2nd year project - Lecture 7

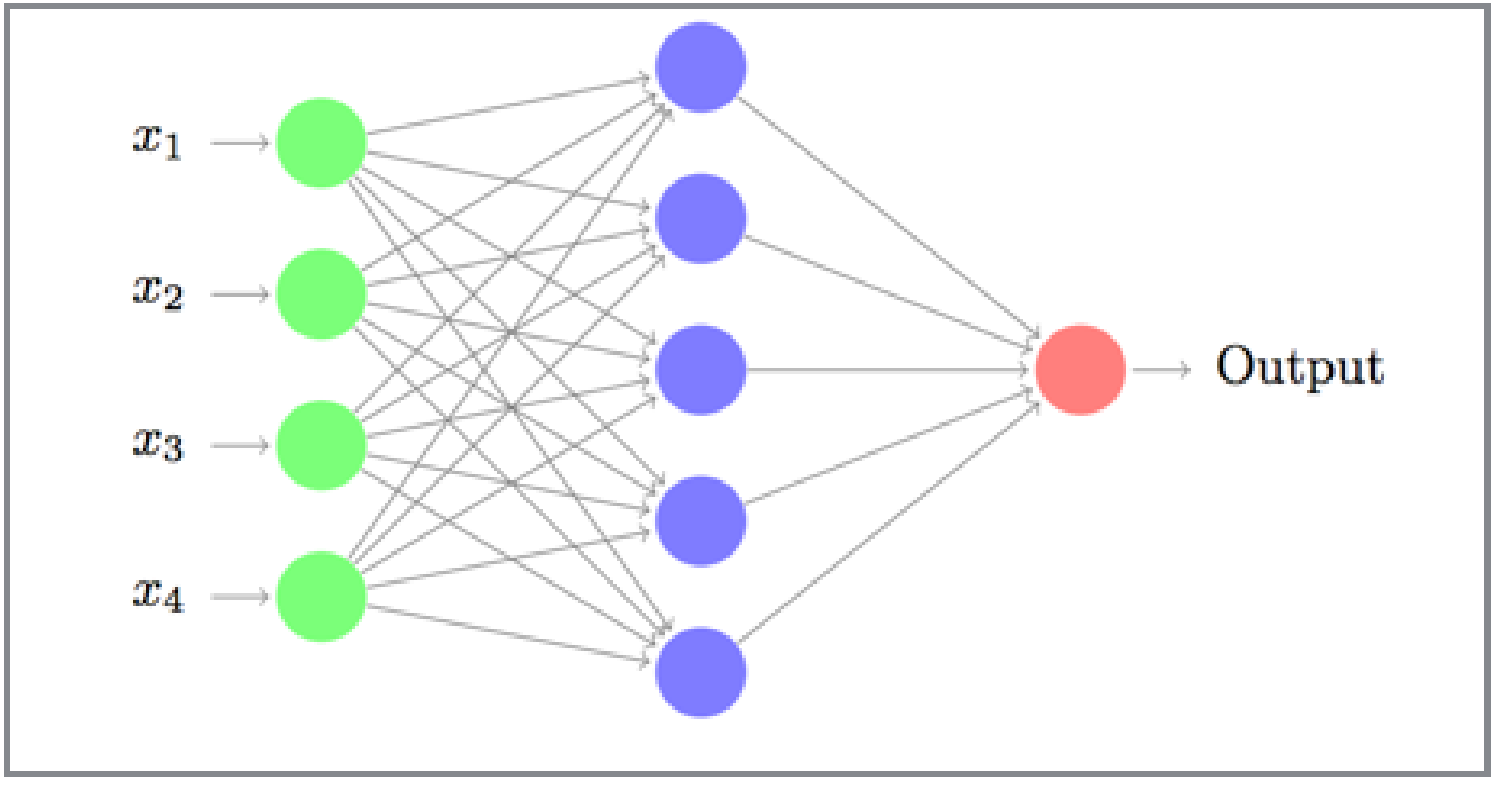
Vector Semantics

Until now

- RegEx, command line tools
- Experimental setup, annotation
- Tokenization, POS tagging, classification, language models
- Feedforward neural networks

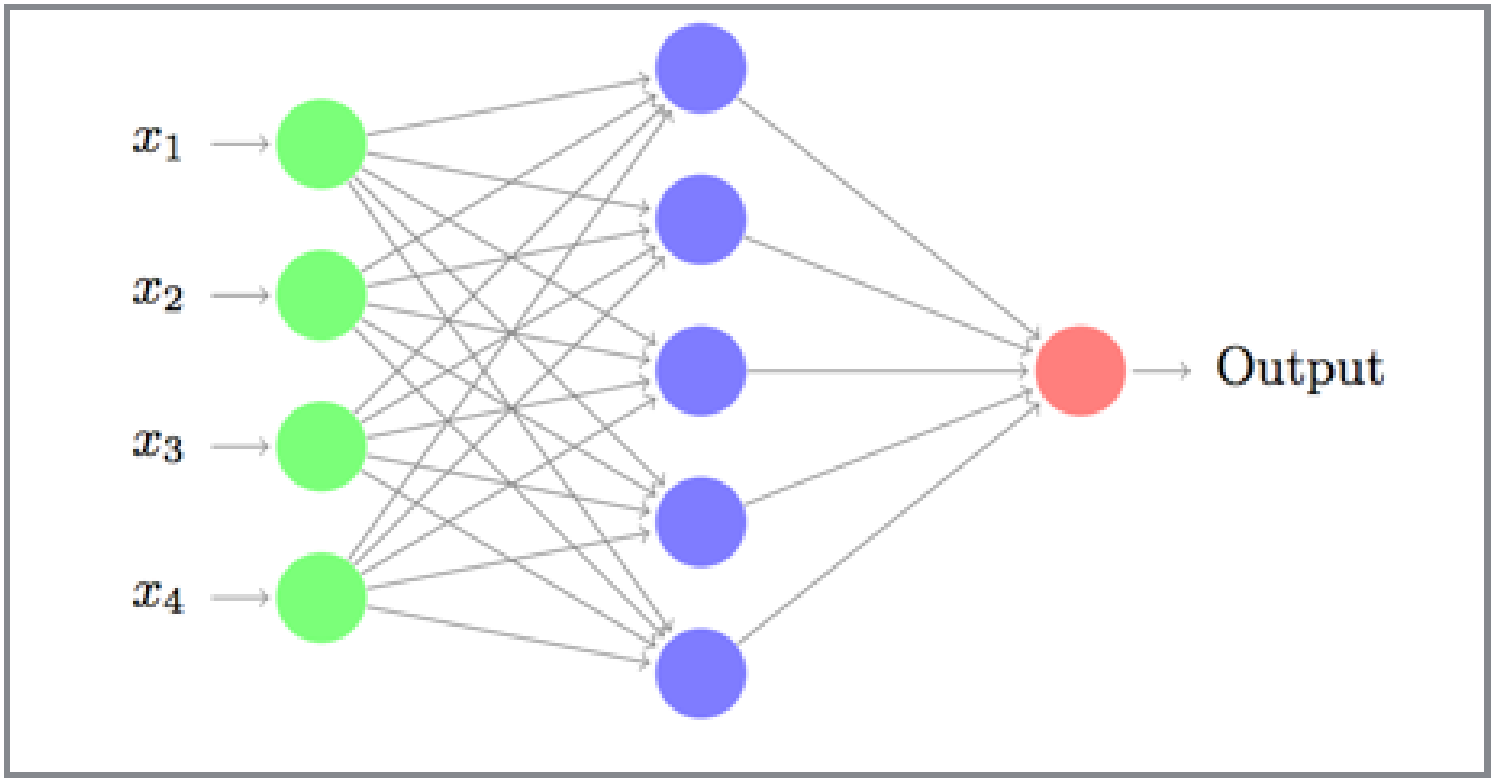
Recap: Feed-forward Neural Network

$$NN_{MLP1}(\mathbf{x}) = \sigma(\mathbf{x}\mathbf{W}^{\mathbf{1}} + \mathbf{b}^{\mathbf{1}})\mathbf{W}^{\mathbf{2}} + \mathbf{b}^{\mathbf{2}}$$



Recap: Feed-forward Neural Network

$$NN_{MLP1}(\mathbf{x}) = \sigma(\mathbf{x}\mathbf{W}^{\mathbf{1}} + \mathbf{b}^{\mathbf{1}})\mathbf{W}^{\mathbf{2}} + \mathbf{b}^{\mathbf{2}}$$



What is the input $\mathbf{x}$?

What makes language so challenging

- Ambiguity
- Zipf's law
- Non-deterministic
- Language changes constantly

Lecture outline

- **Representations of words**
  - thesaurus
- Elaborate on the difference between **sparse discrete** (one-hot/sparse binary/sparse count-based/n-hot) and **dense continuous** feature representations
- How to acquire **word representations** / **distributional similarity** / vector semantics
  - Traditional (count) vs
  - Neural (predict): word2vec

How do we represent the meaning of a word?

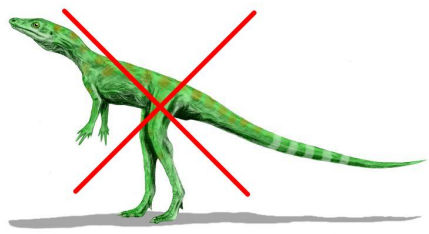
Definition: **meaning** (Webster dictionary)

- the idea that is represented by a word, phrase etc.
- the idea that a person wants to express using words, signs, etc
- the idea that is expressed in a work of writing, art, etc

WordNet

A lexicon database, containing word (senses) and relations between them!

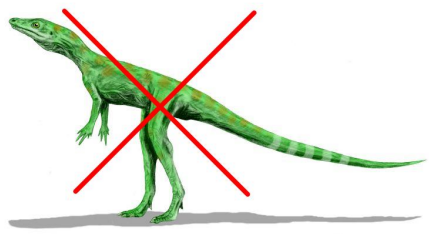
- Word senses are grouped into synsets
- Is a *thesaurus*



WordNet

A lexicon database, containing word (senses) and relations between them!

- Word senses are grouped into synsets
- Is a *thesaurus*



```
from nltk.corpus import wordnet as wn
wn.synsets('duck')
```

see also `http://wordnetweb.princeton.edu/perl/webwn`

Output:

```
[Synset('duck.n.01'),
 Synset('duck.n.02'),
 Synset('duck.n.03'),
 Synset('duck.n.04'),
 Synset('duck.v.01'),
 Synset('duck.v.02'),
 Synset('dip.v.10'),
 Synset('hedge.v.01')]
```

```
>>> wn.synsets('duck')[4].definition()
'to move (the head or body) quickly downwards or away'
>>> wn.synsets('duck')[4].examples()
['Before he could duck, another stone struck him']
```

```
>>> wn.synsets('duck')[4].definition()
'to move (the head or body) quickly downwards or away'
>>> wn.synsets('duck')[4].examples()
['Before he could duck, another stone struck him']

>>> wn.synsets('duck')[4].hypernyms()
[Synset('move.v.03')]
>>> wn.synsets('duck')[2].hypernyms()
[Synset('poultry.n.02')]
```

```
>>> thing = wn.synsets('duck')[0]
>>> for i in range(13):
...     print(thing.hypernyms()[0])
...     thing = thing.hypernyms()[0]
...
Synset('anseriform_bird.n.01')
Synset('waterfowl.n.01')
Synset('aquatic_bird.n.01')
Synset('bird.n.01')
Synset('vertebrate.n.01')
Synset('chordate.n.01')
Synset('animal.n.01')
Synset('organism.n.01')
Synset('living_thing.n.01')
Synset('whole.n.02')
Synset('object.n.01')
Synset('physical_entity.n.01')
Synset('entity.n.01')
```

Problems with resources like WordNet

- Great resource but missing **nuance** (e.g., 'proficient' is a synonym for 'good'. It is only correct in some contexts)
- Missing new meanings of words (very hard to keep up-to-date!)
- Subjective
- Requires human labor to create and maintain
- The main one is English, also available for more languages, but less comprehensive
- Typos/variations of words are hard to find

A solution via Distributional Hypothesis

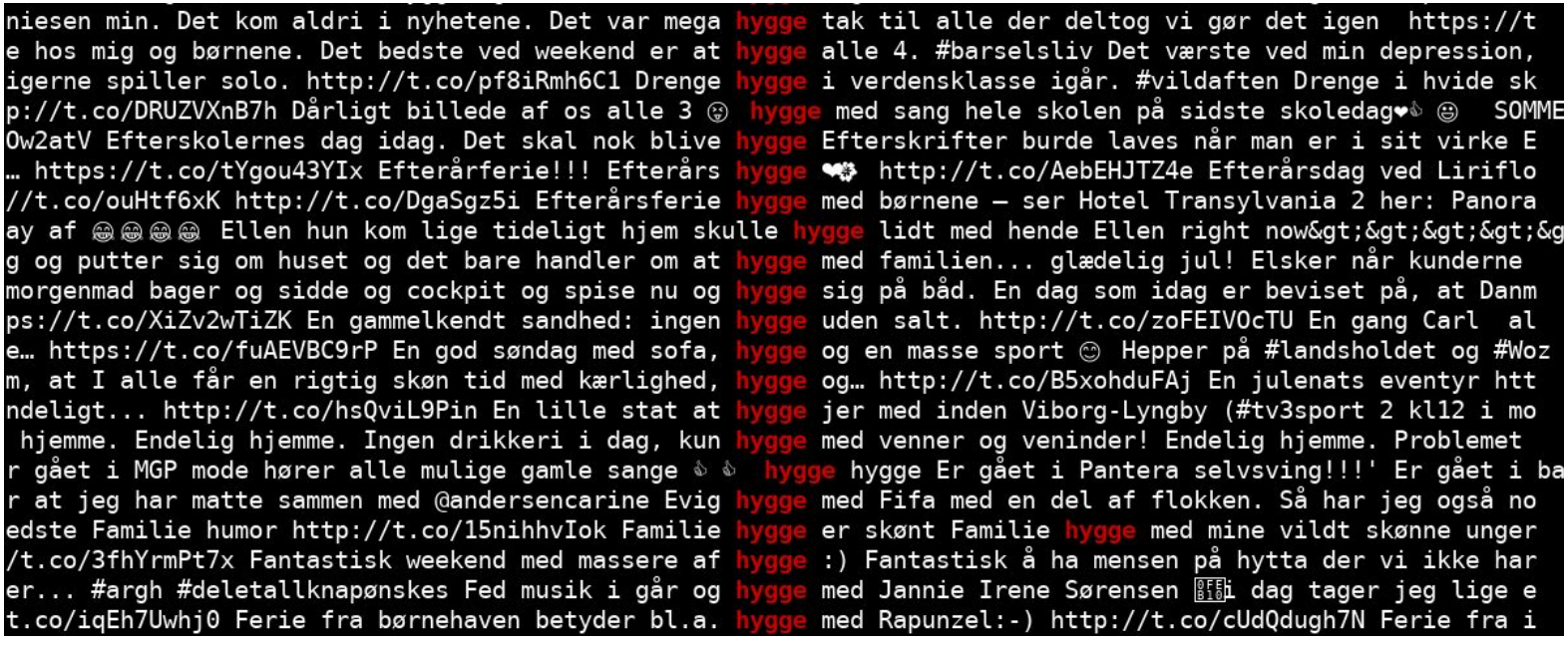
Formulated in the 50s by linguists (e.g., Harris 1954, Firth 1957)

A solution via Distributional Hypothesis

Formulated in the 50s by linguists (e.g., Harris 1954, Firth 1957)

"You shall know a word by the company it keeps" (Firth, J. R. 1957:11)

Wei Zhongxian i 1620'erne. Eunukkerne lod kejserne hygge sig med sine haremskvinder, mens de selv berigede
tnere fra ind- og udland. Der bydes på oplevelser, hygge og musik på Rue Mark lidt syd for Rudkøbings bygræ
k ifølge denne undersøgende kendt for "god mad" og hygge omkring måltidet. Danskernes mest almindelige målt
ofte som et begreb for „typisk dansk“. Navneordet hygge indebærer noget rart, afslappet, trygt og genkende
ke fysiske, behov er balancerede. Oftest forbindes hygge med social omgang. Negationen af "hyggelig" er "u
ke ord, "gezelligheid", har et lignende koncept af hygge både med hensyn til komfort og hygge. På tysk bety
amfund, hvor folk går mere ud for at arbejde eller hygge sig og samtidig bruger mindre tid hjemme. Udover a
 at træne og elsker at blive trænet med eller bare hygge med familien. Den vil ofte knytte sig til en beste
nter et par gange hver måned, blandt andet spil og hygge på Dybdalsparken. Foreningen arrangerer også en ræ
øde og lav forsøg. Det kan man sagtens få en masse hygge ud af. Se om du kan finde nogle guides på nettet t
 stedet forestillede man sig, at medlemmerne kunne hygge sig, mindes og bytte billeder. Det orienterende mø
fyrtårnet, men hver dag ror de ind til byen for at hygge sig med Vera og tilflytteren Jonas, kaldet 44. De
meget afslappet og sympatisk. Han kan godt lide at hygge og laver tit kager og kaffe til besætningen – spec
ttere studierelevante emner under Seminaret, eller hygge sig med fodboldturnering og andre sociale aktivite
art Christian var hjemme igen, kunne han alligevel hygge sig og grine. Han bruger dette eksempel overfor Ja
 de begrænsede midler man havde til rådighed kunne hygge sig sammen. De fleste bygninger med mere fra Seefl
lle færge, der udvikler sig til en konflikt mellem hygge og den udvikling, der truer med at ødelægge den lo
 Ej heller dette ægteskab skaffede Baggesen huslig hygge og rolige livsvilkår. Fanny, en kølig natur, der h
llige tricks eller bare køre rundt med vennerne og hygge sig. Skateboardet har altid været forvist fra cyke
s jævne, borgerlige, lidt småtskårne kunst en egen hygge og elskværdighed ved sig, der i forbindelse med de
ger om børn fra forældrehylden. De større børn kan hygge sig i deres område med gode bøger, pc'er og bordfo


niesen min. Det kom aldri i nyhetene. Det var mega hygge tak til alle der deltog vi gør det igen https://t
e hos mig og børnene. Det bedste ved weekend er at hygge alle 4. #barselsliv Det værste ved min depression,
igerne spiller solo. http://t.co/pf8iRmh6C1 Drenge hygge i verdensklasse igår. #vildaften Drenge i hvide sk
p://t.co/DRUZVXnB7h Dårligt billede af os alle 3 hygge med sang hele skolen på sidste skoledag❤ SOMME
Ow2atV Efterskolernes dag idag. Det skal nok blive hygge Efterskrifter burde laves når man er i sit virke E
… https://t.co/tYgou43YIx Efterårferie!!! Efterårs hygge http://t.co/AebEHJTZ4e Efterårsdag ved Liriflo
//t.co/ouHtf6xK http://t.co/DgaSgz5i Efterårsferie hygge med børnene – ser Hotel Transylvania 2 her: Panora
ay af Ellen hun kom lige tideligt hjem skulle hygge lidt med hende Ellen right now>>>>&g
g og putter sig om huset og det bare handler om at hygge med familien... glædelig jul! Elsker når kunderne
morgenmad bager og sidde og cockpit og spise nu og hygge sig på båd. En dag som idag er beviset på, at Danm
ps://t.co/XiZv2wTiZK En gammelkendt sandhed: ingen hygge uden salt. http://t.co/zoFEIVOcTU En gang Carl al
e… https://t.co/fuAEVBC9rP En god søndag med sofa, hygge og en masse sport Hepper på #landsholdet og #Woz
m, at I alle får en rigtig skøn tid med kærlighed, hygge og… http://t.co/B5xohduFAj En julenats eventyr htt
ndeligt... http://t.co/hsQviL9Pin En lille stat at hygge jer med inden Viborg-Lyngby (#tv3sport 2 kl12 i mo
hjemme. Endelig hjemme. Ingen drikkeri i dag, kun hygge med venner og veninder! Endelig hjemme. Problemet
r gået i MGP mode hører alle mulige gamle sange hygge hygge Er gået i Pantera selvsving!!!' Er gået i ba
r at jeg har matte sammen med @andersencarine Evig hygge med Fifa med en del af flokken. Så har jeg også no
edste Familie humor http://t.co/15nihhvIok Familie hygge er skønt Familie hygge med mine vildt skønne unger
/t.co/3fhYrmPt7x Fantastisk weekend med massere af hygge :) Fantastisk å ha mensen på hytta der vi ikke har
er... #argh #deletallknapønskes Fed musik i går og hygge med Jannie Irene Sørensen i dag tager jeg lige e
t.co/iqEh7Uwhj0 Ferie fra børnehaven betyder bl.a. hygge med Rapunzel:-) http://t.co/cUdQdugh7N Ferie fra i

"The company it keeps": Representing words by their context

- **Core idea:** The meaning of a word is represented by the words frequently occur close-by

Why talk about representations?

- Machine Learning, features are representations
- Better representations, better performance

(*some slides adapted from S.Riedel*)

What makes a good representation?

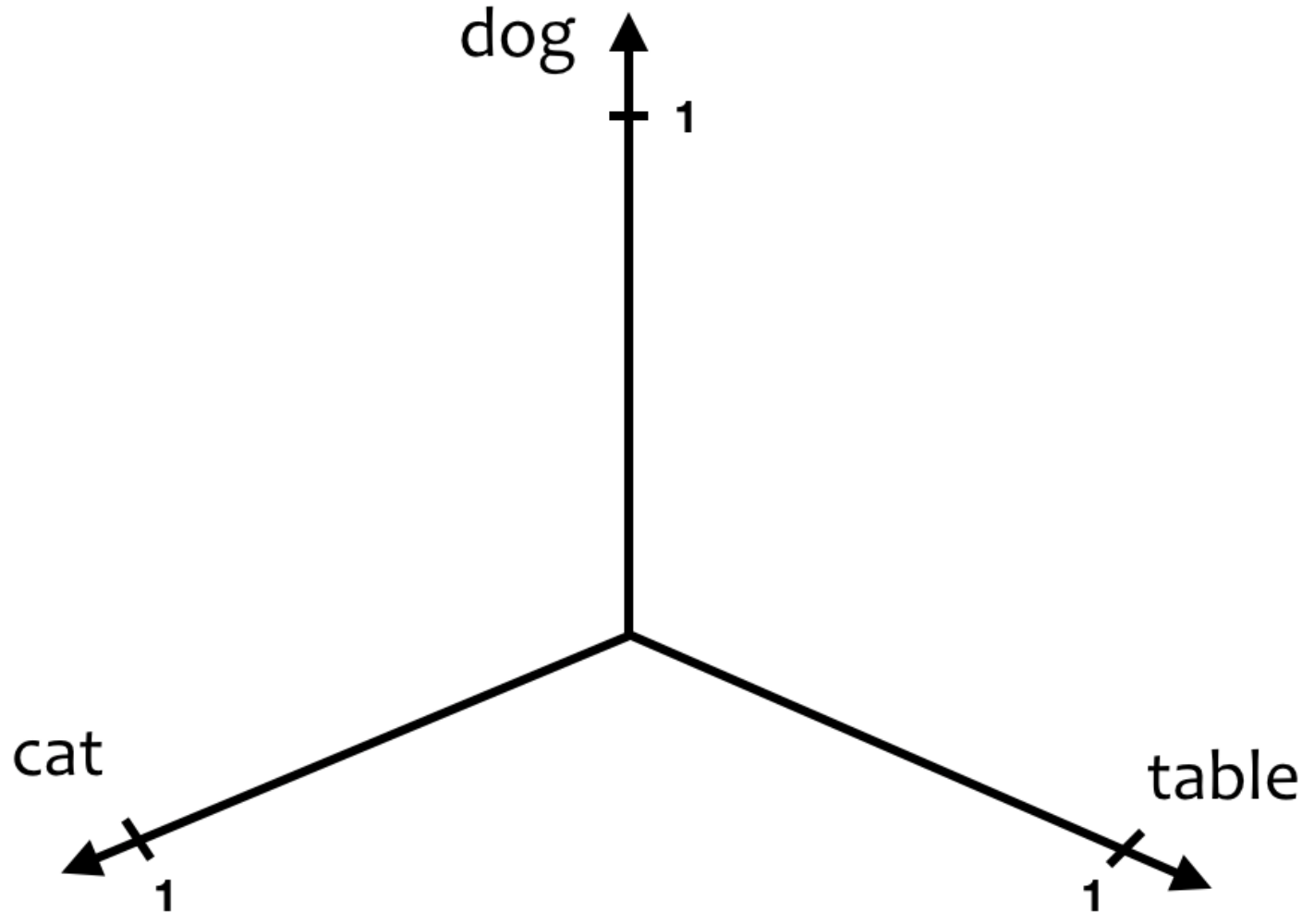
1. Representations are **distinct**
2. **Similar** words have **similar** representations

Sparse Binary Example

sb = sparse binary

- $\mathbb{V} = \{\text{cat}, \text{dog}, \text{table}\}$
- $f_{sb}(\text{cat}) = [1, 0, 0]$
- $f_{sb}(\text{dog}) = [0, 1, 0]$
- $f_{sb}(\text{table}) = [0, 0, 1]$

Sparse Binary Representations Visualised



Similarity on discrete representations? E.g., Hamming distance:

$$\mathbf{x}_{cat} \wedge \mathbf{x}_{dog} = 0$$

But our vectors are orthogonal. There is no natural notion of similarity in a set of one-hot vectors.

Problem with words as discrete symbols

Example: in web search, if user searches for "Seattle motel", we would like to match documents containing "Seattle hotel".

But:

motel = [0 0 0 0 0 0 0 0 0 0 1 0 0 0 0]
hotel = [0 0 0 0 0 0 0 1 0 0 0 0 0 0 0]

These two vectors are orthogonal.

There is no natural notion of **similarity** for one-hot vectors!

Solution:

- Could try to rely on WordNet's list of synonyms to get similarity?
  - But it is well-known to fail badly: incompleteness, etc.
- **Instead: learn to encode similarity in the vectors themselves**

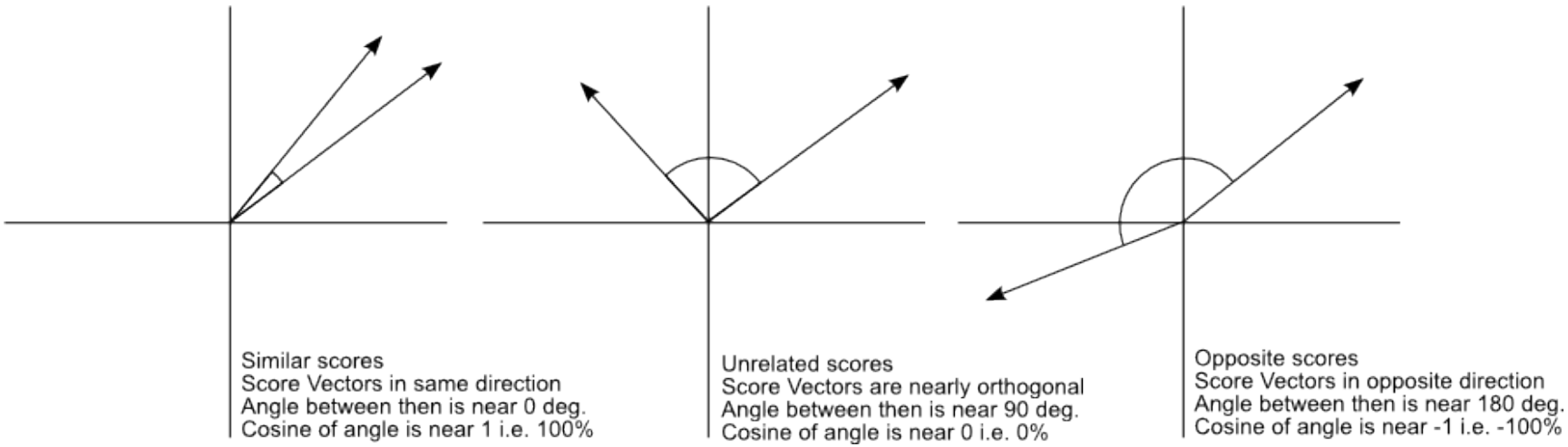
Cosine Similarity

$$cos_{\vec{u},\vec{v}} = \frac{\vec{u} \cdot \vec{v}}{||\vec{u}||||\vec{v}||} = \frac{\sum_1^n u_i v_i}{\sqrt{\sum_1^n u_i^2}\sqrt{\sum_1^n v_i^2}} \quad (1)$$

- $cos(u, v) \mapsto [-1, 1]$
- $cos(u, v) = 1$; identical
- $cos(u, v) = -1$; opposites
- $cos(u, v) = 0$; orthogonal

Note that the denominator normalizes for distance (norm of u * norm of v), we only care about direction!

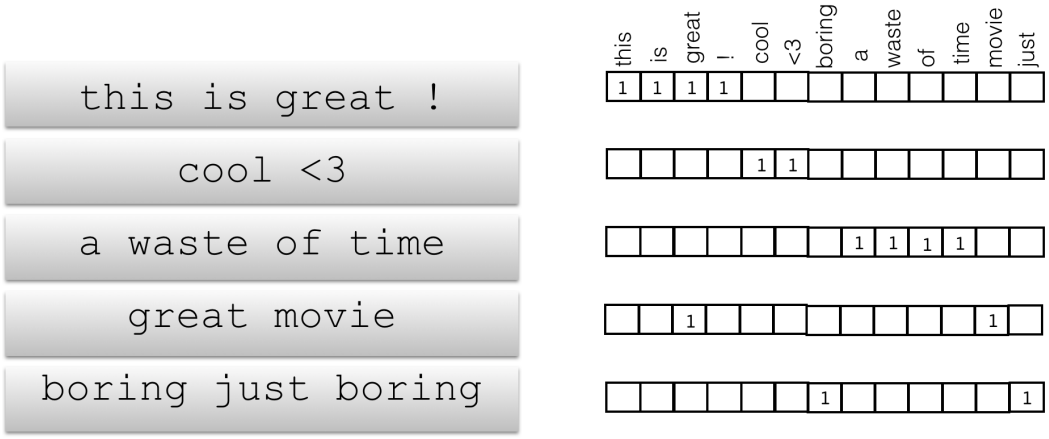
Cosine Similarity Visualised



Sparse Binary Similarities

- $cos(f_{sb}(\text{cat}), f_{sb}(\text{dog})) = 0$
- $cos(f_{sb}(\text{cat}), f_{sb}(\text{table})) = 0$
- $cos(f_{sb}(\text{table}), f_{sb}(\text{dog})) = 0$

Document representations



- Sparse binary vector
- For longer document, counts can/should be used instead of binary values (called *term-document matrix*)

Commonly used in information retrieval

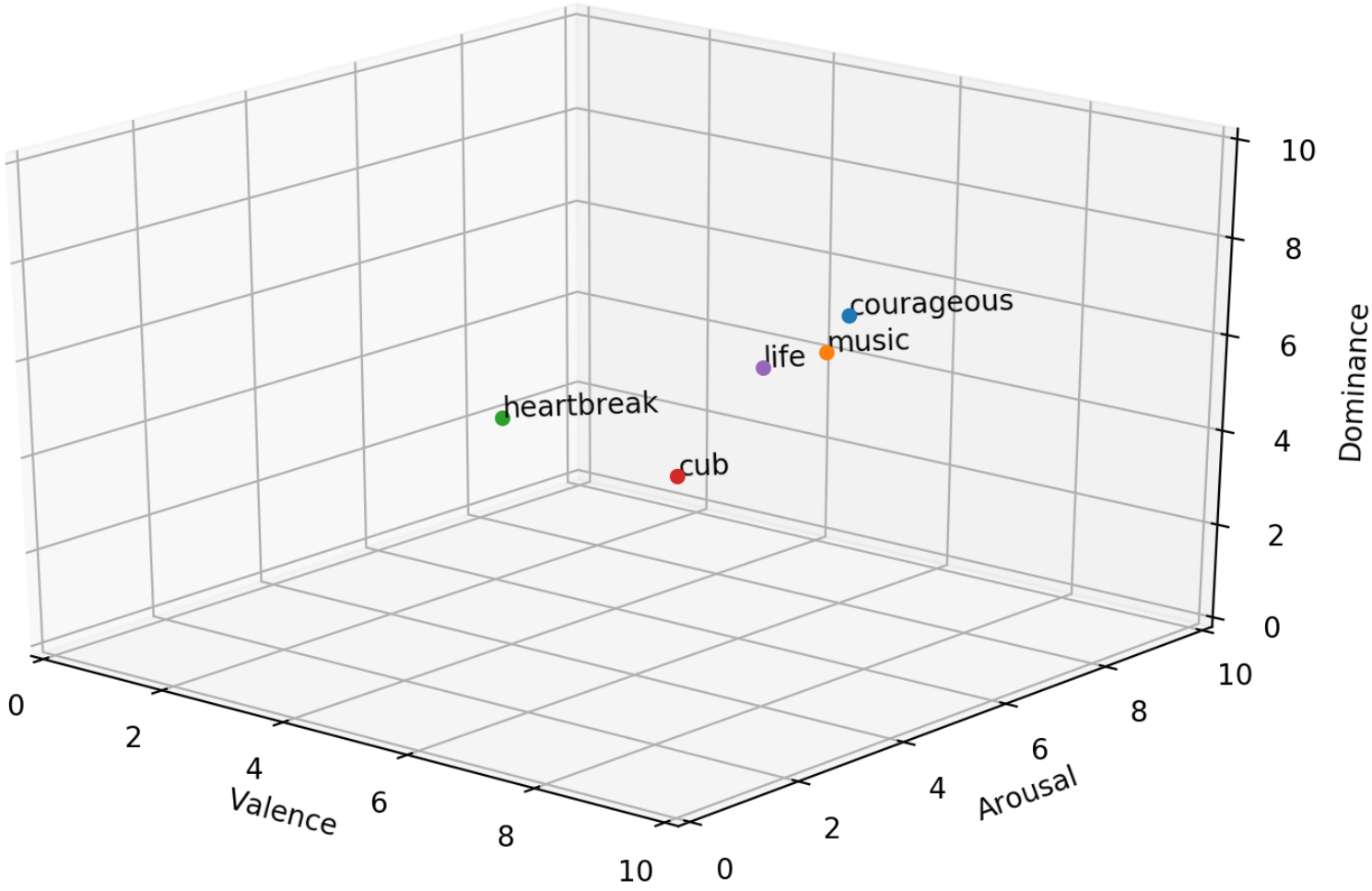
From sparse binary to dense continuous representations

Represent words with N (3-1000) continuous values.

From sparse binary to dense continuous representations

Represent words with N (3-1000) continuous values.

Word	Valence	Arousal	Dominance
courageous	8.05	5.50	7.38
music	7.67	5.57	6.50
heartbreak	2.45	5.65	3.58
cub	6.71	3.95	4.24
life	6.68	5.59	5.89


courageous
music
life
heartbreak
cub
Valence
Arousal
Dominance
0
2
4
6
8
10

Probably the biggest jump when moving from traditional linear models with sparse inputs to deep neural networks is to stop representing each feature as a unique dimension, but instead represent them as **dense vectors** (Goldberg, 2015).

Probably the biggest jump when moving from traditional linear models with sparse inputs to deep neural networks is to stop representing each feature as a unique dimension, but instead represent them as **dense vectors** (Goldberg, 2015).Formally:

- $f_{dc}(w) \mapsto \mathbb{R}^d$
- "Embed" words as matrix rows
- Dimensionality: d (hyperparameter)
- Word embedding matrix: $W \in \mathbb{R}^{|\mathbb{V}| \times d}$

Dense Continuous Example

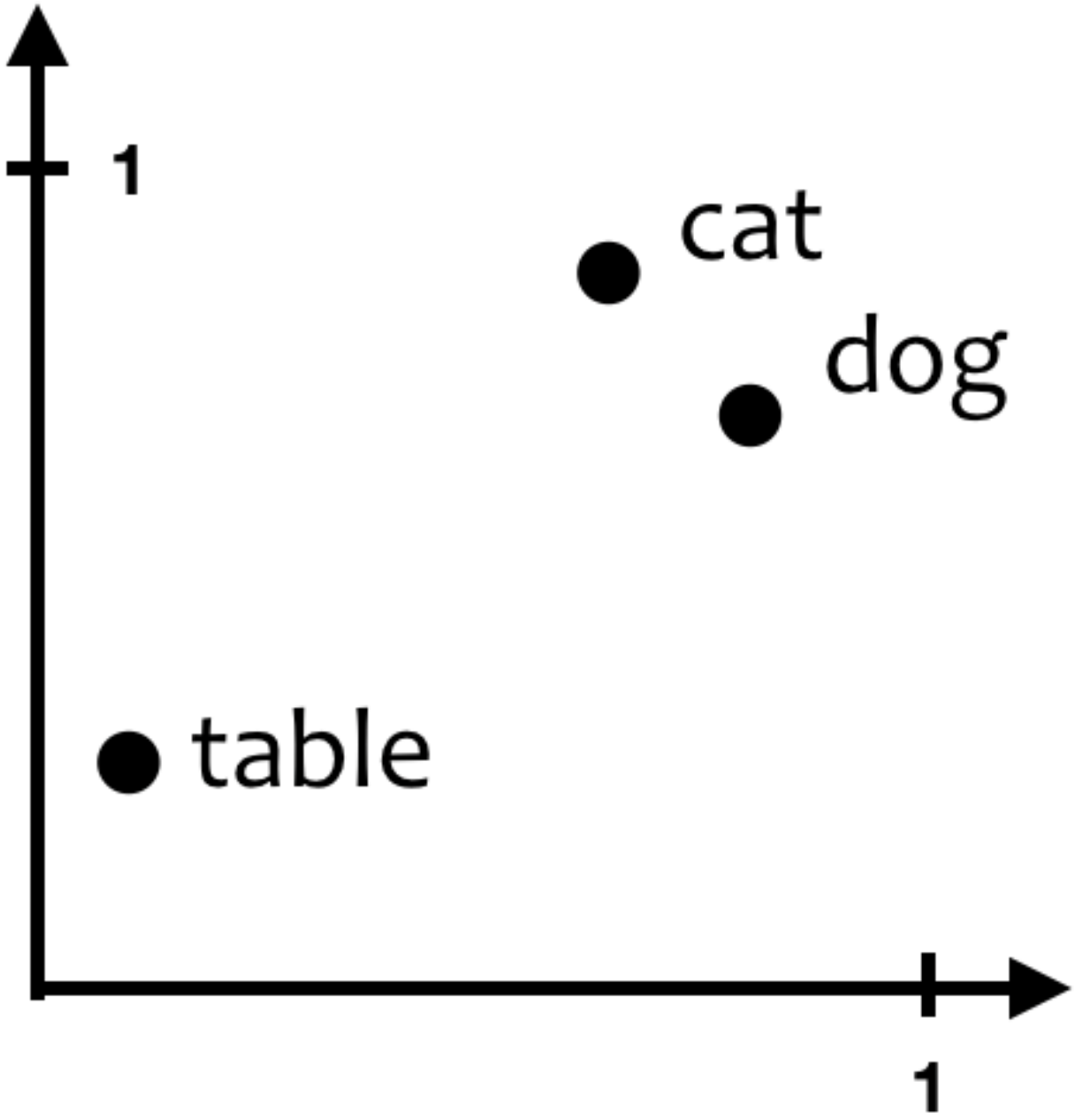
dc = dense continuous

- $\mathbb{V} = \{\text{cat}, \text{dog}, \text{table}\}$
- $d = 2$
- $W \in \mathbb{R}^{3\times 2}$

where:

- $f_{dc}(\text{cat}) = [0.7, 0.8]$
- $f_{dc}(\text{dog}) = [0.75, 0.6]$
- $f_{dc}(\text{table}) = [0.1, 0.15]$

Dense Continuous Representations Visualised



Word vectors

Instead of using discrete representations, we will **embed** words into a high-dimensional feature space and represent each word by a lower-dimensional dense *vector* (aka. **word vector**), chosen such that its representation is close to vectors of words that appear in similar contexts.

Word vectors

Note: *word vectors* are sometimes called *word embeddings* or *word representations*.

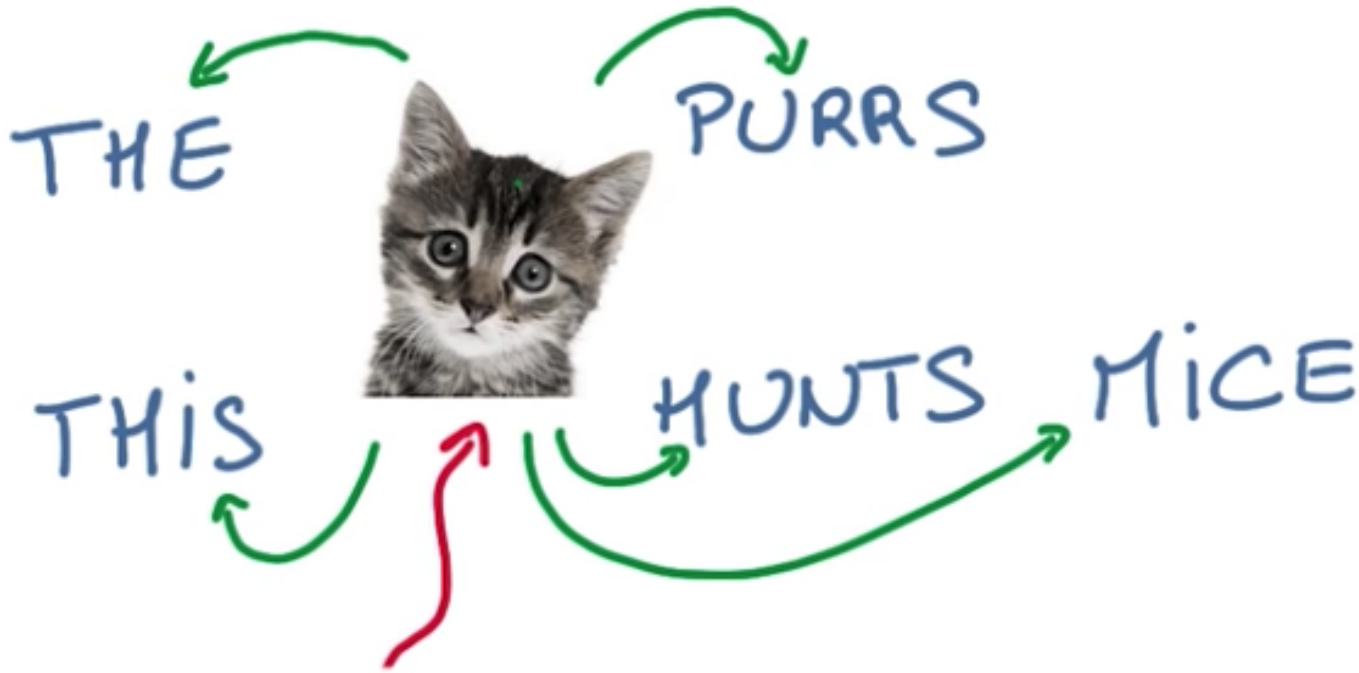
$$\textit{linguistics} = \begin{pmatrix} 0.286 \\ 0.792 \\ -0.177 \\ -0.107 \\ 0.109 \\ -0.542 \\ 0.349 \\ 0.271 \end{pmatrix}$$

expect =

$$\begin{pmatrix} 0.286 \\ 0.792 \\ -0.177 \\ -0.107 \\ 0.109 \\ -0.542 \\ 0.349 \\ 0.271 \\ 0.487 \end{pmatrix}$$

How can we obtain such word embeddings?

"The company it keeps" Core idea: learn the meaning of a word by looking at lots of lots of text (*unsupervised learning*) Core assumption: *similar words tend to occur in similar contexts*


THE
PURRS
THIS
HUNTS
MICE

- Traditionally: decompose a word co-occurence matrix (vector space models, distributional semantics)
- Neural world: a simple idea that works very well

Method 1 (traditional): Count! (aka Count-based methods)

We can represent documents based on the words they contain:

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

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

From this we can also represent words based on which documents they appear:

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

Figure 6.5 The term-document matrix for four words in four Shakespeare plays. The red boxes show that each word is represented as a row vector of length four.

From this we can also represent words based on which documents they appear:

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

Figure 6.5 The term-document matrix for four words in four Shakespeare plays. The red boxes show that each word is represented as a row vector of length four.

This is more useful for information retrieval

Co-occurences

We can represent the "company" of a word in terms of a **word context (word co-occurence) matrix**. On the rows we have the words, on the columns their context.**Contexts** can be of different types, for example:

- entire documents
- paragraphs
- a window around the word

Getting word vectors by count-based method:

- Create the word-context matrix (count how often word appears in context) of $|v|$ * $|v|$ (vocabulary size)
- Optionally:
- weight the counts (e.g., PMI or tf-idf)
- Optionally reduce dimensions using singular value decomposition (SVD)

What you get: a vector representation of each word; can measure the closeness of words in this resulting *word vector space*

Getting word vectors by count-based method:

- Create the word-context matrix (count how often word appears in context) of $|v|$ * $|v|$ (vocabulary size)
- Optionally:
- weight the counts (e.g., PMI or tf-idf)
- Optionally reduce dimensions using singular value decomposition (SVD)

What you get: a vector representation of each word; can measure the closeness of words in this resulting *word vector space*

	...	cat	dog	weather	...
...					
cat		-	77	0	
dog		77	-	1	
weather		0	1	-	
...					

When both dimensions in the matrix are words, PMI is most commonly used: $PMI(w, c) = log_2 \frac{P(w,c)}{P(w)P(c)}$

- w : word
- c : context

Case study, setup:

- English tweets of 1 day (1788398)
- nltk TweetTokenizer + lowercasing
- Counts based on window of 2
- Top 50,000 words
- Cosine versus PMI

	cos_dist	pmi
cat-dog	0.31	4.62
cat-weather	0.55	0.00
cat-the	0.26	0.18
cat-cats	0.36	5.16

- Cosine distance: higher = more distant
- PMI: higher = more informative

TF-IDF

- Term Frequency - Inverse Document Frequency
- Takes into account: How often does the word occur in the document, how often does the word occur in all documents
- How important is a word in a document?

We are going to assume documents are collections of tweets from the same user

- so we model the informativity of words for a specific user: how distinguishing are words for this user.

First try:

- Count the most frequent words for this user

$tf_{t,d} = count(t, d)$

First try:

- Count the most frequent words for this user

$tf_{t,d} = count(t, d)$

```
cat twitUsers/robvanderg.txt | cut -f 2 | grep -o "\b[a-zA-
sort | uniq -c | sort -nr | head
     27 the
     15 to
     15 t
     15 https
     15 co
     13 is
      9 it
      9 and
      7 this
      7 in
```

```
from nltk.tokenize import TweetTokenizer
tok = TweetTokenizer()
def path2freqs(path):
    freqs = {}
    for line in open(path):
        line = line.split('\t')[1] # remove username
        for word in tok.tokenize(line):
            if word not in freqs:
                freqs[word] = 1
            else:
                freqs[word] += 1
    return freqs
userFreqs = path2freqs('twitUsers/robvanderg.txt')
print('\n'.join([k + '\t' +str(userFreqs[k]) for k in sorte
```

Output:

```
,36
the 27
. 16
to 15
! 13
is 13
? 13
: 10
( 10
) 10
```

- Are these really informative?
- Can we do better?

TF-IDF

- In how many of the documents does this word occur?

$$\frac{C(word, docs)}{|docs|}$$

- A rare word: $1/100 = 0.01$ (the chance that a random document contains this word)

TF-IDF

- In how many of the documents does this word occur?

$$\frac{C(word, docs)}{|docs|}$$

- A rare word: $1/100 = 0.01$ (the chance that a random document contains this word)
- But we want to know how common a word is: reverse

$$\frac{|docs|}{C(word, docs)}$$

- A rare word: $100/1 = 100$
- Inverse Document Frequencey: i.e. It occurs in 1 out 100 documents

$$idf_t = \frac{N}{df_t}$$

TF-IDF

Combine: $w_{t_d} = tf_{t,d} * idf_t \;\; w_{t,d} = count(t,d) * \frac{N}{df_t}$

Highest TF-IDF for this user:

```
position 404.00 4
training 303.00 3
model 252.50 5
applying 202.00 2
scope 202.00 2
hesitate 202.00 2
Deadline 202.00 2
14-03 202.00 2
https://t.co/ePBhAVxPTJ 202.00 2
https://t.co/8TQpvkit6V 202.00 2
```

TF-IDF for frequent words:

```
,49.81 36
the 37.36 27
. 20.72 16
to 19.18 15
! 16.41 13
is 17.51 13
? 20.52 13
: 10.74 10
( 28.06 10
```

Method 2 - Predict! (aka Prediction-based methods)

- **Idea**: directly learn word vectors, i.e., **predict** the words with a neural network

Main idea of prediction-based approaches

- instead of capturing co-occurence statistics of words
- **predict context** (surrounding words of every word) based on word, or word based on context.
- Most prominent approach: word2vec (Mikolov et al., 2013)

Basic idea of learning neural network word embeddings

We define a model that aims to predict between a center word w_t and context words in terms of word vectors

$$p(\text{context} | w_t) = \ldots$$

which has a loss function, e.g.,

$$J = 1 - p(w_{-t} | w_t)$$

We look at many positions t in a big language corpus

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

- slides from http://web.stanford.edu/class/cs224n/slides/cs224n-2021-lecture01-wordvecs1.pdf

Idea: Directly learn low-dimensional word vectors based on ability to predict

- Old idea. Relevant for this lecture & deep learning:
  - Learning representations by back-propagating errors. (Rumelhart et al., 1986)
  - A neural probabilistic language model (Bengio et al., 2003)
  - NLP (almost) from Scratch (Collobert & Weston, 2008)
  - A recent, even simpler and faster model: word2vec (Mikolov et al. 2013) → intro now
  - The GloVe model from Stanford (Pennington, Socher, and Manning 2014) connects back to matrix factorization

Word2vec (Mikolov et al. 2013) is a framework for learning word vectors

Idea:

- We have a large corpus of text
- Every word in a fixed vocabulary is represented by a vector
- Go through each position *t* in the text, which has a center word *c* and context ("outside") words *o*
- Use the similarity of the word vectors for *c* and *o* to calculate the probability of *o* given *c* (or vice versa)
- Keep adjusting the word vectors to maximize this probability

‘Word2vec‘ contains two algorithms

1. Skip-grams
2. CBOW

In the lecture we will look at skip-grams, in the lab you will implement CBOW

Preview of the two models

[Mikolov et al., 2013](https://arxiv.org/pdf/1301.3781.pdf)

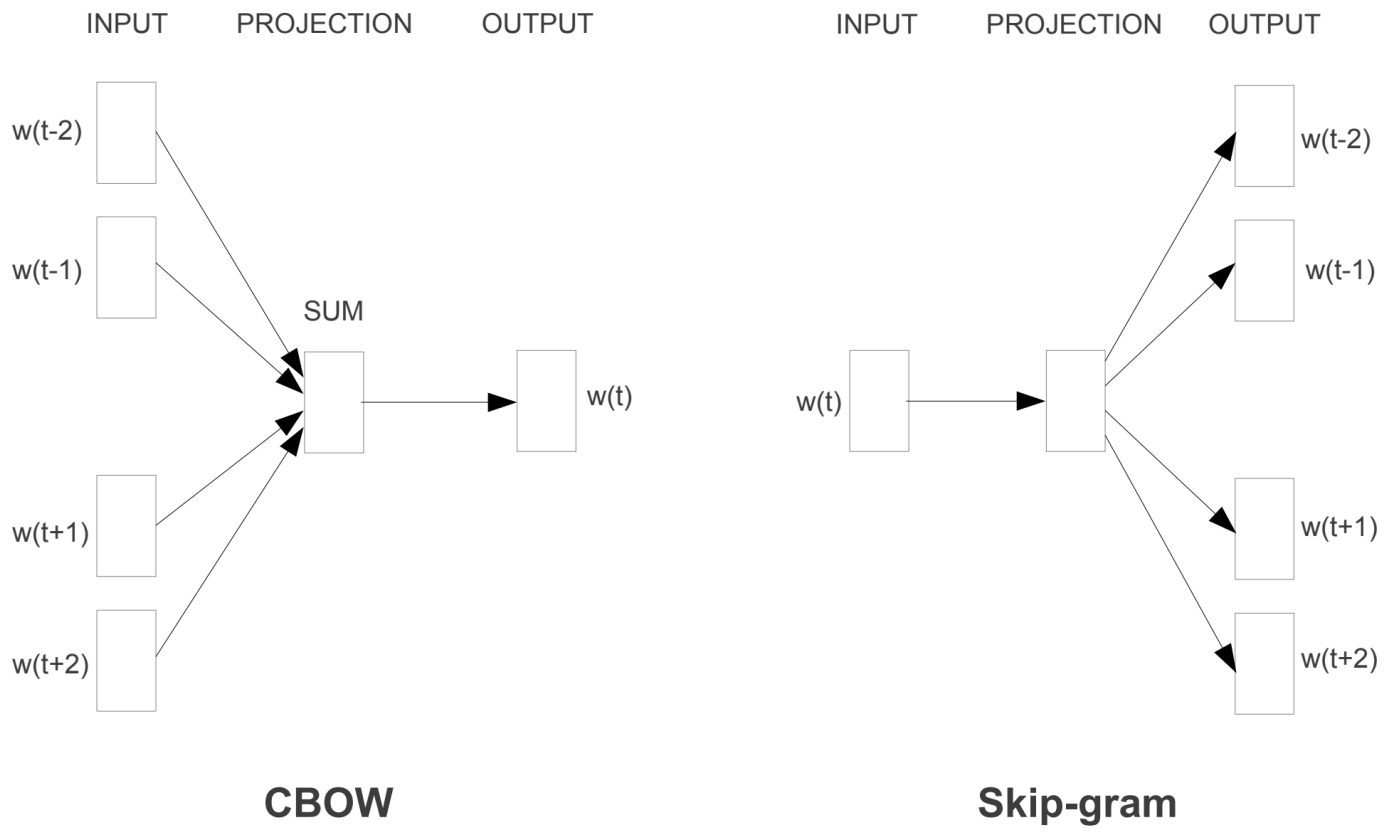
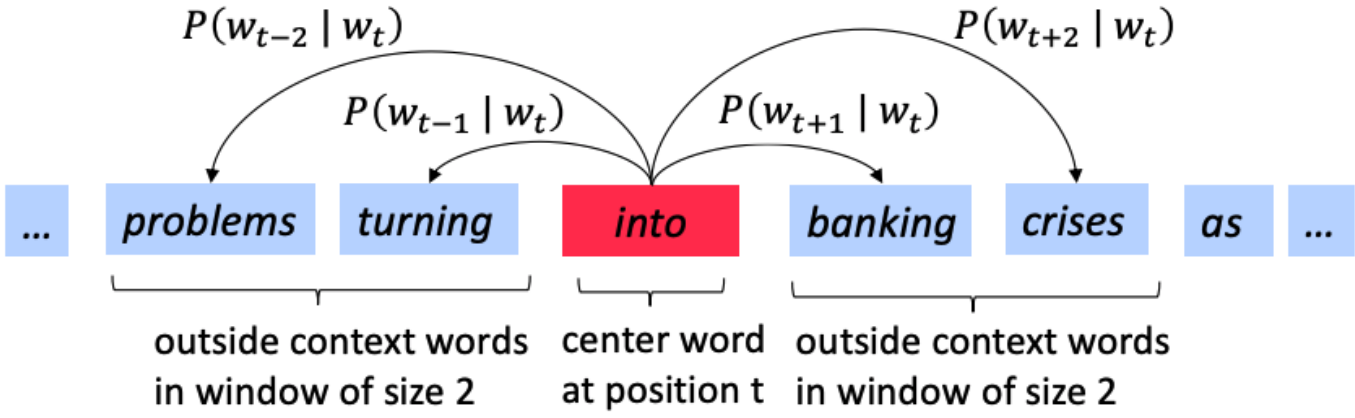


Figure 1: New model architectures. The CBOW architecture predicts the current word based on the context, and the Skip-gram predicts surrounding words given the current word.

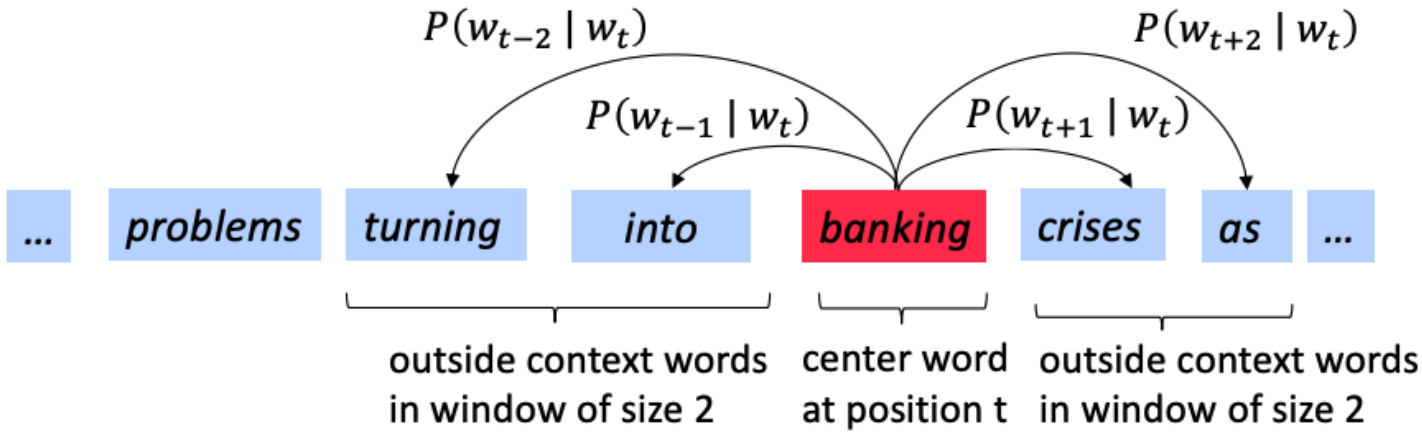
Word2Vec Overview

- Example windows and process for computing $P(w_{t+j} \mid w_t)$



Word2Vec Overview

- Example windows and process for computing $P(w_{t+j} \mid w_t)$



Note: only one probability distribution, that of the output word appearing close to the center word

Details of word2vec

- Given a large corpus of text
- Go through each position t in the text, which has a center word c and context ("outside") words o
- Use the similarity of the word vectors for c and o to calculate the probability of o given c

Word2vec: objective function

For each position $t = 1, \ldots, T$, predict context words within a window of fixed size m, given center word w_j.

$$\text{Likelihood} = L(\theta) = \prod_{t=1}^{T} \prod_{\substack{-m \le j \le m \\ j \neq 0}} P(w_{t+j} \mid w_t; \theta)$$

θ is all variables to be optimized

sometimes called *cost* or *loss* function

The objective function $J(\theta)$ is the (average) negative log likelihood:

$$J(\theta) = -\frac{1}{T} \log L(\theta) = -\frac{1}{T} \sum_{t=1}^{T} \sum_{\substack{-m \le j \le m \\ j \neq 0}} \log P(w_{t+j} \mid w_t; \theta)$$

Minimizing objective function $\Leftrightarrow$ Maximizing predictive accuracy

θ is the vector representation of the words

Word2vec: objective function

- We want to minimize the objective function:

$$J(\theta) = -\frac{1}{T}\sum_{t=1}^{T}\sum_{\substack{-m\leq j\leq m \\ j\neq 0}} \log P\left(w_{t+j} \mid w_t; \theta\right)$$

- Question: How to calculate $P\left(w_{t+j} \mid w_t; \theta\right)$?
- Answer: We will *use two* vectors per word *w*:
  - v_w when *w* is a center word
  - u_w when *w* is a context word
- Then for a center word *c* and a context word *o*:

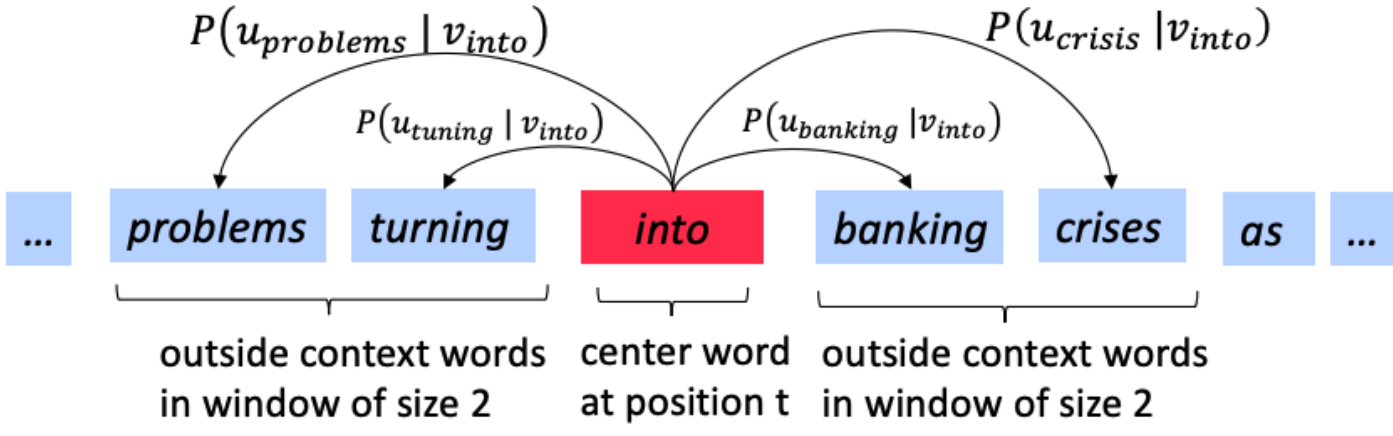
$$P(o|c) = \frac{\exp(u_o^T v_c)}{\sum_{w\in V}\exp(u_w^T v_c)}$$

Dot product

- dot product is a kind of similarity function: bigger if u and v more similar
- softmax to put them into a probability distribution

Word2Vec Overview with Vectors

- Example windows and process for computing $P(w_{t+j} \mid w_t)$
- $P(u_{problems} \mid v_{into})$ short for $P(problems \mid into\ ; u_{problems}, v_{into}, \theta)$



Word2vec: prediction function

Exponentiation makes anything positive

$$P(o|c) = \frac{\exp(u_o^T v_c)}{\sum_{w \in V} \exp(u_w^T v_c)}$$

Dot product compares similarity of *o* and *c*.
$u^T v = u.v = \sum_{i=1}^{n} u_i v_i$
Larger dot product = larger probability

Normalize over entire vocabulary
to give probability distribution

- This is an example of the **softmax function** $\mathbb{R}^n \to \mathbb{R}^n$

$$\text{softmax}(x_i) = \frac{\exp(x_i)}{\sum_{j=1}^{n} \exp(x_j)} = p_i$$

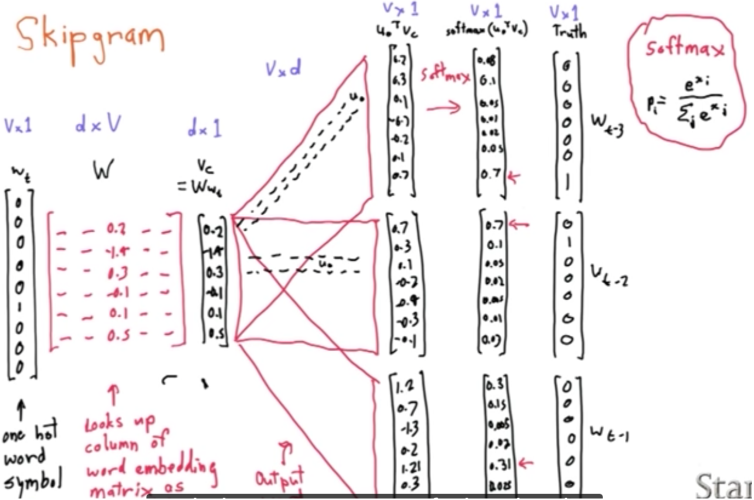
- The softmax function maps arbitrary values x_i to a probability distribution p_i
  - "max" because amplifies probability of largest x_i
  - "soft" because still assigns some probability to smaller x_i
  - Frequently used in Deep Learning

To train the model: Compute all vector gradients!

- Recall: θ represents all model parameters, in one long vector
- In our case with *d*-dimensional vectors and *V*-many words:

$$\theta = \begin{bmatrix} v_{aardvark} \\ v_a \\ \vdots \\ v_{zebra} \\ u_{aardvark} \\ u_a \\ \vdots \\ u_{zebra} \end{bmatrix} \in \mathbb{R}^{2dV}$$

- Remember: every word has two vectors
- We optimize these parameters by walking down the gradient


Skipgram
V x 1
d x V
d x 1
V x d
V x 1
V x 1
V x 1
$u_o^T v_c$
softmax($u_o^T v_c$)
Truth
w_t
W
$v_c = W w_t$
softmax
u_o
w_{t-3}
w_{t-2}
w_{t-1}
Softmax
$p_i = \frac{e^{x_i}}{\sum_j e^{x_j}}$
one hot word symbol
looks up column of word embedding matrix as
output

The probability of a word in the context (o) given the current word c is:

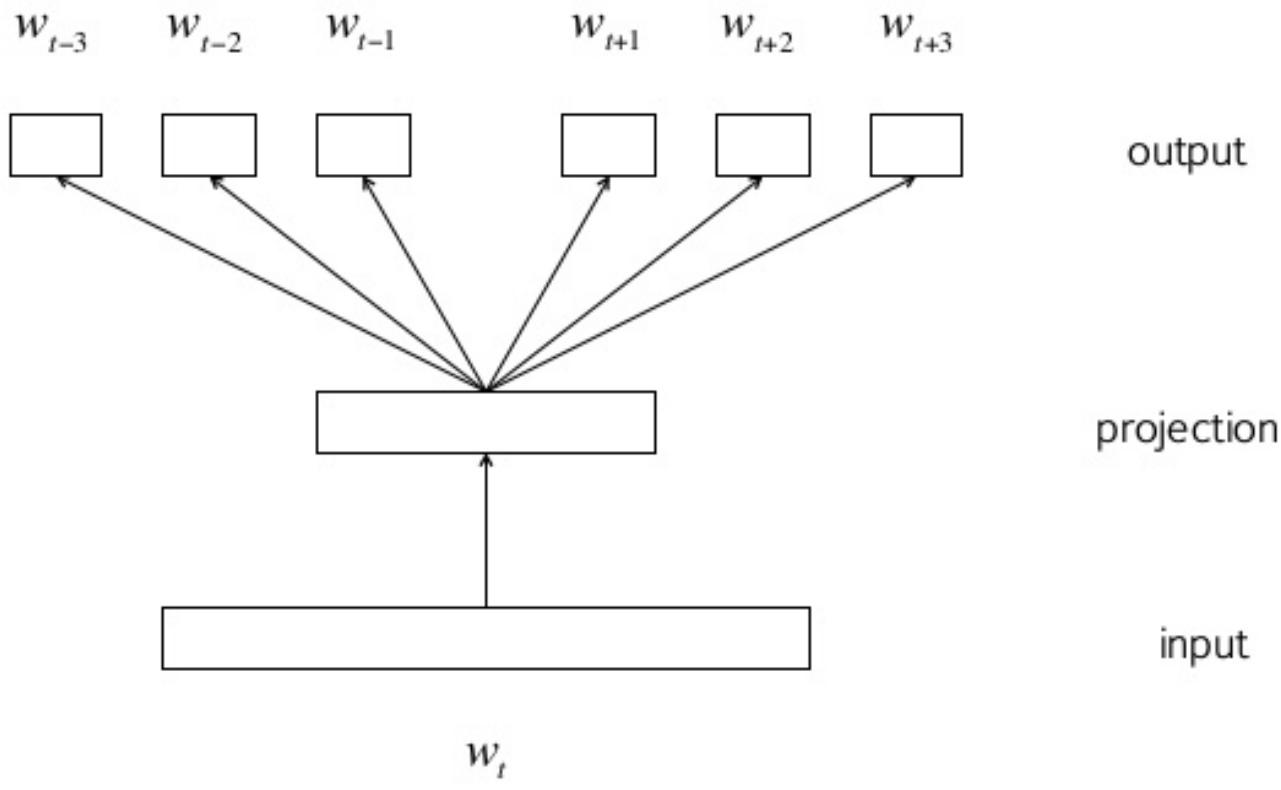
$$p(o|c) = \frac{exp(u_o^T v_c)}{\sum_{w=1}^{W} exp(u_w^T v_c)}$$

- What is the most computationally expensive part?

Solutions:

- hierarchical softmax: calculate the softmax more efficiently
- negative subsampling: compare positive to negative class
- subsampling: downsample frequent words

Skip-gram model



Result: two embedding spaces v for center words and u for context words

- v is kept as a lookup matrix for downstream tasks
- u is thrown away

What can we do with these embeddings?

What can we do with these embeddings?

- Find distance between words (also find close words)
- Represent words with numbers! very useful for NLP, which we will see in coming weeks
- ...

What can we do with these embeddings?

- Find distance between words (also find close words)
- Represent words with numbers! very useful for NLP, which we will see in coming weeks
- ...
- Solve word analogies?

```
>>> import gensim.models
>>> import pprint
>>> twitEmbs = gensim.models.KeyedVectors.load_word2vec_for
...                                          'twitter.bin', binary=T
>>>
>>> pprint.pprint(twitEmbs.most_similar('tuesday'))
[('wednesday', 0.978216826915741),
 ('thursday', 0.967643857002258 3),
 ('monday', 0.9522584676742554),
 ('friday', 0.941669583320617 7),
 ('saturday', 0.9219995141029358),
 ('tuesday,', 0.8900223970413208),
 ('sunday', 0.884716570377 3499),
 ('wednesday,', 0.876380562782 2876),
 ('thursday,', 0.873512208461 7615),
 ('thrusday', 0.865790069103241)]
```

```
>>> pprint.pprint(twitEmbs.most_similar('thrusday'))
[('wensday', 0.922013700083923),
 ('thurday', 0.888260722160339),
 ('thursday', 0.886656803787231),
 ('wednsday', 0.884959459304 8096),
 ('wednesday', 0.883491158485 4126),
 ('saterday', 0.86933833360672),
 ('tuesday', 0.865790128707 8857),
 ('firday', 0.865045785903 9307),
 ('wendsday', 0.863972783088 6841),
 ('wenesday', 0.862274467945 0989)]
```

OUTPUT:

```
[('wensday', 0.922013700083923),
 ('thurday', 0.888260722160339),
 ('thursday', 0.886656820774078 4),
 ('wednsday', 0.884959518909 4543),
 ('wednesday', 0.883491098880 7678),
 ('saterday', 0.869338274002 0752),
 ('tuesday', 0.86579006910 3241),
 ('firday', 0.86504572629928 59),
 ('wendsday', 0.86397278308868 41),
 ('wenesday', 0.86227440834045 41)]
```

Word Analogies

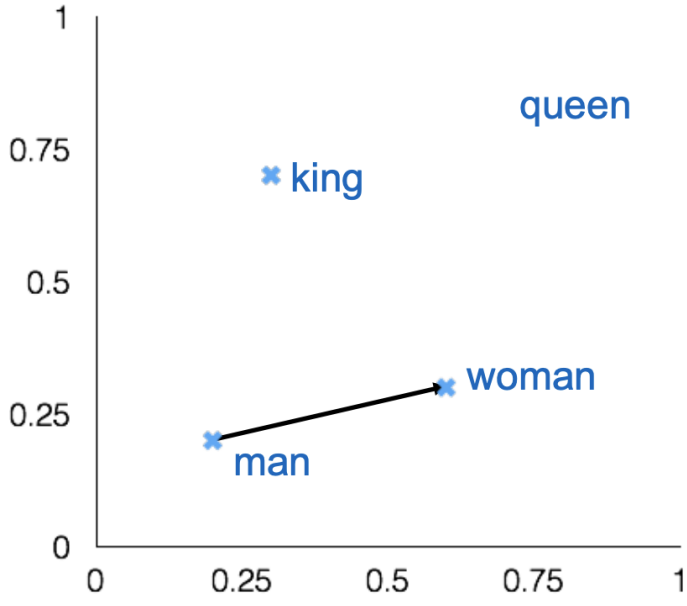
Test for linear relationships, examined by Mikolov et al.

a:b :: c:?

$$d = \arg\max_x \frac{(w_b - w_a + w_c)^T w_x}{||w_b - w_a + w_c||}$$

man:woman :: king:?

+	king	[0.30 0.70]
−	man	[0.20 0.20]
+	woman	[0.60 0.30]
	queen	[0.70 0.80]



However, be aware that the reality is different!

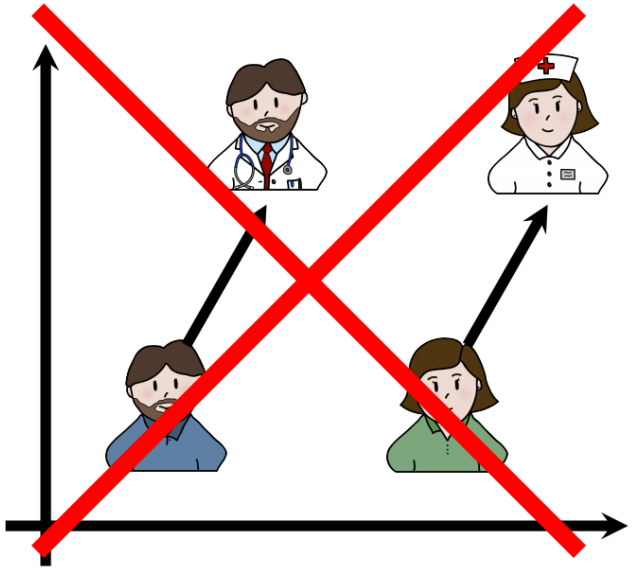
However, be aware that the reality is different!

Fair is Better than Sensational:
Man is to Doctor as Woman is to Doctor

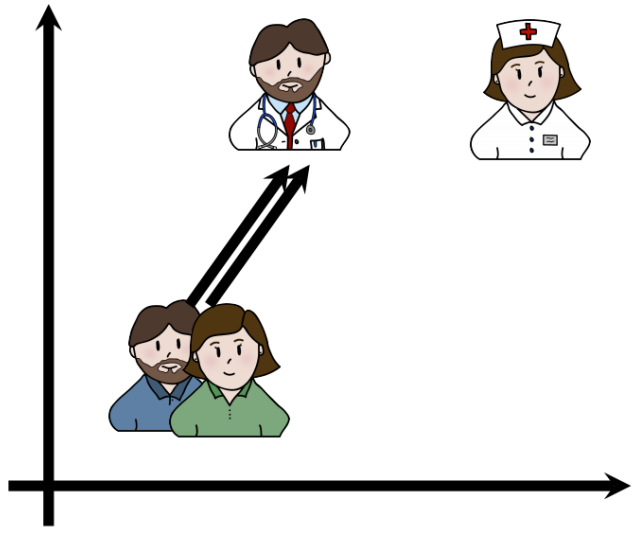
Malvina Nissim, Rik van Noord and Rob van der Goot

university of groningen

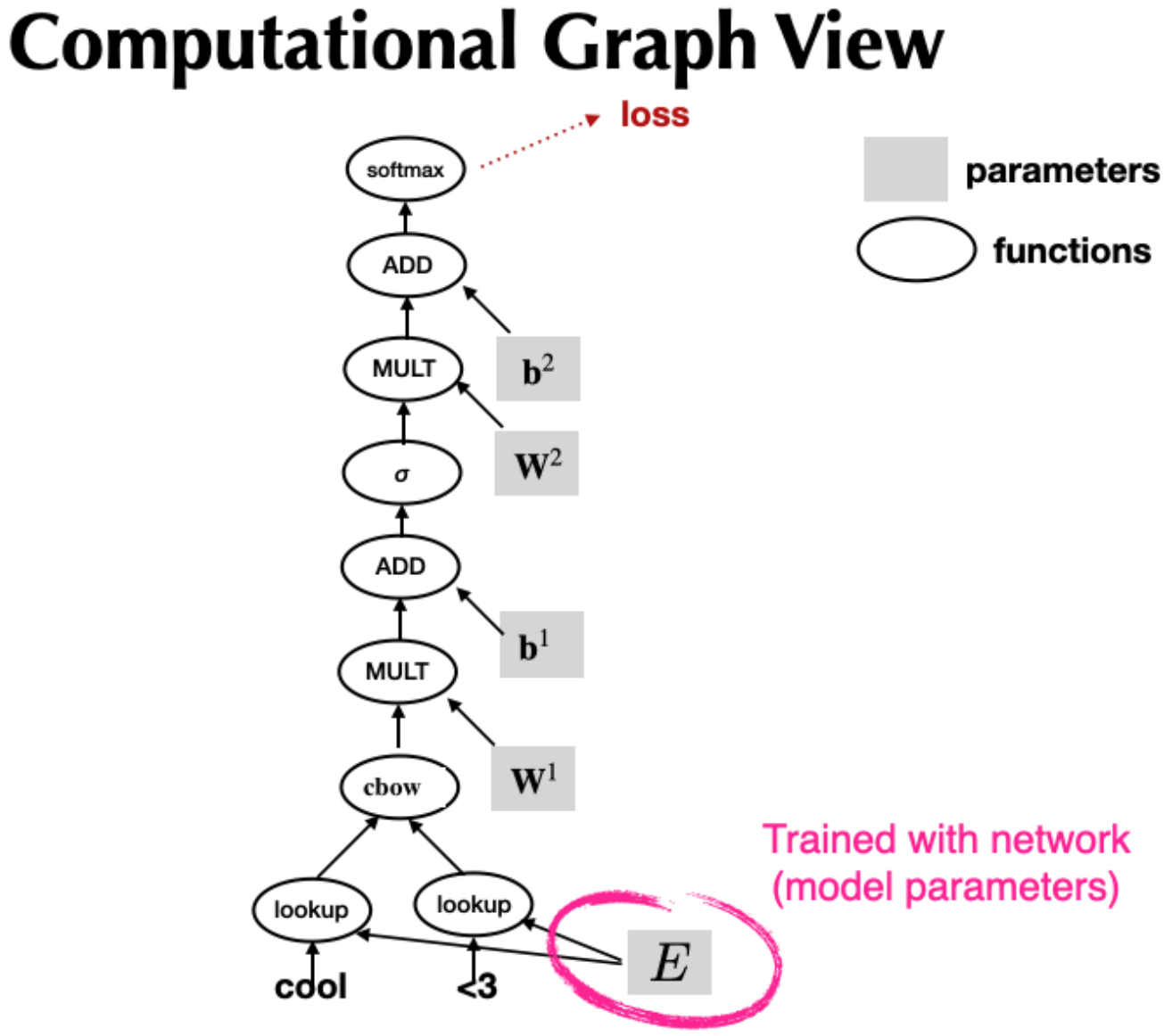
What people think embeddings encode:



What embeddings really encode:



In a FFNN


Computational Graph View
loss
softmax
ADD
MULT
σ
ADD
MULT
cbow
lookup
lookup
cool
<3
b2
W2
b1
W1
E
parameters
functions
Trained with network
(model parameters)

To sum up: Word Embeddings

Remember, today we have seen two ways to get embeddings:

- Traditional methods (also called 'count' methods)
- Neural method #1 (also called 'predict' methods)

What makes language so challenging

- Ambiguity
- Zipf's law
- Non-deterministic
- Language changes constantly

Inputs of different lengths

In our classification example today we have one simplification: the input is always of the same size (namely, n words, a fixed window). However, in NLP we typically never have fixed size inputs, sentences are of different length. The neural network however needs inputs of fixed size. So how to deal with it? That's for the next lecture.

References

- Baroni et al., (2014) Don't count, predict! A systematic comparison of context-counting vs. context-predicting semantic vectors
- Mikolov et al., (2013) Distributed Representations of Words and Phrases and their Compositionality