Word embeddings

Marco Kuhlmann
Department of Computer and Information Science

LINKÖPING UNIVERSITY

The distributional principle

- The **distributional principle** states that words that occur in similar contexts tend to have similar meanings.
- ‘You shall know a word by the company it keeps.’

  Firth (1957)

Words and contexts

What do the following sentences tell us about *tesgüino*?

- A bottle of *tesgüino* is on the table.
- Everybody likes *tesgüino*.
- *Tesgüino* makes you drunk.
- We make *tesgüino* out of corn.

Examples from Lin (1998)

Word embeddings

- A **word embedding** is a mapping of words to points in a vector space such that nearby words (points) are similar in terms of their distributional properties.

- This idea is similar to the vector space model of information retrieval, where the dimensions of the vector space correspond to the terms that occur in a document.

  points = documents, nearby points = similar topic

Lin et al. (2015)

Term–document matrix

	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	5	117	0	0

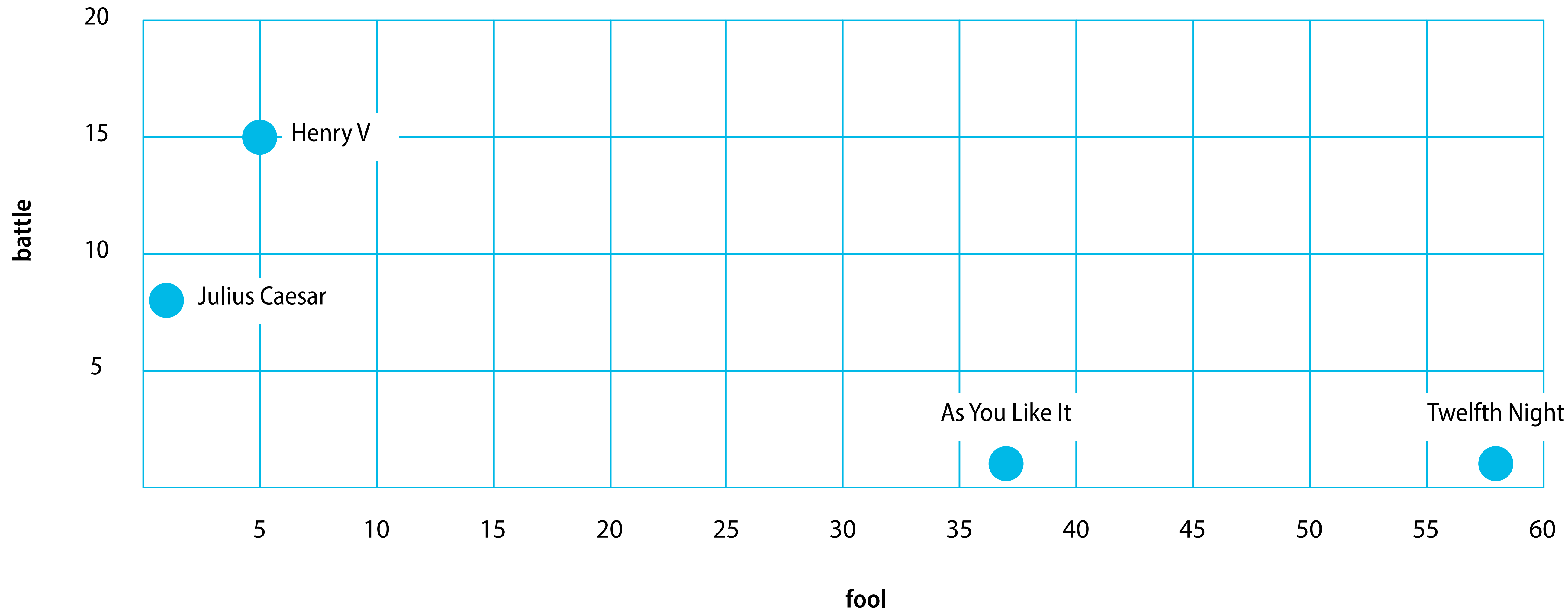
Jurafsky and Martin (2017)

Term–document matrix

	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	5	117	0	0

Jurafsky and Martin (2017)

Document embedding



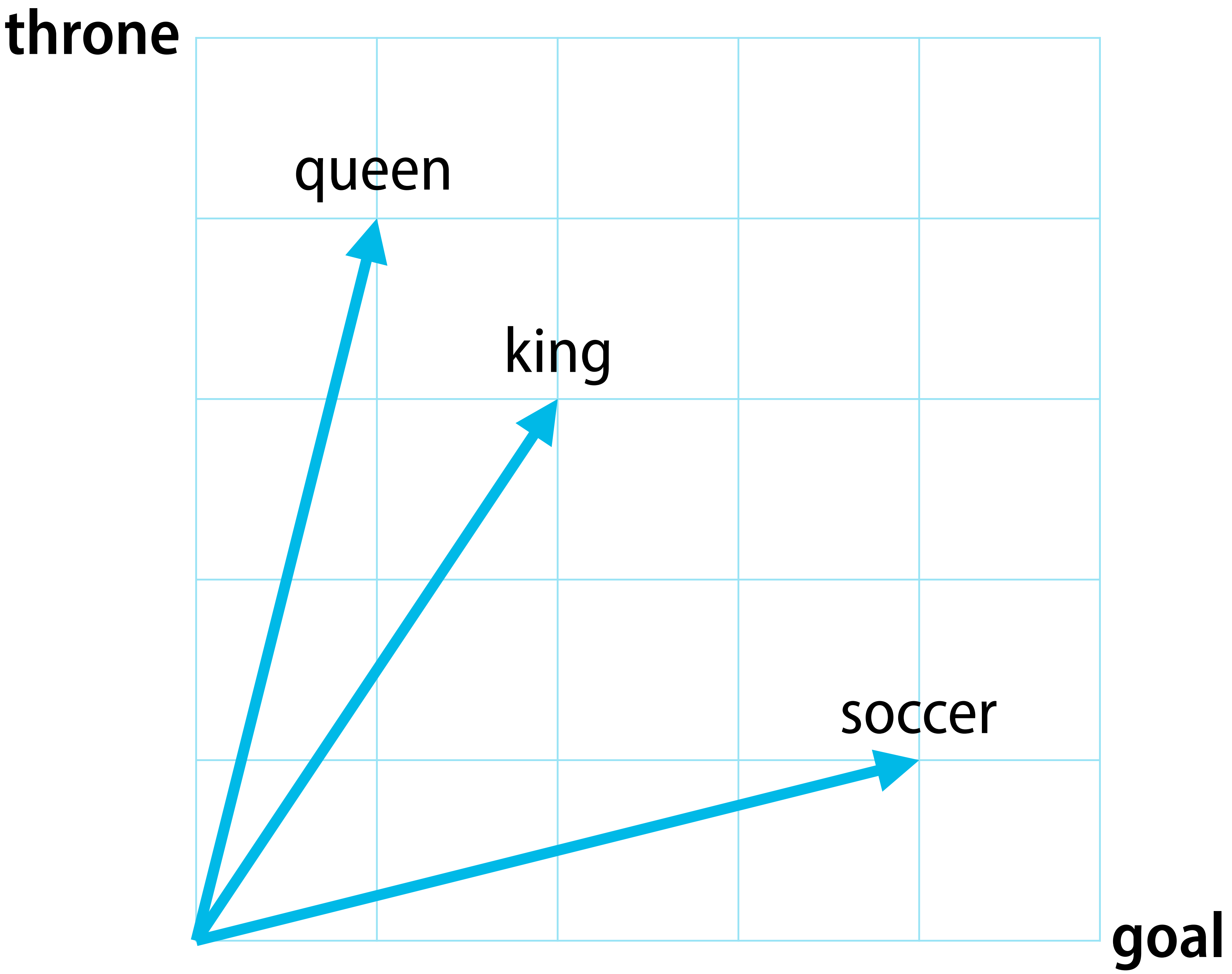
Word–context matrix

		context words						
		crown	throne	reign	Sweden	match	goal	play
target words	queen	2	4	1	2	1	1	0
	king	2	3	1	3	0	2	0
	soccer	0	1	0	4	3	4	2
	hockey	0	0	0	1	2	1	1

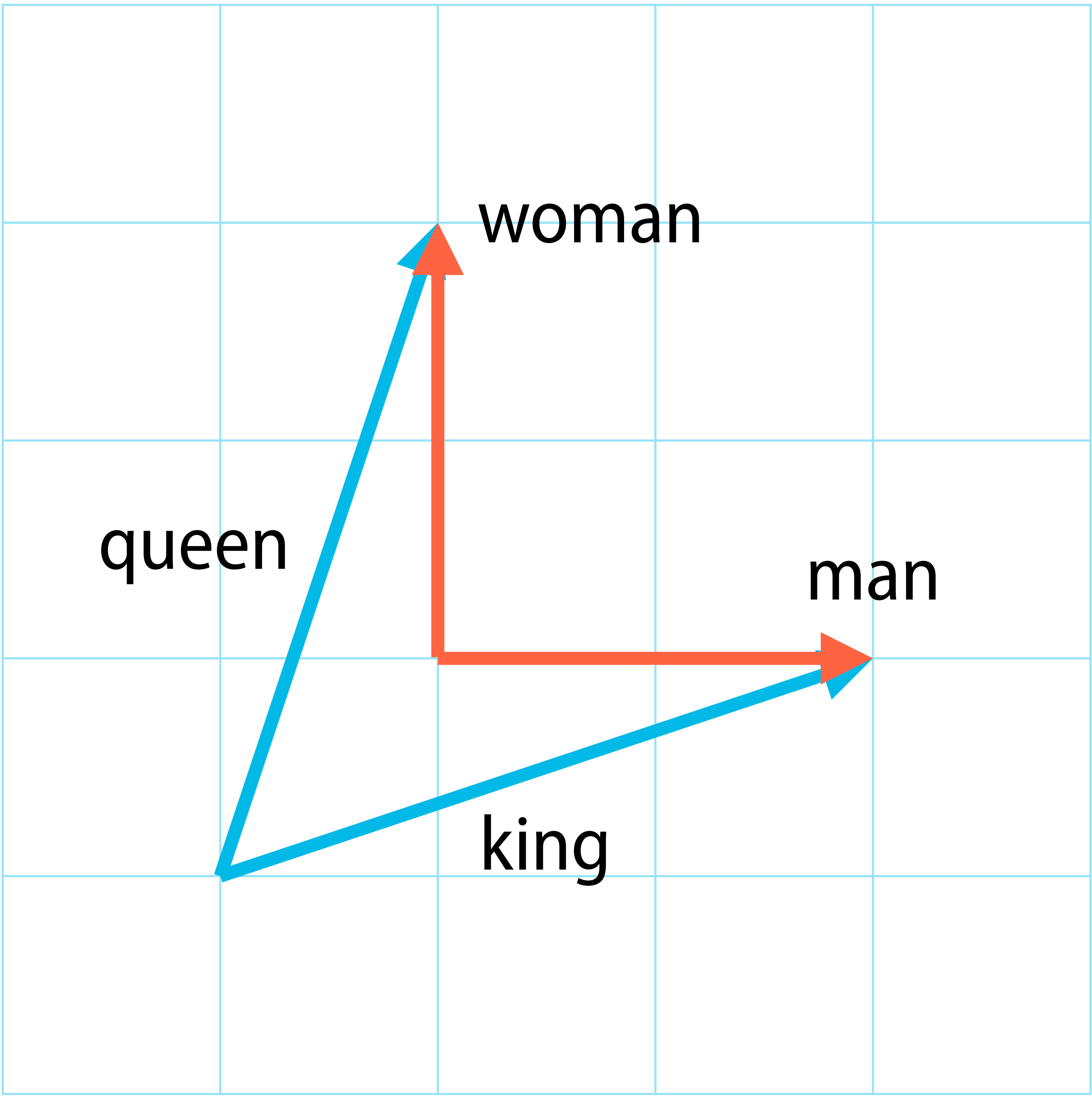
Word–context matrix

target words		context words						
		crown	throne	reign	Sweden	match	goal	play
	queen	2	4	1	2	1	1	0
	king	2	3	1	3	0	2	0
	soccer	0	1	0	4	3	4	2
	hockey	0	0	0	1	2	1	1

Word embedding



Compositional structure of word embeddings



Sparse vectors versus dense vectors

- The rows of word–context matrices are long and sparse.

  length corresponds to number of context words = on the order of 10^4

- We prefer word vectors that are short and dense.

  length on the order of 10^2

- The intuition is that such vectors may be better at capturing generalisations, and easier to use in machine learning.

Simple applications of word embeddings

- Finding similar words
- Answering odd one out questions
- Computing the similarity of short documents

Recognising textual entailment

Two doctors perform surgery on patient.	
Entail	Doctors are performing surgery.
Neutral	Two doctors are performing surgery on a man.
Contradict	Two surgeons are having lunch.

Example from Bowman et al. (2015)

Limitations of word embeddings

- Definition of similarity is completely operational: words are similar if used in similar contexts. But there are many facets of 'similarity'.

  Is a *cat* more similar to a *dog* or to a *tiger*?

- Text data does not reflect many of the more 'trivial' properties of words.

  'Black sheep' stick out, 'white sheep' are just 'sheep'.

- Text corpora reflect human biases in the real world, including stereotypes about race and gender.

  king – man + woman = queen, doctor – man + woman = ?

Goldberg (2017)

Questions related to word embeddings

- Which measure of association strength?

  pointwise mutual information

- Which measure of similarity?

  cosine similarity

- Which definition of context?

  linear context, syntactic context

- Which algorithm to learn word embeddings from text?

  matrix factorisation, direct learning of the low-dimensional vectors

Use of word embeddings in neural networks

Use of word embeddings in neural networks

- Neural networks are witnessing remarkable successes in natural language processing.
- They have started to replace the 'classical' statistical or linear models that defined the state of the art for the past 20 years.
- With neural networks comes a thinking in terms of distributed representations or 'patterns of activations'.
- Word embeddings fit in naturally into this mindset.

Sparse vectors versus dense vectors

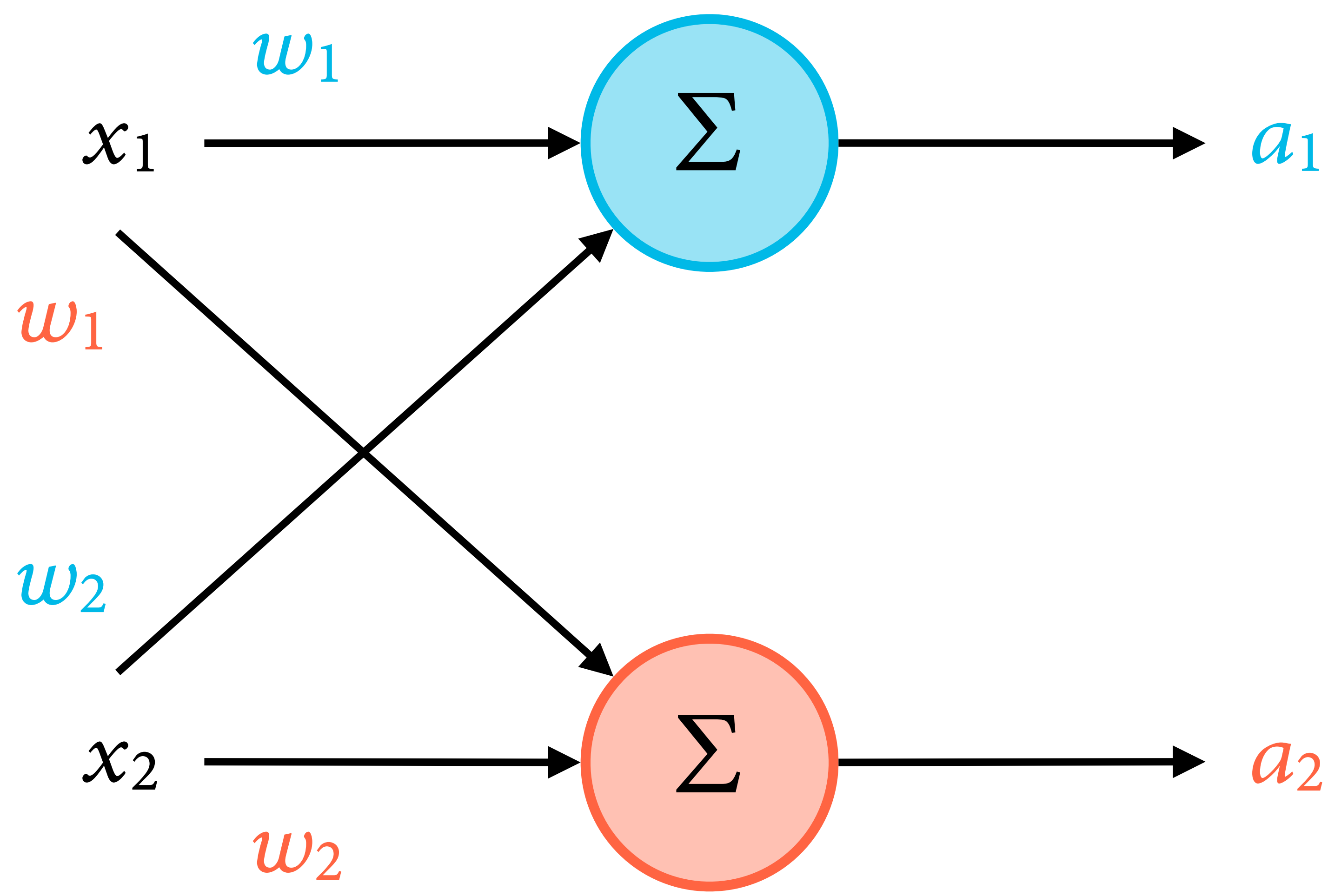
- In classical NLP, features take binary values: on or off. This typically gives rise to long but sparse feature vectors.

  typical length on the order of 10^4 (size of the vocabulary)

- In neural networks, features can take any continuous value. This makes it possible to use short but dense feature vectors.

  typical length on the order of 10^2

The multi-class perceptron



activation = weighted sum of the features

Interpretation of feature weights

- Features whose weights are zero do not contribute to the activation; such features are ignored.
- Features whose weights are positive cause the activation to increase – they suggest that the input belongs to the class.
- Features whose weights are negative cause the activation to decrease – they suggest that the input falls outside of the class.

Part-of-speech tagging with a perceptron

jag	bad	om	en	kort	bit
NN	9,36				
PN	81,72				
VB	−9,18				

Part-of-speech tagging with a perceptron

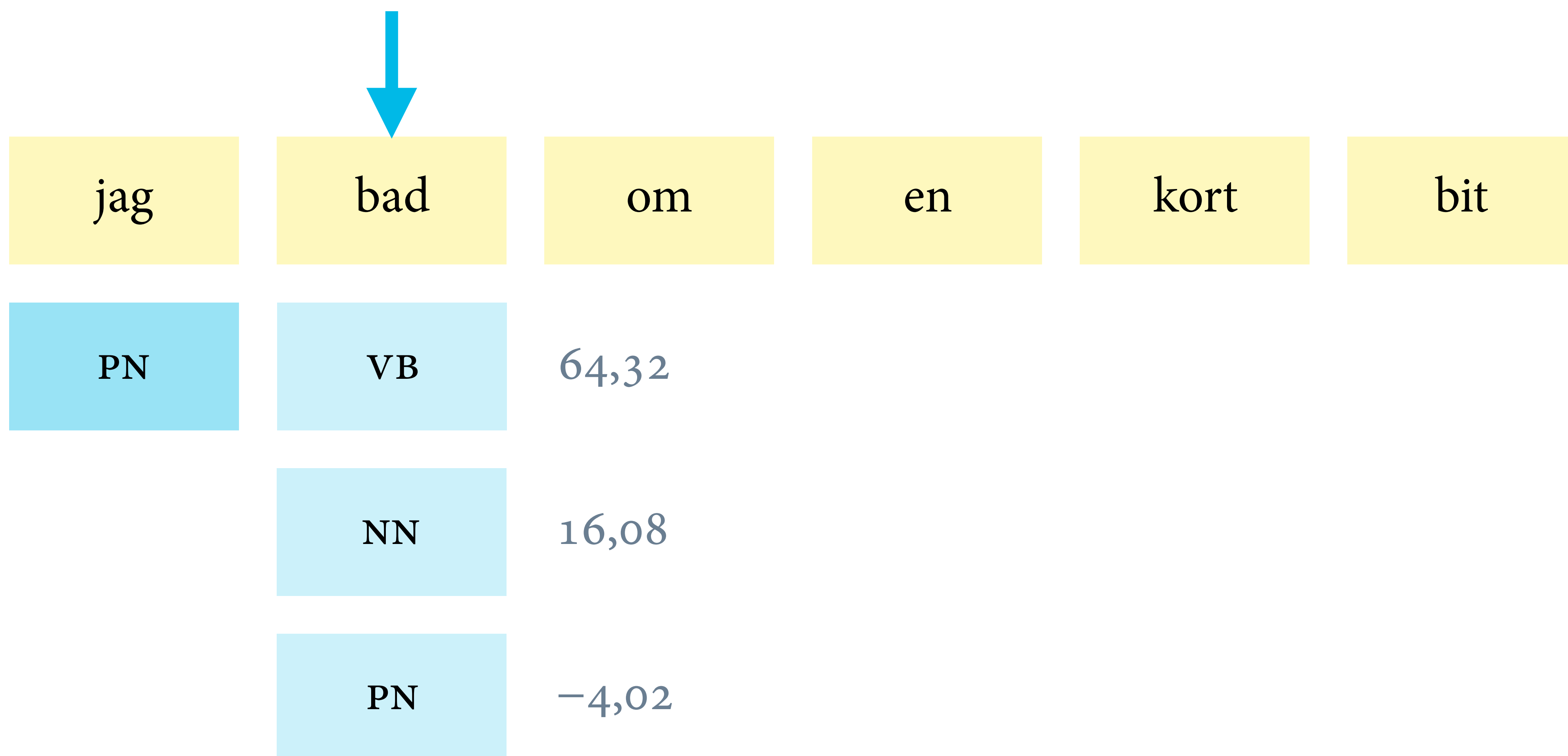
jag	bad	om	en	kort	bit

PN	81,72
NN	9,36
VB	−9,18

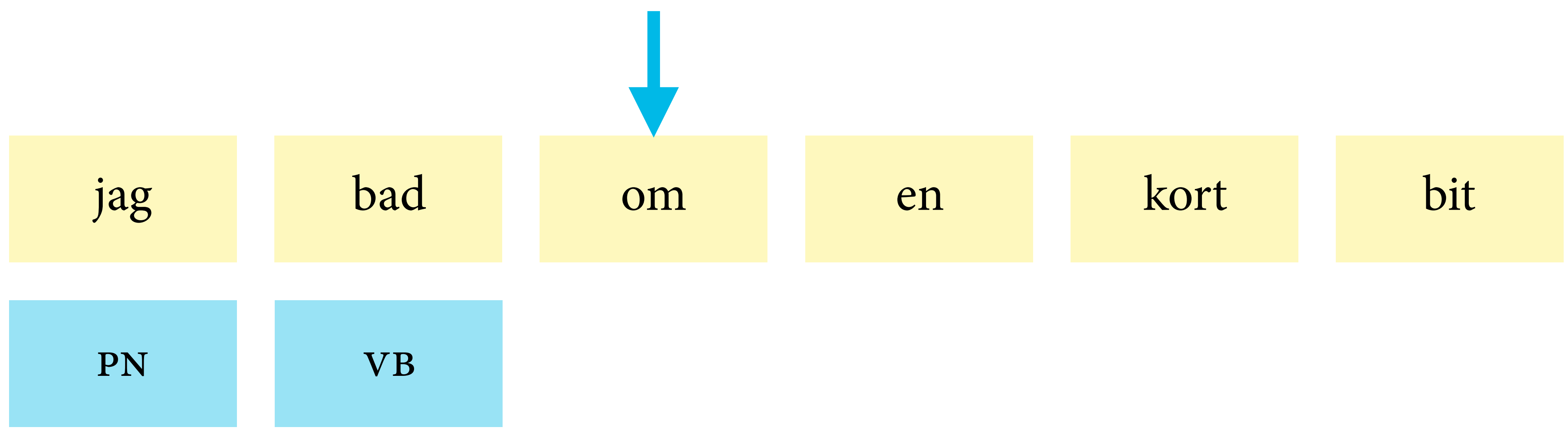
Part-of-speech tagging with a perceptron

jag	bad	om	en	kort	bit
PN	NN	16,08			
	PN	−4,02			
	VB	64,32			

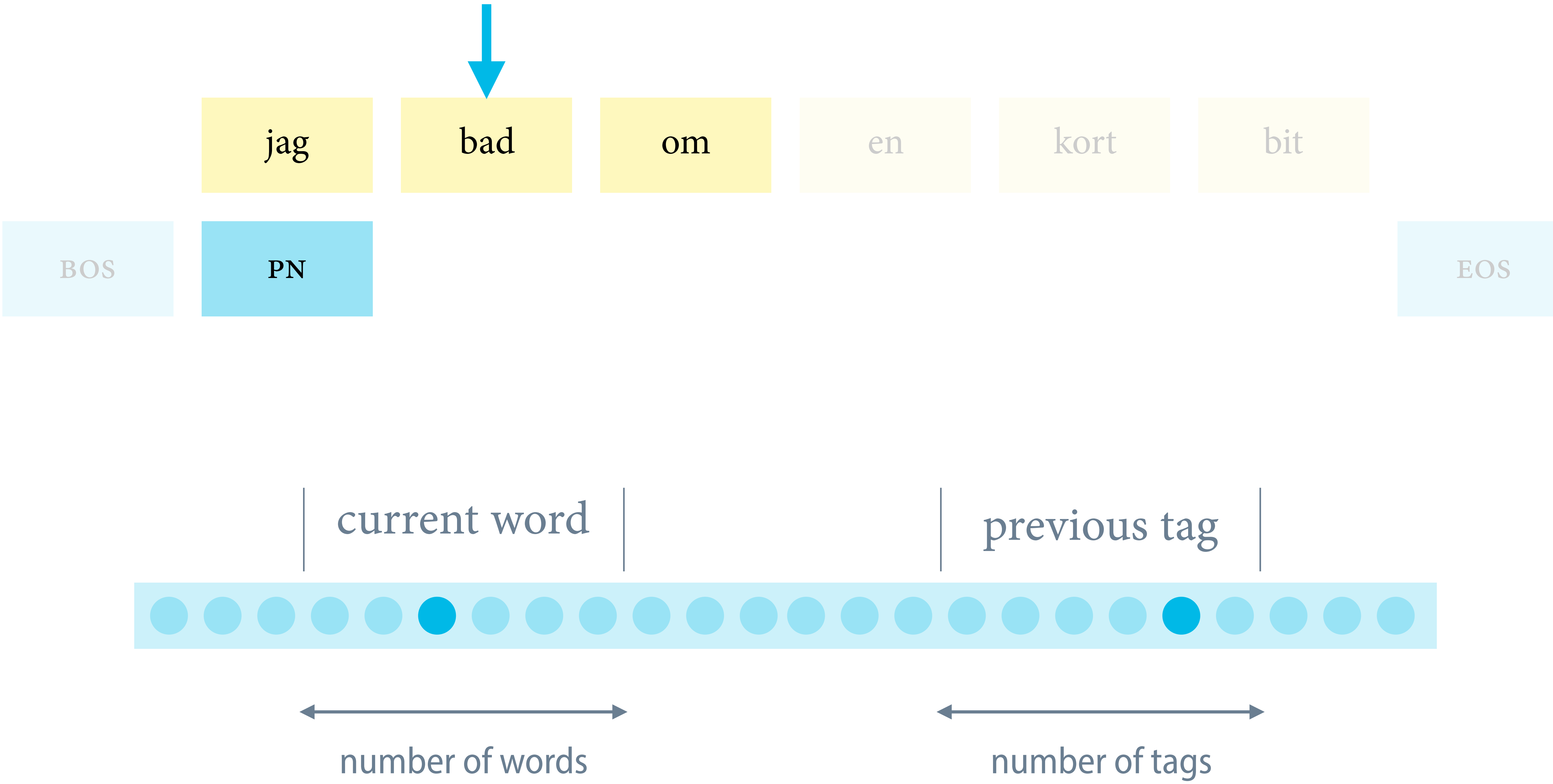
Part-of-speech tagging with a perceptron



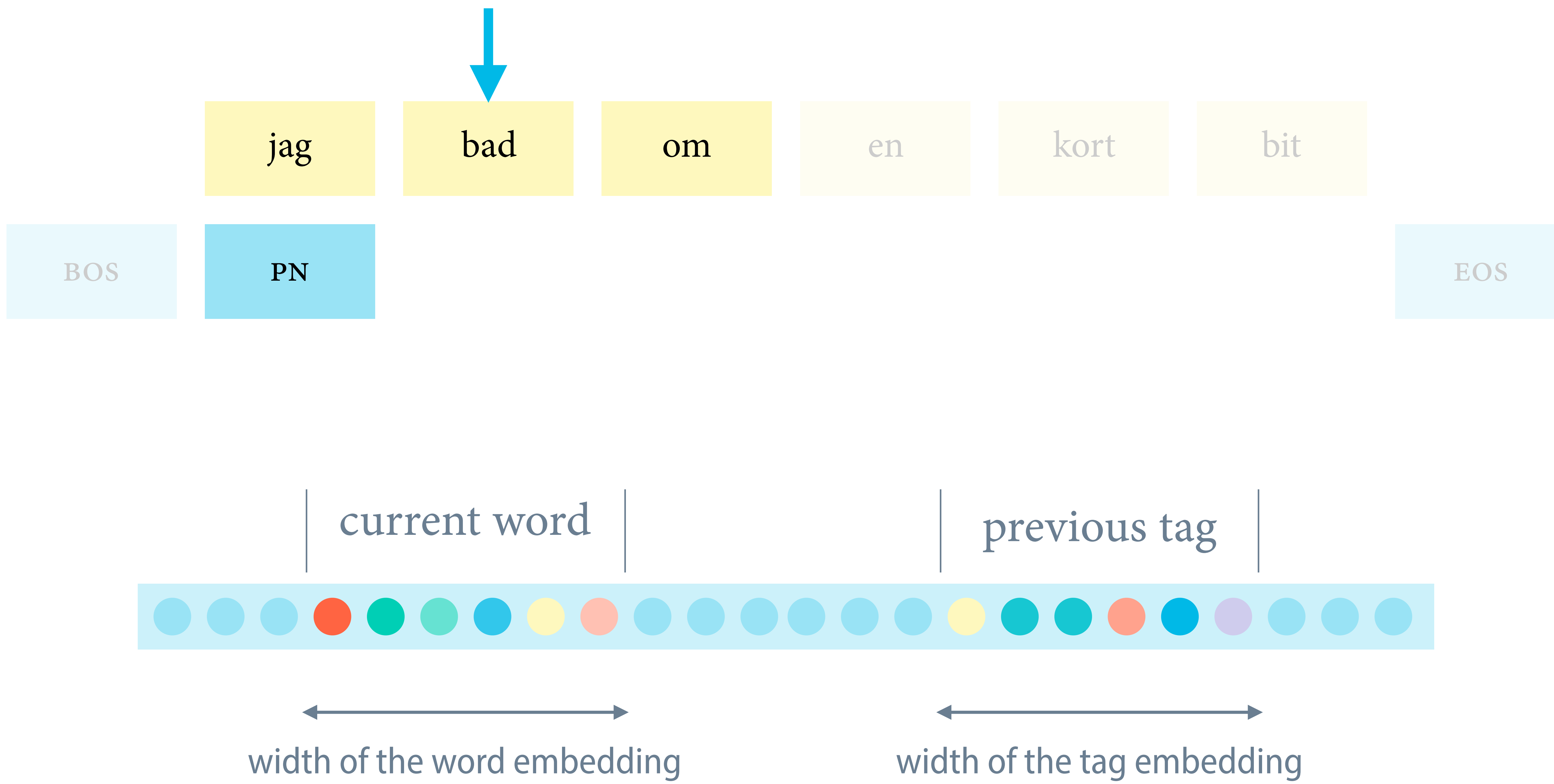
Part-of-speech tagging with a perceptron



Sparse feature vectors



Dense feature vectors



Using neural networks to replace linear models

- Neural networks with word embeddings can often be used as drop-in replacements for linear models.

  multi-class perceptron → feed-forward neural network

- Current models typically either match or improve upon the performance of the ‘classical’ models.

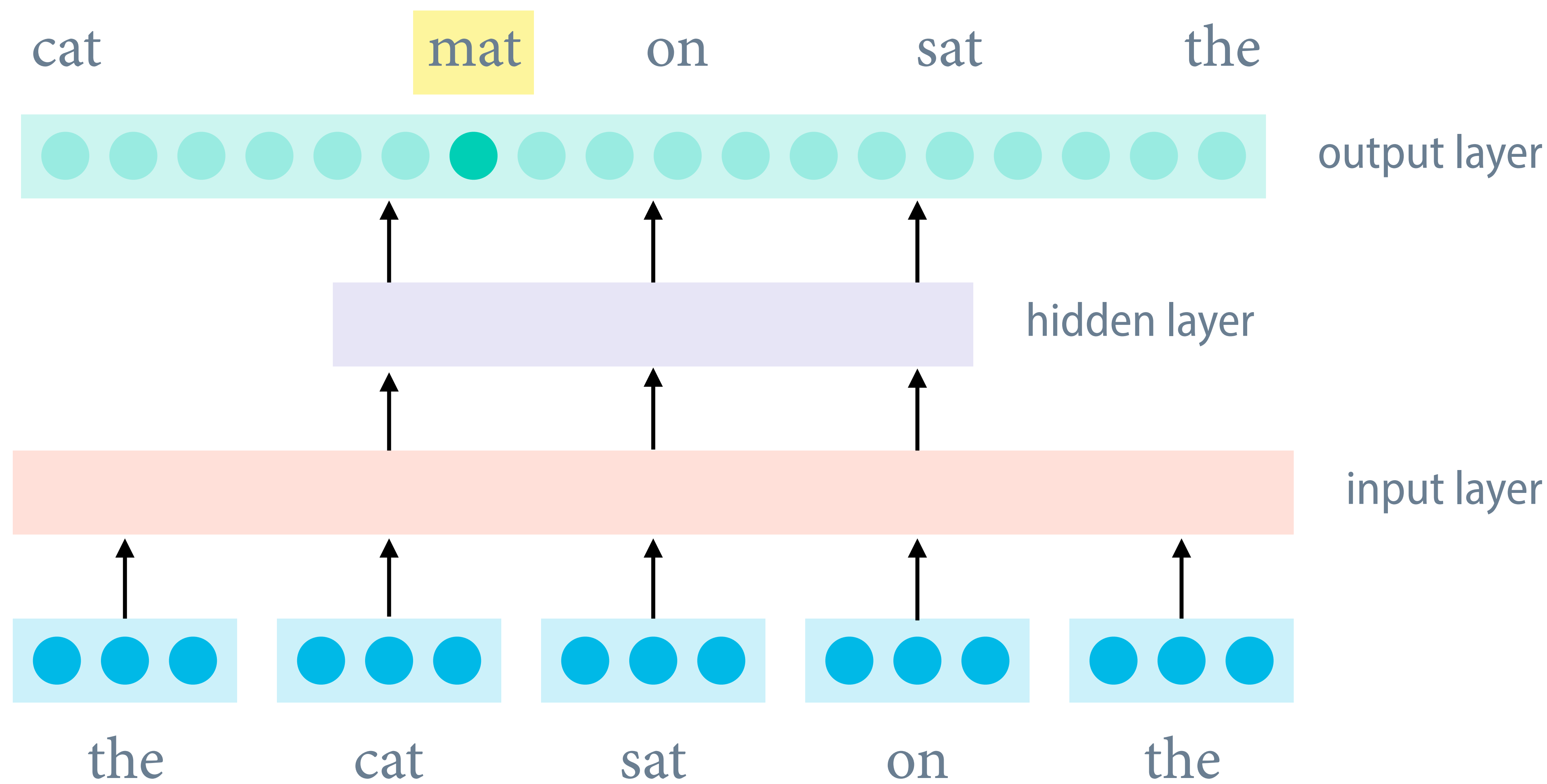
Problems with statistical language models

- Smoothing techniques are intricate and based on manually designed back-off strategies to lower-order models.
- Scaling to large values of n is infeasible because the number of model parameters grows fast and statistics are sparse.

  remember Heaps' law
- Since words in the vocabulary are regarded as atomic units, there is no generalisation across contexts.

  observing *black car* and *blue car* does not influence estimates for *red car*

A simple neural language model



Training the model

- The model is trained with n-grams, where the first $n - 1$ words are used as input and the last word is used as the target label.

  input = *the cat sat on the*, target label = *mat*

- Conceptually, the model is trained by minimising cross-entropy loss. In practice, alternative losses or approximations are used.

  minimising cross-entropy loss = maximising likelihood

Advantages of neural language models

- The model can achieve better perplexity than statistical models with Kneser–Ney smoothing, and scales to much larger n.

- Words in different positions share parameters, making them share statistical strength.

  Everything must pass through the hidden layer.

- The network can learn that in some contexts, only sub-parts of the n-gram are informative.

  implicit back-off

Goldberg (2017)

Use of word embeddings in neural networks

- Neural networks are witnessing remarkable successes in natural language processing.
- They have started to replace the 'classical' statistical or linear models that defined the state of the art for the past 20 years.
- With neural networks comes a thinking in terms of distributed representations or 'patterns of activations'.
- Word embeddings fit in naturally into this mindset.

Questions related to word embeddings

- Which measure of association strength?

  pointwise mutual information

- Which measure of similarity?

  cosine similarity

- Which definition of context?

  linear context, syntactic context

- Which algorithm to learn word embeddings from text?

  matrix factorisation, direct learning of the low-dimensional vectors

Pointwise mutual information

Pointwise mutual information

- Raw counts favour pairs that involve very common contexts.

  the cat, *a cat* will receive higher weight than *cute cat*, *small cat*

- We want a measure that favours contexts in which the target word occurs more often than other words.

- A suitable measure is **pointwise mutual information (PMI)**:

$$\mathrm{PMI}(x, y) = \log \frac{P(x, y)}{P(x)P(y)}$$

Example from Goldberg (2017)

Pointwise mutual information

- We want to use PMI to measure the associative strength between a word w and a context c in a data set D:

$$\mathrm{PMI}(w, c) = \log \frac{P(w, c)}{P(w)P(c)}$$

- We can estimate the relevant probabilities by counting:

$$\mathrm{PMI}(w, c) = \log \frac{\#(w, c)/|D|}{\#(w)/|D| \cdot \#(c)/|D|} = \log \frac{\#(w, c) \cdot |D|}{\#(w) \cdot \#(c)}$$

Positive pointwise mutual information

- Note that PMI is infinitely small for unseen word–context pairs, and undefined for unseen words.
- In **positive pointwise mutual information (PPMI)**, all negative and undefined values are replaced by zero:

$$\mathrm{PPMI}(w, c) = \max\big(\mathrm{PMI}(w, c), 0\big)$$

- Because PPMI assigns high values to rare events, it is advisable to apply a count threshold or smooth the probabilities.

Computing PPMI on a word–context matrix

	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

Computing PPMI on a word–context matrix

	aardvark	computer	data	pinch	result	sugar
apricot	$\frac{0/19}{2/19 \cdot 0/19}$	$\frac{0/19}{2/19 \cdot 3/19}$	$\frac{0/19}{2/19 \cdot 7/19}$	$\frac{1/19}{2/19 \cdot 2/19}$	$\frac{0/19}{2/19 \cdot 5/19}$	$\frac{1/19}{2/19 \cdot 2/19}$
pineapple	$\frac{0/19}{2/19 \cdot 0/19}$	$\frac{0/19}{2/19 \cdot 3/19}$	$\frac{0/19}{2/19 \cdot 7/19}$	$\frac{1/19}{2/19 \cdot 2/19}$	$\frac{0/19}{2/19 \cdot 5/19}$	$\frac{1/19}{2/19 \cdot 2/19}$
digital	$\frac{0/19}{4/19 \cdot 0/19}$	$\frac{2/19}{4/19 \cdot 3/19}$	$\frac{1/19}{4/19 \cdot 7/19}$	$\frac{0/19}{4/19 \cdot 2/19}$	$\frac{1/19}{4/19 \cdot 5/19}$	$\frac{0/19}{4/19 \cdot 2/19}$
information	$\frac{0/19}{11/19 \cdot 0/19}$	$\frac{1/19}{11/19 \cdot 3/19}$	$\frac{6/19}{11/19 \cdot 7/19}$	$\frac{0/19}{11/19 \cdot 2/19}$	$\frac{4/19}{11/19 \cdot 5/19}$	$\frac{0/19}{11/19 \cdot 2/19}$

Computing PPMI on a word–context matrix

	aardvark	computer	data	pinch	result	sugar
apricot	undefined	$\log_2 0.00$	$\log_2 0.00$	$\log_2 4.75$	$\log_2 0.00$	$\log_2 4.75$
pineapple	undefined	$\log_2 0.00$	$\log_2 0.00$	$\log_2 4.75$	$\log_2 0.00$	$\log_2 4.75$
digital	undefined	$\log_2 3.17$	$\log_2 0.68$	$\log_2 0.00$	$\log_2 0.95$	$\log_2 0.00$
information	undefined	$\log_2 0.58$	$\log_2 1.48$	$\log_2 0.00$	$\log_2 1.38$	$\log_2 0.00$

Computing PPMI on a word–context matrix

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

Cosine similarity

Distance-based similarity

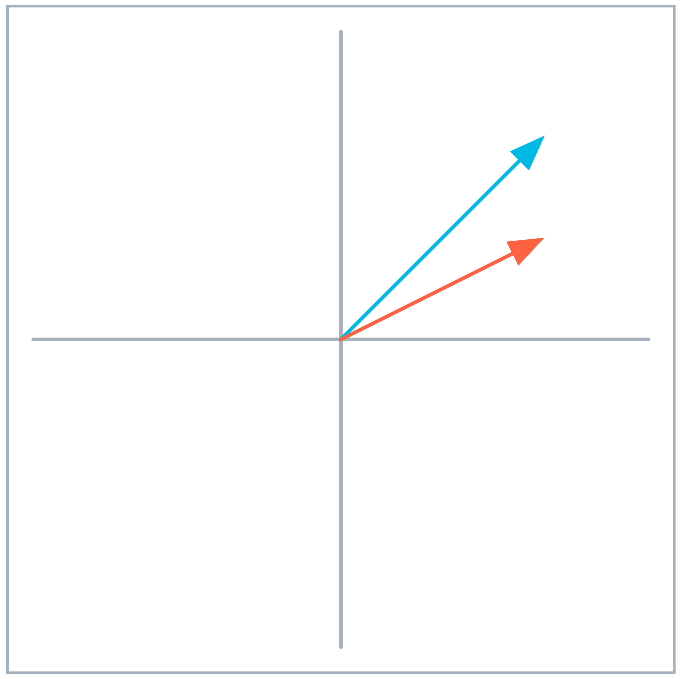
- If we can represent words as vectors, then we can measure word similarity as the distance between the word vectors.
- Most measures of vector similarity are based on the dot product or inner product from linear algebra.

The dot product

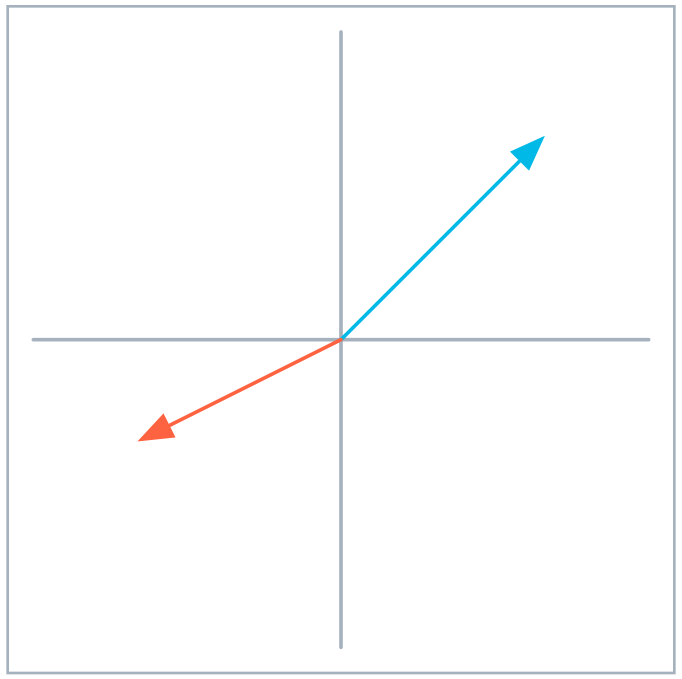
v_1	v_2	w_1	w_2
$+2$	$+2$	$+2$	$+1$

v_1	v_2	w_1	w_2
$+2$	$+2$	-2	-1

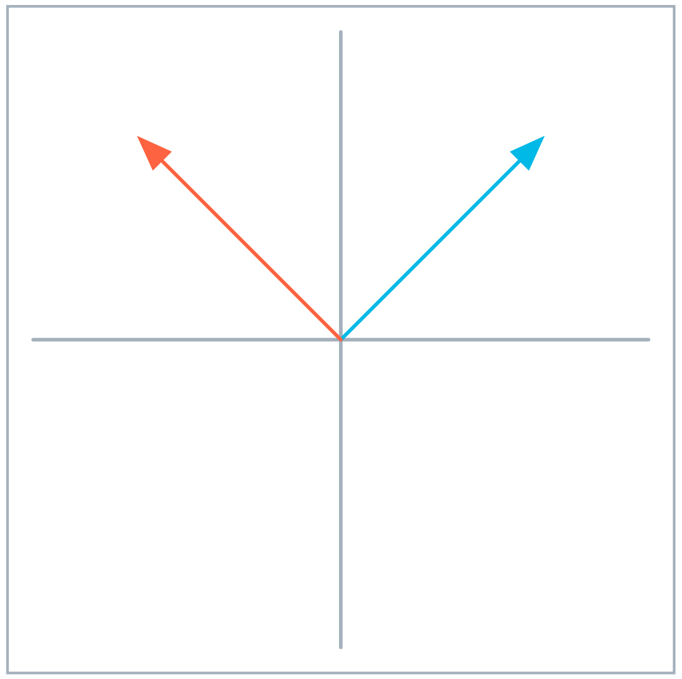
v_1	v_2	w_1	w_2
$+2$	$+2$	-2	$+2$



$$\boldsymbol{v} \cdot \boldsymbol{w} = +6$$



$$\boldsymbol{v} \cdot \boldsymbol{w} = -6$$



$$\boldsymbol{v} \cdot \boldsymbol{w} = \pm 0$$

Problems with the dot product

- The dot product will be higher for vectors that represent words that have high co-occurrence counts or PPMI values.
- This means that, all other things being equal, the dot product of two words will be greater if the words are frequent.
- This makes the dot product problematic because we would like a similarity metric that is independent of frequency.

Cosine similarity

- We can fix the dot product as a metric by computing with unit vectors, that is, normalising for vector length:

$$\cos(\boldsymbol{v}, \boldsymbol{w}) = \frac{\boldsymbol{v}}{|\boldsymbol{v}|} \cdot \frac{\boldsymbol{w}}{|\boldsymbol{w}|} = \frac{\boldsymbol{v} \cdot \boldsymbol{w}}{|\boldsymbol{v}||\boldsymbol{w}|} = \frac{\sum_{i=1}^{d} v_i w_i}{\sqrt{\sum_{i=1}^{d} v_i^2}\sqrt{\sum_{i=1}^{d} w_i^2}}$$

- This length-normalised dot product is the **cosine similarity**, whose values range from −1 (opposite) to +1 (identical).

cosine of the angle between the two vectors

Properties of cosine similarity

- The cosine similarity between two vectors ranges from -1 (point into opposite directions) to $+1$ (point into the same direction).
- A cosine similarity of 0 means that the two vectors are unrelated (orthogonal).
- When using raw frequencies or PPMI, vector components are non-negative, and the range of cosine similarity is $[0, 1]$.

Different types of contexts

Syntactic contexts

- Some work replaces the linear context within a sentence with a syntactic context defined on dependency trees.
- The context of a word is defined to be its proximity in a dependency tree for the sentence in which it occurs.

  syntactic head + relation towards the head (subject, object)
- In this approach, two words will be close in vector space if they can fill the same role in a sentence.

Target word (embedded)	Continuous bag of 5 words	Continuous bag of 2 words	Syntactic dependencies
batman	nightwing, aquaman, catwoman, superman, manhunter	superman, superboy, aquaman, catwoman, batgirl	superman, superboy, supergirl, catwoman, aquaman
hogwarts	dumbledore, hallows, half-blood, malfoy, snape	evernight, sunnydale, garderobe, blandings, collinwood	sunnydale, collinwood, calarts, greendale, millfield
florida	gainesville, fla, jacksonville, tampa, lauderdale	fla, alabama, gainesville, tallahassee, texas	texas, louisiana, georgia, california, carolina
dancing	singing, dance, dances, dancers, tap-dancing	singing, dance, dances, breakdancing, clowning	singing, rapping, breakdancing, miming, busking

Examples from Goldberg (2017)

Obtaining word embeddings

Obtaining word embeddings

- Word embeddings can be easily trained from any text corpus using available tools.

  word2vec, Gensim, GloVe

- Pre-trained word vectors for English, Swedish, and various other languages are available for download.

  word2vec, Swectors, Polyglot project, spaCy

Algorithms for computing word embeddings

- word embeddings via matrix factorisation
- word embeddings via neural language models
- word2vec

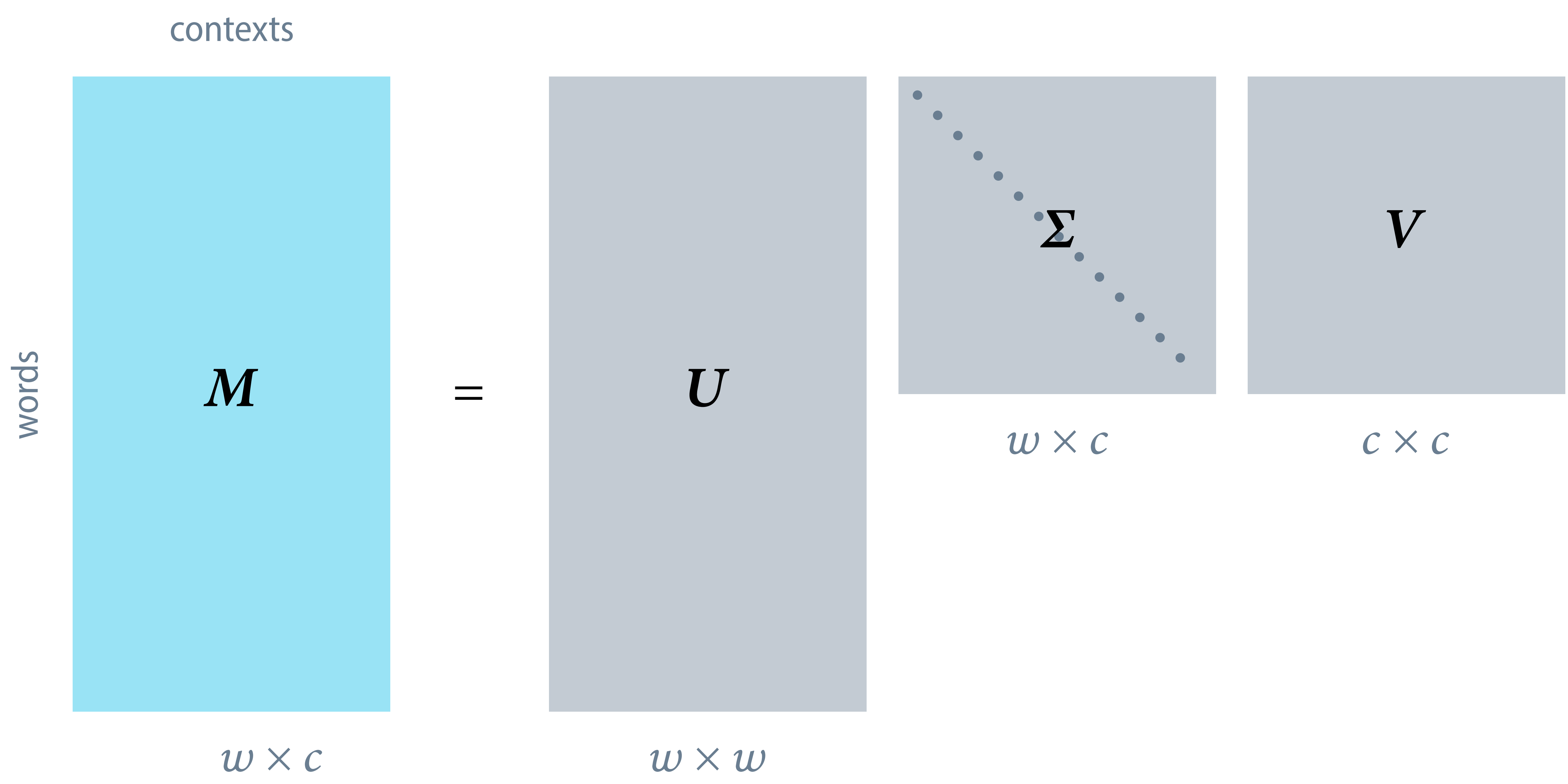
Word embeddings via singular value decomposition

- We would like to have word vectors that are short and dense.
- One idea is to approximate the word–context matrix by another matrix with fewer columns.

  in practice: based on the PPMI version of the word–context matrix
- This problem can be solved by computing the **singular value decomposition** of the word–context matrix.

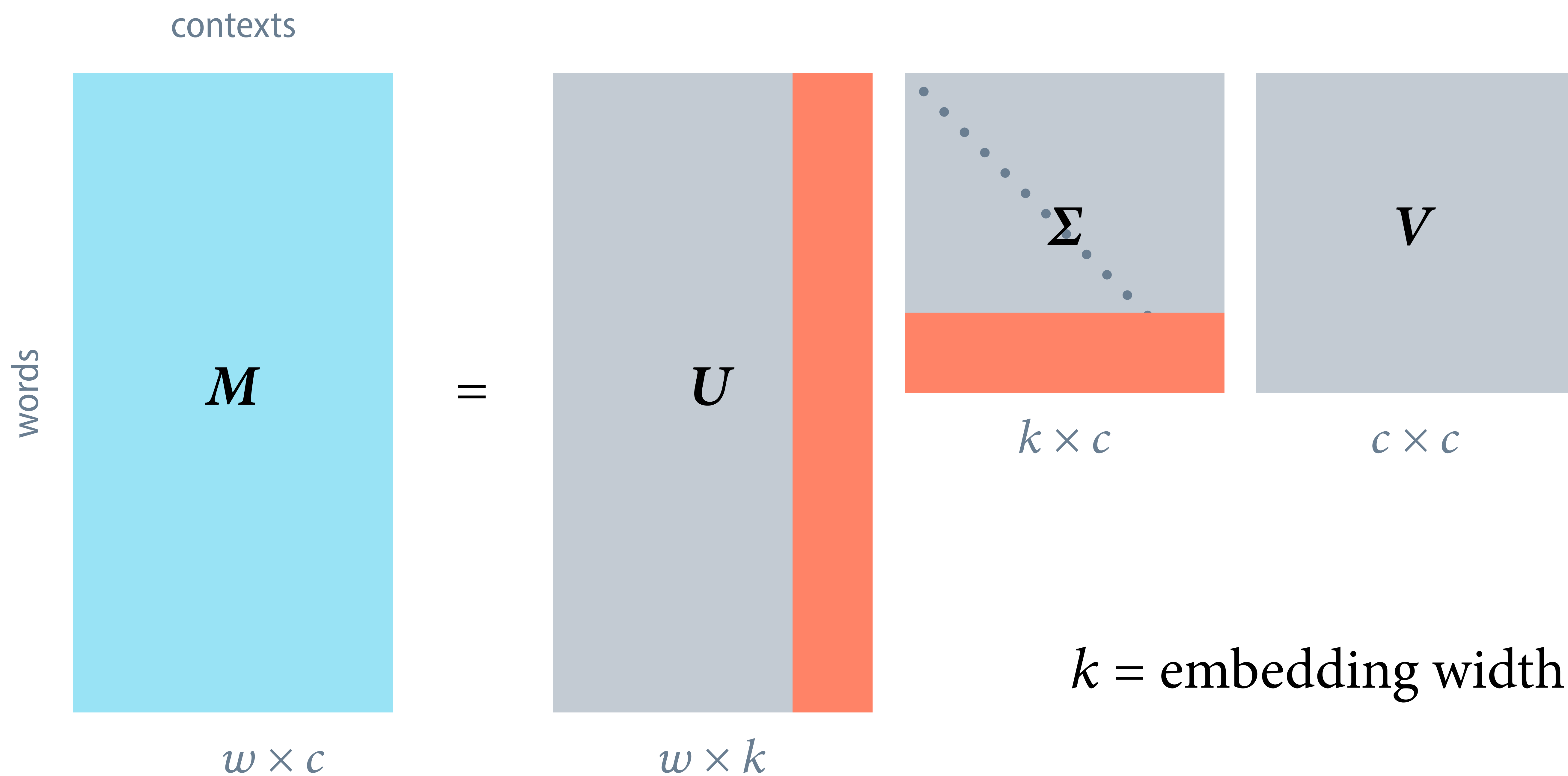
  also used in principal component analysis, latent semantic analysis

Singular value decomposition



Landauer and Dumais (1997)

Truncated singular value decomposition



k = embedding width

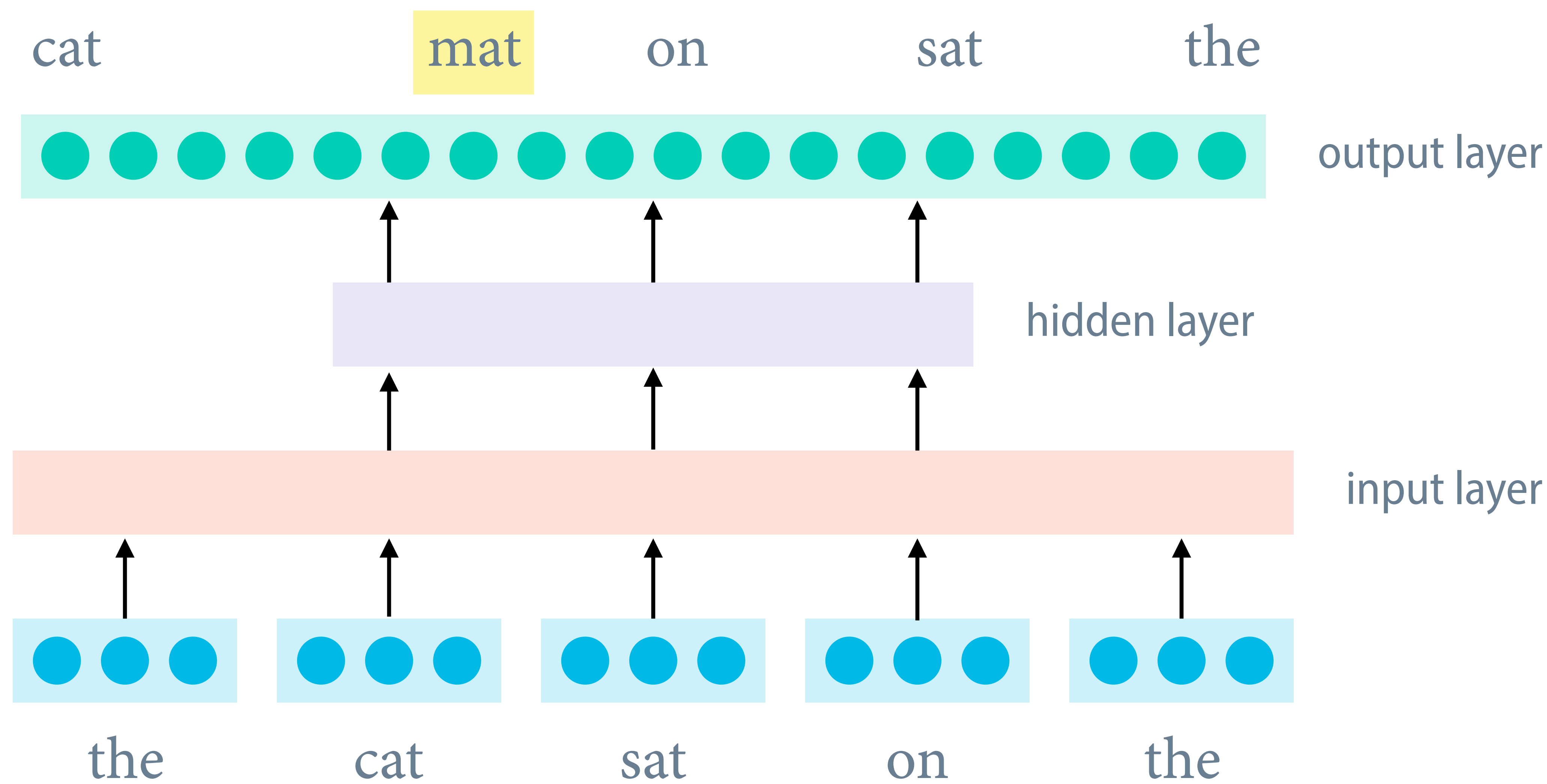
Word embeddings via singular value decomposition

- Each row of the (truncated) matrix $\boldsymbol{U}$ is a k-dimensional vector that represents the 'most important' information about a word.

- A practical problem is that computing the singular value decomposition for large matrices is computationally expensive.

  but has to be done only once!

Reminder: A simple neural language model



From neural language models to word embeddings

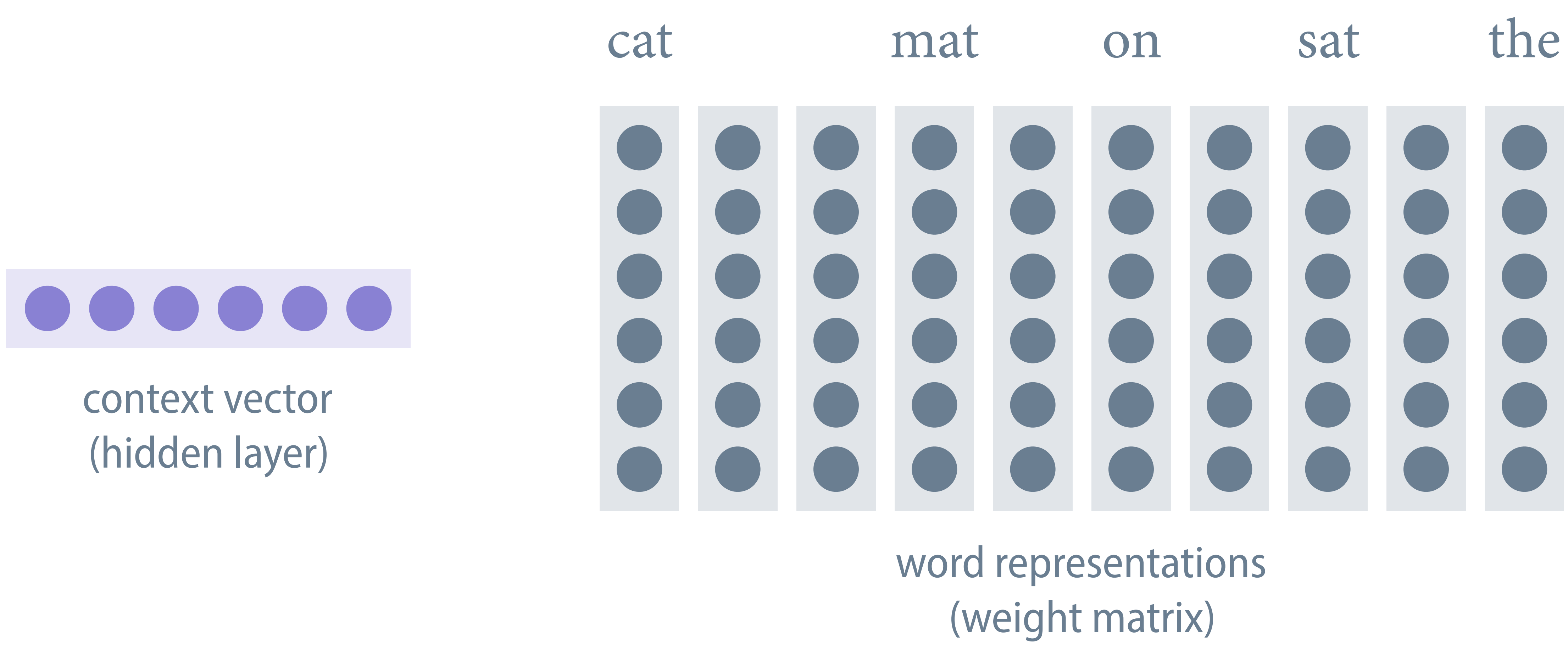
- The neural language model is trained to predict the probability of the next word being w, given the preceding words:

$$\hat{\boldsymbol{y}} = P(w \mid \text{preceding words}) = \text{softmax}(\boldsymbol{hW} + \boldsymbol{b})$$

- Each row of the matrix $\boldsymbol{W}$ is a $\dim(\boldsymbol{h})$-dimensional vector that is associated with some vocabulary item w.

- We can view this vector as a representation of w that captures its compatibility with the context represented by the vector $\boldsymbol{h}$.

Network weights = word embeddings



Intuitively, words that occur in similar contexts will have similar word representations.

Training word embeddings using a language model

- Initialise the word vectors with random values.

  typically by uniform sampling from an interval around 0

- Train the language model on large volumes of text.

  word2vec: 100 billion words

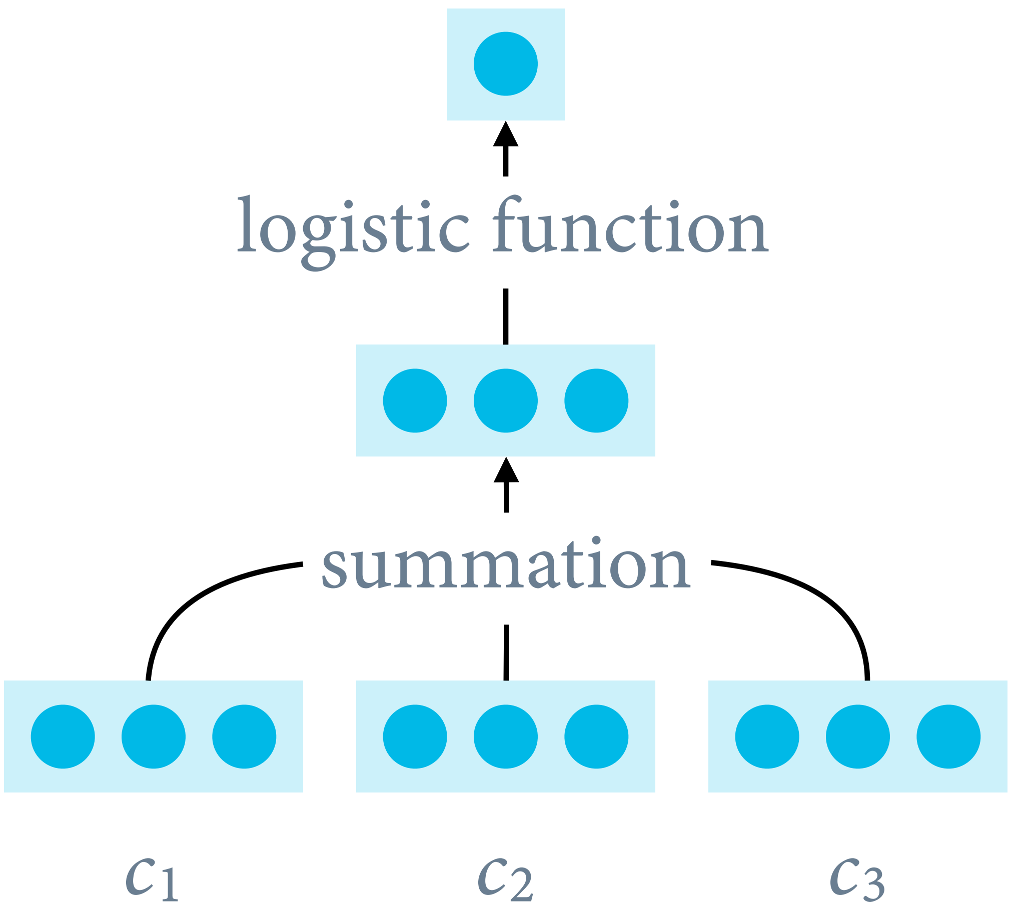
- Read off word embeddings from the weight matrices.

Google's word2vec

- Google's word2vec implements two different training algorithms for word embeddings: **continuous bag-of-words** and **skip-gram**.
- Both algorithms obtain word embeddings as 'side products' of a binary prediction task: 'Is this an actual word–context pair?'
- Positive examples are generated from a corpus. Negative examples are generated by taking k copies of a positive example and randomly replacing the target word with some other word.

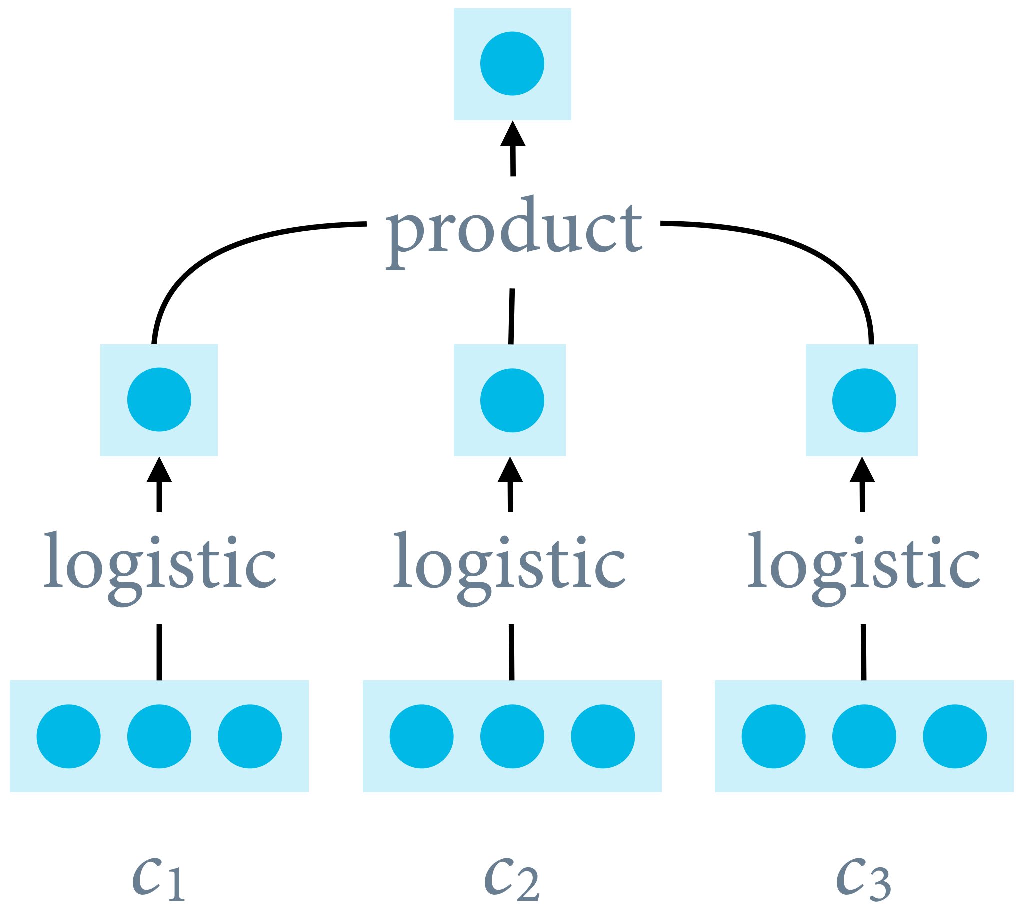
Continuous bag-of-words

Is $(w; c_1, c_2, c_3)$ an actual word–context pair?



Skip-gram

Is $(w; c_1, c_2, c_3)$ an actual word–context pair?



Connecting the two worlds

- The two algorithmic approaches that we have seen take two seemingly very different perspectives: 'count-based' and 'neural'.
- Remarkably, it turns out that the two are closely related.
- In particular, a careful analysis of the skip-gram model reveals that this model is implicitly factorising the PMI matrix – without ever constructing the word–context matrix!

  Levy and Goldberg (2014)

Questions related to word embeddings

- Which measure of association strength?

  pointwise mutual information

- Which measure of similarity?

  cosine similarity

- Which definition of context?

  linear context, syntactic context

- Which algorithm to learn word embeddings from text?

  matrix factorisation, direct learning of the low-dimensional vectors