Word Representation

SLP3 Ch 6; INLP Ch 14

Word representations
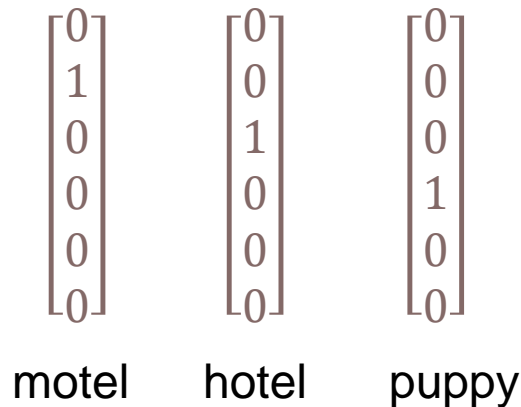
- In traditional NLP, we regard words as discrete symbols
- Problems
  - no natural notion of similarity
    - motel vs hotel
  - may rely on a thesaurus, e.g., WordNet, but:
    - Compiling or updating a thesaurus requires a lot of work, thus:
      - Unavailable for low-resource languages / domains
      - Not always up-to-date
      - Many words and phrases are missing
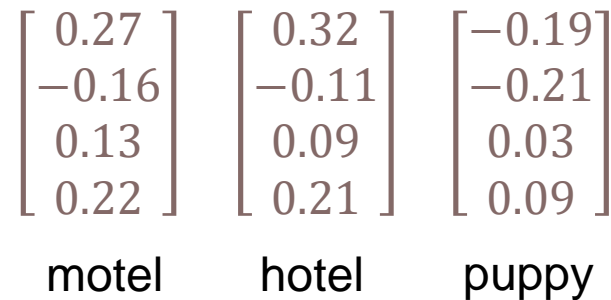
Word vectors

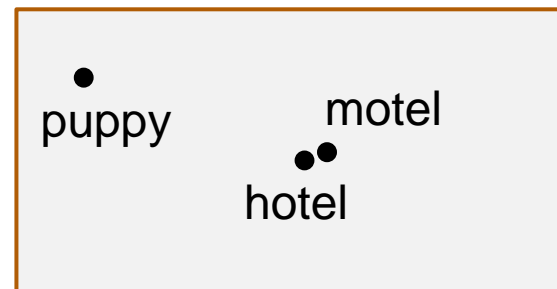▸ Representing each word in a vector space

   ▸ also known as word embeddings

one-hot vectors

$$\begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \quad \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} \quad \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}$$

motel   hotel   puppy

dense vectors

$$\begin{bmatrix} 0.27 \\ -0.16 \\ 0.13 \\ 0.22 \end{bmatrix} \quad \begin{bmatrix} 0.32 \\ -0.11 \\ 0.09 \\ 0.21 \end{bmatrix} \quad \begin{bmatrix} -0.19 \\ -0.21 \\ 0.03 \\ 0.09 \end{bmatrix}$$

motel   hotel   puppy

project to 2D

Word vectors

- Representing each word in a vector space
  - also known as word embeddings
- Advantages
  - Can capture similarity between words with vector distances
  - Easier to process in many ML systems
  - Can be learned from data!

Word vectors

- Sparse vector representations
  - Co-occurrence matrices
- Dense vector representations
  - Singular value decomposition
  - Word2vec
  - Contextualized word embedding (will be discussed later)

Static word embedding

Sparse vector representations
-- co-occurrence matrices

Co-occurrence matrices

- We represent how often a word occurs in a document
  - Term-document matrix
- Or how often a word co-occurs with another word
  - Term-term matrix (or word-word co-occurrence matrix or word-context matrix)

Term-document matrix

▸ Each cell: count of word w in a document

▸ Each document is a **count vector in** $\mathbb{N}^V$ (a column below)

Shakespeare's plays

	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	1	1	8	15
soldier	2	2	12	36
fool	37	58	1	5
clown	6	117	0	0

Term-document matrix

▸ Each cell: count of word w in a document

▸ Each document is a **count vector in** $\mathbb{N}^V$ (a column below)

▸ Two documents are similar if their vectors are similar

Shakespeare's plays

	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	1	1	8	15
soldier	2	2	12	36
fool	37	58	1	5
clown	6	117	0	0

Term-document matrix

▸ Each word is a **count vector in** $\mathbb{N}^D$ (a row below)

Shakespeare's plays

	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	1	1	8	15
soldier	2	2	12	36
fool	37	58	1	5
clown	6	117	0	0

▸

Term-document matrix

▸ Each word is a **count vector in** $\mathbb{N}^D$ (a row below)

▸ Two words are similar if their vectors are similar

Shakespeare's plays

	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	1	1	8	15
soldier	2	2	12	36
fool	37	58	1	5
clown	6	117	0	0

Word-word matrix

- Each cell: count of word co-occurrence
- Instead of co-occurrence in entire documents, use smaller contexts
  - Paragraph
  - Window of ±4 words
- A word is now defined by a vector over counts of context words
  - Instead of each vector being of length D, each vector is now of length |V|
- The word-word matrix is |V|x|V|

Word-word matrix (context of ± 7 words)

sugar, a sliced lemon, a tablespoonful of **apricot** preserve or jam, a pinch each of,

their enjoyment. Cautiously she sampled her first **pineapple** and another fruit whose taste she likened

well suited to programming on the digital **computer**. In finding the optimal R-stage policy from

for the purpose of gathering data and **information** necessary for the study authorized in the

	aardvark	computer	data	pinch	result	sugar	…
apricot	0	0	0	1	0	1	
pineapple	0	0	0	1	0	1	
digital	0	2	1	0	1	0	
information	0	1	6	0	4	0	
…	…						

Word-word matrix (context of ±7 words)

▸ ## Similar words have similar contexts & vectors

sugar, a sliced lemon, a tablespoonful of **apricot** preserve or jam, a pinch each of,

their enjoyment. Cautiously she sampled her first **pineapple** and another fruit whose taste she likened

well suited to programming on the digital **computer**. In finding the optimal R-stage policy from

for the purpose of gathering data and **information** necessary for the study authorized in the

	aardvark	computer	data	pinch	result	sugar	…
apricot	0	0	0	1	0	1	
pineapple	0	0	0	1	0	1	
digital	0	2	1	0	1	0	
information	0	1	6	0	4	0	
…	…						

Word-word matrix

- We showed only 4x6, but the real matrix is 50,000 x 50,000
  - So it's very sparse: most values are 0.
  - That's OK: there are lots of efficient algorithms for sparse matrices.
- The size of windows depends on your goals.
  - The shorter the windows ($\pm$1-3), the more syntactic the representation
  - The longer the windows ($\pm$4-10), the more semantic the representation

Co-occurrence matrices

- The co-occurrence matrices we have seen represent each cell by word frequencies.

- But raw frequency is a bad representation.

  - Frequency is clearly useful: if *sugar* appears a lot near *apricot*, that's useful information.

  - But overly frequent words like *the*, *it,* or *they* are not very informative about the context.

  - It's a paradox! How can we balance these two conflicting constraints?

Weighting

- tf-idf
  - usually used for term-document matrix
- PMI (pointwise mutual information)
  - usually used for term-term matrix

TF-IDF

‣ Term frequency

  ‣ $\text{tf}_{t,d} = \log_{10}(\text{count}(t,d)+1)$

‣ Document frequency of term t

  ‣ df_t is the number of documents that t occurs in.

‣ Inverse document frequency (idf)

$$\text{idf}_t \;=\; \log_{10}\left(\frac{N}{\text{df}_t}\right)$$

  ‣ N is the total number of documents in the collection

Word	df	idf
Romeo	1	1.57
salad	2	1.27
Falstaff	4	0.967
forest	12	0.489
battle	21	0.246
wit	34	0.037
fool	36	0.012
good	37	0
sweet	37	0

TF-IDF

- Final tf-idf weighted value for a word

$$w_{t,d} = \text{tf}_{t,d} \times \text{idf}_t$$

- Raw counts:

	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	1	0	7	13
good	114	80	62	89
fool	36	58	1	4
wit	20	15	2	3

- tf-idf:

	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	0.074	0	0.22	0.28
good	0	0	0	0
fool	0.019	0.021	0.0036	0.0083
wit	0.049	0.044	0.018	0.022

PMI

- PMI (pointwise mutual information)
  - Do events x and y co-occur more/less than if they were independent?

  $$PMI(X,Y) = \log_2 \frac{P(X,Y)}{P(X)P(Y)}$$

  - Here, events are word occurrences.

PPMI (Positive Pointwise Mutual Information)

▸ PMI ranges from $(-\infty, +\infty)$

▸ But negative values are problematic

   ▸ Things are co-occurring **less than** we expect by chance

   ▸ Unreliable without enormous corpora

      ▸ Imagine w1 and w2 whose probability is each 10^{-6}

      ▸ Hard to be sure P(w1,w2) is significantly lower than 10^{-12}

   ▸ Plus it's not clear people are good at "unrelatedness"

▸ So we just replace negative PMI values by 0

▸ Positive PMI (PPMI) between w1 and w2:

$$PPMI(w_1, w_2) = \max(\log_2 \frac{P(w_1, w_2)}{P(w_1)P(w_2)}, 0)$$

Computing PPMI on a term-context matrix

- Matrix F with W rows (words) and C columns (contexts)
- f_{ij} is # of times w_i occurs in context c_j

	computer	data	pinch	result	sugar	...
apricot	0	0	1	0	1	
pineapple	0	0	1	0	1	
digital	2	1	0	1	0	
information	1	6	0	4	0	

$$p_{ij} = f_{ij} \Big/ \sum_{i=1}^{W}\sum_{j=1}^{C} f_{ij} \qquad p_{i*} = \sum_{j=1}^{C} f_{ij} \Big/ \sum_{i=1}^{W}\sum_{j=1}^{C} f_{ij} \qquad p_{*j} = \sum_{i=1}^{W} f_{ij} \Big/ \sum_{i=1}^{W}\sum_{j=1}^{C} f_{ij}$$

$$\text{PMI}_{ij} = \log_2 \frac{p_{ij}}{p_{i*}\, p_{*j}} \qquad \text{PPMI}_{ij} = \begin{cases} \text{PMI}_{ij}, & \text{PMI}_{ij} > 0 \\ 0, & otherwise \end{cases}$$

Computing PPMI on a term-context matrix

$$p_{ij} = f_{ij} \Big/ \sum_{i=1}^{W} \sum_{j=1}^{C} f_{ij} \qquad p_{i*} = \sum_{j=1}^{C} f_{ij} \Big/ \sum_{i=1}^{W} \sum_{j=1}^{C} f_{ij} \qquad p_{*j} = \sum_{i=1}^{W} f_{ij} \Big/ \sum_{i=1}^{W} \sum_{j=1}^{C} f_{ij}$$

	P(w, context)					
	computer	data	pinch	result	sugar	**P(w)**
apricot	0.00	0.00	0.05	0.00	0.05	0.11
pineapple	0.00	0.00	0.05	0.00	0.05	0.11
digital	0.11	0.05	0.00	0.05	0.00	0.21
information	0.05	0.32	0.00	0.21	0.00	0.58
P(context)	0.16	0.37	0.11	0.26	0.11	

Computing PPMI on a term-context matrix

	P(w, context)					
	computer	data	pinch	result	sugar	P(w)
apricot	0.00	0.00	0.05	0.00	0.05	0.11
pineapple	0.00	0.00	0.05	0.00	0.05	0.11
digital	0.11	0.05	0.00	0.05	0.00	0.21
information	0.05	0.32	0.00	0.21	0.00	0.58
P(context)	0.16	0.37	0.11	0.26	0.11	

PMI(information,data) = log2 (.32 / (.37*.58)) = .57

PPMI	computer	data	pinch	result	sugar
apricot	-	-	2.25	-	2.25
pineapple	-	-	2.25	-	2.25
digital	1.66	0.00	-	0.00	-
information	0.00	0.57	0.00	0.47	-

Weighting PMI

‣ PMI is biased toward infrequent events

- ‣ Very rare words have very high PMI values

‣ Two solutions

- ‣ Give rare words slightly higher probabilities
- ‣ Use add-k smoothing (which has a similar effect)

Use Laplace (add-k) smoothing

Add-2 smoothed	computer	data	pinch	result	sugar	...
apricot	2	2	3	2	3	
pineapple	2	2	3	2	3	
digital	4	3	2	3	2	
information	3	8	2	6	2	

Add-2 smoothed	P(w, context)					
	computer	data	pinch	result	sugar	P(w)
apricot	0.03	0.03	0.05	0.03	0.05	0.20
pineapple	0.03	0.03	0.05	0.03	0.05	0.20
digital	0.07	0.05	0.03	0.05	0.03	0.24
information	0.05	0.14	0.03	0.10	0.03	0.36
P(context)	0.19	0.25	0.17	0.22	0.17	

Use Laplace (add-k) smoothing

PPMI	computer	data	pinch	result	sugar
apricot	-	-	2.25	-	2.25
pineapple	-	-	2.25	-	2.25
digital	1.66	0.00	-	0.00	-
information	0.00	0.57	0.00	0.47	-

Only shrink **non-zero and valid** PPMI values

PPMI (add-2)	computer	data	pinch	result	sugar
apricot	0.00	0.00	0.56	0.00	0.56
pineapple	0.00	0.00	0.56	0.00	0.56
digital	0.62	0.00	0.00	0.00	0.00
information	0.00	0.58	0.00	0.37	0.00

Dense vector representations
-- singular value decomposition (SVD)

SVD-based method

- 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.

SVD-based method

SVD
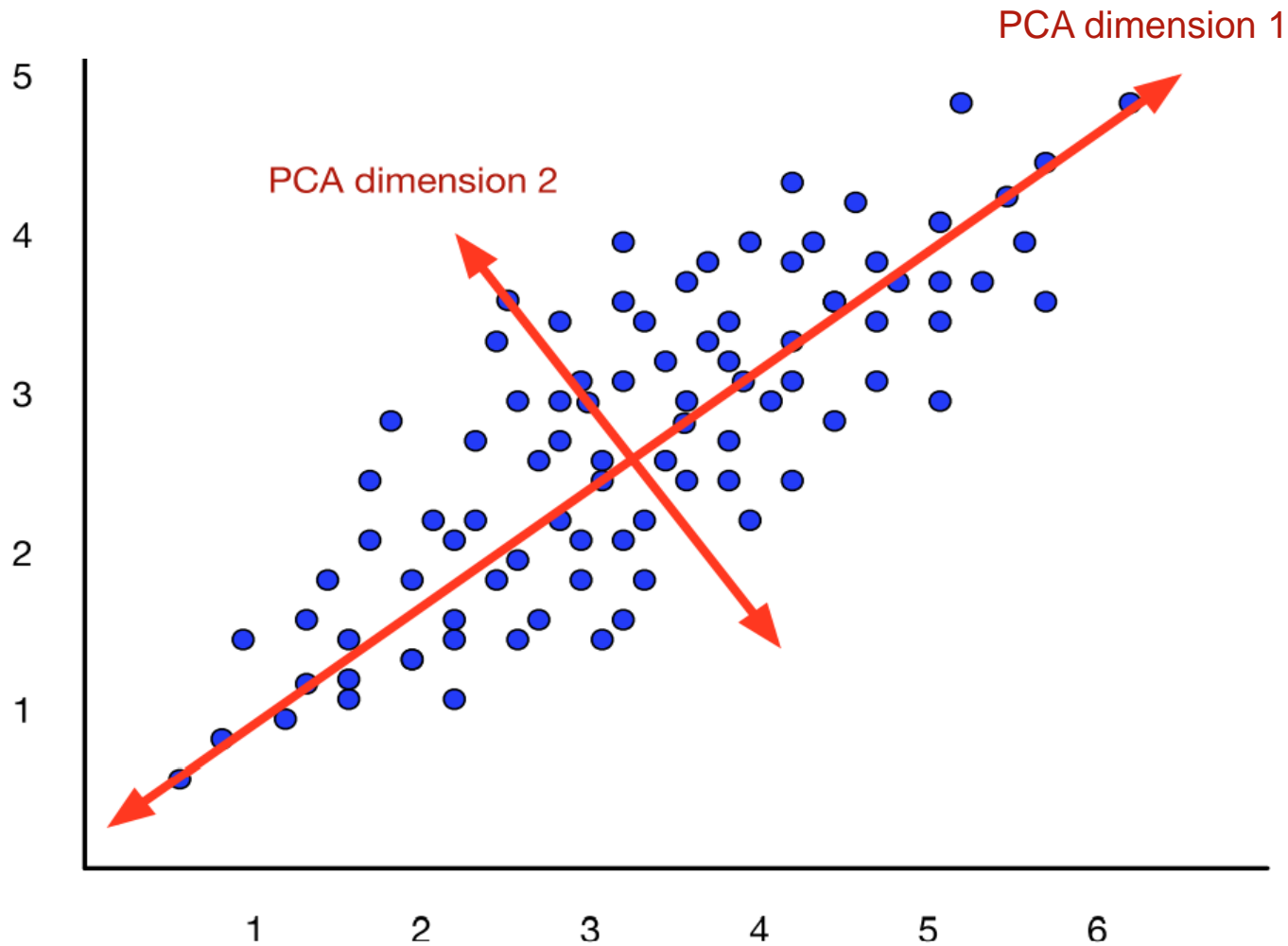
▸ Any *w x c* matrix *X* equals the product of 3 matrices:

  ▸ **W**: rows correspond to original, but *m* columns represent dimensions 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

  ▸ **C**: columns correspond to original, but *m* rows represent dimensions in a new latent space

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

SVD

$$X_{w \times c} = W_{w \times m} \; S_{m \times m} \; C_{m \times c}$$
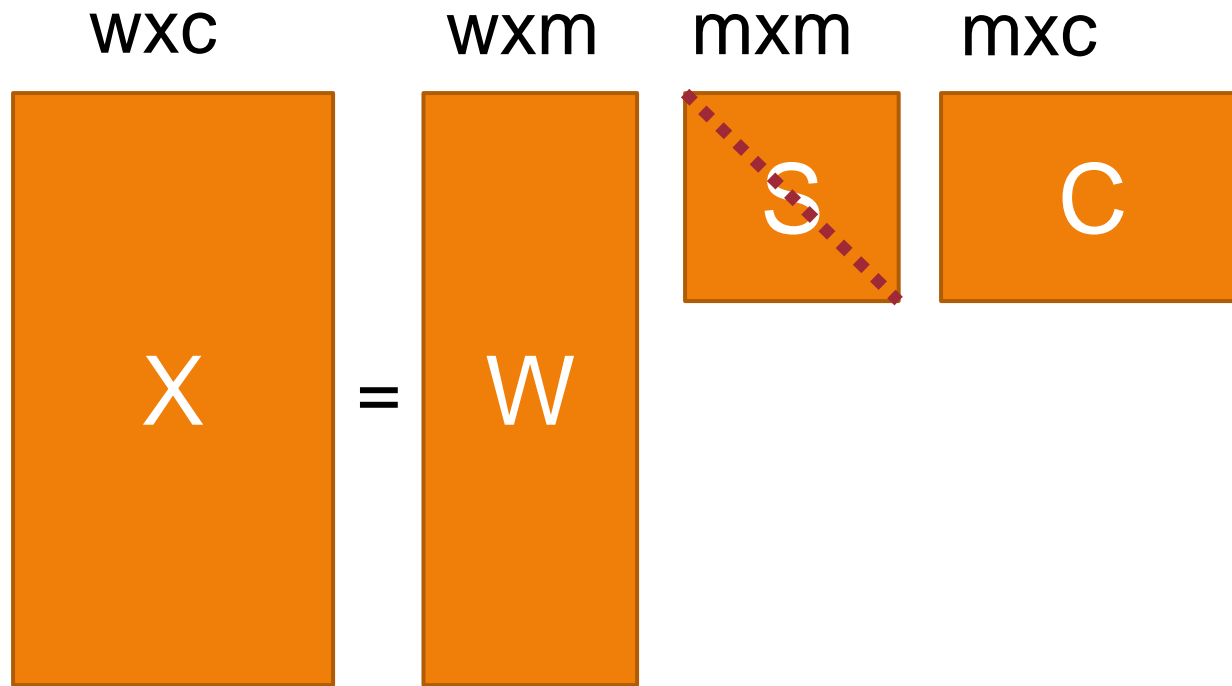
Latent Semantic Analysis (LSA)

▸ **SVD applied to term-document matrix** (Deerwester et al, 1988)

　▸ Instead of keeping all m dimensions, we just keep the top k singular values, say 300.

　▸ The result is a least-squares approximation to the original X

　▸ Use each row of W as a k-dimensional word representation

▸ Can also be applied to term-term matrix
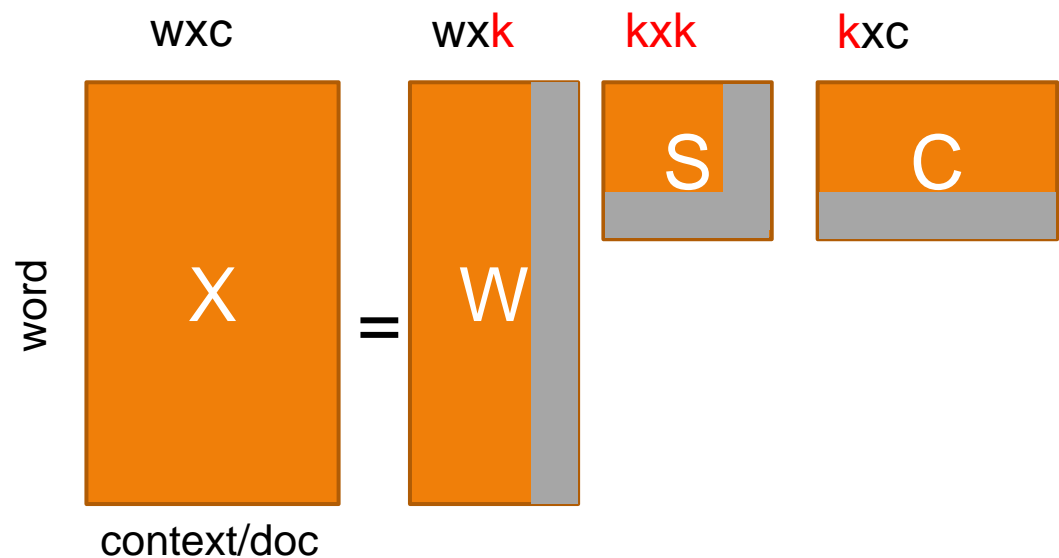
Latent Semantic Analysis (LSA)

‣ Dense SVD vectors often work better than sparse vectors

- ‣ Denoising: eliminate low-order dimensions that represent unimportant information

- ‣ Better generalizability to unseen data

- ‣ Fewer dimensions: smaller subsequent models, easier to train

Dense vector representations
-- Word2vec

Word2vec

- Neural word embedding
  - Learning word representations using neural networks
    - Word2vec, GloVe
  - Advantages
    - Can be integrated into end-to-end neural systems
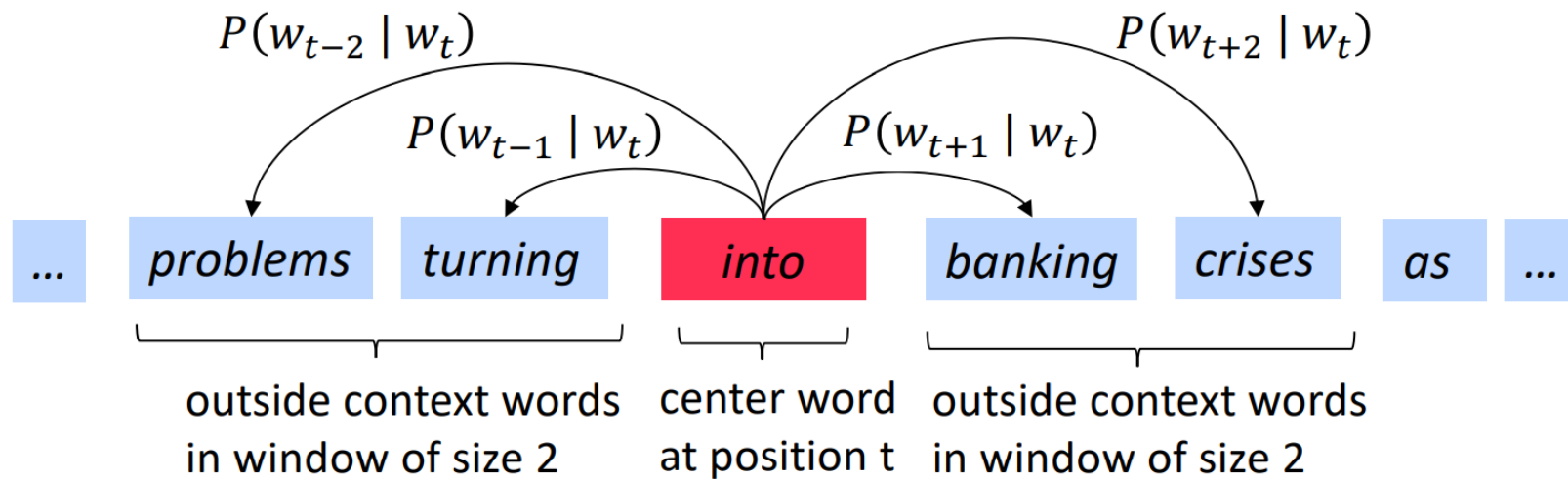   - Can become more meaningful through supervised learning from downstream tasks

Word2vec overview

▸ Word2vec (Mikolov et al. 2013) is a framework for learning word vectors

▸ Idea:

  ▸ We have a large corpus of text

  ▸ Every word in a fixed vocabulary is represented by a vector

  ▸ Go through each position t in the text, which has a center word c and several context ("outside") words o

  ▸ Use the similarity of word vectors for c and o to calculate the probability of o given c (or vice versa)

  ▸ Keep adjusting the word vectors to maximize this probability

▸

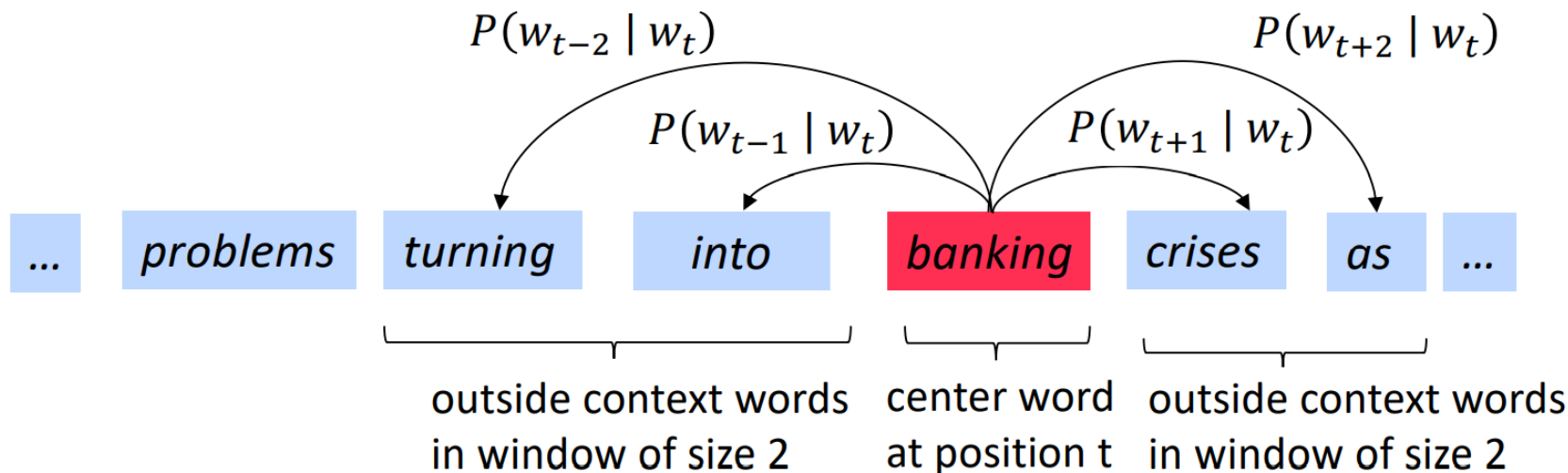# Word2vec skip-grams

▸ Example

$P(w_{t-2} \mid w_t)$ $P(w_{t+2} \mid w_t)$

$P(w_{t-1} \mid w_t)$ $P(w_{t+1} \mid w_t)$

… | problems | turning | into | banking | crises | as | …

outside context words in window of size 2   center word at position t   outside context words in window of size 2

Word2vec skip-grams

▸ Example

$$P(w_{t-2} \mid w_t)$$

$$P(w_{t+2} \mid w_t)$$

$$P(w_{t-1} \mid w_t)$$

$$P(w_{t+1} \mid w_t)$$

| … | problems | turning | into | banking | crises | as | … |

outside context words in window of size 2

center word at position t

outside context words in window of size 2

Word2vec skip-grams

▸ How to calculate $\log P(w_{t+j}|w_t; \theta)$

▸ Two vectors per word w:

  ▸ v_w when w is a center word; u_w when w is a context word

▸ Then for a center word c and a context word o

$$P(o|c) = \frac{\exp(u_o^T v_c)}{\sum_{w \in V} \exp(u_w^T v_c)}$$

Dot-product similarity; exp to make it positive

Normalize over entire vocabulary

▸ This is called a softmax function

  ▸ mapping arbitrary values to a probability distribution
  ▸ "max" because it amplifies largest values
  ▸ "soft" because it still retains smaller values

▸

Word2vec skip-grams object function

▸ For each $t = 1, \cdots, T$, predict context words within a window of fixed size m, given center word w_j.

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

θ is all variables to be optimized

sometimes called *cost* or *loss* function

The objective function $J(\theta)$ is the (average) negative log likelihood:

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

Minimizing objective function ⟺ Maximizing predictive accuracy

Word2vec skip-grams object function

$$J(\theta) = -\frac{1}{T} \log L(\theta) = -\frac{1}{T} \sum_{t=1}^{T} \sum_{\substack{-m \leq j \leq m \\ j \neq 0}} \log P\left(w_{t+j} \mid w_t; \theta\right)$$
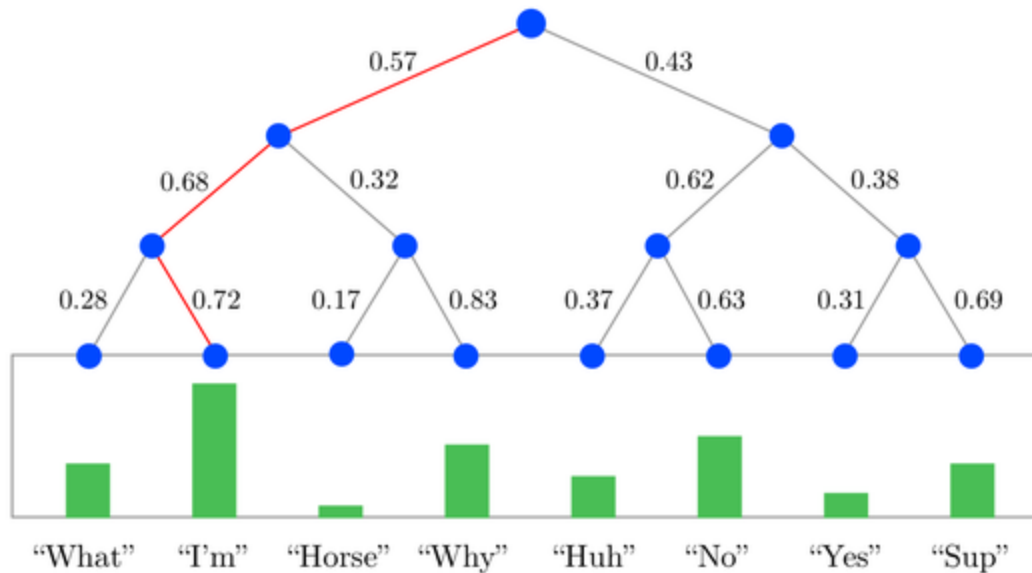
▸ Directly optimize $J(\theta)$ involves computing softmax
   ▸ Expensive because of normalizing over entire vocabulary

$$P(o|c) = \frac{\exp(u_o^T v_c)}{\sum_{w \in V} \exp(u_w^T v_c)}$$

Word2vec skip-grams object function

▸ One solution: hierarchical softmax
  ▸ $O(|V|) \rightarrow O(\log|V|)$

Word2vec skip-grams object function

‣ Another solution: negative sampling

  ‣ Take K negative samples (using word probabilities) instead of entire vocabulary

  $$P(o|c) = \frac{\exp(u_o^T v_c)}{\sum_{k=1}^{K} \exp(u_k^T v_c)}$$

  ‣ Maximize probability of real outside word, and minimize probability of negative samples (*contrastive learning*)

  ‣ A more widely used objective function for each pair of center and context words:

  $$-\log\left(\sigma(\boldsymbol{u}_o^T \boldsymbol{v}_c)\right) - \sum_{k=1}^{K} \log(\sigma(-\boldsymbol{u}_k^T \boldsymbol{v}_c)))$$

Optimization

‣ Stochastic gradient descent

  ‣ Gradient descent

    ‣ Perform update in the steepest downhill direction

    ‣ Ex: $g(w_1, w_2)$

    ‣ Update:

$$w_1 \leftarrow w_1 + \alpha * \frac{\partial g}{\partial w_1}(w_1, w_2)$$

$$w_2 \leftarrow w_2 + \alpha * \frac{\partial g}{\partial w_2}(w_1, w_2)$$
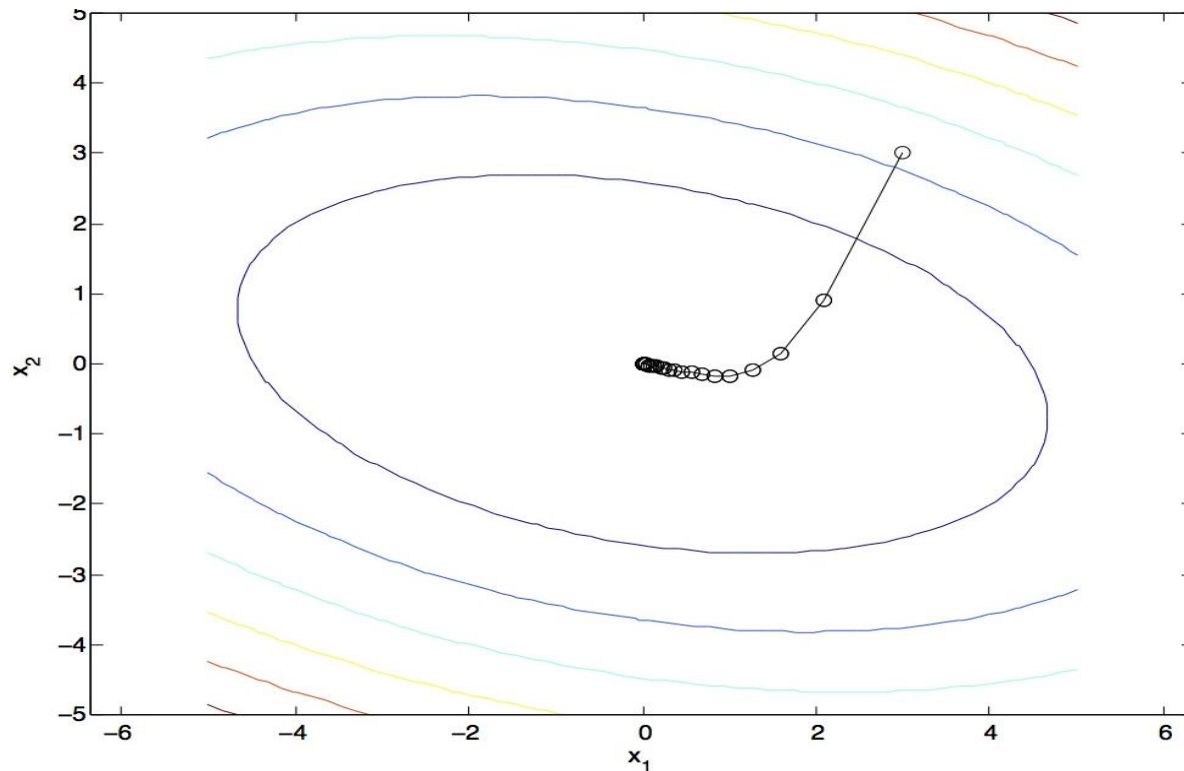
    ‣ In vector notation:

$$w \leftarrow w + \alpha * \nabla_w g(w)$$

    ‣ Nowadays: gradient computing using auto-differentiation

Optimization

- Stochastic gradient descent
  - Gradient descent
    - Perform update in the steepest downhill direction

Optimization

- Stochastic gradient descent
  - Gradient descent
    - Perform update in the steepest downhill direction
  - Stochastic
    - Perform update with gradient on one training example (or a mini-batch of examples)

Word2vec skip-grams

- Since we have two embeddings for a word
  - v_w when w is a center word
  - u_w when w is a context word
- We can either
  - Use v_w
  - Sum them
  - Concatenate them to make a double-length embedding

Word2vec CBOW

- Skip-grams (SG)
  - Predict context ("outside") words (position independent) given center word
- Continuous Bag of Words (CBOW)
  - Predict center word from (bag of) context words

Evaluating vector representations

Evaluate word embeddings

- Related to general evaluation in NLP: Intrinsic vs extrinsic
- Intrinsic:
  - Evaluation on a specific/intermediate subtask
  - Fast to compute
  - Helps to understand that system
  - Not clear if really helpful unless correlation to real task is established
- Extrinsic:
  - Evaluation on a real task
  - Can take a long time
  - Unclear if the subsystem is the problem or its interaction or other subsystems
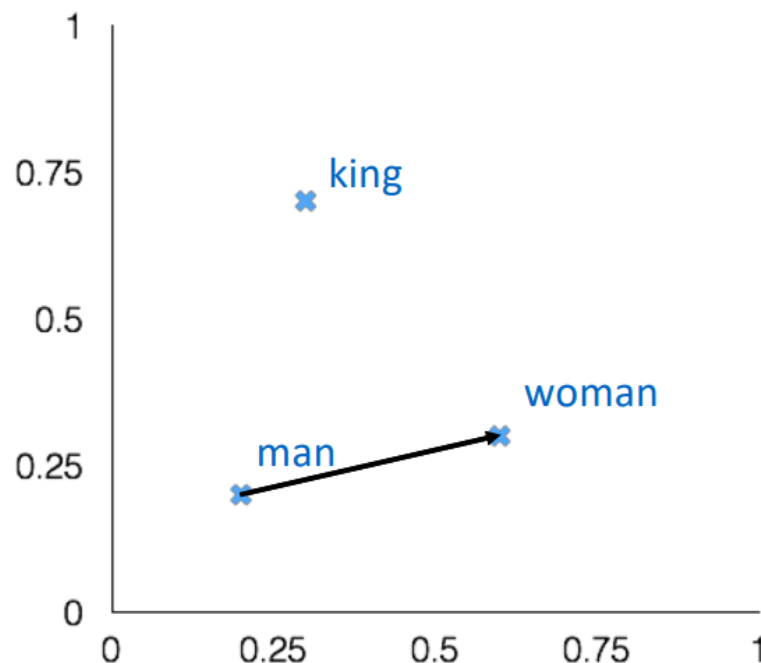
Intrinsic evaluation

‣ Word Vector Analogies

a:b :: c:?

man:woman :: king:?

$$d = \arg\max_i \frac{(x_b - x_a + x_c)^T x_i}{||x_b - x_a + x_c||}$$

‣ Evaluate word vectors by how well their cosine distance after addition captures intuitive semantic and syntactic analogy questions

‣ Problem: What if the information is there but not linear

Summary

Word Representation

- Sparse vector representations
  - Co-occurrence matrices
  - Weighting: tf-idf, PPMI
- Dense vector representations
  - Singular value decomposition
    - Latent Semantic Analysis
  - Word2vec
    - Skip-gram
- Evaluation