Text Mining 3 Representation Learning

Madrid Summer School on
Advanced Statistics and Data Mining

Florian Leitner
Data Catalytics, S.L.
leitner@datacatytics.com

Representation learning

- a **transformation of raw data** to a representation that can be effectively exploited in machine learning tasks
- **obviates feature engineering** (manually developing a representation to use for the classifier)
- many feature learning techniques do not required labeled data (i.e., are fully **unsupervised**)

Word representations

A trivial approach is to use a token's string itself as the representation; Numerically encode that leads us to a sparse, **one-hot** vector:

$$\text{"}\textit{tutorial}\text{"} := [0\ 0\ 0\ 0\ \ldots\ 0\ 0\ 0\ 0\ \mathbf{1}\ 0\ 0\ 0\ 0\ \ldots\ 0\ 0\ 0]$$

Problem: every such vector v is orthogonal to all others, so:

$$\mathbf{v}_1^{\mathrm{T}} \cdot \mathbf{v}_2 = \mathbf{0}$$

In other words, there is no notion of similarity between those vectors.

Therefore, the goal of word representations is to [numerically] **quantify the similarity of related words**.

From one-hot encoding to word embeddings

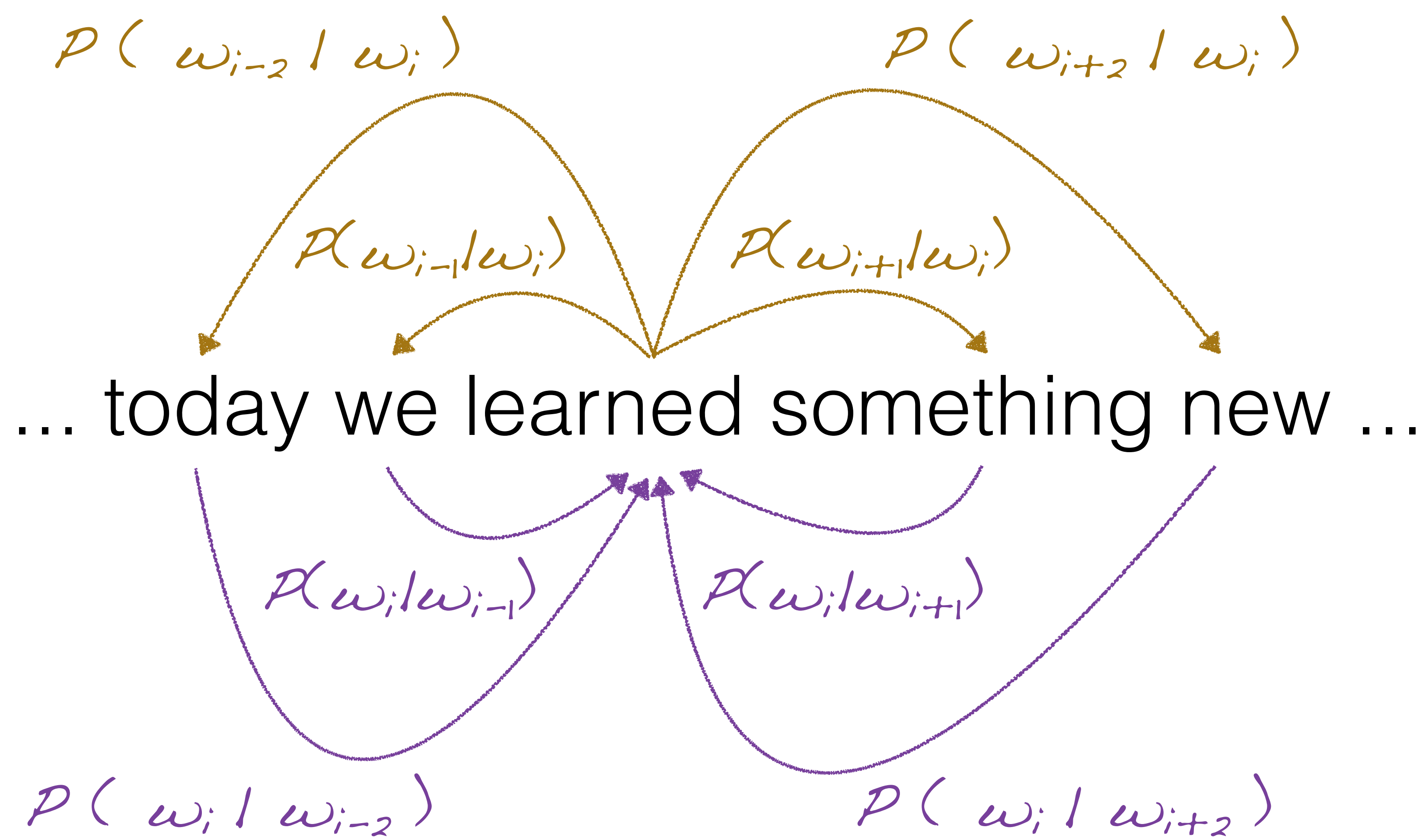
```
fun   = [1.0, 0.0, …, 0.0, 0.0, …, 0.0]
enjoy = [0.0, 0.0, …, 1.0, 0.0, …, 0.0]
like  = [0.0, 0.0, …, 0.0, 0.0, …, 1.0]
```

```
fun   = [0.6, 0.0, …, 0.3, 0.0, …, 0.1]
enjoy = [0.4, 0.0, …, 0.5, 0.0, …, 0.1]
like  = [0.2, 0.0, …, 0.2, 0.0, …, 0.6]
```

"You shall know a word by the company it keeps"

- J. R. Firth, **1957**:11
  (so the idea of word embeddings is definitely not new...)
- That is, the **context** (the surrounding words) of a word is dependent on the word itself; Put it slightly differently: a word "dictates" the possible words you can find in its surrounding.

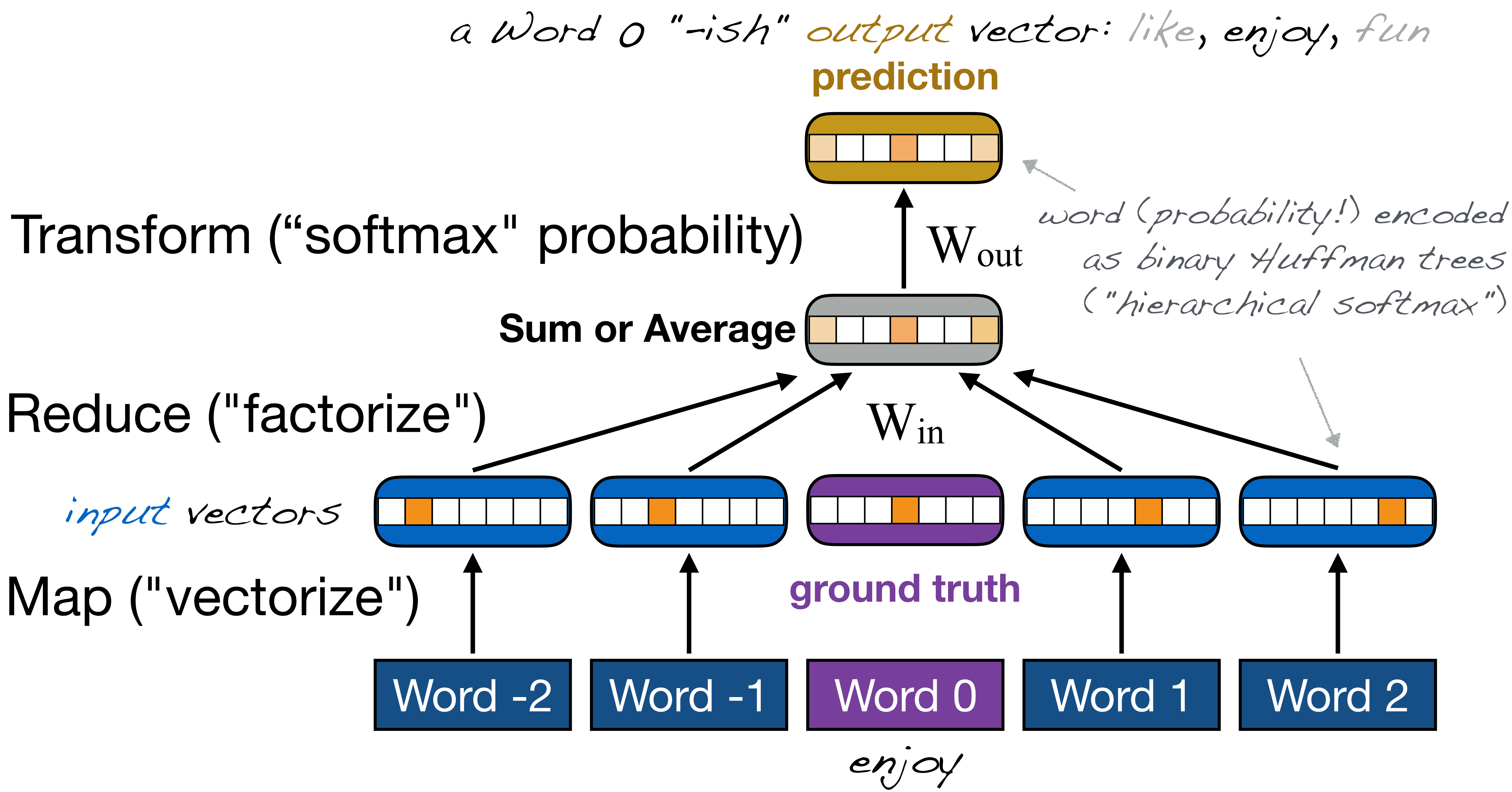
Predicting the surrounding of words *(and vice versa)*



(skip-gram)

(CBOW)

Word embeddings with neural networks (CBOW model)

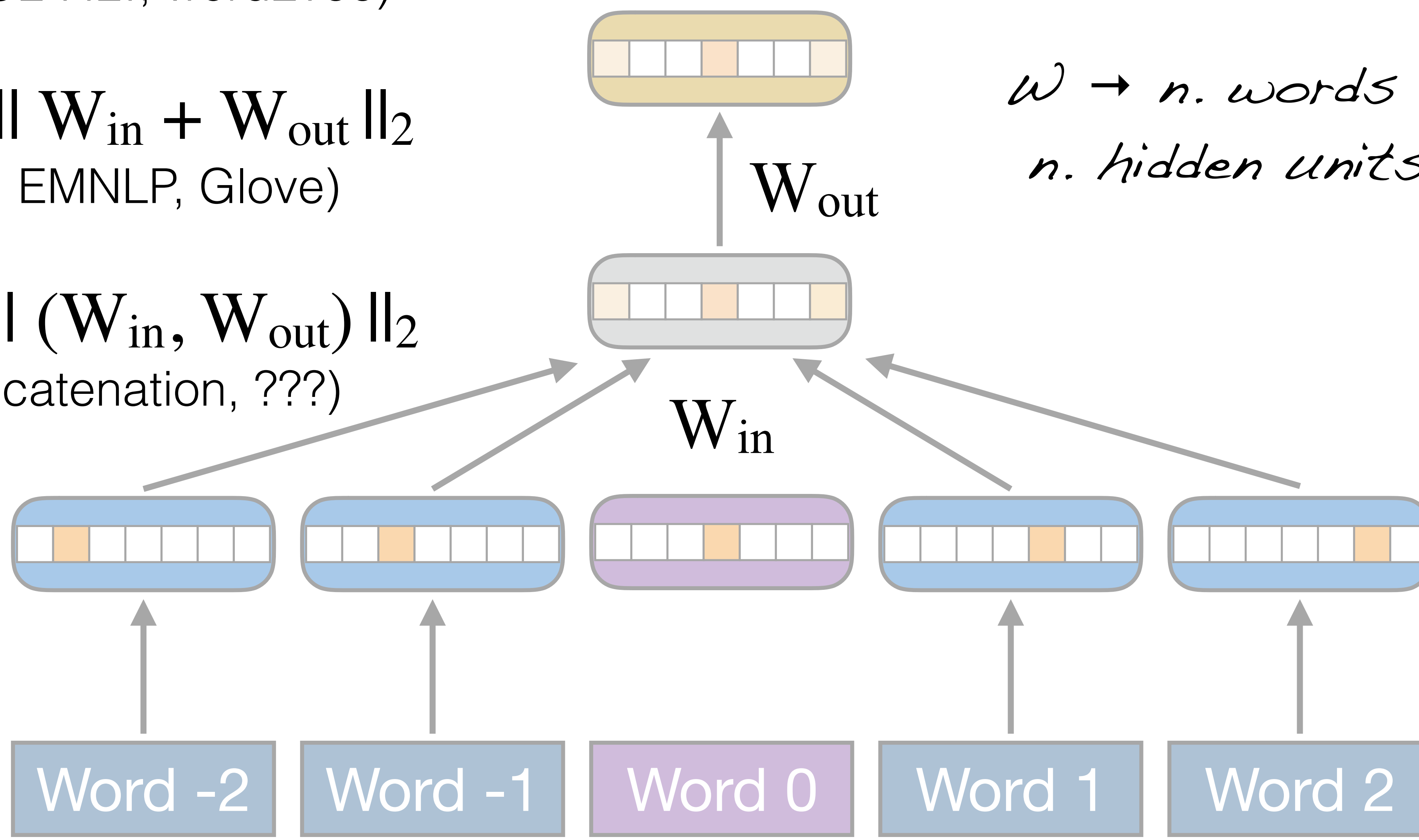


Where are the final word embeddings (vectors)?

Embeddings := $\| W_{in} \|_2$
(Mikolov 2013, NAACL-HLT, word2vec)

Embeddings := $\| W_{in} + W_{out} \|_2$
(Pennington 2014, EMNLP, Glove)

Embeddings := $\| (W_{in}, W_{out}) \|_2$
(in+out vector concatenation, ???)

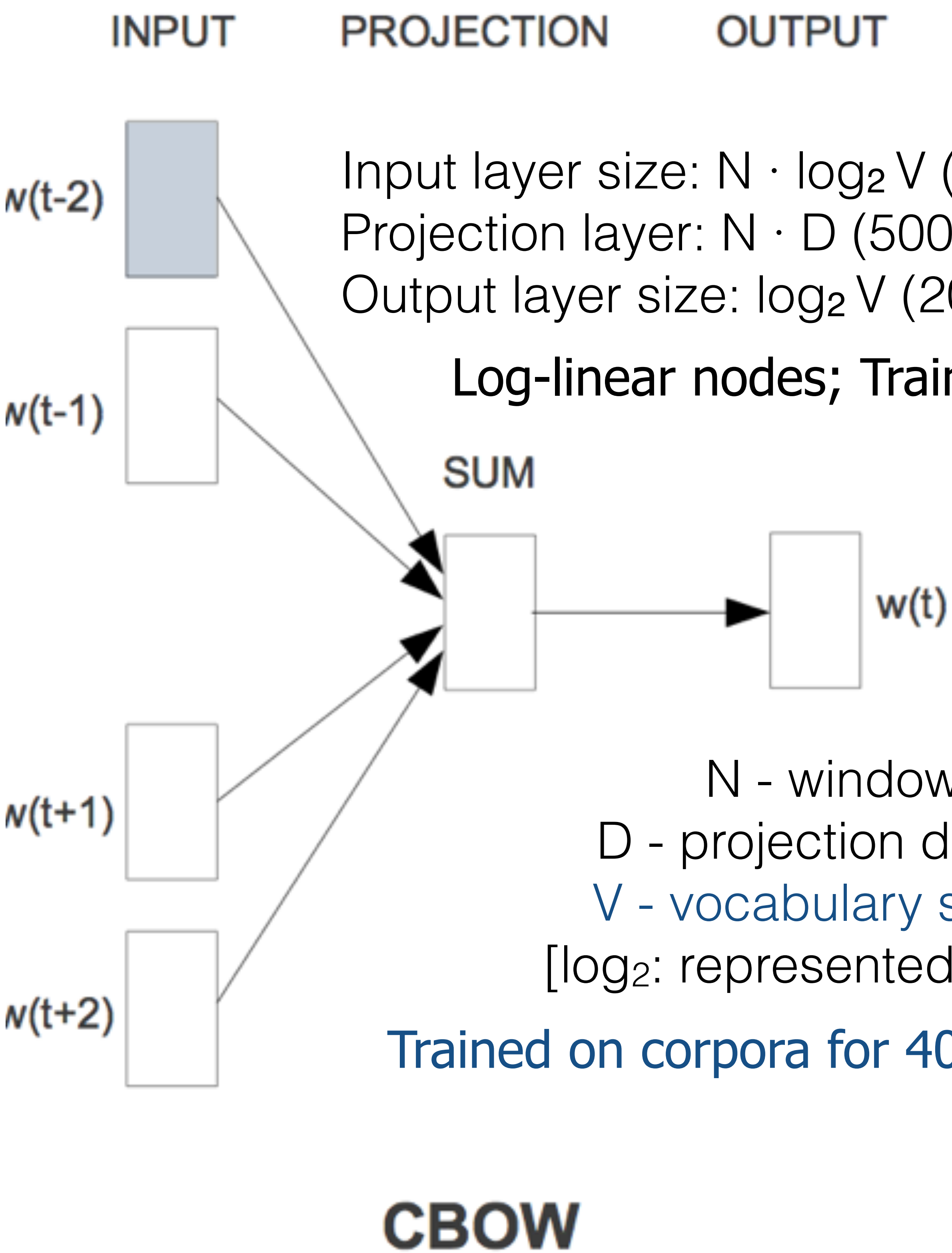


W → n. words × n. hidden units

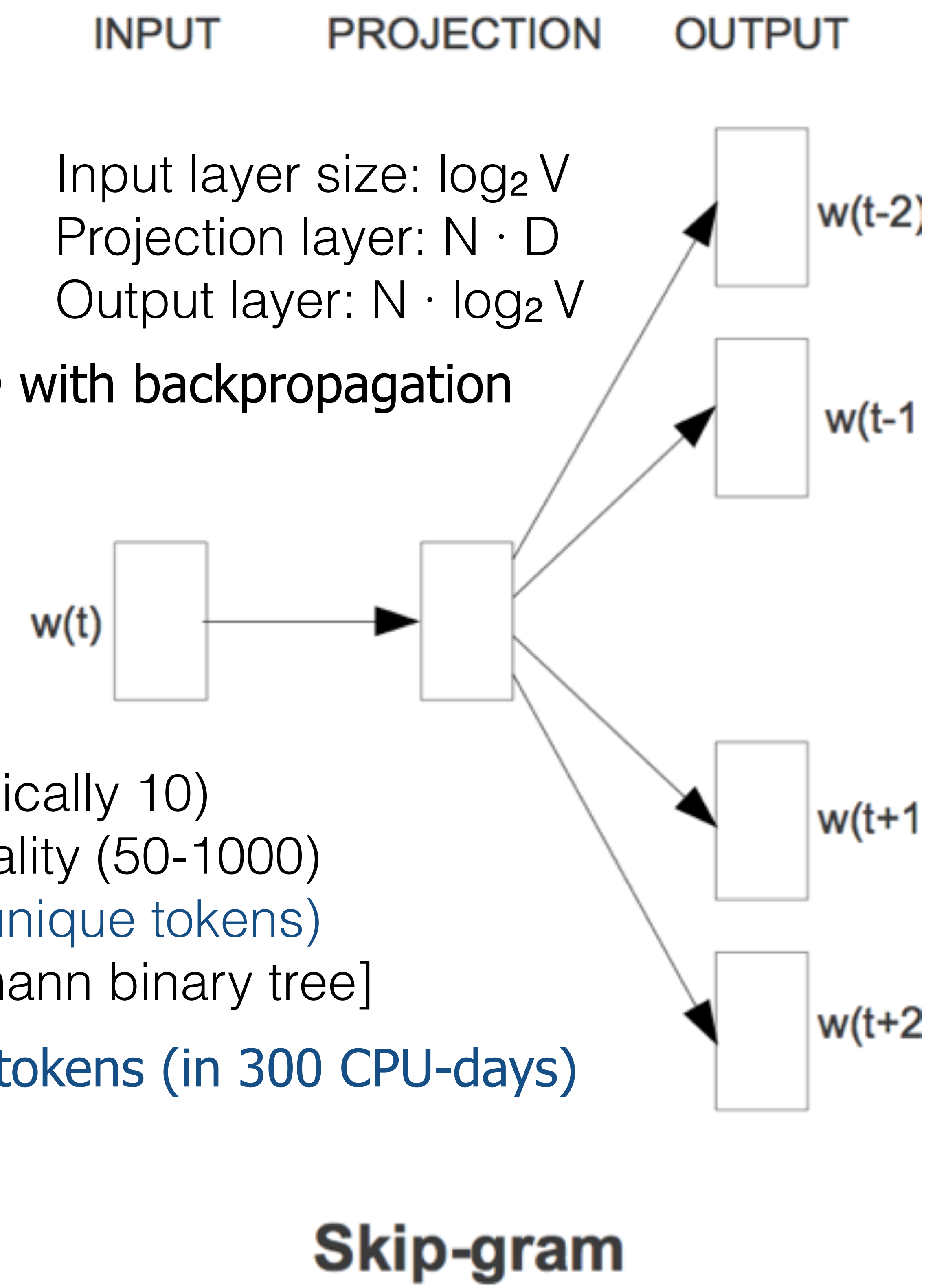
NB: softmax CBOW is easier to understand, but skip-gram with negative subsampling (SGNS) performs better

Neural network models of language

word2vec - Thomas Mikolov et al. - Google - 2013



predict a word from its surrounding



predict the surrounding of a word

Input layer size: $N \cdot \log_2 V$ (200)
Projection layer: $N \cdot D$ (500-20k)
Output layer size: $\log_2 V$ (20)

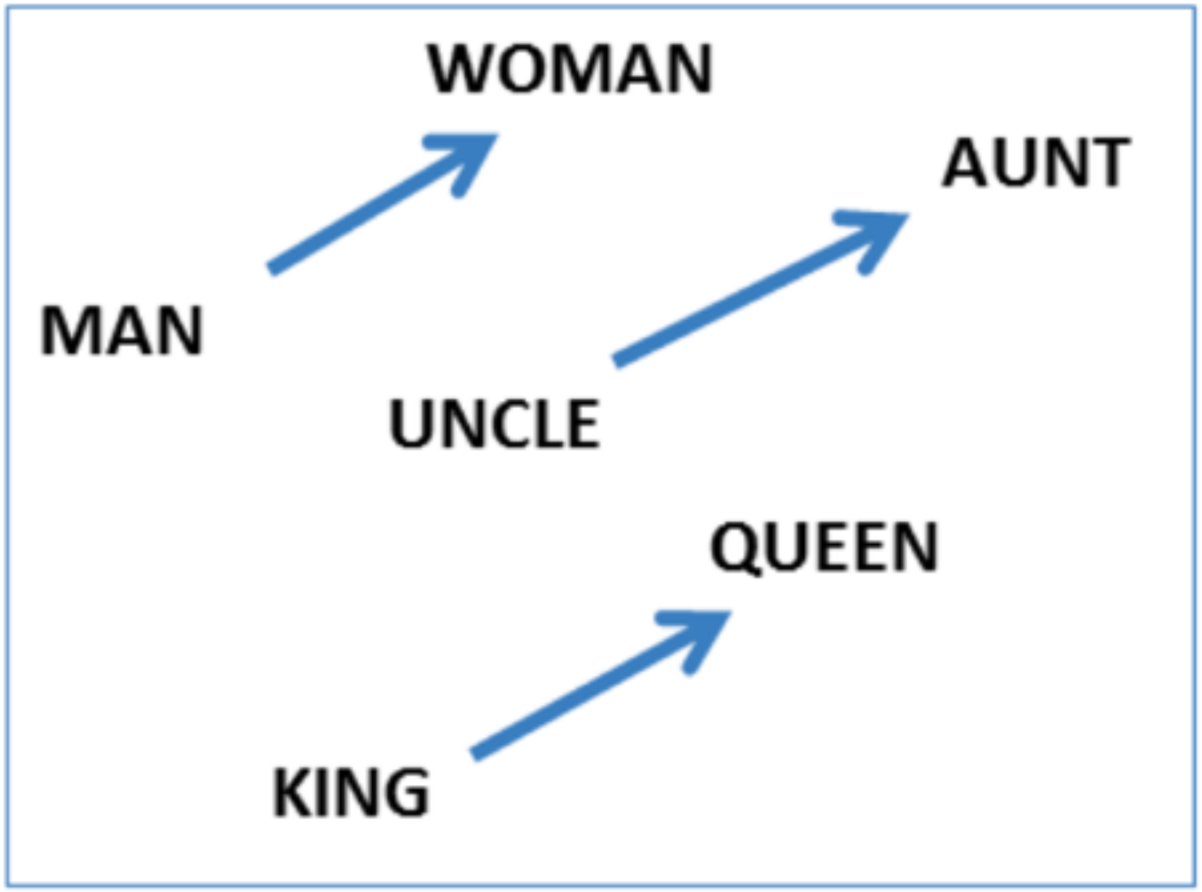
Input layer size: $\log_2 V$
Projection layer: $N \cdot D$
Output layer: $N \cdot \log_2 V$

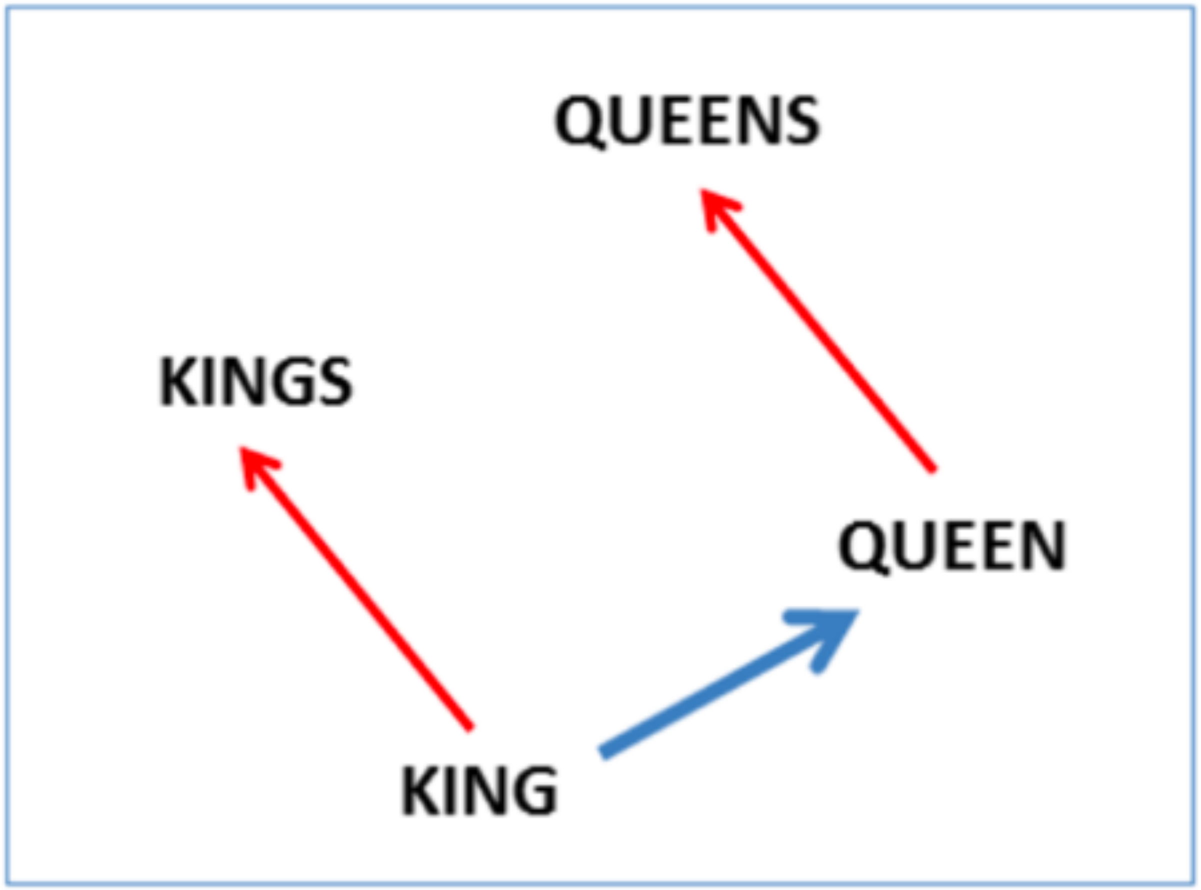
Log-linear nodes; Training: SGD with backpropagation

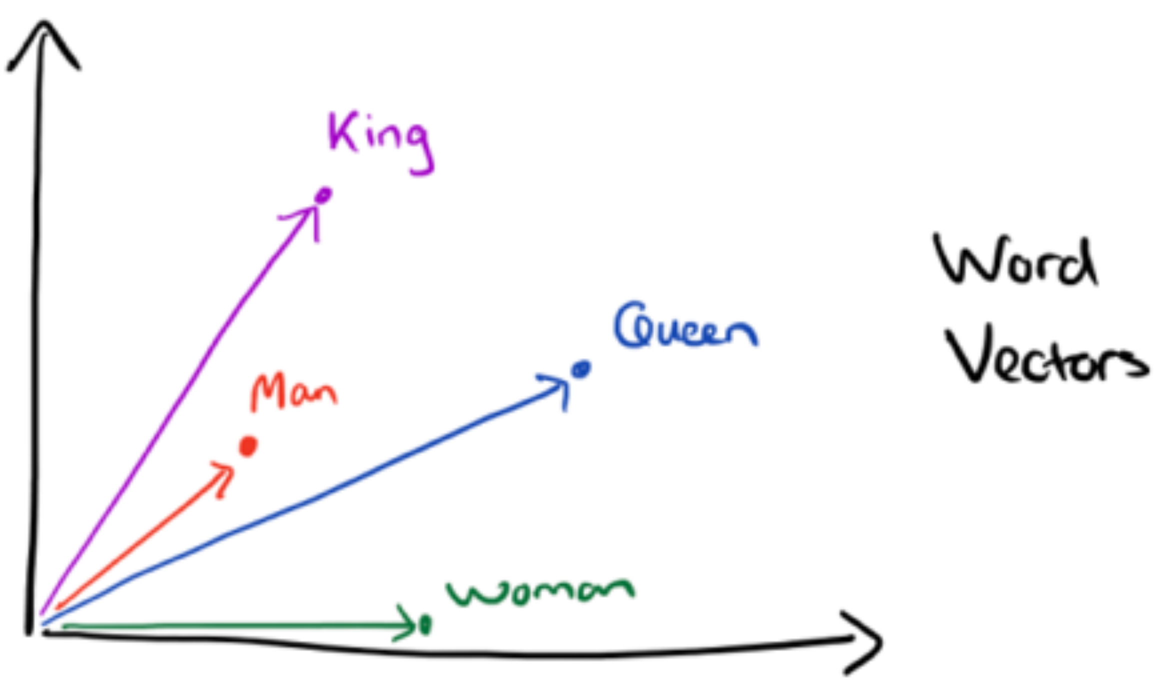
N - window size (typically 10)
D - projection dimensionality (50-1000)
V - vocabulary size (**1M** unique tokens)
[$\log_2$: represented as Huffmann binary tree]

Trained on corpora for 40M to **6B** tokens (in 300 CPU-days)

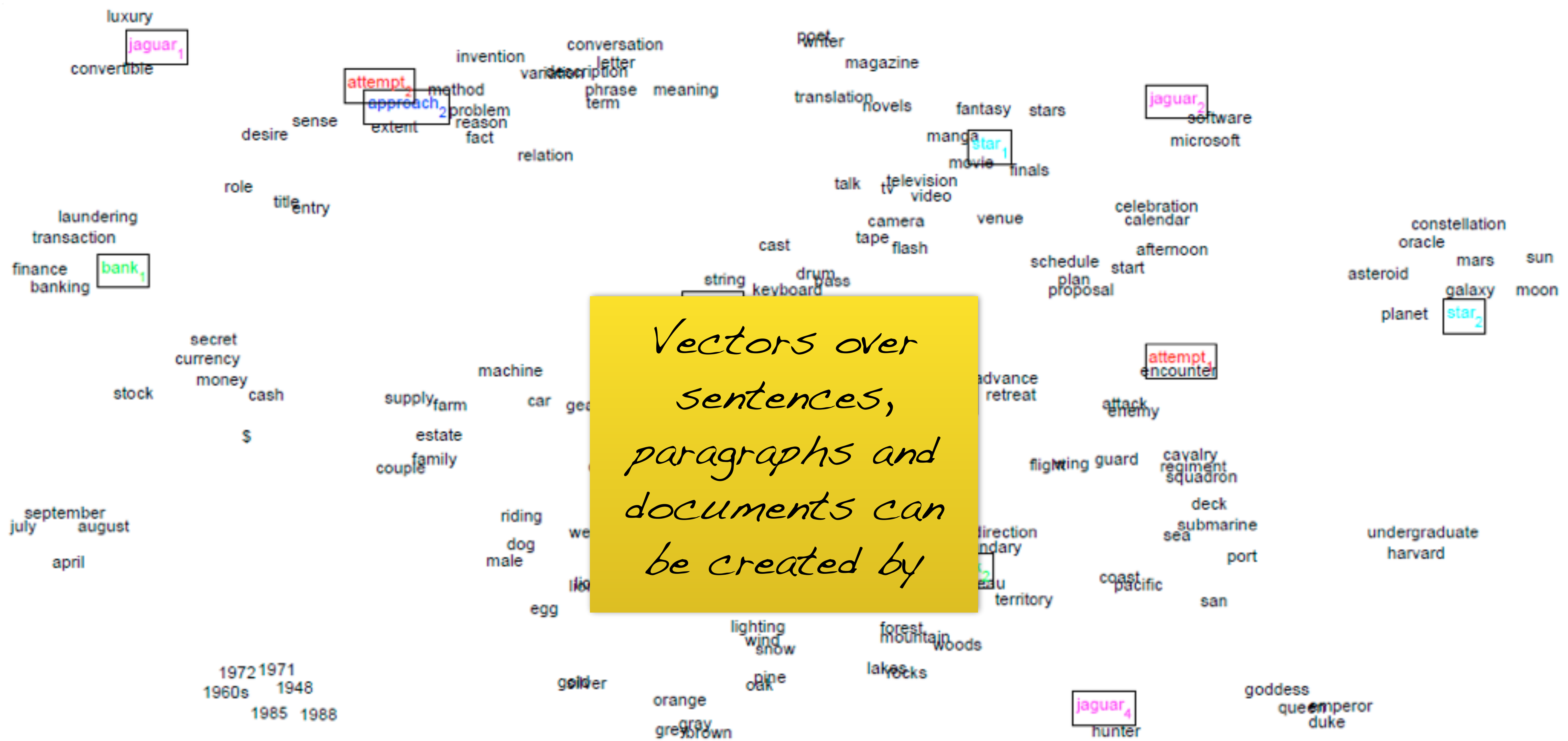
King - Man + Woman = ?







word2vec: co-occurrence probs.
GloVe: ratio of co-occ. probabilities



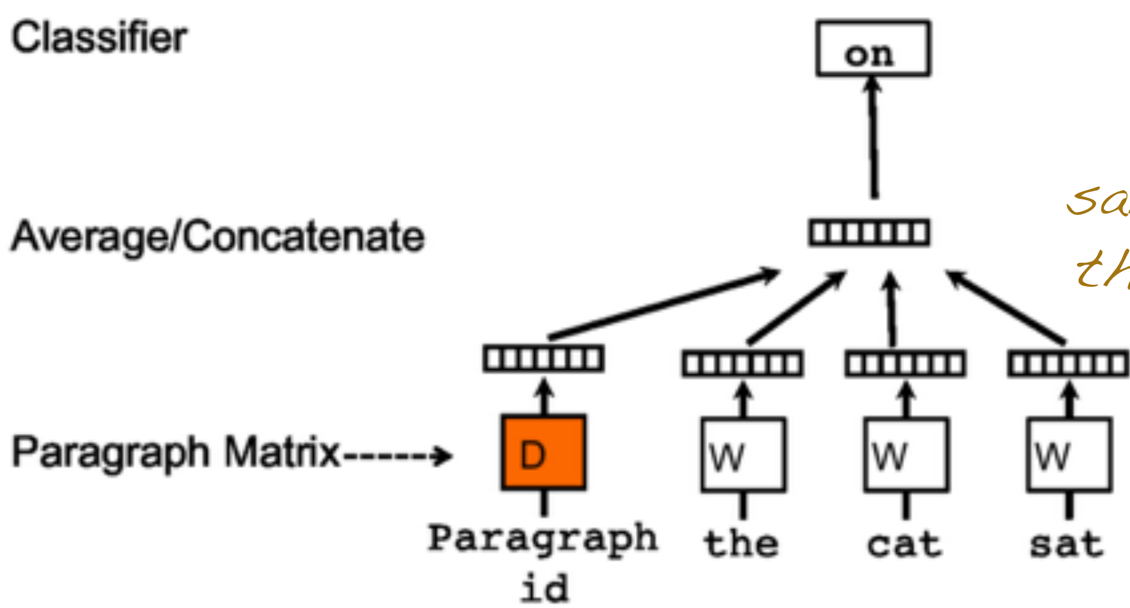
Vectors over sentences, paragraphs and documents can be created by

Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). Efficient Estimation of Word Representations in Vector Space. ICLR Workshop.

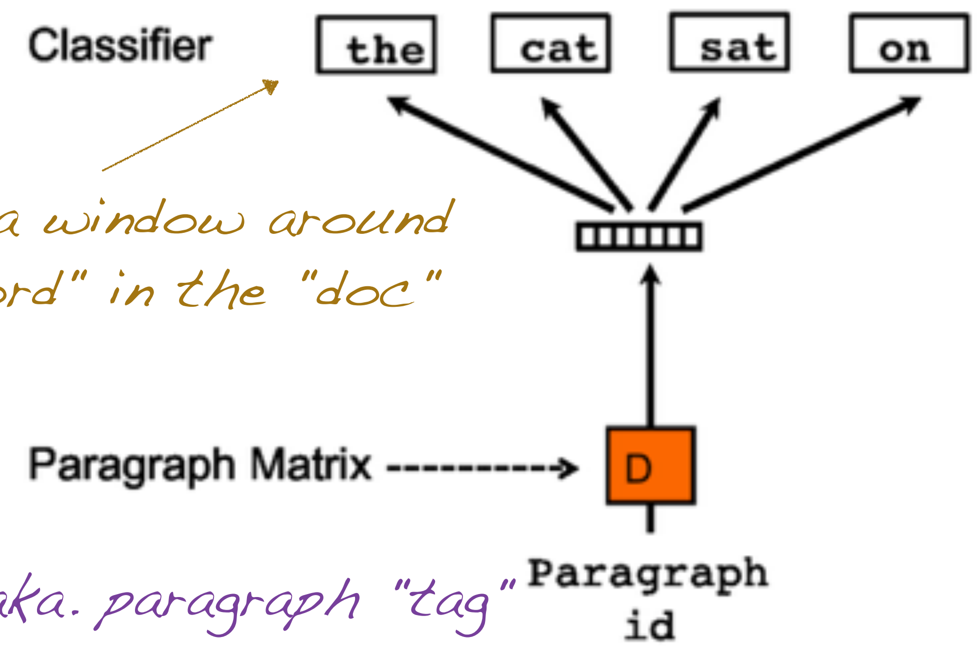
Pennington, J., Socher, R., & Manning, C. D. (2014). Glove: Global vectors for word representation. In Proceedings of the Empiricial Methods in Natural Language Processing (pp. 1532–1543).

Text embeddings with Paragraph Vectors (doc2vec)

a "doc" is some piece of text: a sentence, a tweet, a paragraph, or even a whole document



sampled from a window around the "target word" in the "doc"



aka. paragraph "tag"

PV-Distributed Memory (DM)
Predict the target word from the paragraph vector and the doc's words (from a window over the doc centered at the target word).

Distributed BOW-PV
Predict the doc's words (from some window over the doc) from the paragraph vector.

Le's & Mikolov's recommendation: train both models (using concatenation) and combine them.

Le, Q., and Mikolov, T. (2014). Distributed Representations of Sentences and Documents. 2014

Paragraph Vectors (doc2vec)

- Base idea is the same as word embeddings
  - c.f. CBOW/SGNS models
- But the **paragraph vector** D **needs to be inferred** when using this model ("in production")
  - i.e., you **predict** the embedding
  - c.f. **looking up** the embedding vector for words
- D is a **tag** for each doc
  - used as memory for that doc during training
  - typically just a unique integer per doc

Out-of-vocabulary (OOV) words: character n-grams

Problem: no embedding for words not seen during training

Solution: instead learn the embeddings of a word's n-grams

split each word into its **character** n-grams (typically, $n = [3, 6]$; and just use the word "as is" for tokens with character lengths < 4)

learn to embed the n-grams, with the target embedding being the average over the predicted n-gram embeddings

fastText: Joulin et al., 2016, arXiv (Facebook)

Cheap Solution: bucket all words into a fixed-size hash-table (smaller than the actual vocabulary) and allow for collisions (also known as the "**hashing trick**")

Statistical models of language and polysemy

- Polysemous words have multiple meanings (e.g., “bank”).
  - This is a real problem in scientific texts because polysemy is frequent.
- One idea: Create **context vectors** for each sense of a word (vector).
  - MSSG - Neelakantan et al. - 2015
- Caveat: Performance isn’t much better than for the skip-gram model by Mikolov et al., while training is ~5x slower.
- Simpler approach (partial solution only): use collocations
  - Either train the embeddings over the merged collocations (tomorrow's lesson), or [also] use bigrams as your embedding inputs (vs. of the [unigram] tokens)

Word embeddings: Applications in TM & NLP

- Opinion mining (Maas et al., 2011)
- Paraphrase detection (Socher et al., 2011)
- Chunking (Turian et al., 2010; Dhillon and Ungar, 2011)
- Named entity recognition (Neelakantan and Collins, 2014; Passos et al., 2014; Turian et al., 2010)
- Dependency parsing (Bansal et al., 2014)