PennState College of the Liberal Arts

C-SoDA Center for Social Data Analytics

Day 3 - Word Embeddings

Advanced Text as Data: Natural Language Processing
Essex Summer School in Social Science Data Analysis

Burt L. Monroe (Instructor) & Sam Bestvater (TA)
Pennsylvania State University

July 28, 2021

Slides are mostly those of Jurafsky and Martin for *Speech and Language Processing*, 3rd edition (all those with light brown border on left)

Where we're headed: word embeddings

- The old (Mikolov, et al. 2013 — word2vec) new hotness in NLP / text-as-data.
- Represent each word as a vector of real numbers, a location in *n*-dimensional space.
- Premise: "You shall know a word by the company it keeps." (Firth 1957)
- Goal: words that appear in the same "contexts" should be near each other, have similar vector representations.
- Context is typically defined as a window of neighboring words.

```
the quick brown fox jumps over the lazy dog
```

([Context words], focus word) with window size of 1:

```
([the, brown], quick), ([quick, fox], brown ...
```

Desiderata

What should a theory of word meaning do for us?

Let's look at some desiderata

From lexical semantics, the linguistic study of word meaning

Lemmas and senses

lemma

mouse (N)

sense

1. any of numerous small rodents...
2. a hand-operated device that controls a cursor...

Modified from the online thesaurus WordNet

A sense or "concept" is the meaning component of a word

Lemmas can be polysemous (have multiple senses)

Relations between senses: Synonymy

Synonyms have the same meaning in some or all contexts.

- filbert / hazelnut
- couch / sofa
- big / large
- automobile / car
- vomit / throw up
- water / H_20

Relations between senses: Synonymy

Note that there are probably no examples of perfect synonymy.

- Even if many aspects of meaning are identical
- Still may differ based on politeness, slang, register, genre, etc.

Relation: Synonymy?

water/H_2O

"H_2O" in a surfing guide?

big/large

my big sister != my large sister

Relation: **Similarity**

Words with similar meanings. Not synonyms, but sharing some element of meaning

```
car, bicycle
cow, horse
```

Ask humans how similar 2 words are

word1	word2	similarity
vanish	disappear	9.8
behave	obey	7.3
belief	impression	5.95
muscle	bone	3.65
modest	flexible	0.98
hole	agreement	0.3

SimLex-999 dataset (Hill et al., 2015)

Relation: Word relatedness

Also called "word association"

Words can be related in any way, perhaps via a semantic frame or field

- `coffee, tea`: **similar**
- `coffee, cup`: **related**, not similar

Semantic field

Words that

- cover a particular semantic domain
- bear structured relations with each other.

hospitals

surgeon, scalpel, nurse, anaesthetic, hospital

restaurants

waiter, menu, plate, food, menu, chef

houses

door, roof, kitchen, family, bed

Relation: Antonymy

Senses that are opposites with respect to only one feature of meaning

Otherwise, they are very similar!

`dark/light    short/long fast/slow    rise/fall`
`hot/cold      up/down    in/out`

More formally: antonyms can

- define a binary opposition or be at opposite ends of a scale
  - `long/short, fast/slow`
- Be *reversives*:
  - `rise/fall, up/down`

Connotation (sentiment)

- Words have **affective** meanings
  - Positive connotations (*happy*)
  - Negative connotations (*sad*)
- Connotations can be subtle:
  - Positive connotation: *copy, replica, reproduction*
  - Negative connotation: *fake, knockoff, forgery*
- Evaluation (sentiment!)
  - Positive evaluation (*great*, *love*)
  - Negative evaluation (*terrible*, *hate*)

Connotation

Osgood et al. (1957)

Words seem to vary along 3 affective dimensions:

- **valence**: the pleasantness of the stimulus
- **arousal**: the intensity of emotion provoked by the stimulus
- **dominance**: the degree of control exerted by the stimulus

	Word	Score		Word	Score
Valence	love	1.000		toxic	0.008
	happy	1.000		nightmare	0.005
Arousal	elated	0.960		mellow	0.069
	frenzy	0.965		napping	0.046
Dominance	powerful	0.991		weak	0.045
	leadership	0.983		empty	0.081

Values from NRC VAD Lexicon (Mohammad 2018)

So far

Concepts or word senses

- Have a complex many-to-many association with **words** (homonymy, multiple senses)

Have relations with each other

- Synonymy
- Antonymy
- Similarity
- Relatedness
- Connotation

Vector semantics

Computational models of word meaning

Can we build a theory of how to represent word meaning, that accounts for at least some of the desiderata?

We'll introduce **vector semantics**

The standard model in language processing!

Handles many of our goals!

Ludwig Wittgenstein

PI #43:

"The meaning of a word is its use in the language"

Let's define words by their usages

One way to define "usage":

words are defined by their environments (the words around them)

Zellig Harris (1954):

If A and B have almost identical environments we say that they are synonyms.

What does recent English borrowing *ongchoi* mean?

Suppose you see these sentences:

- Ong choi is delicious **sautéed with garlic**.
- Ong choi is superb **over rice**
- Ong choi **leaves** with salty sauces

And you've also seen these:

- …spinach **sautéed with garlic over rice**
- Chard stems and **leaves** are **delicious**
- Collard greens and other **salty** leafy greens

Conclusion:

- Ongchoi is a leafy green like spinach, chard, or collard greens
  - We could conclude this based on words like "leaves" and "delicious" and "sauteed"

Ongchoi: *Ipomoea aquatica "Water Spinach"*

空心菜
kangkong
rau muống
…

Yamaguchi, Wikimedia Commons, public domain

Idea 1: Defining meaning by linguistic distribution

Let's define the meaning of a word by its distribution in language use, meaning its neighboring words or grammatical environments.

Idea 2: Meaning as a point in space (Osgood et al. 1957)

3 affective dimensions for a word

- **valence**: pleasantness
- **arousal**: intensity of emotion
- **dominance**: the degree of control exerted

	Word	Score		Word	Score
Valence	love	1.000		toxic	0.008
	happy	1.000		nightmare	0.005
Arousal	elated	0.960		mellow	0.069
	frenzy	0.965		napping	0.046
Dominance	powerful	0.991		weak	0.045
	leadership	0.983		empty	0.081

NRC VAD Lexicon (Mohammad 2018)

Hence the connotation of a word is a vector in 3-space

Idea 1: Defining meaning by linguistic distribution

Idea 2: Meaning as a point in multidimensional space

Defining meaning as a point in space based on distribution

Each word = a vector   (not just "good" or "w_{45}")

Similar words are "**nearby in semantic space**"

We build this space automatically by seeing which words are **nearby in text**

We define meaning of a word as a vector

Called an "embedding" because it's embedded into a space (see textbook)

The standard way to represent meaning in NLP

Every modern NLP algorithm uses embeddings as the representation of word meaning

Fine-grained model of meaning for similarity

Intuition: why vectors?

Consider sentiment analysis:

- With **words**, a feature is a word identity
  - Feature 5: 'The previous word was "terrible"'
  - requires **exact same word** to be in training and test
- With **embeddings**:
  - Feature is a word vector
  - 'The previous word was vector [35,22,17…]
  - Now in the test set we might see a similar vector [34,21,14]
  - We can generalize to **similar but unseen** words!!!

We'll discuss 2 kinds of embeddings

tf-idf

- Information Retrieval workhorse!
- A common baseline model
- **Sparse** vectors
- Words are represented by (a simple function of) the **counts** of nearby words

Word2vec

- **Dense** vectors
- Representation is created by training a classifier to **predict** whether a word is likely to appear nearby
- Later we'll discuss extensions called **contextual embeddings**

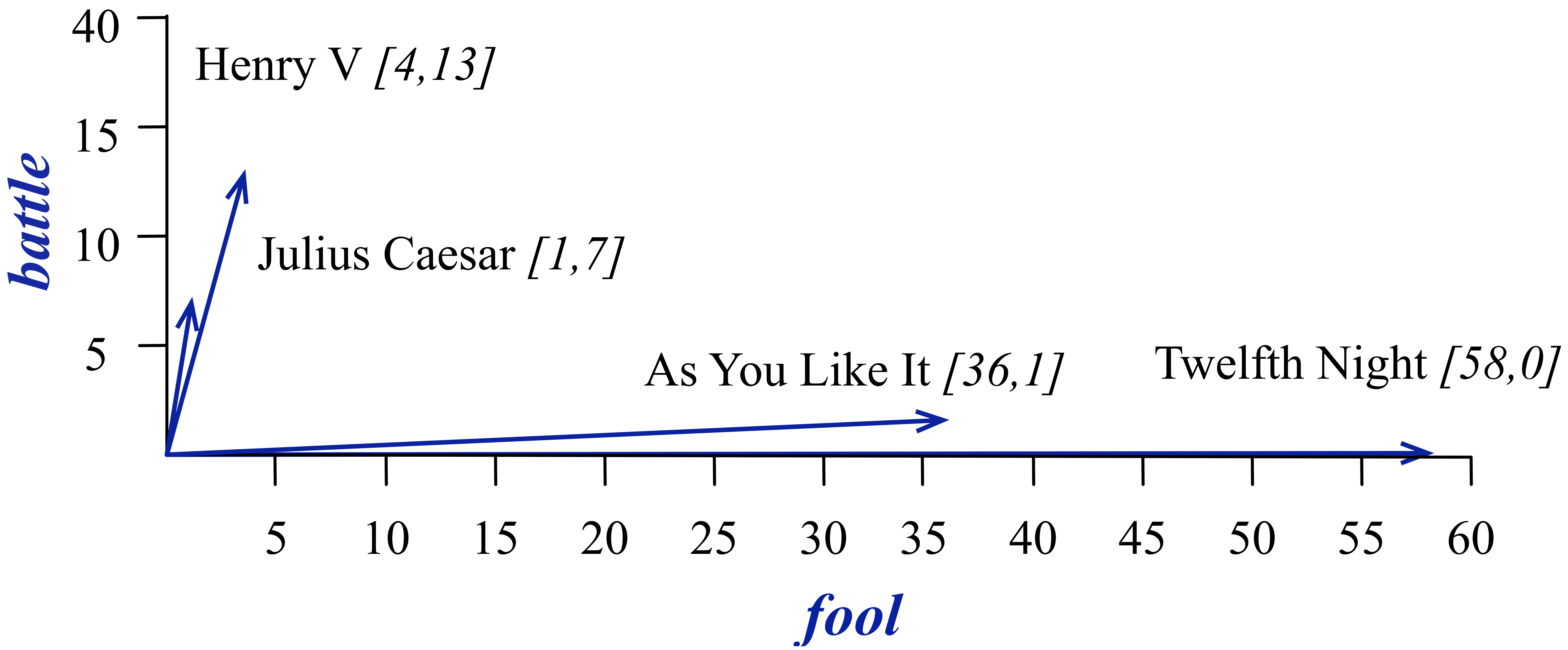
Words and vectors

Term-document matrix

Each document is represented by a vector of words

	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

Visualizing document vectors



Vectors are the basis of information retrieval

	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

Vectors are similar for the two comedies

But comedies are different than the other two

Comedies have more *fools* and *wit* and fewer *battles*.

Idea for word meaning: Words can be vectors too!!!

	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

battle is "the kind of word that occurs in Julius Caesar and Henry V"

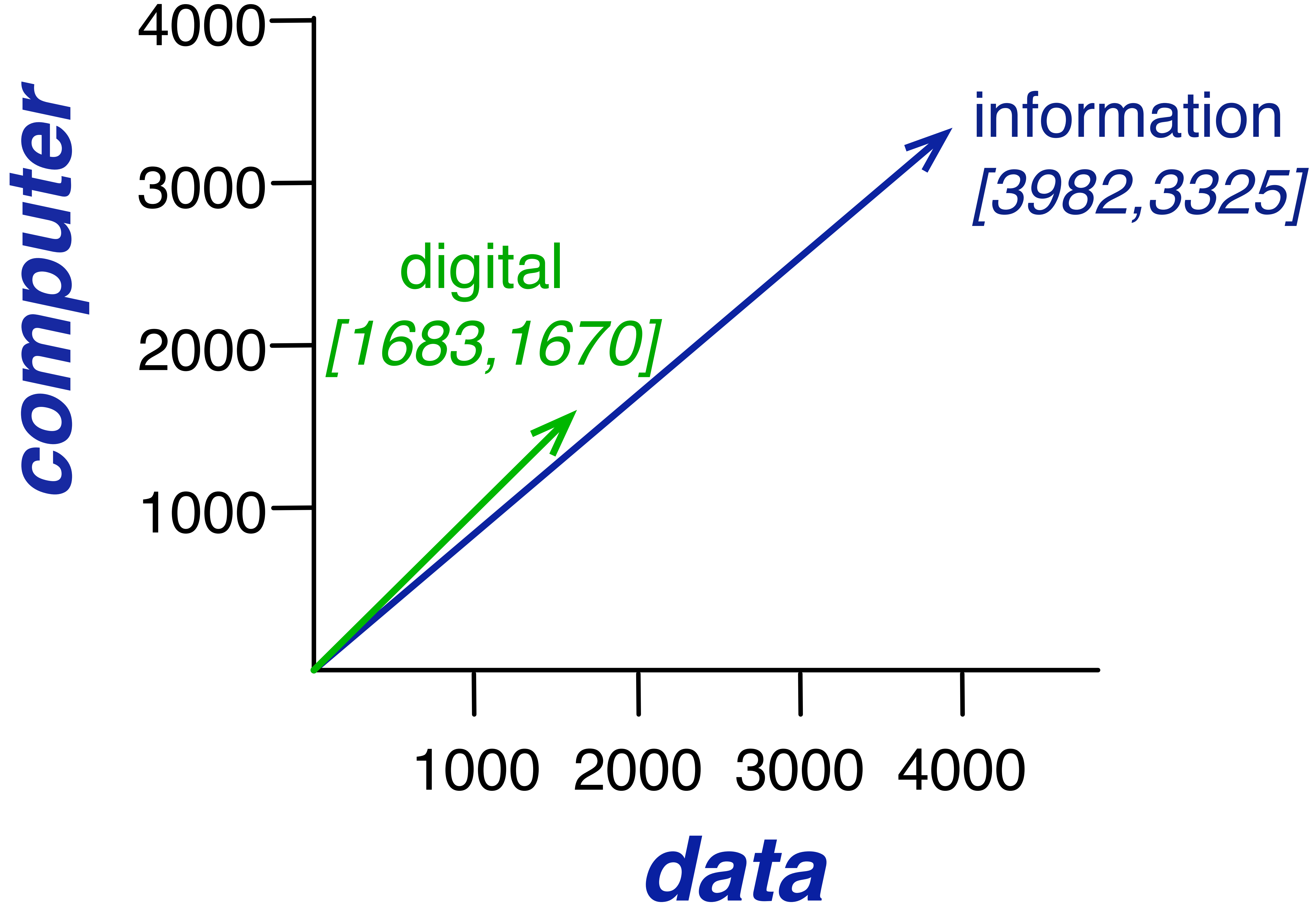
fool is "the kind of word that occurs in comedies, especially Twelfth Night"

More common: word-word matrix (or "term-context matrix")

Two **words** are similar in meaning if their context vectors are similar

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

	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	...


4000
3000
2000
1000
computer
information
[3982,3325]
digital
[1683,1670]
1000
2000
3000
4000
data

Cosine for computing word similarity

Computing word similarity: Dot product and cosine

The dot product between two vectors is a scalar:

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

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

Dot product can thus be a useful similarity metric between vectors

Problem with raw dot-product

Dot product favors long vectors

Dot product is higher if a vector is longer (has higher values in many dimension)

Vector length:

$$|\mathbf{v}| = \sqrt{\sum_{i=1}^{N} v_i^2}$$

Frequent words (of, the, you) have long vectors (since they occur many times with other words).

So dot product overly favors frequent words

Alternative: cosine for computing word similarity

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

Based on the definition of the dot product between two vectors a and b

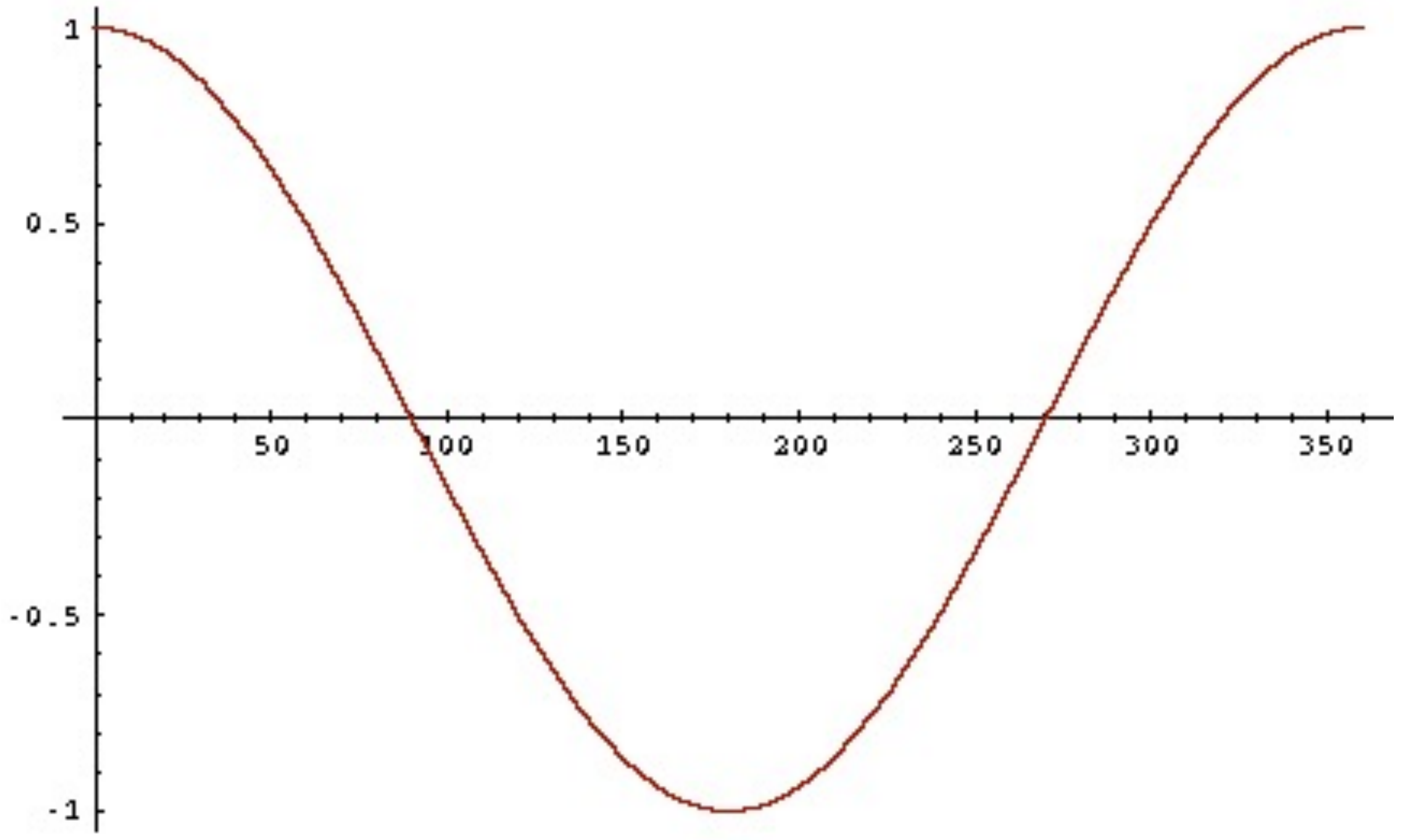
$$\begin{aligned} \mathbf{a}\cdot\mathbf{b} &= |\mathbf{a}||\mathbf{b}|\cos\theta \\ \frac{\mathbf{a}\cdot\mathbf{b}}{|\mathbf{a}||\mathbf{b}|} &= \cos\theta \end{aligned}$$

Cosine as a similarity metric

-1: vectors point in opposite directions

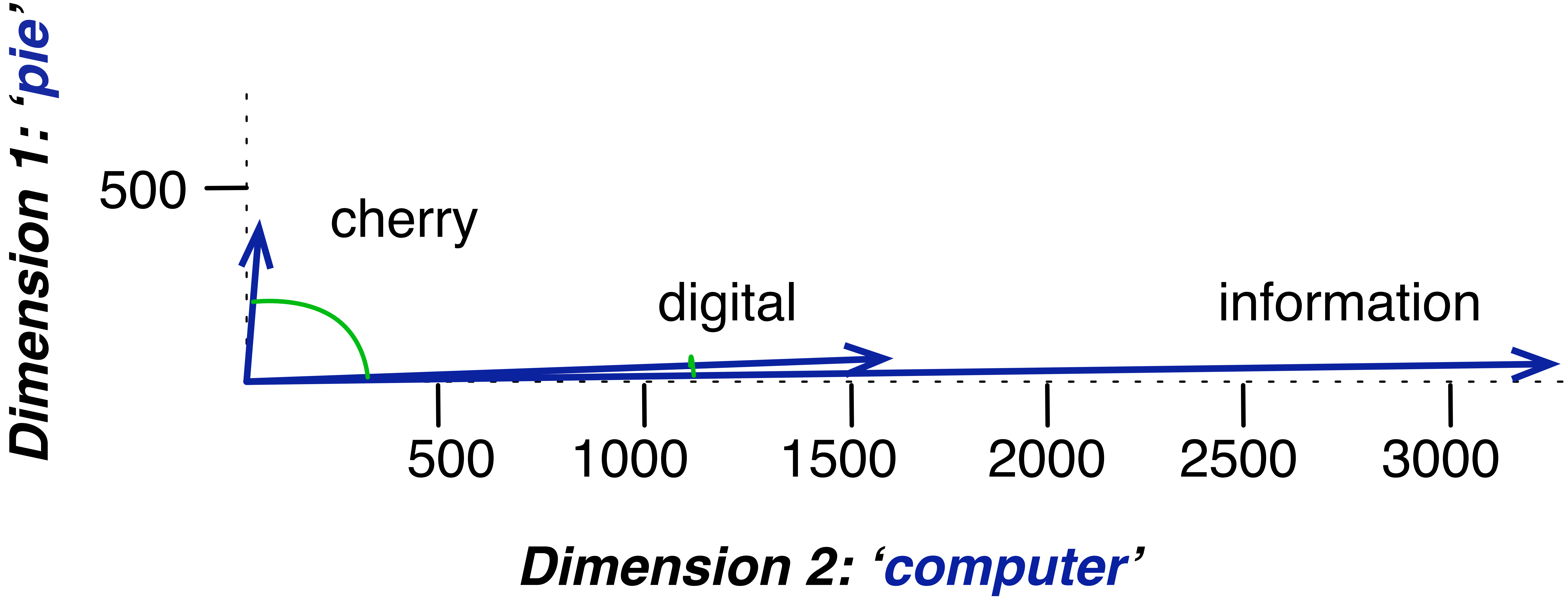
+1: vectors point in same directions

0: vectors are orthogonal



But since raw frequency values are non-negative, the cosine for term-term matrix vectors ranges from 0–1

Visualizing cosines (well, angles)



tf-idf

But raw frequency is a bad representation

- The co-occurrence matrices we have seen represent each cell by word frequencies.
- Frequency is clearly useful; if *sugar* appears a lot near *apricot*, that's useful information.
- But overly frequent words like *the*, *it,* or *they* are not very informative about the context
- It's a paradox! How can we balance these two conflicting constraints?

Two common solutions for word weighting

tf-idf: tf-idf value for word t in document d:

$$w_{t,d} = \mathrm{tf}_{t,d} \times \mathrm{idf}_t$$

Words like "the" or "it" have very low idf

PMI: (Pointwise mutual information)

- $\text{PMI}(w_1, w_2) = log \frac{p(w_1,w_2)}{p(w_1)p(w_2)}$

See if words like "good" appear more often with "great" than we would expect by chance

Term frequency (tf)

$\mathrm{tf}_{t,d} = \mathrm{count}(t,d)$

Instead of using raw count, we squash a bit:

$\mathrm{tf}_{t,d} = \log_{10}(\mathrm{count}(t,d)+1)$

Document frequency (df)

df_t *is* the number of documents *t* occurs in.

(note this is not collection frequency: total count across all documents)

"*Romeo*" is very distinctive for one Shakespeare play:

	Collection Frequency	**Document Frequency**
Romeo	113	1
action	113	31

Inverse document frequency (idf)

$$\text{idf}_t = \log_{10}\left(\frac{N}{\text{df}_t}\right)$$

N is the total number of documents in the collection

Word	**df**	**idf**
Romeo	1	1.57
salad	2	1.27
Falstaff	4	0.967
forest	12	0.489
battle	21	0.246
wit	34	0.037
fool	36	0.012
good	37	0
sweet	37	0

Final tf-idf weighted value for a word

$$w_{t,d} = \mathrm{tf}_{t,d} \times \mathrm{idf}_t$$

Raw counts:

	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

tf-idf:

	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	0.074	0	0.22	0.28
good	0	0	0	0
fool	0.019	0.021	0.0036	0.0083
wit	0.049	0.044	0.018	0.022

PPMI

Pointwise Mutual Information

Pointwise mutual information:
Do events x and y co-occur more than if they were independent?

$$\text{PMI}(X,Y) = \log_2 \frac{P(x,y)}{P(x)P(y)}$$

PMI between two words: (Church & Hanks 1989)
Do words x and y co-occur more than if they were independent?

$$\text{PMI}(word_1, word_2) = \log_2 \frac{P(word_1, word_2)}{P(word_1)P(word_2)}$$

Positive Pointwise Mutual Information

- PMI ranges from $-\infty$ to $+\infty$
- But the negative values are problematic
  - Things are co-occurring **less than** we expect by chance
  - Unreliable without enormous corpora
    - Imagine w1 and w2 whose probability is each 10^{-6}
    - Hard to be sure p(w1,w2) is significantly different than 10^{-12}
  - Plus it's not clear people are good at "unrelatedness"
- So we just replace negative PMI values by 0
- Positive PMI (**PPMI**) between word1 and word2:

$$\text{PPMI}(word_1, word_2) = \max\left(\log_2 \frac{P(word_1, word_2)}{P(word_1)P(word_2)}, 0\right)$$

Computing PPMI on a term-context matrix

Matrix F with W rows (words) and C columns (contexts)

f_{ij} is # of times w_i occurs in context c_j

$$p_{ij} = \frac{f_{ij}}{\sum_{i=1}^{W}\sum_{j=1}^{C} f_{ij}} \qquad p_{i*} = \frac{\sum_{j=1}^{C} f_{ij}}{\sum_{i=1}^{W}\sum_{j=1}^{C} f_{ij}} \qquad p_{*j} = \frac{\sum_{i=1}^{W} f_{ij}}{\sum_{i=1}^{W}\sum_{j=1}^{C} f_{ij}}$$

	computer	data	result	pie	sugar	count(w)
cherry	2	8	9	442	25	486
strawberry	0	0	1	60	19	80
digital	1670	1683	85	5	4	3447
information	3325	3982	378	5	13	7703
count(context)	4997	5673	473	512	61	11716

$$pmi_{ij} = \log_2 \frac{p_{ij}}{p_{i*}p_{*j}} \qquad ppmi_{ij} = \begin{cases} pmi_{ij} & \text{if } pmi_{ij} > 0 \\ 0 & \text{otherwise} \end{cases}$$

$$p_{ij} = \frac{f_{ij}}{\sum_{i=1}^{W} \sum_{j=1}^{C} f_{ij}}$$

	computer	data	result	pie	sugar	count(w)
cherry	2	8	9	442	25	486
strawberry	0	0	1	60	19	80
digital	1670	1683	85	5	4	3447
information	3325	3982	378	5	13	7703
count(context)	4997	5673	473	512	61	11716

p(w=information,c=data) = 3982/111716 = .3399

p(w=information) = 7703/11716 = .6575

p(c=data) = 5673/11716 = .4842

$$p(w_i) = \frac{\sum_{j=1}^{C} f_{ij}}{N} \qquad p(c_j) = \frac{\sum_{i=1}^{W} f_{ij}}{N}$$

	p(w,context)					p(w)
	computer	**data**	**result**	**pie**	**sugar**	**p(w)**
cherry	0.0002	0.0007	0.0008	0.0377	0.0021	0.0415
strawberry	0.0000	0.0000	0.0001	0.0051	0.0016	0.0068
digital	0.1425	0.1436	0.0073	0.0004	0.0003	0.2942
information	0.2838	0.3399	0.0323	0.0004	0.0011	0.6575
p(context)	0.4265	0.4842	0.0404	0.0437	0.0052	

$$pmi_{ij} = \log_2 \frac{p_{ij}}{p_{i*}p_{*j}}$$

	p(w,context)					p(w)
	computer	data	result	pie	sugar	p(w)
cherry	0.0002	0.0007	0.0008	0.0377	0.0021	0.0415
strawberry	0.0000	0.0000	0.0001	0.0051	0.0016	0.0068
digital	0.1425	0.1436	0.0073	0.0004	0.0003	0.2942
information	0.2838	0.3399	0.0323	0.0004	0.0011	0.6575
p(context)	0.4265	0.4842	0.0404	0.0437	0.0052	

pmi(information,data) = $\log_2$ (.3399 / (.6575*.4842)) = .0944

Resulting PPMI matrix (negatives replaced by 0)

	computer	data	result	pie	sugar
cherry	0	0	0	4.38	3.30
strawberry	0	0	0	4.10	5.51
digital	0.18	0.01	0	0	0
information	0.02	0.09	0.28	0	0

Weighting PMI

PMI is biased toward infrequent events

- Very rare words have very high PMI values

Two solutions:

- Give rare words slightly higher probabilities
- Use add-one smoothing (which has a similar effect)

Weighting PMI: Giving rare context words slightly higher probability

Raise the context probabilities to $\alpha = 0.75$:

$$\mathrm{PPMI}_\alpha(w,c) = \max(\log_2 \frac{P(w,c)}{P(w)P_\alpha(c)}, 0)$$

$$P_\alpha(c) = \frac{count(c)^\alpha}{\sum_c count(c)^\alpha}$$

This helps because $P_\alpha(c) > P(c)$ for rare *c*

Consider two events, P(a) = .99 and P(b)=.01

$$P_\alpha(a) = \frac{.99^{.75}}{.99^{.75} + .01^{.75}} = .97 \quad P_\alpha(b) = \frac{.01^{.75}}{.01^{.75} + .01^{.75}} = .03$$

word2vec

Sparse versus dense vectors

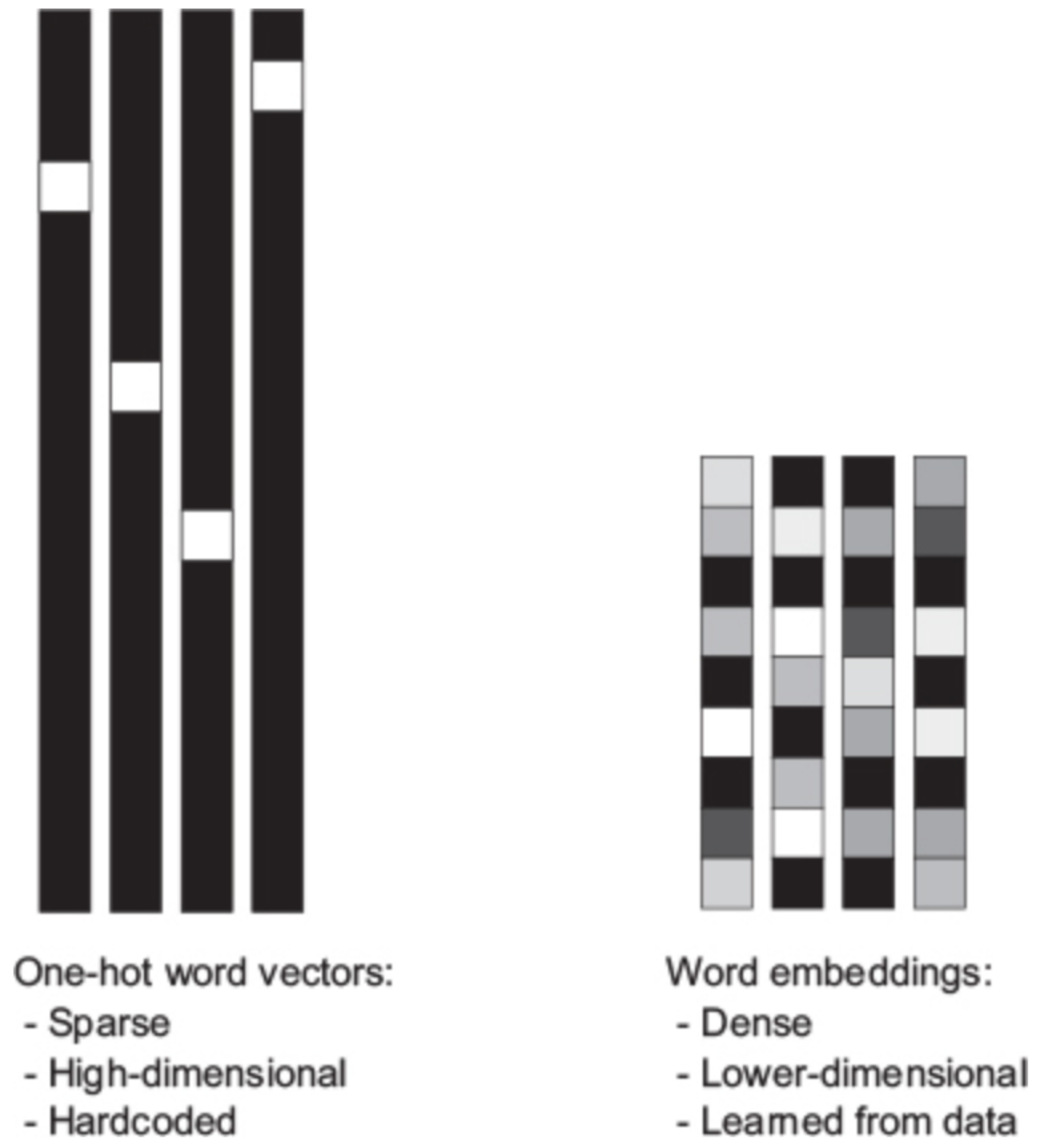
tf-idf (or PMI) vectors are

- **long** (length |V|= 20,000 to 50,000)
- **sparse** (most elements are zero)

Alternative: learn vectors which are

- **short** (length 50-1000)
- **dense** (most elements are non-zero)

Figure 6.2. Whereas word representations obtained from one-hot encoding or hashing are sparse, high-dimensional, and hardcoded, word embeddings are dense, relatively low-dimensional, and learned from data.



Sparse versus dense vectors

Why dense vectors?

- Short vectors may be easier to use as **features** in machine learning (fewer weights to tune)
- Dense vectors may **generalize** better than explicit counts
- Dense vectors may do better at capturing synonymy:
  - *car* and *automobile* are synonyms; but are distinct dimensions
    - a word with *car* as a neighbor and a word with *automobile* as a neighbor should be similar, but aren't
- **In practice, they work better**

Common methods for getting short dense vectors

"Neural Language Model"-inspired models

- Word2vec (skipgram, CBOW), GloVe

Singular Value Decomposition (SVD)

- A special case of this is called LSA – Latent Semantic Analysis

Alternative to these "static embeddings":

- Contextual Embeddings (ELMo, BERT)
- Compute distinct embeddings for a word in its context
- Separate embeddings for each token of a word

Simple static embeddings you can download!

Word2vec (Mikolov et al)

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

GloVe (Pennington, Socher, Manning)

http://nlp.stanford.edu/projects/glove/

Word2vec

Popular embedding method

Very fast to train

Code available on the web

Idea: **predict** rather than **count**

Word2vec provides various options. We'll do:

skip-gram with negative sampling (SGNS)

Word2vec

Instead of **counting** how often each word *w* occurs near "*apricot*"

- Train a classifier on a binary **prediction** task:
  - Is *w* likely to show up near "*apricot*"?

We don’t actually care about this task

- But we'll take the learned classifier weights as the word embeddings

Big idea: **self-supervision**:

- A word c that occurs near apricot in the corpus cats as the gold "correct answer" for supervised learning
- No need for human labels
- Bengio et al. (2003); Collobert et al. (2011)

If one wanted to give an example of an NLP application, one of the best examples would be the next-word prediction feature of a smartphone keyboard. It's a feature that billions of people use hundreds of times every day.

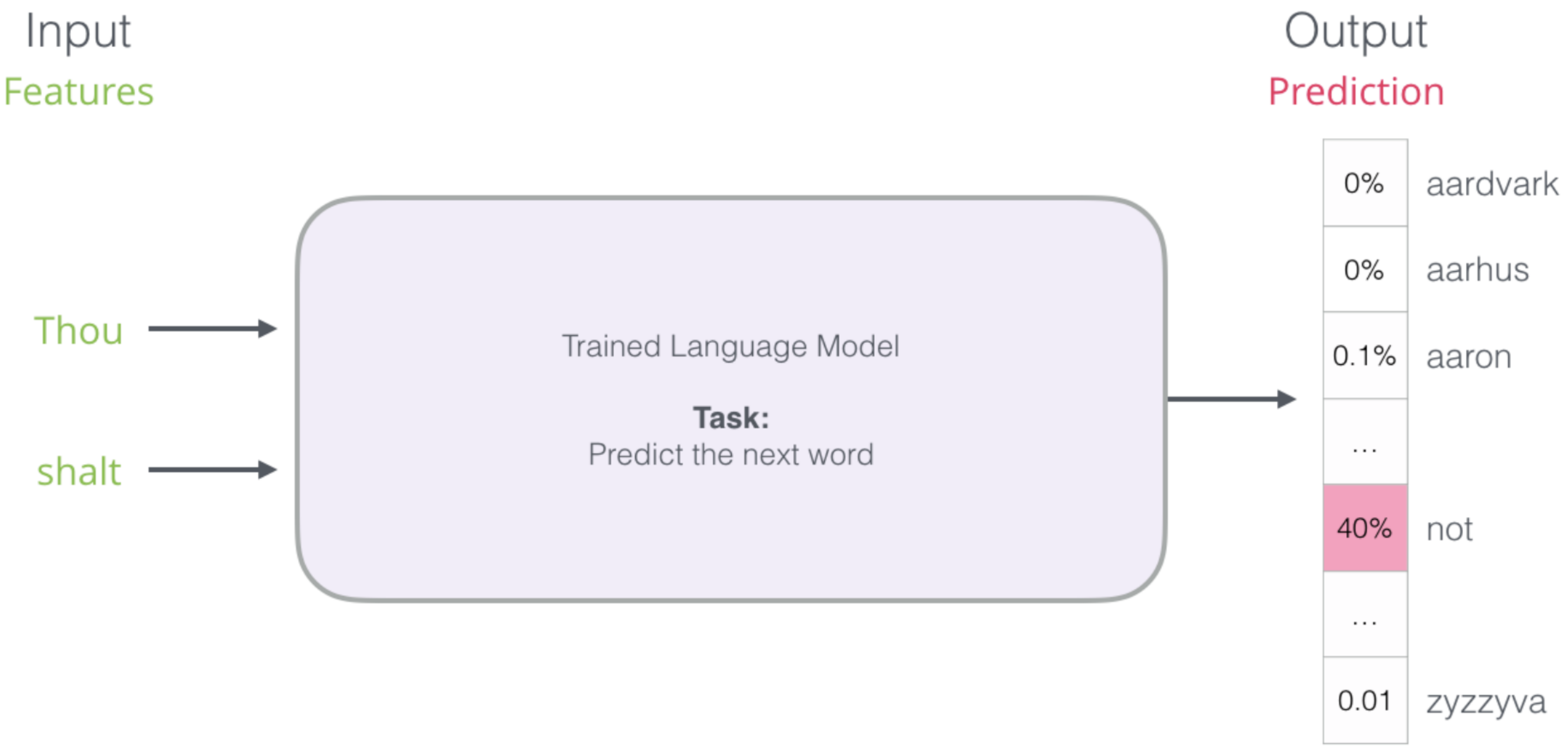
Next-word prediction is a task that can be addressed by a *language model*. A language model can take a list of words (let's say two words), and attempt to predict the word that follows them.

In the screenshot above, we can think of the model as one that took in these two green words (`thou shalt`) and returned a list of suggestions ("not" being the one with the highest probability):

input/feature #1 | input/feature #2 | output/label

Thou shalt ____

Source: Jay Alammar, 2019, "The Illustrated word2vec



The output of the neural language model is a probability score for all the words the model knows. We're referring to the probability as a percentage here, but 40% would actually be represented as 0.4 in the output vector.

Source: Jay Alammar, 2019, “The Illustrated word2vec

Thou shalt not make a machine in the likeness of a human mind

Sliding window across running text

thou	shalt	not	make	a	machine	in	the	…

Dataset

input 1	input 2	output
thou	shalt	not

Thou shalt not make a machine in the likeness of a human mind

Sliding window across running text

thou	shalt	not	make	a	machine	in	the	…
thou	shalt	not	make	a	machine	in	the	

Dataset

input 1	input 2	output
thou	shalt	not
shalt	not	make

Thou shalt not make a machine in the likeness of a human mind

Sliding window across running text

thou	shalt	not	make	a	machine	in	the	…
thou	shalt	not	make	a	machine	in	the	
thou	shalt	not	make	a	machine	in	the	
thou	shalt	not	make	a	machine	in	the	
thou	shalt	not	make	a	machine	in	the	

Dataset

input 1	input 2	output
thou	shalt	not
shalt	not	make
not	make	a
make	a	machine
a	machine	in

Source: Jay Alammar, 2019, “The Illustrated word2

Jay was hit by a ______

Source: Jay Alammar, 2019, “The Illustrated word2vec

Jay was hit by a ______ bus

Source: Jay Alammar, 2019, “The Illustrated word2vec

word2vec - skipgram with negative sampling (SGNS)

Thou shalt not make a machine in the likeness of a human mind

thou	shalt	not	make	a	machine	in	the	…

input word	target word
not	thou
not	shalt
not	make
not	a

skipgrams

Thou shalt not make a machine in the likeness of a human mind

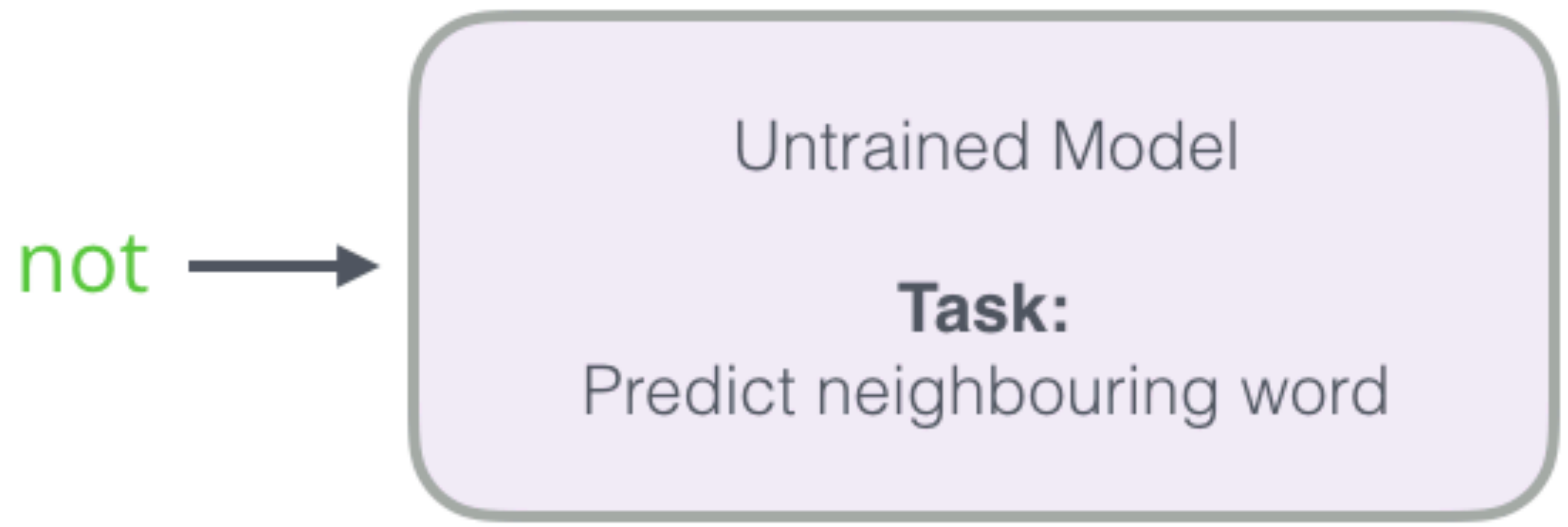
thou	shalt	not	make	a	machine	in	the	…

thou	shalt	not	make	a	machine	in	the	…

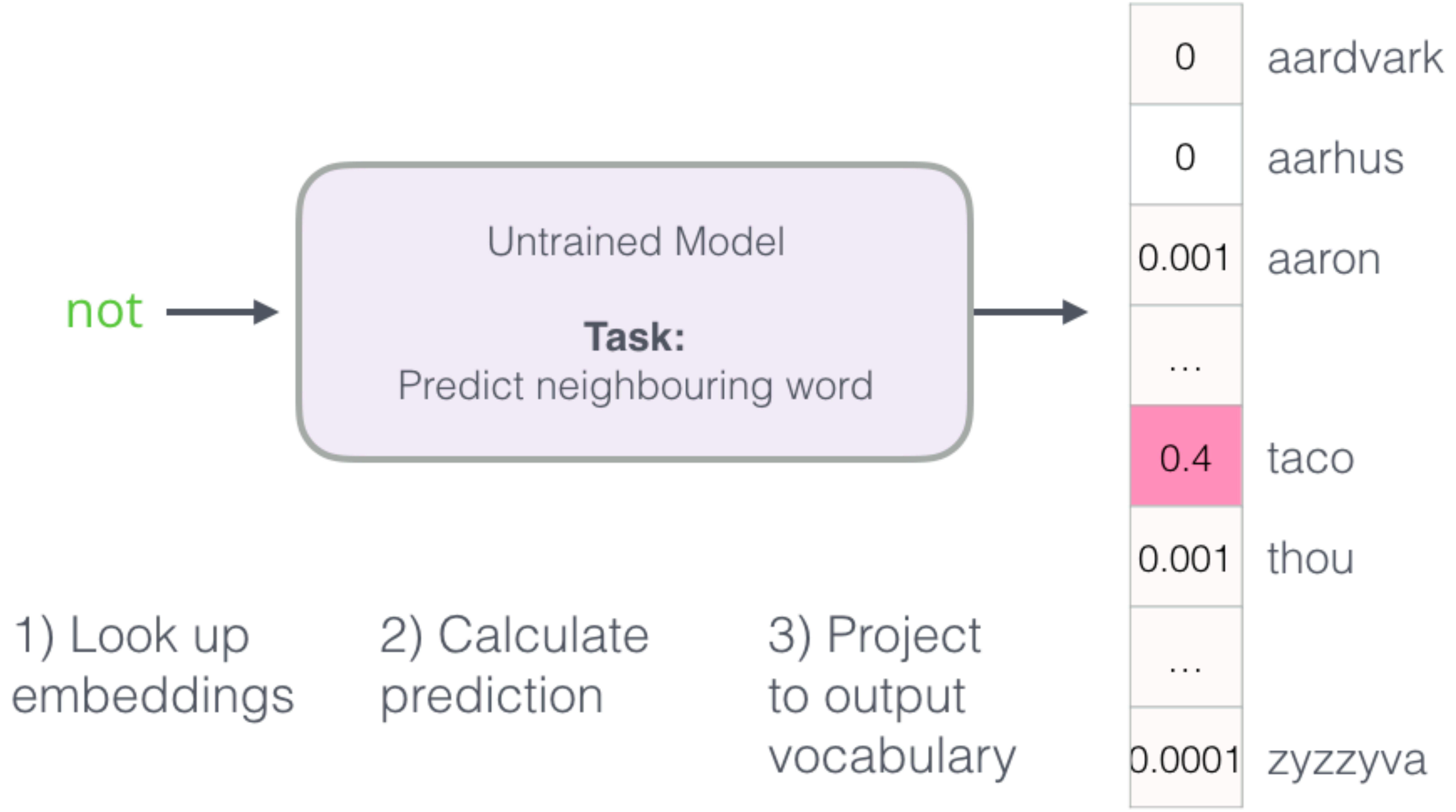
input word	target word
not	thou
not	shalt
not	make
not	a
make	shalt
make	not
make	a
make	machine

Source: Jay Alammar, 2019, "The Illustrated word2vec

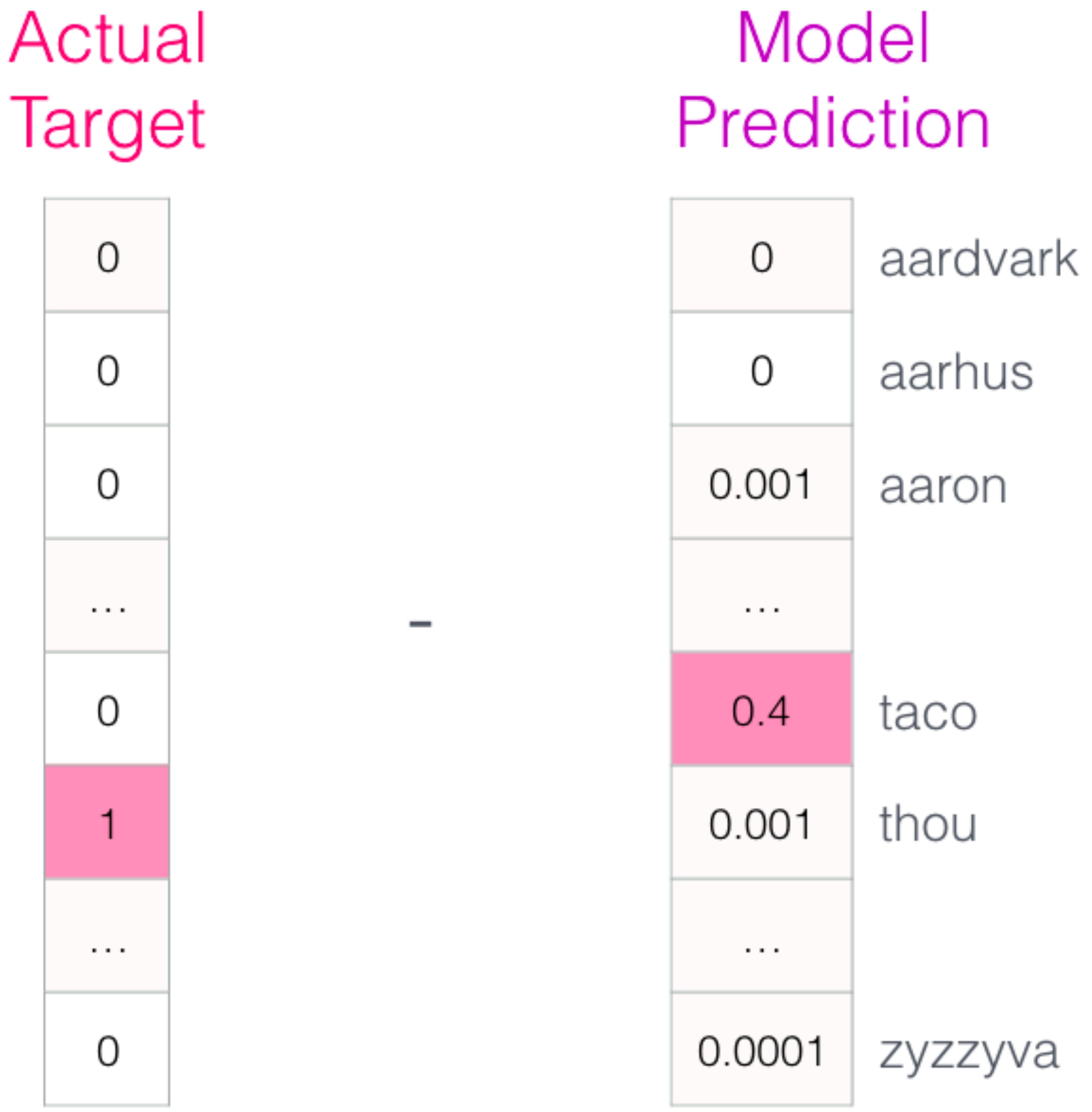
input word	target word
not	thou
not	shalt
not	make
not	a
make	shalt
make	not
make	a
make	machine
a	not
a	make
a	machine
a	in
machine	make
machine	a
machine	in
machine	the
in	a
in	machine
in	the
in	likeness



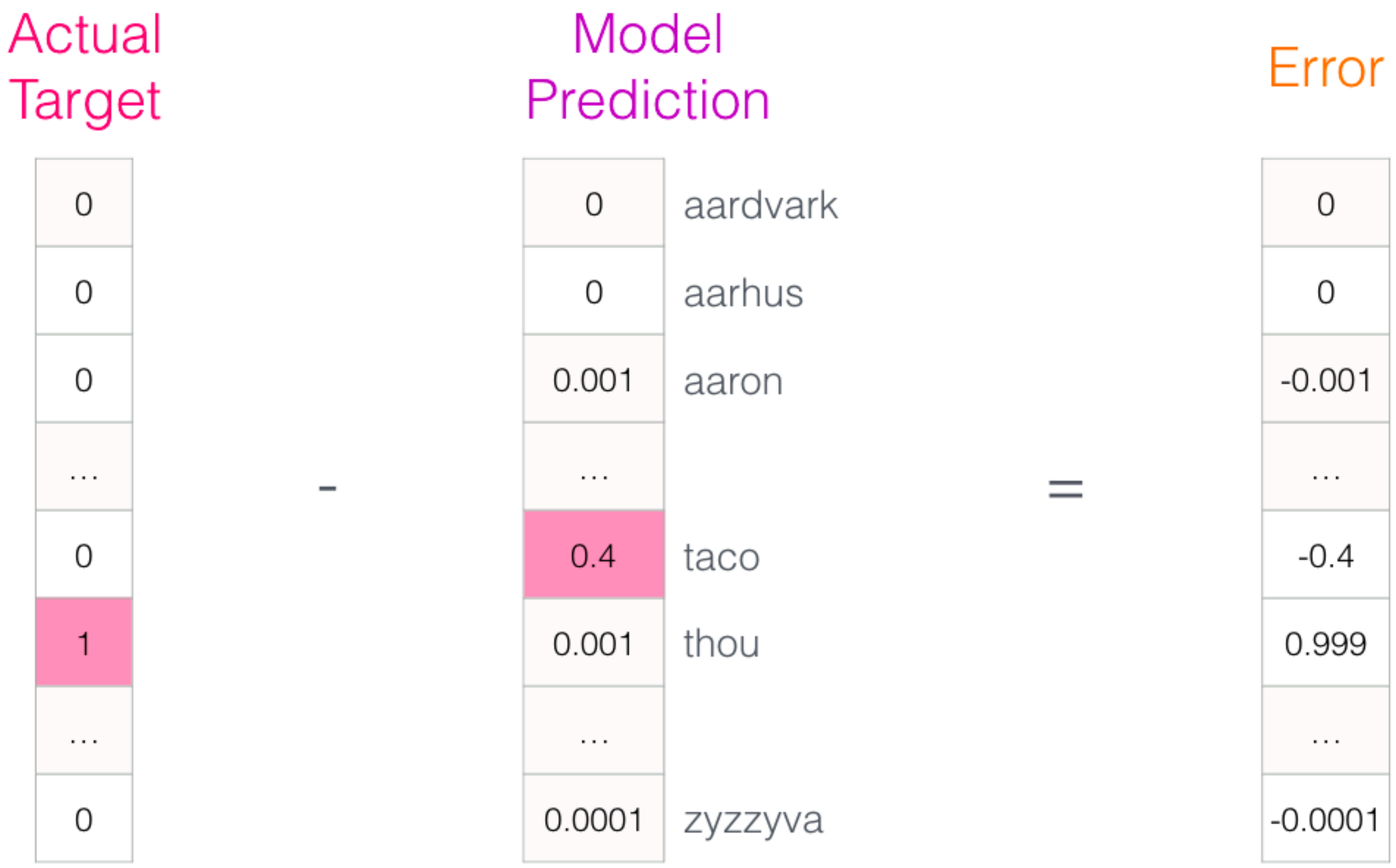
Source: Jay Alammar, 2019, “The Illustrated word2vec



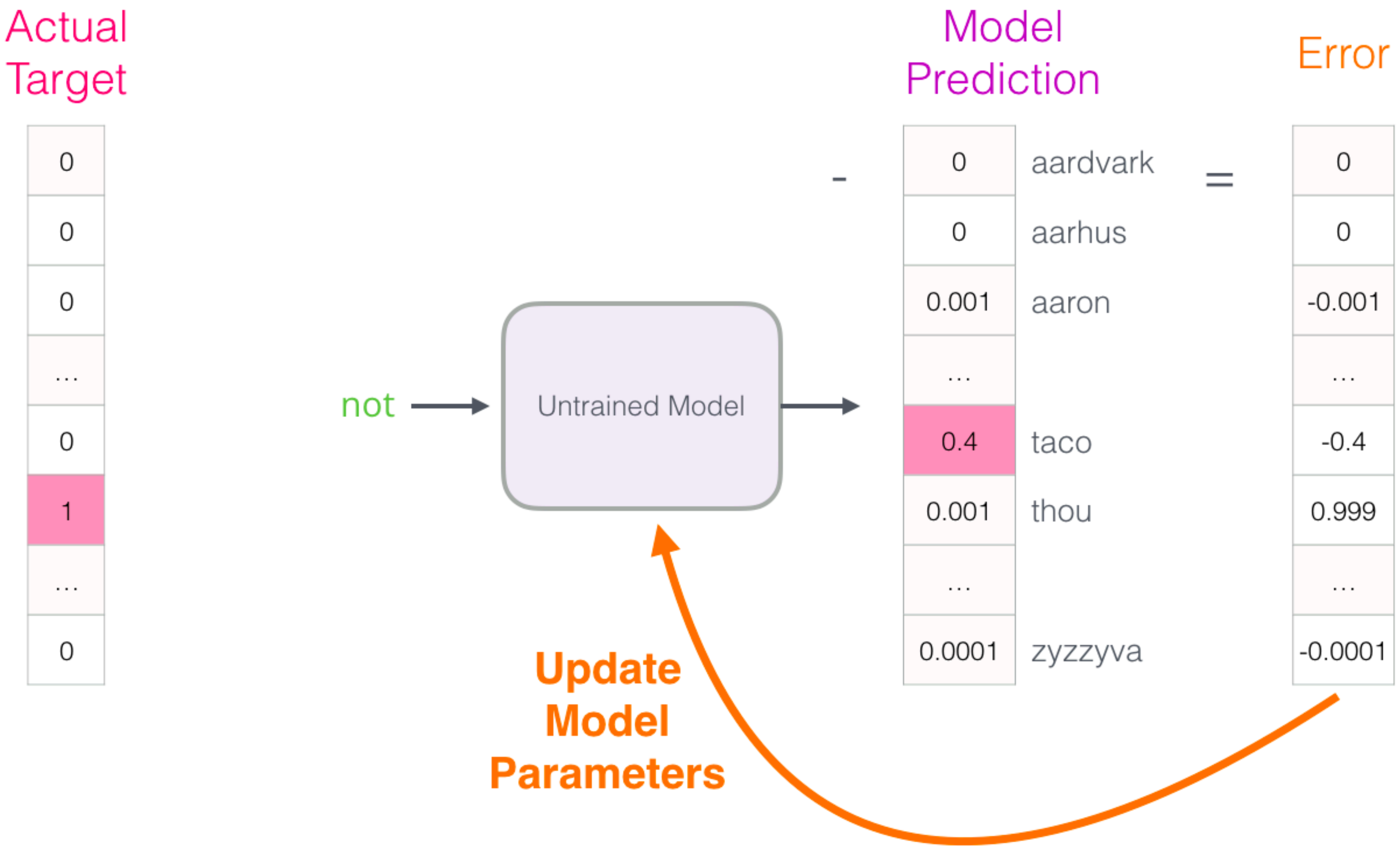
Source: Jay Alammar, 2019, “The Illustrated word2vec”



Source: Jay Alammar, 2019, “The Illustrated word2vec

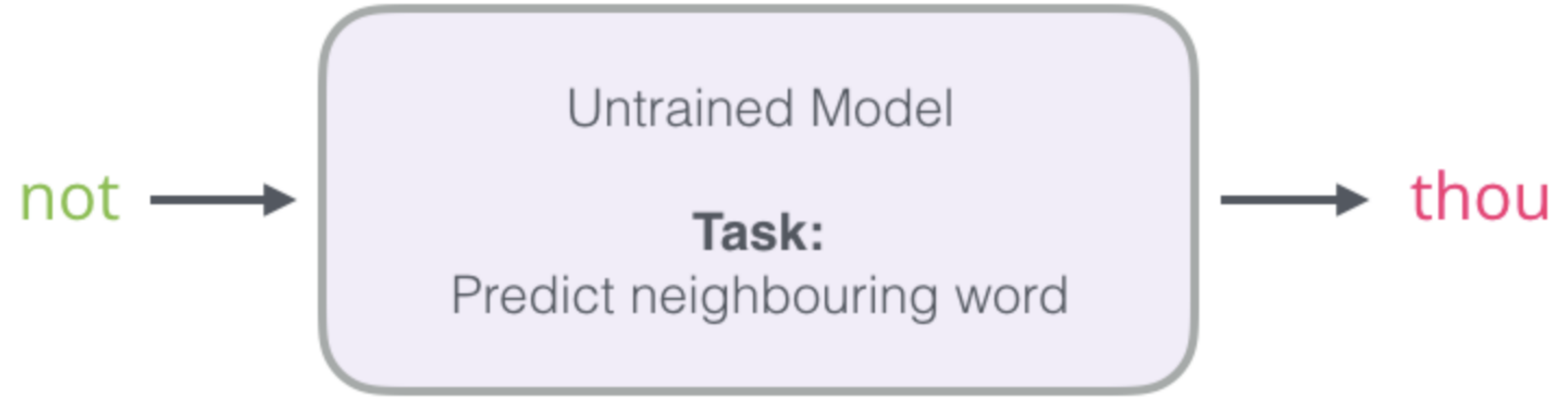


Source: Jay Alammar, 2019, “The Illustrated word2vec



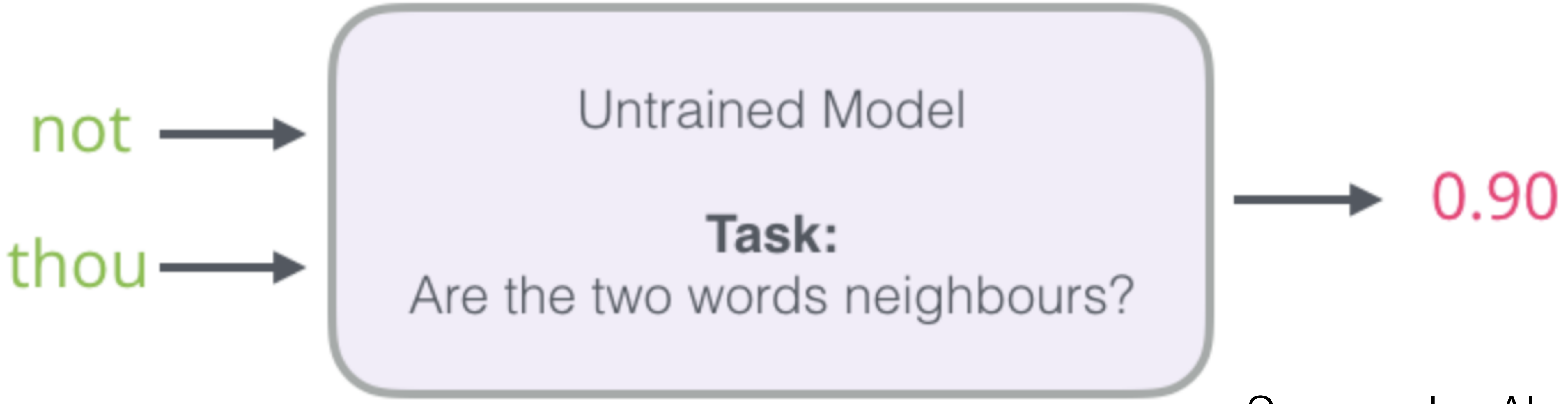
Source: Jay Alammar, 2019, “The Illustrated word2vec

Change Task from



And switch it to a model that takes the input and output word, and outputs a score indicating if they’re neighbors or not (0 for “not neighbors”, 1 for “neighbors”).

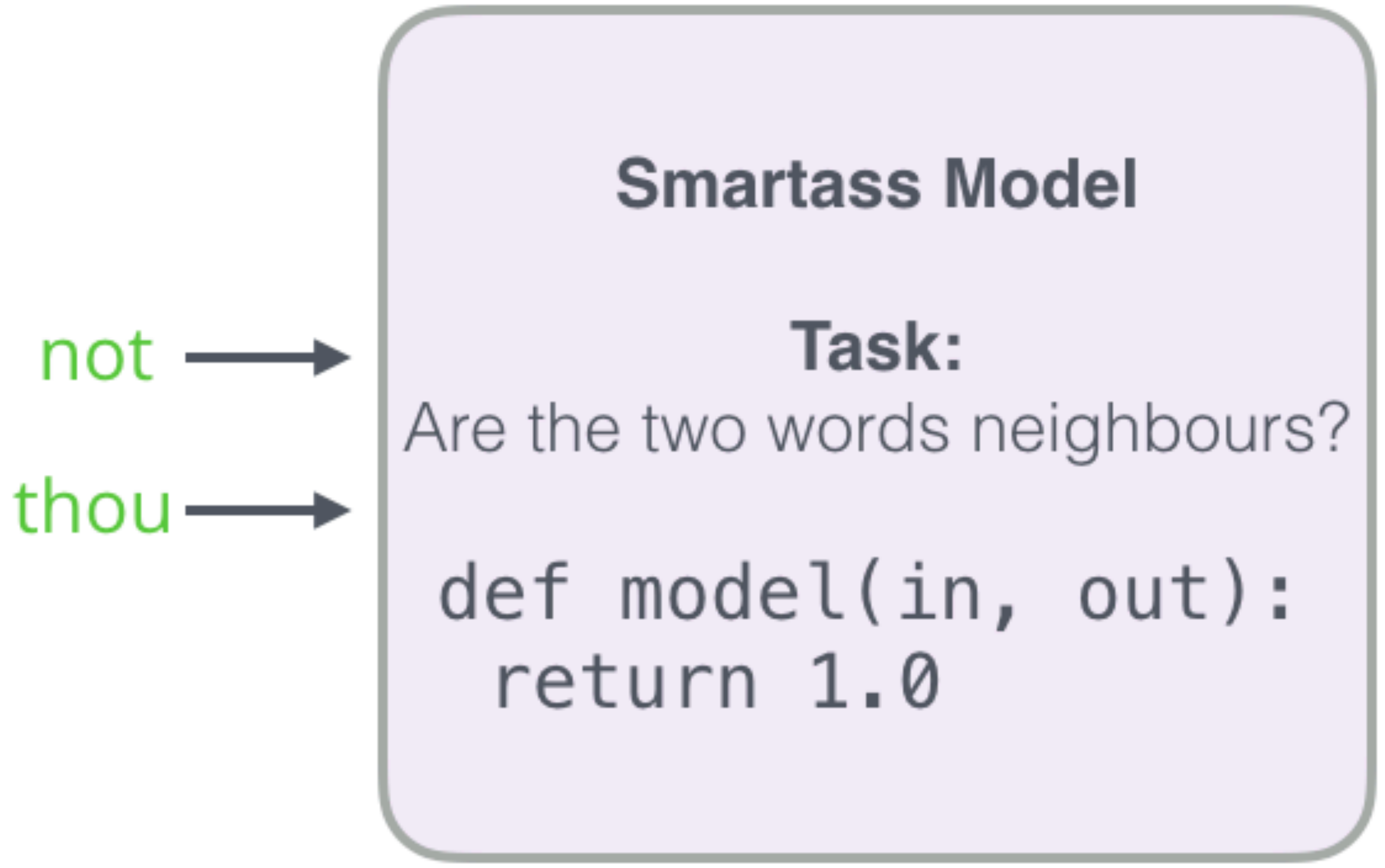
To:



Source: Jay Alammar, 2019, “The Illustrated word2ve

input word	target word
not	thou
not	shalt
not	make
not	a
make	shalt
make	not
make	a
make	machine

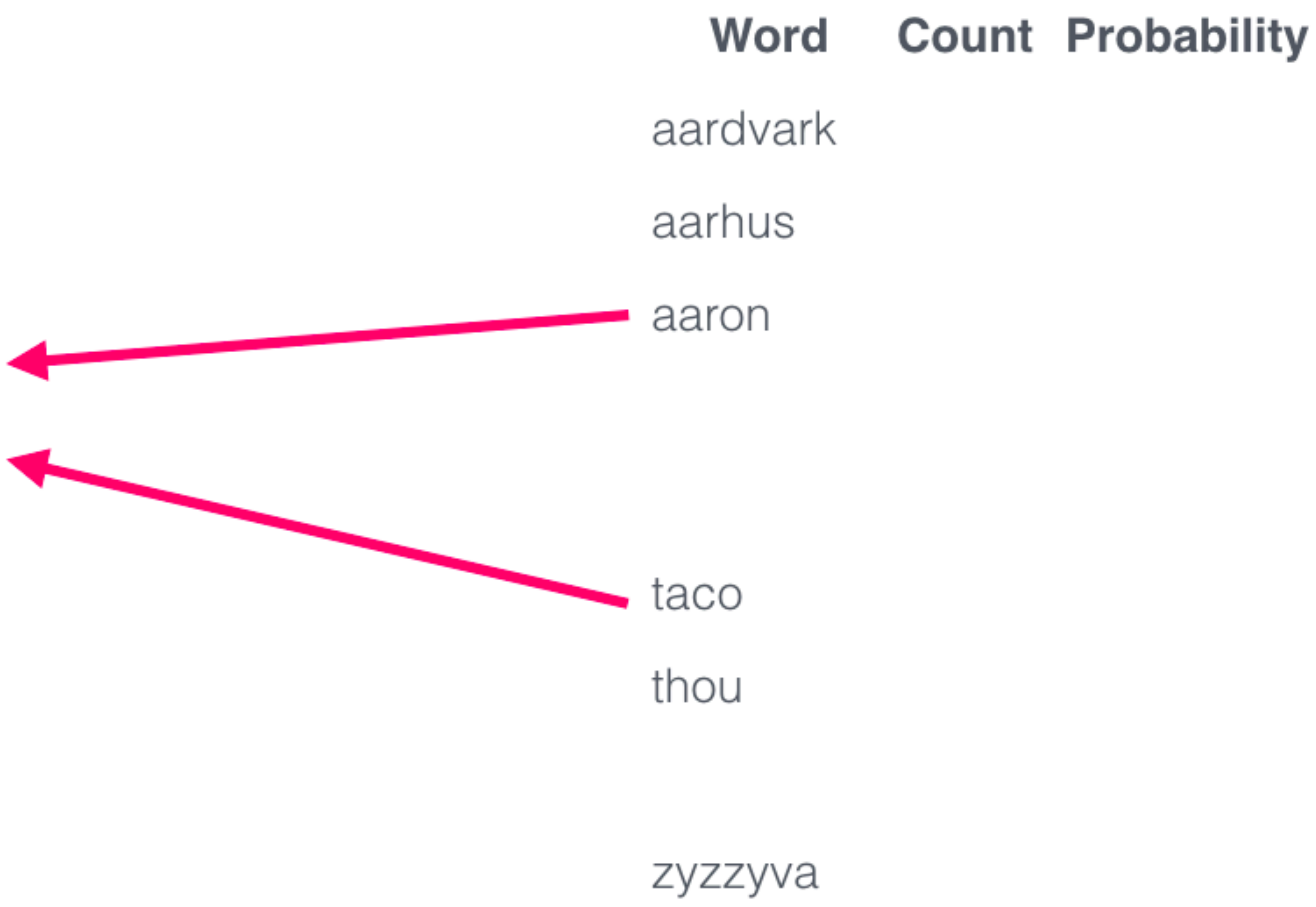
input word	output word	target
not	thou	1
not	shalt	1
not	make	1
not	a	1
make	shalt	1
make	not	1
make	a	1
make	machine	1



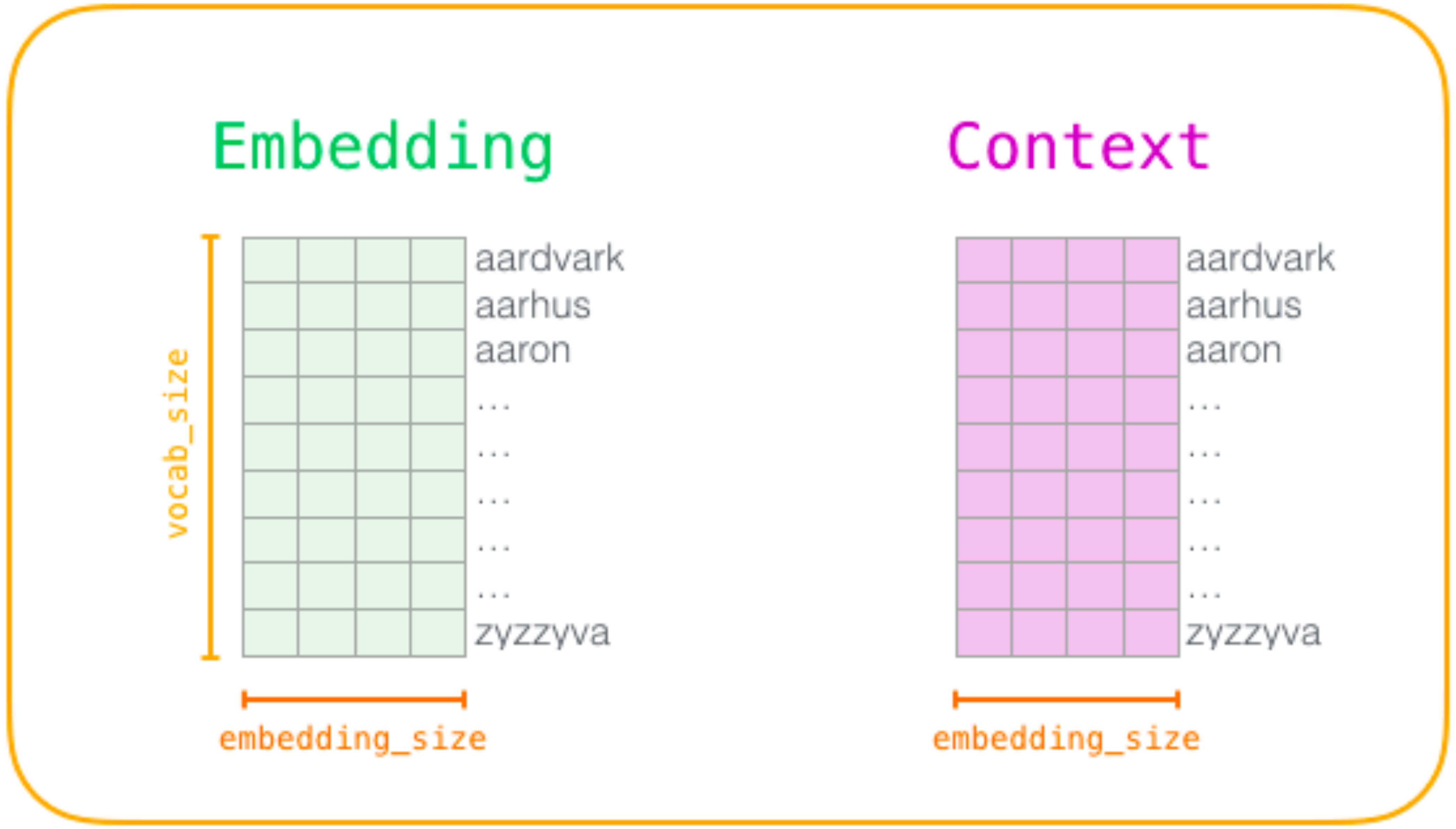
Source: Jay Alammar, 2019, “The Illustrated word2v

input word	output word	target
not	thou	1
not	aaron	0
not	taco	0
not	shalt	1
not	make	1

Pick randomly from vocabulary
(random sampling)



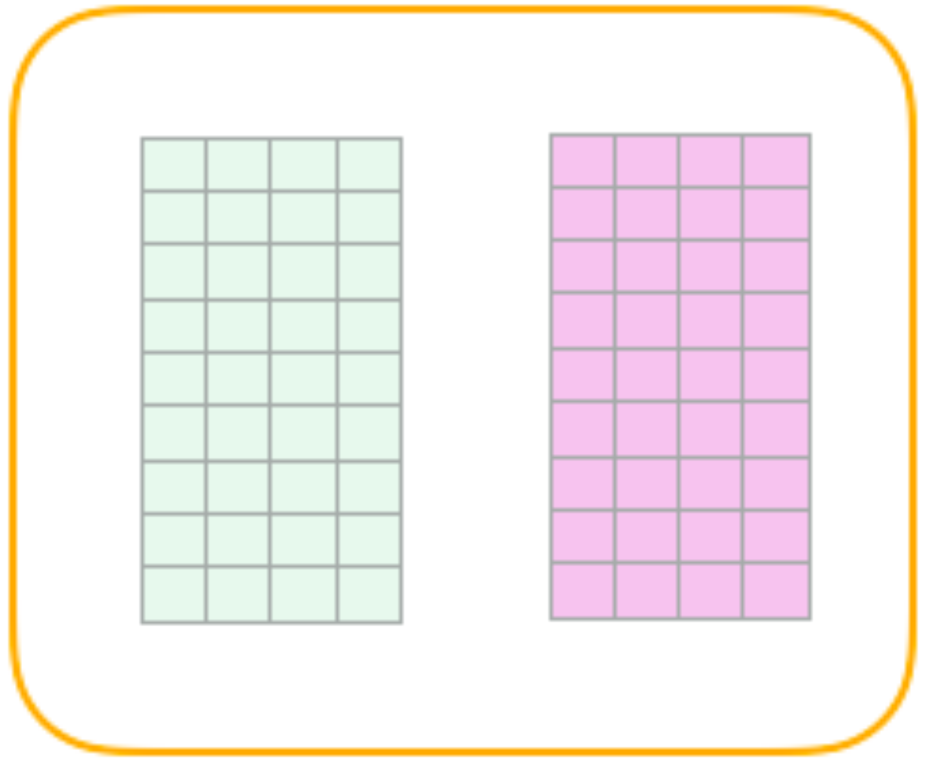
Source: Jay Alammar, 2019, "The Illustrated word2vec"



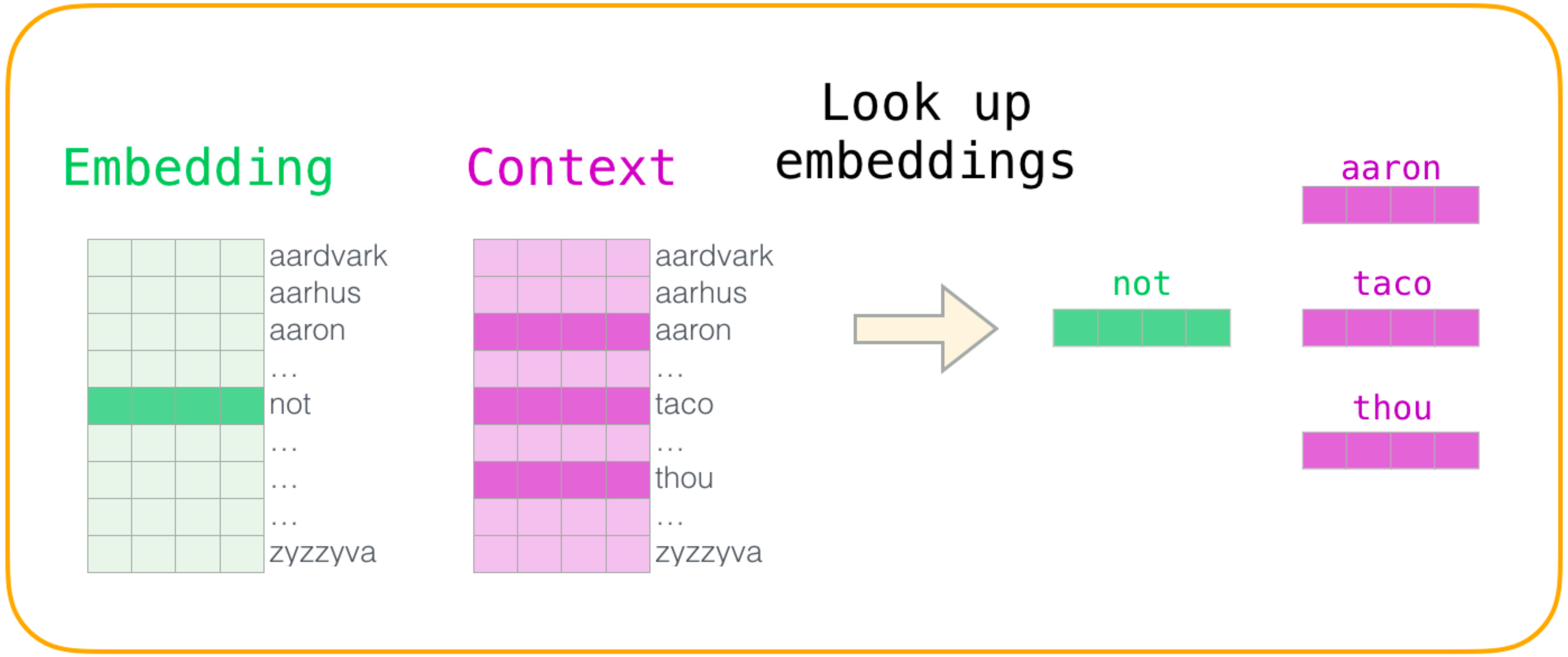
dataset

input word	output word	target
not	thou	1
not	aaron	0
not	taco	0
not	shalt	1
not	mango	0
not	finglonger	0
not	make	1
not	plumbus	0
...	...	...

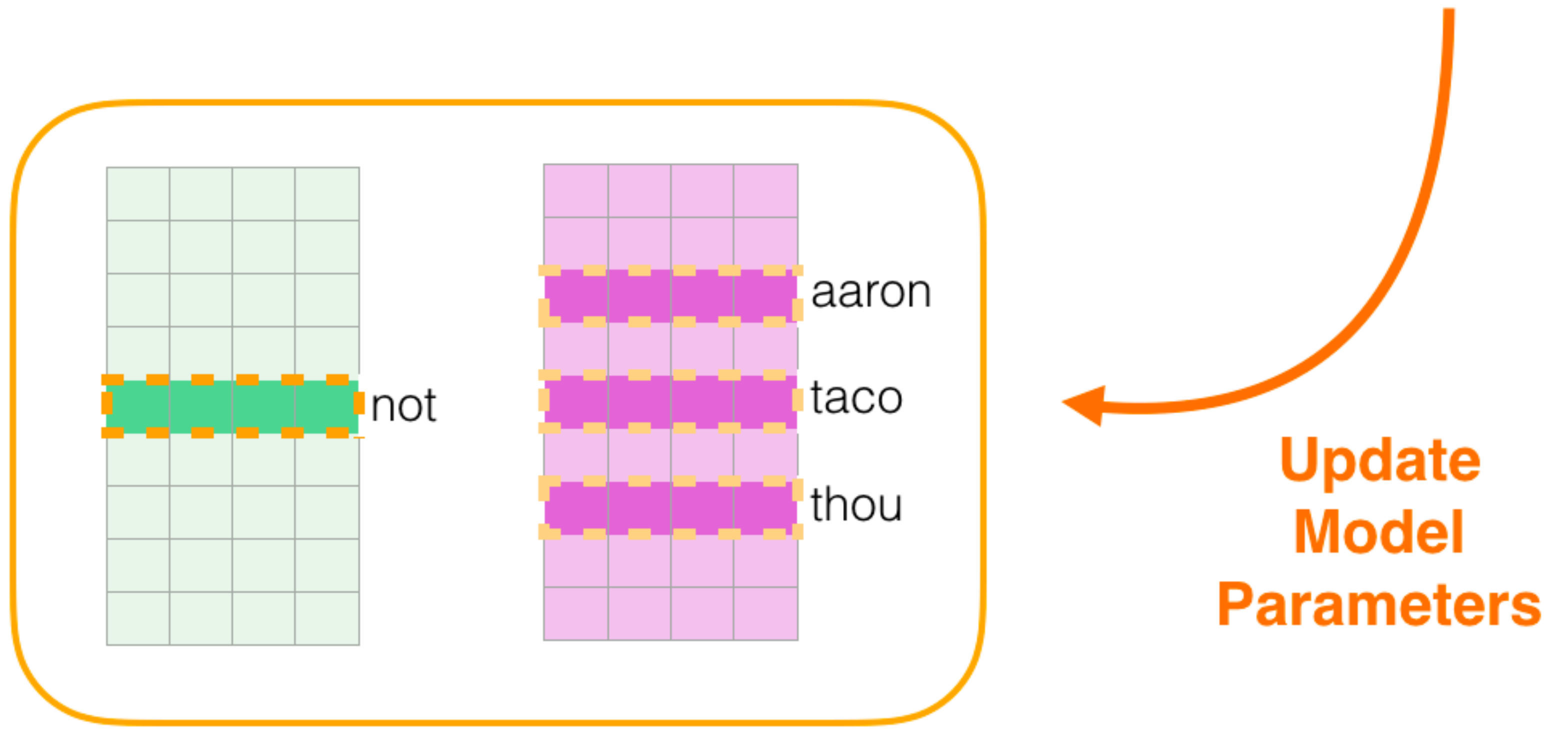
model



Source: Jay Alammar, 2019, “The Illustrated word2ve



input word	output word	target	input • output	sigmoid()	Error
not	thou	1	0.2	0.55	0.45
not	aaron	0	-1.11	0.25	-0.25
not	taco	0	0.74	0.68	-0.68



Source: Jay Alammar, 2019, “The Illustrated word2ve

Approach: predict if candidate word *c* is a "neighbor"

1. Treat the target word *t* and a neighboring context word *c* as **positive examples**.
2. Randomly sample other words in the lexicon to get negative examples
3. Use logistic regression to train a classifier to distinguish those two cases
4. Use the learned weights as the embeddings

Skip-Gram Training Data

Assume a +/- 2 word window, given training sentence:

…lemon, a [tablespoon of apricot jam, a] pinch…

c1 c2 [target] c3 c4

Skip-Gram Classifier

(assuming a +/- 2 word window)

…lemon, a [tablespoon of apricot jam, a] pinch…

c1 c2 [target] c3 c4

Goal: train a classifier that is given a candidate (**w**ord, **c**ontext) pair

(apricot, jam)

(apricot, aardvark)

…

And assigns each pair a probability:

$P(+|w, c)$

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

Similarity is computed from dot product

Remember: two vectors are similar if they have a high dot product

- Cosine is just a normalized dot product

So:

- Similarity(w,c) $\propto w \cdot c$

We'll need to normalize to get a probability

- (cosine isn't a probability either)

Turning dot products into probabilities

Sim(w,c) ≈ w · c

To turn this into a probability

We'll use the sigmoid from logistic regression:

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

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

How Skip-Gram Classifier computes $P(+|w, c)$

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

This is for one context word, but we have lots of context words.
We'll assume independence and just multiply them:

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

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

Skip-gram classifier: summary

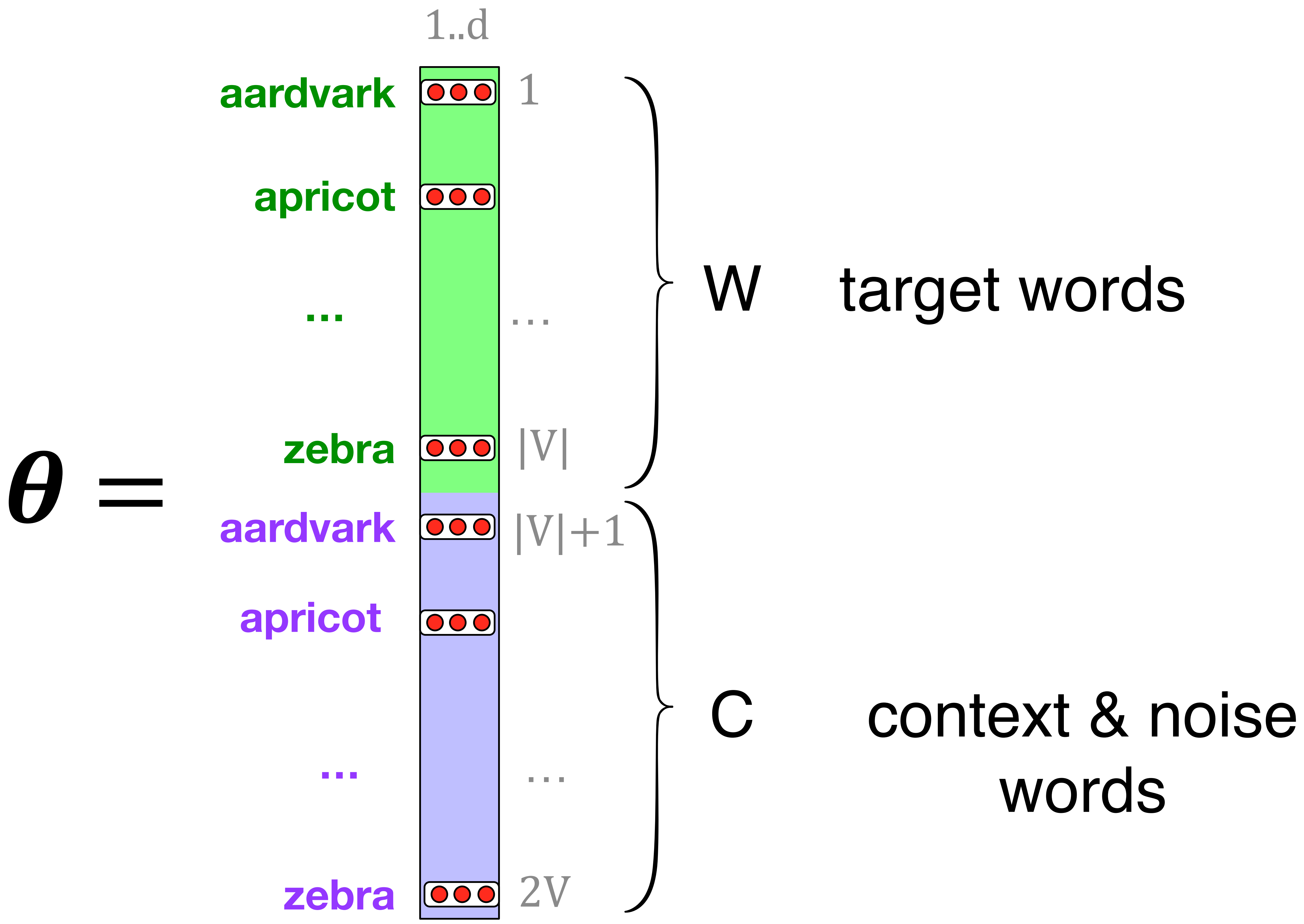
A probabilistic classifier, given

- a test target word *w*
- its context window of *L* words $c_{1:L}$

Estimates probability that w occurs in this window based on similarity of w (embeddings) to $c_{1:L}$ (embeddings).

To compute this, we just need embeddings for all the words.

These embeddings we'll need: a set for w, a set for c



Skip-Gram Training data

…lemon, a [tablespoon of apricot jam, a] pinch…

c1 c2 [target] c3 c4

positive examples +

t	c
apricot	tablespoon
apricot	of
apricot	jam
apricot	a

Skip-Gram Training data

…lemon, a [tablespoon of apricot jam, a] pinch…

c1 c2 [target] c3 c4

positive examples +

t	c
apricot	tablespoon
apricot	of
apricot	jam
apricot	a

For each positive example we'll grab k negative examples, sampling by frequency

Skip-Gram Training data

…lemon, a [tablespoon of apricot jam, a] pinch…

c1 c2 [target] c3 c4

positive examples +

t	c
apricot	tablespoon
apricot	of
apricot	jam
apricot	a

negative examples -

t	c	t	c
apricot	aardvark	apricot	seven
apricot	my	apricot	forever
apricot	where	apricot	dear
apricot	coaxial	apricot	if

Word2vec: how to learn vectors

Given the set of positive and negative training instances, and an initial set of embedding vectors

The goal of learning is to adjust those word vectors such that we:

- **Maximize** the similarity of the target word, context word pairs (w , c_{pos}) drawn from the positive data
- **Minimize** the similarity of the (w , c_{neg}) pairs drawn from the negative data.

Loss function for one *w with* c_{pos} , c_{neg1} ...c_{negk}

Maximize the similarity of the target with the actual context words, and minimize the similarity of the target with the *k* negative sampled non-neighbor words.

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

Learning the classifier

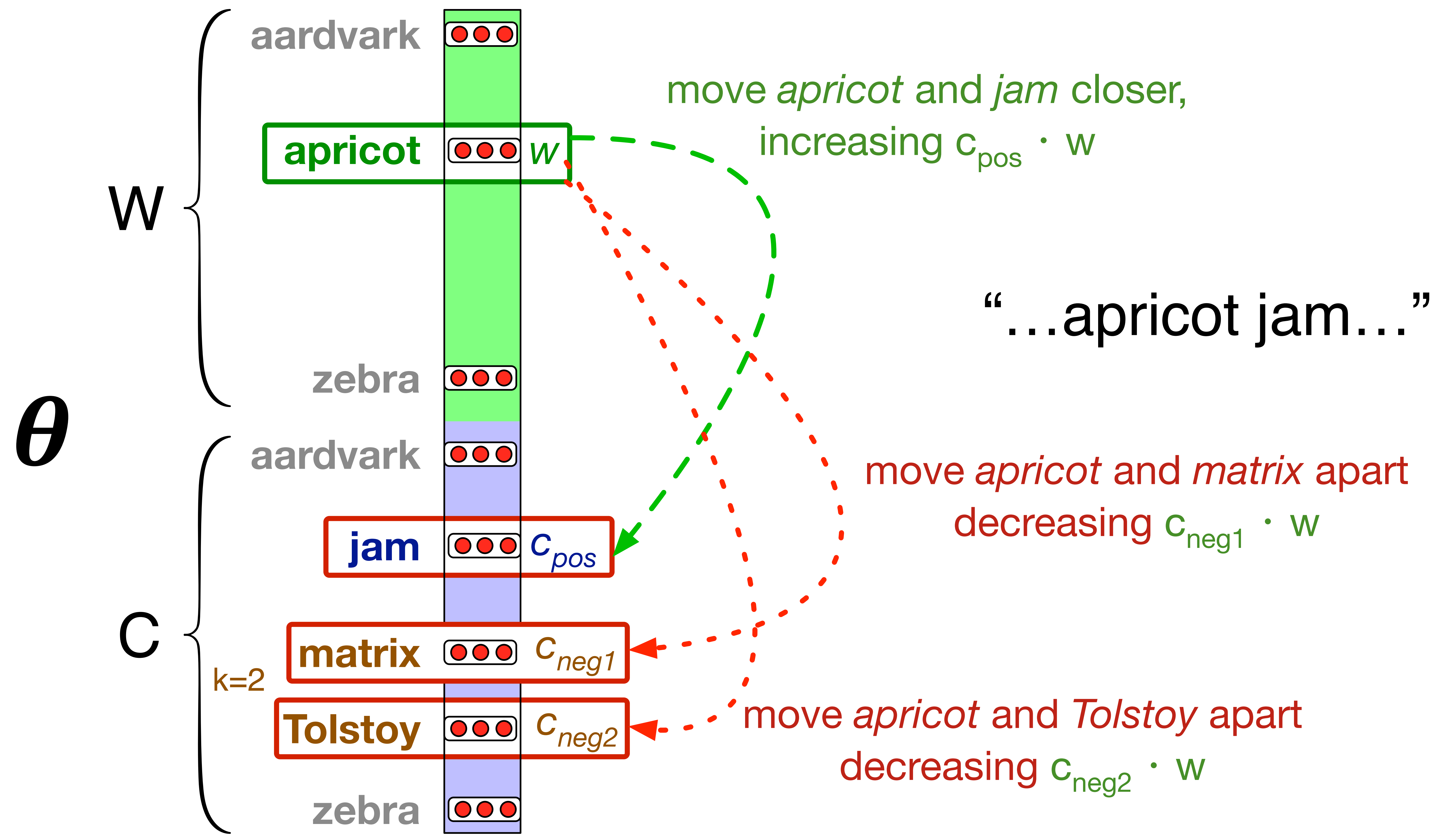
How to learn?

- Stochastic gradient descent!

We'll adjust the word weights to

- make the positive pairs more likely
- and the negative pairs less likely,
- over the entire training set.

Intuition of one step of gradient descent



Two sets of embeddings

SGNS learns two sets of embeddings

Target embeddings matrix W

Context embedding matrix C

It's common to just add them together, representing word *i* as the vector $w_i + c_i$

Summary: How to learn word2vec (skip-gram) embeddings

Start with V random d-dimensional vectors as initial embeddings

Train a classifier based on embedding similarity

- Take a corpus and take pairs of words that co-occur as positive examples
- Take pairs of words that don't co-occur as negative examples
- Train the classifier to distinguish these by slowly adjusting all the embeddings to improve the classifier performance
- Throw away the classifier code and keep the embeddings.

Properties of embeddings

The kinds of neighbors depend on window size

Small windows (C= +/- 2) : nearest words are syntactically similar words in same taxonomy

- *Hogwarts* nearest neighbors are other fictional schools
- *Sunnydale, Evernight, Blandings*

Large windows (C= +/- 5) : nearest words are related words in same semantic field

- *Hogwarts* nearest neighbors are Harry Potter world:
- *Dumbledore, half-blood, Malfoy*

The effect of window size on word embeddings

window size: 3		window size: 30		window size: 300	
putting	able	effective	crimes	diseases	tourism
bringing	can	deliver	offences	disease	visitors
taking	unable	improve	offence	vaccine	museums
giving	trying	strengthen	prosecutions	obesity	tourist
introducing	prepared	improving	murder	screening	citizenship
providing	will	efficient	criminals	diabetes	Media
looking	willing	sustainable	crime	pregnancy	holiday
making	wants	develop	arrested	HIV	holidays
talking	want	delivering	cases	medical	music
publishing	happy	ensuring	prosecution	babies	Olympics

Examples of words that cluster near one another in word2vec (skip-gram) embeddings estimated on the House of Commons corpus with different windows.

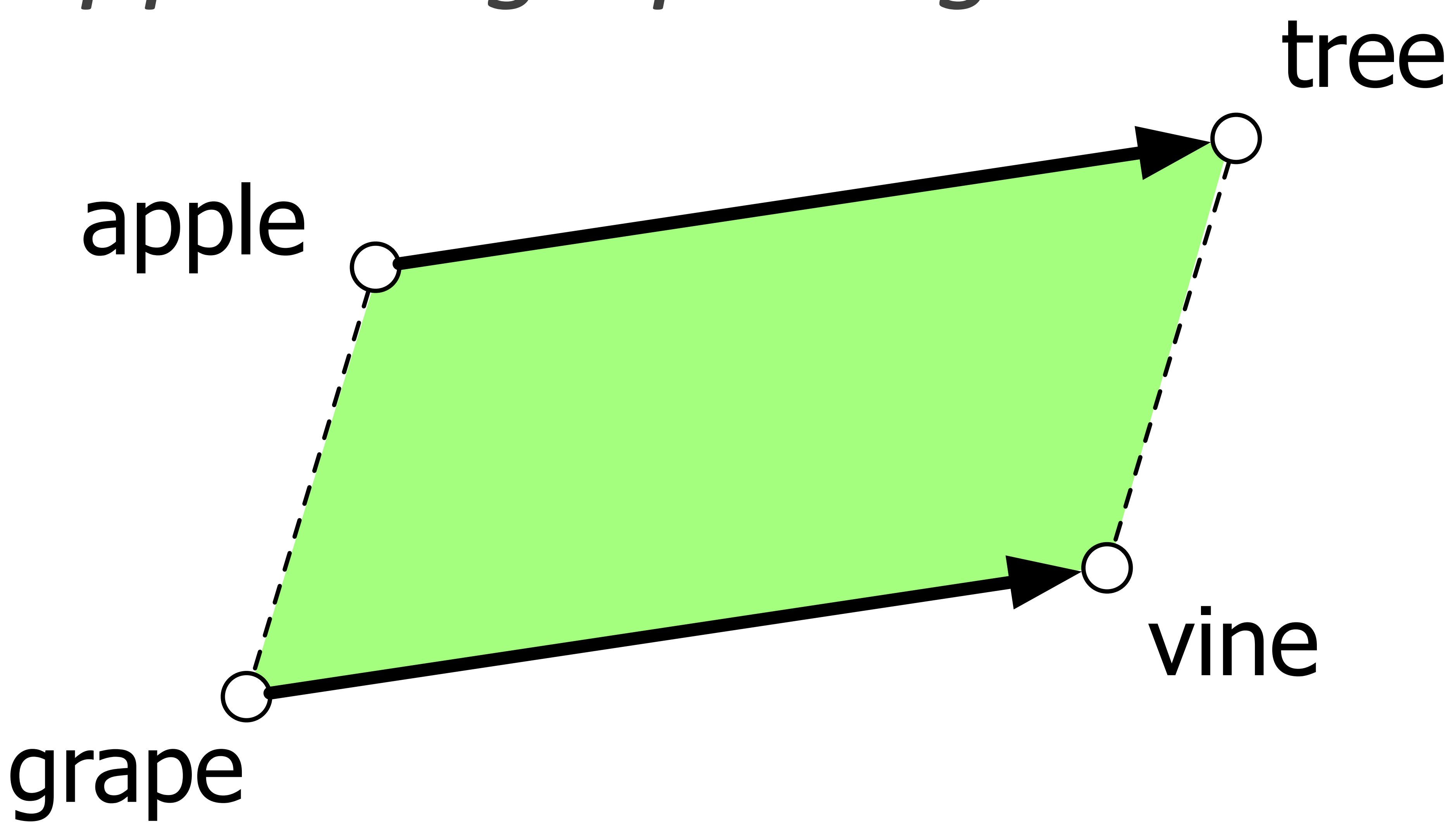
Source: Goist and Monroe 2018

Analogical relations

The classic parallelogram model of analogical reasoning
(Rumelhart and Abrahamson 1973)

To solve: *"apple is to tree as grape is to ______"*

Add $\overrightarrow{tree} - \overrightarrow{apple}$ *to* $\overrightarrow{grape}$ *to get* $\overrightarrow{vine}$



Analogical relations via parallelogram

The parallelogram method can solve analogies with both sparse and dense embeddings (Turney and Littman 2005, Mikolov et al. 2013b)

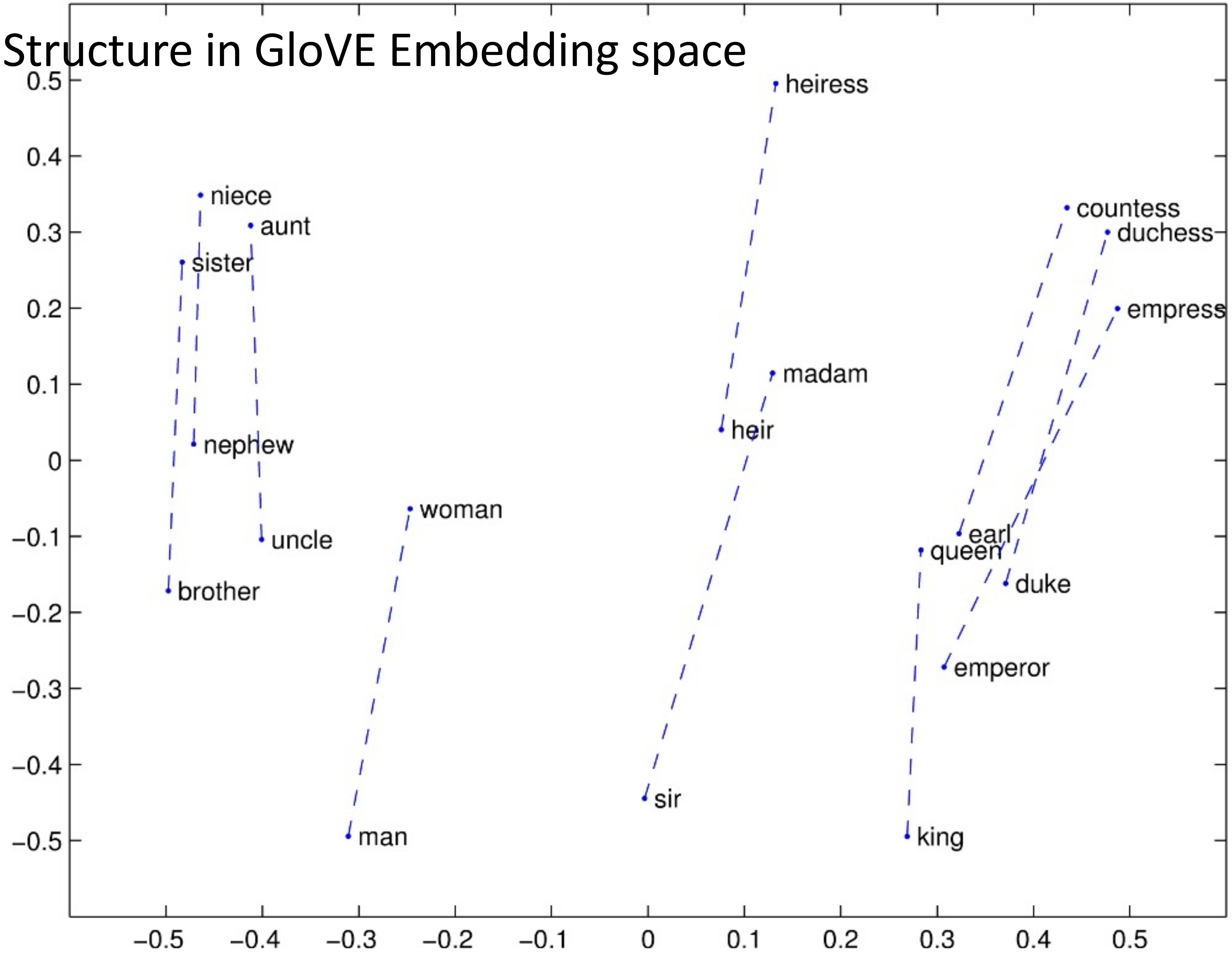
$\overrightarrow{\text{king}} - \overrightarrow{\text{man}} + \overrightarrow{\text{woman}}$ is close to $\overrightarrow{\text{queen}}$

$\overrightarrow{\text{Paris}} - \overrightarrow{\text{France}} + \overrightarrow{\text{Italy}}$ is close to $\overrightarrow{\text{Rome}}$

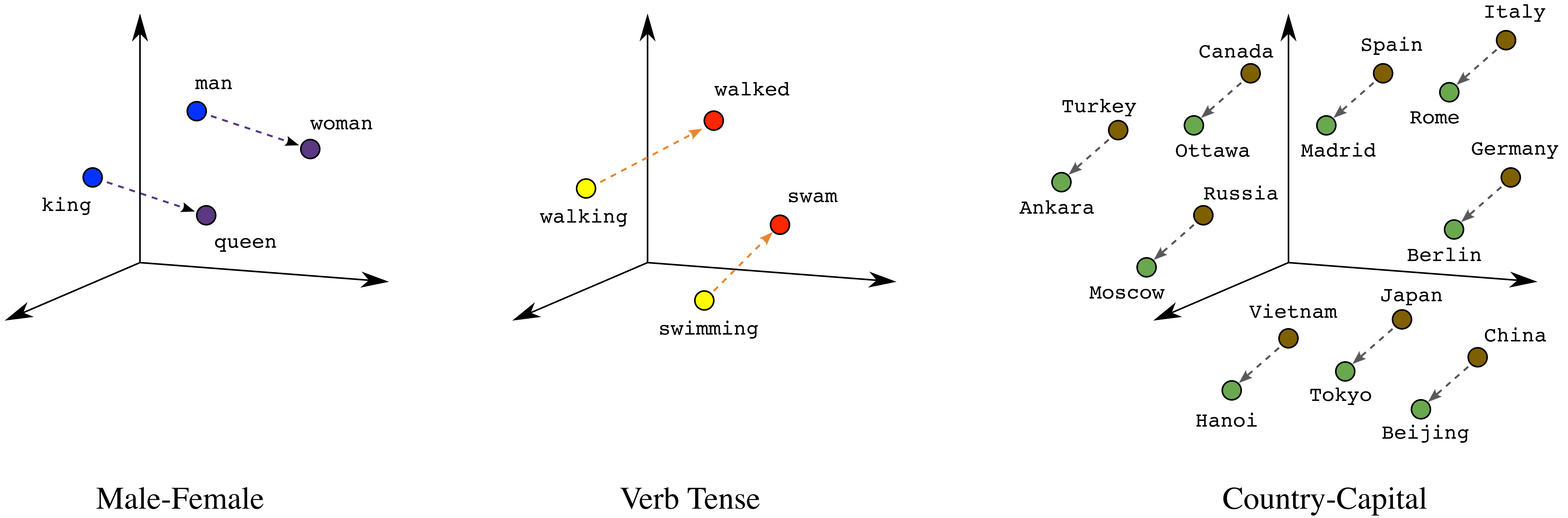
For a problem a:a*::b:b*, the parallelogram method is:

$$\hat{b}^* = \underset{x}{\operatorname{argmax}} \ \text{distance}(x, a^* - a + b)$$

Structure in GloVE Embedding space



If you know one thing about word embeddings — vector semantics



Demonstrated through "Analogy completion tasks" …
Man is to *woman* as *king* is to ??

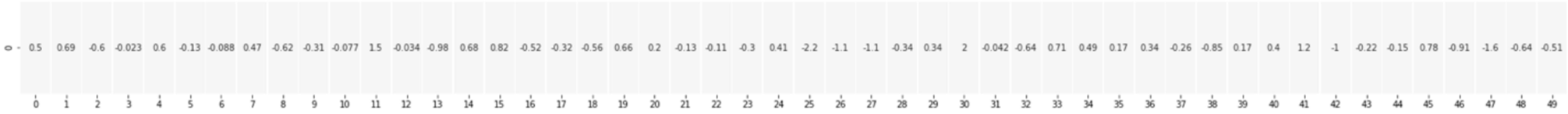
$$V_{woman} - V_{man} + V_{king} = ?? \cong V_{queen}$$

Illustration source: https://www.tensorflow.org/tutorials/representation/word2vec

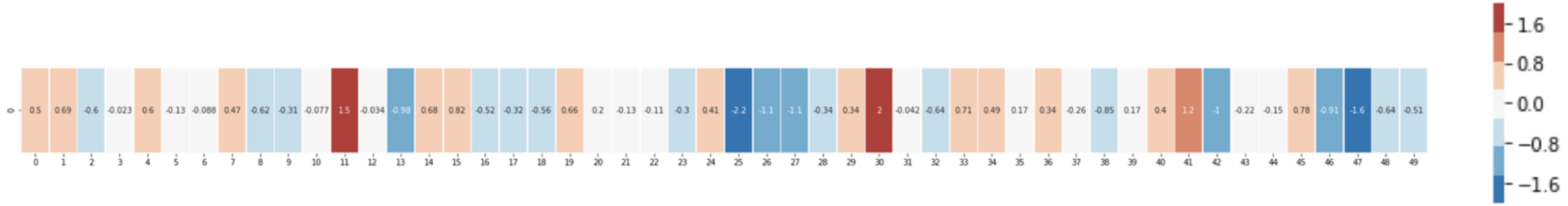
This is a word embedding for the word “king” (GloVe vector trained on Wikipedia):

```
[ 0.50451 , 0.68607 , -0.59517 , -0.022801, 0.60046 , -0.13498 , -0.08813 , 0.47377 , -0.61798 , -0.31012 ,
-0.076666, 1.493 , -0.034189, -0.98173 , 0.68229 , 0.81722 , -0.51874 , -0.31503 , -0.55809 , 0.66421 , 0.1961
, -0.13495 , -0.11476 , -0.30344 , 0.41177 , -2.223 , -1.0756 , -1.0783 , -0.34354 , 0.33505 , 1.9927 ,
-0.04234 , -0.64319 , 0.71125 , 0.49159 , 0.16754 , 0.34344 , -0.25663 , -0.8523 , 0.1661 , 0.40102 , 1.1685 ,
-1.0137 , -0.21585 , -0.15155 , 0.78321 , -0.91241 , -1.6106 , -0.64426 , -0.51042 ]
```

It’s a list of 50 numbers. We can’t tell much by looking at the values. But let’s visualize it a bit so we can compare it other word vectors. Let’s put all these numbers in one row:

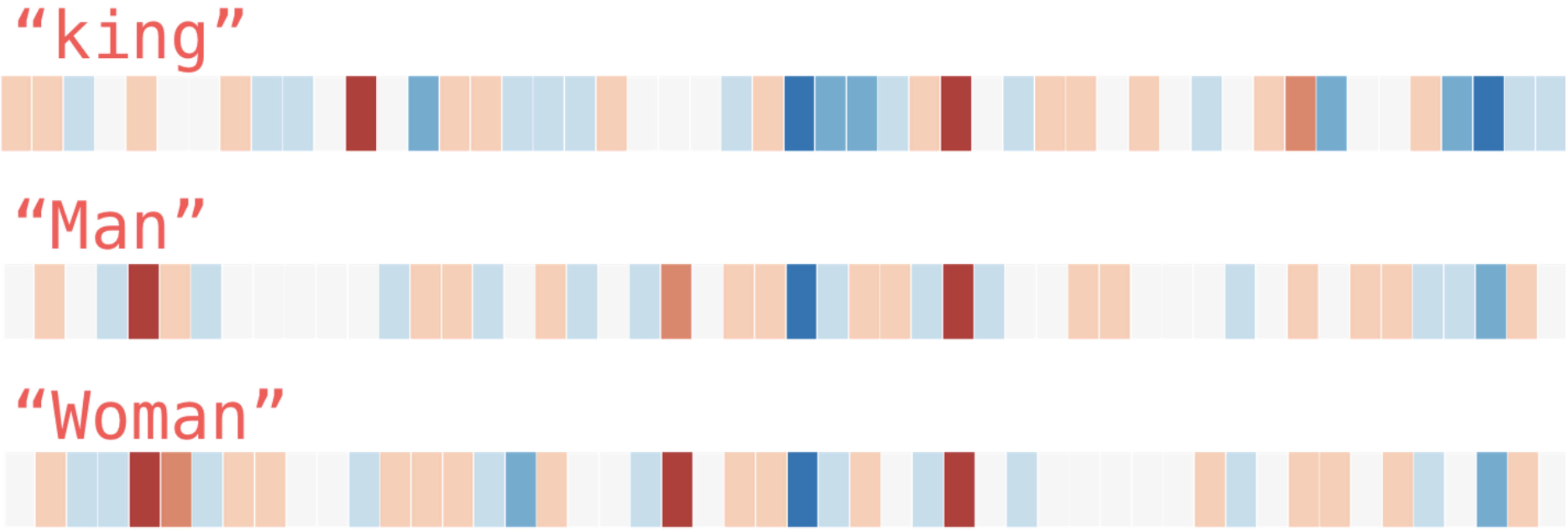


Let’s color code the cells based on their values (red if they’re close to 2, white if they’re close to 0, blue if they’re close to -2):



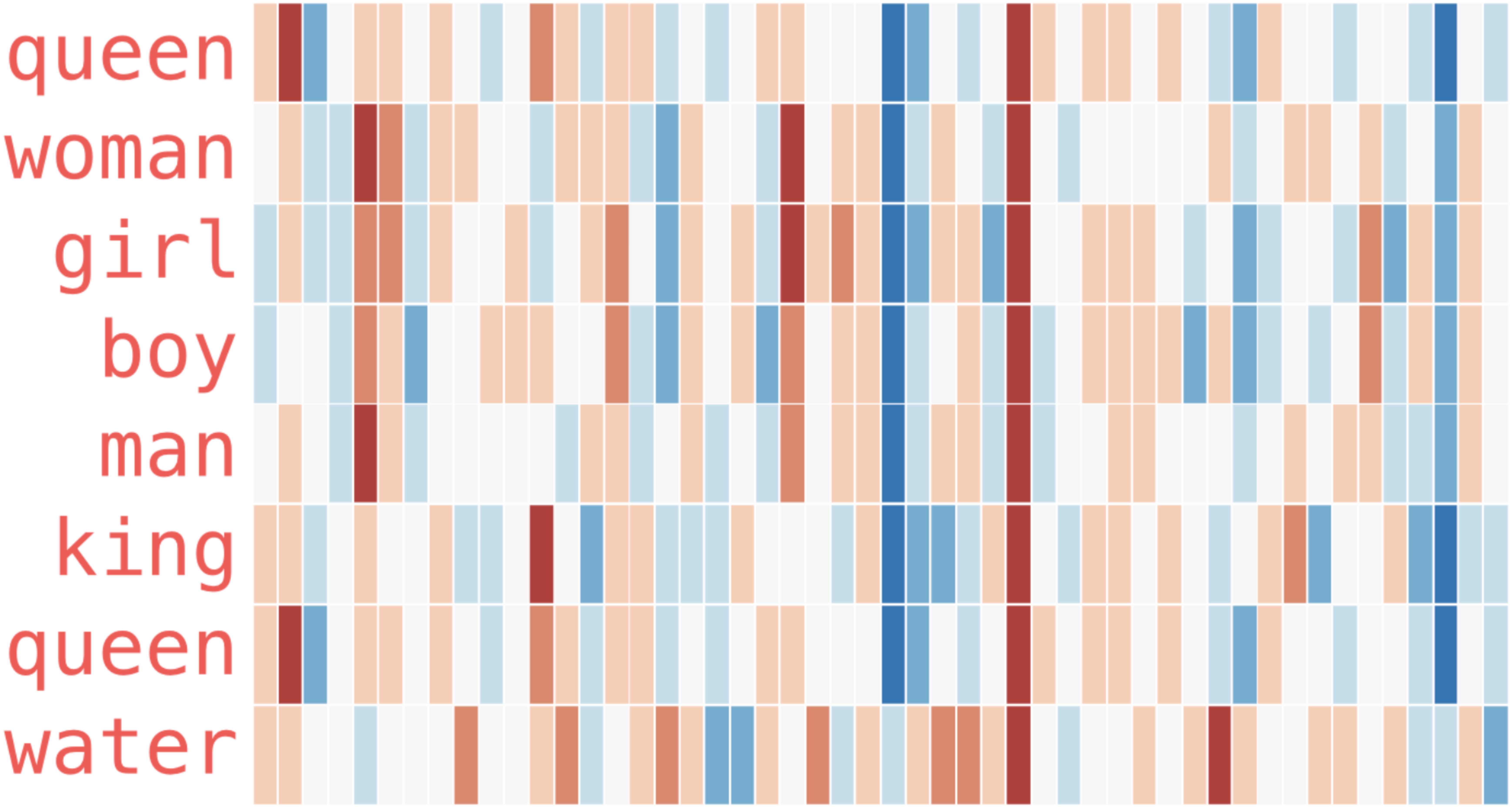
Source: Jay Alammar, 2019, “The Illustrated word2vec

We'll proceed by ignoring the numbers and only looking at the colors to indicate the values of the cells. Let's now contrast "King" against other words:



Source: Jay Alammar, 2019, "The Illustrated word2vec

Here’s another list of examples (compare by vertically scanning the columns looking for columns with similar colors):



Source: Jay Alammar, 2019, “The Illustrated word2vec

Analogies

> "Words can carry any burden we wish. All that's required is agreement and a tradition upon which to build." ~God Emperor of Dune

The famous examples that show an incredible property of embeddings is the concept of analogies. We can add and subtract word embeddings and arrive at interesting results. The most famous example is the formula: “king” - “man” + “woman”:

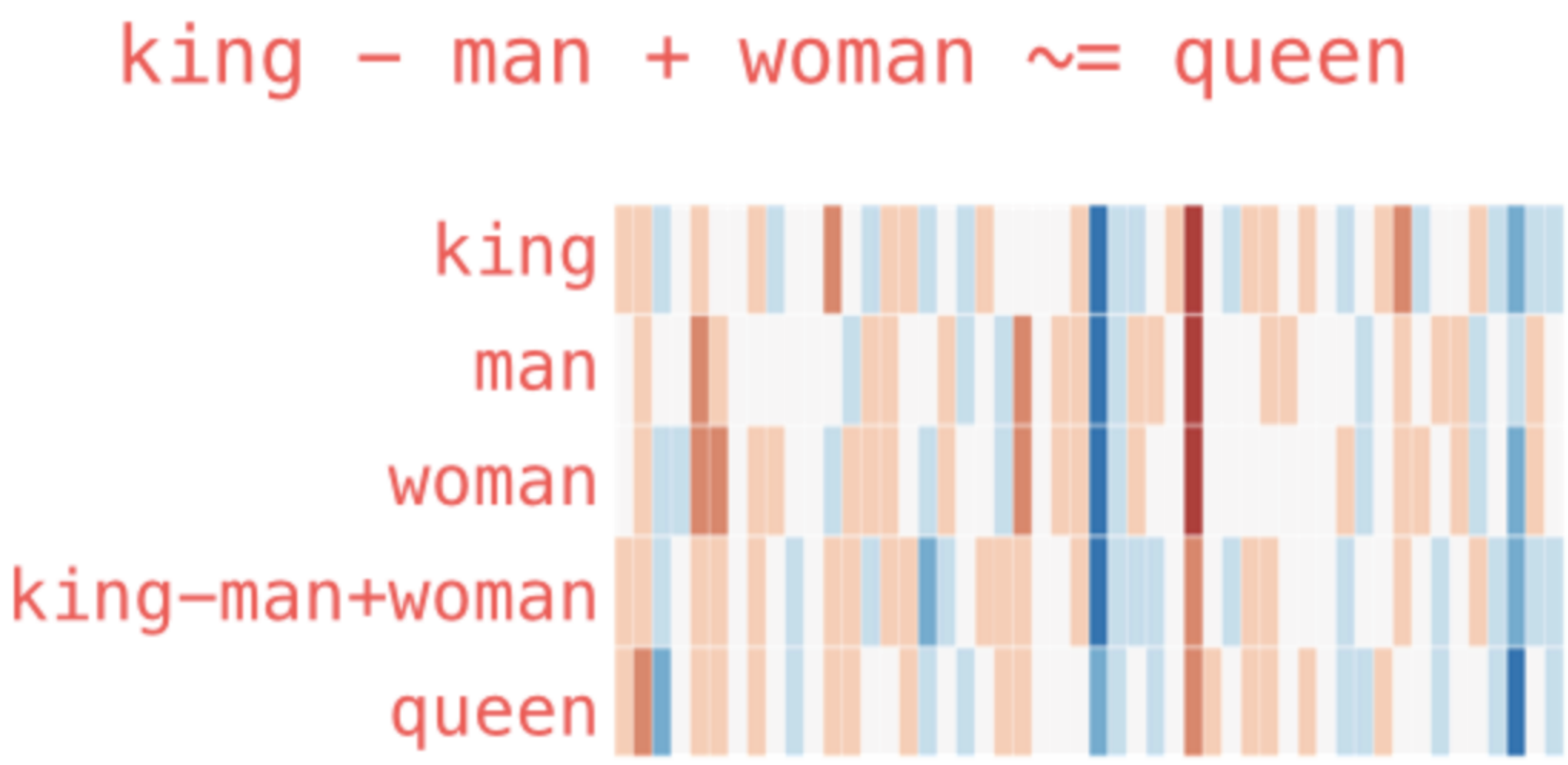
```
model.most_similar(positive=["king","woman"], negative=["man"])
```

```
[('queen', 0.8523603677749634),
 ('throne', 0.7664333581924438),
 ('prince', 0.7592144012451172),
 ('daughter', 0.7473883032798767),
 ('elizabeth', 0.7460219860076904),
 ('princess', 0.7424570322036743),
 ('kingdom', 0.7337411642074585),
 ('monarch', 0.721449077129364),
 ('eldest', 0.7184862494468689),
 ('widow', 0.7099430561065674)]
```

Using the Gensim library in python, we can add and subtract word vectors, and it would find the most similar words to the resulting vector. The image shows a list of the most similar words, each with its cosine similarity.

Source: Jay Alammar, 2019, “The Illustrated word2vec

We can visualize this analogy as we did previously:



The resulting vector from "king-man+woman" doesn't exactly equal "queen", but "queen" is the closest word to it from the 400,000 word embeddings we have in this collection.

Source: Jay Alammar, 2019, “The Illustrated word2vec

Caveats with the parallelogram method

It only seems to work for frequent words, small distances and certain relations (relating countries to capitals, or parts of speech), but not others. (Linzen 2016, Gladkova et al. 2016, Ethayarajh et al. 2019a)

Understanding analogy is an open area of research (Peterson et al. 2020)

Embeddings as a window onto historical semantics

Train embeddings on different decades of historical text to see meanings shift

~30 million books, 1850-1990, Google Books data

William L. Hamilton, Jure Leskovec, and Dan Jurafsky. 2016. Diachronic Word Embeddings Reveal Statistical Laws of Semantic Change. Proceedings of ACL.

Embeddings reflect cultural bias!

Bolukbasi, Tolga, Kai-Wei Chang, James Y. Zou, Venkatesh Saligrama, and Adam T. Kalai. "Man is to computer programmer as woman is to homemaker? debiasing word embeddings." In *NeurIPS*, pp. 4349-4357. 2016.

Ask "Paris : France :: Tokyo : x"

- x = Japan

Ask "father : doctor :: mother : x"

- x = nurse

Ask "man : computer programmer :: woman : x"

- x = homemaker

Algorithms that use embeddings as part of e.g., hiring searches for programmers, might lead to bias in hiring

Historical embedding as a tool to study cultural biases

Garg, N., Schiebinger, L., Jurafsky, D., and Zou, J. (2018). Word embeddings quantify 100 years of gender and ethnic stereotypes. Proceedings of the National Academy of Sciences 115(16), E3635–E3644.

- Compute a **gender or ethnic bias** for each adjective: e.g., how much closer the adjective is to "woman" synonyms than "man" synonyms, or names of particular ethnicities
  - Embeddings for **competence** adjective (*smart, wise, brilliant, resourceful, thoughtful, logical)* are biased toward men, a bias slowly decreasing 1960-1990
  - Embeddings for **dehumanizing** adjectives (barbaric, monstrous, bizarre) were biased toward Asians in the 1930s, bias decreasing over the 20th century.
- These match the results of old surveys done in the 1930s

Results of embeddings depend on

- Method (word2vec a neural network, GloVe a matrix decomposition)
- Corpus on which they're trained.
- What counts as a token (see, fasttext, sense2vec)
- Window size and, for larger windows, "sentence" boundaries
- Size of embedding space
- Extensions include doc2vec, sentence2vec, Swivel, contextual embeddings like ELMo

Demonstrations

- Estimating your own embeddings in gensim (Python) [word2vec, fasttext]

- There is a simple tutorial on how to estimate your own GloVe embeddings in R with text2vec here: http://text2vec.org/glove.html

  - A tutorial showing how to replicate that example from within quanteda is here: https://quanteda.io/articles/pkgdown/replication/text2vec.html

Using pre-trained embeddings

Assume you have pre-trained embeddings E.
How do you use them in your model?

- Option 1: Adapt E during training

Disadvantage: only words in training data will be affected.

- Option 2: Keep E fixed, but add another hidden layer that is learned for your task
- Option 3: Learn matrix $T \in \dim(emb) \times \dim(emb)$ and use rows of $E' = ET$ (adapts all embeddings, not specific words)
- Option 4: Keep E fixed, but learn matrix $\Delta \in R^{|V| \times \dim(emb)}$ and use $E' = E + \Delta$ or $E' = ET + \Delta$ (this learns to adapt specific words)

Source: Julia Hockenmaier, Illinois, Slides for CS447