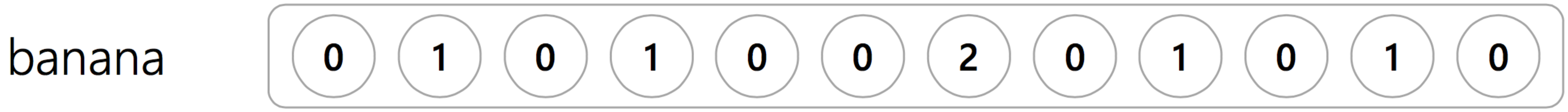
BIG DATA and AI for business

Deep Learning (2)
Word Embedding

Decisions, Operations & Information Technologies
Robert H. Smith School of Business
Fall, 2020

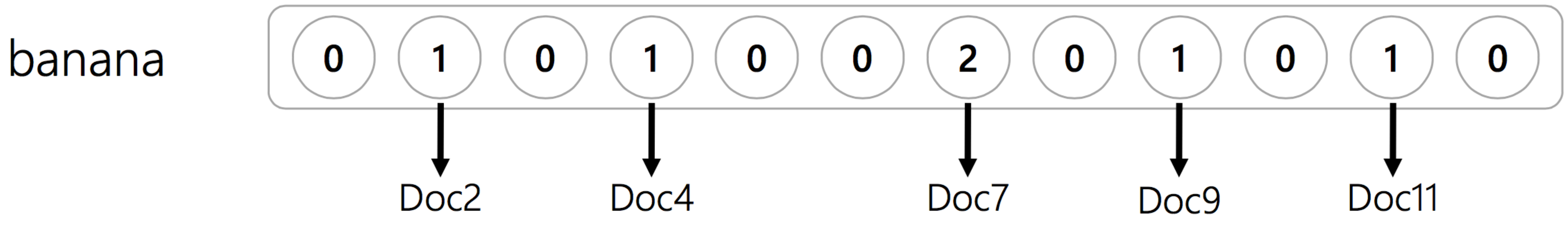
Vector space models

- Represent an item (e.g., word) as a vector of numbers



Vector space models

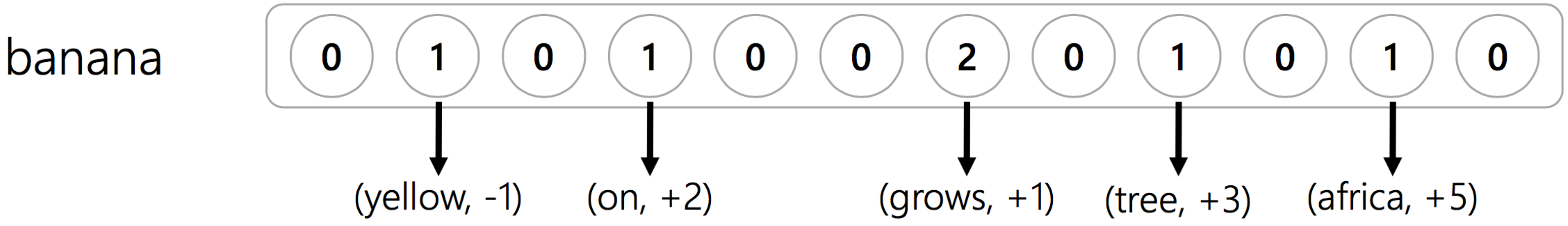
- Represent an item (e.g., word) as a vector of numbers



- The vector can correspond to documents in which the word occurs

Vector space models

- Represent an item (e.g., word) as a vector of numbers



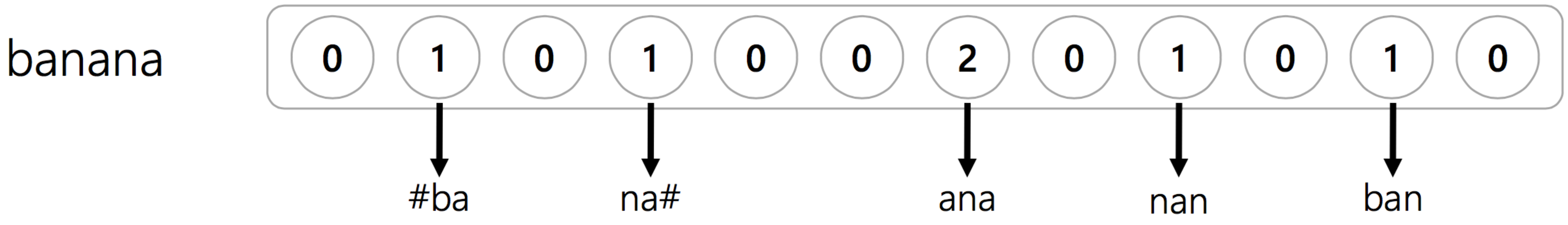
- The vector can correspond to neighboring word context.

"yellow banana grows on trees in africa"

-1 0 +1 +2 +3 +4 +5

Vector space models

- Represent an item (e.g., word) as a vector of numbers



- The vector can correspond to character trigrams in the word.

Notions of relatedness

- Comparing two vectors (e.g., using cosine similarity) estimates how similar the two words are. However, the notion of relatedness depends on what vector representation you have chosen for the words.

Seattle similar to denver?

Because they are both cities

OR

seattle similar to seahawks

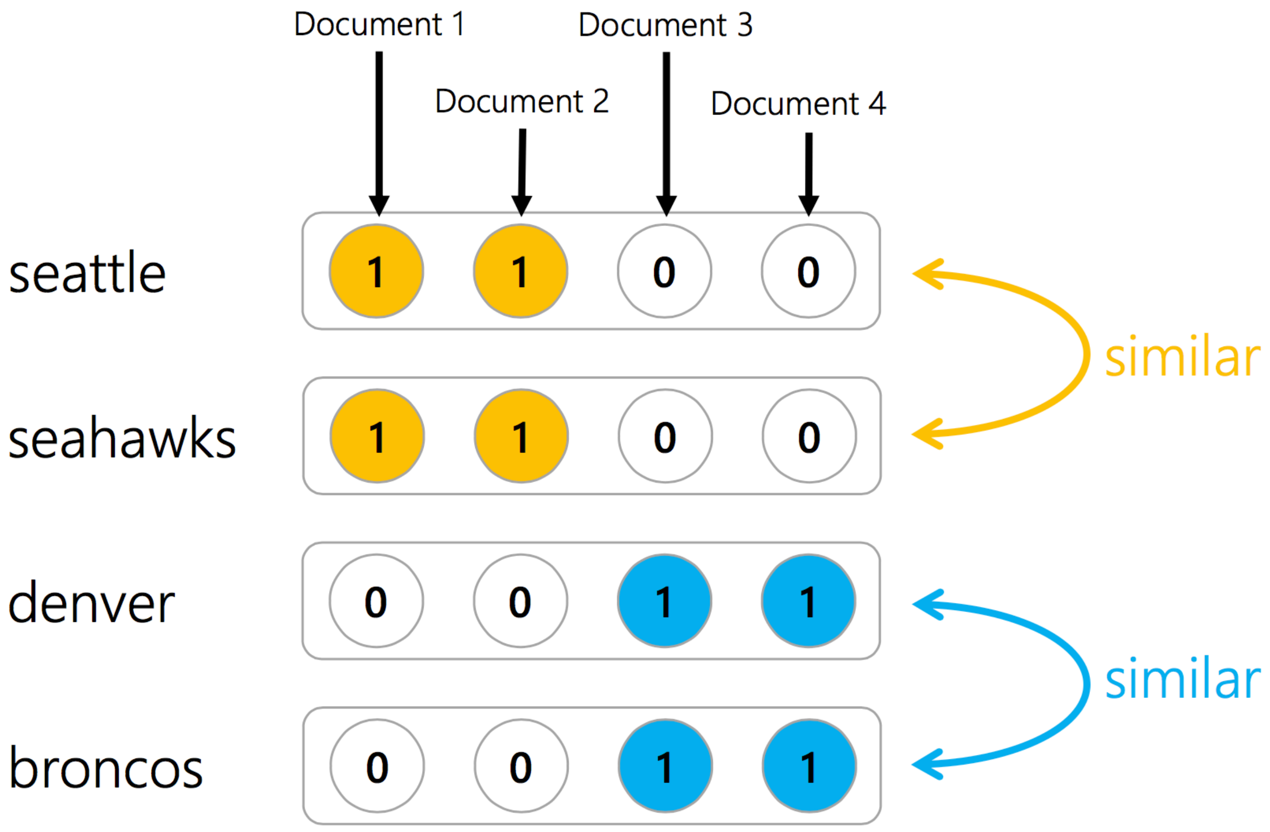
because “seattle seahawks”

Let's consider the following example…

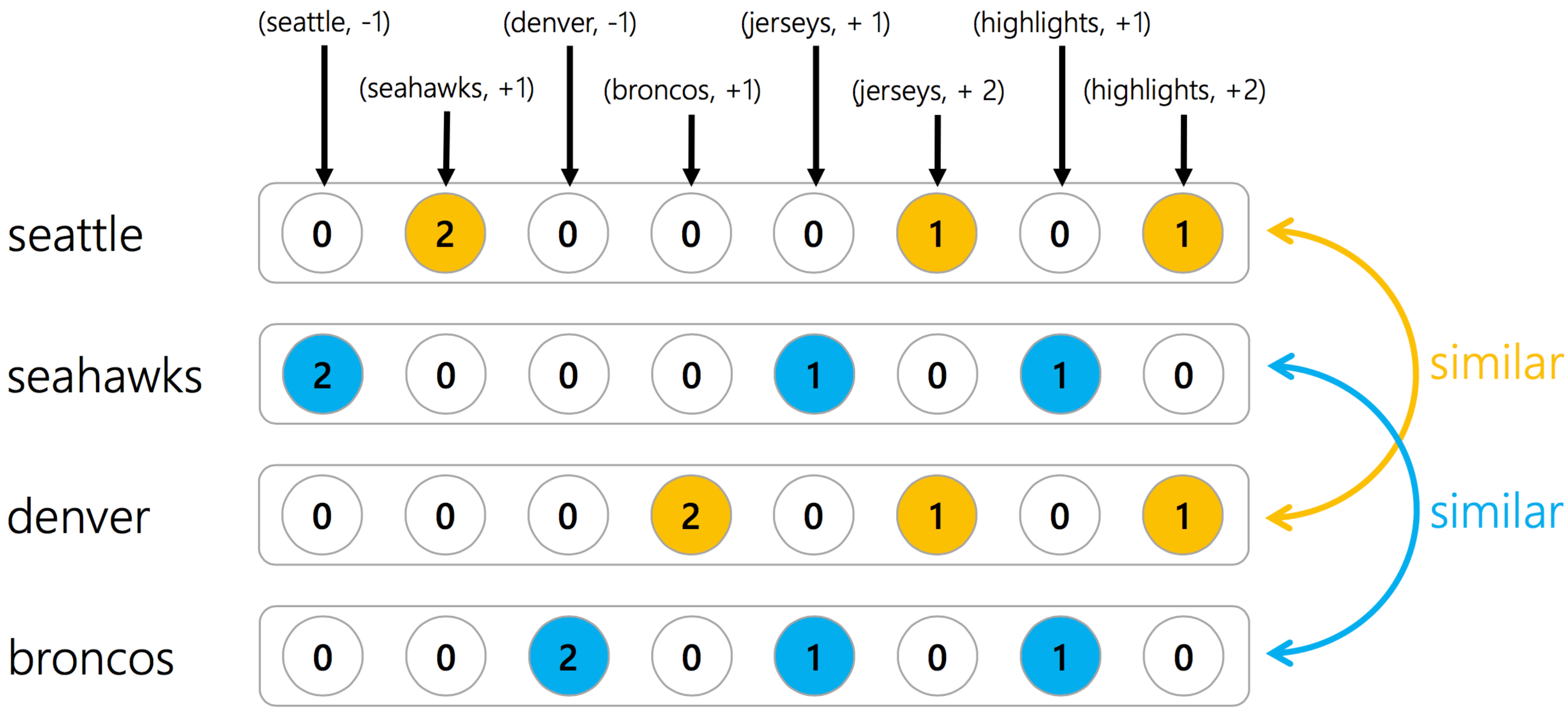
- We have four (tiny) documents,

Document 1 : "seattle seahawks jerseys"
Document 2 : "seattle seahawks highlights"
Document 3 : "denver broncos jerseys"
Document 4 : "denver broncos highlights"

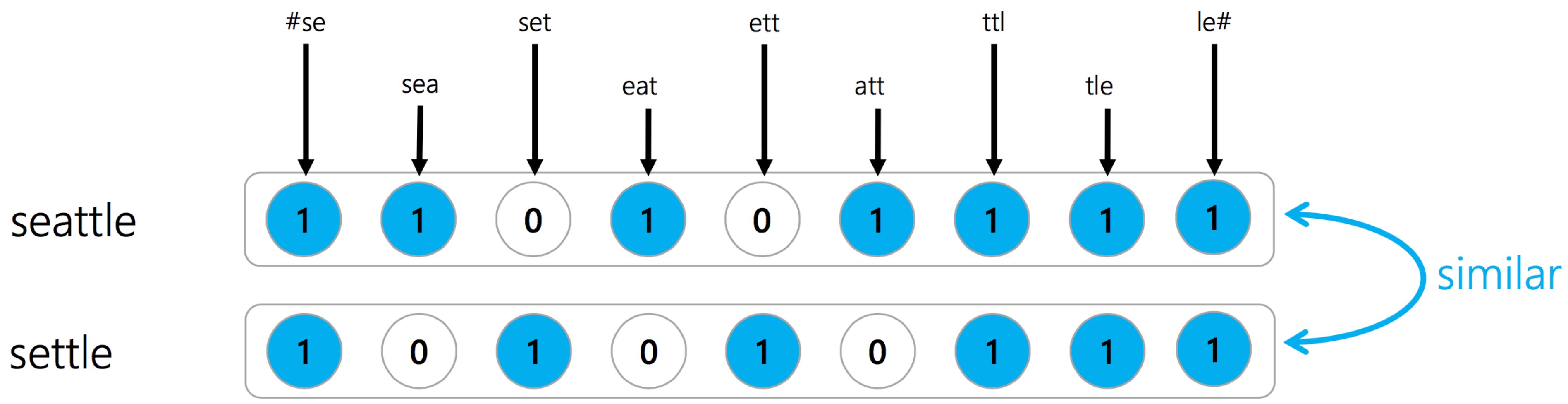
If we use document occurrence vectors



If we use word context vectors



If we use character trigram vectors



Word analogy task

- man is to woman as king is to ____?
- good is to best as smart is to ____?
- china is to beijing as russia is to ____?

- Turns out the word-context based vector model we just learnt is good for such analogy tasks,

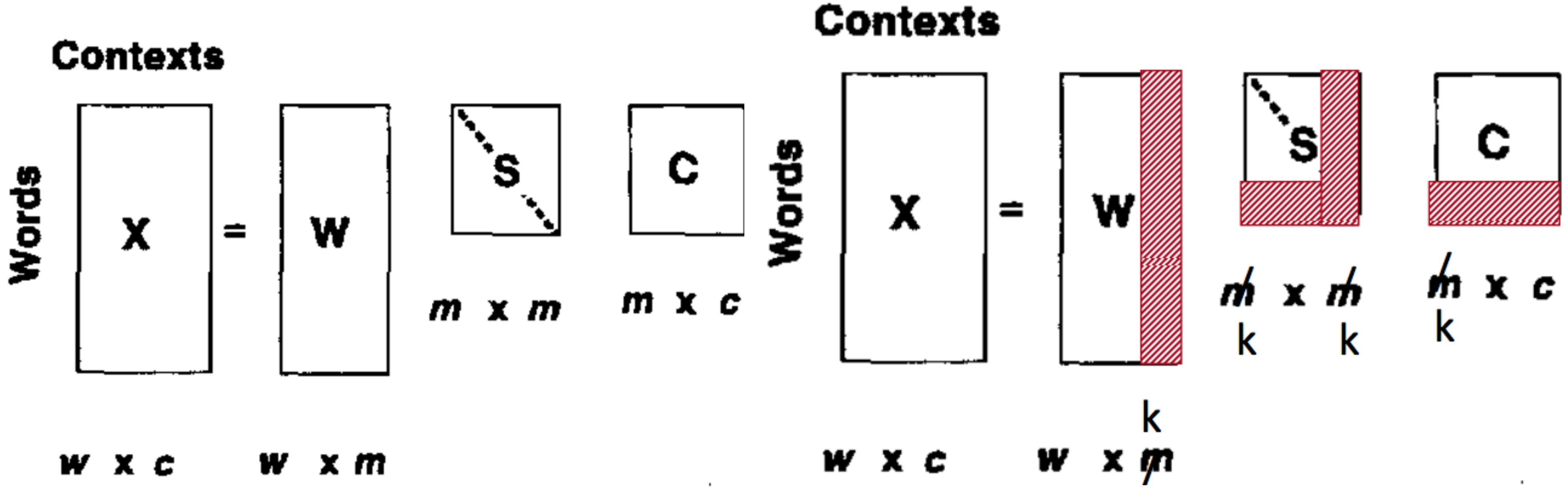
$$[\text{king}] - [\text{man}] + [\text{woman}] \approx [\text{queen}]$$

Potential approaches…

- Approach 1: Use existing thesauri or ontologies like WordNet and Snomed CT (for medical). Drawbacks:
  - Manual
  - Not context specific
- Approach 2: Use co-occurrences for word similarity. Drawbacks:
  - Quadratic space needed
  - Relative position and order of words not considered

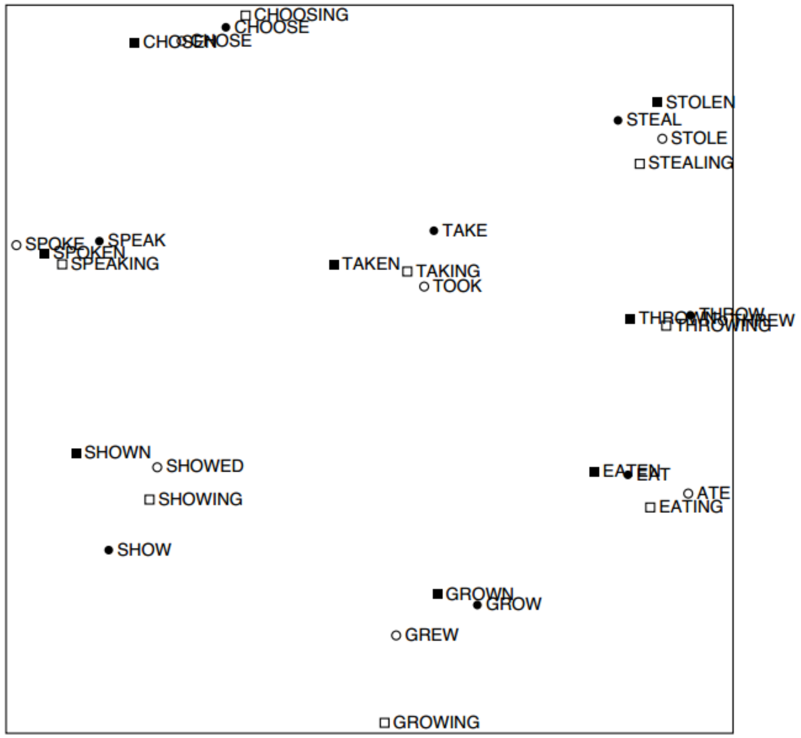
Approach 3: low dimensional vectors

- Store only "important" information in fixed, low dimensional vector.
- Single Value Decomposition (SVD) on co-occurrence matrix
  - is the best rank k approximation to X, in terms of least squares
  - Motel = [0.286, 0.792, -0.177, -0.107, 0.109, -0.542, 0.349, 0.271]



Approach 3: low dimensional vectors

- An Improved Model of Semantic Similarity Based on Lexical Co-Occurrence, Rohde et al. 2005

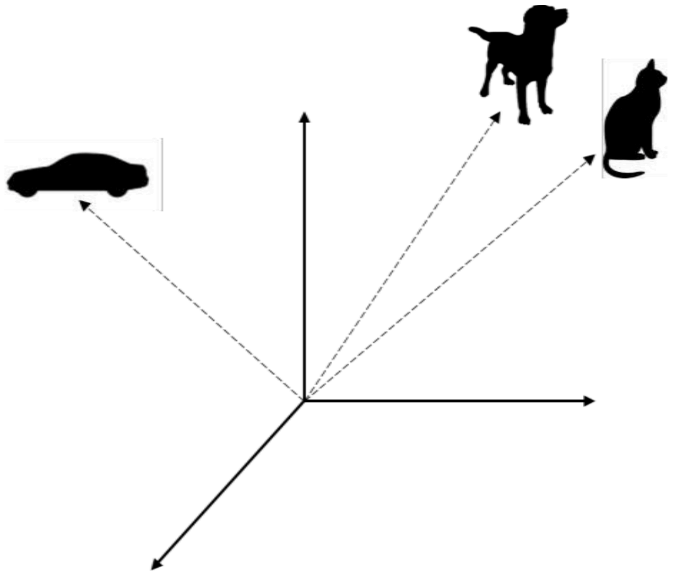


Problems with SVD

- Computational cost scales quadratically for n x m matrix: O(mn^2)
- Hard to incorporate new words or documents
- Does not consider order of words

Embedding

- The vectors we have been discussing so far are very **high- dimensional** (thousands, or even millions) and **sparse**.
- But there are techniques to learn lower-dimensional dense vectors for words using the same intuitions.
- These dense vectors are called embeddings.



Learning dense embeddings

Matrix Factorization

Factorize word-context matrix.

	Context$_1$	Context$_1$	....	Context$_k$
Word$_1$				
Word$_2$				
⋮				
Word$_n$				

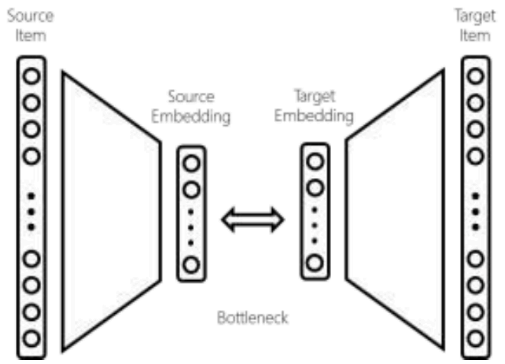
E.g.,

LDA (Word-Document),

GloVe (Word-NeighboringWord)

Neural Networks

A neural network with a bottleneck, word and context as input and output respectively.



E.g.,

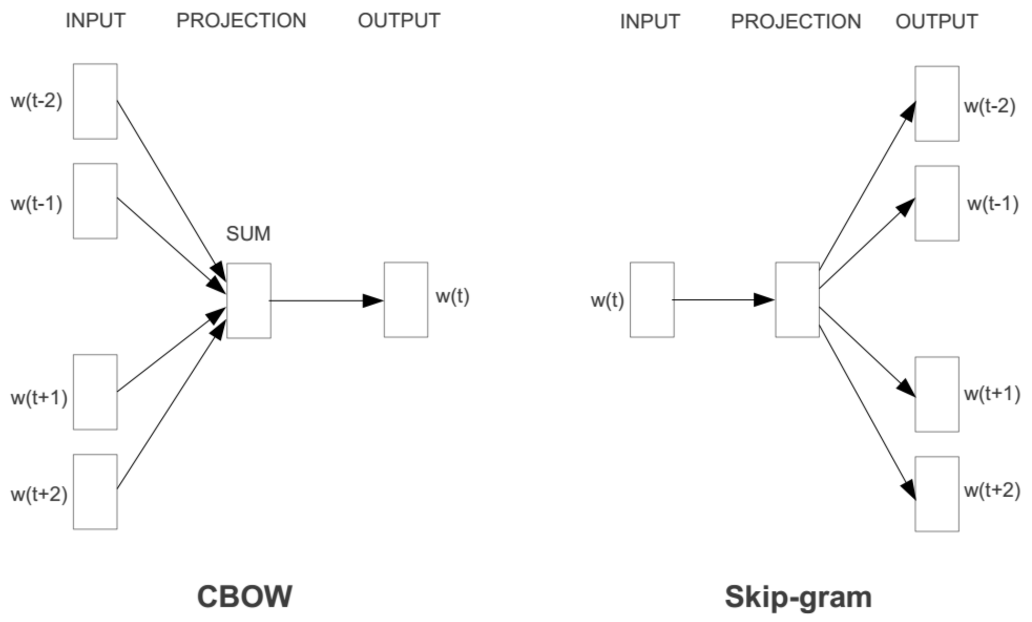
Word2vec (Word-NeighboringWord)

word2vec approach to represent the meaning of word

- Represent each word with a low-dimensional vector
- Word similarity = vector similarity
- Key idea: Predict surrounding words of every word
- Faster and can easily incorporate a new sentence/document or add a word to the vocabulary

Represent the meaning of word – word2vec

- 2 basic neural network models:
  - Continuous Bag of Word (CBOW): use a window of word to predict the middle word
  - **Skip-gram (SG)**: use a word to predict the surrounding ones in window.

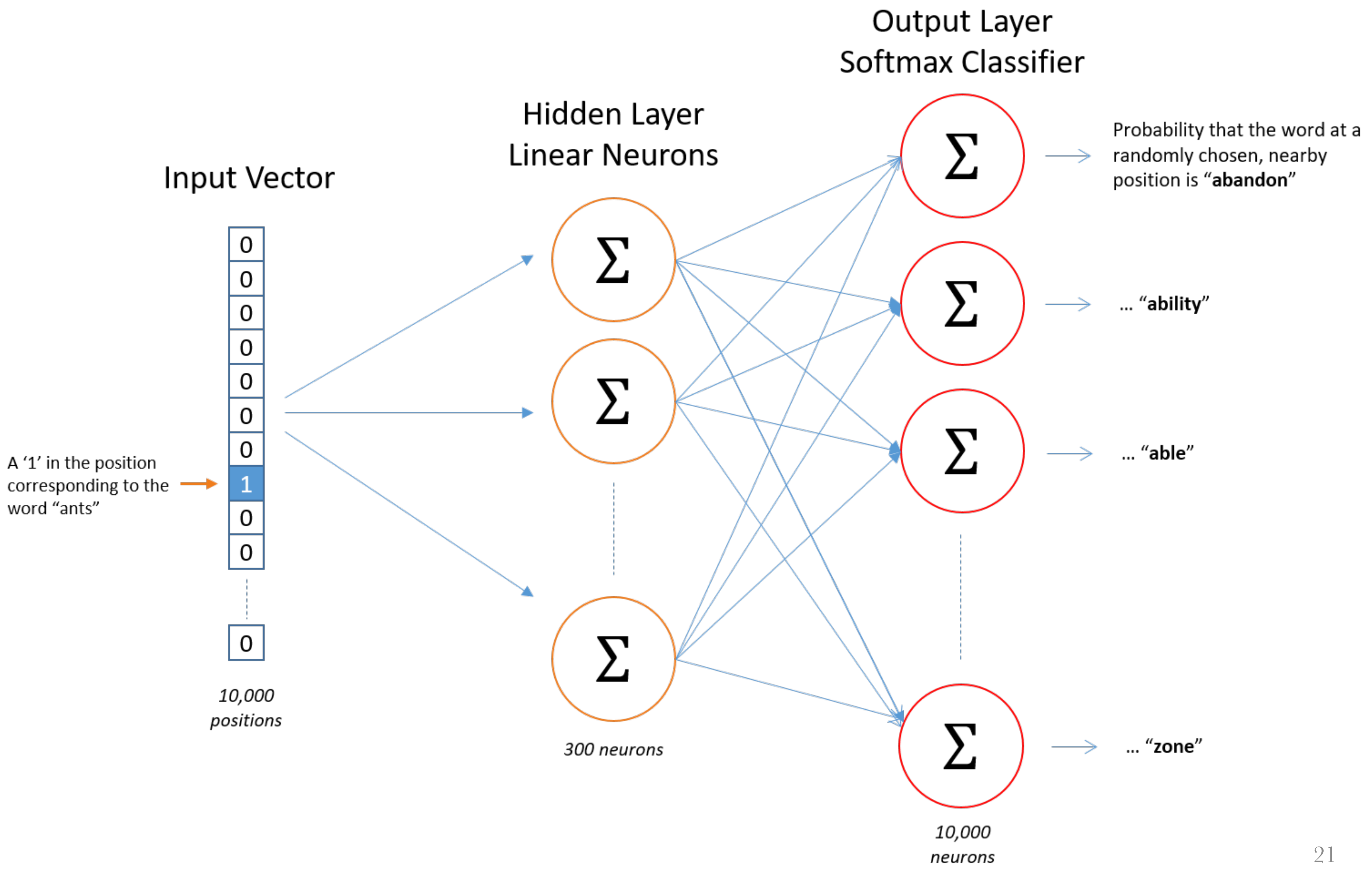


The model

- We're going to train a simple neural network with a **single hidden layer** to perform a certain task
  - We are not actually going to use that neural network for the task we trained it on!

- Instead, **the goal is actually just to learn the weights of the hidden layer**
  - These weights are actually the "word vectors"

Model details

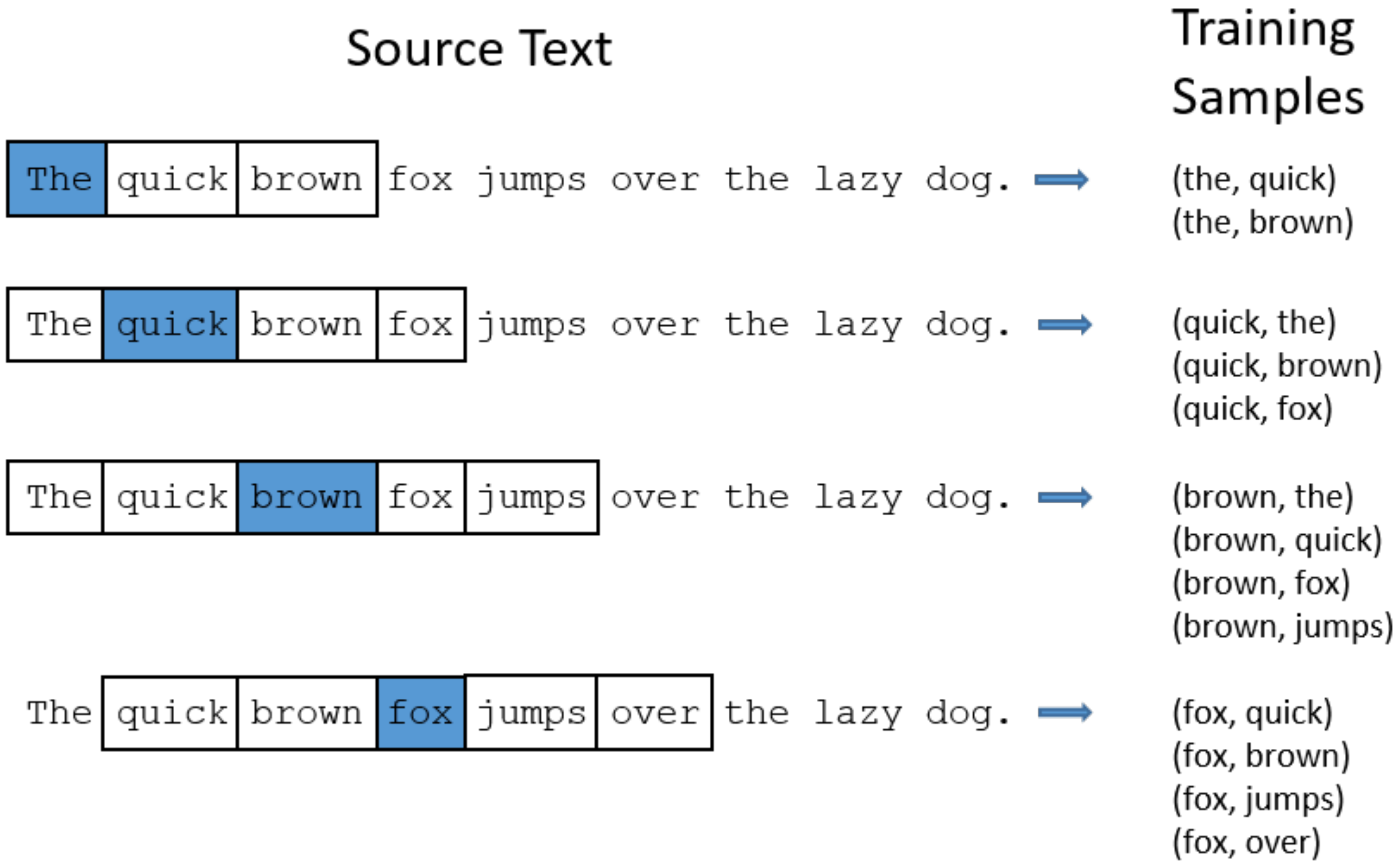


A fake task - intuition

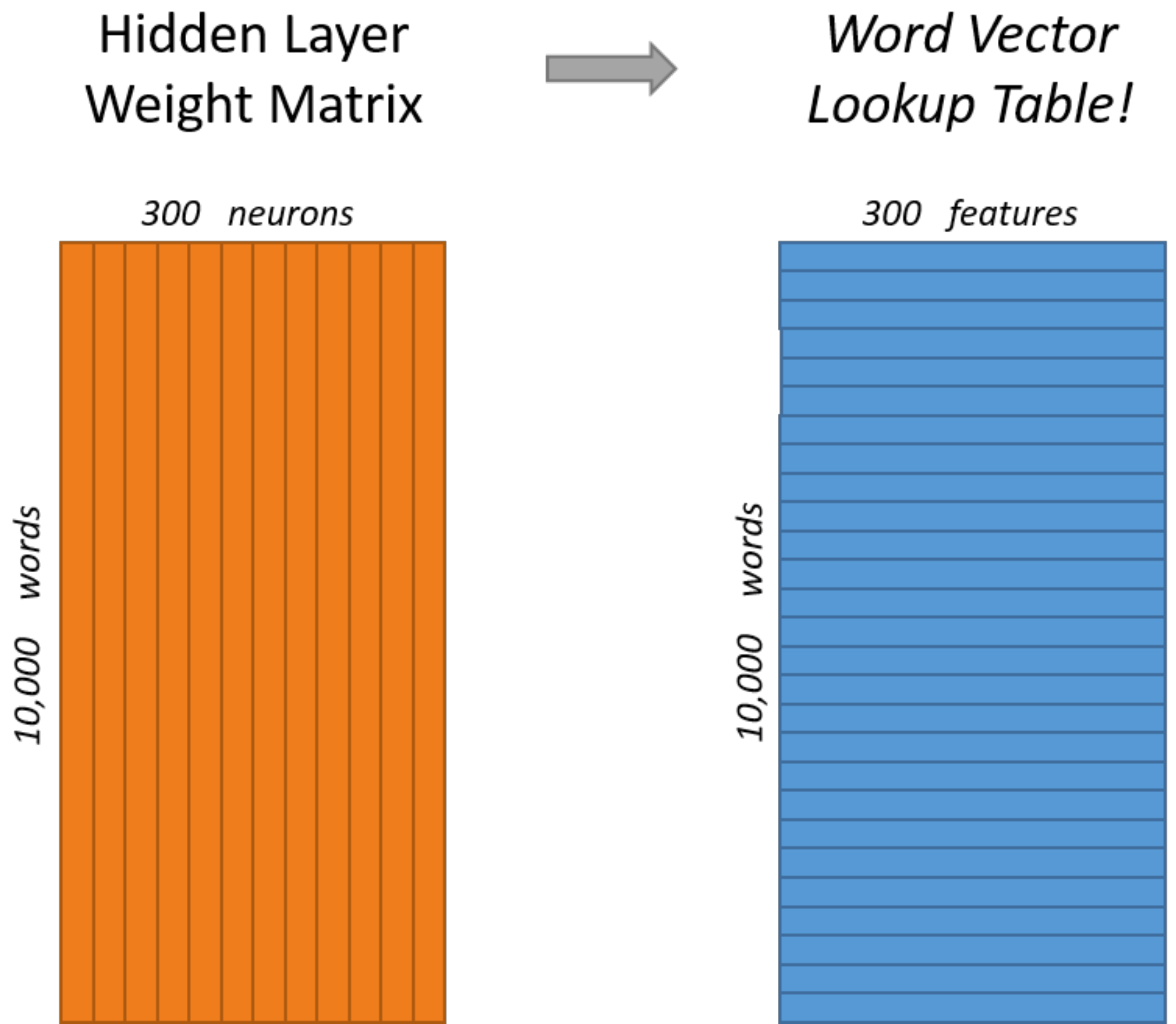
- Given a specific word in the middle of a sentence (the input word), look at the words nearby and pick one at random.
  - The network is going to tell us the probability for every word in our vocabulary of being the “nearby word” that we chose.
    - E.g., if you gave the trained network the input word “Soviet”, the output probabilities are going to be much higher for words like “Union” and “Russia” than for unrelated words like “watermelon” and “kangaroo”.
- "nearby" means "window size"
  - A typical window size might be 5, meaning 5 words behind and 5 words ahead (10 in total).

The fake task

- Training samples



The hidden layer



The hidden layer

- The hidden layer of this model is really just operating as a lookup table. The output of the hidden layer is just the "word vector" for the input word.

$$[0 \quad 0 \quad 0 \quad \boxed{1} \quad 0] \times \begin{bmatrix} 17 & 24 & 1 \\ 23 & 5 & 7 \\ 4 & 6 & 13 \\ \boxed{10 \quad 12 \quad 19} \\ 11 & 18 & 25 \end{bmatrix} = [10 \quad 12 \quad 19]$$

Efficiency issue

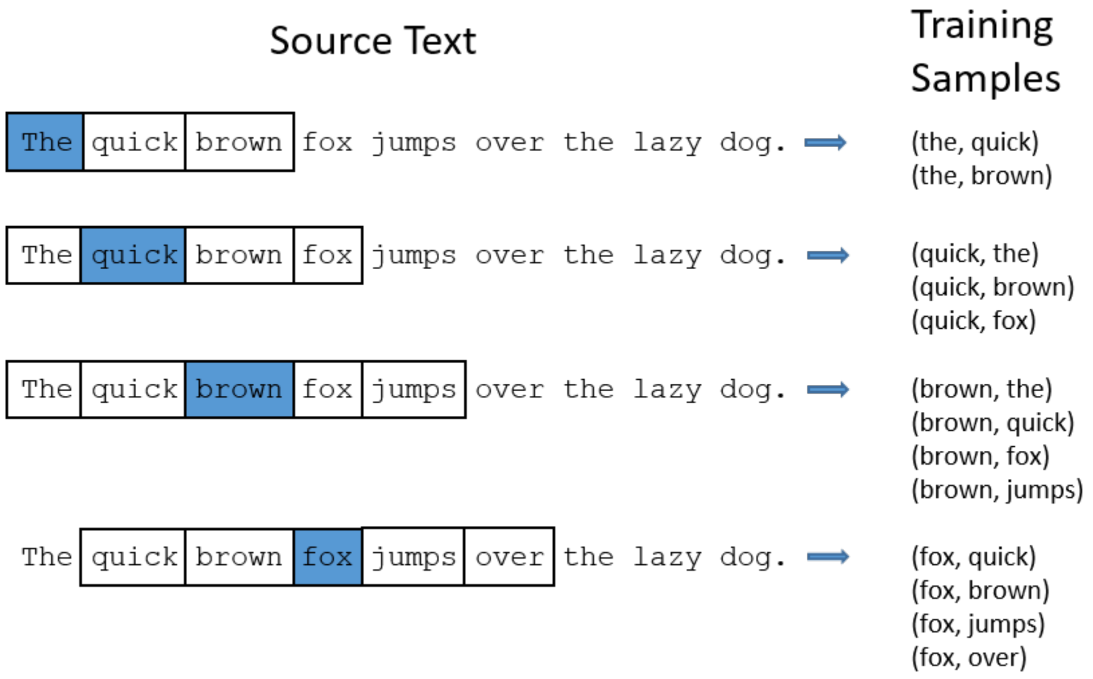
- Recall that the neural network had two weight matrices–a hidden layer and output layer. Both of these layers would have a weight matrix with 300 x 10,000 = 3 million weights each!

- Three innovations:
  1. Treating common word pairs or phrases as single "words" in their model.
  2. Subsampling frequent words to decrease the number of training examples.
  3. Modifying the optimization objective with a technique they called "Negative Sampling", which causes each training sample to update only a small percentage of the model's weights.

1. Word pairs and phrases

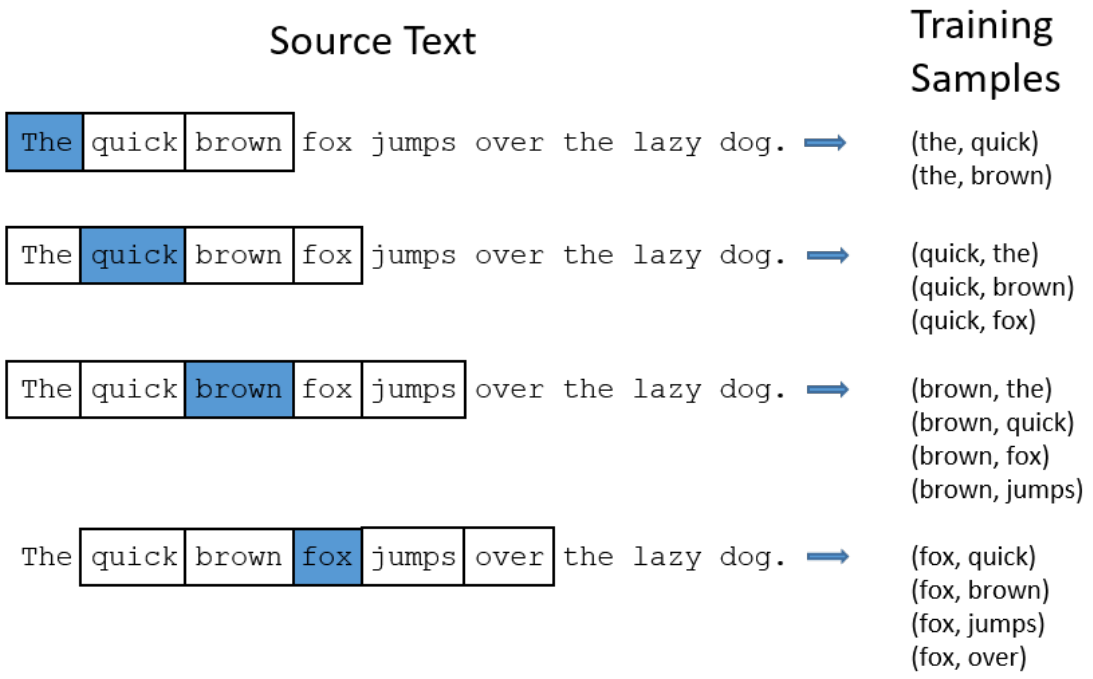
- The authors pointed out that a word pair like “Boston Globe” (a newspaper) has a much different meaning than the individual words “Boston” and “Globe”. So it makes sense to treat “Boston Globe”, wherever it occurs in the text, as a **single word** with its own word vector representation.

2. Subsampling frequent words



- There are two "problems" with common words like "the":
  - When looking at word pairs, ("fox", "the") doesn't tell us much about the meaning of "fox". "the" appears in the context of pretty much every word.
  - We will have many more samples of ("the", …) than we need to learn a good vector for "the".

2. Subsampling frequent words



- For each word we encounter in our training text, there is a chance that we will effectively delete it from the text.
- The probability that we cut the word is related to the word's frequency.

2. Subsampling frequent words

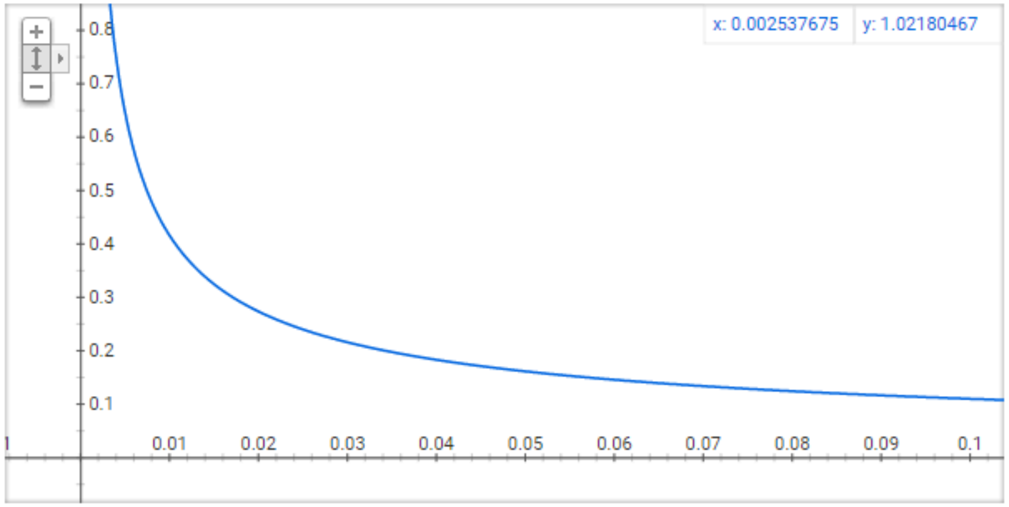
- If we have a window size of 10, and we remove a specific instance of “the” from our text:
  - As we train on the remaining words, “the” will not appear in any of their context windows.
  - We’ll have 10 fewer training samples where “the” is the input word.

Sampling rate

- w_i is the word, $z(w_i)$ is the fraction of the total words in the corpus that are that word.
- $P(w_i)$ is the probability of keeping the word:

$$P(w_i) = (\sqrt{\frac{z(w_i)}{0.001}} + 1) \cdot \frac{0.001}{z(w_i)}$$

Graph for (sqrt(x/0.001)+1)*0.001/x



Sampling rate

- $P(w_i) = 1.0$ (100% chance of being kept) when $z(w_i) <= 0.0026$.
    - This means that only words which represent more than 0.26% of the total words will be subsampled.

- $P(w_i) = 0.5$ (50% chance of being kept) when $z(w_i) = 0.00746$.
- $P(w_i) = 0.033$ (3.3% chance of being kept) when $z(w_i) = 1.0$.
    - That is, if the corpus consisted entirely of word w_i, which of course is ridiculous.

3. Negative sampling

- As we discussed above, the size of our word vocabulary means that our skip-gram neural network has a tremendous number of weights, all of which would be updated slightly by every one of our billions of training samples!

- Negative sampling addresses this by having each training sample only modify a small percentage of the weights, rather than all of them. Here's how it works.

3. Negative sampling

- Given a word pair (“fox”, “quick”), for the output neuron corresponding to “quick” to output a 1, and for all of the other thousands of output neurons to output a 0.
  - We randomly select just a small number of “negative” words to update the weights for
  - We will also still update the weights for our “positive” word (which is the word “quick” in our current example).

3. Negative sampling

- We will just be updating the weights for our positive word ("quick"), plus the weights for 5 other words that we want to output 0.
  - That's a total of 6 output neurons, and 1,800 weight values total.
  - That's only 0.06% of the 3M weights in the output layer!

Selecting negative samples

$$P(w_i) = \frac{f(w_i)^{3/4}}{\sum_{j=0}^{n} \left(f(w_j)^{3/4} \right)}$$

Some interesting results

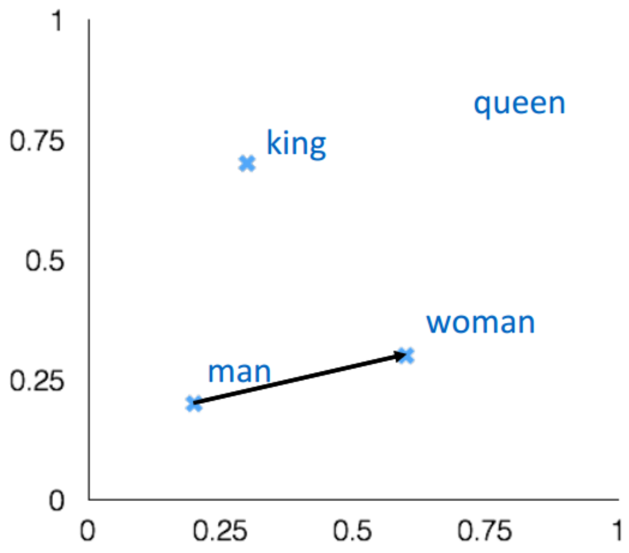
Word Analogies

Test for linear relationships, examined by Mikolov et al. (2014)

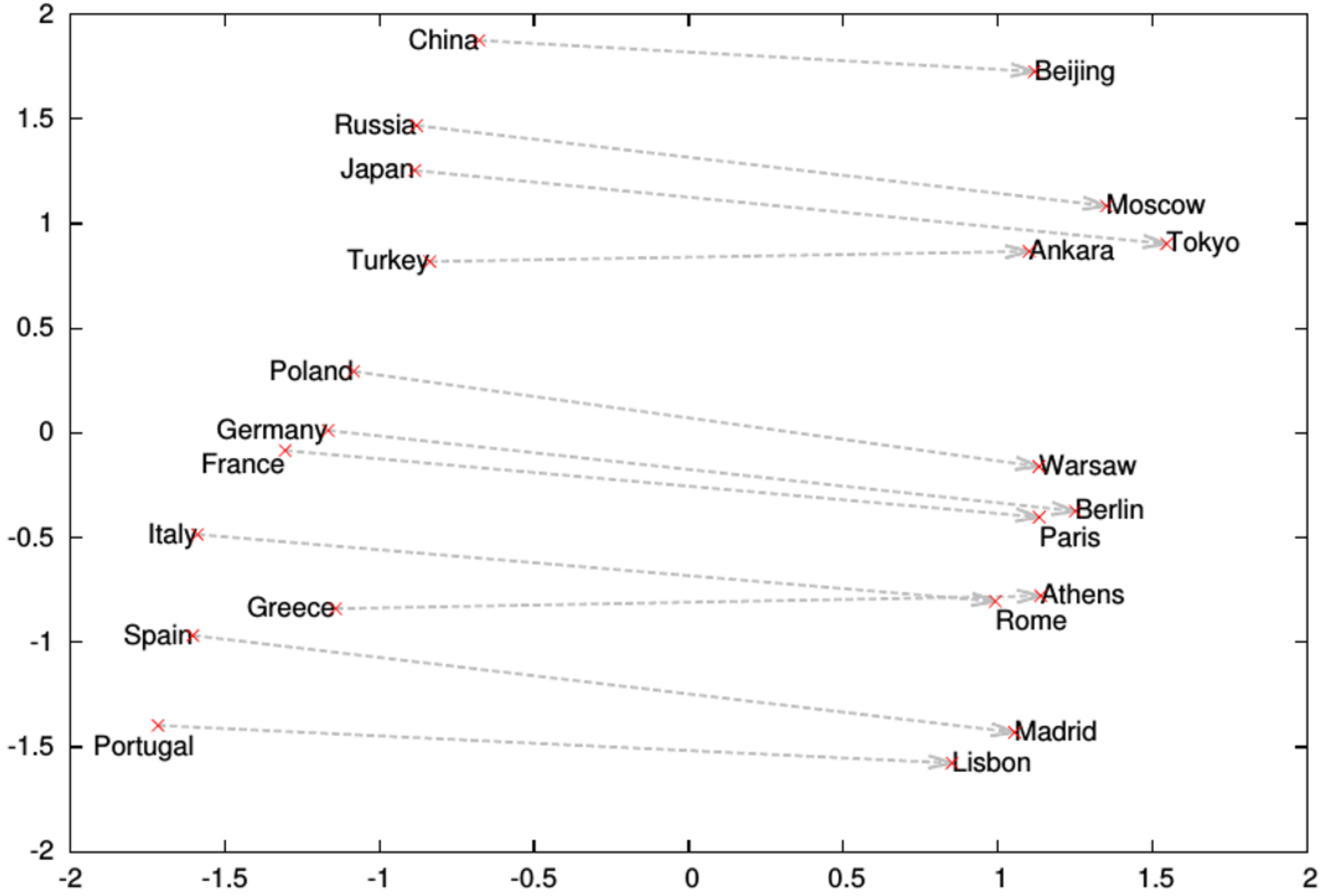
$\boxed{\text{a:b :: c:?}} \longrightarrow \boxed{d = \arg\max_x \frac{(w_b - w_a + w_c)^T w_x}{||w_b - w_a + w_c||}}$

man:woman :: king:?

+	king	[0.30 0.70]
-	man	[0.20 0.20]
+	woman	[0.60 0.30]
	queen	[0.70 0.80]



Word analogies



Resources

- Stanford CS224d: Deep Learning for NLP
  - http://cs224d.stanford.edu/index.html
  - The best
- "word2vec Parameter Learning Explained", Xin Rong
  - https://ronxin.github.io/wevi/
- Word2Vec Tutorial - The Skip-Gram Model
  - http://mccormickml.com/2016/04/19/word2vec-tutorial-the-skip-gram-model/