

Natural Language Processing



Module 1.2: Word Representations

Today

1. Constructing count-based vectors
2. Learning word embeddings with neural networks
3. Skip-gram model for word embeddings
4. What can word embeddings do

12:00 first lab session

HXLY 202; HXLY 206; HXLY 210; HXLY 219; HXLY 221; HXLY 225

Words as Vectors

Words as Vectors

Let's say we are trying to detect whether a sentence is grammatical

We know the label for some sentences and are trying to figure out labels for the others

Sequence	Label
I live in London	Positive
I live in Paris	Positive
I live in Tallinn	?
I live in yellow	?

Words as Vectors

If the system knew which words are similar, that would help.

Sequence	Label
I live in London	Positive
I live in Paris	Positive
I live in Tallinn	?
I live in yellow	?

✕ Tallinn

✕ London

✕ Paris

✕ yellow

✕ red

✕ blue

✕ green

Words as Vectors

If we knew which words are similar to others, that would help.

Sequence	Label
I live in London	Positive
I live in Paris	Positive
I live in Tallinn	Positive
I live in yellow	Negative

✕ Tallinn

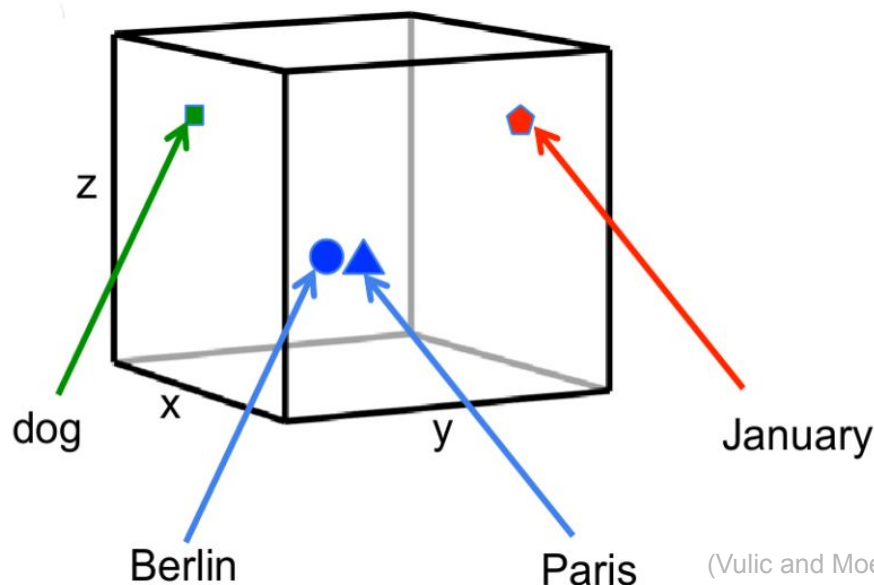
✕ London
✕ Paris

✕ yellow
✕ red ✕ blue
✕ green

Words as Vectors

Let's represent words as vectors, so that similar words have similar vectors.

Vectors can be thought of as coordinates in high-dimensional space.



(Vulic and Moens, 2016)

1-hot vectors

How can we represent words as vectors?

One option:

- Each element represents a different word.
- Have “1” in the position that represents the current word. Otherwise the vector is full of zeros.
- Also known as “1-hot” or “1-of-V” representation.

bear=[1, 0, 0]

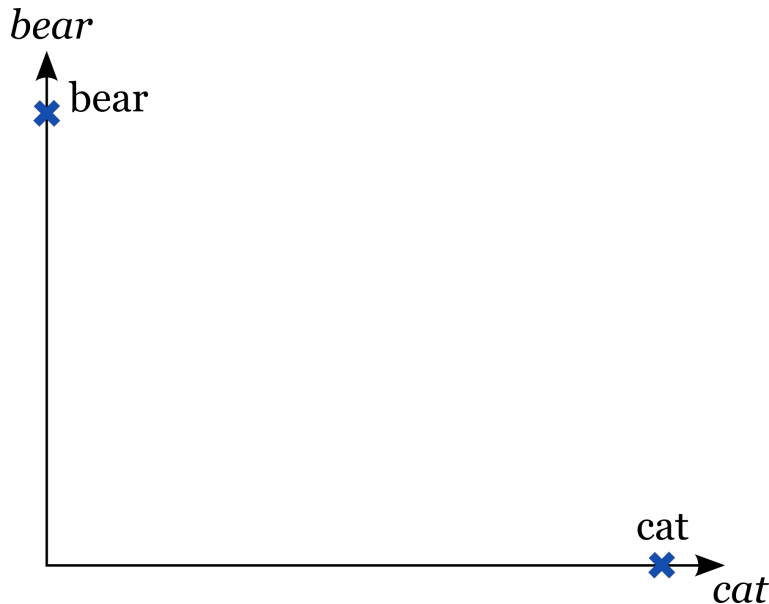
cat=[0, 1, 0]

frog=[0, 0, 1]

	<i>bear</i>	<i>cat</i>	<i>frog</i>
bear	1	0	0
cat	0	1	0
frog	0	0	1

1-hot vectors

When using 1-hot vectors, we can't fit many and they tell us very little.
Need a separate dimension for every word we want to represent (170K+).
All the vectors are orthogonal and equally distant.



Mapping of words to concepts

We could map words to broader concepts. For example, using WordNet:

<http://wordnetweb.princeton.edu/perl/webwn>

<https://www.nltk.org/howto/wordnet.html>

“cat”, “kitten” → “feline mammal”

“London”, “Paris”, “Tallinn” → “national capital”

“yellow”, “green” → “colour”

	<i>feline mammal</i>	<i>national capital</i>	<i>colour</i>
Tallinn	0	1	0
London	0	1	0
yellow	0	0	1

Mapping of words to concepts

- Maps some similar words together
- Reduces vector size

but...

- Relies on the completeness of a manually curated database
- Misses out on rare and new meanings
- Vectors are still orthogonal and equally distant from each other
- Disambiguating word meanings is difficult

“mouse” → “rodent mammal”

or

“mouse” → “computer device” ?

Distributed vectors

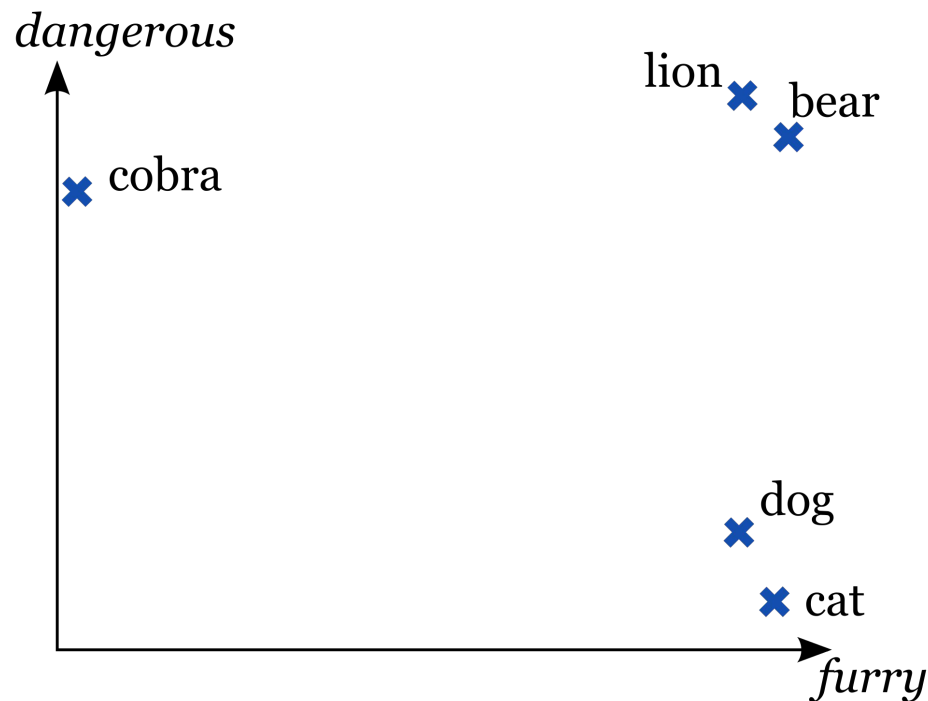
Each element represents a property and these are shared between the words.

Also known as **distributed** representations.

	<i>furry</i>	<i>dangerous</i>	<i>mammal</i>
bear	0.9	0.85	1
cat	0.85	0.15	1
frog	0	0.05	0

Distributed vectors

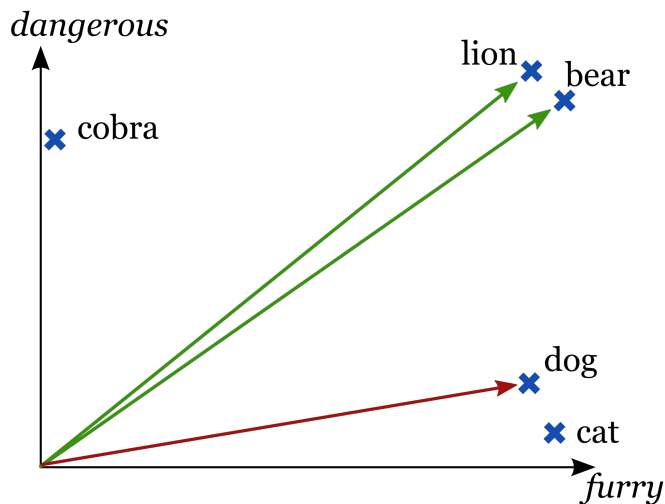
Distributed vectors group similar words/objects together



	<i>furry</i>	<i>dangerous</i>
bear	0.9	0.85
cat	0.85	0.15
cobra	0.0	0.8
lion	0.85	0.9
dog	0.8	0.15

Distributed vectors

Can use cosine to calculate similarity between two words



$$\cos(a, b) = \frac{\sum_i a_i \cdot b_i}{\sqrt{\sum_i a_i^2} \cdot \sqrt{\sum_i b_i^2}}$$

$$\cos(\text{lion}, \text{bear}) = 0.998$$

$$\cos(\text{lion}, \text{dog}) = 0.809$$

$$\cos(\text{cobra}, \text{dog}) = 0.727$$

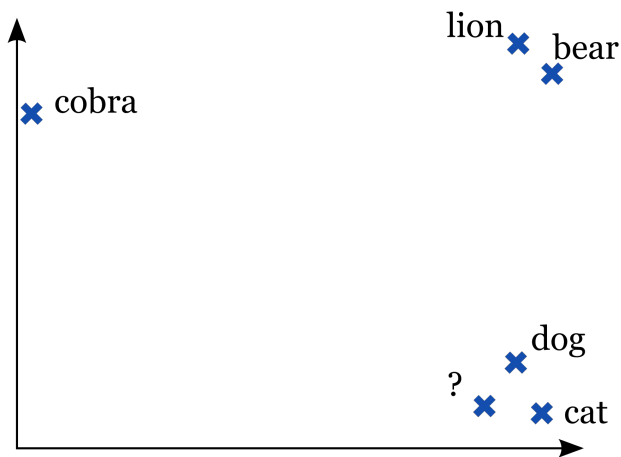
Could also use L2 norm, but that is sensitive to vector length $\sqrt{\sum_{i=1}^n (q_i - d_i)^2}$

Note: cosine is a similarity score, L1 is a distance

Distributed vectors

We can infer some information based only on the vector of the word

We don't even need to know the labels on the vector elements



We don't want to be creating these features manually.

Distributional hypothesis

Words which are similar in meaning occur in similar contexts.

(Harris, 1954)

You shall know a word by the company it keeps

(Firth, 1957)

He is reading a **magazine**

I was reading a **newspaper**

This **magazine** published my story

The **newspaper** published an article

She buys a **magazine** every month

He buys this **newspaper** every day

Count-based vectors

Let's count how often a word occurs together with specific other words (within a context window of a particular size).

He is **reading** a **magazine**

I was **reading** a **newspaper**

This **magazine** **published** my story

The **newspaper** **published** an article

She **buys** a **magazine** every month

He **buys** this **newspaper** every day

	reading	a	this	published	my	buys	the	an	every	month	day
magazine	1	2	1	1	1	1	0	0	1	1	0
newspaper	1	1	1	1	0	1	1	1	1	0	1

TF-IDF

Problem: Common words (“a”, “the”, “it”, “this”, ...) dominate the counts

Can weight the vectors using TF-IDF: $\text{TF-IDF}_{w,d,D} = \text{TF}_{w,d} \text{IDF}_{w,D}$

Term Frequency (TF):

Upweights words w that are more
Important to d

Feature or context word

Frequency of w occurring together with d

$$\text{TF}_{w,d} = \frac{\text{count}(w, d)}{\sum_{w'} \text{count}(w', d)}$$

Main word or document

Inverse Document Frequency (IDF):

Downweights words that appear
everywhere

Size of document collection

Number of documents in D that contain w

$$\text{IDF}_{w,D} = \log \frac{|D|}{|\{d \in D : w \in d\}|}$$

Word Embeddings

Word Embeddings

The count-based vectors are still very large (possibly 170K+ elements)

They are also very sparse - mostly full of zeros

Instead: Let's allocate a number of parameters for each word and allow a neural network to automatically learn what the useful values should be.

Often referred to as “**word embeddings**”, as we are embedding the words into a real-valued low-dimensional space.

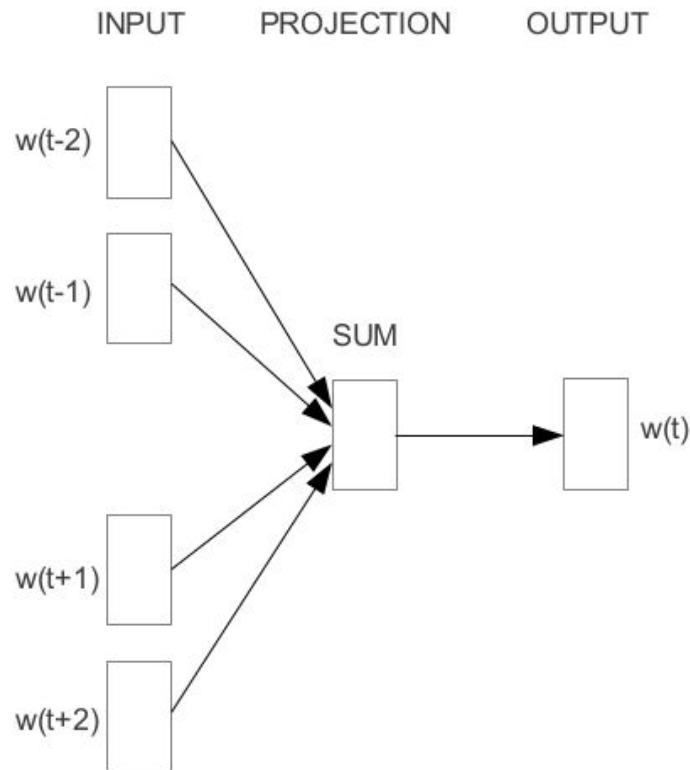
The resulting vectors are usually short (~ 300 -1K) and dense (non-zero).

Continuous Bag-of-Words (CBOW)

Predict the **target word** w_t based on the surrounding **context words**

$w_{t-j} \dots w_{t-2} w_{t-1}$? $w_{t+1} w_{t+2} \dots w_{t+j}$

Mikolov et. al. 2013. Efficient Estimation of Word Representations in Vector Space.

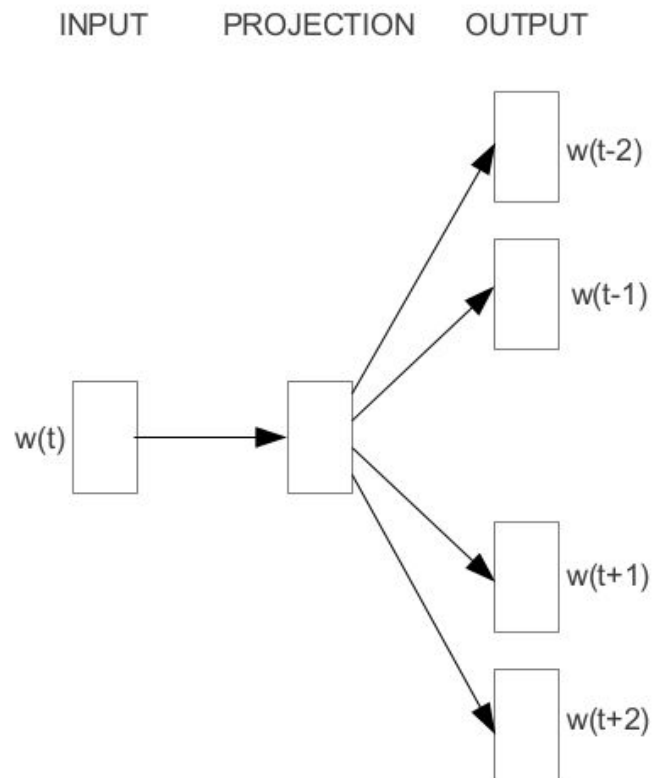


Skip-gram

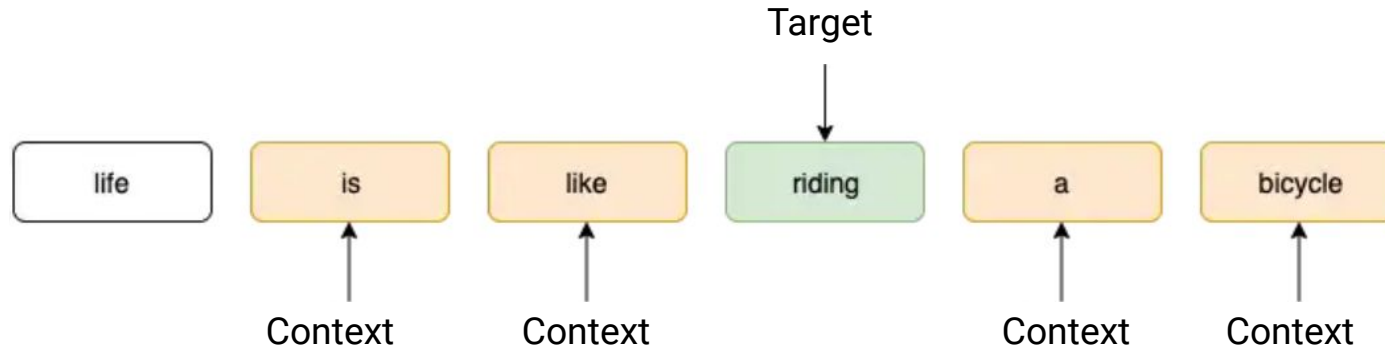
Predict the context words based on the target word w_t

? ? ? w_t ? ? ?

Mikolov et. al. 2013. Efficient Estimation of Word Representations in Vector Space.



Skip-gram



Given **riding** at position 3, predict **is**, **like**, **a**, **bicycle** at positions 1, 2, 4 and 5 respectively. May decide to ignore stop words such as **is** and **a**.

Skip-gram

Give 1-hot vector of the target word as input x

Map that to the **embedding of that target word**, using weight matrix W :

$$h = xW$$

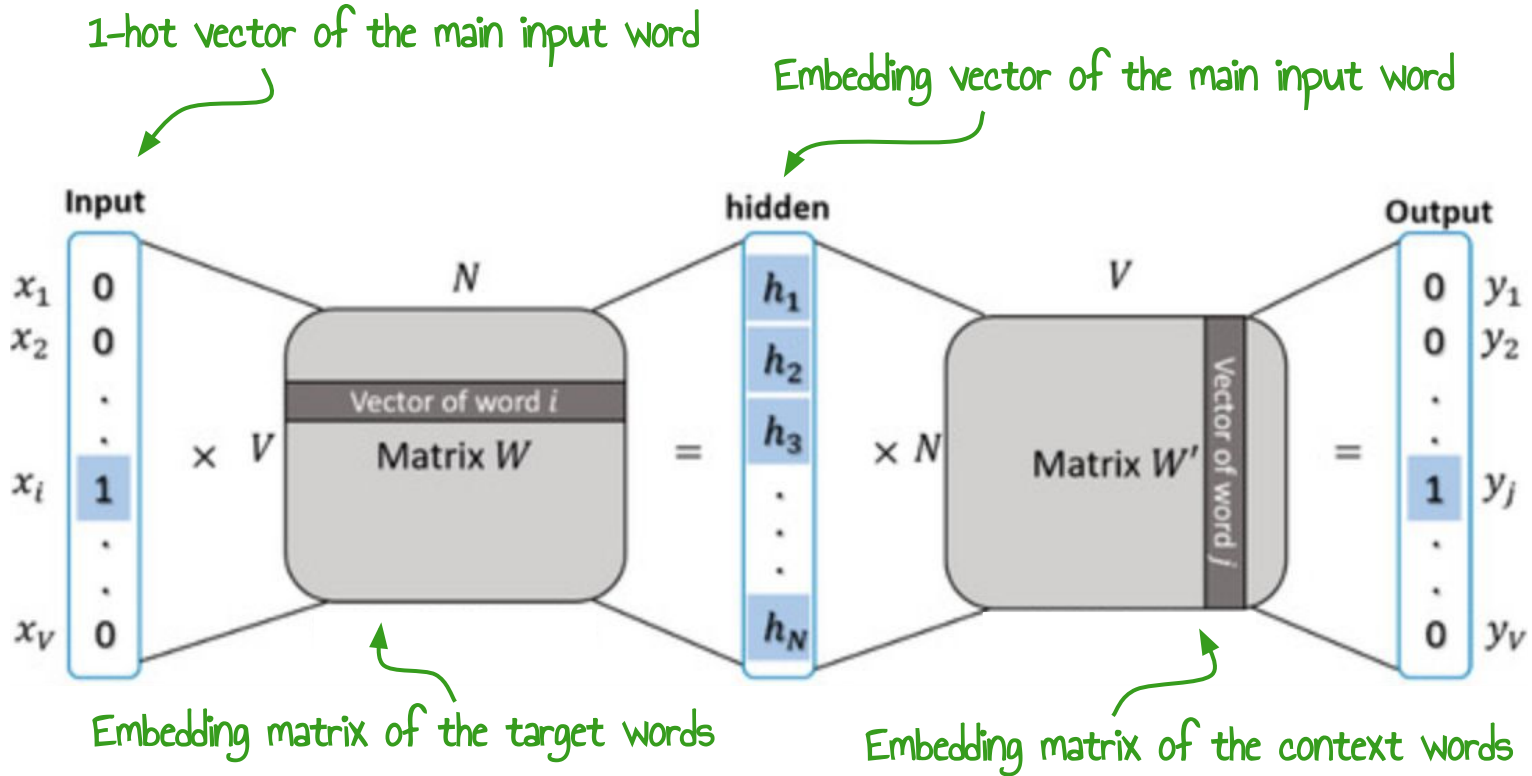
Map the embedding of the target word to the **possible context words**, using weight matrix W' :

$$y = hW'$$

y has length of whole vocabulary. Apply **softmax** to make y into a probability distribution over the whole vocabulary.

$$y' = \text{softmax}(y)$$

Skip-gram



We can actually use either matrix for our embeddings

Skip-gram

Essentially, the whole model is just **two matrices of embeddings**.

Directly optimizing for the embedding of the target word from W to be similar to the embedding of the context word from W' .

There is sort-of a **hidden layer** step, but we're not applying a non-linear activation function there.

Skip-gram model

Window size determines the maximum context location at which the words need to be predicted. If $c = 2$ we will be predicting the words at context location $(t-2)$, $(t-1)$, $(t+1)$ and $(t+2)$, therefore 4 words.

Source Text	Training Samples						
<table><tr><td>The</td><td>quick</td><td>brown</td></tr></table> fox jumps over the lazy dog. ➡	The	quick	brown	(the, quick) (the, brown)			
The	quick	brown					
<table><tr><td>The</td><td>quick</td><td>brown</td><td>fox</td></tr></table> jumps over the lazy dog. ➡	The	quick	brown	fox	(quick, the) (quick, brown) (quick, fox)		
The	quick	brown	fox				
<table><tr><td>The</td><td>quick</td><td>brown</td><td>fox</td><td>jumps</td></tr></table> over the lazy dog. ➡	The	quick	brown	fox	jumps	(brown, the) (brown, quick) (brown, fox) (brown, jumps)	
The	quick	brown	fox	jumps			
<table><tr><td>The</td><td>quick</td><td>brown</td><td>fox</td><td>jumps</td><td>over</td></tr></table> the lazy dog. ➡	The	quick	brown	fox	jumps	over	(fox, quick) (fox, brown) (fox, jumps) (fox, over)
The	quick	brown	fox	jumps	over		

Skip-gram model

The loss function to optimize:

find word representations that are useful for predicting the correct context words w_{t+j} given a target word w_t by maximising the average log probability, for context window c

$$\frac{1}{T} \sum_{t=1}^T \sum_{-c \leq j \leq c, j \neq 0} \log p(w_{t+j} | w_t; \theta)$$

- Categorical cross-entropy
- T - the number of documents in our corpus

Skip-gram model

Probability is defined using the softmax function

$$p(w_{t+j}|w_t) = \frac{\exp(u_{w_{t+j}}^\top h_{w_t})}{\sum_{w'=1}^W \exp(u_{w'}^\top h_{w_t})}$$

where h_w and u_w are the target and context vector representations of word w

Skip-gram model

Downside: In order to compute a single forward pass of the model, we have to sum across the entire corpus vocabulary (in the softmax)

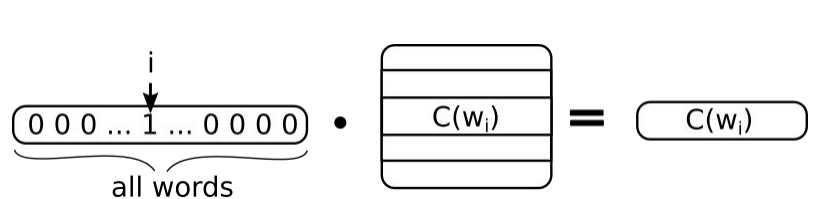
This gets inefficient with large vocabularies and large embeddings

E.g. for 300-dimensional embeddings and $V = 10\text{K}$ vocabulary, we have to do 3 million feature-weight multiplications, with **each** embedding matrix (6M total).

Embedding matrices

Two ways of thinking about an embedding matrix.

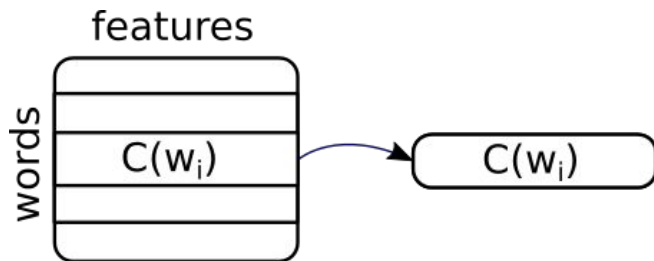
1. It is a normal **weight matrix**, multiplied with a 1-hot input vector



The diagram illustrates the first way of thinking about an embedding matrix. It shows a 1-hot input vector (a row of zeros with a '1' at position i) multiplied by an embedding matrix $C(w_i)$ (a column of values) to produce the word embedding $C(w_i)$. The input vector is labeled 'all words' and the output is labeled $C(w_i)$.

$$\underbrace{[0 \ 0 \ 0 \ \dots \ 1 \ \dots \ 0 \ 0 \ 0 \ 0]}_{\text{all words}} \cdot \begin{bmatrix} \vdots \\ C(w_i) \\ \vdots \end{bmatrix} = \begin{bmatrix} \vdots \\ C(w_i) \\ \vdots \end{bmatrix}$$
$$[0 \ 0 \ 0 \ 1 \ 0] \times \begin{bmatrix} 17 & 24 & 1 \\ 23 & 5 & 7 \\ 4 & 6 & 13 \\ 10 & 12 & 19 \\ 11 & 18 & 25 \end{bmatrix} = [10 \ 12 \ 19]$$

2. **Each row** contains a word embedding, which we need to extract



Skip-gram model - approximation

Approximation: instead of multi-class classification, train the model using a **binary classification objective** (logistic regression) to discriminate real context words (w_{t+1}) from k other (noise) words

Negative Sampling, where we maximise:

$$\log p(D = 1|w_t, w_{t+1}) + k \mathbb{E}_{\tilde{c} \sim P_{noise}} [\log p(D = 0|w_t, \tilde{c})]$$

where $p(D = 1|w_t, w_{t+1})$ is binary logistic regression probability of seeing the word w_t in the context w_{t+1} . Approximate expectation by drawing k contrastive context words \tilde{c} from a noise distribution

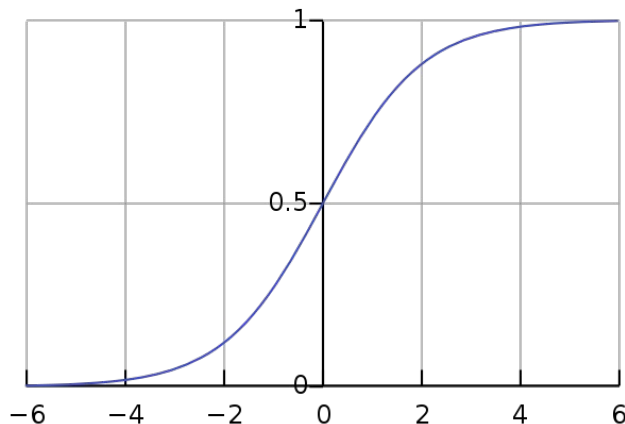
Skip-gram model - negative sampling

Probability that was defined using the **softmax function**

$$p(w_{t+j}|w_t) = \frac{\exp(u_{w_{t+j}}^\top h_{w_t})}{\sum_{w'=1}^W \exp(u_{w'}^\top h_{w_t})}$$

is now defined by **sigmoid function**

$$p(D = 1|w_t, w_{t+1}) = \frac{1}{1 + \exp(-u_{w_{t+1}}^\top h_{w_t})}$$



The model now has $|V|$ binary classifiers, one for each possible context word

Skip-gram model - negative sampling

The quick brown fox jumps over the lazy dog.

To predict quick from the, select k noisy (contrastive) words which are NOT in the context window of the

- For example, $k = 1$, and randomly selected noisy word = sheep
- Compute loss function for observed and contrastive pairs. The objective, at this time step becomes

$$\log p(D = 1 | \text{the}, \text{quick}) + \log(p(D = 0 | \text{the}, \text{sheep}))$$

Skip-gram model - negative sampling

How to select which words to use as negative examples?

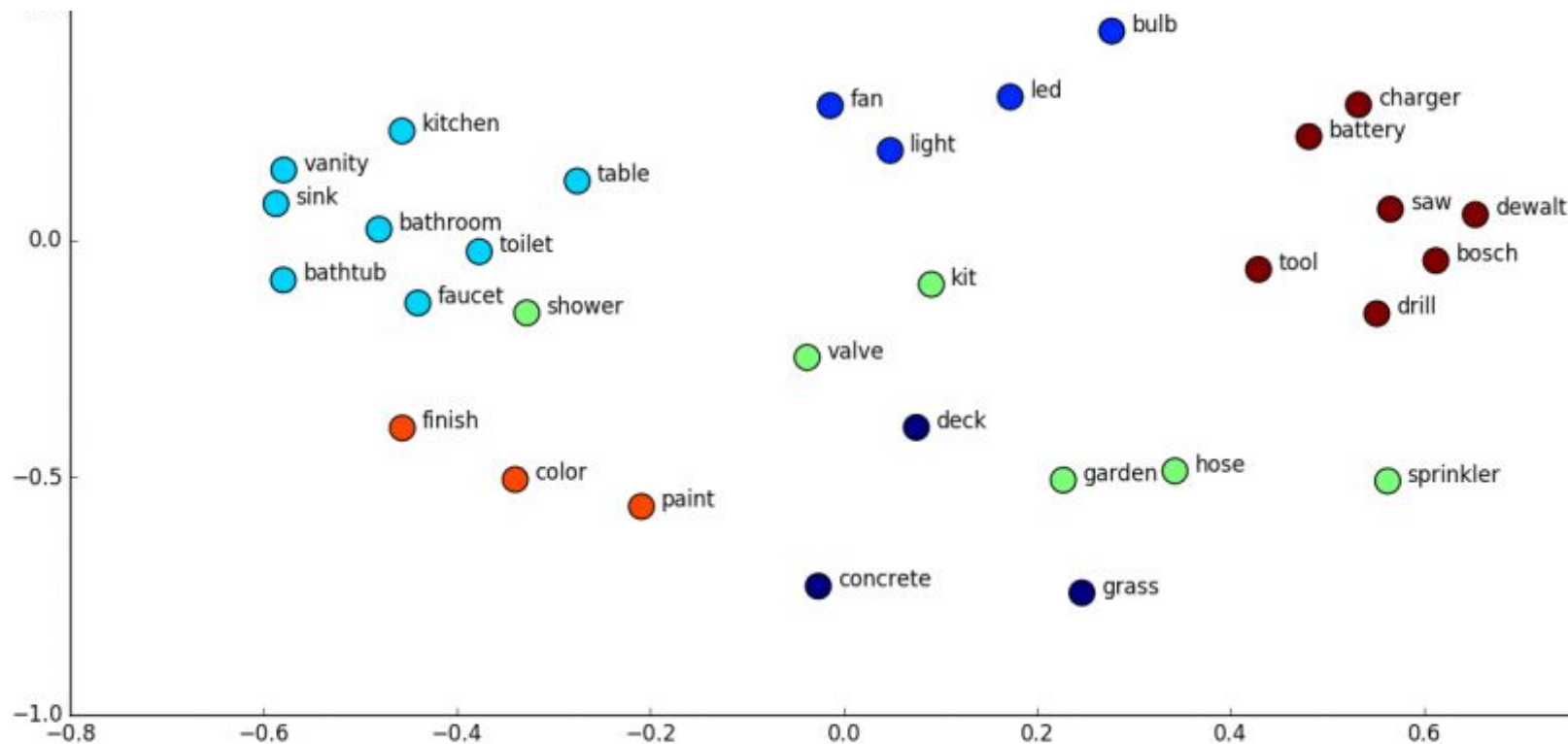
- Randomly or by frequency (selecting more frequent words more frequently)
- 5-20 negative words works well for smaller datasets
- 2-5 negative words works well for large datasets
 - For a model with weight matrix 300x10K, need to update weights for 1 positive word, plus 5 negative samples.
 - 6 output neurons, so $6 * 300 = 1800$ weight parameters need to be updated. That is only 0.06% of the 3M weights in the output layer.

Examples of what vectors capture

We can use them to find the most similar other words, given an input word

FRANCE	JESUS	XBOX	REDDISH	SCRATCHED	MEGABITS
AUSTRIA	GOD	AMIGA	GREENISH	NAILED	OCTETS
BELGIUM	SATI	PLAYSTATION	BLUISH	SMASHED	MB/S
GERMANY	CHRIST	MSX	PINKISH	PUNCHED	BIT/S
ITALY	SATAN	IPOD	PURPLISH	POPPED	BAUD
GREECE	KALI	SEGA	BROWNISH	CRIMPED	CARATS
SWEDEN	INDRA	PSNUMBER	GREYISH	SCRAPED	KBIT/S
NORWAY	VISHNU	HD	GRAYISH	SCREWED	MEGAHERTZ
EUROPE	ANANDA	DREAMCAST	WHITISH	SECTIONED	MEGAPIXELS
HUNGARY	PARVATI	GEFORCE	SILVERY	SLASHED	GBIT/S
SWITZERLAND	GRACE	CAPCOM	YELLOWISH	RIPPED	AMPERES

Examples of what vectors capture



Analogy Recovery

Task:

a is to b as c is to d

For example:

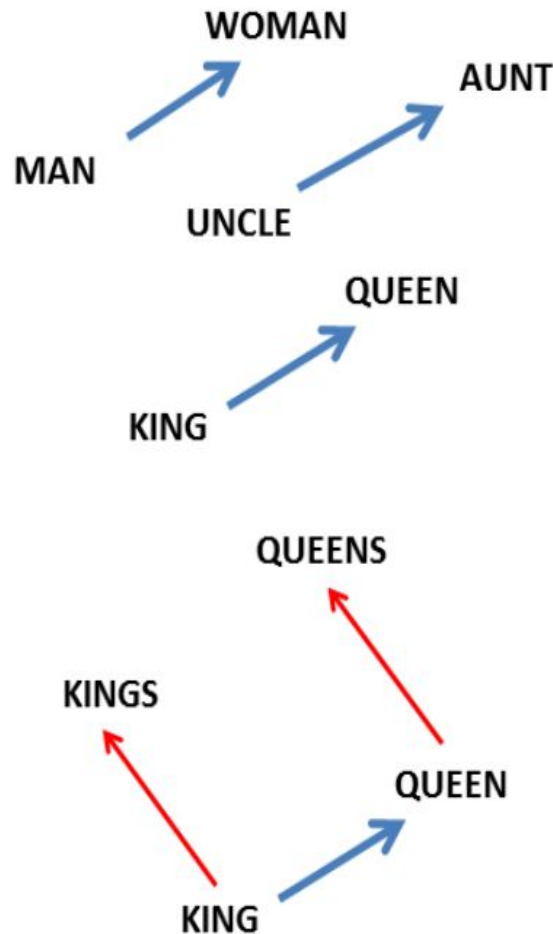
'apple' is to 'apples' as 'car' is to 'cars'

Idea: The offset of the vectors should reflect their relation.

$$a - b \approx c - d$$

$$d \approx c - a + b$$

$$\text{queen} \approx \text{king} - \text{man} + \text{woman}$$



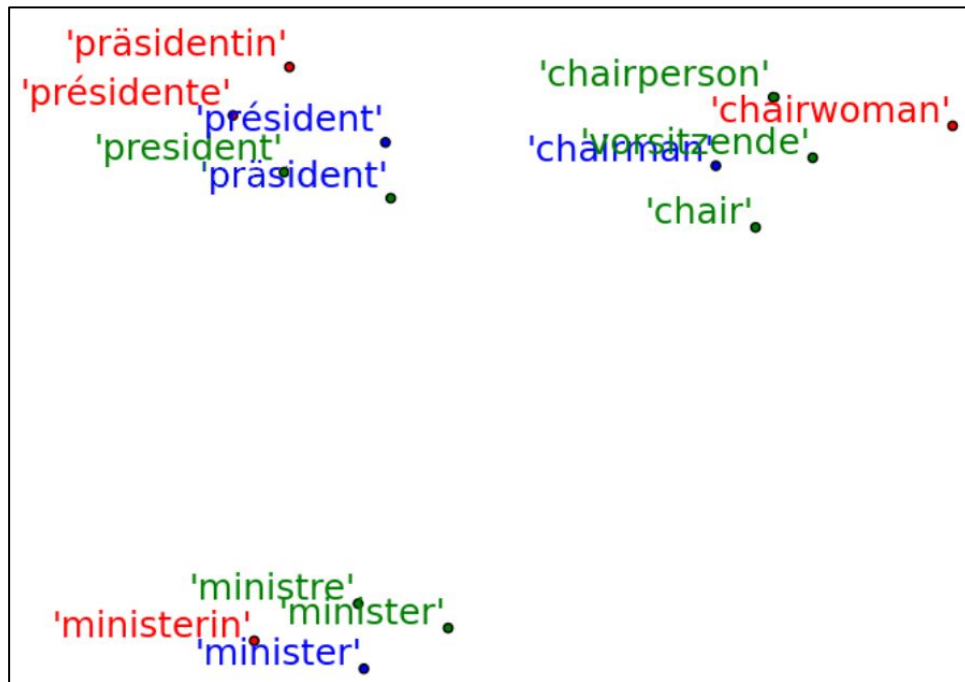
Analogy Recovery

Relationship	Example 1	Example 2	Example 3
France - Paris	Italy: Rome	Japan: Tokyo	Florida: Tallahassee
big - bigger	small: larger	cold: colder	quick: quicker
Miami - Florida	Baltimore: Maryland	Dallas: Texas	Kona: Hawaii
Einstein - scientist	Messi: midfielder	Mozart: violinist	Picasso: painter
Sarkozy - France	Berlusconi: Italy	Merkel: Germany	Koizumi: Japan
copper - Cu	zinc: Zn	gold: Au	uranium: plutonium
Berlusconi - Silvio	Sarkozy: Nicolas	Putin: Medvedev	Obama: Barack
Microsoft - Windows	Google: Android	IBM: Linux	Apple: iPhone
Microsoft - Ballmer	Google: Yahoo	IBM: McNealy	Apple: Jobs
Japan - sushi	Germany: bratwurst	France: tapas	USA: pizza

Multilingual word embeddings

The words from English, German and French are mapped into clusters based on meaning.

The colours indicate gender:
blue=male
red=female
green=neutral



Multimodal word embeddings



- day + night =



- flying + sailing =



- bowl + box =



Kiros et. al. 2014 "Unifying Visual-Semantic Embeddings with Multimodal Neural Language Models"

Things to consider

- Word embeddings require only plain text - which we have a lot
- They are very easy to use
- Already pretrained vectors also available (trained on 100B words)
- They will help your models start from an informed position

but....

- Require large amounts of data to train
- Low quality for rare words
- No coverage for unseen words
- Antonyms tend to have similar distributions: e.g. “good” and “bad”
- Does not consider morphological similarity: “car” and “cars” are independent
- Does not differentiate between different meanings of a word

Pre-trained word embeddings

Glove

Homepage: <https://nlp.stanford.edu/projects/glove/>

Vector dimensionality: 50 / 100 / 200 / 300

Vocabulary size: 400,000 - 2.2 million

Training data: Wikipedia + GigaWord / CommonCrawl / Twitter

Fast-text

Homepage: <https://github.com/facebookresearch/fastText/>

Vector dimensionality: 300

Training data: Common Crawl and Wikipedia

Trained for 157 different languages

Other types of word embeddings

- CBOW: Continuous Bag of Words (Mikolov et al., 2013)
 - Glove (Pennington, Socher and Manning, 2014)
 - Fasttext (Joulin et al., 2016)
 - Contextualised word embeddings
 - ELMo (Peters et al., 2018)
 - BERT (Devlin et al., 2018)
 - RoBERTa (Liu et al., 2019)
 - ALBERT (Lan et al., 2019)
 - XLNet (Yang et al., 2019)
 - GPT-2 (Radford et al., 2019)
 - ELECTRA (Clark et al., 2020)
 - DistilBERT (Sanh et al., 2020)
 - DeBERTa (He et al., 2020)
 - GPT-3 (Brown et al., 2020)
- ← We'll come back to these



