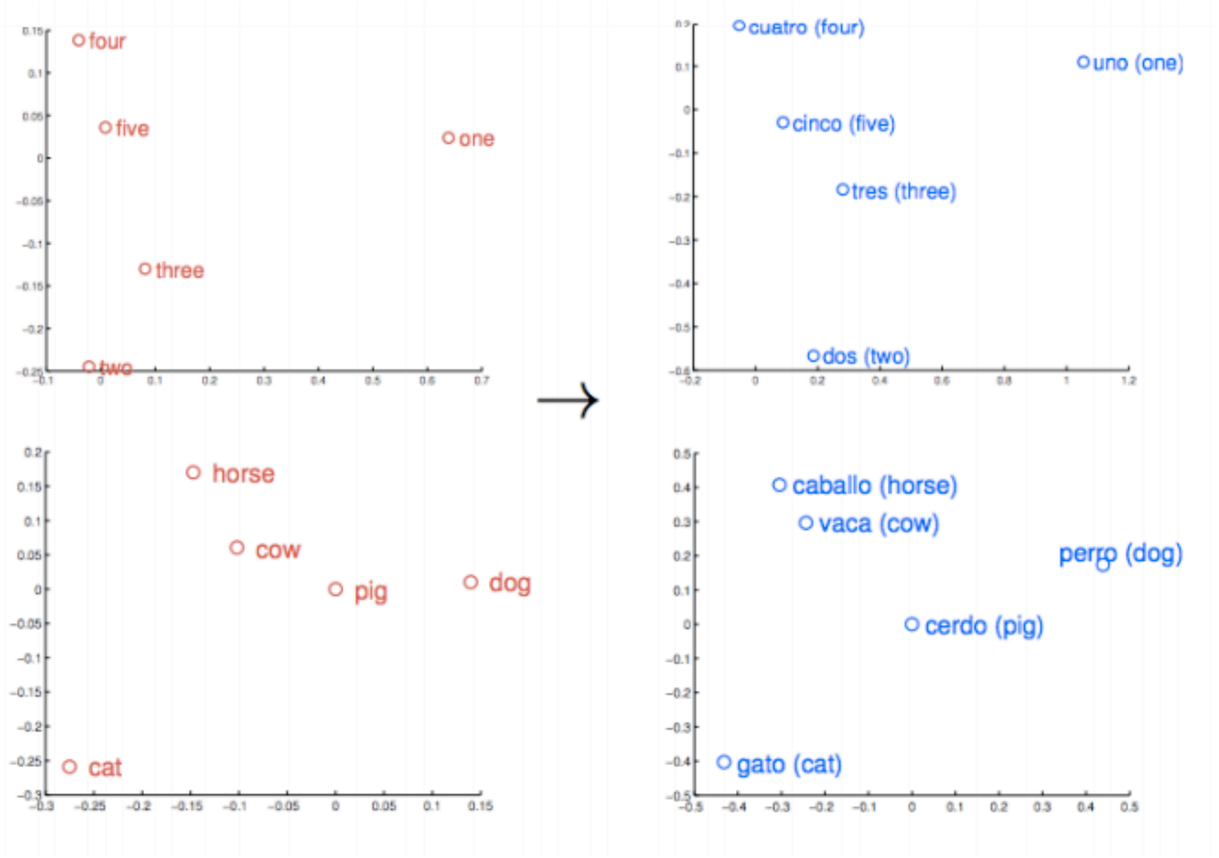
Word analogy test: $a : a^* :: b : b^*$

$$b^* = \arg\max_{w \in V} \cos(e(w), e(a^*) - e(a) + e(b))$$

Word embeddings

- They have some other nice properties too!

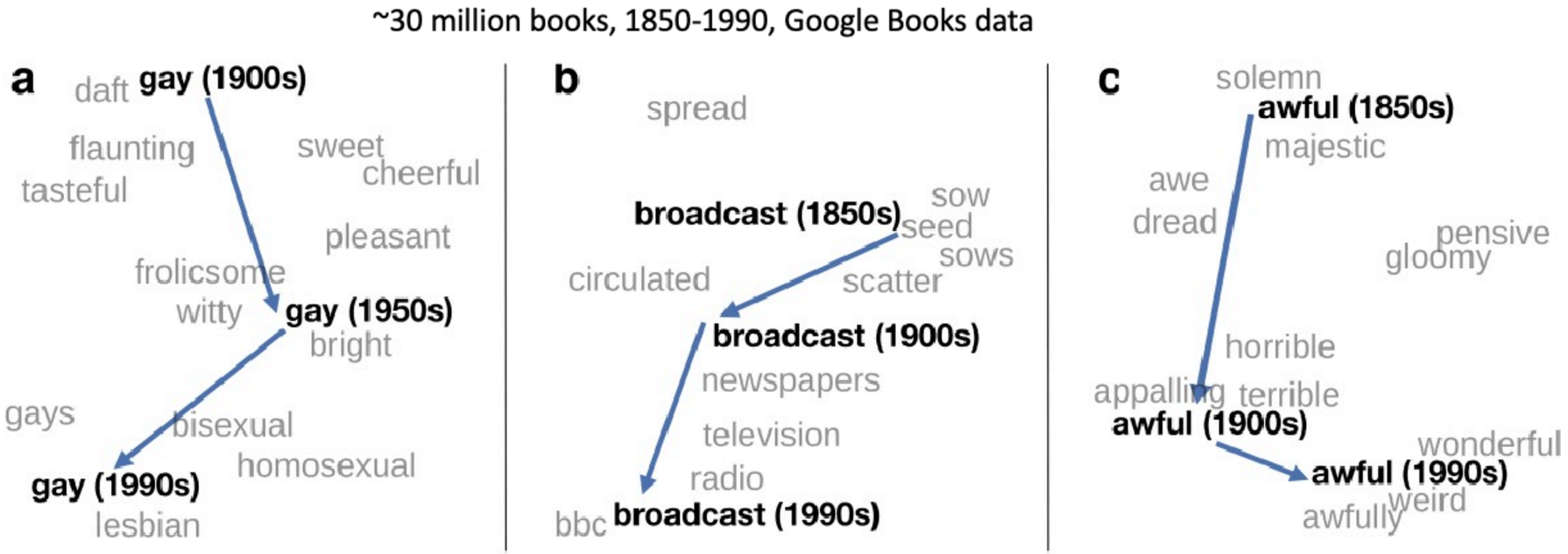
$$v(\text{cuatro}) \approx Wv(\text{four})$$



(Mikolov et al, 2013): Exploiting Similarities among Languages for Machine Translation

Embeddings as a window onto historical semantics

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



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