Vector Semantics

Dense Vectors

Sparse versus dense vectors

- PPMI vectors are
  - **long** (length |V|= 20,000 to 50,000)
  - **sparse** (most elements are zero)
- Alternative: learn vectors which are
  - **short** (length 200-1000)
  - **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 represented as distinct dimensions; this fails to capture similarity between a word with *car* as a neighbor and a word with *automobile* as a neighbor

Three methods for getting short dense vectors

- Singular Value Decomposition (SVD)
  - A special case of this is called LSA – Latent Semantic Analysis
- "Neural Language Model"-inspired predictive models
  - skip-grams and CBOW
- Brown clustering

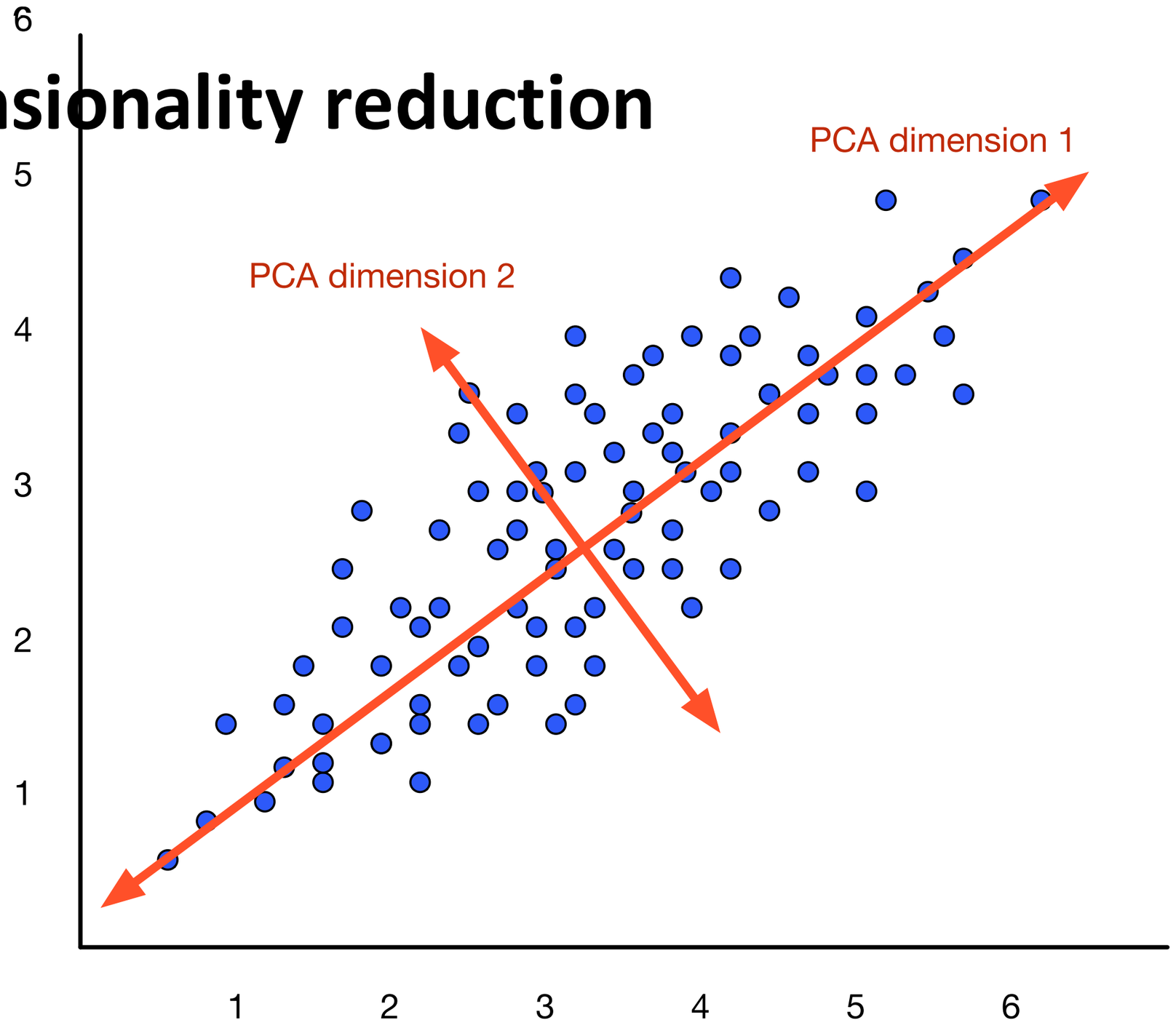
Vector Semantics

Dense Vectors via SVD

Intuition

- Approximate an N-dimensional dataset using fewer dimensions
- By first rotating the axes into a new space
- In which the highest order dimension captures the most variance in the original dataset
- And the next dimension captures the next most variance, etc.
- Many such (related) methods:
  - PCA – principle components analysis
  - Factor Analysis
  - SVD

Dimensionality reduction



Singular Value Decomposition

Any rectangular w x c matrix X equals the product of 3 matrices:

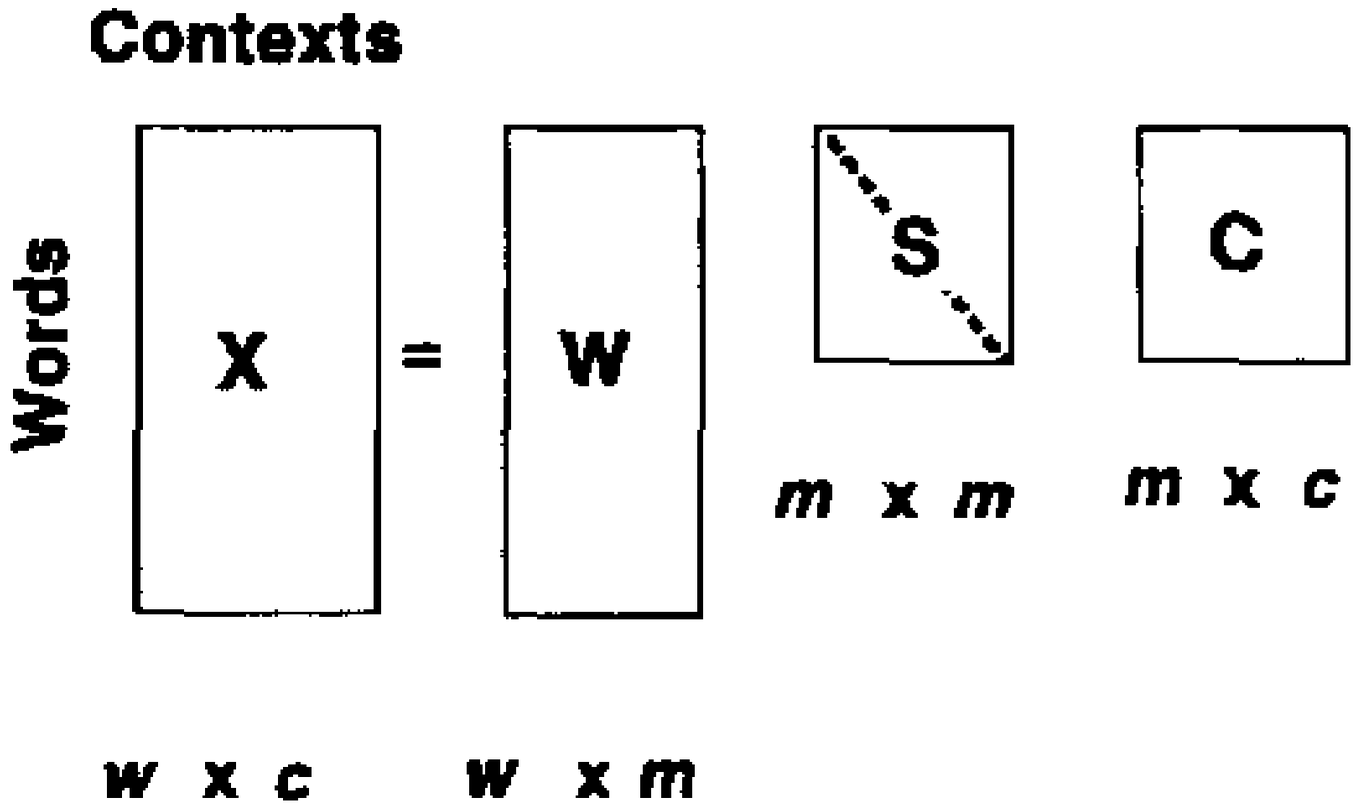
W: rows corresponding to original but m columns represents a dimension in a new latent space, such that

- M column vectors are orthogonal to each other
- Columns are ordered by the amount of variance in the dataset each new dimension accounts for

S: diagonal *m* x *m* matrix of **singular values** expressing the importance of each dimension.

C: columns corresponding to original but m rows corresponding to singular values

Singular Value Decomposition

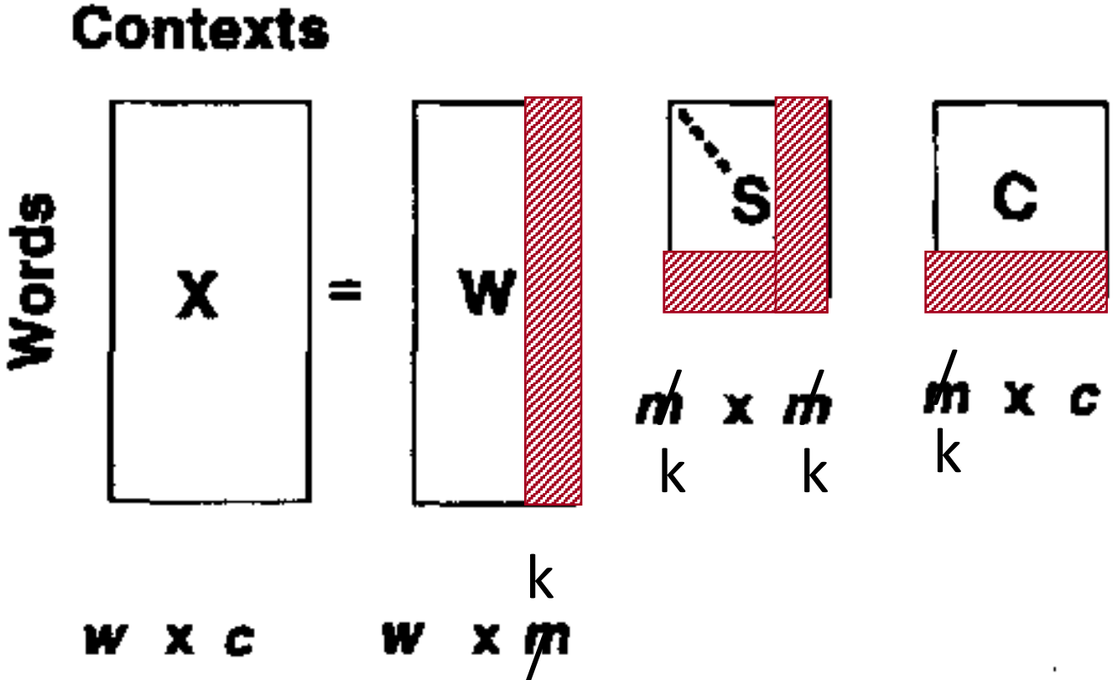


Landuaer and Dumais 1997

SVD applied to term-document matrix: Latent Semantic Analysis

Deerwester et al (1988)

- If instead of keeping all m dimensions, we just keep the top k singular values. Let's say 300.
- The result is a least-squares approximation to the original X
- But instead of multiplying, we'll just make use of W.
- Each row of W:
  - A k-dimensional vector
  - Representing word W



LSA more details

- 300 dimensions are commonly used
- The cells are commonly weighted by a product of two weights
  - Local weight: Log term frequency
  - Global weight: either idf or an entropy measure

Let's return to PPMI word-word matrices

- Can we apply to SVD to them?

SVD applied to term-term matrix

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

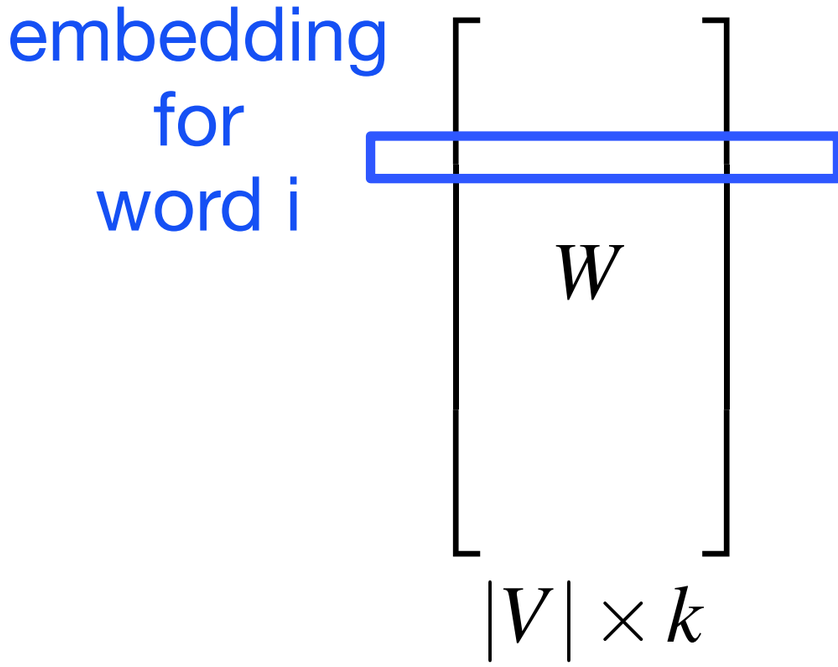
(I'm simplifying here by assuming the matrix has rank |V|)

Truncated SVD on term-term matrix

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

Truncated SVD produces embeddings

- Each row of W matrix is a k-dimensional representation of each word *w*
- K might range from 50 to 1000
- Generally we keep the top k dimensions, but some experiments suggest that getting rid of the top 1 dimension or even the top 50 dimensions is helpful (Lapesa and Evert 2014).



Embeddings versus sparse vectors

- Dense SVD embeddings sometimes work better than sparse PPMI matrices at tasks like word similarity
  - Denoising: low-order dimensions may represent unimportant information
  - Truncation may help the models generalize better to unseen data.
  - Having a smaller number of dimensions may make it easier for classifiers to properly weight the dimensions for the task.
  - Dense models may do better at capturing higher order co-occurrence.

Vector Semantics

Embeddings inspired by neural language models: skip-grams and CBOW

Prediction-based models: An alternative way to get dense vectors

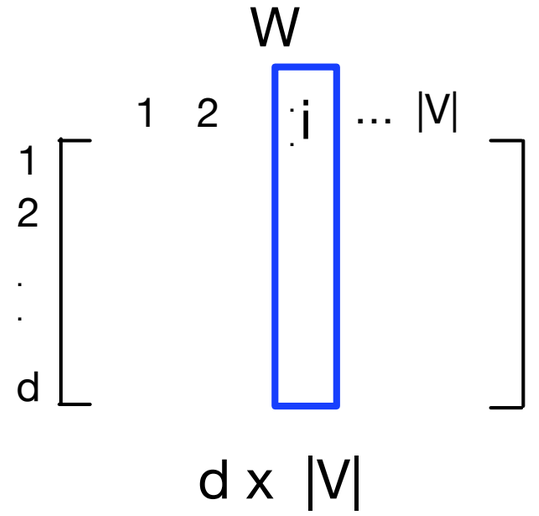
- **Skip-gram** (Mikolov et al. 2013a) **CBOW** (Mikolov et al. 2013b)
- Learn embeddings as part of the process of word prediction.
- Train a neural network to predict neighboring words
  - Inspired by **neural net language models**.
  - In so doing, learn dense embeddings for the words in the training corpus.
- Advantages:
  - Fast, easy to train (much faster than SVD)
  - Available online in the `word2vec` package
  - Including sets of pretrained embeddings!

Skip-grams

- Predict each neighboring word
  - in a context window of 2*C* words
  - from the current word.
- So for C=2, we are given word w_t and predicting these 4 words:

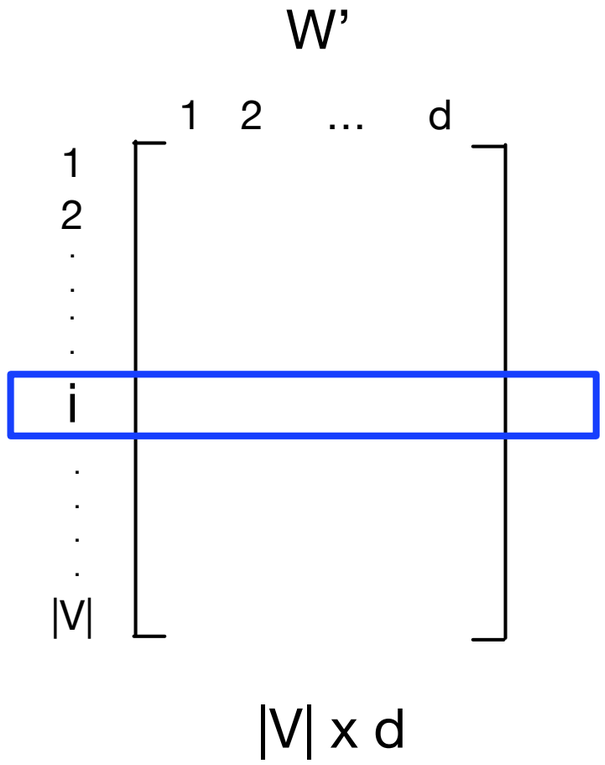
$$[w_{t-2}, w_{t-1}, w_{t+1}, w_{t+2}]$$

Skip-grams learn 2 embeddings for each w



input embedding *v,* in the input matrix *W*

- Column *i* of the input matrix *W* is the $1 \times d$ embedding v_i for word *i* in the vocabulary.



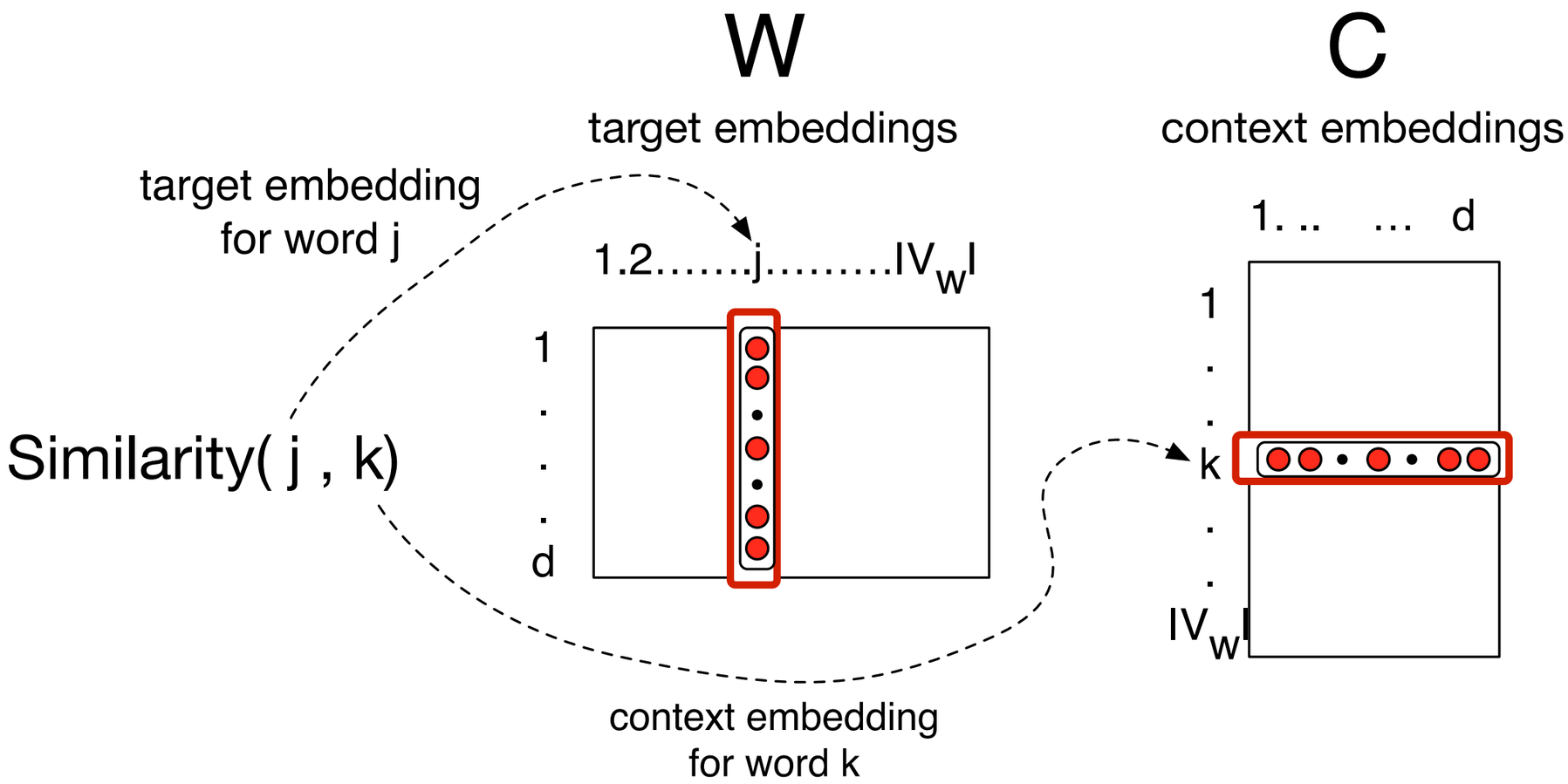
output embedding *v*', in output matrix W'

- Row *i* of the output matrix *W*' is a $d \times 1$ vector embedding v'_i for word *i* in the vocabulary.

Setup

- Walking through corpus pointing at word $w(t)$, whose index in the vocabulary is j, so we'll call it w_j ($1 < j < |V|$).
- Let's predict $w(t+1)$, whose index in the vocabulary is k ($1 < k < |V|$). Hence our task is to compute $P(w_k|w_j)$.

Intuition: similarity as dot-product between a target vector and context vector



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:
  - Similarity(j,k) $\propto c_k \cdot v_j$
- We'll need to normalize to get a probability

Turning dot products into probabilities

- $\text{Similarity(j,k)} = c_k \cdot v_j$

- We use softmax to turn into probabilities

$$p(w_k|w_j) = \frac{exp(c_k \cdot v_j)}{\sum_{i \in |V|} exp(c_i \cdot v_j)}$$

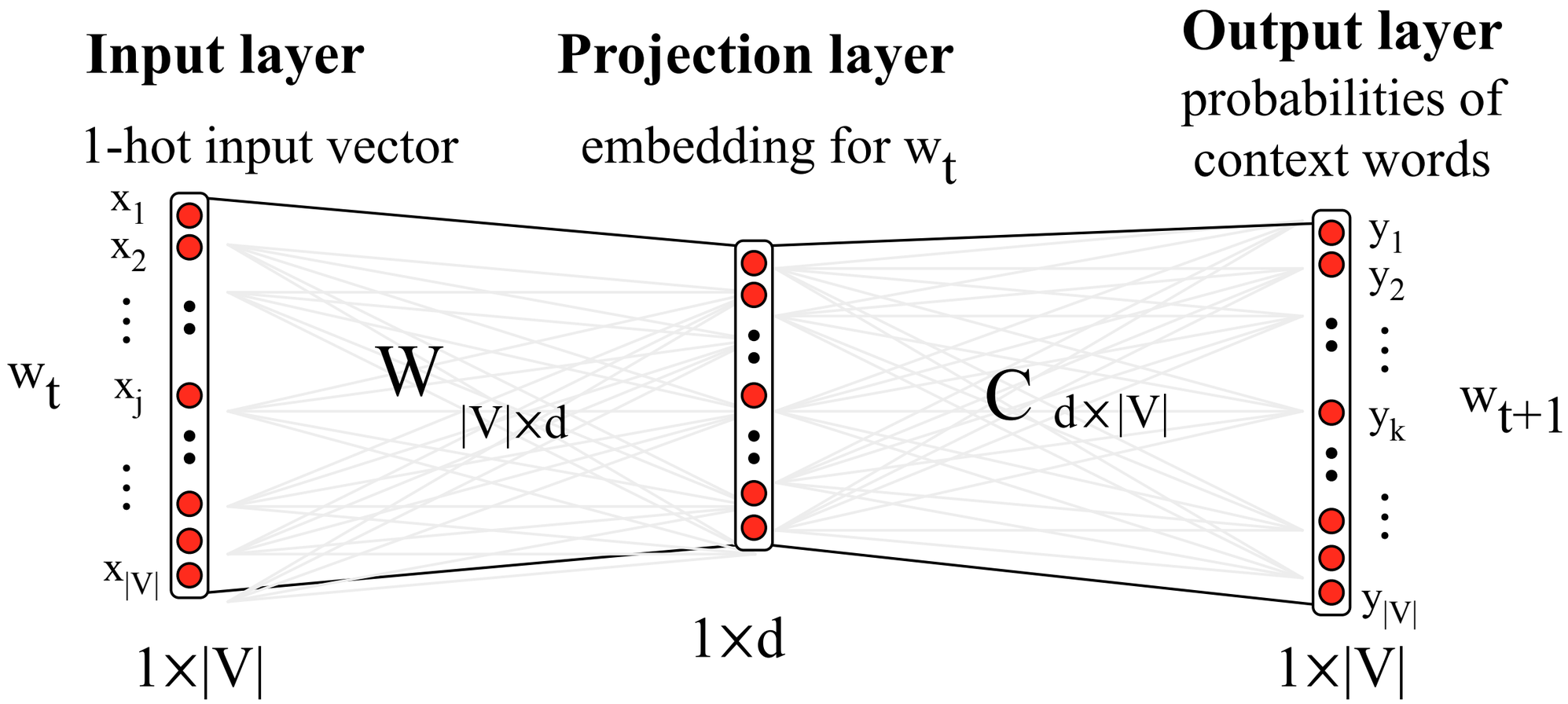
Embeddings from W and W’

- Since we have two embeddings, v_j and c_j for each word w_j
- We can either:
  - Just use v_j
  - Sum them
  - Concatenate them to make a double-length embedding

Learning

- Start with some initial embeddings (e.g., random)
- iteratively make the embeddings for a word
  - more like the embeddings of its neighbors
  - less like the embeddings of other words.

Visualizing W and C as a network for doing error backprop



One-hot vectors

- A vector of length |V|
- 1 for the target word and 0 for other words
- So if “popsicle” is vocabulary word 5
- The **one-hot vector** is
- [0,0,0,0,1,0,0,0,0…….0]

w_0 w_1 w_j $w_{|V|}$

0 0 0 0 0 … 0 0 0 0 1 0 0 0 0 0 … 0 0 0 0

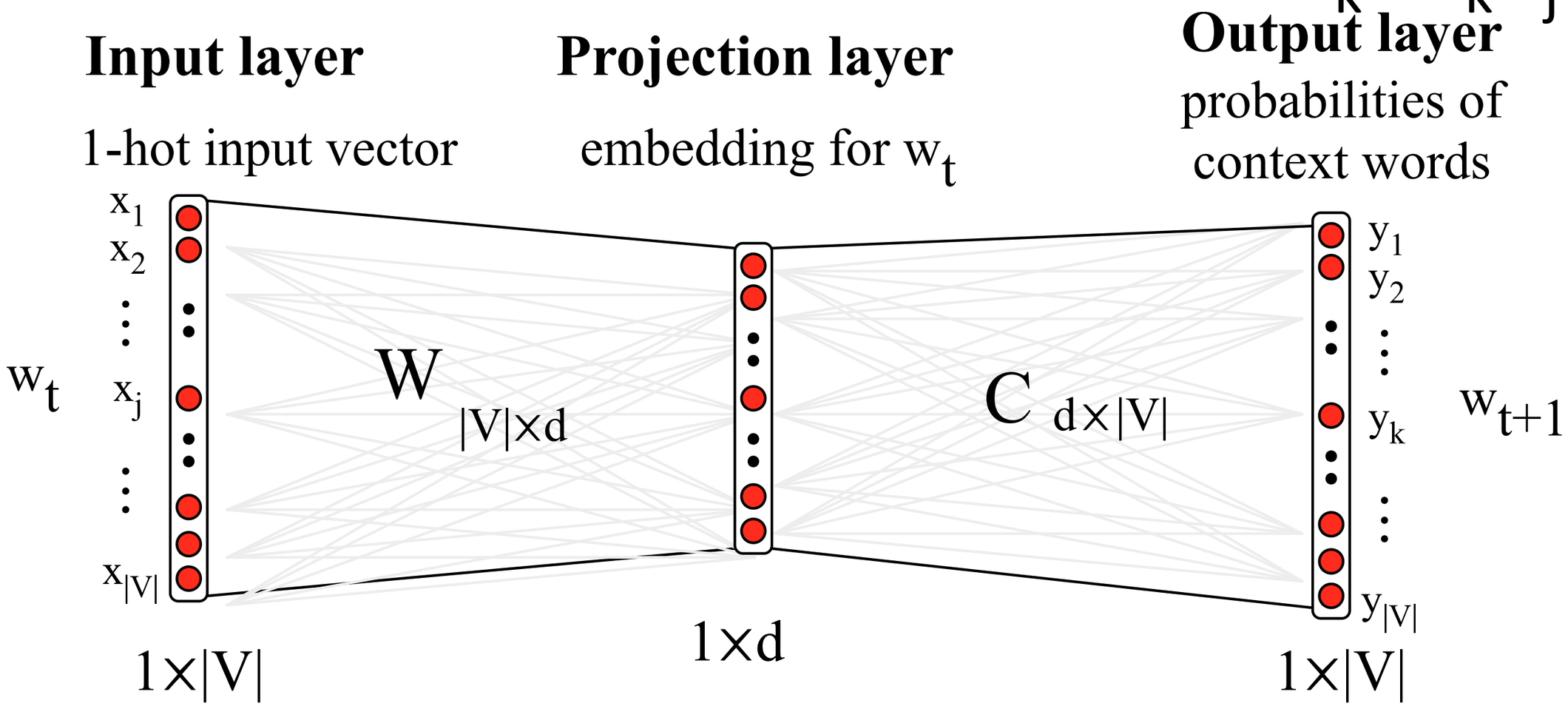
Skip-gram

$$h = v_j$$

$$o = Ch$$

$$o_k = c_k h$$

$$o_k = c_k \cdot v_j$$



Problem with the softamx

- The denominator: have to compute over every word in vocab

$$p(w_k|w_j) = \frac{exp(c_k \cdot v_j)}{\sum_{i \in |V|} exp(c_i \cdot v_j)}$$

- Instead: just sample a few of those negative words

Goal in learning

- Make the word like the context words

```
lemon,  a [tablespoon of apricot preserves or] jam
           c1          c2    w     c3       c4
```

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

- We want this to be high:

$$\sigma(c1 \cdot w) + \sigma(c2 \cdot w) + \sigma(c3 \cdot w) + \sigma(c4 \cdot w)$$

- And not like *k* randomly selected "noise words"

```
[cement metaphysical dear coaxial    apricot attendant whence forever puddle]
 n1     n2           n3   n4              n5        n6      n7       n8
```

- We want this to be low:

$$\sigma(n1 \cdot w) + \sigma(n2 \cdot w) + \ldots + \sigma(n8 \cdot w)$$

Skipgram with negative sampling: Loss function

$$\log \sigma(c \cdot w) + \sum_{i=1}^{\kappa} \mathbb{E}_{w_i \sim p(w)} \left[\log \sigma(-w_i \cdot w) \right]$$

Relation between skipgrams and PMI!

- If we multiply WW'^T
- We get a |V|x|V| matrix M , each entry m_{ij} corresponding to some association between input word i and output word j
- Levy and Goldberg (2014b) show that skip-gram reaches its optimum just when this matrix is a shifted version of PMI:

$$WW'^T = M^{\text{PMI}} - \log k$$

- So skip-gram is implicitly factoring a shifted version of the PMI matrix into the two embedding matrices.

Properties of embeddings

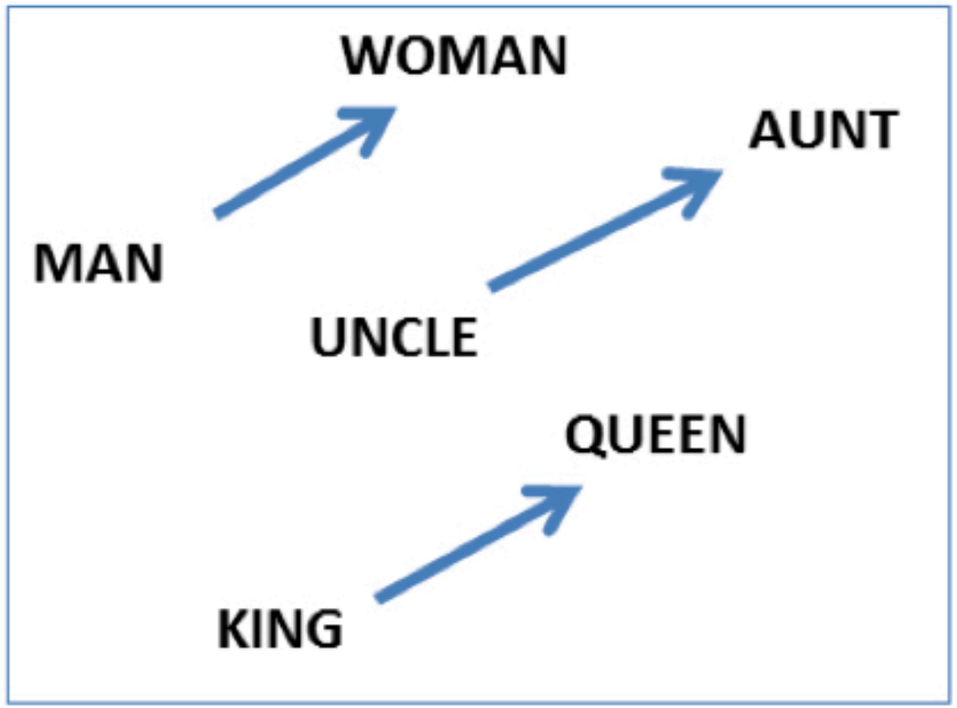
- Nearest words to some embeddings (Mikolov et al. 20131)

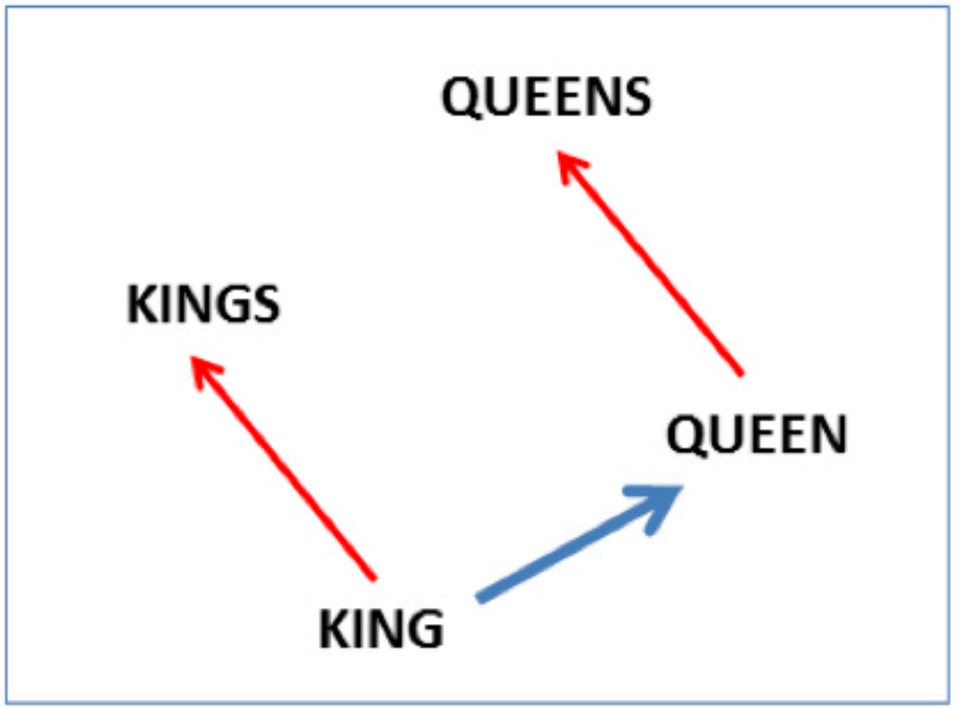
target:	Redmond	Havel	ninjutsu	graffiti	capitulate
	Redmond Wash.	Vaclav Havel	ninja	spray paint	capitulation
	Redmond Washington	president Vaclav Havel	martial arts	grafitti	capitulated
	Microsoft	Velvet Revolution	swordsmanship	taggers	capitulating

Embeddings capture relational meaning!

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

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





Vector Semantics

Brown clustering

Brown clustering

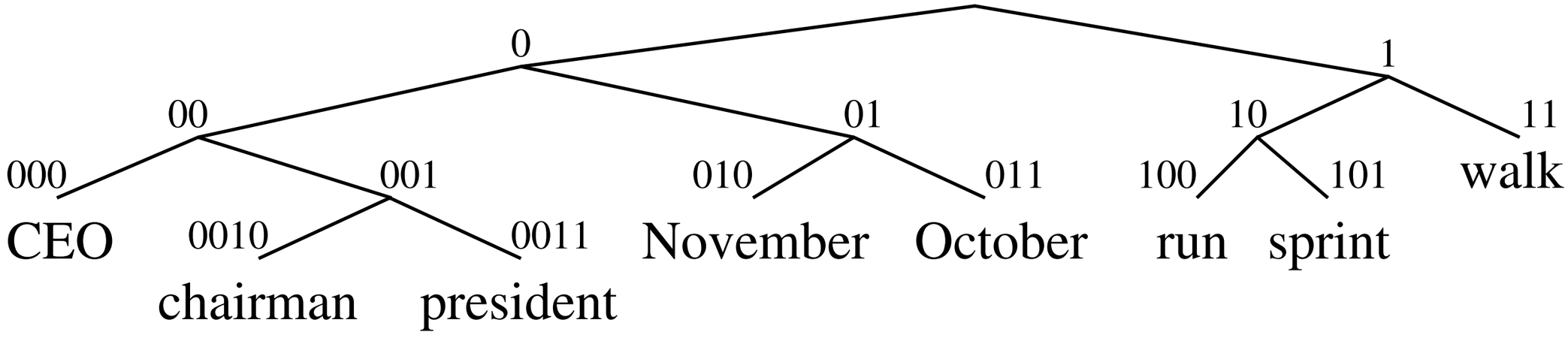
- An agglomerative clustering algorithm that clusters words based on which words precede or follow them
- These word clusters can be turned into a kind of vector
- We'll give a very brief sketch here.

Brown clustering algorithm

- Each word is initially assigned to its own cluster.
- We now consider consider merging each pair of clusters. Highest quality merge is chosen.
  - Quality = merges two words that have similar probabilities of preceding and following words
  - (More technically quality = smallest decrease in the likelihood of the corpus according to a class-based language model)
- Clustering proceeds until all words are in one big cluster.

Brown Clusters as vectors

- By tracing the order in which clusters are merged, the model builds a binary tree from bottom to top.
- Each word represented by binary string = path from root to leaf
- Each intermediate node is a cluster
- Chairman is 0010, "months" = 01, and verbs = 1



Brown cluster examples

Friday Monday Thursday Wednesday Tuesday Saturday Sunday weekends Sundays Saturdays
June March July April January December October November September August
pressure temperature permeability density porosity stress velocity viscosity gravity tension
anyone someone anybody somebody
had hadn’t hath would’ve could’ve should’ve must’ve might’ve
asking telling wondering instructing informing kidding reminding bothering thanking deposing
mother wife father son husband brother daughter sister boss uncle
great big vast sudden mere sheer gigantic lifelong scant colossal
down backwards ashore sideways southward northward overboard aloft downwards adrift

Class-based language model

- Suppose each word was in some class c_i:

$$P(w_i|w_{i-1}) = P(c_i|c_{i-1})P(w_i|c_i)$$

$$P(\text{corpus}|C) = \prod_{i-1}^{n} P(c_i|c_{i-1})P(w_i|c_i)$$