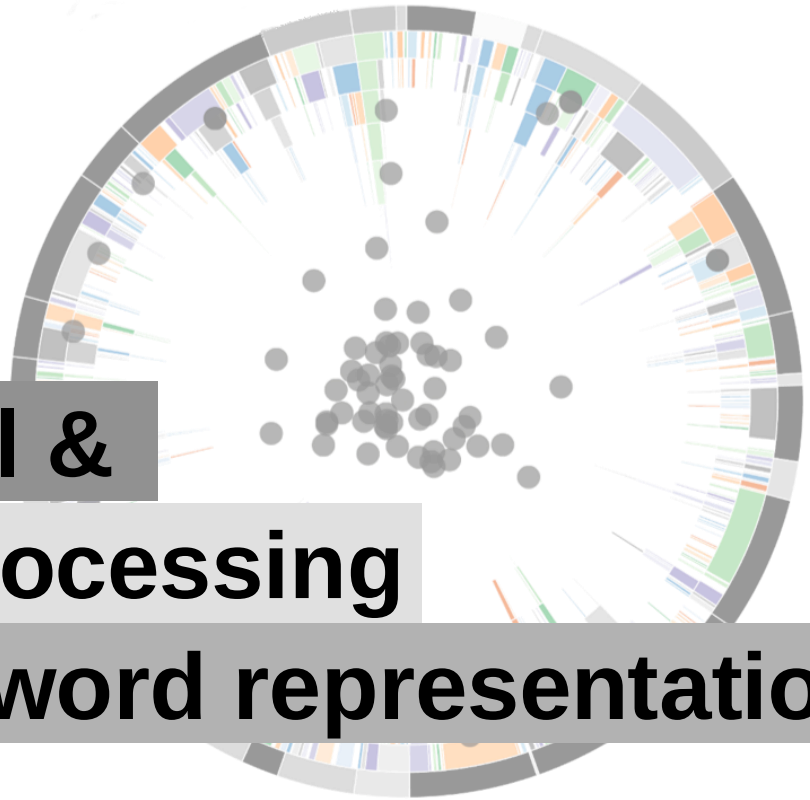
Cluster of Excellence
The Politics of Inequality

Steinbeis Transfer Center
Linguistic Data Analysis

UNIVERSITÄT PASSAU



Information Retrieval & Natural Language Processing

Week 7: Distributed word representations

Annette Hautli-Janisz, Prof. Dr.

12 December 2024

IR & NLP: Course schedule winter 2024/25

	When?	What?
Week 1	17 October 2024	Introduction
Week 2	24 October 2024	Indexing, Boolean IR
Week 3	31 October 2024	--
Week 4	7 November 2024	--
Week 5	14 November 2024	Scoring, term weighting, the vector space model
Week 6	21 November 2024	Relevance and probabilistic IR
Week 7	28 November 2024	Tolerant retrieval and index compression
Week 8	5 December 2024	Evaluation in IR
Week 9	12 December 2024	Distributed word representations for IR

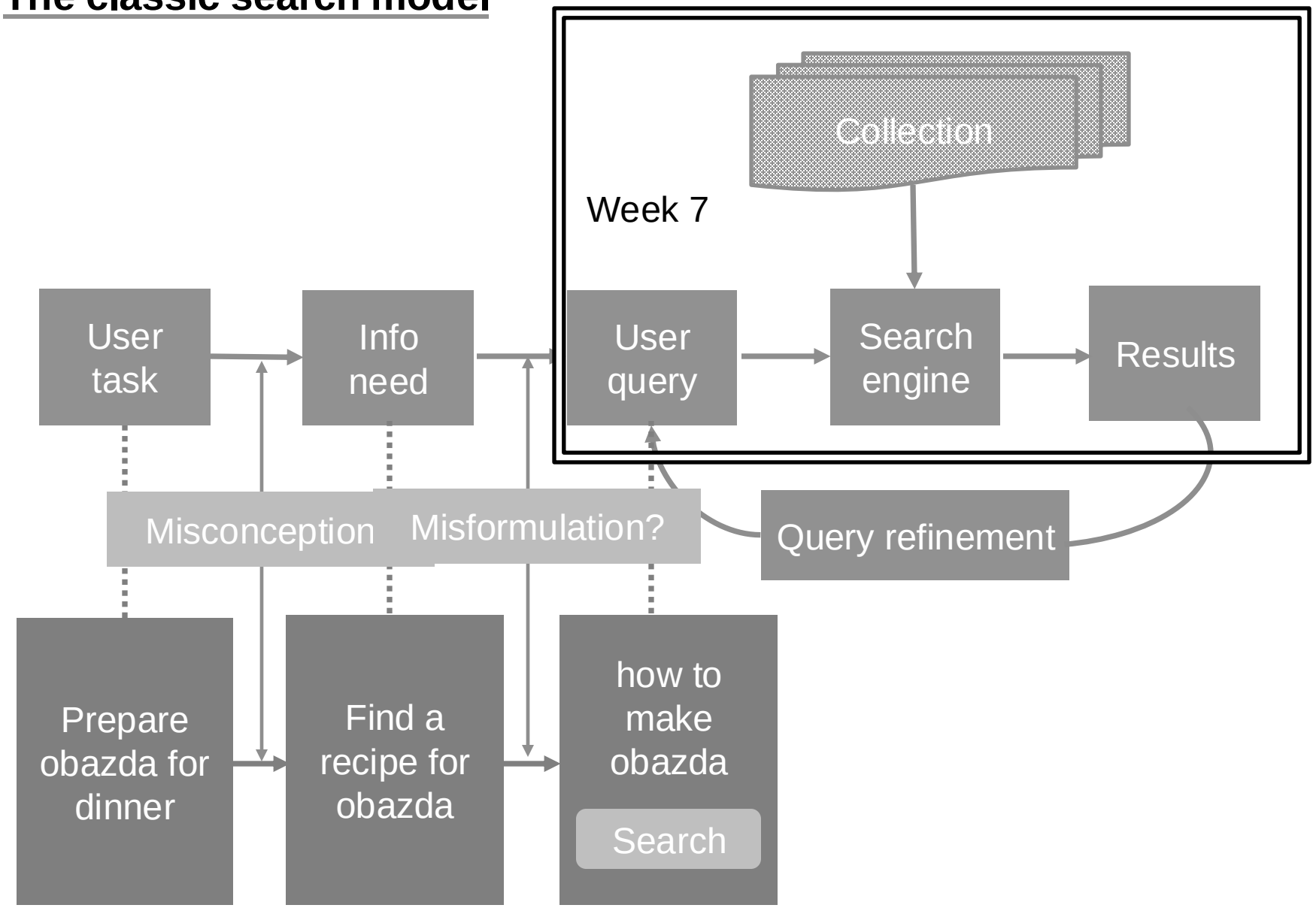
IR & NLP: Course schedule winter 2024/25

	When?	What?
Week 10	19 December 2024	Natural Language Processing
Week 11	9 January 2025	NLP with Python
Week 12	16 January 2025	Personalization in IR systems
Week 13	23 January 2025	AI & ethics
Week 14	30 January 2025	Recap
Week 15	6 February 2025	No class (conference trip)

(also on stud.IP)

Exam dates now on stud.IP.

The classic search model



How can we more robustly match a user's search intent?

One of the challenges in information retrieval (IR) is the vocabulary mismatch problem, which happens when the terms between queries and documents are lexically different but semantically similar.

We want to *understand* a query, not just do string.equals().

- User searches for "Dell notebook battery size", we'd like to match documents discussing "Dell laptop battery capacity".
- User searches for "Passau hotel", we'd also like to match documents containing "Passau motel"

→ A pure keyword-matching IR system does nothing to help.

How can we more robustly match a user's search intent?

Simple things we have already discussed that can help:

- spelling correction
- case folding
- stemming
- lemmatization

We'd still like to *understand* the query.

How can we more robustly match a user's search intent?

Expanding the document and/or the query.

Include information on word similarity:

- A manual thesaurus of synonyms for query expansion (Part 1 today)
  e.g., WordNet
- A measure of word similarity
  Calculated from a big document collection (Part 2 today)
  Calculated by query log mining (common on the web)

A manual thesaurus

There are several lexical semantic resources available in Natural Language Processing which capture semantic properties of words and semantic relations to other words.

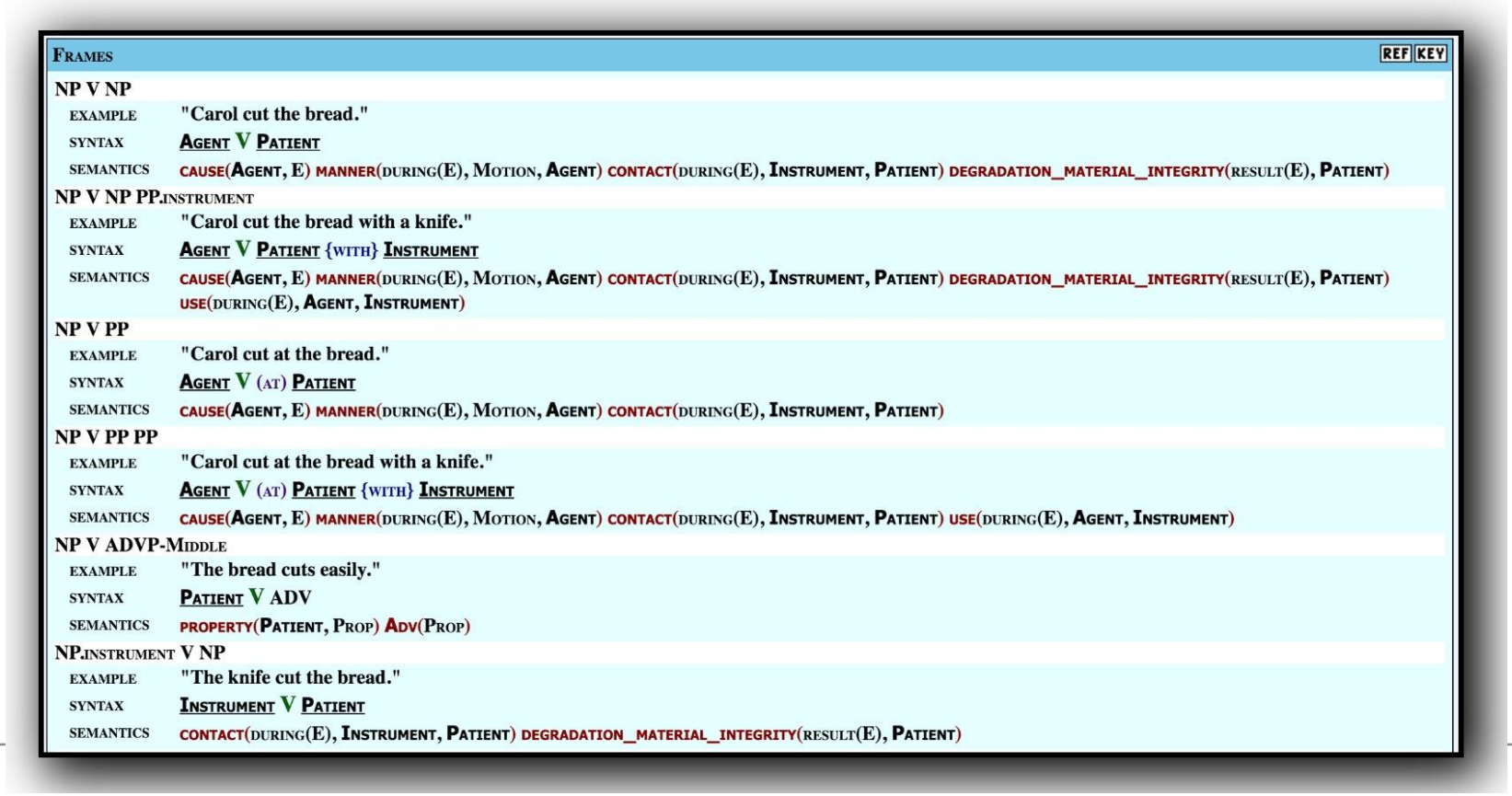
One of the most prominent ones is WordNet: wordnet.princeton.edu

Other ones:

- VerbNet
- PropBank
- FrameNet

A manual thesaurus

VerbNet: encodes verbs, their meaning and the function of their arguments



FRAMES REF KEY

NP V NP

EXAMPLE "Carol cut the bread."

SYNTAX AGENT V PATIENT

SEMANTICS CAUSE(AGENT, E) MANNER(DURING(E), MOTION, AGENT) CONTACT(DURING(E), INSTRUMENT, PATIENT) DEGRADATION_MATERIAL_INTEGRITY(RESULT(E), PATIENT)

NP V NP PP.INSTRUMENT

EXAMPLE "Carol cut the bread with a knife."

SYNTAX AGENT V PATIENT {WITH} INSTRUMENT

SEMANTICS CAUSE(AGENT, E) MANNER(DURING(E), MOTION, AGENT) CONTACT(DURING(E), INSTRUMENT, PATIENT) DEGRADATION_MATERIAL_INTEGRITY(RESULT(E), PATIENT) USE(DURING(E), AGENT, INSTRUMENT)

NP V PP

EXAMPLE "Carol cut at the bread."

SYNTAX AGENT V (AT) PATIENT

SEMANTICS CAUSE(AGENT, E) MANNER(DURING(E), MOTION, AGENT) CONTACT(DURING(E), INSTRUMENT, PATIENT)

NP V PP PP

EXAMPLE "Carol cut at the bread with a knife."

SYNTAX AGENT V (AT) PATIENT {WITH} INSTRUMENT

SEMANTICS CAUSE(AGENT, E) MANNER(DURING(E), MOTION, AGENT) CONTACT(DURING(E), INSTRUMENT, PATIENT) USE(DURING(E), AGENT, INSTRUMENT)

NP V ADVP-MIDDLE

EXAMPLE "The bread cuts easily."

SYNTAX PATIENT V ADV

SEMANTICS PROPERTY(PATIENT, PROP) ADV(PROP)

NP.INSTRUMENT V NP

EXAMPLE "The knife cut the bread."

SYNTAX INSTRUMENT V PATIENT

SEMANTICS CONTACT(DURING(E), INSTRUMENT, PATIENT) DEGRADATION_MATERIAL_INTEGRITY(RESULT(E), PATIENT)

An automatically generated thesaurus

Speech and Language Processing. Daniel Jurafsky & James H. Martin, 2021, Vector Semantics and Embeddings, Chapter 6.

Sections 6.1: Lexical Semantics

Section 6.2. : Vector Semantics

Section 6.3: Words and vectors

Section 6.4: Cosine for measuring similarity

Section 6.7: Word2vec

An automatically generated thesaurus

Fundamental notion is the *distributional hypothesis*:

Definition 1: Two words are similar if they co-occur with similar words.
Definition 2: Two words are similar if they occur in a given grammatical relation with the same words.

Famous quote by Firth 1957, p. 11: *"You shall know a word by the company it keeps."* – one of the most successful ideas of modern statistical NLP.

…government debt problems turning into	***banking***	*crises as happened in 2009…*
…saying that Europe needs unified	***banking***	*regulation to replace the hodgepodge…*
…India has just given its	***banking***	*system a shot in the arm…*

These words represent the concept of 'banking'

An automatically generated thesaurus

Vector semantics instantiates this linguistic hypothesis by learning representations of the meaning of words (“embeddings”) directly from their distributions in text.

Orthogonal approach to manual thesaurus classification: use a self-supervised way to learn from input instead of creating representations by hand.

Today: static embeddings of word2vec.

Vector semantics

Standard way to represent word meaning in NLP, helping us to model synonymy or word relatedness.

For example: the word ‘ongchoi’ which you haven’t heard before. But you see it in those contexts:

Ongchoi is delicious sauteed with garlic.	...spinach sauteed with garlic over rice...
Ongchoi is superb over rice.	...chard stems and leaves are delicious...
...ongchoi leaves with salty sauces...	...collard greens and other salty leafy greens

‘Ongchoi’ occurs with the same words as ‘spinach’, ‘chard’ and ‘collard greens’ → ‘ongchoi’ must be a similar concept

Vector semantics

This methodology is what vector semantics does: it counts the words in the context of ‘ongchoi’.

- Vector semantics: represent a word as a point in a multidimensional semantic space.
- The semantic space is derived (in ways we’ll see) from the distributions of word neighbors.
- Vectors for representing words are called *embeddings.*

Vector semantics

Figure 6.1 A two-dimensional (t-SNE) projection of embeddings for some words and phrases, showing that words with similar meanings are nearby in space. The original 60-dimensional embeddings were trained for sentiment analysis. Simplified from Li et al. (2015) with colors added for explanation.

Distinct regions for positive, negative and neutral words.

Semantic properties of embeddings

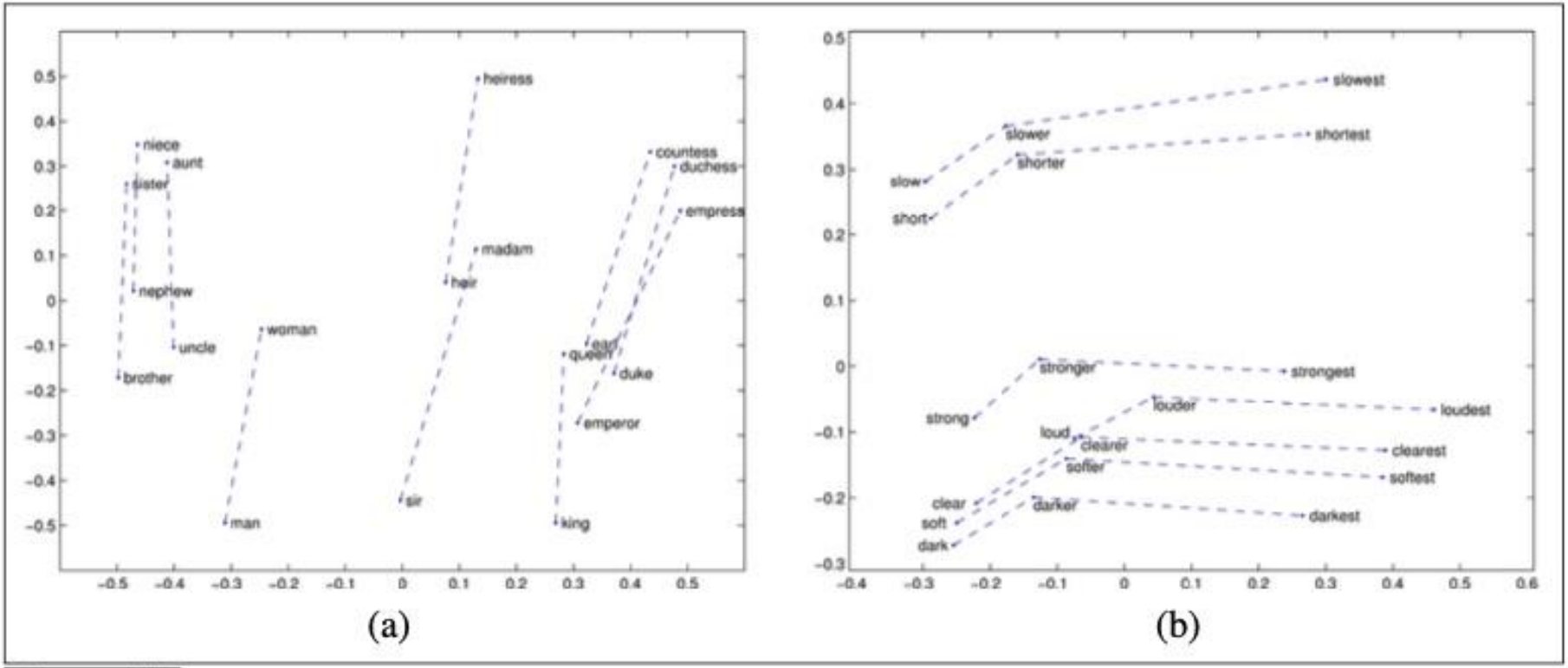


Figure 6.16 Relational properties of the GloVe vector space, shown by projecting vectors onto two dimensions. (a) $(\overrightarrow{\text{king}}) - \overrightarrow{\text{man}} + \overrightarrow{\text{woman}}$ is close to $\overrightarrow{\text{queen}}$. (b) offsets seem to capture comparative and superlative morphology (Pennington et al., 2014).

Vector semantics

There are two models that are most commonly used:

- the tf-idf model (you've heard about this): simple function of the counts of nearby words – an important baseline.
  - Problem: very long, sparse vectors, i.e., mostly zeros because most words simply never in the context of others.

- the word2vec model family: constructing short, dense vectors that have useful semantic properties.

Standard way to compute semantic similarity between embeddings: cosine similarity.

Recap: the term-document matrix

	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

Figure 6.3 The term-document matrix for four words in four Shakespeare plays. The red boxes show that each document is represented as a column vector of length four.

Each document is represented as a count vector.

In the Shakespeare example, document vectors are of dimension 4.

In reality, the vectors representing each document have dimensionality |v|, the vocabulary size.

Recap: the term-document matrix

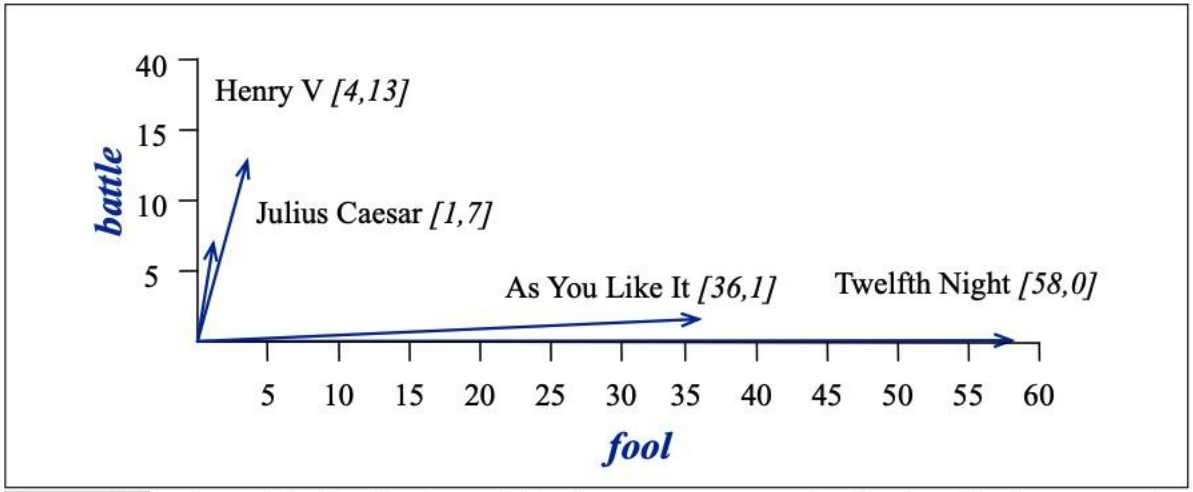


Figure 6.4 A spatial visualization of the document vectors for the four Shakespeare play documents, showing just two of the dimensions, corresponding to the words *battle* and *fool*. The comedies have high values for the *fool* dimension and low values for the *battle* dimension.

How can this way of encoding information be used in Information Retrieval?

Words as vectors: word dimensions

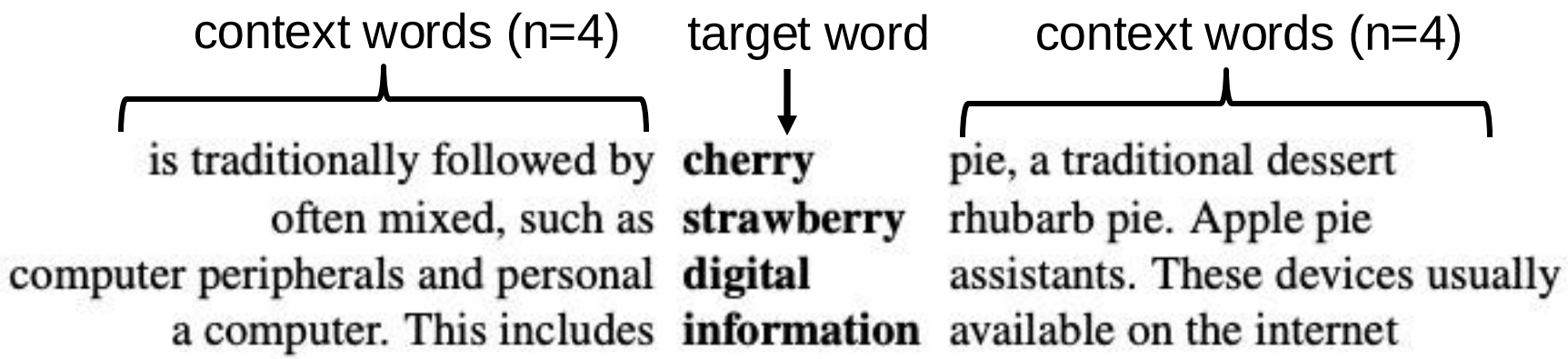
An alternative to the term-document matrix is to use the term-term matrix (or the term-context matrix), in which the columns are labeled by the context words rather than documents.

This matrix is of dimensionality |v| x |v|.

Rows are target words.

Columns are context words, the words that co-occur with the target word in a window of n words (or even in the same document).

Words as vectors: word dimensions



→ word-word co-occurrence matrix

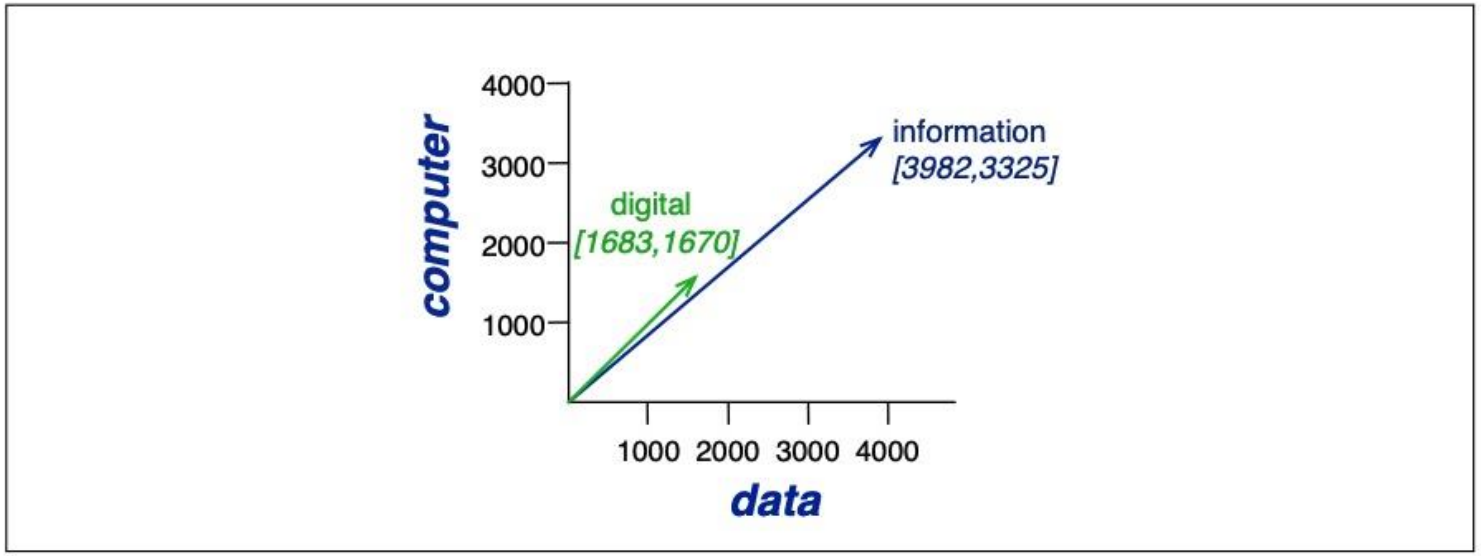
	aardvark	...	computer	data	result	pie	sugar	...
cherry	0	...	2	8	9	442	25	...
strawberry	0	...	0	0	1	60	19	...
digital	0	...	1670	1683	85	5	4	...
information	0	...	3325	3982	378	5	13	...

Figure 6.6 Co-occurrence vectors for four words in the Wikipedia corpus, showing six of the dimensions (hand-picked for pedagogical purposes). The vector for *digital* is outlined in red. Note that a real vector would have vastly more dimensions and thus be much sparser.

Words as vectors: word dimensions

	aardvark	...	computer	data	result	pie	sugar	...
cherry	0	...	2	8	9	442	25	...
strawberry	0	...	0	0	1	60	19	...
digital	0	...	1670	1683	85	5	4	...
information	0	...	3325	3982	378	5	13	...

Figure 6.6 Co-occurrence vectors for four words in the Wikipedia corpus, showing six of the dimensions (hand-picked for pedagogical purposes). The vector for *digital* is outlined in red. Note that a real vector would have vastly more dimensions and thus be much sparser.



Cosine for measuring similarity

To measure the similarity between two target words *v* and *w* (with the same dimensions!) we use the cosine of the angle between the vectors.

The cosine is based on the dot product:

$$\text{dot product}(\mathbf{v},\mathbf{w}) = \mathbf{v}\cdot\mathbf{w} = \sum_{i=1}^{N} v_i w_i = v_1 w_1 + v_2 w_2 + \ldots + v_N w_N$$

The dot product tends to be high when two vectors have large values in the same dimensions.

When is the dot product 0?

Cosine for measuring similarity

Problem of the raw dot product: favors longer vectors, with higher values in each dimensions → favors more frequent target words.

But we want to know similarity regardless of of their frequency.

→ Normalize for length:

$$\text{cosine}(\mathbf{v},\mathbf{w}) = \frac{\mathbf{v}\cdot\mathbf{w}}{|\mathbf{v}||\mathbf{w}|} = \frac{\sum_{i=1}^{N} v_i w_i}{\sqrt{\sum_{i=1}^{N} v_i^2}\sqrt{\sum_{i=1}^{N} w_i^2}}$$

Cosine for measuring similarity

Example:

	pie	data	computer
cherry	442	8	2
digital	5	1683	1670
information	5	3982	3325

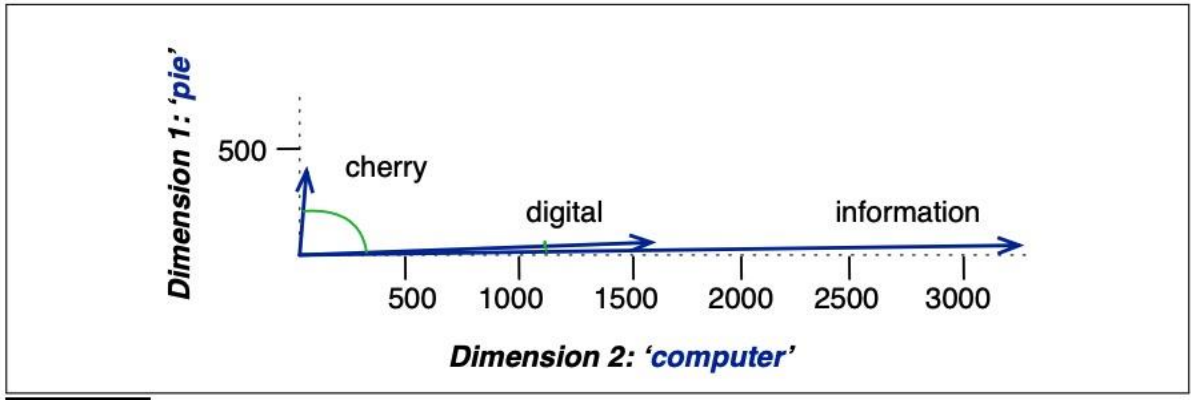
→ Which of the words ‘cherry’ or ‘digital’ is closer to ‘information’? Compute the normalized cosine.

$$\text{cosine}(\mathbf{v},\mathbf{w}) = \frac{\mathbf{v}\cdot\mathbf{w}}{|\mathbf{v}||\mathbf{w}|} = \frac{\sum_{i=1}^{N} v_i w_i}{\sqrt{\sum_{i=1}^{N} v_i^2}\sqrt{\sum_{i=1}^{N} w_i^2}}$$

Cosine for measuring similarity

Example:

	pie	data	computer
cherry	442	8	2
digital	5	1683	1670
information	5	3982	3325



“New” approach: Neural embeddings

So far: long vectors, usually sparse, dimensions corresponding to words in the vocabulary.

Now: a more powerful word representation: *embeddings.*

- Short, dense vectors.
- Number of dimensions *d* ranging from 50-1000.

Dense vectors work better in every NLP task than sparse vectors. “We don’t completely understand all the reasons for this” (p. 17, IIR, Chapter 6), one reason: classifier has to learn fewer weights when using fewer dimensions.

Basic idea of learning neural network word embeddings

We define a model that aims to predict a relation between a target word w_t and context words in terms of word vectors, e.g., p(context | w_t)

which has a loss function, e.g., J = 1 – p(w_{-t}|w_t)

We look at many positions *t* in a big language corpus

We keep adjusting the vector representations of words to minimize this loss.

Word2vec

Word2vec is a shallow, two-layer neural network which is trained to reconstruct linguistic contexts of words.

The intuition behind it: instead of counting how often each target word *w* appears in the context of ‘banking’, we’ll train a classifier on a binary prediction task:

“Is word *w* likely to show up in the context of ‘banking’?

Revolutionary intuition: self-supervised! If target word *w* occurs in the context of ‘banking’, it is a gold correct answer!

→ no need for hand-labelled data

Context

Word2vec models build on context, i.e., the embedding is learned by looking at nearby words.

Intuition: If a group of words is always found close to the same words, they will end up having similar embeddings (countries, animals, etc.)

Here: window size of ± 2 context words

turning into banking crises	(**turning**, into), (**turning**, banking)
turning **into** banking crises	(**into**, turning), (**into**, banking), (**into**, crises)
turning into **banking** crises	(**banking**, turning), (**banking**, into), (**banking**, crises)
turning into banking **crises**	(**crises**, into), (**crises**, banking)

Context

We can’t feed the neural network inputs as actual characters → represent words “mathematically”.

Start with a one-hot encoding: vocabulary as vector dimensions, ‘1’ which represents the corresponding word in the vocabulary, all other dimensions with ‘0’.

What is the one-hot encoding of the example ‘turning into banking crises’ with a window size of ± 2 context words?

Word2vec is a family of algorithms

Mikolov et al. 2013: a framework for learning word vectors.

Static embeddings:

- one embedding for word *w* as the target word, one embedding if word *w* is a context word
- the embedding is static, i.e., not dependent on the context the word is used in

(Contextual embeddings like BERT representations: the vector of each word is different in different contexts)

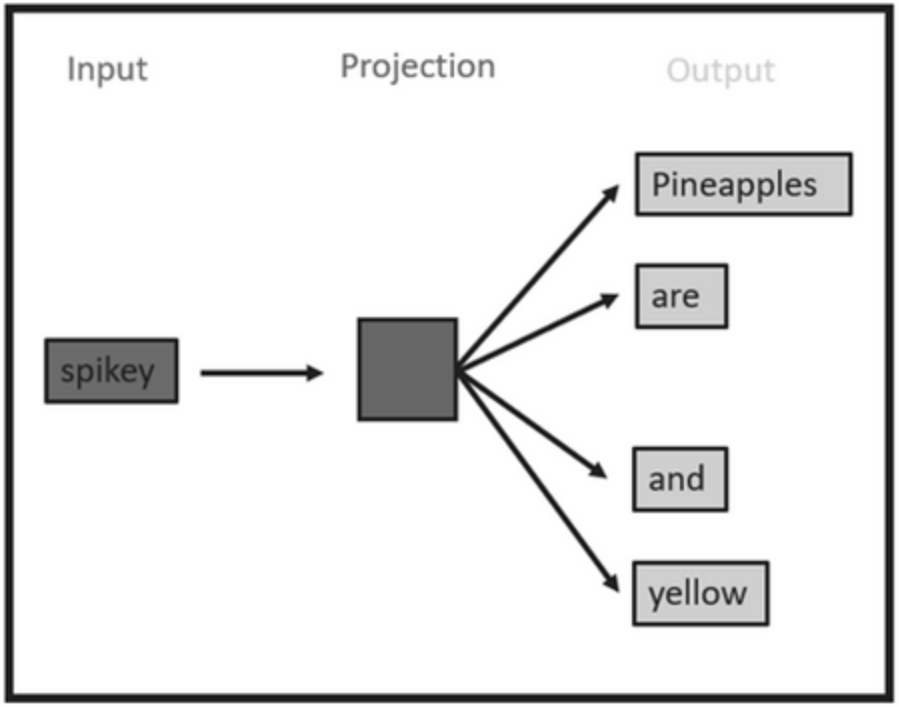
Word2vec is a family of algorithms

Two algorithms to learn word2vec embeddings:

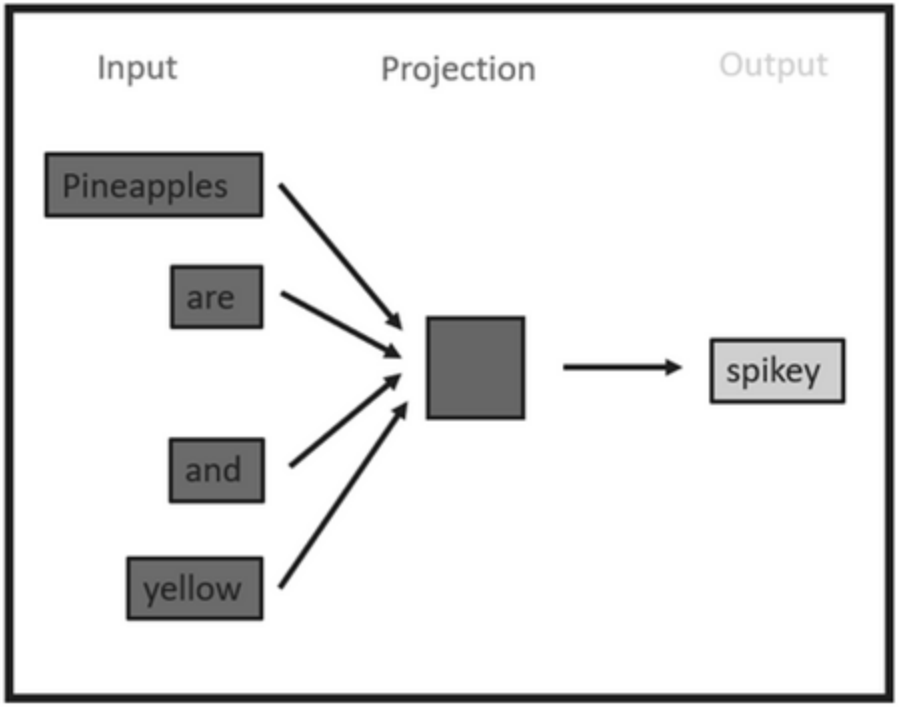
1. Skip-gram (SG): Predict context words given target (position independent)

2. Continuous Bag of Word (CBOW): Predict target word from bag-of-words context

Word2vec is a family of algorithms

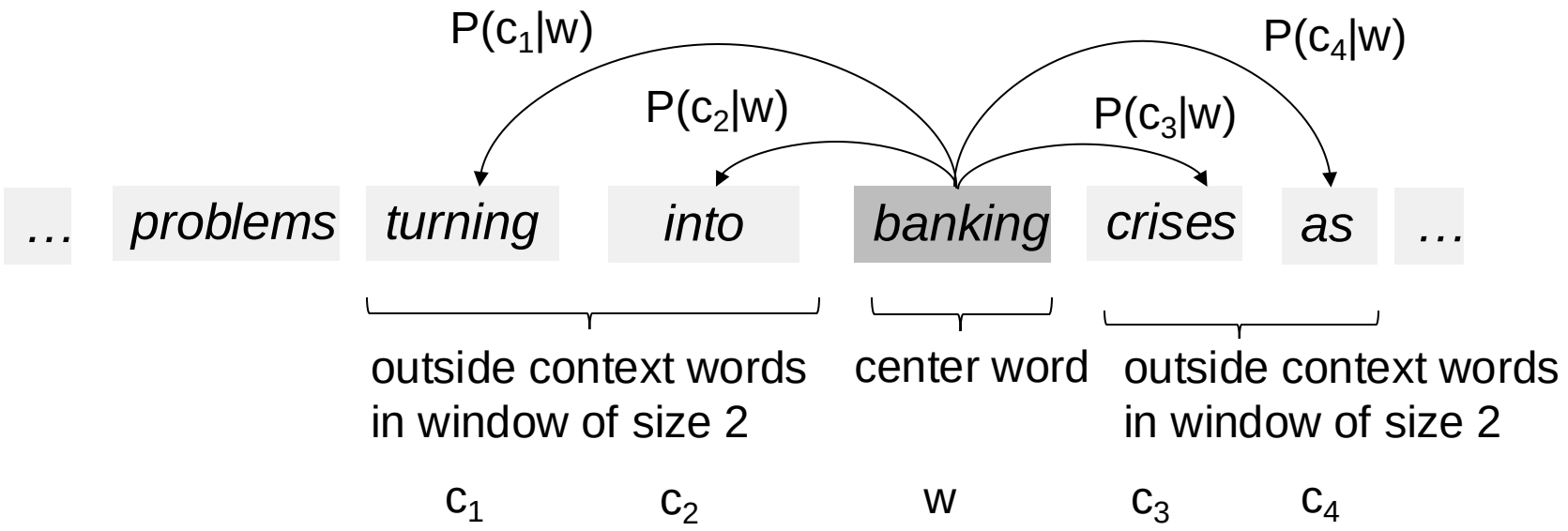


Skip-gram



CBOW

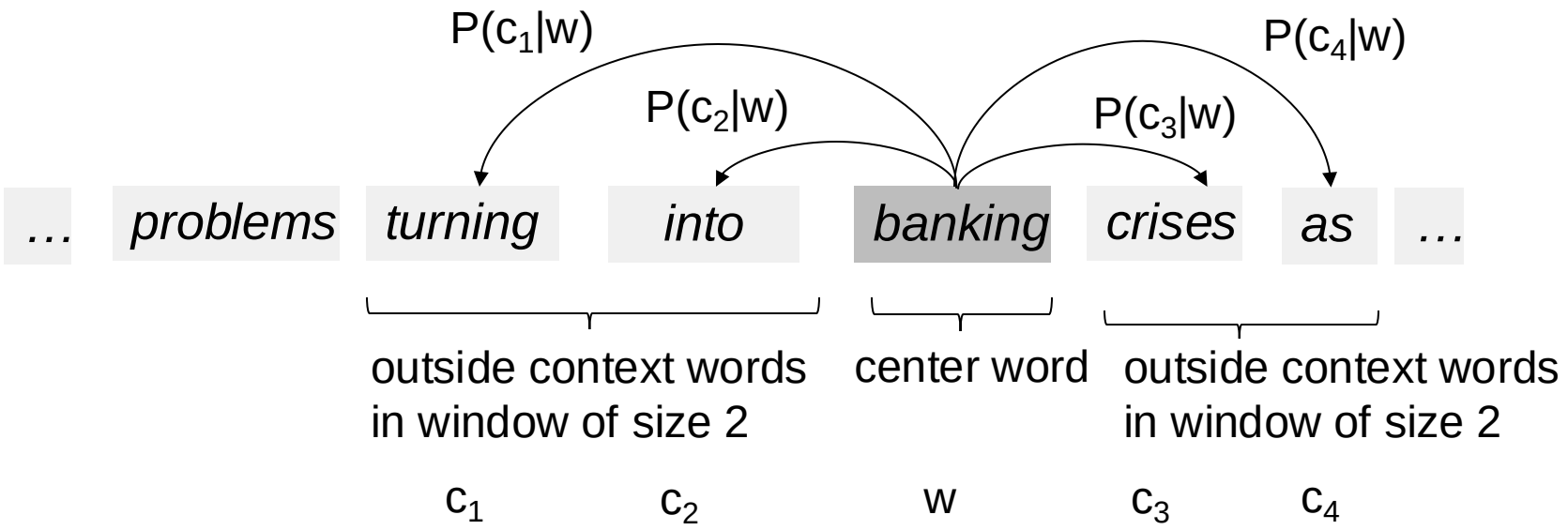
The classifier in skip-gram



Train a classifier so that it returns the probability that *c* is a real context word of *w*: P(+|w,c).

- Treat the target word and a neighbouring context word as positive examples.
- Randomly sample other words in the lexicon to get negative samples.

The classifier in skip-gram



Positive examples

w	$\mathbf{c_{pos}}$
banking	turning
banking	into
banking	crises
banking	as

Negative examples (here: ratio = 2)

w	$\mathbf{c_{neg}}$	***w***	$\mathbf{c_{neg}}$
banking	aardvark	banking	seven
banking	my	banking	forever
banking	where	banking	dear
banking	coaxial	banking	if

The classifier in skip-gram

The probability that word *c* is not a context word is

P(-w|w,c) = 1- P(+|w,c).

How does the classifier compute probability P? Embedding similarity, aka the dot product between two embeddings.

The dot product between c and w is not a probability, therefore use the sigmoid function (the fundamental core of logistic regression), which returns a number between 0 and 1.

The classifier in skip-gram

The probability that word *c* is not a context word is P(-w|w,c) = 1- P(+|w,c).

How does the classifier compute probability P? Embedding similarity, aka the dot product between two embeddings:

$$Similarity(w,c) \approx \mathbf{c} \cdot \mathbf{w}$$

The dot product between c and w is not a probability, therefore use the sigmoid function (the fundamental core of logistic regression), which returns a number between 0 and 1.

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

The classifier in skip-gram

Probability that c is indeed a context word of w:

$$P(+|w,c) = \sigma(\mathbf{c}\cdot\mathbf{w}) = \frac{1}{1+\exp(-\mathbf{c}\cdot\mathbf{w})}$$

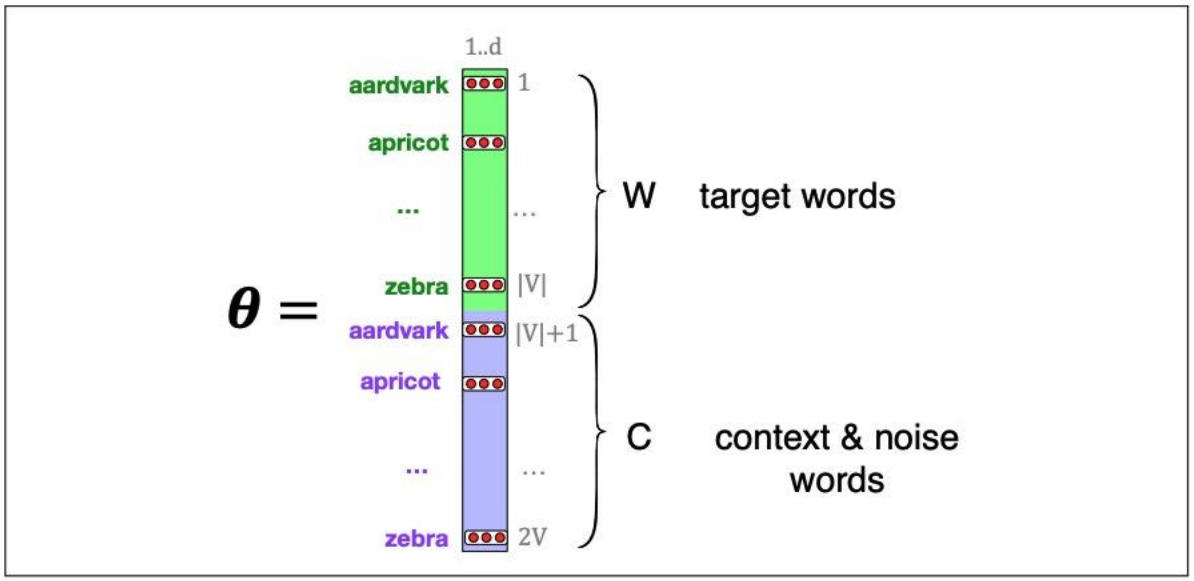
Probability that c is not a context word of w:

$$\begin{aligned} P(-|w,c) &= 1 - P(+|w,c) \\ &= \sigma(-\mathbf{c}\cdot\mathbf{w}) = \frac{1}{1+\exp(\mathbf{c}\cdot\mathbf{w})} \end{aligned}$$

The classifier in skip-gram

Skip-gram assumption: all context words are independent, therefore

$$P(+|w, c_{1:L}) = \prod_{i=1}^{L} \sigma(\mathbf{c_i} \cdot \mathbf{w})$$



Learning skip-gram embeddings

Input:

- a corpus of text
- a chosen vocabulary size N
- a random embedding vector for each of the N vocabulary words

Use positive and negative examples of target/context word occurrences:

w	c_{pos}
banking	turning
banking	into
banking	crises
banking	as

w	c_{neg}
banking	aardvark
banking	my
banking	where
banking	coaxial

w	c_{neg}
banking	seven
banking	forever
banking	dear
banking	if

Learning skip-gram embeddings

Given the set of positive and negative training instances and an initial set of embeddings, the goal of the learning algorithm is to adjust those embeddings in order to

- maximize the similarity of the target word, context word pairs (c, w_{pos}) drawn from the positive examples
- minimize the similarity of the (c, w_{neg}) pairs from the negative examples.

Learning skip-gram embeddings

This aim can be expressed as the following loss function:

$$
\begin{aligned}
L &= -\log\left[P(+|w,c_{pos})\prod_{i=1}^{k}P(-|w,c_{neg_i})\right] \\
&= -\left[\log P(+|w,c_{pos})+\sum_{i=1}^{k}\log P(-|w,c_{neg_i})\right] \\
&= -\left[\log P(+|w,c_{pos})+\sum_{i=1}^{k}\log\left(1-P(+|w,c_{neg_i})\right)\right] \\
&= -\left[\log\sigma(c_{pos}\cdot w)+\sum_{i=1}^{k}\log\sigma(-c_{neg_i}\cdot w)\right]
\end{aligned}
$$

We minimize L using stochastic gradient descent (SGD).

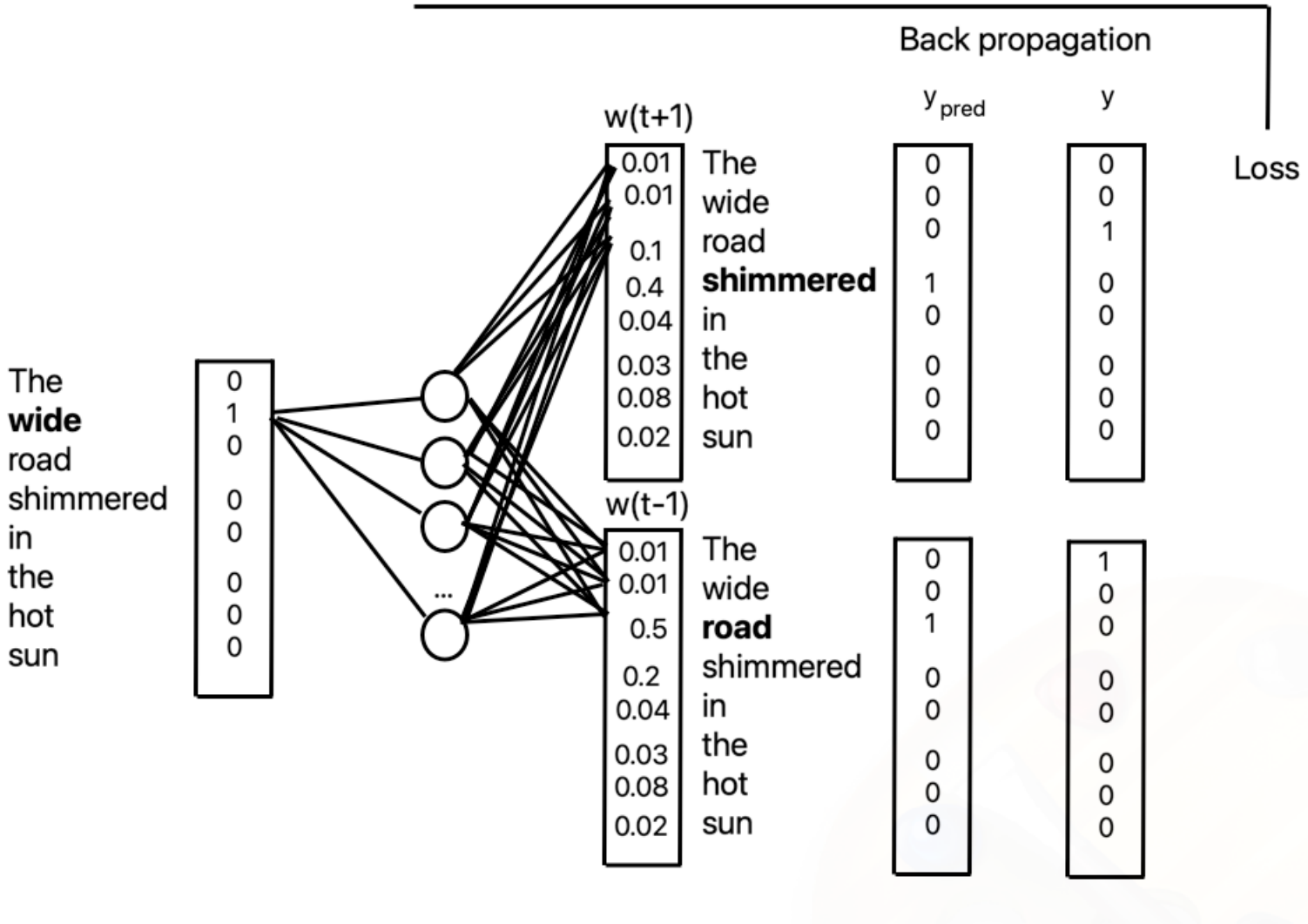
Learning skip-gram embeddings

SGD: the basic idea dates back to the 1950s

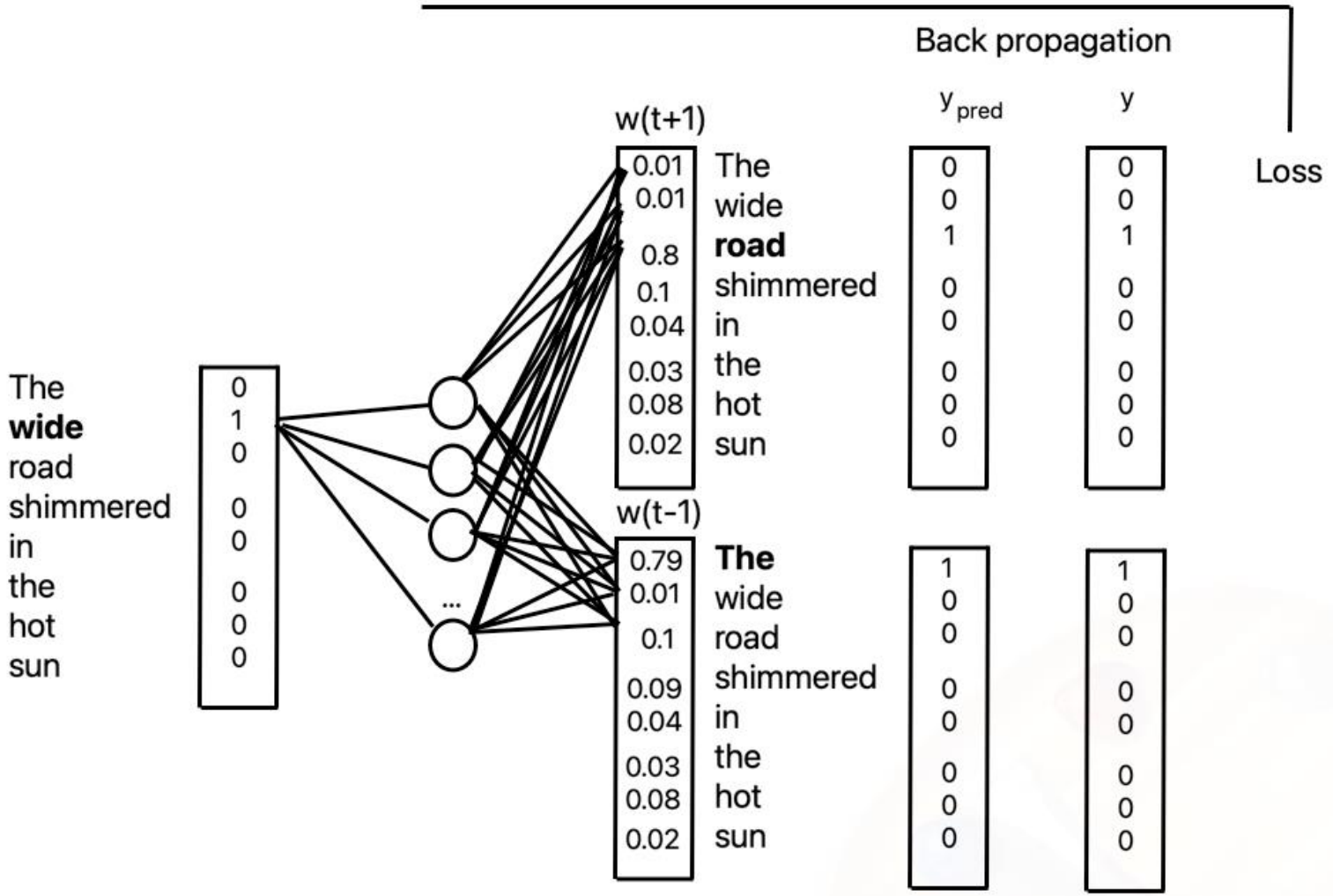
Iterative process for optimizing an objective function with suitable smoothness properties.

Especially useful in high-dimensional scenarios, as it reduces the very high computational burden.

Learning skip-gram embeddings



Learning skip-gram embeddings



Semantic properties of embeddings

Different types of similarity or association possible by varying the size of the context window:

- Shorter context windows: more syntactic representations, the most similar words tend to be semantically similar words with the same parts of speech
- Longer context windows: words that are topically related.

For example, in Levy and Goldberg (2014a): Skip-gram, *w* = 'Hogwarts'

- window of +/- 2, most similar were names of other fictional schools: 'Sunnydale' (Buffy the Vampire Slayer) or 'Evernight' (from a vampire series)
- window of +/- 5, 'Dumbledore', 'Malfoy', 'half-blood'

Trained word embeddings available

word2vec: https://code.google.com/archive/p/word2vec/

GloVe: https://nlp.stanford.edu/projects/glove/

FastText: https://fasttext.cc/

Differ in algorithms, training data, dimensions, cased/uncased...

Sent2vec

Pagliardini et al. 2018

Sentence embedding is the average of the source word embeddings of its constituent words.

Augmented by

- learning source embeddings for not only unigrams but also n-grams of words present in each sentence
- averaging the n-gram embeddings along with the words

Sent2vec

- Used for keyphrase extraction
- Idiom token classification
- Sentence summarization
- Machine translation
- Paraphrasing
- ...

(see papers on https://www.aclanthology.org)

Doc2vec

Also known as ‘paragraph embedding’.

Embeddings for entire documents or paragraphs.

Again, two algorithms:

- Distributed Memory of Paragraph Vector (use word embeddings and a document id token to learn the paragraph embedding, preserves word order in the paragraph)
- Distributed Bag of Words of Paragraph Vector (uses the document id token to predict randomly sampled words)

Application of embeddings in IR

Nalisnick, Mitra, Craswell & Caruana. 2016. Improving Document Ranking with Dual Word Embeddings. *WWW 2016 Companion.* http://research.microsoft.com/pubs/260867/pp1291-Nalisnick.pdf

Mitra, Nalisnick, Craswell & Caruana. 2016. A Dual Embedding Space Model for Document Ranking. arXiv:1602.01137 [cs.IR]

Builds on BM25 model idea of “aboutness”

- Not just term repetition indicating aboutness
- Relationship between query terms and *all* terms in the document indicates aboutness (BM25 uses only query terms)

Application of embeddings in IR

Many papers on using doc2vec to model

- document aboutness
- query-document similarity

Lots of applications where knowing word context or similarity helps prediction:

- Synonym handling in search
- Ad serving
- …

What else can neural nets do in IR?

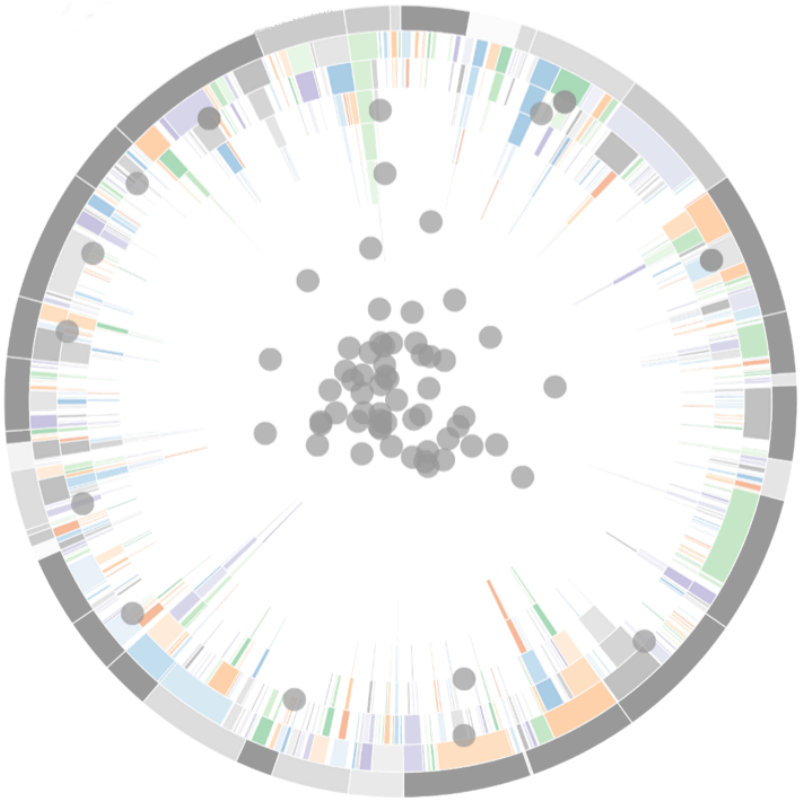
- Use a neural network as a supervised reranker.
- Use query and document embedding for ranking.
- Assume you have (q, d, rel) data, learn a neural network to predict relevance of the (q, d) pair.

Cluster of Excellence
The Politics of Inequality

Steinbeis Transfer Center
Linguistic Data Analysis

UNIVERSITÄT PASSAU

Thank you.
Questions?
Comments?



Annette Hautli-Janisz, Prof. Dr.
cornlp-teaching@uni-passau.de