Dense Vectors

Dr. Demetrios Glinos
University of Central Florida

CAP6640 – Computer Understanding of Natural Language

Today

- Introduction
- Singular Value Decomposition
- Skip-Grams and CBOW
- Brown Clustering

Dense Vectors

- PPMI vectors are
  - long – length |V|, typically 20,000 to 50,000 terms
  - sparse - most elements are zero

- Suppose we can represent words using vectors that are
  - short – typically 50 to 1,000 terms
  - dense – most elements are nonzero

Advantages of Short Dense Vectors

- Easier to use as features in machine learning
  - fewer weights to tune

- May generalize better and avoid overfitting
  - depending on how we define the elements of the dense vecto

- May do a better job of representing synonyms

  Example:

  - *car* and *automobile* are synonyms
  - but they are different dimensions in a typical sparse vector model

Methods for Producing Short Dense Vectors

- Directionality reduction methods
  - e.g., SVD
- Neural network approaches
  - skip-grams
  - CBOW
- Agglomerative clustering
  - Brown clustering

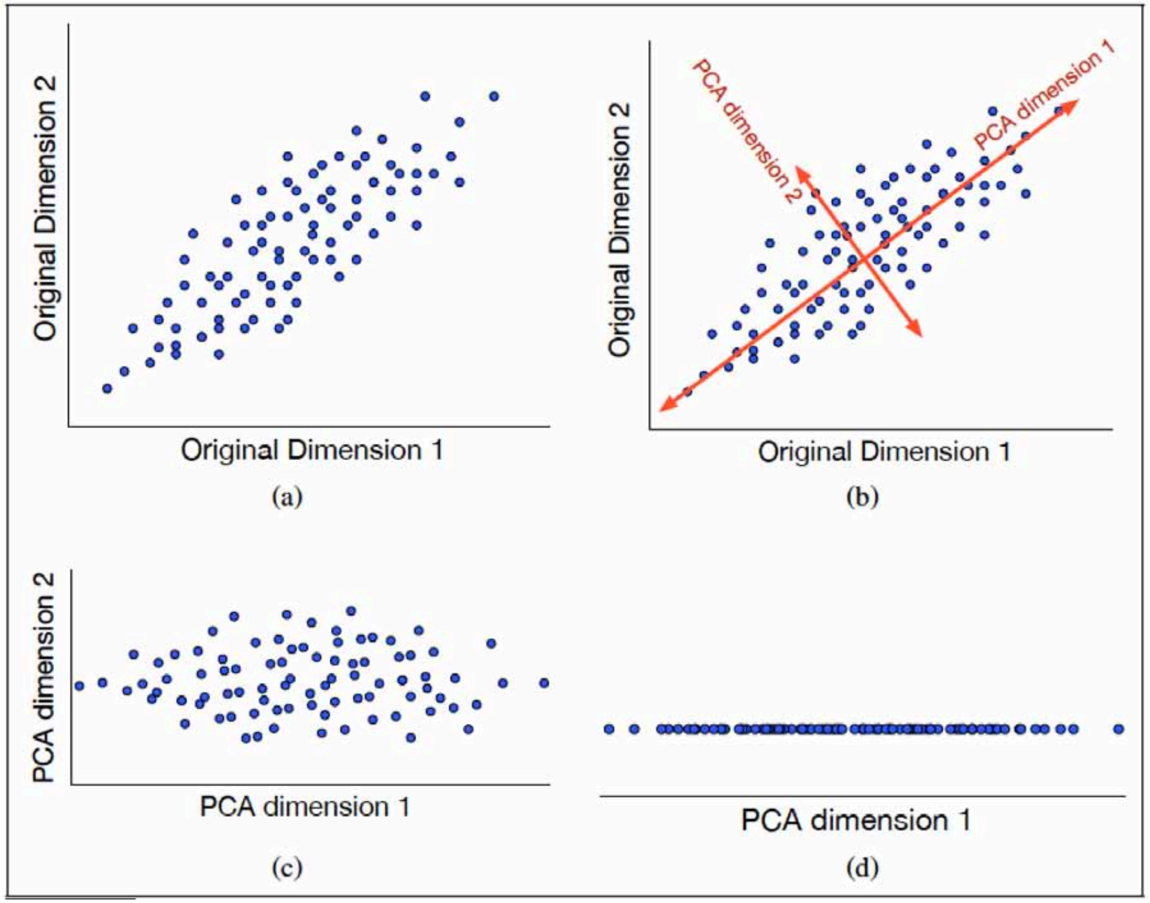
Today

- Introduction
- Singular Value Decomposition
- Skip-Grams and CBOW
- Brown Clustering

Singular Value Decomposition

- A classic dimension-reducing mathematical technique that finds the most important dimensions of a data set
  - i.e., the dimensions along which the data varies the most
- Can be applied to any rectangular matrix
- Intuition:
  - rotate the axes into a new space
  - highest order dimension captures the highest variance in the original dataset
  - next dimension captures the next highest variance, etc.
- Many related methods do this
  - PCA (principal component analysis)
  - Factor Analysis
  - SVD

Dimension Reduction: Visualization



source: J&M (3d Ed. draft), Fig. 16.1

Note: some information is becessarily lost when drop a dimension, the remaining dimension preserves the most that any one dimension could

Singular Value Decomposition

- SVD factors any w x c matrix X into the product of 3 matrices

$$X = W\ S\ C$$

- where

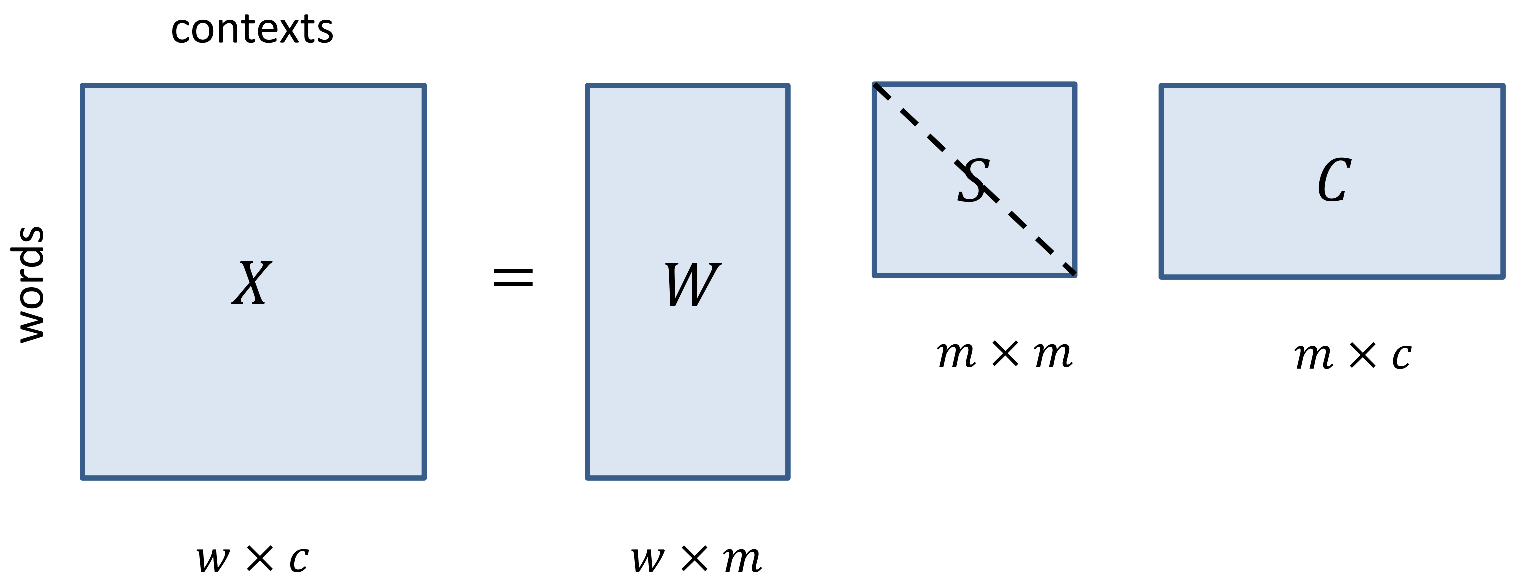
W : matrix with rows same as X but has m columns in the new latent space such that

- M column vectors are orthogonal to each other
- columns are in descending order of variance in the dataset for each new dimension

S : a diagonal m x m matrix of singular values, which express the importance of each dimension

C : has columns corresponding to the original, but has m rows corresponding to the singular values

SVD Matrix Factorization

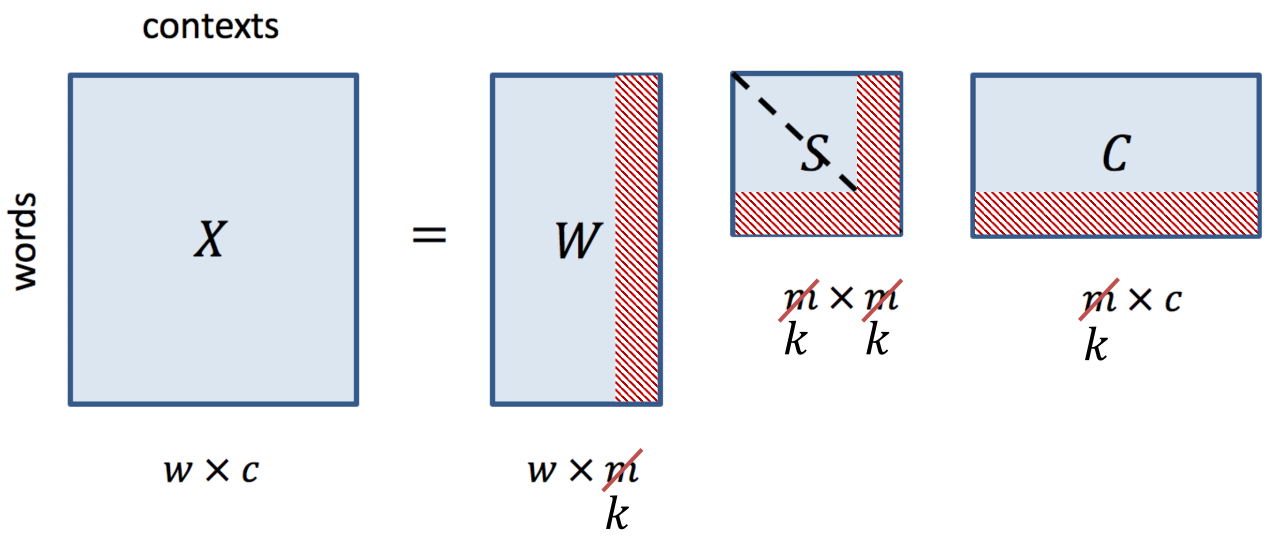


$$X = W\ S\ C$$

Latent Semantic Analysis

- applies SVD to a $|V| \times c$ term-document matrix X, where c = # documents
- instead of keeping all m dimensions, just keep the top k dimensions
- result is a least-squares approximation to the original X
- and instead of multiplying, we just use W
- each row of W is a k-dimensional vector that represents a word in V

Latent Semantic Analysis


contexts
words
X
$=$
W
S
C
$w \times c$
$w \times \cancel{m}$ k
$\cancel{m} \times \cancel{m}$ k k
$\cancel{m} \times c$ k

- 300 dimensions are commonly used
- cells are weighted by product of two weights
  - Local weight: log term frequency = $\log f(i,j) + 1$
  - Global weight: a version of the entropy = $1 + \frac{\sum_j p(i,j) \log p(i,j)}{log D}$

where D = # documents

SVD applied to term-term matrix

- We can apply SVD to a co-occurrence matrix X, and use only the top k dimensions

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

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

Figure 16.2 SVD factors a matrix X into a product of three matrices, W, Σ, and C. Taking the first k dimensions gives a $|V| \times k$ matrix W_k that has one k-dimensioned row per word that can be used as an embedding.

source: J&M (3d Ed. draft)

Summary: SVD Embedding

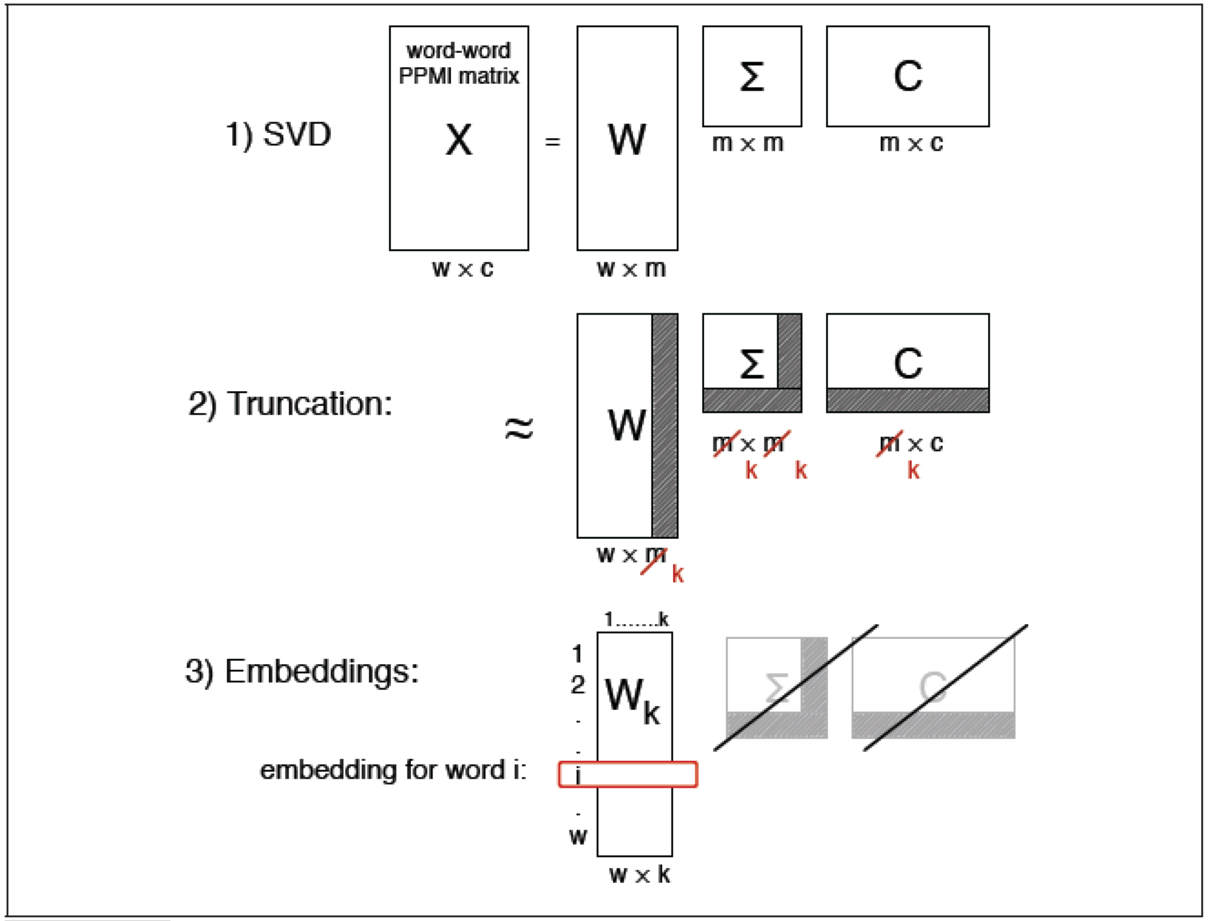


Figure 16.3 Sketching the use of SVD to produce a dense embedding of dimensionality k from a sparse PPMI matrix of dimensionality c. The SVD is used to factorize the word-word PPMI matrix into a W, Σ, and C matrix. The Σ and C matrices are discarded, and the W matrix is truncated giving a matrix of k-dimensionality embedding vectors for each word.

source: J&M (3d Ed. draft)

Embeddings versus sparse vectors

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

Today

- Introduction
- Singular Value Decomposition
- Skip-Grams and CBOW
- Brown Clustering

Dense Vectors from Prediction-Based Models

- Neural network language models are given a word and predict context words
- Intuition
  - the prediction process can be used to learn embeddings
  - because words with similar meanings often occur near each other in text
  - what neural models do is to learn an embedding by starting with a random vector and then iteratively evolving a word's embedding to be more like the embeddings of neighboring words
- most popular methods
  - skip-gram
  - CBOW
  - both are contained in the **`word2vec`** package
    - originally developed at Google
    - implementations in C, Java/Scala, and Python
    - algorithms are fast, efficient to train, and available online

Skip-grams

- Task is to predict each neighboring word
  - in a context window of 2C words
  - relative to the current word

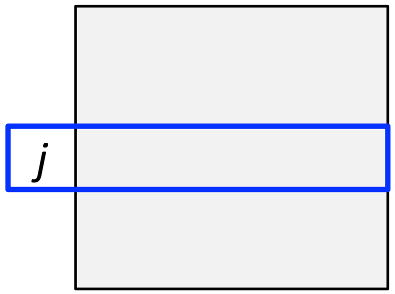
  - Example: For C = 2, and given word w_t we wish to predict these 4 words:

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

Skip-grams

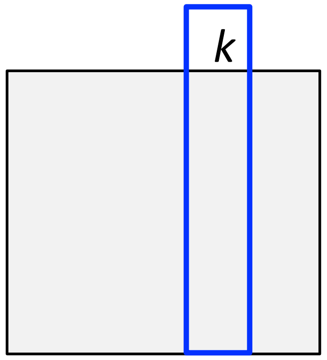
- Skip grams learn 2 separate embeddings for each word
    - a word (target) embedding in word matrix W

$v_j =$



    - a context embedding in context matrix C

$c_k =$



Skip-Gram Similarity Intuition

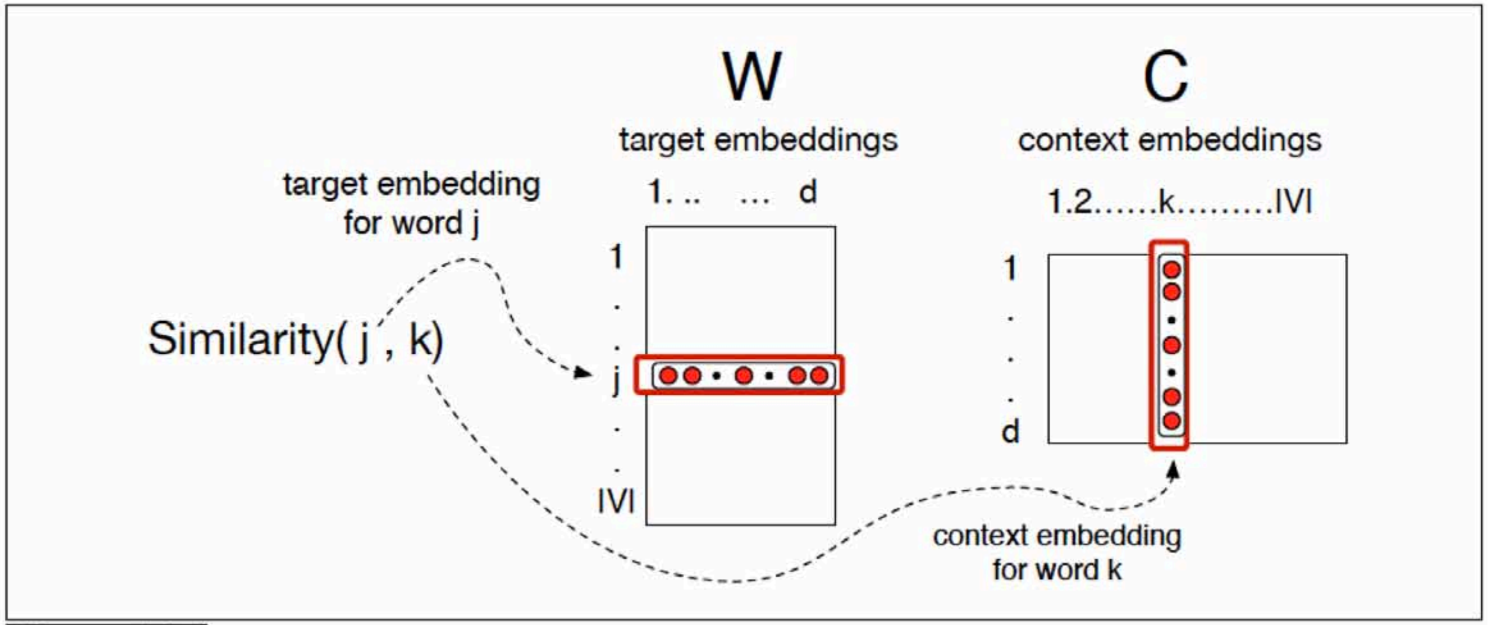


Figure 16.4

source: J&M (3d Ed. draft)

Skip-gram Prediction Task

- Skip-gram computes similarity as the dot product of the word (target) vector for w_j and the context vector for w_k

$$Similarity(j,k) = p(w_k | w_j) \propto c_k \cdot v_j$$

- Since dot product is in range (-∞,+∞) we must normalize to get a probability
- We use the *softmax* function, which converts an arbitrary real vector into one with probabilities that sum to 1

$$\sigma : \mathbb{R}^K \to [0,1]^K$$

$$\sigma(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \quad \text{for } j = 1, \ldots, K.$$

- Result is:

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

CBOW

- Continuous Bag of Words (CBOW)
  - essentially a mirror image of the skip-gram model
  - also a predictive model
  - but here, predicts the current word w_t from the words in the surrounding context window
  - produces similar embeddings to skip-gram, but they have slightly different behavior
    - often try both for a task and choose the one that performs better

Learning Word and Context Embeddings

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

- Basic probability is optimized when
  - a word's vector is closest to the words near it (numerator)
  - and farther from all other words (denominator)
- Computing the denominator is very expensive computationally
- skip-gram with negative sampling
  - approximates the denominator
  - during training, for each word in the context window, choose k random "noise" words outside of window
  - noise words are sampled from the vocabulary V according to their weighted unigram probability (in practice, usually this value raised to .75 power)
  - training involves iteratively shifting values (e.g., using gradient descent) to maximize dot product with the words in the context window and to minimize the dot products with the noise words

Example: Skip-Gram with Negative Sampling

Suppose we have running text with current word *apricot* and context words as shown:

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

Now, in practice, skip-gram actually uses a sigmoid function σ of the dot product, where

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

So, we want to maximize $\sigma(c1 \cdot w) + \sigma(c2 \cdot w) + \sigma(c3 \cdot w) + \sigma(c4 \cdot w)$

Suppose we also choose 2 random noise words according to their unigram frequencies:

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

For which we wish to minimize $\sigma(n1 \cdot w) + \sigma(n2 \cdot w) + \ldots + \sigma(n8 \cdot w)$

Producing the loss function: $\log \sigma(c,w) + \sum_{i=1}^{K} E_{w_i \sim p(w)}[\log \sigma(-w_i \cdot w)]$

Summary: Skip-Gram Learning

- Start with initial random embeddings for target and context matrices
- Iteratively make the embeddings for a word
  - more like the embeddings of its neighbors
  - less like the embeddings of other words
- Result is two embedding matrices
  - we can just use the resultant target matrix for embedding into the vector space
  - we can sum the two vectors
  - we can concatenate them to make a double-length embedding
- Window size is usually a tuning parameter
  - smaller windows capture more syntactic information
  - larger windows capture more semantic and relational information

Relation Between Skip-Grams and PMI

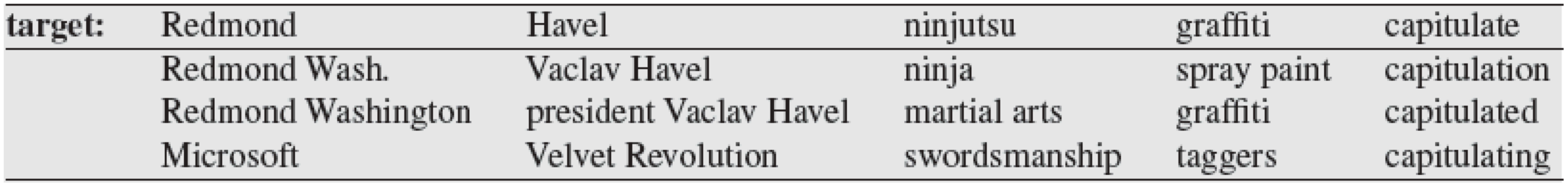
- If we multiply the target and context matrices we get a |V| x |V| matrix X
- Each entry x_{ij} of X corresponds to some association between input word i and output word j
- Levy and Goldberg (2014) have shown that this skip-gram reaches its optimum value just when this matrix is a shifted version of PMI

$$WC = X^{PMI} - \log k$$ where k is the number of negative samples in the skip-gram

- Thus, skip-gram is implicitly factorizing a shifted version of the PMI matrix into the two embedding matrices W and C, similar to what SVD did, although of course using a different factorization

Properties of Embeddings

- Some results from a phrase-based version of the skip-gram algorithm



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

Figure 16.7 Examples of the closest tokens to some target words using a phrase-based extension of the skip-gram algorithm (Mikolov et al., 2013a).

source: J&M (3d Ed. draft)

Note how similarities are found with words/phrases that are semantically, but not morphologically, similar

Vector Offsets

- *Offsets* between vector embeddings can sometimes capture important relations between words

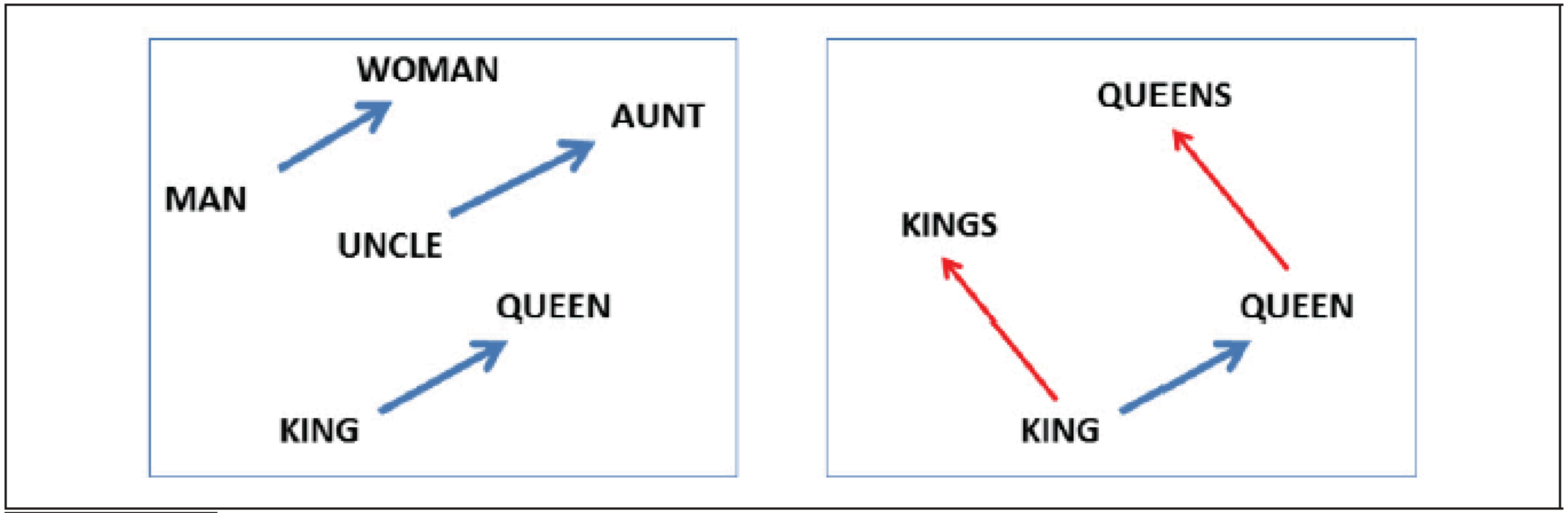


Figure 16.8 Vector offsets showing relational properties of the vector space, shown by projecting vectors onto two dimensions using PCA. In the left panel, 'king' - 'man' + 'woman' is close to 'queen'. In the right, we see the way offsets seem to capture grammatical number (Mikolov et al., 2013b).

source: J&M (3d Ed. draft)

Today

- Introduction
- Singular Value Decomposition
- Skip-Grams and CBOW
- Brown Clustering

Class-Based Language Models

- Class-based language model
  - each word w belongs to class c with probability P(w|c)
  - transition probabilities are modeled as transitions between classes, not words

$$P(w_i \mid w_{i-1}) = P(c_i \mid c_{i-1})\, P(w_i \mid c_i)$$

- A class-based LM can assign a probability to an entire corpus given a particular clustering C as follows:

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

- Not generally used for machine translation or speech recognition
- But important for Brown clustering

Brown Clustering

- **Brown clustering** is an agglomerative clustering technique

  1. Each word is initially assigned to its own cluster
  2. Consider merging each pair of clusters and merge the pair whose merger resuls in the smallest decrease in the likelihood of the corpus
  3. Continue clustering until all words are in one big cluster

- Two words will most likely be clustered if they have similar probabilities for preceding and following words
  - i.e., if they are contextually similar
  - this leads to more coherent clusters

Binary Tree of Clusters

- By keeping track of the order in which clusters are merged, the model builds a binary tree from the bottom up
  - leaves are the words in V
  - each intermediate node represents the cluster that is formed by merging its children
- Method is a "hard clustering" algorithm – each word belongs to exactly 1 cluster
- Intermediate nodes represent potentially useful features

Example: Brown Cluster

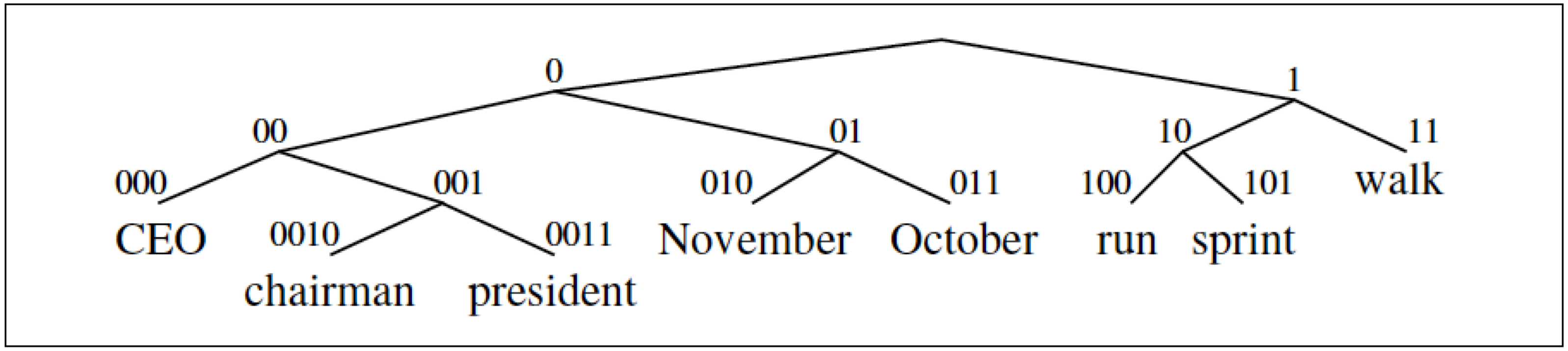


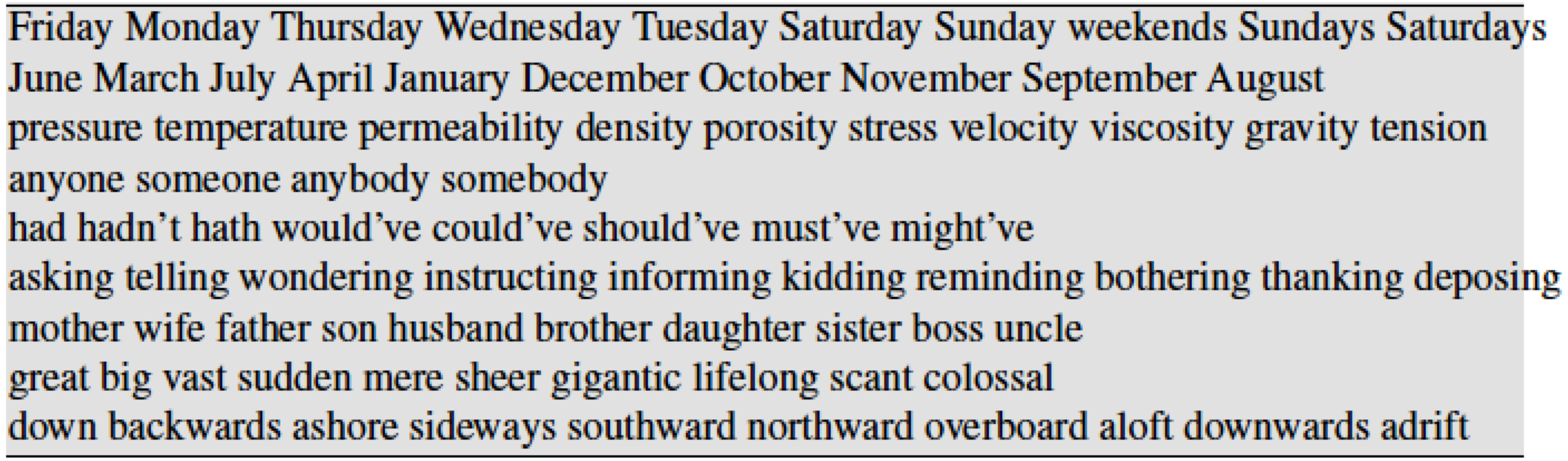
Figure 16.9 Brown clustering as a binary tree. A full binary string represents a word; each binary prefix represents a larger class to which the word belongs and can be used as a vector representation for the word. After Koo et al. (2008).

source: J&M (3d Ed. draft)

- after clustering, a word or feature can be represented by the binary string (vector) that corresponds to its path from the root node
  - use 0 for left, 1 for right, at each choice point

- For example above
  - *chairman* is 0010, *months* = 01, *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

Figure 16.10 Some sample Brown clusters from a 260,741-word vocabulary trained on 366 million words of running text (Brown et al., 1992). Note the mixed syntactic-semantic nature of the clusters.

source: J&M (3d Ed. draft)

Brown Clustering Complexity

- Basic (naïve algorithm) is O(n^5)

  - for |V| = n
    - n iterations
    - consider n^2 merges
    - for each merge, compute the value of the clustering by summing over n^2 terms

- In practice, use more efficient O(n^3) algorithms that use tables to pre-compute values for each merge