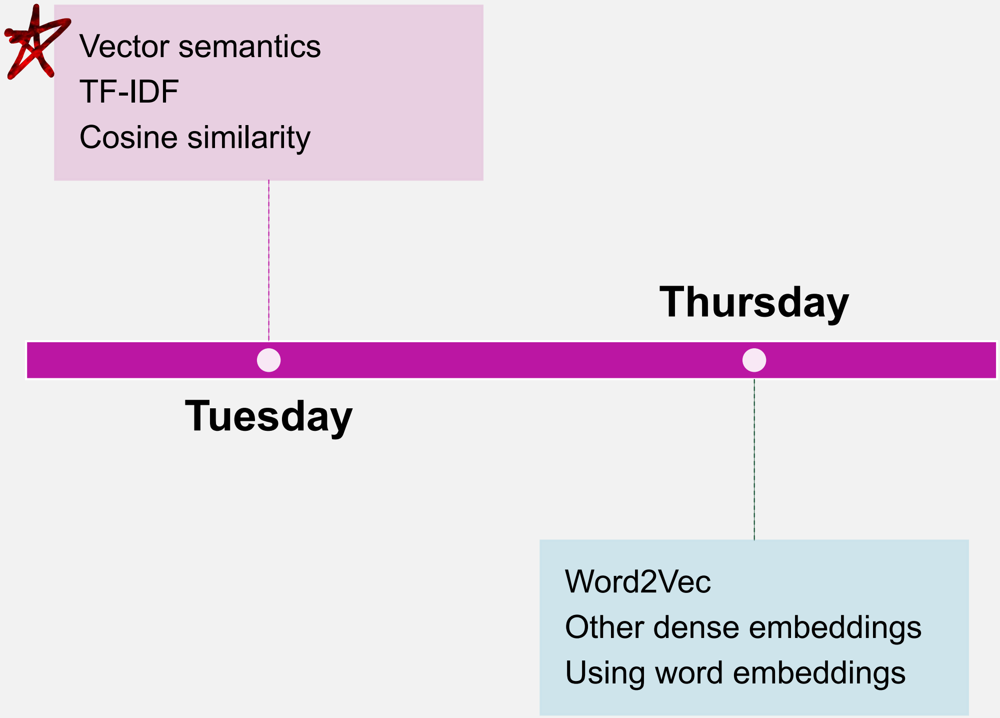
Vector Semantics

Natalie Parde

UIC CS 421

This Week's Topics



Vector semantics
TF-IDF
Cosine similarity

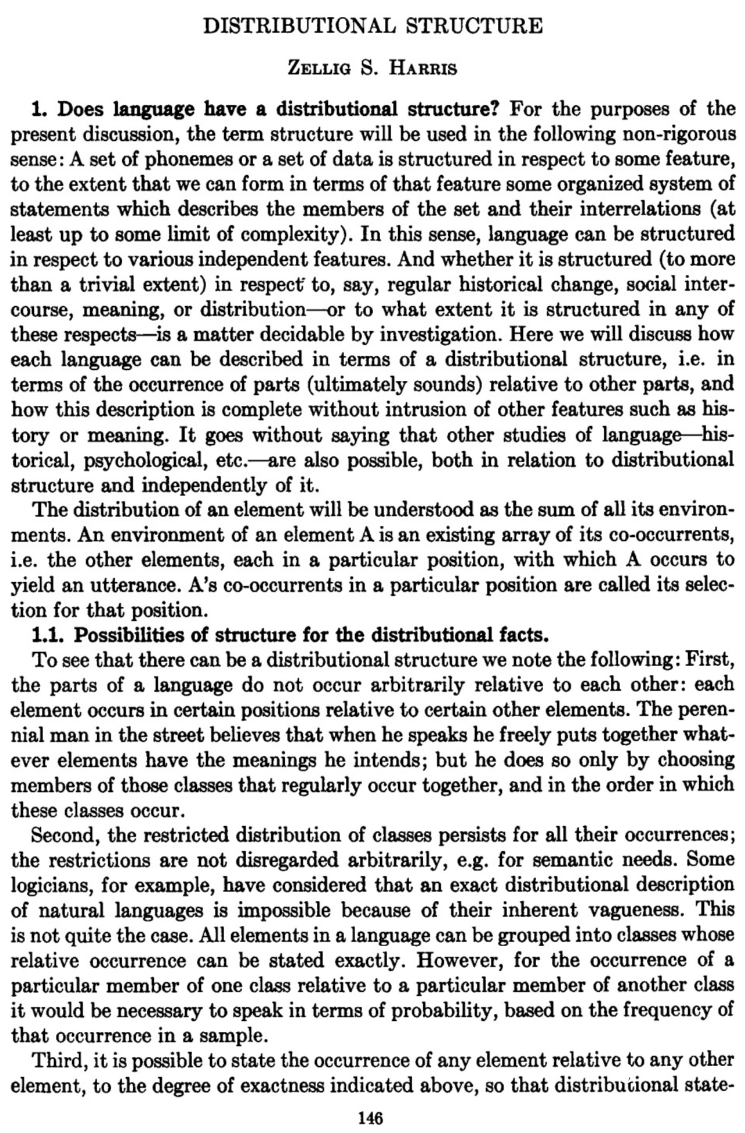
Tuesday

Thursday

Word2Vec
Other dense embeddings
Using word embeddings

Vector Semantics

- Facilitates a form of **representation learning** based on the notion that similar words tend to occur in similar environments
    - This notion is known as the distributional hypothesis, which was first formulated by linguists in the 1950s
        - Joos (1950)
        - Harris (1954)
        - Firth (1957)
- Self-supervised



DISTRIBUTIONAL STRUCTURE

Zellig S. Harris

1. Does language have a distributional structure? For the purposes of the present discussion, the term structure will be used in the following non-rigorous sense: A set of phonemes or a set of data is structured in respect to some feature, to the extent that we can form in terms of that feature some organized system of statements which describes the members of the set and their interrelations (at least up to some limit of complexity). In this sense, language can be structured in respect to various independent features. And whether it is structured (to more than a trivial extent) in respect to, say, regular historical change, social intercourse, meaning, or distribution—or to what extent it is structured in any of these respects—is a matter decidable by investigation. Here we will discuss how each language can be described in terms of a distributional structure, i.e. in terms of the occurrence of parts (ultimately sounds) relative to other parts, and how this description is complete without intrusion of other features such as history or meaning. It goes without saying that other studies of language—historical, psychological, etc.—are also possible, both in relation to distributional structure and independently of it.

The distribution of an element will be understood as the sum of all its environments. An environment of an element A is an existing array of its co-occurrents, i.e. the other elements, each in a particular position, with which A occurs to yield an utterance. A's co-occurrents in a particular position are called its selection for that position.

1.1. Possibilities of structure for the distributional facts.

To see that there can be a distributional structure we note the following: First, the parts of a language do not occur arbitrarily relative to each other: each element occurs in certain positions relative to certain other elements. The perennial man in the street believes that when he speaks he freely puts together whatever elements have the meanings he intends; but he does so only by choosing members of those classes that regularly occur together, and in the order in which these classes occur.

Second, the restricted distribution of classes persists for all their occurrences; the restrictions are not disregarded arbitrarily, e.g. for semantic needs. Some logicians, for example, have considered that an exact distributional description of natural languages is impossible because of their inherent vagueness. This is not quite the case. All elements in a language can be grouped into classes whose relative occurrence can be stated exactly. However, for the occurrence of a particular member of one class relative to a particular member of another class it would be necessary to speak in terms of probability, based on the frequency of that occurrence in a sample.

Third, it is possible to state the occurrence of any element relative to any other element, to the degree of exactness indicated above, so that distributional state-

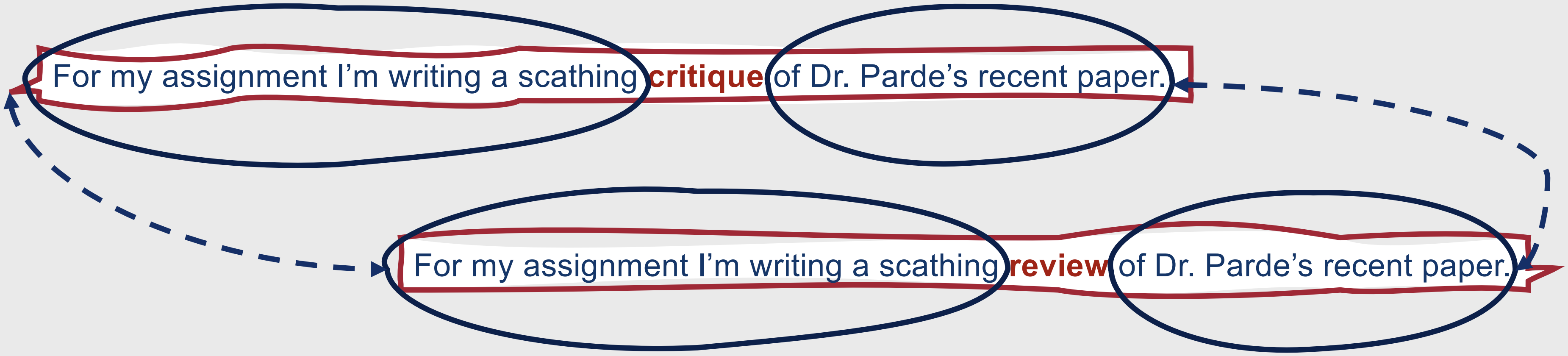
146

Vector Semantics

For my assignment I’m writing a scathing **critique** of Dr. Parde’s recent paper.

For my assignment I’m writing a scathing **review** of Dr. Parde’s recent paper.

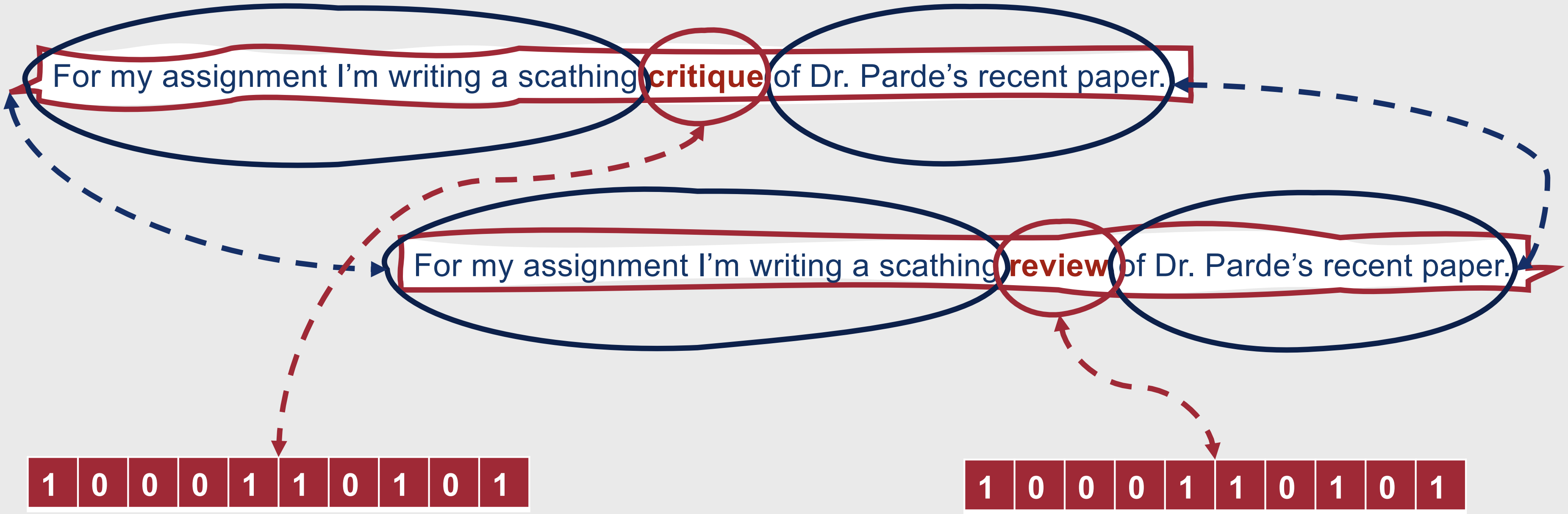
Vector Semantics



For my assignment I'm writing a scathing **critique** of Dr. Parde's recent paper.

For my assignment I'm writing a scathing **review** of Dr. Parde's recent paper.

Vector Semantics



For my assignment I'm writing a scathing **critique** of Dr. Parde's recent paper.

For my assignment I'm writing a scathing **review** of Dr. Parde's recent paper.

1	0	0	0	1	1	0	1	0	1

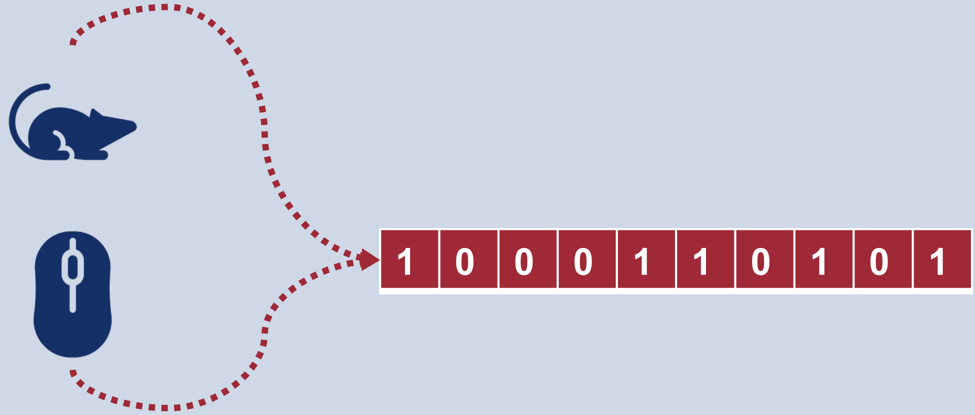
1	0	0	0	1	1	0	1	0	1

There are many ways to make use of the distributional hypothesis!

- **Classical word vectors**
  - Bag of words representations and their variations
- **Implicitly learned word vectors**
  - Word2Vec
  - GloVe
- All of these approaches seek to encode the same linguistic phenomena observed in studies of lexical semantics

Lemmas and Senses

- **Lemma:** The base form of a word
  - Papers → paper
  - Mice → mouse
- **Word Sense:** Different aspects of meaning for a word
  - Mouse (1): A small rodent
  - Mouse (2): A device to control a computer cursor
- Words with the same lemma should (hopefully!) reside near one another in vector space
- Words with the same sense might also reside near one another in vector space, depending on the representation learning technique



1	0	0	0	1	1	0	1	0	1

Synonymy

- When a word sense for one word is (nearly) identical to the word sense for another word
- **Synonymy:** Two words are synonymous if they are substitutable for one another in any sentence without changing the situations in which the sentence would be true
  - This means that the words have the same **propositional meaning**

For my assignment I’m writing a scathing **critique** of Dr. Parde’s recent paper.

For my assignment I’m writing a scathing **review** of Dr. Parde’s recent paper.

Word Similarity and Relatedness

- **Word similarity:** Words are not synonyms, but they can be used in the same contexts as one another
- **Word Relatedness:** Words are associated with one another based on their shared participation in an event

Natalie grabbed the **purple** coffee mug.

Natalie grabbed the **green** coffee mug.

coffee

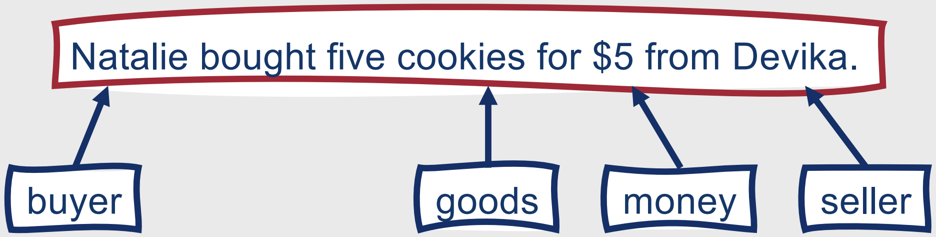
cup

espresso

cafe

Semantic Frames

- **Semantic Frame:** A set of words that denote perspectives or participants in a particular type of event
  - Commercial Transaction = {buyer, seller, goods, money}
- **Semantic Role:** A participant's underlying role with respect to the main verb in the sentence


Natalie bought five cookies for $5 from Devika.

Connotation

- Also referred to as **affective meaning**
- The aspects of a word’s meaning that are related emotions, sentiment, opinions, or evaluations
  - **Valence:** Positivity
    - High: Happy, satisfied
    - Low: Unhappy, annoyed
  - **Arousal:** Intensity of emotion
    - High: Excited, frenzied
    - Low: Relaxed, calm
  - **Dominance:** Degree of control
    - High: Important, controlling
    - Low: Awed, influenced

	Valence	Arousal	Dominance
courageous	8.05	5.5	7.38
music	7.67	5.57	6.5
heartbreak	2.45	5.65	3.58
cub	6.71	3.95	4.24
life	6.68	5.59	5.89

Word vector! (Osgood et al., 1957)

How, then, should we represent meaning?

- Many, many approaches!
- Two classic strategies:
  - **Bag of words representations:** A word is a string of letters, or an index in a vocabulary list
  - **Logical representation:** A word defines its own meaning (“dog” = DOG)

How, then, should we represent meaning?

- Many, many approaches!
- Two classic strategies:
  - **Bag of words representations:** A word is a string of letters, or an index in a vocabulary list
  - ~~**Logical representation:** A word defines its own meaning ("dog" = DOG)~~

Bag of words features implement a simple form of vector semantics.

- **Two words with very similar sets of contexts (i.e., similar distributions of neighboring words) are assumed to have very similar meanings**
- We represent this context using vectors
- For bag of words:
  - Define a word as a single vector point in an *n*-dimensional space, where *n* = vocabulary size
  - The value stored in a dimension corresponds to the presence of a context word in the same sample as the target word

Example: Context-Based BOW Vector

To earn a Bachelor of Science in Computer Science degree from UIC, students need to complete university, college, and department degree requirements. The Department of Computer Science **degree** requirements are outlined below. Students should consult the College of Engineering section for additional degree requirements and college academic policies.

Context Window = 4

Example: Context-Based BOW Vector

To earn a Bachelor of Science in Computer Science degree from UIC, students need to complete university, college, and department degree requirements. The Department of Computer Science **degree** requirements are outlined below. Students should consult the College of Engineering section for additional degree requirements and college academic policies.

Context Window = 4

bachelor	of	science	in	computer	department	requirements	are	…	students	need	complete	outlined	college	below	consult	engineering
0	1	1	0	1	1	1	1	…	0	0	0	1	0	1	0	0

Example: Context-Based BOW Vector

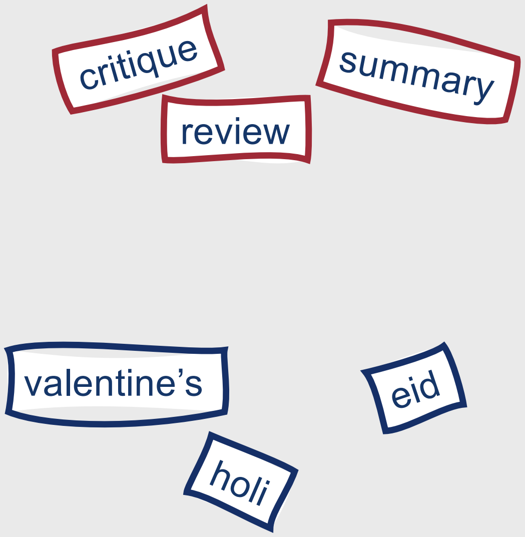
To earn a Bachelor of Science in Computer Science degree from UIC, students need to complete university, college, and department degree requirements. The Department of Computer Science **degree** requirements are outlined below. Students should consult the College of Engineering section for additional degree requirements and college academic policies.

Context Window = 8

bachelor	of	science	in	computer	department	requirements	are	…	students	need	complete	outlined	college	below	consult	engineering
0	1	1	0	1	2	2	1	…	1	0	0	1	0	1	1	0

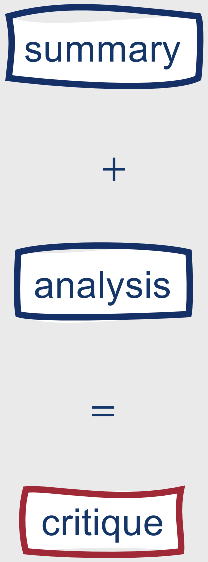
The goal is for the values in these vector representations to correspond with dimensions of meaning.

- Assuming this is the case, we should be able to:
  - Cluster vectors into semantic groups
  - Perform operations that are semantically intuitive



The goal is for the values in these vector representations to correspond with dimensions of meaning.

- Assuming this is the case, we should be able to:
  - Cluster vectors into semantic groups
  - Perform operations that are semantically intuitive



summary

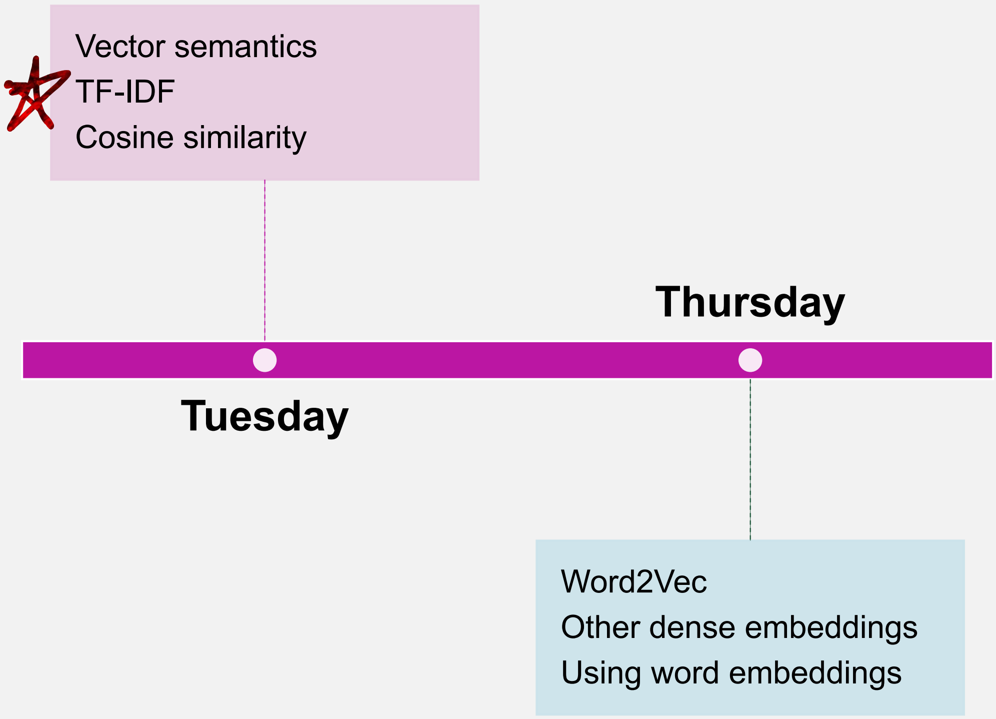
+

analysis

=

critique

This Week's Topics



Vector semantics

☆ TF-IDF

Cosine similarity

Word2Vec

Other dense embeddings

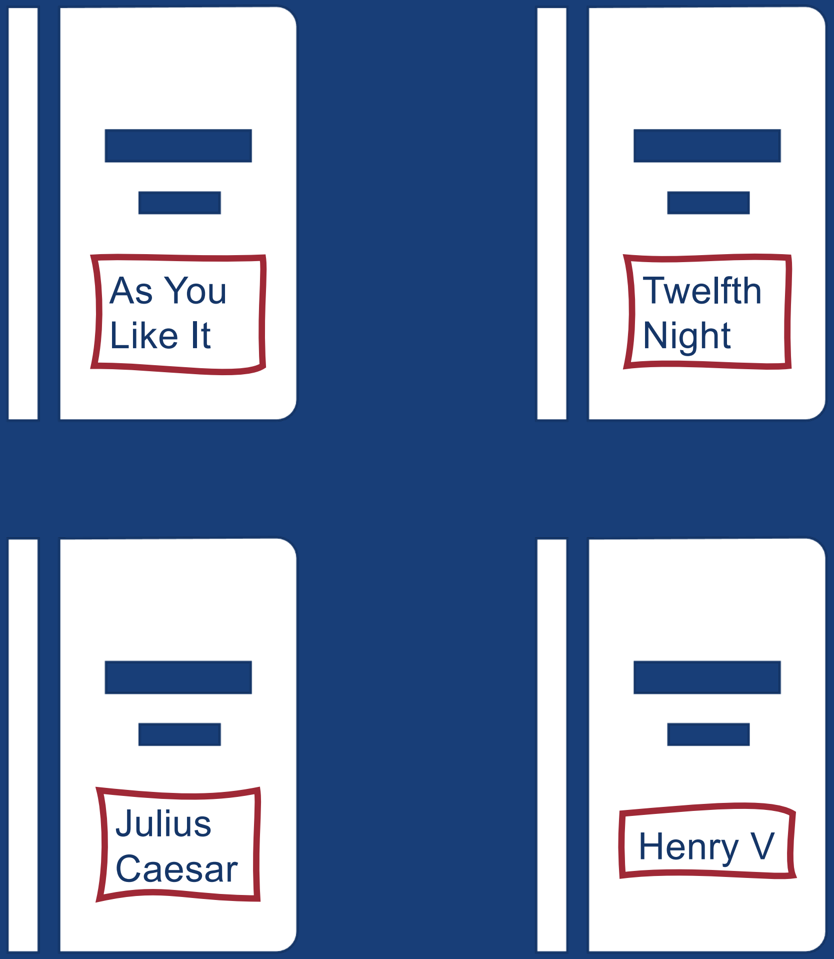
Using word embeddings

Are there alternatives to using frequency in bag-of-words representations?

- TF-IDF:
  - Term Frequency * Inverse Document Frequency
  - Meaning of a word is defined by the counts of words in the *same* document, as well as *overall*

TF-IDF originated as a tool for information retrieval.

- Rows: Words in a vocabulary
- Columns: Documents in a corpus

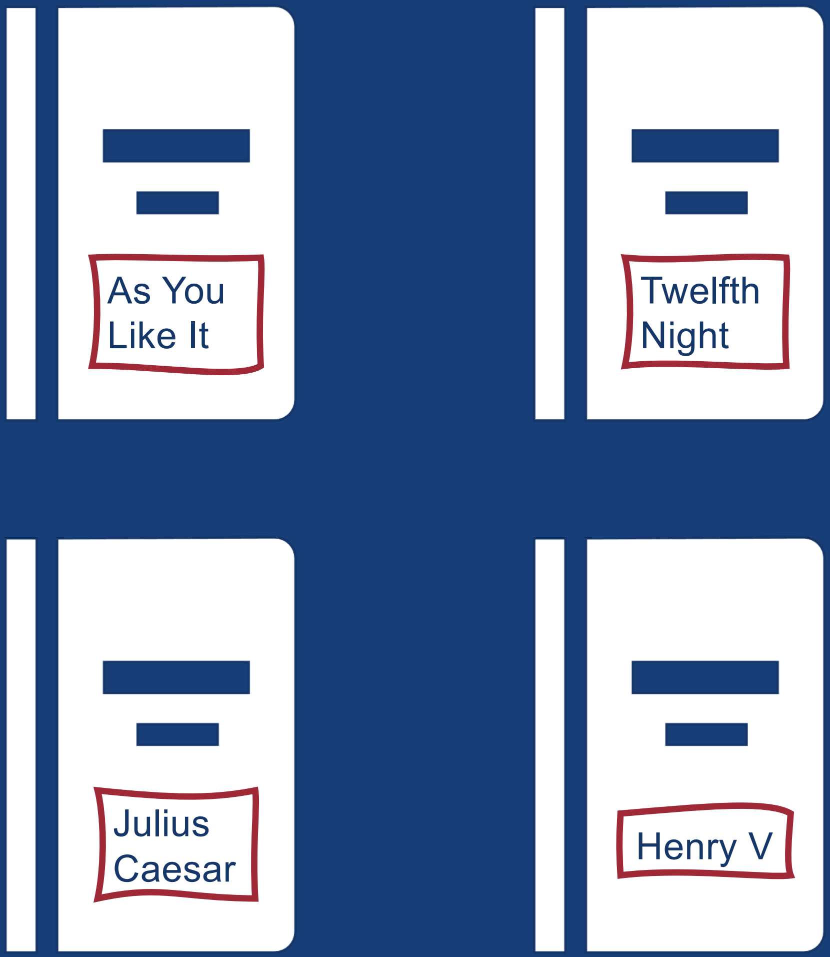


TF-IDF originated as a tool for information retrieval.

- Rows: Words in a vocabulary
- Columns: Documents in a corpus

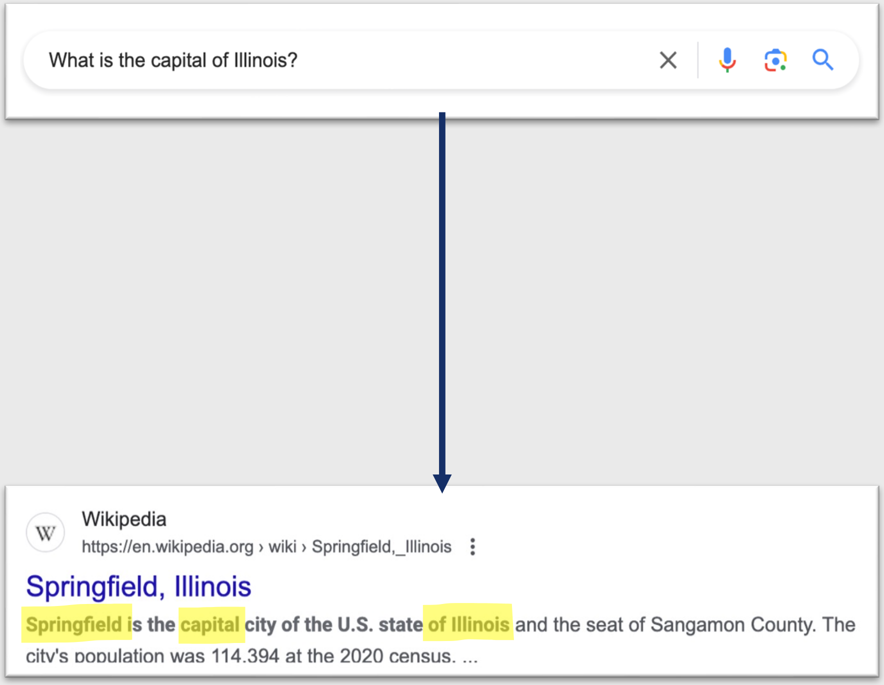
	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	1	0	7	13
good	114	80	62	89
fool	36	58	1	4
wit	20	15	2	3

"wit" appears 3 times in Henry V



Why information retrieval?

- A common goal in web search is to find documents that are similar to the search query
- These documents are likely to have information that is of interest to the person performing the search



What is the capital of Illinois?

Wikipedia
https://en.wikipedia.org › wiki › Springfield,_Illinois

Springfield, Illinois

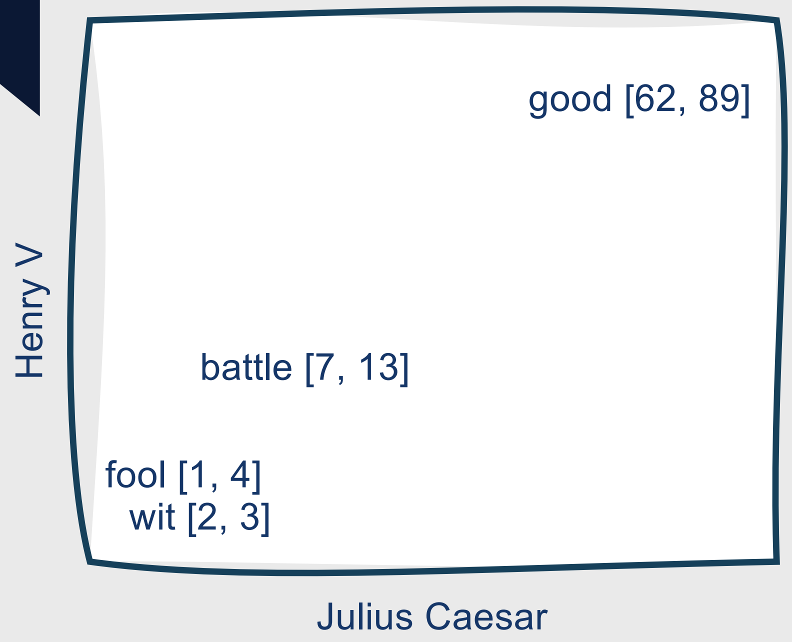
Springfield is the capital city of the U.S. state of Illinois and the seat of Sangamon County. The city's population was 114,394 at the 2020 census. ...

In a term-document matrix, rows can be viewed as word vectors.

- Each dimension corresponds to a document
- Words with **similar vectors** occur in **similar documents**
- This is one way to define **term frequency** vectors

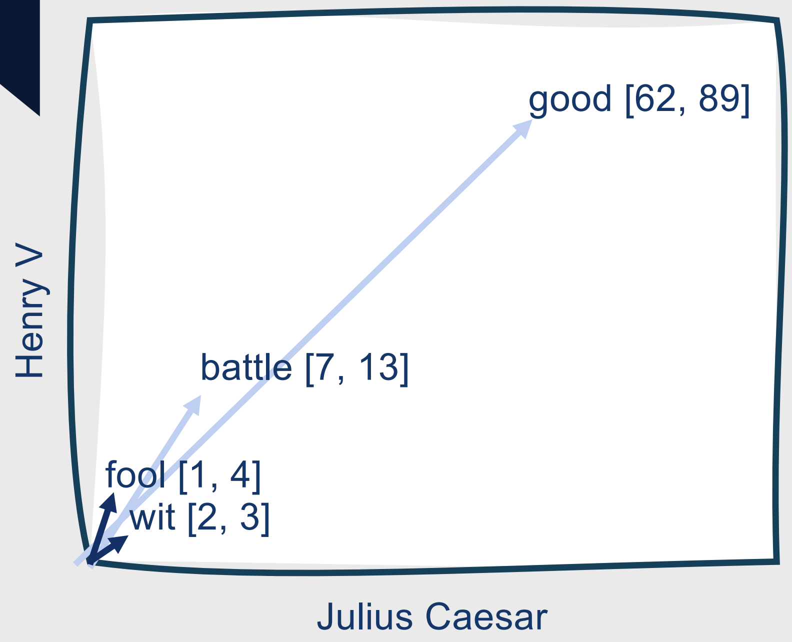
	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	1	0	7	13
good	114	80	62	89
fool	36	58	1	4
wit	20	15	2	3

In a term-document matrix, rows can be viewed as word vectors.



	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

In a term-document matrix, rows can be viewed as word vectors.



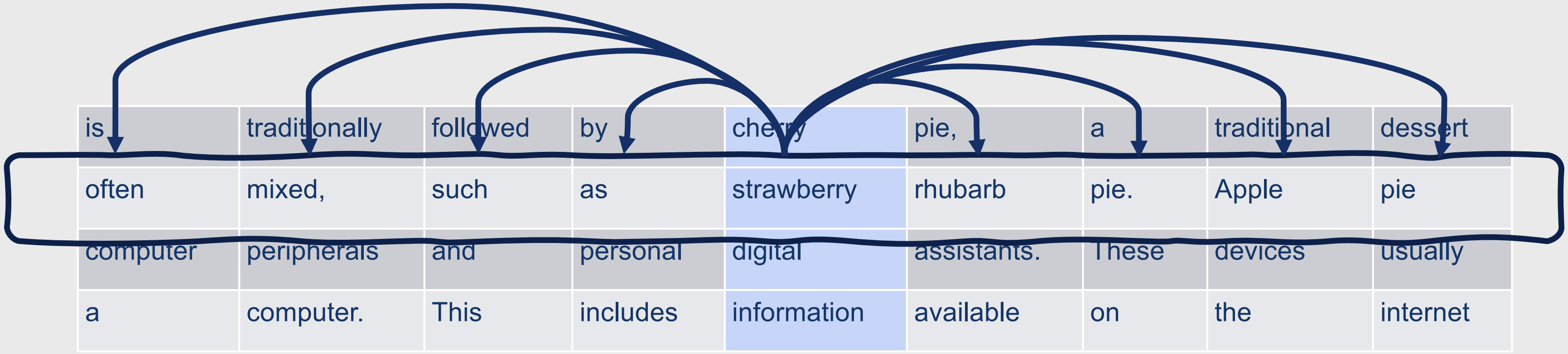
	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	1	0	7	13
good	114	80	62	89
fool	36	58	1	4
wit	20	15	2	3

Different Types of Context

- If we use a defined **word context window** for vector representations, the columns are also labeled by words
  - Each cell records the number of times the row (target) word and the column (context) word co-occur in some context in a training corpus
  - This is just like what we saw in the earlier context-based bag-of-words example!

Another Example Context Window (Size = 4)

- Take each occurrence of a word (e.g., strawberry)
- Count the context words in the four-word spans before and after it to get a word-word co-occurrence matrix



is	traditionally	followed	by	cherry	pie,	a	traditional	dessert
often	mixed,	such	as	strawberry	rhubarb	pie.	Apple	pie
computer	peripherals	and	personal	digital	assistants.	These	devices	usually
a	computer.	This	includes	information	available	on	the	internet

Where does TF-IDF enter the picture?

- Some words co-occur frequently with many words, so won't be very informative
  - *the*, *it*, *they*
- We want to know about **words that co-occur frequently with one another, but less frequently across all texts**

TF-IDF

- **Term Frequency:** The frequency of the word *t* in the document *d*
  - $tf_{t,d} = \text{count}(t, d)$
- **Document Frequency:** The number of documents in which the word *t* occurs

Computing TF-IDF

- **Inverse Document Frequency:** The inverse of document frequency, where *N* is the total number of documents in the collection
  - $idf_t = \frac{N}{df_t}$
- IDF is higher when the term occurs in fewer documents
  - Document = Whatever is considered an instance or context in your dataset
- It is often useful to perform these computations in log space
  - TF: $\log_{10}(tf_{t,d}+1)$ → Make sure to smooth so you don't try to take the log of 0!
  - IDF: $\log_{10} idf_t$

Computing TF*IDF

- TF-IDF combines TF and IDF
  - $tfidf_{t,d} = tf_{t,d} \times idf_t$

Assume we're looking at a subset of a 37-document corpus of Shakespearean plays….

Example: Computing TF-IDF

- TF-IDF(battle, d_1) = ?

	d_1	d_2	d_3	d_4
battle	1	0	7	13
good	114	80	62	89
fool	36	58	1	4
wit	20	15	2	3

Example: Computing TF-IDF

- TF-IDF(battle, d_1) = ?
- **TF(battle, d_1) = 1**

	d_1	d_2	d_3	d_4
battle	1	0	7	13
good	114	80	62	89
fool	36	58	1	4
wit	20	15	2	3

Example: Computing TF-IDF

- TF-IDF(battle, d_1) = ?
- TF(battle, d_1) = 1
- **IDF(battle) = N/DF(battle) = 37/21 = 1.76**

	d_1	d_2	d_3	d_4
battle	1	0	7	13
good	114	80	62	89
fool	36	58		
wit	20	15		

word	df
battle	21
good	37
fool	3
wit	34

Overall document frequencies from our 37 plays

Example: Computing TF-IDF

- TF-IDF(battle, d_1) = ?
- TF(battle, d_1) = 1
- IDF(battle) = N/DF(battle) = 37/21 = 1.76
- **TF-IDF(battle, d_1) = 1 * 1.76 = 1.76**

	d_1	d_2	d_3	d_4
battle	1	0	7	13
good	114	80	62	89
fool	36	58	1	4
wit	20	15	2	3

Example: Computing TF-IDF

- TF-IDF(battle, d_1) = ?
- TF(battle, d_1) = 1
- IDF(battle) = N/DF(battle) = 37/21 = 1.76
- TF-IDF(battle, d_1) = 1 * 1.76 = 1.76
- **Alternately, TF-IDF(battle, d_1) = $log_{10}(1+1) * log_{10} 1.76 =$ 0.074**

	d_1	d_2	d_3	d_4
battle	1	0	7	13
good	114	80	62	89
fool	36	58	1	4
wit	20	15	2	3

Example: Computing TF-IDF

- TF-IDF(battle, d_1) = ?
- TF(battle, d_1) = 1
- IDF(battle) = N/DF(battle) = 37/21 = 1.76
- TF-IDF(battle, d_1) = 1 * 1.76 = 1.76
- Alternately, TF-IDF(battle, d_1) = $log_{10}(1+1) * \ log_{10} 1.76 = 0.074$

	d_1	d_2	d_3	d_4
battle	0.074	0	7	13
good	114	80	62	89
fool	36	58	1	4
wit	20	15	2	3

To convert our entire term frequency matrix to a TF-IDF matrix, we need to repeat this calculation for each element.

	d_1	d_2	d_3	d_4
battle	0.074	0.000	0.220	0.280
good	0.000	0.000	0.000	0.000
fool	0.019	0.021	0.004	0.008
wit	0.049	0.044	0.018	0.022

How does the TF-IDF matrix compare to the original term frequency matrix?

	d_1	d_2	d_3	d_4
battle	1	0	7	13
good	114	80	62	89
fool	36	58	1	4
wit	20	15	2	3

	d_1	d_2	d_3	d_4
battle	0.074	0.000	0.220	0.280
good	0.000	0.000	0.000	0.000
fool	0.019	0.021	0.004	0.008
wit	0.049	0.044	0.018	0.022

How does the TF-IDF matrix compare to the original term frequency matrix?

	d_1	d_2	d_3	d_4
battle	1	0	7	13
good	114	80	62	89
fool	36	58	1	4
wit	20	15	2	3

	d_1	d_2	d_3	d_4
battle	0.074	0.000	0.220	0.280
good	0.000	0.000	0.000	0.000
fool	0.019	0.021	0.004	0.008
wit	0.049	0.044	0.018	0.022

Occurs in every document …not important in the overall scheme of things!

How does the TF-IDF matrix compare to the original term frequency matrix?

	d_1	d_2	d_3	d_4
battle	1	0	7	13
good	114	80	62	89
fool	36	58	1	4
wit	20	15	2	3

	d_1	d_2	d_3	d_4
battle	0.074	0.000	0.220	0.280
good	0.000	0.000	0.000	0.000
fool	0.019	0.021	0.004	0.008
wit	0.049	0.044	0.018	0.022

Increases the importance of seeing rarer words like “battle”

Note that TF-IDF vectors are sparse.

- Many (usually most) cells have values of 0
- This can make learning difficult

	d_1	d_2	d_3	d_4	d_5	d_6	d_7
battle	0.1	0.0	0.0	0.0	0.2	0.0	0.3
good	0.0	0.0	0.0	0.0	0.0	0.0	0.0
fool	0.0	0.0	0.0	0.0	0.0	0.0	0.0
wit	0.0	0.0	0.0	0.0	0.0	0.0	0.0

We'll learn about other word representation techniques soon!

- However, TF-IDF remains a useful starting point for vector space models
- TF-IDF vectors are generally used with feature-based machine learning algorithms
  - Logistic Regression
  - Naïve Bayes

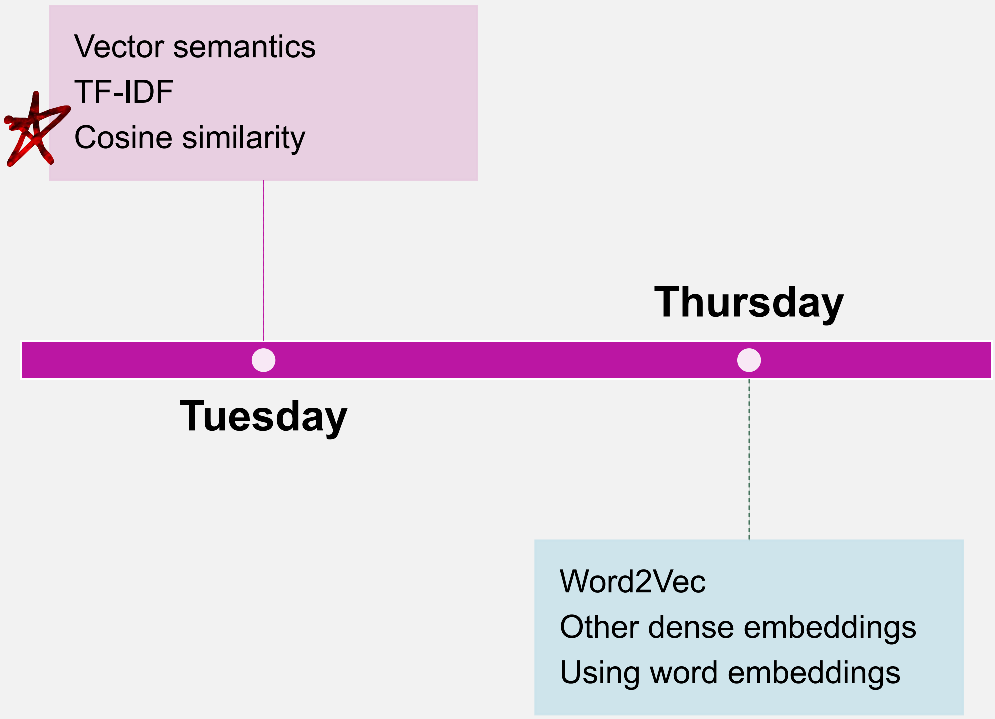
What we know so far….

- **Word vectors:** Vectors of numbers used to encode language
  - Each vector represents a point in an n-dimensional semantic space
- Simple techniques to create word vectors:
  - Co-occurrence frequency (**bag of words**)
  - **TF-IDF**

1	0	0	1	0	1	1	0	0	1

0.7	0	0	0	0	0.9	0.1	0	0	0.5

This Week's Topics



Tuesday

Vector semantics

TF-IDF

Cosine similarity

Thursday

Word2Vec

Other dense embeddings

Using word embeddings

We can use these vectors to measure semantic similarity between words.

- Popular Approach: **Cosine similarity**
  - Based on the **dot product** from linear algebra
    - $\mathbf{v} \cdot \mathbf{w} = \sum_{i=1}^{N} v_i w_i = v_1 w_1 + v_2 w_2 + \cdots + v_N w_N$
- Intuition:
  - Similar vectors (e.g., large values in the same dimensions) will have high cosine similarity
  - Dissimilar vectors (e.g., zeros or low values in different dimensions) will have low cosine similarity

Natalie Parde - UIC CS 421

Normalized Dot Product = Cosine Similarity

- We compute a **normalized dot product** to avoid issues related to word frequency
  - Non-normalized dot product will be higher for frequent words, regardless of how similar they are
  - We normalize by dividing the dot product by the lengths of the two vectors
- The cosine similarity metrics between two vectors **v** and **w** can thus be computed as:
  - $\text{cosine}(\mathbf{v},\mathbf{w}) = \frac{\mathbf{v}\cdot\mathbf{w}}{|\mathbf{v}||\mathbf{w}|} = \frac{\sum_{i=1}^{N} v_i w_i}{\sqrt{\sum_{i=1}^{N} v_i^2}\sqrt{\sum_{i=1}^{N} w_i^2}}$
- This value ranges between:
  - 0 (dissimilar) and 1 (similar) for frequency or TF-IDF vectors
  - -1 (dissimilar) and 1 (similar) for embedding vectors that may have negative values

Example: Computing Cosine Similarity

	glitter	data	computer
unicorn	442	8	2
digital	5	1683	1670
information	5	3982	3325

cos(unicorn, information) = ?

Example: Computing Cosine Similarity

	glitter	data	computer
unicorn	442	8	2
digital	5	1683	1670
information	5	3982	3325

$$\text{cos(unicorn, information)} = \frac{[442,8,2]\cdot[5,3982,3325]}{\sqrt{442^2+8^2+2^2}\sqrt{5^2+3982^2+3325^2}}$$

Example: Computing Cosine Similarity

	glitter	data	computer
unicorn	442	8	2
digital	5	1683	1670
information	5	3982	3325

$$\text{cos(unicorn, information)} = \frac{442*5+8*3982+2*3325}{\sqrt{442^2+8^2+2^2}\sqrt{5^2+3982^2+3325^2}}$$

Example: Computing Cosine Similarity

	glitter	data	computer
unicorn	442	8	2
digital	5	1683	1670
information	5	3982	3325

$$\boxed{\text{cos(unicorn, information)} = \frac{442*5+8*3982+2*3325}{\sqrt{442^2+8^2+2^2}\sqrt{5^2+3982^2+3325^2}} = 0.017}$$

Example: Computing Cosine Similarity

	glitter	data	computer
unicorn	442	8	2
digital	5	1683	1670
information	5	3982	3325

$$\cos(\text{unicorn, information}) = \frac{442*5+8*3982+2*3325}{\sqrt{442^2+8^2+2^2}\sqrt{5^2+3982^2+3325^2}} = 0.017$$

$$\cos(\text{digital, information}) = \frac{5*5+1683*3982+1670*3325}{\sqrt{5^2+1683^2+1670^2}\sqrt{5^2+3982^2+3325^2}} = 0.996$$

Example: Computing Cosine Similarity

	glitter	data	computer
unicorn	442	8	2
digital	5	1683	1670
information	5	3982	3325

$$\text{cos(unicorn, information)} = \frac{442*5+8*3982+2*3325}{\sqrt{442^2+8^2+2^2}\sqrt{5^2+3982^2+3325^2}} = 0.017$$

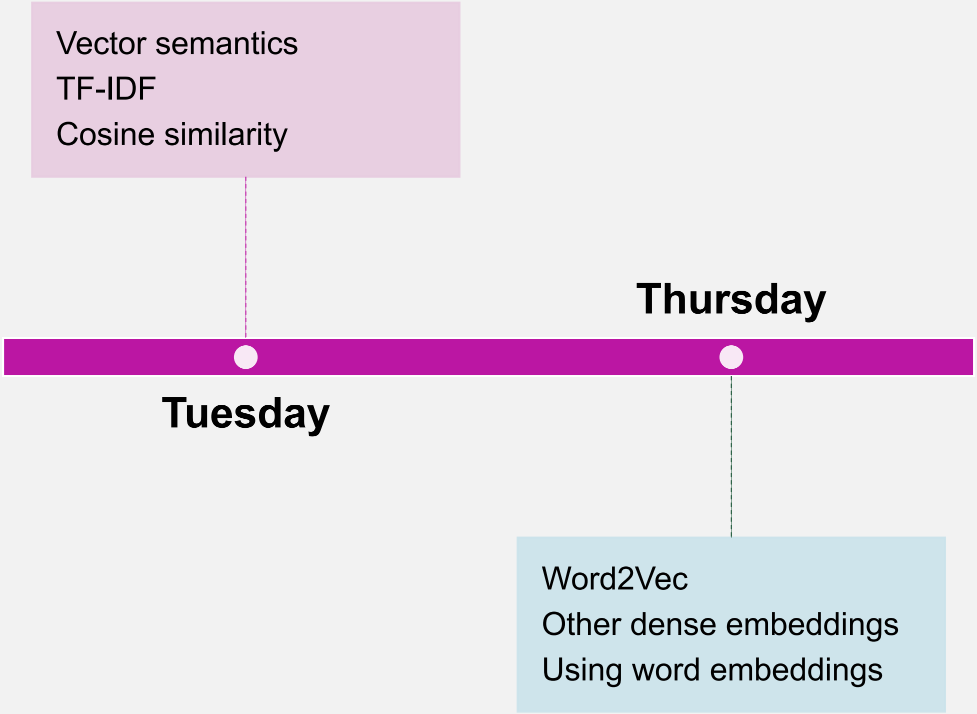
$$\text{cos(digital, information)} = \frac{5*5+1683*3982+1670*3325}{\sqrt{5^2+1683^2+1670^2}\sqrt{5^2+3982^2+3325^2}} = 0.996$$

Result: *information* is way closer to *digital* than it is to *unicorn*!

Summary: Introduction to Vector Semantics

- **Representation learning** is the act of building or learning **word vectors** based on **the distributional hypothesis**
- This process seeks to encode the same linguistic phenomena observed in studies of **lexical semantics**
- Bag-of-words representations are one form of word vector, and **TF-IDF representations** are another
- TF-IDF representations combine simple term frequency with **inverse document frequency** to minimize the impact of words that occur more frequently in general
- The similarity between two vectors can be computed using **cosine similarity**

This Week's Topics



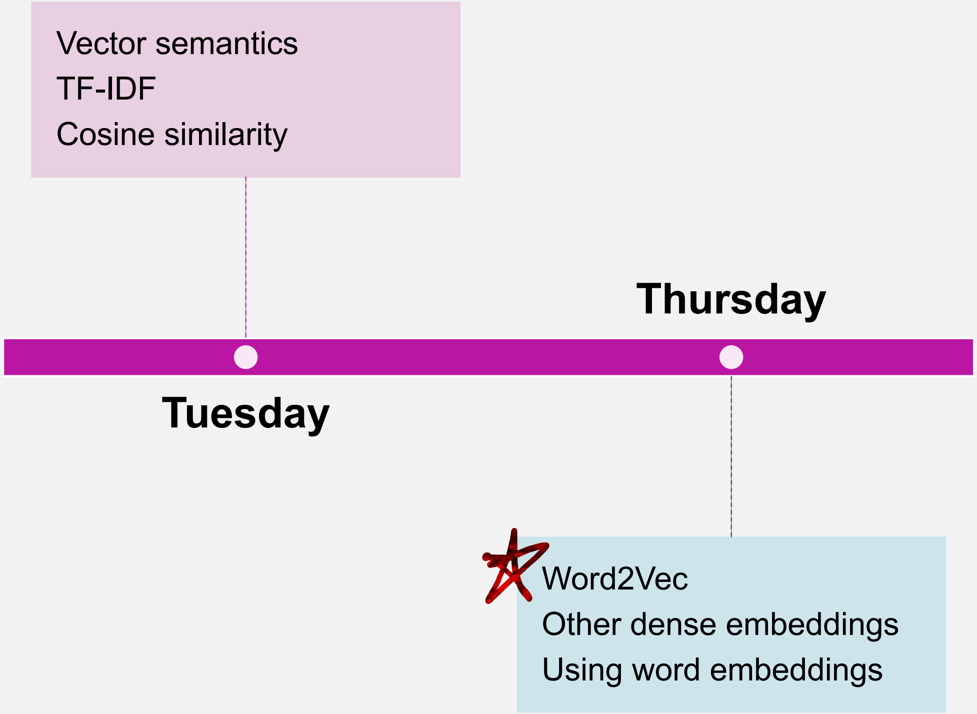
Vector semantics
TF-IDF
Cosine similarity

Tuesday

Thursday

Word2Vec
Other dense embeddings
Using word embeddings

This Week's Topics



Vector semantics
TF-IDF
Cosine similarity

Word2Vec
Other dense embeddings
Using word embeddings

Limitations of Bag-of-Words Style Vectors

- Very **high-dimensional**
- Lots of **empty** (zero-valued) cells
- Struggle with inferring deeper semantic content:
  - Synonyms
  - Antonyms
  - Positive/negative connotations
  - Related contexts

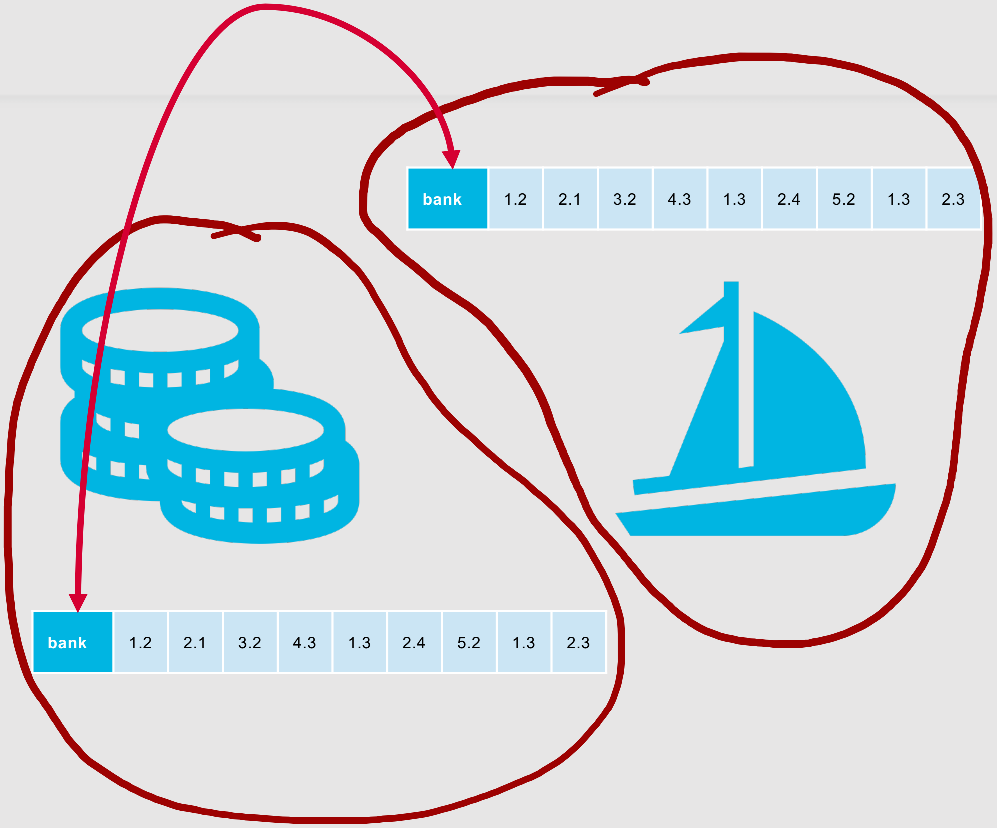
What would our “dream vector” look like?

- **Lower-dimensional** (~ 50-1000 cells)
  - Easier to include as **features**
    - Classifiers have to learn ~100 weights instead of ~50,000
  - Fewer **parameters** → lower chance of overfitting
    - May generalize better to new data
- Most dimensions with **non-zero** values

- We’d also prefer to be able to encode other semantic dimensions of meaning
  - *Good* should be:
    - Far from *bad*
    - Close to *great*
  - For this, we need vector dimensions to correspond to meaning directly, rather than specific words

Enter Word2Vec....

- **Word2Vec:** A method for automatically learning dense word representations from large text corpora
  - Fast
  - Efficient to train
- Not *quite* a dream vector: Word2Vec embeddings are static (homonyms have the same representations)



bank	1.2	2.1	3.2	4.3	1.3	2.4	5.2	1.3	2.3

bank	1.2	2.1	3.2	4.3	1.3	2.4	5.2	1.3	2.3

Word2Vec

- Technically a tool for implementing word vectors:
  - https://code.google.com/archive/p/word2vec
- The algorithm that people usually refer to as *Word2Vec* is the **skip-gram** model with **negative sampling**

Word2Vec Intuition

- Instead of counting how often each word occurs near each context word, train a classifier on a **binary prediction task**
  - Is word *w* likely to occur near context word *c*?
- The twist: **We don’t actually care about the classifier!**
- We use the **learned classifier weights** from this prediction task as our word embeddings

None of this requires manual supervision.

- Text (without any other labels) is framed as **implicitly supervised** training data
  - Given the question: Is word *w* likely to occur near context word *c*?
    - If *w* occurs near *c* in the training corpus, the gold standard answer is “yes”
- Similar setup to **neural language modeling** (neural networks that predict the next word based on prior words), but simpler:
  - Fewer layers
  - Makes **binary yes/no predictions** rather than predicting words

What does the classification task look like?

- **Goal:** Train a classifier that, given a tuple (*t*, *c*) of a target word *t* paired with a context word *c* (e.g., (super, bowl) or (super, laminator)), will return the probability that *c* is a real context word
  - P(+ | t,c)
- Context is defined by our context window (in this case, ± 2 words)

this	sunday,	watch	the	super	bowl	at	5:30	p.m.
		c1	c2	t	c3	c4		

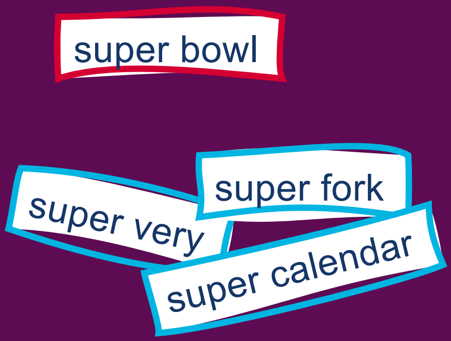
High-Level Overview: How Word2Vec Works

- Treat the target word *w* and a neighboring context word *c* as positive samples

super bowl

High-Level Overview: How Word2Vec Works

- Treat the target word *w* and a neighboring context word *c* as positive samples
- Randomly sample other words in the lexicon to get negative samples



super bowl

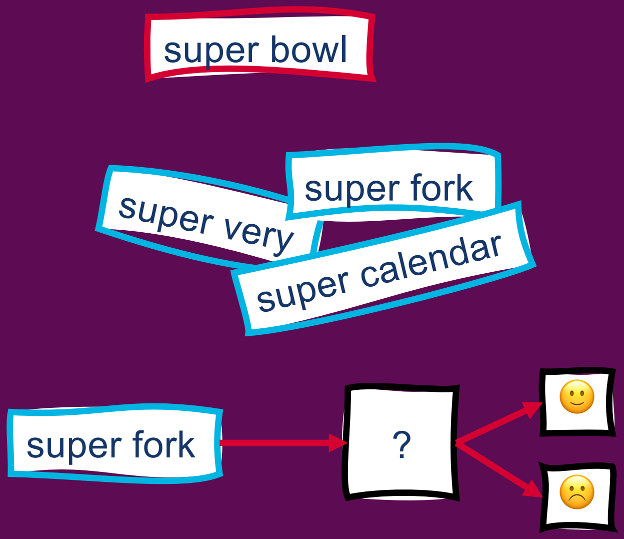
super very

super fork

super calendar

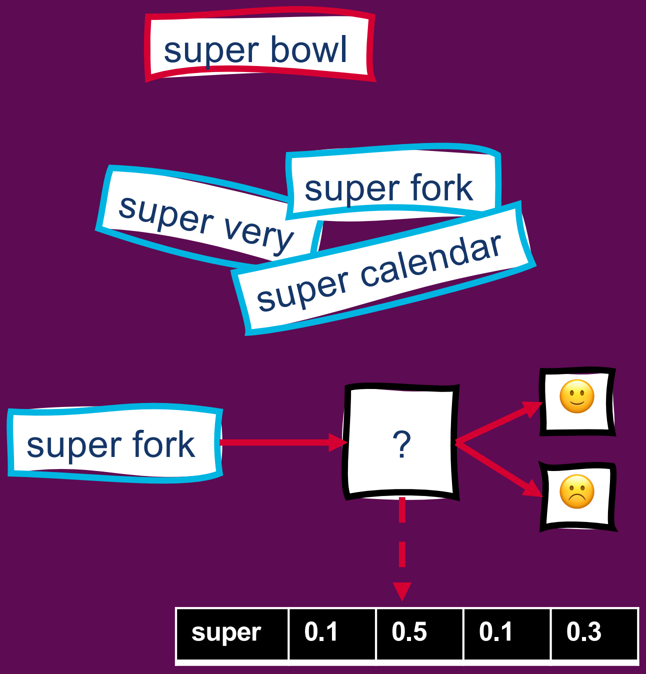
High-Level Overview: How Word2Vec Works

- Treat the target word *w* and a neighboring context word *c* as positive samples
- Randomly sample other words in the lexicon to get negative samples
- Train a classifier to distinguish between those two cases



High-Level Overview: How Word2Vec Works

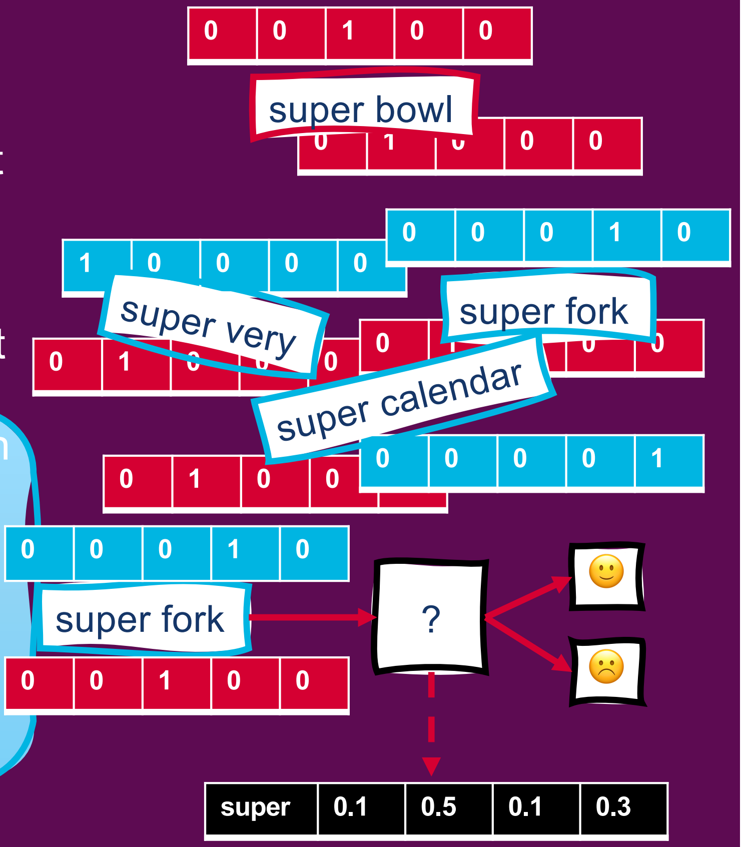
- Treat the target word *w* and a neighboring context word *c* as positive samples
- Randomly sample other words in the lexicon to get negative samples
- Train a classifier to distinguish between those two cases
- Use the weights from that classifier as the word embeddings



super	0.1	0.5	0.1	0.3

High-Level Overview: How Word2Vec Works

- Represent all words in a vocabulary as a vector
- Treat the target word *w* and a neighboring context word *c* as positive samples
- Randomly sample other words in the lexicon to get negative samples
- Find the similarity for each (t,c) pair and use this to calculate P(+|(t,c))
- Train a classifier to maximize these probabilities to distinguish between positive and negative cases
- Use the weights from that classifier as the word embeddings



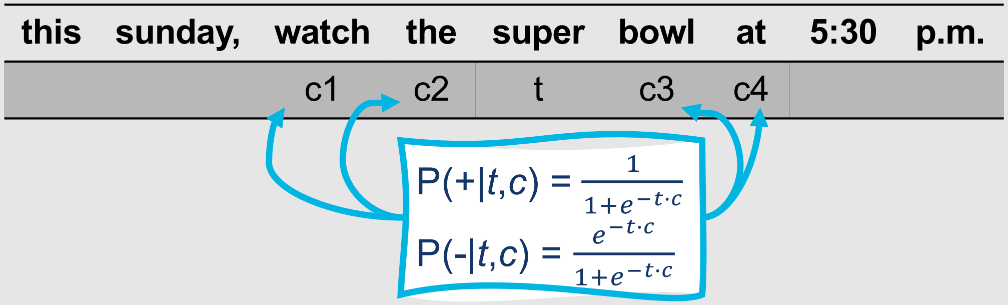
How do we *compute* P(+ | *t*,*c*)?

- This is based on vector similarity
- We can assume that vector similarity is proportional to the dot product between two vectors
  - Similarity(*t*,*c*) $\propto t \cdot c$
- More similar vectors → more likely that *c* occurs near *t*

A dot product gives us a number, not a probability.

- How do we turn it into a probability?
  - **Sigmoid function** (just like we did with logistic regression!)
  - We can set:
    - P(+|t,c) = $\frac{1}{1+e^{-t\cdot c}}$
- Then:
  - P(+ | t,c) = $\frac{1}{1+e^{-t\cdot c}}$
  - P(- | t,c) = 1 - P(+ | t,c) = $\frac{e^{-t\cdot c}}{1+e^{-t\cdot c}}$

What if we want to know the probability that a span of text occurs in the context of the target word?



this	sunday,	watch	the	super	bowl	at	5:30	p.m.
		c1	c2	t	c3	c4		

P(+|t,c) = $\frac{1}{1+e^{-t \cdot c}}$

P(-|t,c) = $\frac{e^{-t \cdot c}}{1+e^{-t \cdot c}}$

- Simplifying assumption: **All context words are independent**
- So, we can just multiply their probabilities:
  - P(+|t,$c_{1:k}$) = $\prod_{i=1}^{k} \frac{1}{1+e^{-t \cdot c_i}}$, or
  - log P(+|t,$c_{1:k}$) = $\sum_{i=1}^{k} \log \frac{1}{1+e^{-t \cdot c_i}}$

With this in mind….

this	sunday,	watch	the	super	bowl	at	5:30	p.m.
		c1	c2	t	c3	c4		
		P(+|*super*, *watch*) = .7	P(+| *super*, *the*) = .5		P(+|*super*, *bowl*) = .9	P(+| *super* *at*) = .5		

$$\boxed{P(+|t,c_{1:k}) = .7 * .5 * .9 * .5 = .1575}$$

- Given *t* and a context window of *k* words $c_{1:k}$, we can assign a probability based on how similar the context window is to the target word
- However, we still have some unanswered questions….
  - **How do we determine our input vectors?**
  - **How do we learn word embeddings** throughout this process (this is the real goal of training our classifier in the first place)?

Input Vectors

- Typically represented as **one-hot vectors**
  - **Binary bag-of-words approach:** Place a “1” in the position corresponding to a given word, and a “0” in every other position

super

0	0	0	0	0	0	1	0	0	0

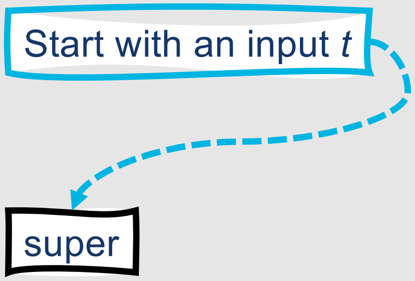
bowl

0	0	1	0	0	0	0	0	0	0

Learned Embeddings….

- Embeddings are the weights learned for a two-layer classifier that predicts P(+ | *t*,*c*)
- Recall from our discussion of logistic regression:
  - $y = \sigma(z) = \frac{1}{1+e^{-z}} = \frac{1}{1+e^{-w \cdot x+b}}$
- This is quite similar to the probability we're trying to optimize:
  - P(+ | *t*,*c*) = $\frac{1}{1+e^{-t \cdot c}}$

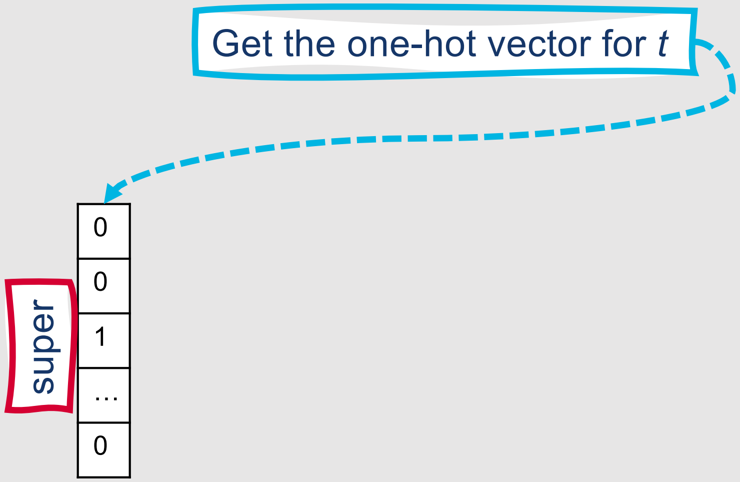
What does this look like?



Start with an input *t*

super

What does this look like?

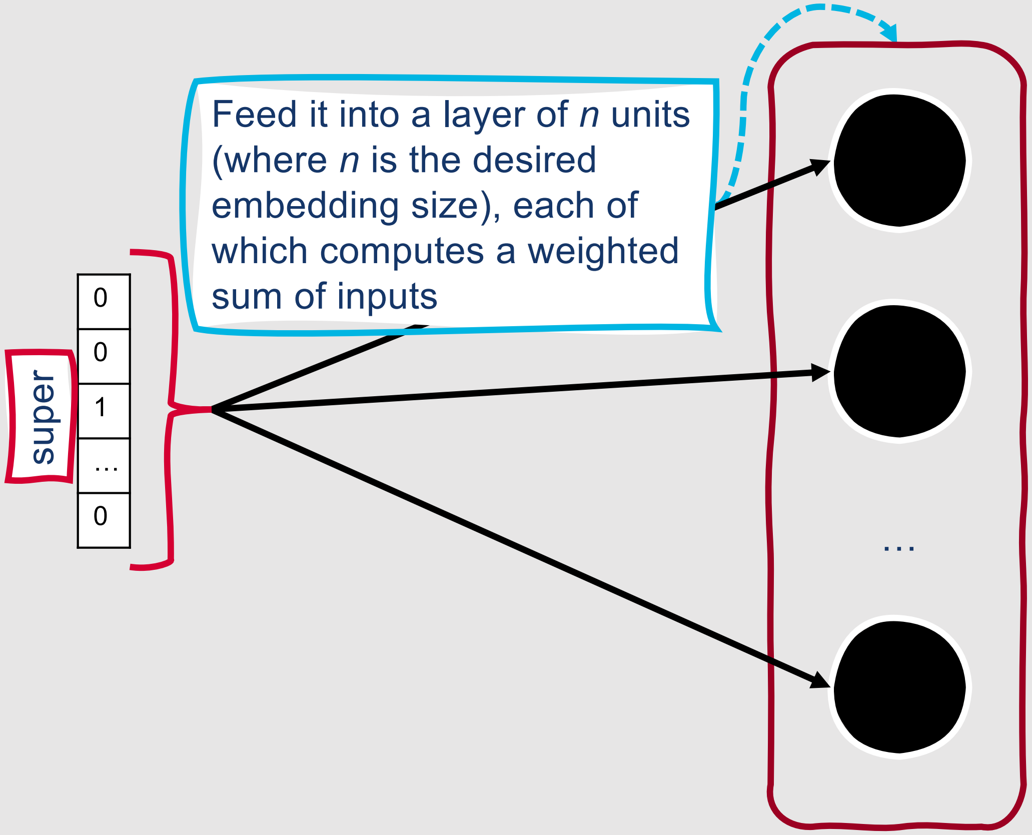


Get the one-hot vector for *t*

super

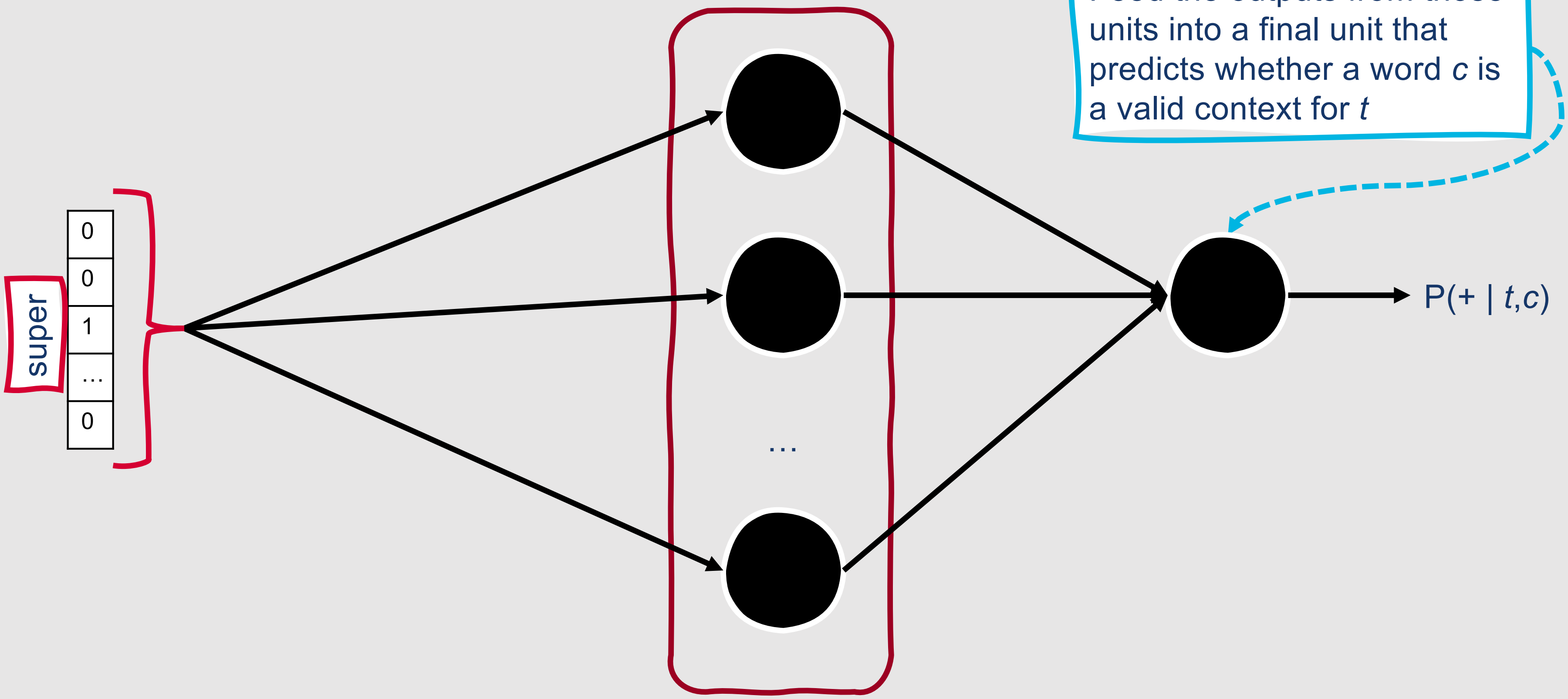
0
0
1
…
0

What does this look like?



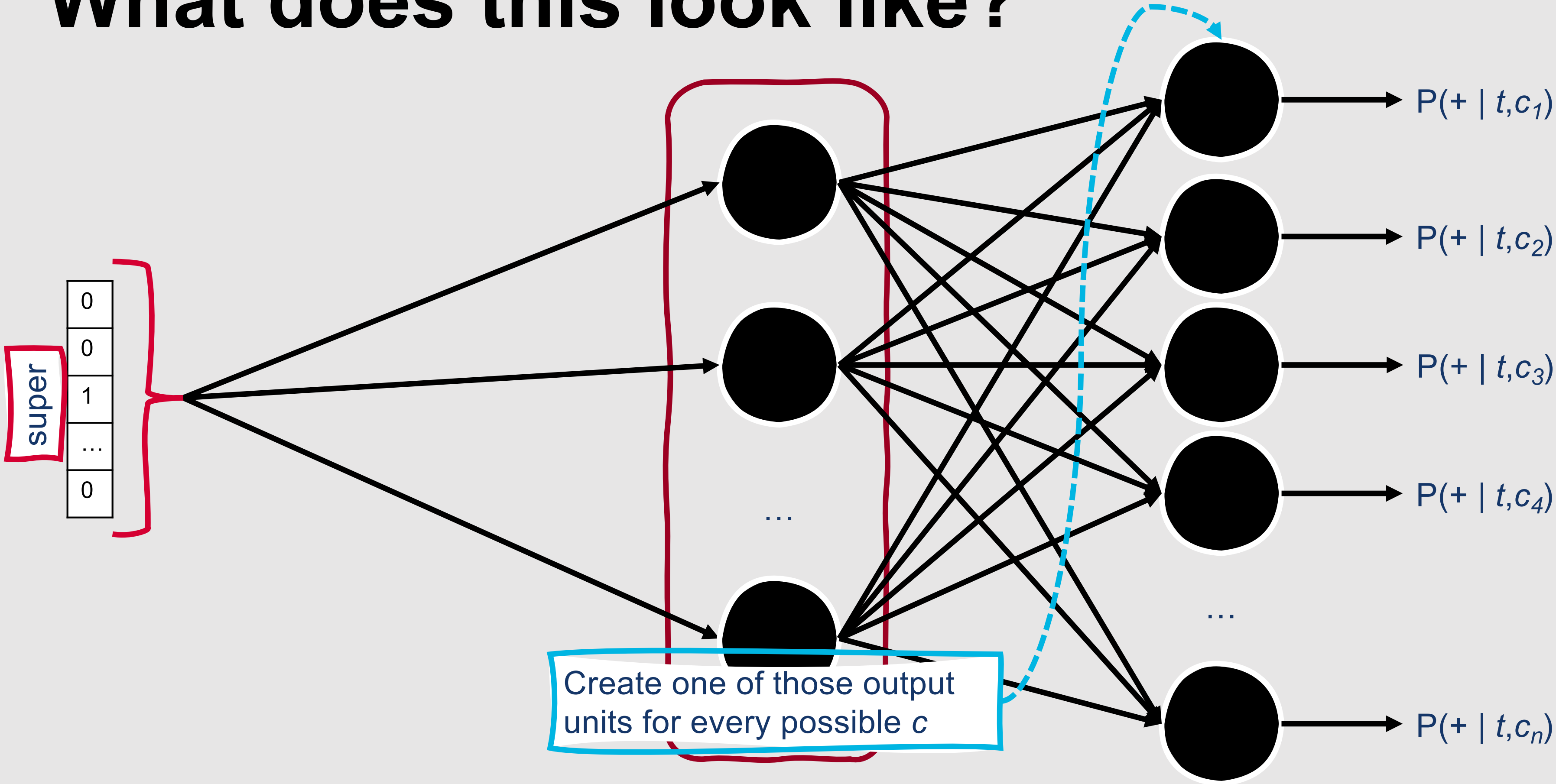
Feed it into a layer of *n* units (where *n* is the desired embedding size), each of which computes a weighted sum of inputs

What does this look like?



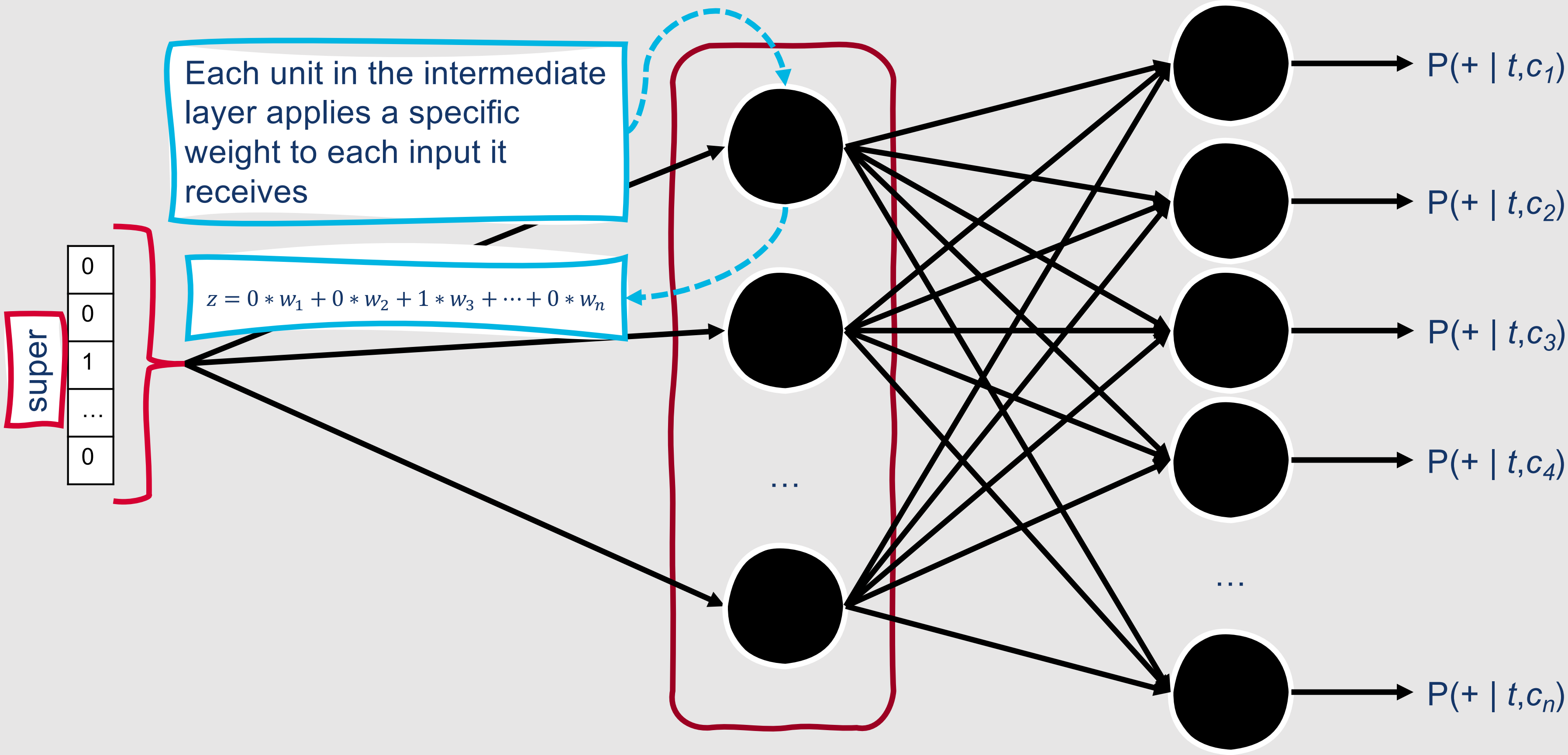
Feed the outputs from those units into a final unit that predicts whether a word *c* is a valid context for *t*

What does this look like?



Create one of those output units for every possible *c*

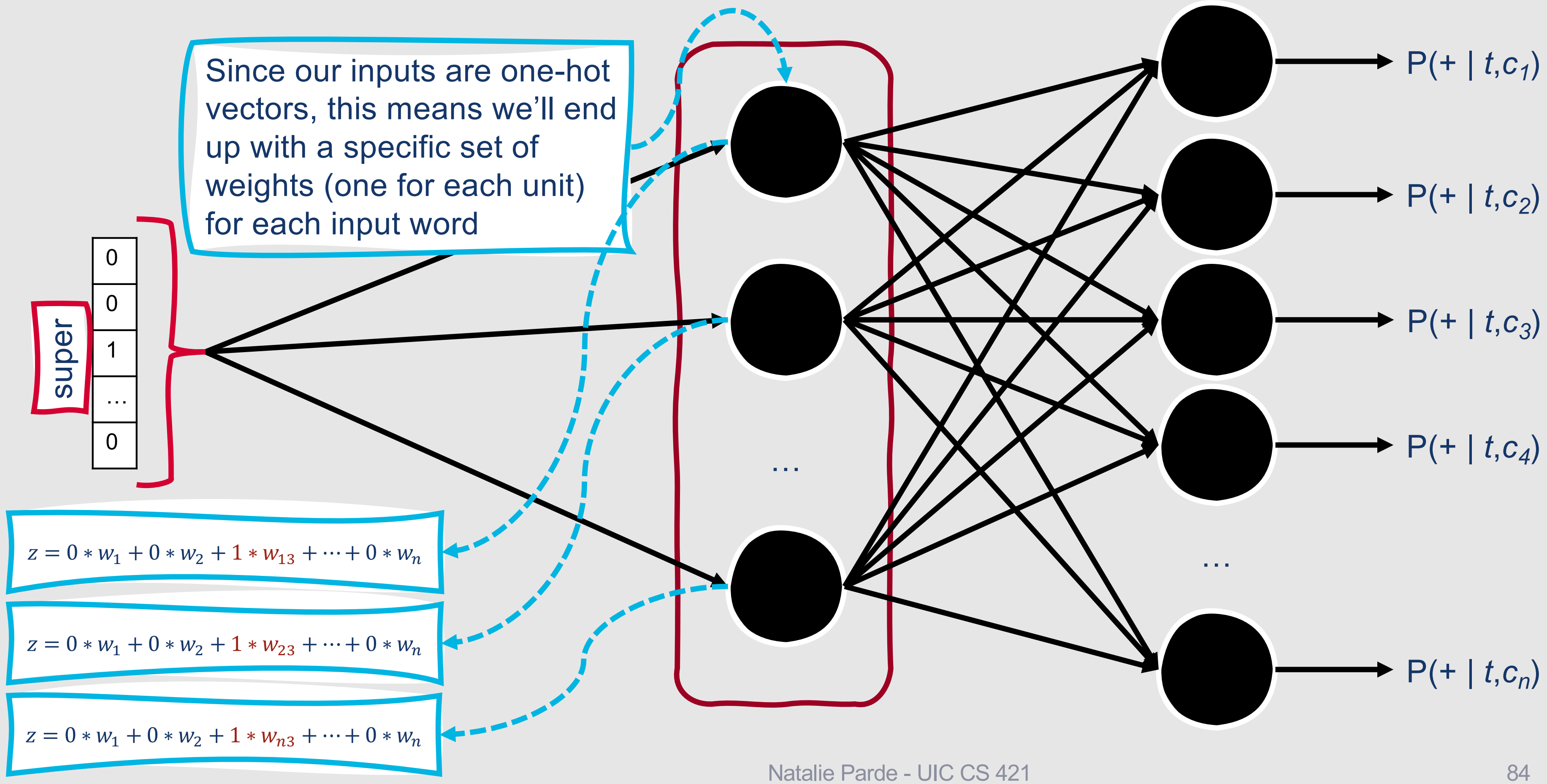
Behind the scenes….



Each unit in the intermediate layer applies a specific weight to each input it receives

$z = 0 * w_1 + 0 * w_2 + 1 * w_3 + \cdots + 0 * w_n$

Behind the scenes….



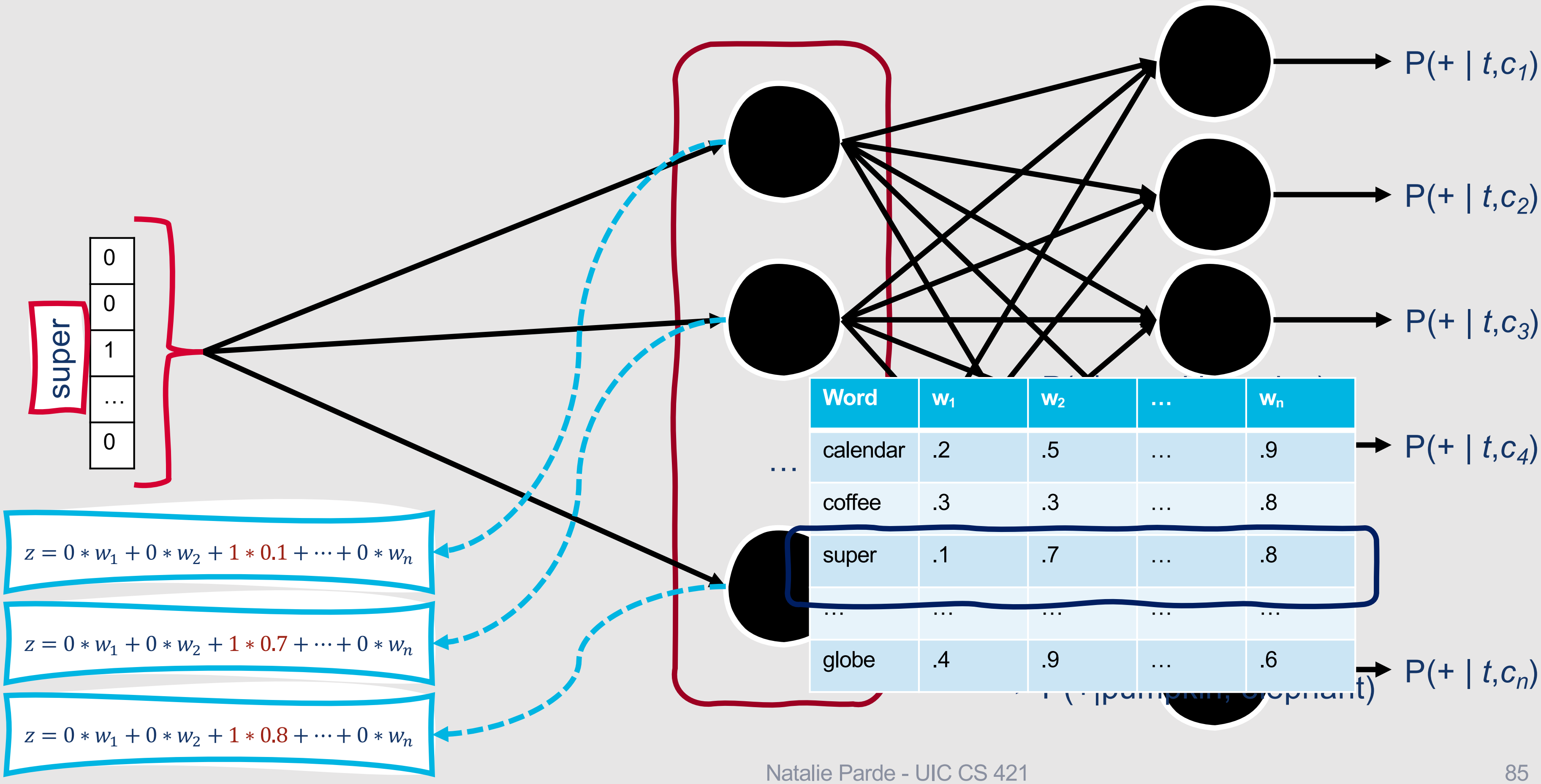
Since our inputs are one-hot vectors, this means we'll end up with a specific set of weights (one for each unit) for each input word

$z = 0 * w_1 + 0 * w_2 + 1 * w_{13} + \cdots + 0 * w_n$

$z = 0 * w_1 + 0 * w_2 + 1 * w_{23} + \cdots + 0 * w_n$

$z = 0 * w_1 + 0 * w_2 + 1 * w_{n3} + \cdots + 0 * w_n$

These are the weights we're interested in! ✓



$z = 0 * w_1 + 0 * w_2 + 1 * 0.1 + \cdots + 0 * w_n$

$z = 0 * w_1 + 0 * w_2 + 1 * 0.7 + \cdots + 0 * w_n$

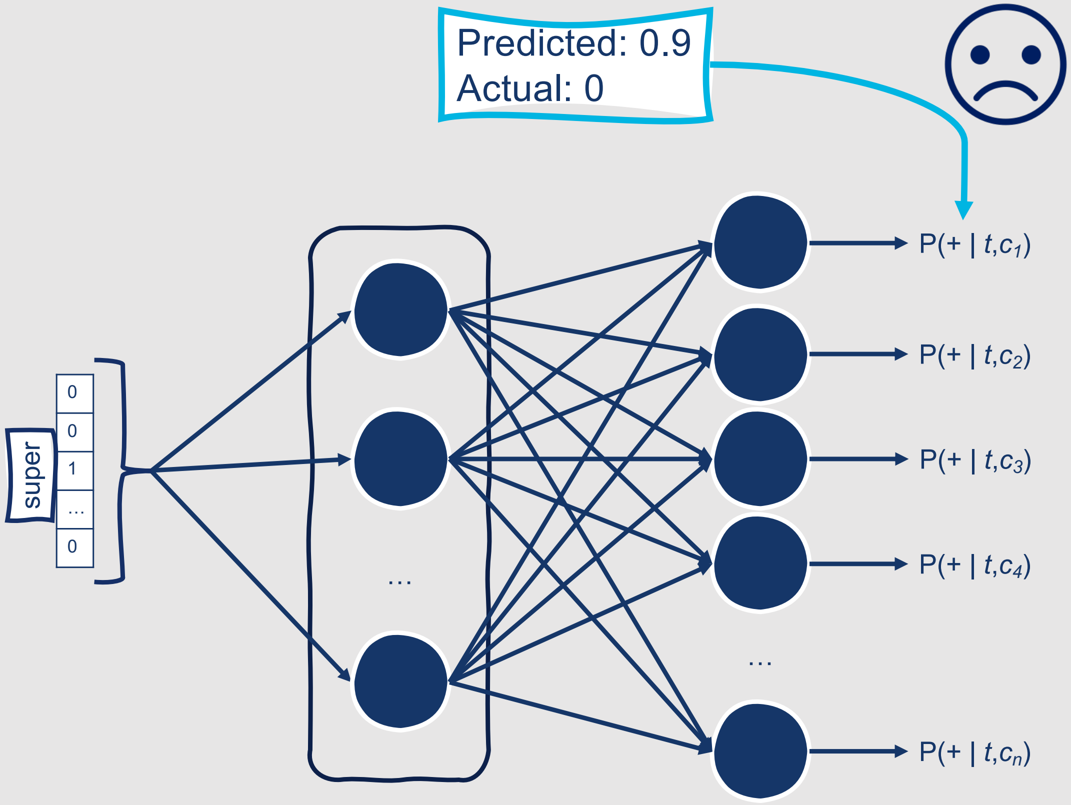
$z = 0 * w_1 + 0 * w_2 + 1 * 0.8 + \cdots + 0 * w_n$

Word	w_1	w_2	…	w_n
calendar	.2	.5	…	.9
coffee	.3	.3	…	.8
super	.1	.7	…	.8
…	…	…	…	…
globe	.4	.9	…	.6

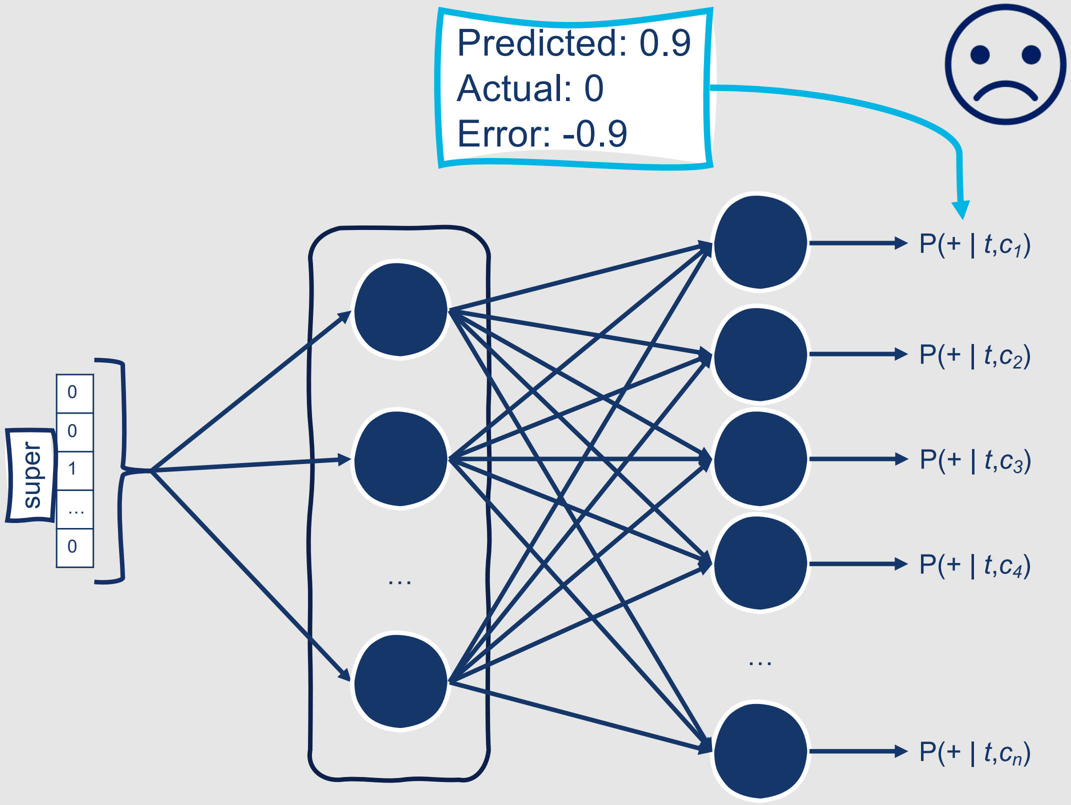
How do we optimize these weights over time?

- Weights are **initialized to some random value** for each word
- They are then iteratively updated to be more similar for words that occur in similar contexts in the training set, and less similar for words that do not
  - Specifically, we want to find weights that maximize P(+|*t*,*c*) for words that occur in similar contexts and minimize P(+|*t*,*c*) for words that do not, given the information we have at the time

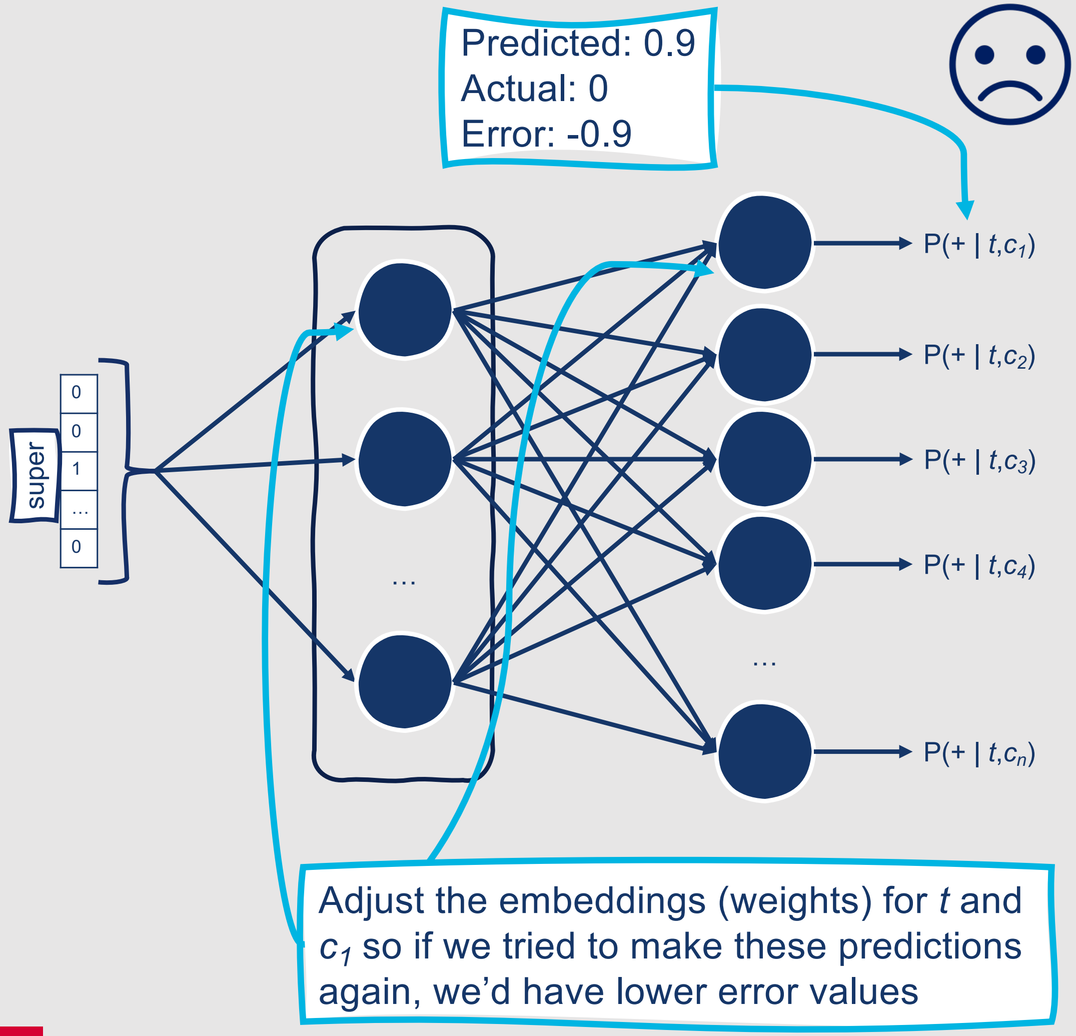
Since we initialize our weights randomly, the classifier's first prediction will almost certainly be wrong.



However, the error values from our incorrect guesses are what allow us to improve our embeddings over time.



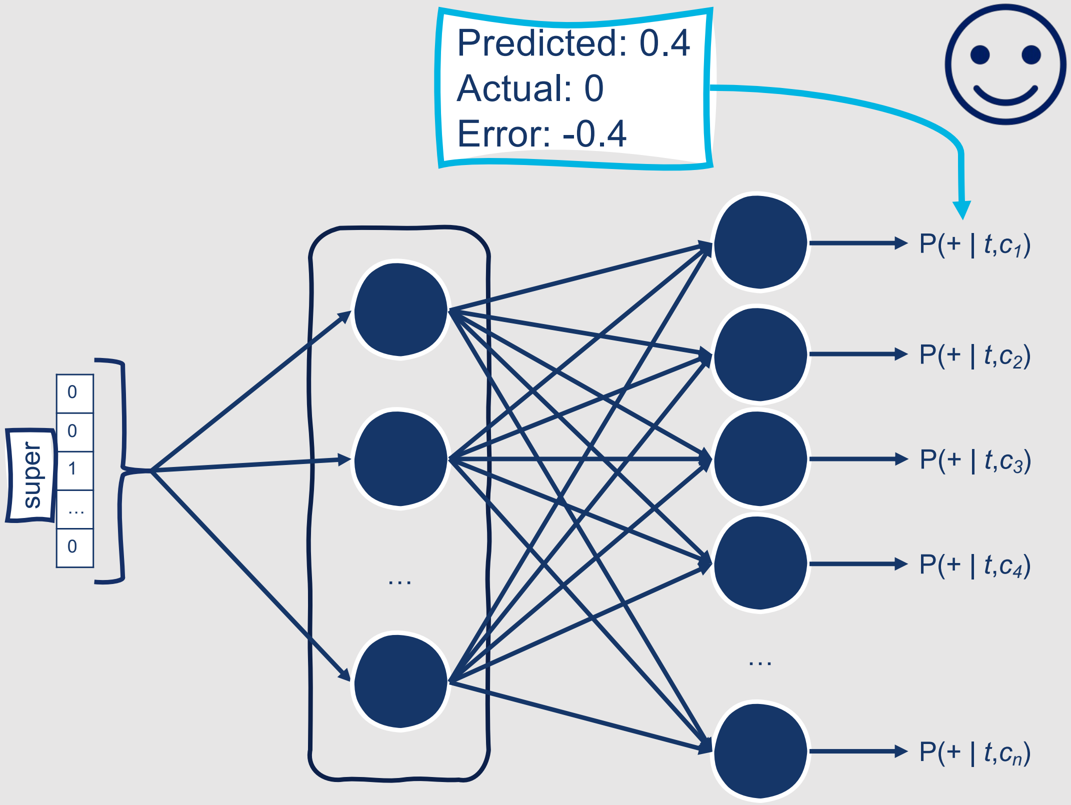
However, the error values from our incorrect guesses are what allow us to improve our embeddings over time.



Predicted: 0.9
Actual: 0
Error: -0.9

Adjust the embeddings (weights) for *t* and c_1 so if we tried to make these predictions again, we'd have lower error values

However, the error values from our incorrect guesses are what allow us to improve our embeddings over time.



What is our training data?

this	sunday,	watch	the	super	bowl	at	5:30
		c1	c2	t	c3	c4	

Positive Examples

t	c
super	watch
super	the
super	bowl
super	at

- We assume that all occurrences of words in similar contexts in our training corpus are **positive samples**

What is our training data?

this	sunday,	watch	the	super	bowl	at	5:30
		c1	c2	t	c3	c4	

Positive Examples

t	c
super	watch
super	the
super	bowl
super	at

- However, we also need negative samples!
- In fact, Word2Vec uses more negative than positive samples (the exact ratio can vary)
- We need to create our own negative examples

What is our training data?

this	sunday,	watch	the	super	bowl	at	5:30
		c1	c2	t	c3	c4	

Positive Examples

t	c
super	watch
super	the
super	bowl
super	at

Negative Examples

t	c
super	calendar
super	exam
super	loud
super	bread
super	cellphone
super	enemy
super	penguin
super	drive

- How to create negative examples?
  - Target word + “noise” word that is sampled from the training set
  - Noise words are chosen according to their weighted unigram frequency $p_\alpha(w)$, where α is a weight:
    - $p_\alpha(w) = \frac{\text{count}(w)^\alpha}{\sum_{w'} \text{count}(w')^\alpha}$
    - Often, $\alpha = 0.75$ to give rarer noise words slightly higher probability of being randomly sampled
- Randomly select noise words according to weighted unigram frequency

Learning Skip-Gram Embeddings

- The model uses these positive and negative samples to:
  - Maximize the vector similarity of the (target, context) pairs drawn from positive examples
  - Minimize the vector similarity of the (target, context) pairs drawn from negative examples
- Parameters (target and context weight vectors) are fine-tuned by:
  - Applying stochastic gradient descent
  - Optimizing a cross-entropy loss function

What if we want to predict a target word from a set of context words instead?

- **Continuous Bag of Words (CBOW)**
  - Another variation of Word2Vec
- Very similar to skip-gram model!

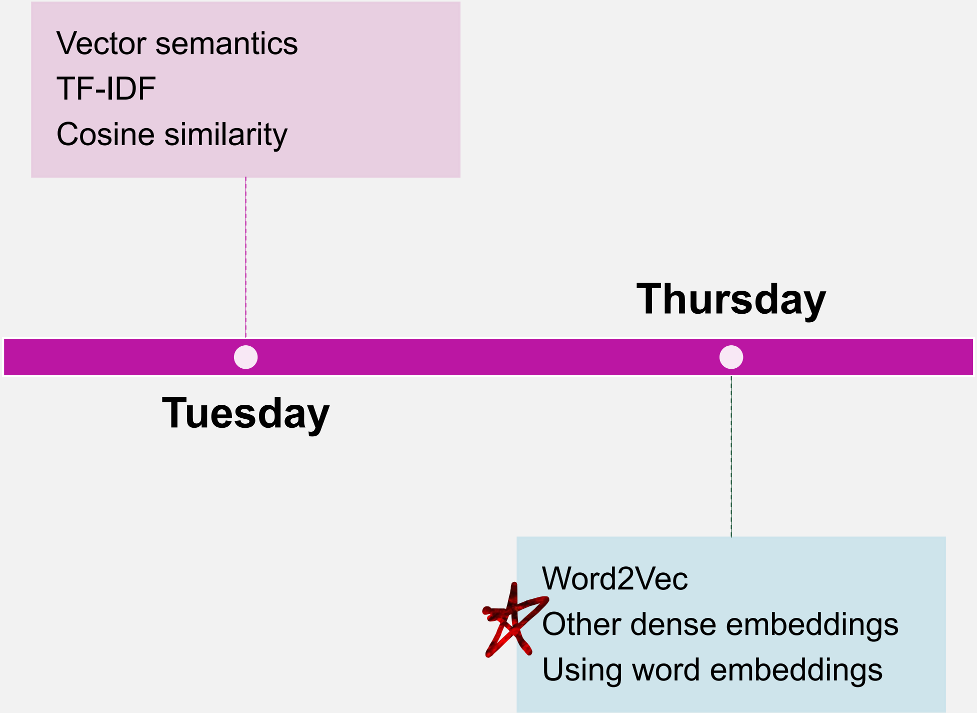
In general, skip-gram embeddings are good with:

- Small datasets
- Rare words and phrases

CBOW embeddings are good with:

- Larger datasets (faster to train)
- Frequent words

This Week's Topics



Tuesday

Vector semantics
TF-IDF
Cosine similarity

Thursday

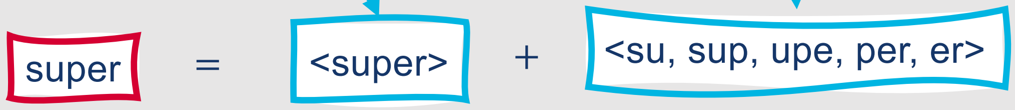
Word2Vec
☆ Other dense embeddings
Using word embeddings

Are there any other variations of Word2Vec?

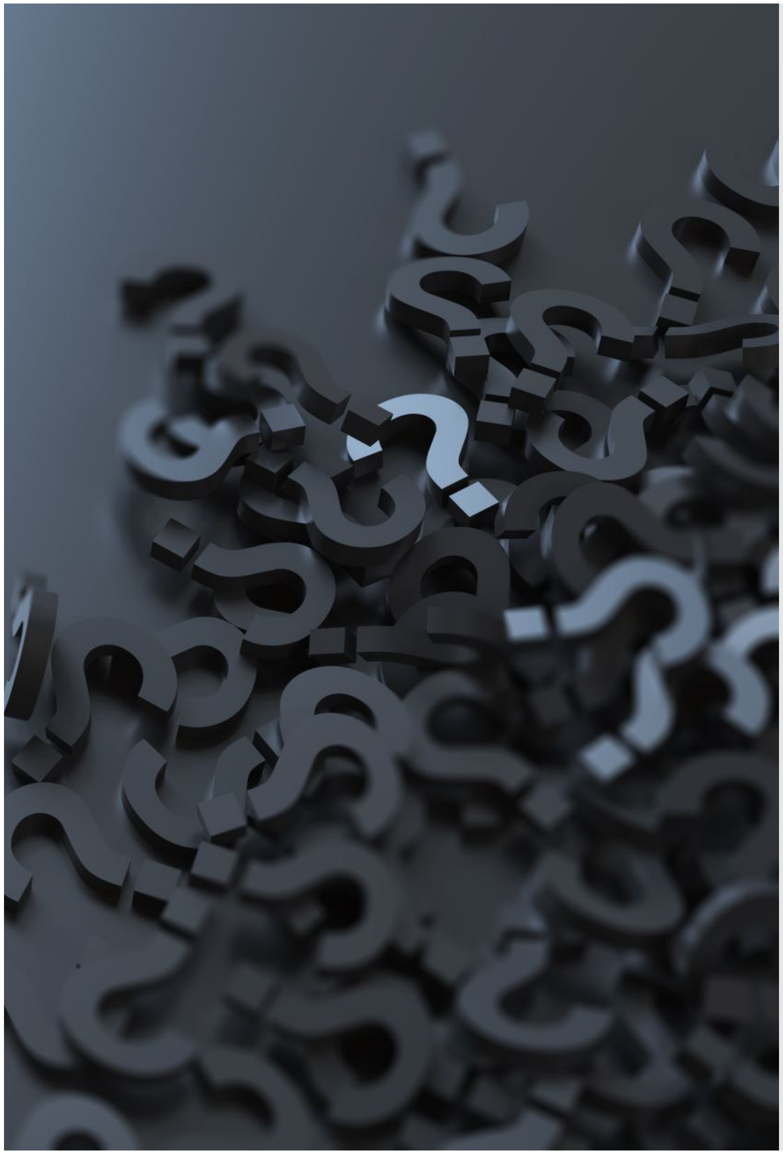
- **fastText**
  - An extension of Word2Vec that also incorporates **subwords**
  - Designed to better handle unknown words and sparsity in language

fastText

- Each word is represented as:
  - Itself
  - A bag of constituent n-grams



super = <super> + <su, sup, upe, per, er>



fastText

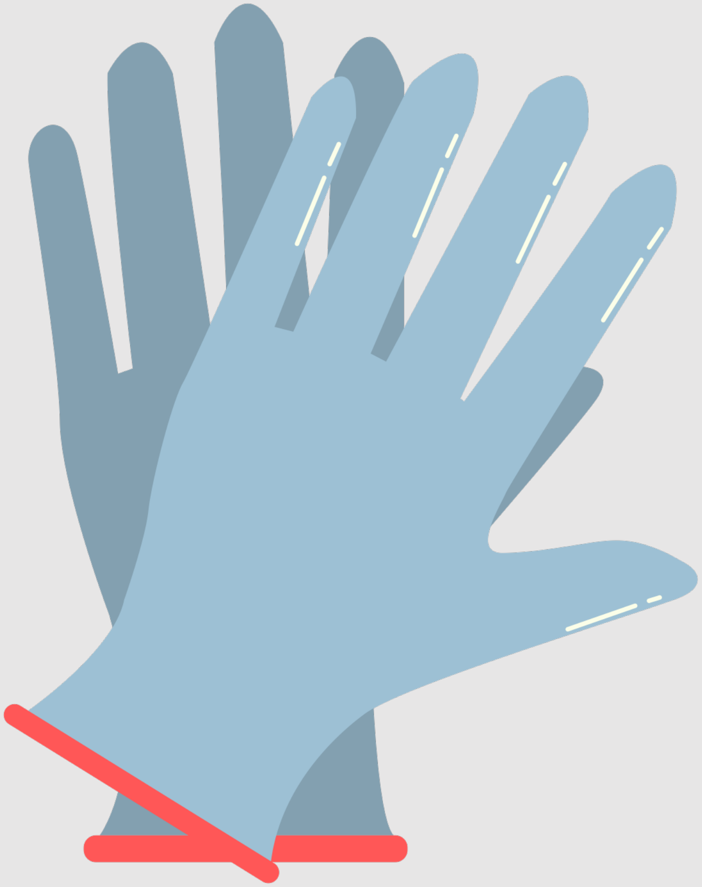
- Skip-gram embedding is learned for each constituent n-gram
- Word is represented by the sum of all embeddings of its constituent n-grams

- Key advantage of this extension?
  - Allows embeddings to be predicted for unknown words based on subword constituents alone

Source code available online: https://fasttext.cc/

Other Types of Dense Word Embeddings

- Word2Vec is an example of a **predictive** word embedding model
  - Learns to predict whether words belong in a target word's context
- Other models are **count-based**
  - Remember co-occurrence matrices?
- GloVE combines aspects of both predictive and count-based models



Global Vectors for Word Representation (GloVe)

- Co-occurrence matrices quickly grow extremely large
- GloVe learns to predict weights in a lower-dimensional space that correspond to the co-occurrence probabilities between words
- Why is this useful?
  - Predictive models → black box
    - They work, but why?
  - GloVe models are easier to interpret

How does GloVe work?

	c_1	…	c_n
t_1	123	…	456
…	…	…	…
t_n	0	…	789

Build a huge word-context co-occurrence matrix

How does GloVe work?

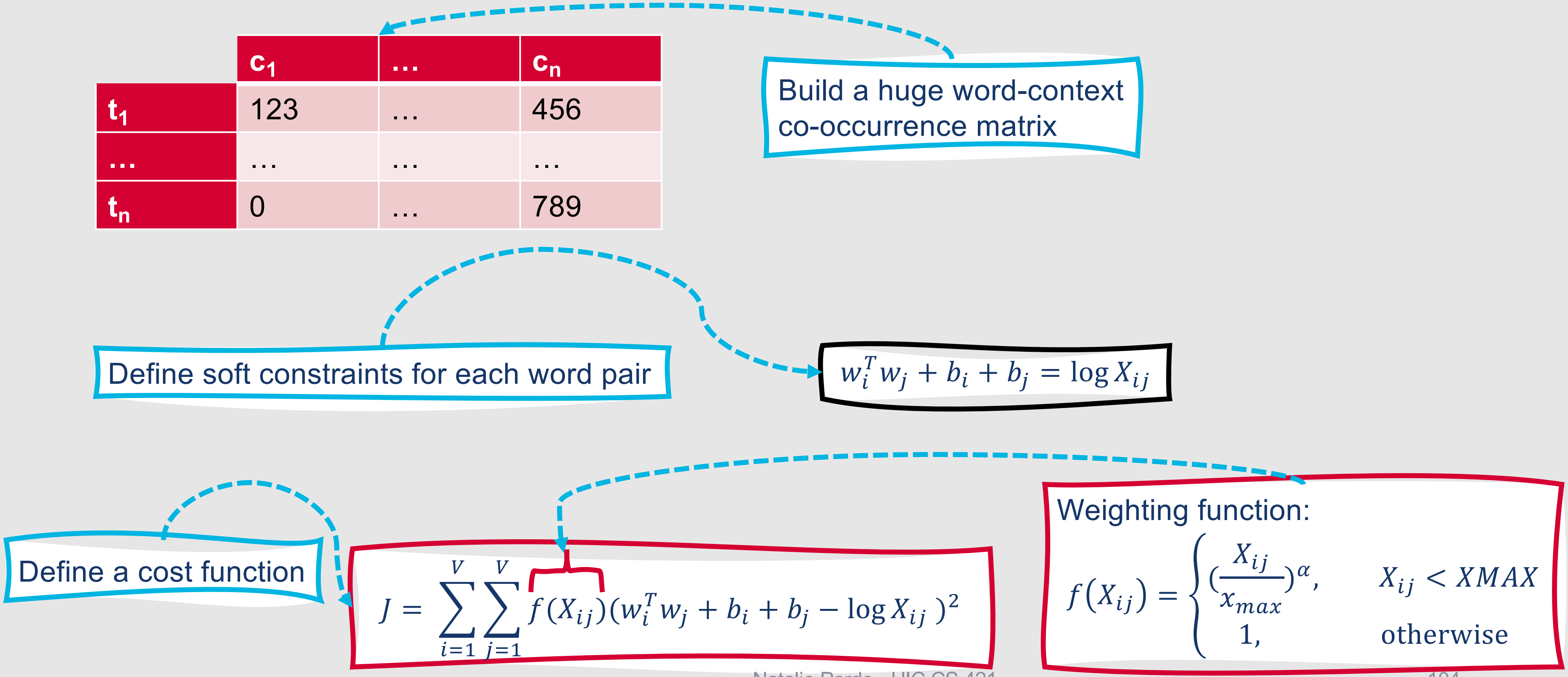
	c_1	…	c_n
t_1	123	…	456
…	…	…	…
t_n	0	…	789

Build a huge word-context co-occurrence matrix

Define soft constraints for each word pair

$$\underbrace{w_i^T}_{\text{Vector for } t_i}\underbrace{w_j}_{\text{Vector for } c_j} + \overbrace{b_i + b_j}^{\text{Scaler biases for } t_i \text{ and } c_j} = \log \underbrace{X_{ij}}_{\text{Co-occurrence count for } t_ic_j}$$

How does GloVe work?



	c_1	…	c_n
t_1	123	…	456
…	…	…	…
t_n	0	…	789

Build a huge word-context co-occurrence matrix

Define soft constraints for each word pair

$$w_i^T w_j + b_i + b_j = \log X_{ij}$$

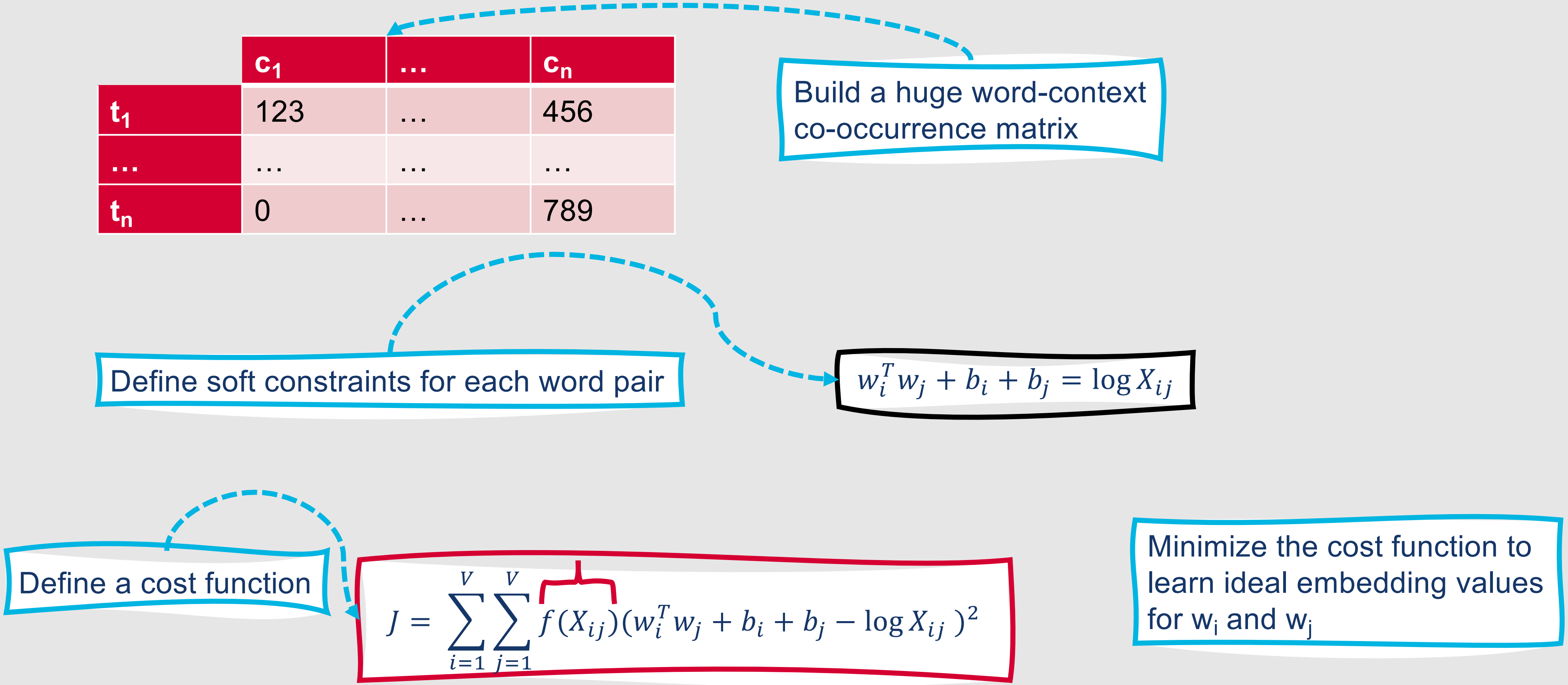
Define a cost function

$$J = \sum_{i=1}^{V}\sum_{j=1}^{V} \overbrace{f(X_{ij})}(w_i^T w_j + b_i + b_j - \log X_{ij})^2$$

Weighting function:

$$f(X_{ij}) = \begin{cases} (\frac{X_{ij}}{x_{max}})^{\alpha}, & X_{ij} < XMAX \\ 1, & \text{otherwise} \end{cases}$$

How does GloVe work?



	c_1	…	c_n
t_1	123	…	456
…	…	…	…
t_n	0	…	789

Build a huge word-context co-occurrence matrix

Define soft constraints for each word pair

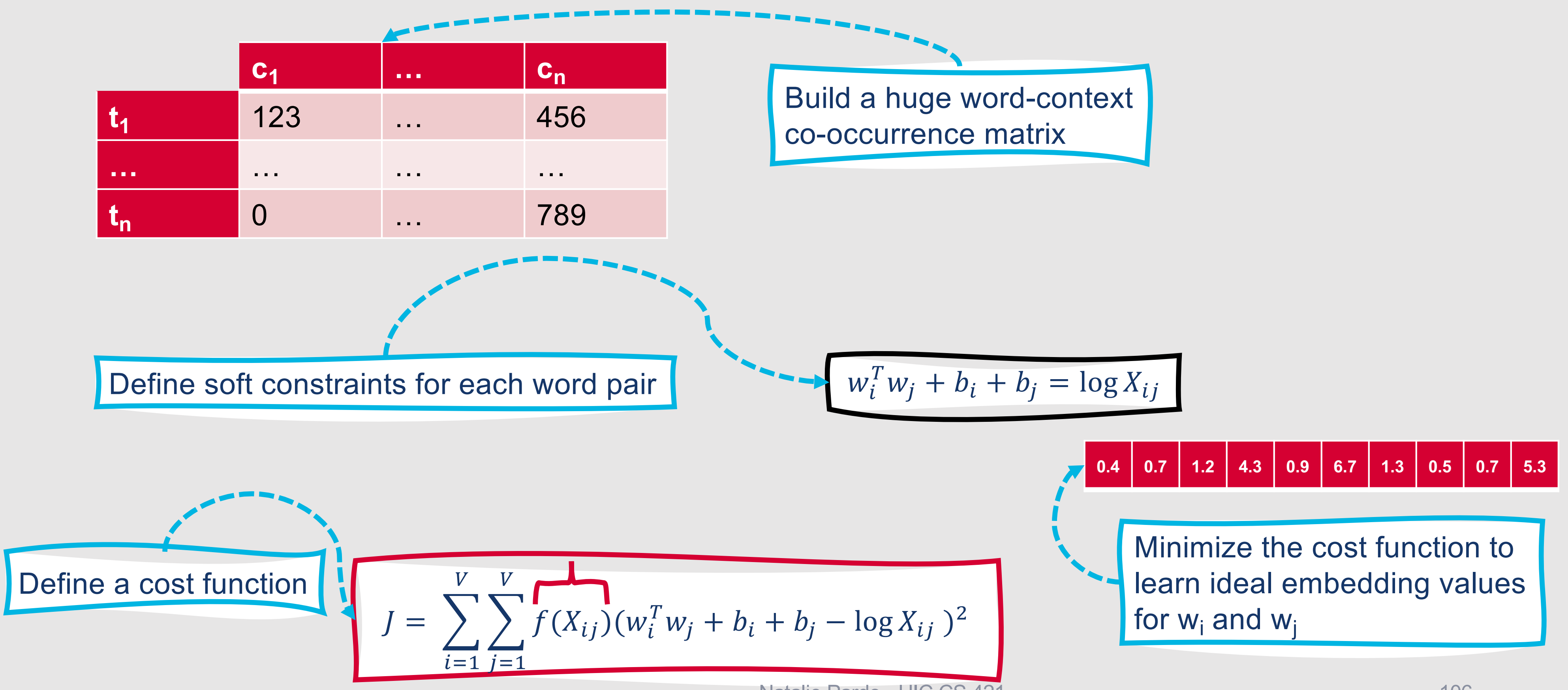
$$w_i^T w_j + b_i + b_j = \log X_{ij}$$

Define a cost function

$$J = \sum_{i=1}^{V} \sum_{j=1}^{V} \overbrace{f(X_{ij})}(w_i^T w_j + b_i + b_j - \log X_{ij})^2$$

Minimize the cost function to learn ideal embedding values for w_i and w_j

How does GloVe work?



	c$_1$	…	c$_n$
t$_1$	123	…	456
…	…	…	…
t$_n$	0	…	789

Build a huge word-context co-occurrence matrix

Define soft constraints for each word pair

$$w_i^T w_j + b_i + b_j = \log X_{ij}$$

Define a cost function

$$J = \sum_{i=1}^{V} \sum_{j=1}^{V} \overbrace{f(X_{ij})}(w_i^T w_j + b_i + b_j - \log X_{ij})^2$$

0.4	0.7	1.2	4.3	0.9	6.7	1.3	0.5	0.7	5.3

Minimize the cost function to learn ideal embedding values for w$_i$ and w$_j$

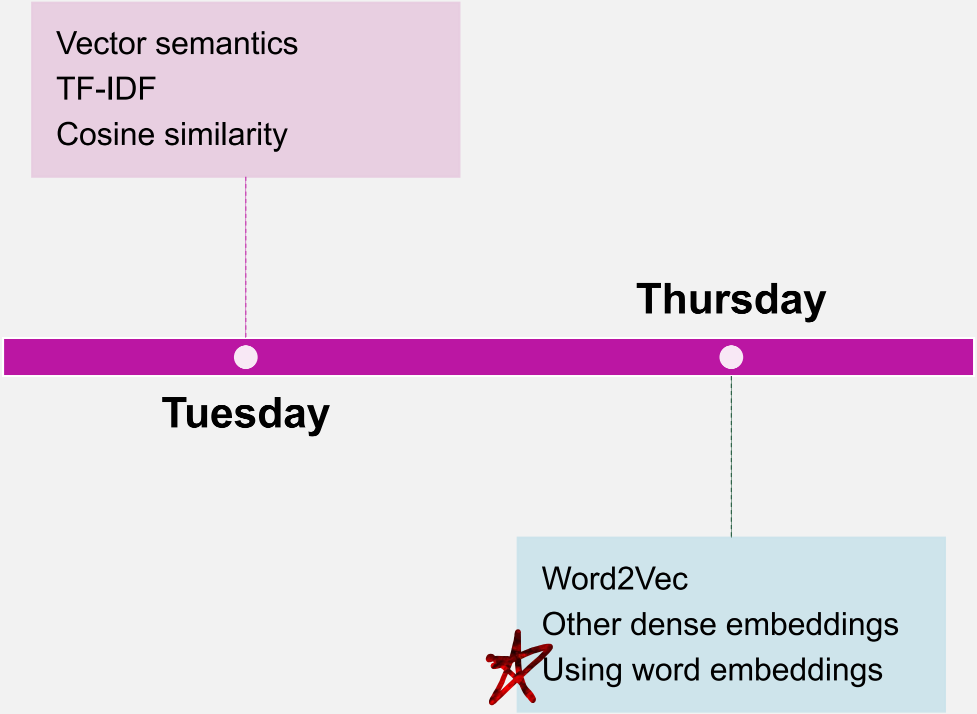
Why does GloVe work?

- Ratios of co-occurrence probabilities have the potential to encode word similarities and differences
- These similarities and differences are useful components of meaning
  - GloVe embeddings perform particularly well on analogy tasks

Which is better …Word2Vec or GloVe?

- In general, Word2Vec and GloVe produce similar embeddings
- Word2Vec → slower to train but less memory intensive
- GloVe → faster to train but more memory intensive
- Word2Vec and Glove both produce context-independent embeddings
- Contextual embeddings:
  - ELMo (Peters et al., 2018; https://www.aclweb.org/anthology/N18-1202/)
  - BERT (Devlin et al., 2019; https://www.aclweb.org/anthology/N19-1423/)

This Week's Topics



Tuesday

Vector semantics
TF-IDF
Cosine similarity

Thursday

Word2Vec
Other dense embeddings
Using word embeddings

Evaluating Vector Models

- Extrinsic Evaluation
  - Add the vectors as features in a downstream NLP task, and see whether and how this changes performance relative to a baseline model
  - Most important evaluation metric for word embeddings!
    - Word embeddings are rarely needed in isolation
    - They are almost solely used to boost performance in downstream tasks

Evaluating Vector Models

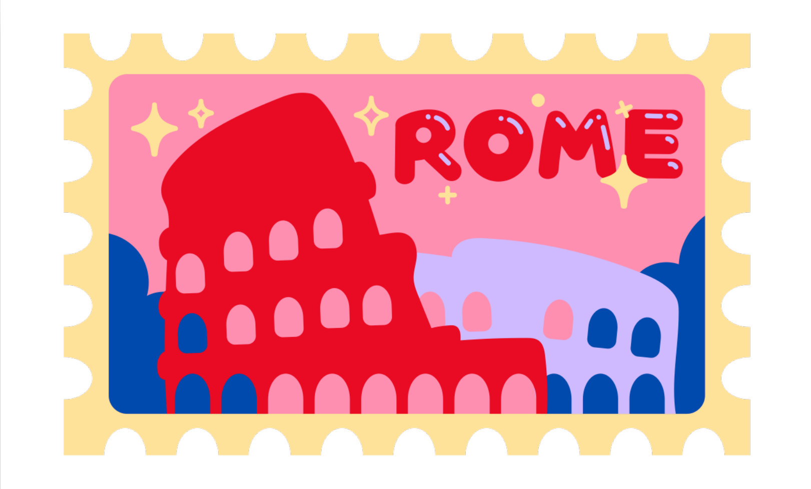
- Intrinsic Evaluation
  - Performance at predicting:
    - Word similarity
    - Text similarity
    - Analogy

Evaluating Performance at Predicting Word Similarity

- Compute the **cosine similarity** between vectors for pairs of words
- Compute the **correlation** between those similarity scores and word similarity ratings for the same pairs of words manually assigned by humans
- Corpora for doing this:
  - WordSim-353
  - SimLex-999
  - TOEFL Dataset
    - *Levied* is closest in meaning to: (a) imposed, (b) believed, (c) requested, (d) correlated

Analogy

- We can capture **relational meanings** in word embeddings by computing the offsets between values in the same columns for different vectors
- Famous examples (Mikolov et al., 2013; Levy and Goldberg, 2014):
  - king - man + woman = queen
  - Paris - France + Italy = Rome



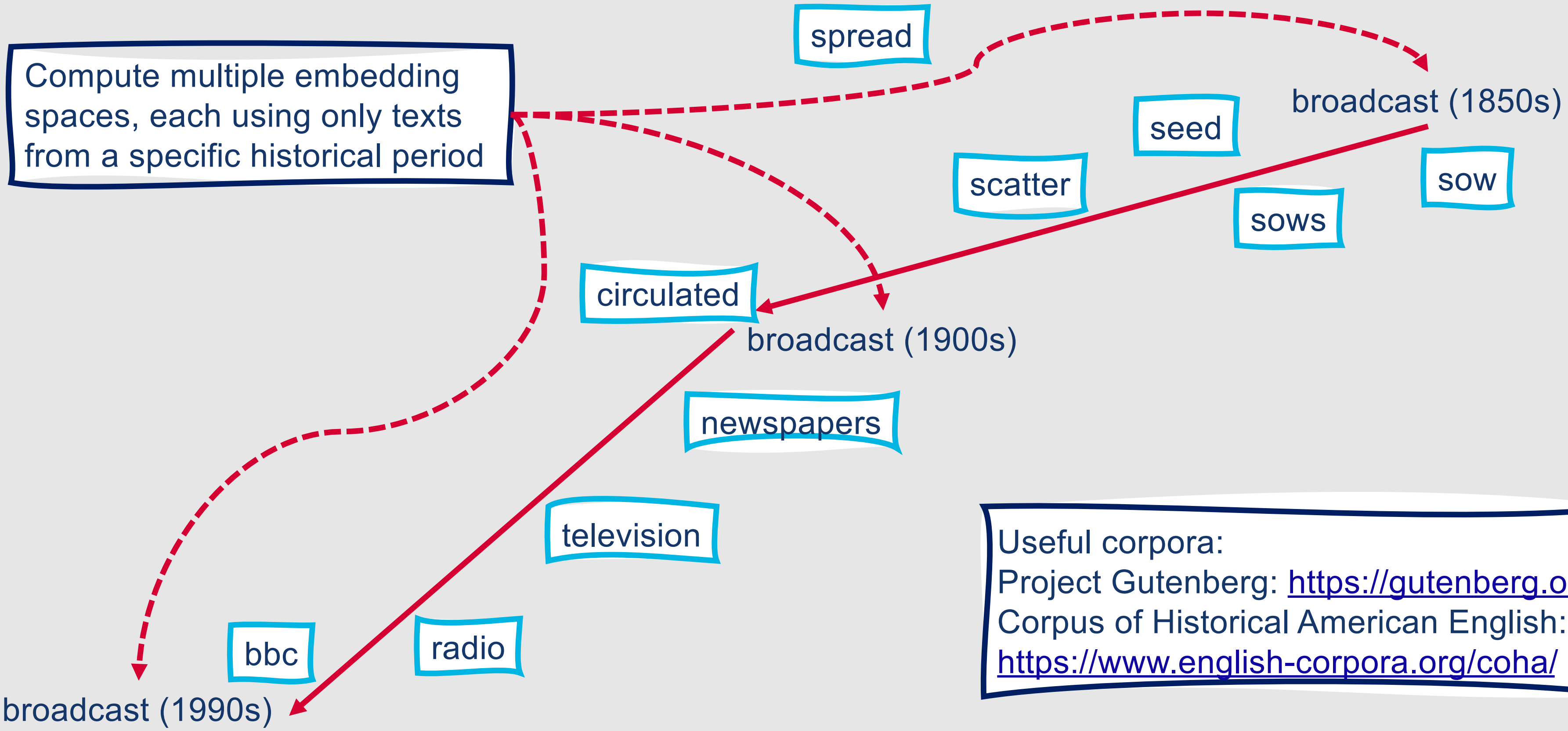
Context window size influences what you learn!

- Shorter context window → more **syntactic representations**
  - Information is from immediately nearby words
  - Most similar words tend to be semantically similar words with the same parts of speech
- Longer context window → more **topical representations**
  - Information can come from longer-distance dependencies
  - Most similar words tend to be topically related, but not necessarily similar (e.g., *diner* and *eats*, rather than *spoon* and *fork*)

Word embeddings have many practical applications.

- Features for text classification tasks
- Representations for computational social science studies
  - Studying word meaning over time
  - Studying implicit associations between words

Embeddings and Historical Semantics



Compute multiple embedding spaces, each using only texts from a specific historical period

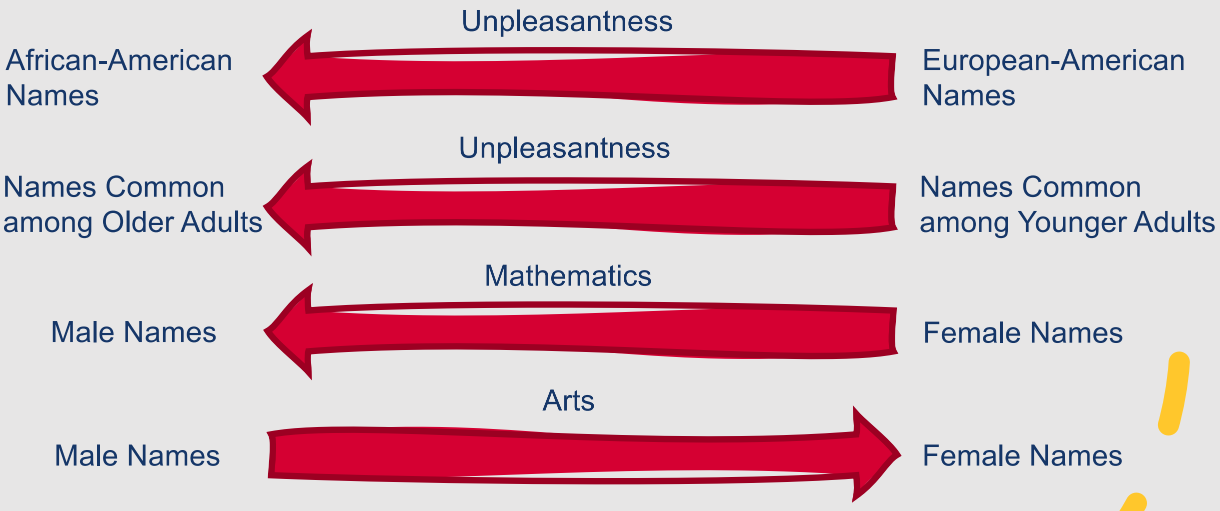
Useful corpora:
Project Gutenberg: https://gutenberg.org
Corpus of Historical American English: https://www.english-corpora.org/coha/

Unfortunately, word embeddings can also end up reproducing implicit biases and stereotypes latent in text.

- Recall: king - man + woman = queen
- Word embeddings trained on news corpora also produce:
  - man - computer programmer + woman = homemaker
  - doctor - father + mother = nurse
- Very problematic for real-world applications (e.g., CV/resume scoring models)

Bias and Embeddings

- Caliskan et al. (2017) identified known, harmful implicit associations in GloVe embeddings



How do we keep the useful associations present in word embeddings, but get rid of the harmful ones?

- Recent research has begun examining ways to **debias** word embeddings by:
    - Transforming embedding spaces to remove gender stereotypes but preserve definitional gender
    - Changing training procedures to eliminate these issues before they arise
- Increasingly active area of study:
    - https://facctconference.org

Summary: Word Embeddings

- **Word2Vec** is a **predictive** word embedding approach that learns word representations by training a classifier to predict whether a **context word** should be associated with a given **target word**
- **GloVe** is a hybrid predictive and **count-based** word embedding approach that learns an optimized, lower-dimensional version of a co-occurrence matrix
- **Word embeddings** can be evaluated through their incorporation in other language tasks, and they can be used to model syntactic and semantic properties of language over time
- Word embeddings may reflect the same **biases** found in the data used to train them