Advanced Machine Learning For Design

Lecture 3 - Machine Learning and Natural Language Processing / Part 2

Evangelos Niforatos

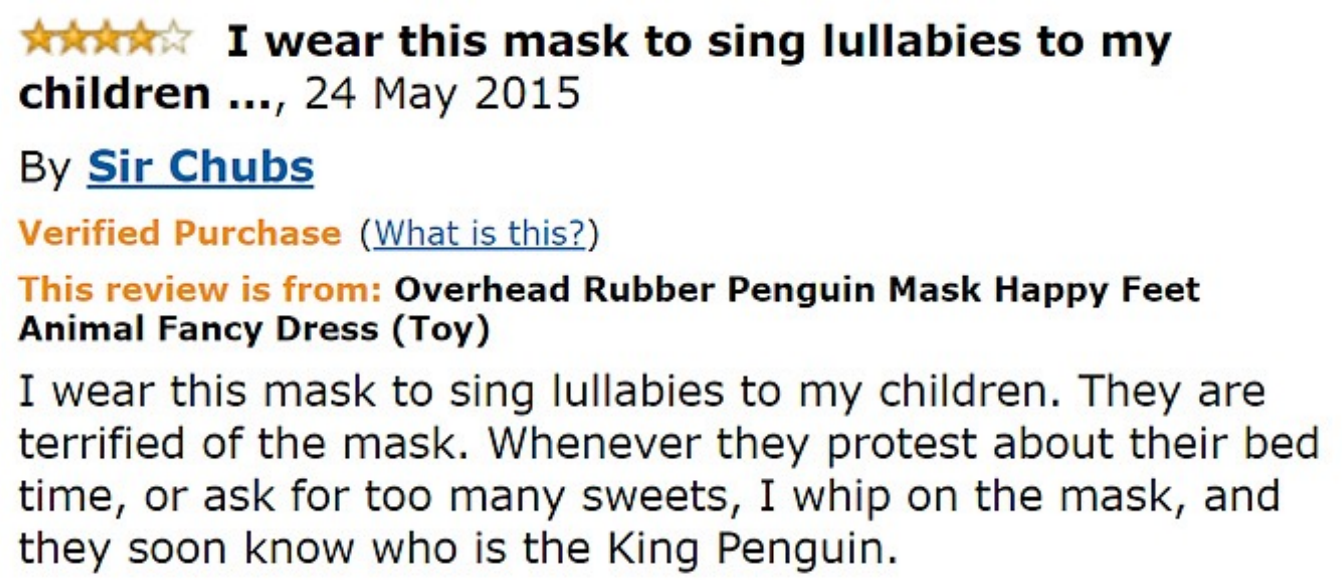
05/10/2022

aml4d-ide@tudelft.nl
https://aml4design.github.io/

Previously, on ML4D….

Textual documents

- A sequence of alphanumerical characters
  - Short: e.g. tweets
  - Long: e.g Web documents, interview transcripts
- Features are (set of) words
  - Words are also syntactically and semantically organised
- Feature values are (set of) words occurrences
- Dimensionality —> at least dictionary size



★★★★☆ **I wear this mask to sing lullabies to my children ...**, 24 May 2015

By **Sir Chubs**

Verified Purchase (What is this?)

This review is from: Overhead Rubber Penguin Mask Happy Feet Animal Fancy Dress (Toy)

I wear this mask to sing lullabies to my children. They are terrified of the mask. Whenever they protest about their bed time, or ask for too many sweets, I whip on the mask, and they soon know who is the King Penguin.

Document →

I	Wear	Mask	...	W(n)	Class
1	1	1		0	Spam
0	0	1		0	Not Spam
					Spam

Main types of NLP Tasks

- Label a region of text
  - e.g. part-of-speech tagging, sentiment classification, or named-entity recognition

- Link two or more regions of text
  - e.g. *coreference* (are two mentions of a real-world thing (e.g. a person, place, or some other named entity(are in fact referencing the same real-world thing?

- Fill in missing information (missing words) based on context

Language Representation

Language = vocabulary and its usage in a specific context captured by textual data

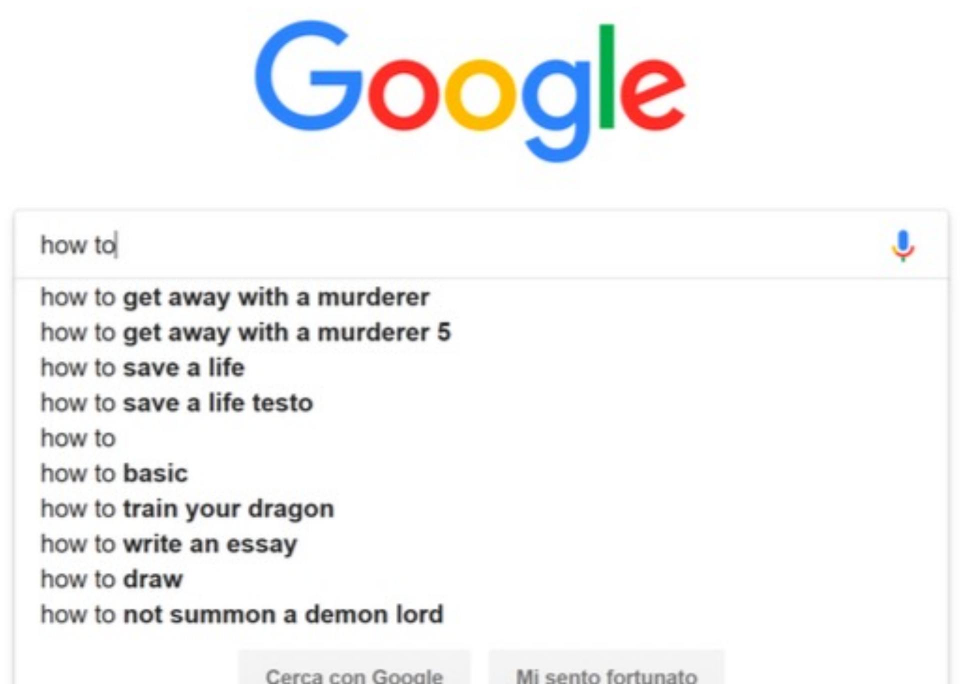
Language Modeling

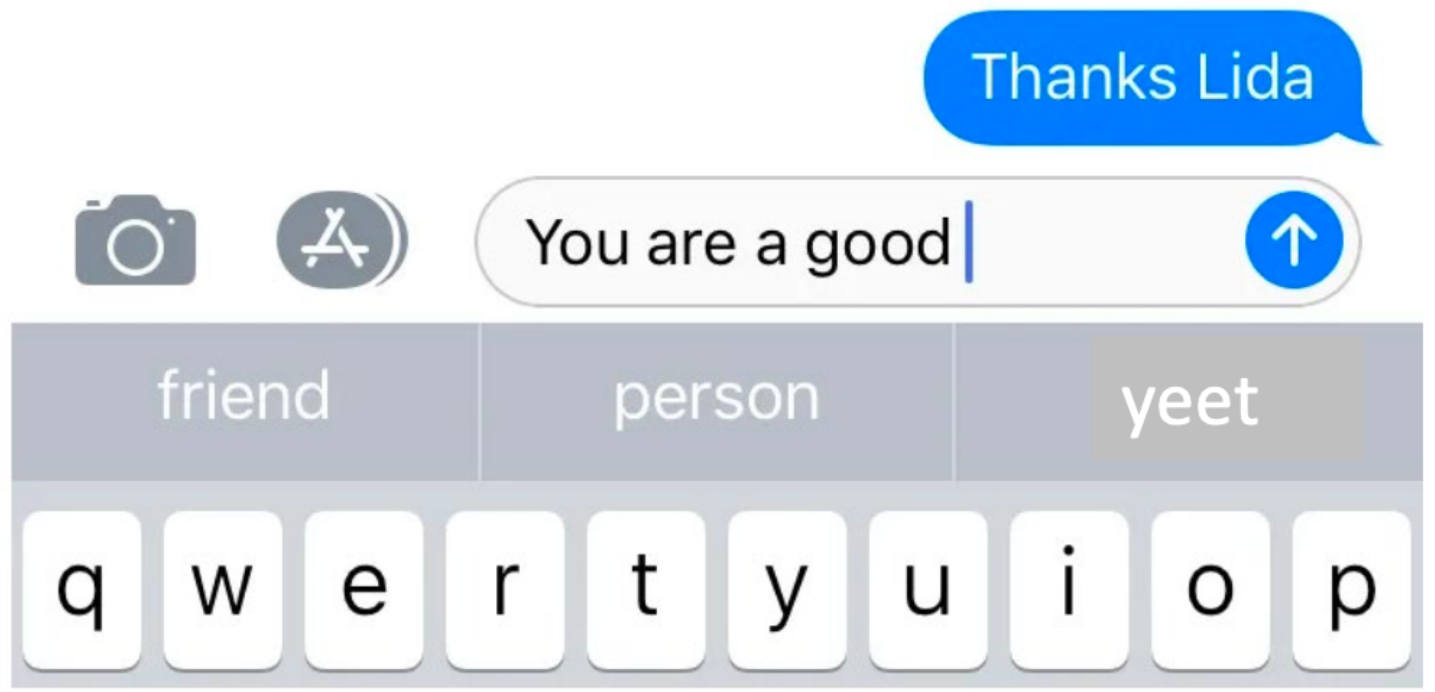
- A collection of statistics learned over a particular language

- Often used to
  - Measure how “important” (or descriptive) a word is in a given document collection
    - e.g. find the set of words that best describe multiple clusters (see Group Assignment 1)
  - Predict how likely a sequence of words is to occur in a given context
    - e.g. find the word(s) that is more likely to occur next

- **A good language model will give this sentence a high probability because this is a completely valid sentence, syntactically and semantically**

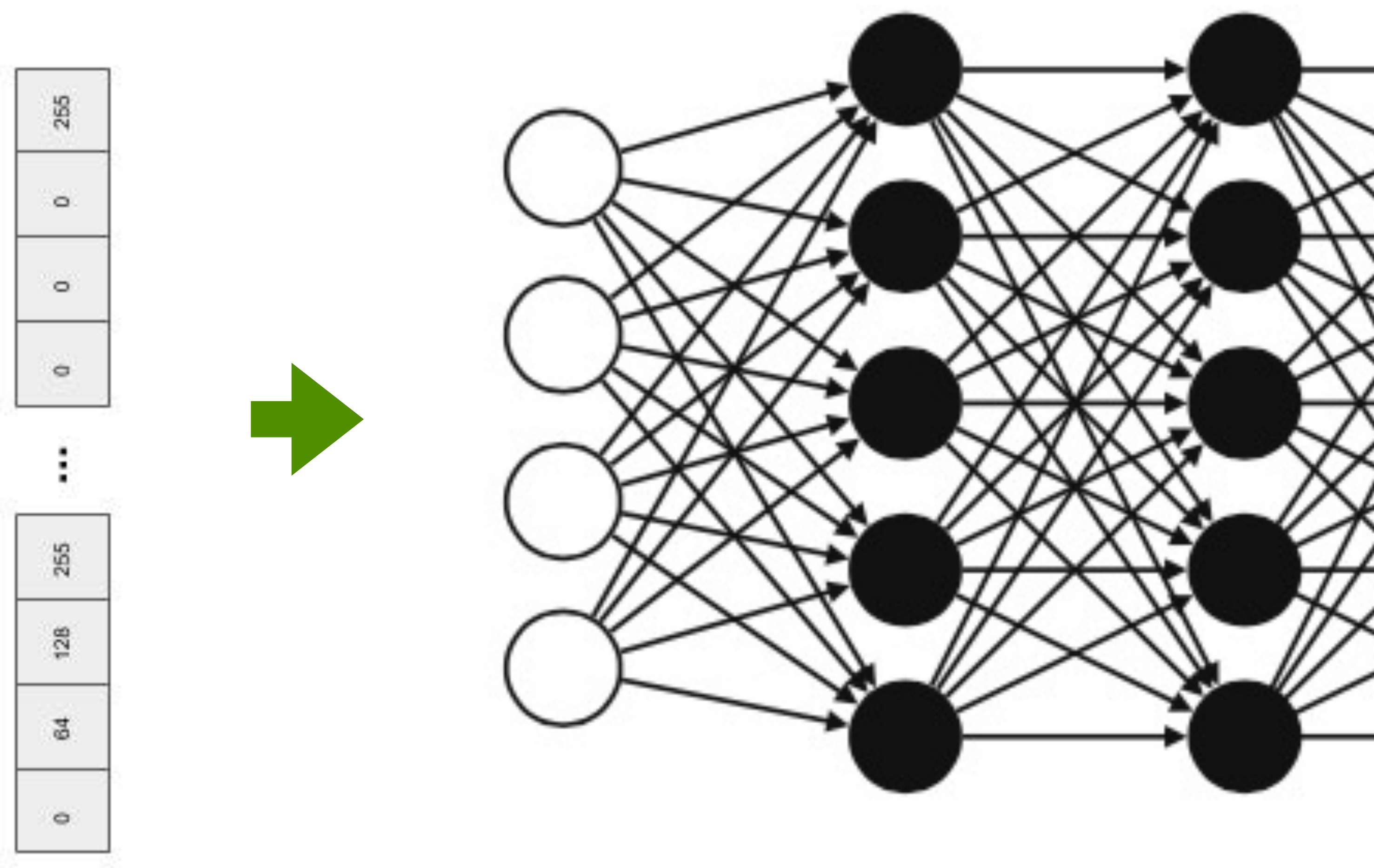
- These probabilities are almost always empirically derived from a text corpora


how to
how to **get away with a murderer**
how to **get away with a murderer 5**
how to **save a life**
how to **save a life testo**
how to
how to **basic**
how to **train your dragon**
how to **write an essay**
how to **draw**
how to **not summon a demon lord**


Thanks Lida
You are a good

The issue with representing words

- Words are discrete symbols
- Machine-learning algorithms cannot process symbolic information as it is
- We need to transform the text into **numbers**
- But we also need a way to express **relationships** between words!



A simple approach

- Assign an incremental number to each word
  - *cat = 1*
  - *dog = 2*
  - *pizza = 3*

- **Problem: There is no notion of similarity!**
  - Is a *cat* as semantically close (similar) to a *dog* as a *dog* is to a *pizza*

- Also, no arithmetic operations
    - Does it make sense to calculate *Dog - Cat* to establish similarity?

Word Embeddings

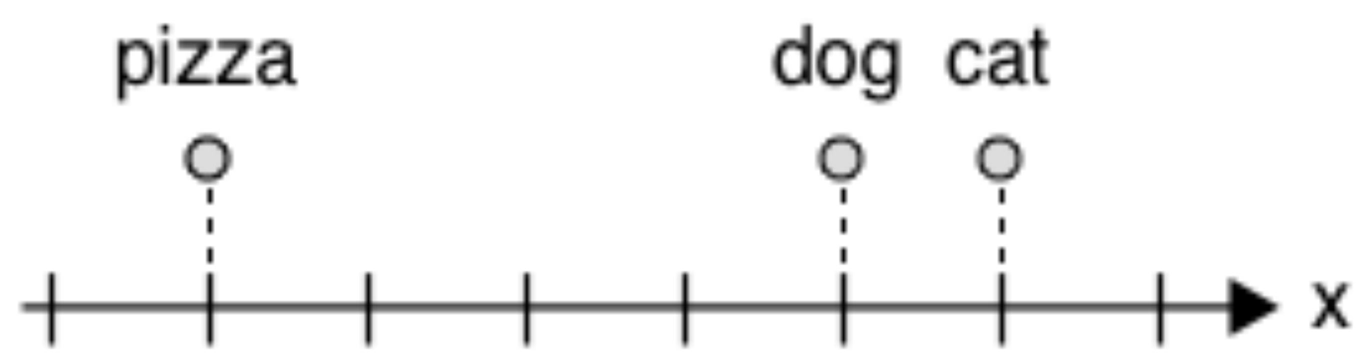
- Embed (represent) words in a numerical *n*-dimensional space

- **Approach 1**: assign numbers to words, and put semantically related words close to each other
  - We can now express that "*dog* is more related to *cat* than to *pizza*"
  - But is *pizza* more related to *dog* than to *cat*?

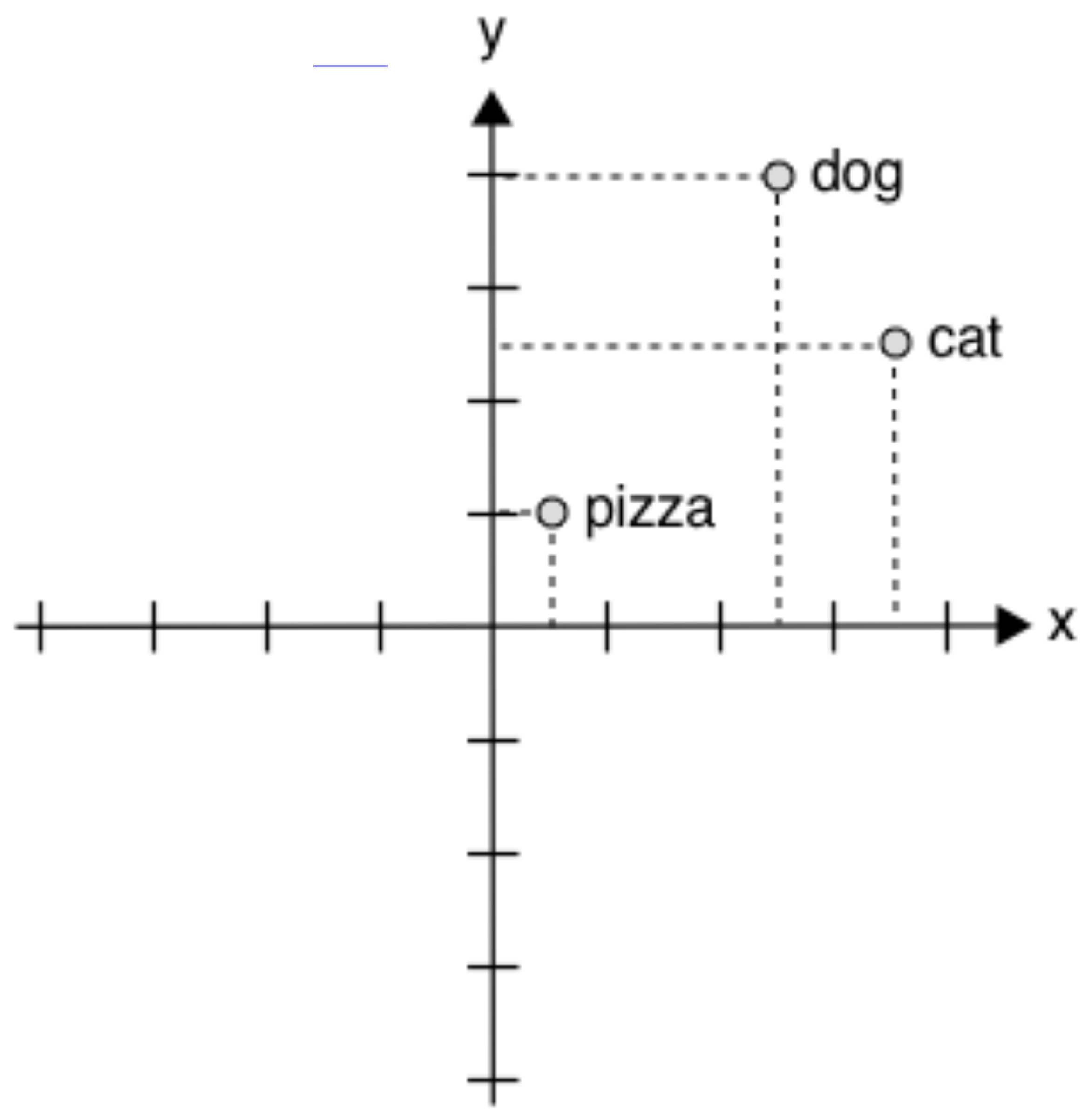
- **Approach 2**: assign multiple numbers (a vector) to words
  - e.g. a 2-dimensional space
    - *cat = [4,2], dog = [3,3], pizza = [1,1]* ← word representation

  - We can calculate distance (and similarity)
    - e.g. Euclidean, or Cosine (angles)
  - But what is the meaning of an axis?

1-Dimension



2-Dimensions



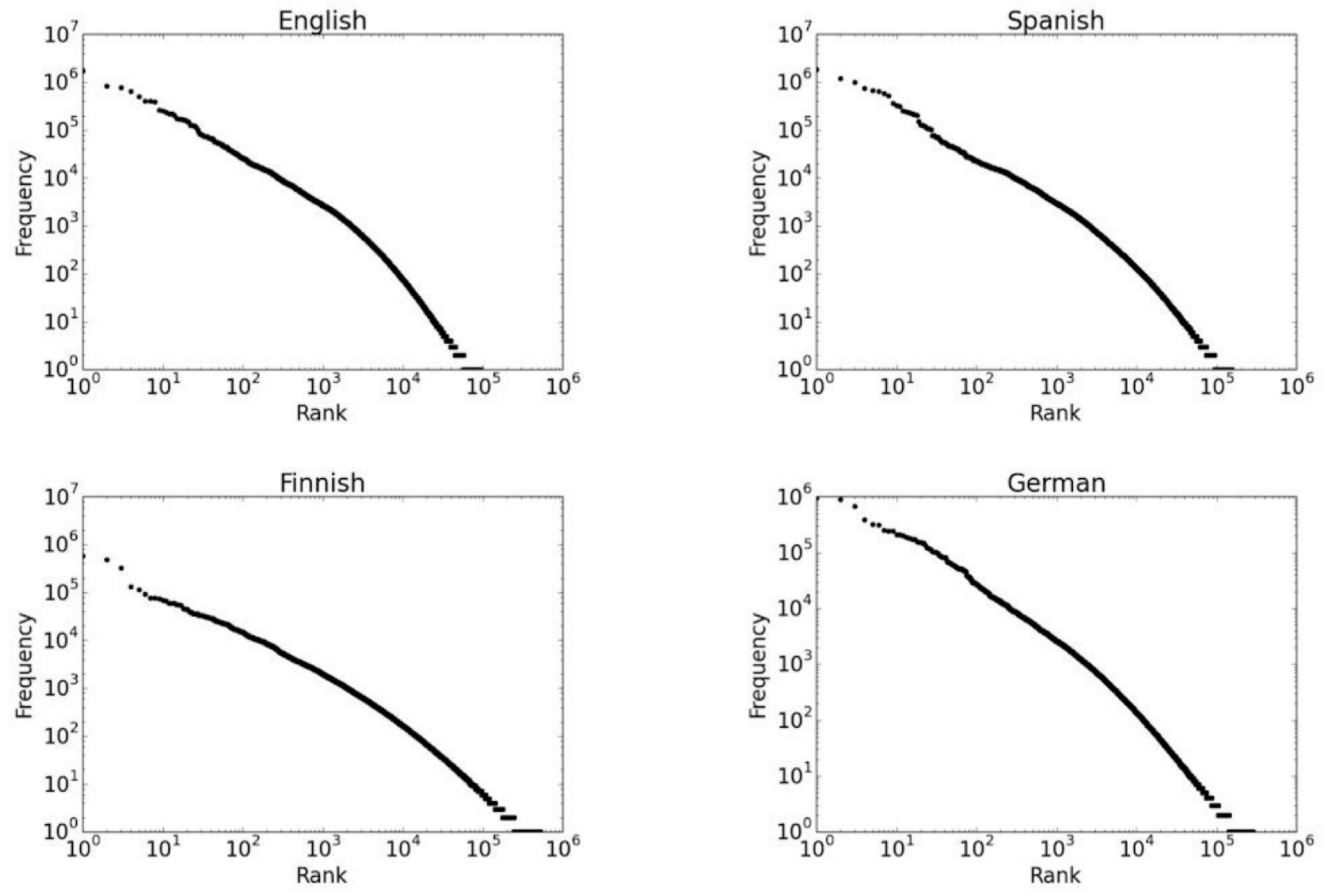
One-Hot Encoding

- Each word in the vocabulary is represented by a one-bit position in a HUGE (**sparse**) vector
  - Vector dimension = size of the dictionary
  - There are an estimated 13 million tokens for the English language

- For example
  - *cat* = `[0,0,0,0,0,0,0,0,0,0,`**`1`**`,0,0,0,…,0]`
  - *dog* = `[0,0,0,0,0,0,0,`**`1`**`,0,0,0,0,0,0,…,0]`
  - *pizza* = `[`**`1`**`,0,0,0,0,0,0,0,0,0,0,0,0,0,…,0]`

- Problems:
  - The size of the vector can be **huge**
    - Remember Zip's law? Easy to reach 10^6 words
    - But we can use stemming, lemmatisation, etc
  - Still, no notion of similarity
    - Each word is an **independent**, discrete entity

Words ordered by their frequency



Independent and identically distributed words assumption

- The simplest (inaccurate) language model assumes that each word in a text **appears independently** on the others
  - The text is modelled as generated by a sequence of independent events
- The probability of a *word* can be estimated as the number of times a word appears in a text corpus
- But high probability does not mean *important* (or descriptive)

Measuring the importance of words

Boolean: $tf_{t,d} = 1$ if t occurs in d, 0 otherwise

Raw Counts: $tf_{t,d} = c_{t,d}$

- $c_{t,d}$ is the number of times t occurs in d

Log-Scaled Counts: $tf_{t,d} = \begin{cases} 1 + \log c_{t,d} & \text{if } c_{t,d} > 0 \\ 0 & \text{otherwise} \end{cases}$

- Reduces relative impact of frequent terms

Normalized Counts: $tf_{t,d} = c_{t,d} / |d|$

- Normalize raw counts by length of document $|d|$

- Term frequency TF
  - Measuring the importance of a word *t* to a document *d*
  - The more frequent, the more important to describe the document

- Inverse document frequency IDF
  - Measuring the importance of a word *t* to a document collection
  - Rare terms are more important than common terms
  - If all (training) documents contain the word *design*, but only a few selected documents contain the word "*machine*", then *machine* is more discriminative in the document collection

$$idf_{t,X} = \log\left(\frac{|X|}{|X_t| + 1}\right)$$

$$tfidf_{t,d,X} = tf_{t,d} \times idf_{t,X}$$

- TF-IDF
  - "Scaling" of a word's importance (in a document) based on both its frequency and collections' importance

N-gram Language models

- A more accurate model takes into account the conditional probabilities among **adjacent** words (e.g. bi-grams)
  - We try to calculate the probability of a word *w* given a word w^{-1}
    - e.g. *computer network* vs. *computer pear*

- The model is more accurate but it is more difficult to be estimated with accuracy

- The N-grams model dependencies deriving from
  - Grammatical rules
    - e.g. an adjective is likely to be followed by a noun
  - Semantic restrictions
    - e.g. *Eat a pear* vs. *Eat a crowbar*
  - Cultural restrictions
    - e.g. *Eat a cat*

$$p(w|\text{eat})$$

eat on	0.16	eat Thai	0.03
eat some	0.06	eat breakfast	0.03
eat lunch	0.06	eat in	0.02
eat dinner	0.05	eat Chinese	0.02
eat at	0.04	eat Mexican	0.02
eat a	0.04	eat tomorrow	0.01
eat indian	0.04	eat dessert	0.007
eat today	0.03	eat British	0.001

- **The probabilities depend on the considered contexts**

Limits of N-grams based LMs

- The model accuracy increases with N
  - The syntactic/semantic contexts are better modelled
- The drawback is the difficulty in the model parameter estimation (the conditional probabilities)
  - If the dictionary contains *D* terms (word forms with inflexions) there are D^N N-grams
    - A corpus *C* words “long” contains *C* N-grams (each word generates exactly a sample for one N-gram)
    - For a significant estimate of the parameters, the corpus size should increase exponentially in the order N of N-grams
    - f.i. given *D=30000* there are 900 million bigrams and a corpus with *C=1.000.000* words would not be adequate to compute an accurate estimate for the language (especially for the rarest bigrams)
    - Hence, the resulting model can be heavily dependent on the corpus exploited in the estimation of the parameters
  - They **do not generalise to unseen words sequences**
- What about using **machine learning**?

Representing words by their contexts

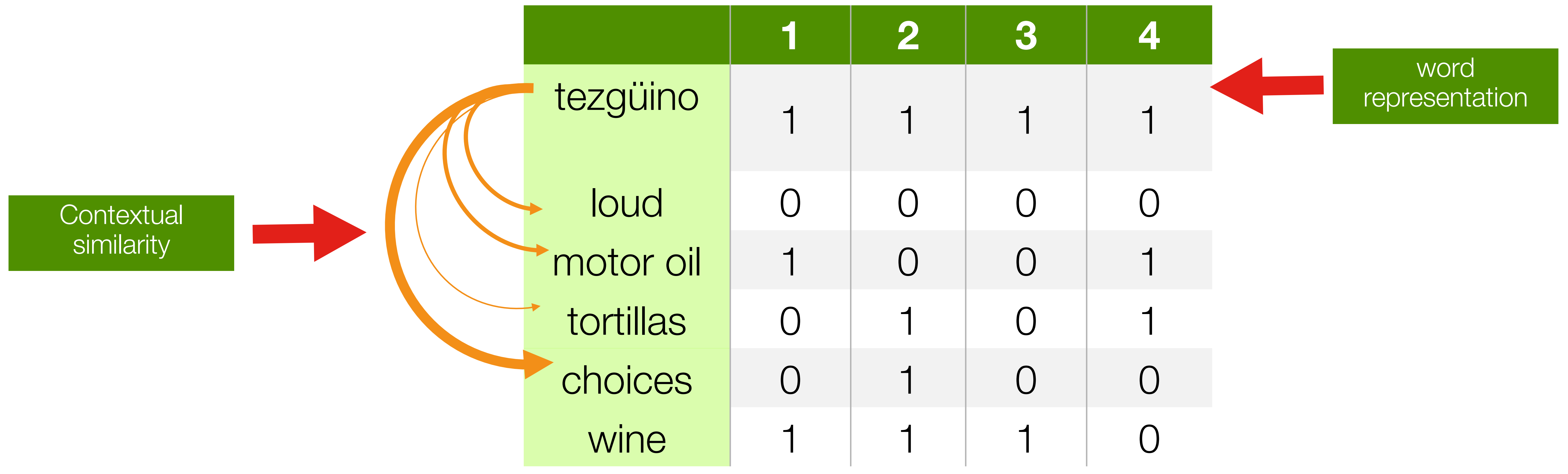
- When a word *w* appears in a text, its context is the set of words that appear nearby (within a fixed-size window)

- **Distributional semantics**: A word's meaning is given by the words that frequently appear close-by

- For example: look at the following contexts:
  - (1) A bottle of ___ is on the table
  - (2) Everybody likes ___
  - (3) Don't have ___ before you drive
  - (4) We make ___ out of corn

- What other words fit into these contexts?

tezgüino



	1	2	3	4
tezgüino	1	1	1	1
loud	0	0	0	0
motor oil	1	0	0	1
tortillas	0	1	0	1
choices	0	1	0	0
wine	1	1	1	0

(Example from Lin, 1998)

Representing words by their contexts

- When a word *w* appears in a text, its context is the set of words that appear nearby (within a fixed-size window)

- **Distributional semantics**: A word’s meaning is given by the words that frequently appear close-by

- For example: look at the following contexts:
  - (1) A bottle of ___ is on the table
  - (2) Everybody likes ___
  - (3) Don’t have ___ before you drive
  - (4) We make ___ out of corn

- What other words fit into these contexts?

“You shall know a word by the company it keeps”

The distributional hypothesis, John Firth (1957)

The contexts in which a word appears tell us a lot about what it means.

Words that appear in similar contexts have similar meanings

Distributional Word Embeddings

- Define dimensions that allow expressing a *context*
  - The vector for any particular word captures how strongly it is associated with each context

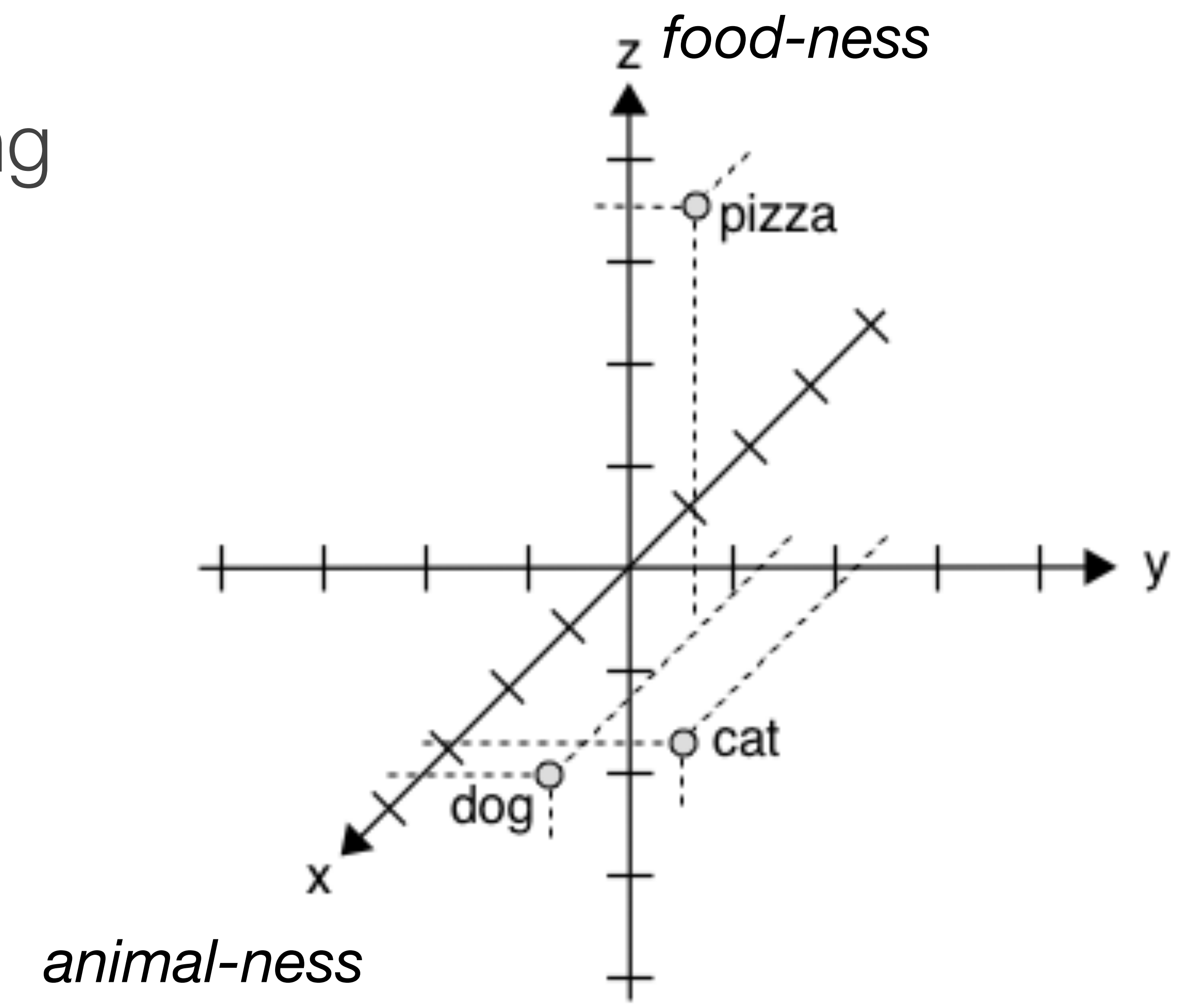
- For instance, on a 3-dimensional space, the axis could have the semantic meaning
    - *x-axis* represents some concept of "*animal-ness*"
    - z-*axis* corresponds to "*food-ness*"

- Of course, defining these axes is very difficult
  - How many?
    - Hopefully, a lot less than the size of the dictionary (**dense vectors**)
    - But at least ~100-dimensional, to be effective

- Also, how do we assign the values associated with the vectors?
  - Tens of millions of numbers to tweak

- **How about using machine learning models? —> later**

3-Dimensions



cat	=	[0.7,0.5,0.1]
dog	=	[0.8,0.3,0.1]
pizza	=	[0.1,0.2,0.8]

Word Embeddings with Machine Learning

How to calculate Word Embeddings?

- By calculating **co-occurrence** counts on the whole dataset
  - Full document: Latent Semantic Analysis
  - Window: SVD Based Methods

- **Iteration Based Methods**: learn one iteration (e.g. sentence) at a time
  - Word2Vec

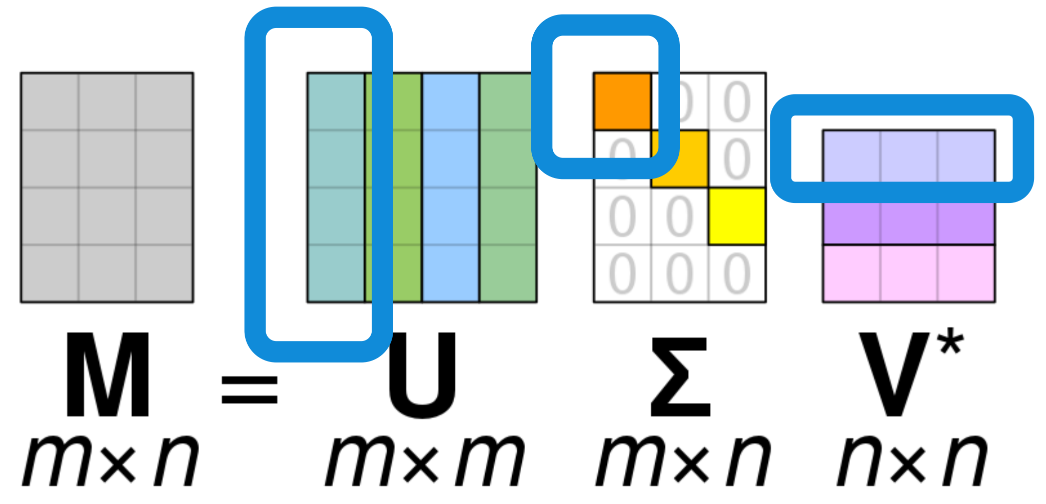
Word-Document Matrix

- Words that are related will often appear in the same documents
  - E.g. *banks*, *bonds*, *stocks*, *money*, etc. are probably likely to appear together
  - But *banks*, *octopus*, *banana*, and *hockey* are probably less likely

- Example corpus:
  - **D1**: *I like deep learning.*
  - **D2**: *I like NLP.*
  - **D3**: *I enjoy flying.*
- The result is a very large matrix
  - Size is a function of the number of words and number of documents

- Then reduce dimensionality using Singular Value Decomposition (SVD)
  - Factorization of a matrix in 3x ones

	D1	D2	D3
I	1	1	1
Like	1	1	0
enjoy	0	1	0
deep	1	0	0
learning	1	0	0
NLP	0	1	0
flying	0	0	1
.	1	1	1


$\mathbf{M}_{m\times n} = \mathbf{U}_{m\times m}\,\mathbf{\Sigma}_{m\times n}\,\mathbf{V}^*_{n\times n}$

Window based co-occurrence matrix

- Window length 1 (more common: 5–10)
- Symmetric (irrelevant whether left or right context)

- Example corpus:
  - **D1**: *I like deep learning.*
  - **D2**: *I like NLP.*
  - **D3**: *I enjoy flying.*

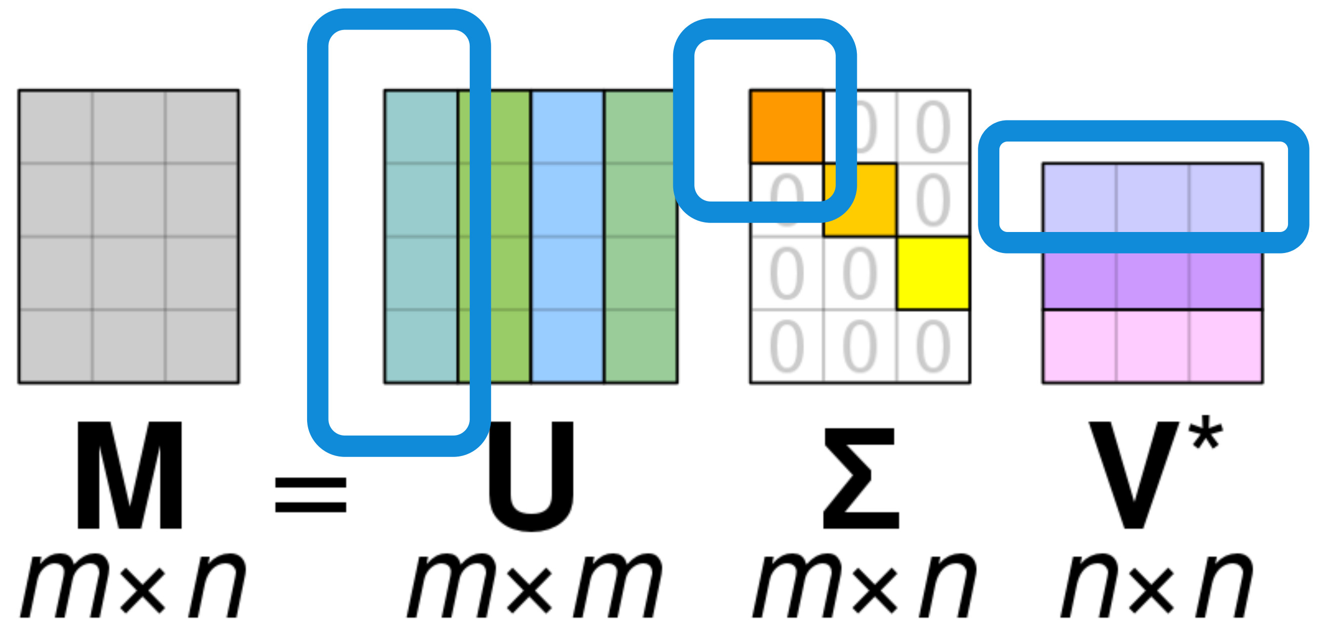
	I	like	enjoy	deep	learning	NLP	flying	.
I	0	2	1	0	0	0	0	0
Like	2	0	0	1	0	1	0	0
enjoy	1	0	0	0	0	0	1	0
deep	0	1	0	0	1	0	0	0
learning	0	0	0	1	0	0	0	1
NLP	0	1	0	0	0	0	0	1
flying	0	0	1	0	0	0	0	1
.	0	0	0	0	1	1	1	0

Co-Occurrence Vectors

- Simple count co-occurrence vectors
  - Vectors increase in size with vocabulary
  - Very high-dimensional: require a lot of storage (though sparse)
  - Subsequent classification models have sparsity issues -> Models are less robust

- Low-dimensional vectors
  - Idea: store "most" of the important information in a fixed, small number of dimensions: a dense vector
  - Usually 25–1000 dimensions

- Dimensionality reduced through Singular Value Decomposition (SVD)



Problems with co-occurrence approaches

- The calculated word vectors are more than sufficient to encode semantic and syntactic (part of speech) information
- But there are many other problems:
  - The dimensions of the matrix change very often (new words are added very frequently and corpus changes in size)
  - The matrix is extremely sparse since most words do not co-occur.
  - The matrix is very high dimensional in general
  - Very expensive to train (i.e. to perform SVD)

- Some clever intervention is needed to adjust the co-occurrence matrix to account for the imbalance in word frequency
  - Ignore stopwords
  - Apply a ramp window – i.e. weight the co-occurrence count based on the distance between the words in the document.
  - Use Pearson correlation and set negative counts to 0 instead of using just raw count

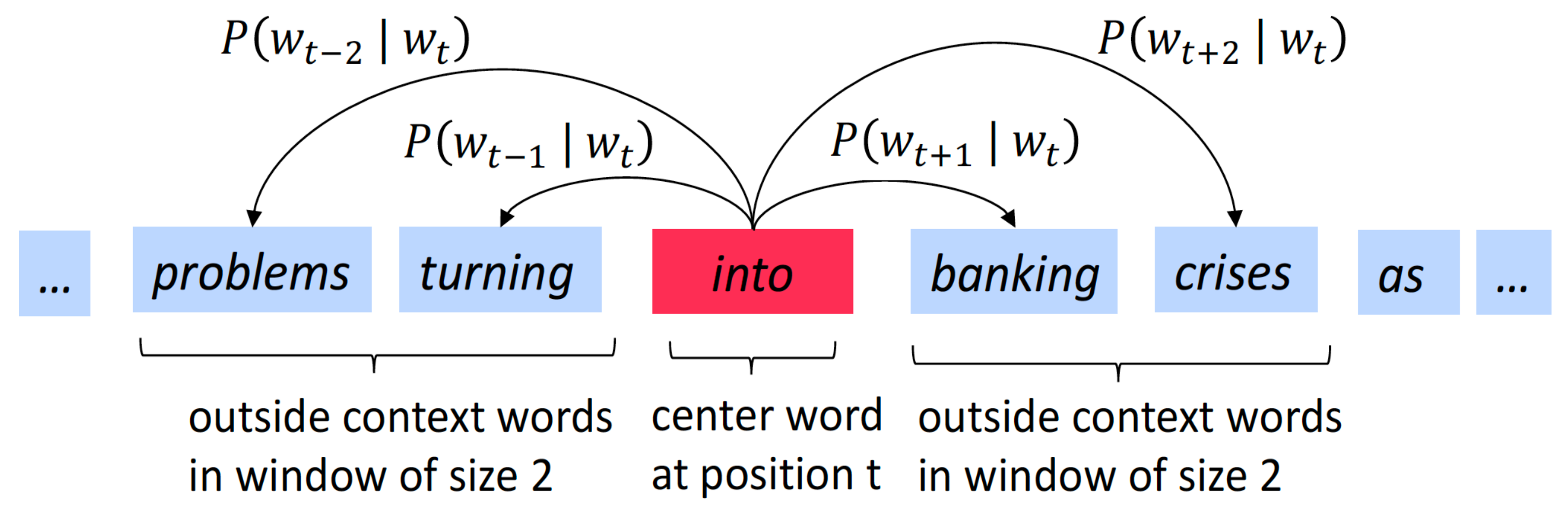
- Iteration-based methods solve many of these issues

Iteration Based Methods - Word2Vec

- Idea: Design a model whose parameters are the word vectors
  - Train a simple neural network with a single hidden layer, using a certain objective
  - At every iteration, evaluate the errors, penalize the model parameters that caused the error

- How?
  - Consider a large corpus of text
  - Define a vocabulary of words and associate each word to a row of the embedding matrix initialised at random
  - Go through each position in the text, which has a **centre** word and a **context** around it (**fixed window**)
  - Adjust the word vectors to **minimise** a prediction error

- Predicting what?
  - Estimate the probability of context given the centre word (**SKIPGRAM**)
  - Estimate the probability of the centre word given its context (**CBOW**)

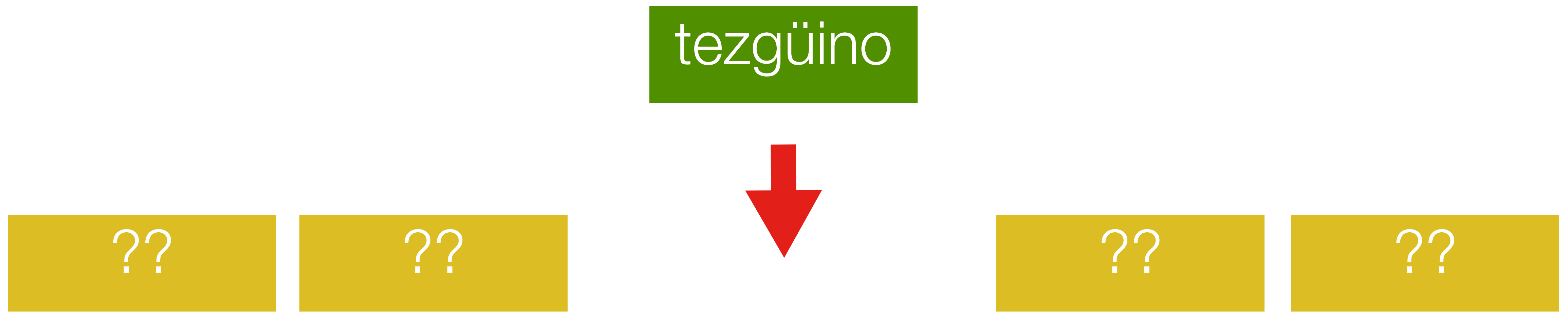


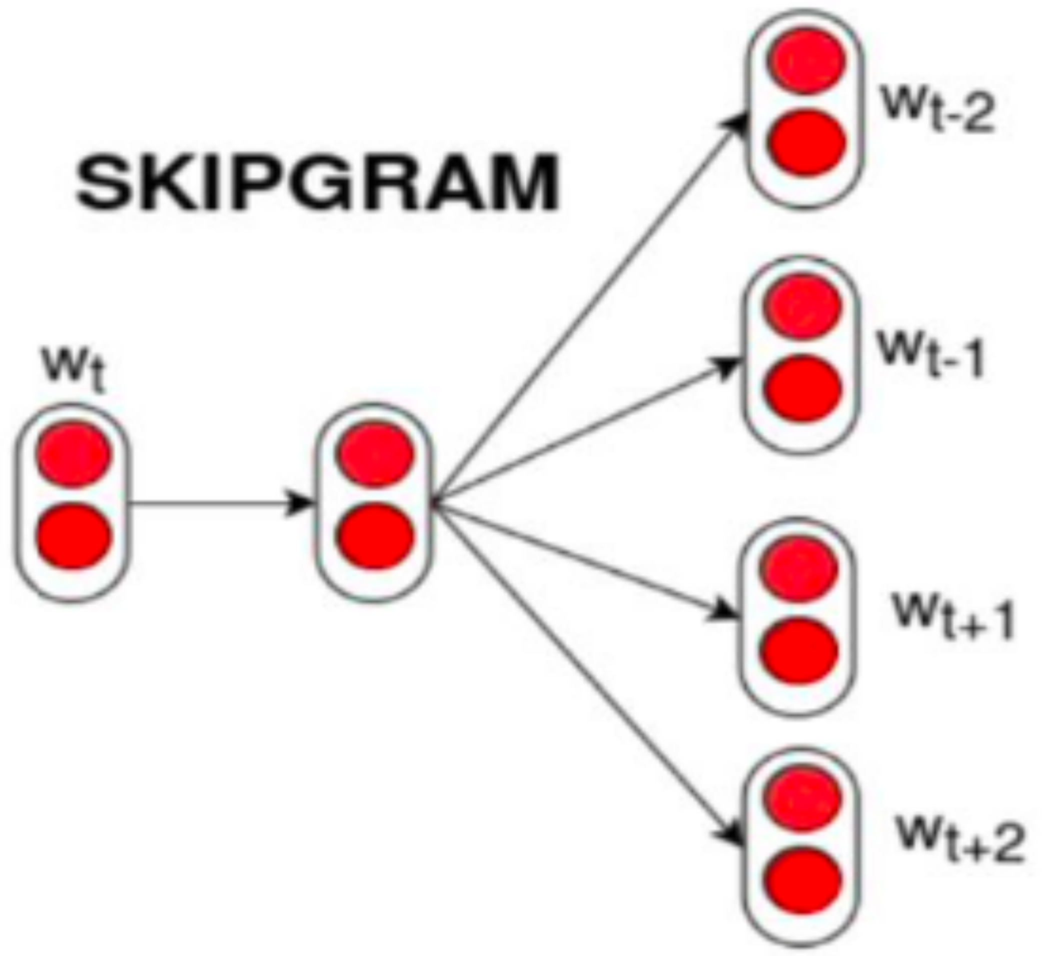
HINT: Calculating the probability P of each word occurring in the vicinity of the center word and within a specified window

SKIPGRAM

- Predicts the probability of context words from a centre word

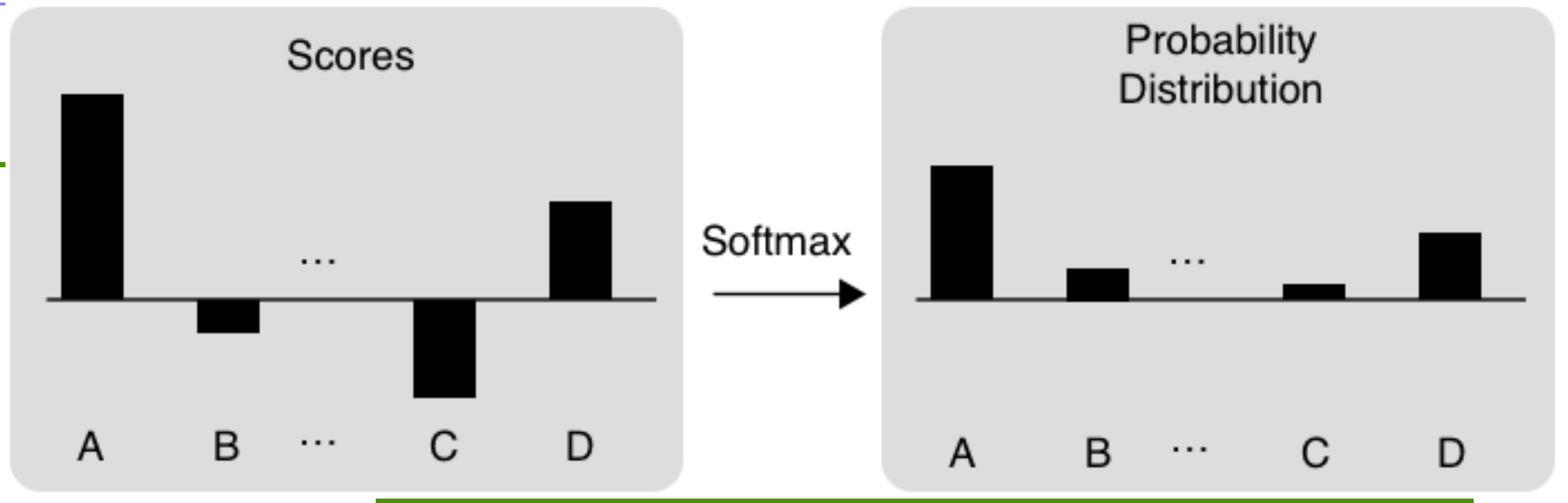
- **Input**: one-hot vector of the centre word (size of the vocabulary)
- **Output**: a single vector; for every word the probability that a word is selected to be in the context window
  - When training this network on word pairs, the input is a one-hot vector representing the input word and the training output is also a one-hot vector representing the output word





- Each word is generated multiple times
  - each time it is conditioned only on a single word

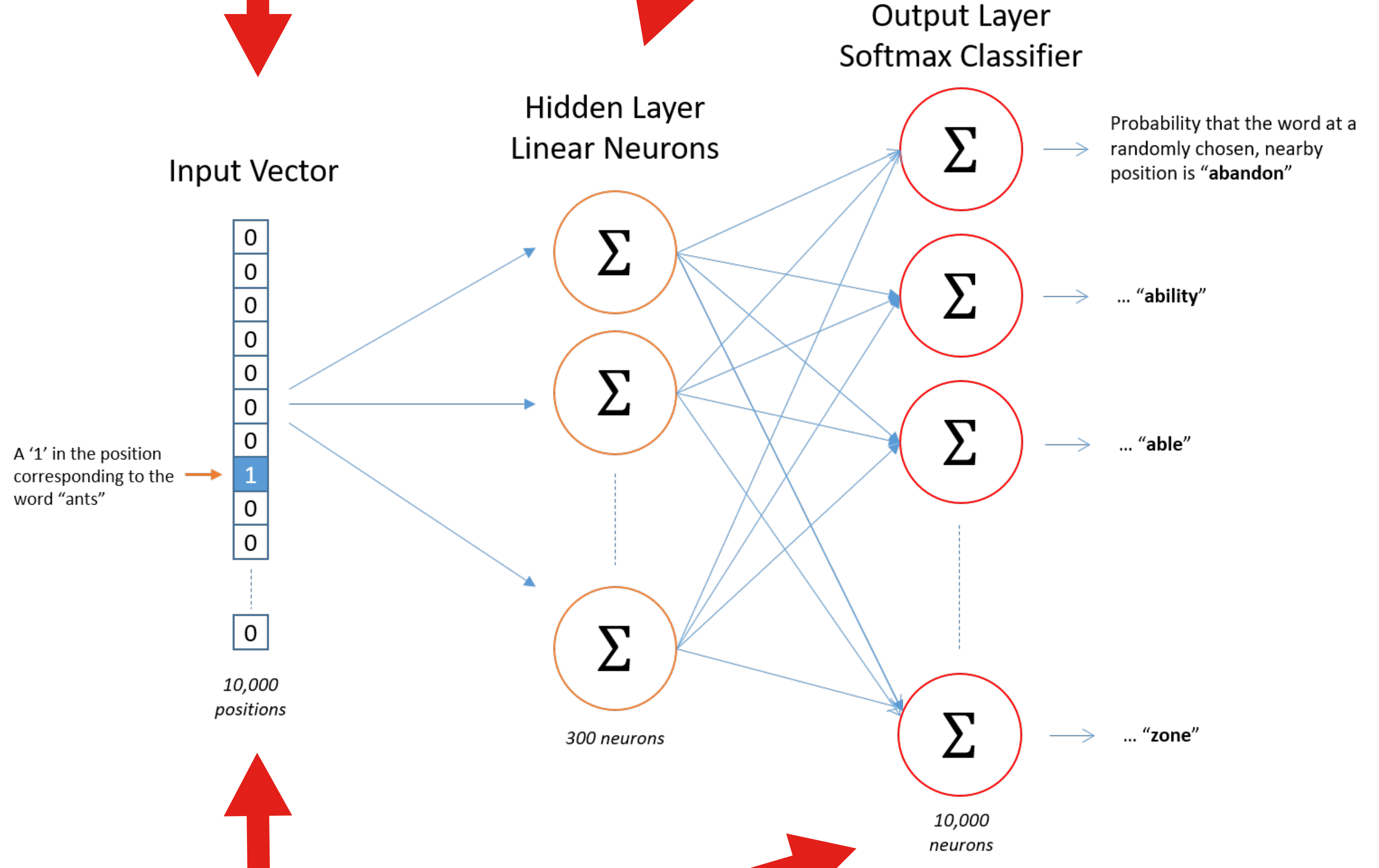
SKIPGRAM Example



This is the word vector that we want to learn

One-hot input vector

Source Text	Training Samples
The quick brown fox jumps over the lazy dog.	(the, quick) (the, brown)
The **quick** brown fox jumps over the lazy dog.	(quick, the) (quick, brown) (quick, fox)
The quick **brown** fox jumps over the lazy dog.	(brown, the) (brown, quick) (brown, fox) (brown, jumps)
The quick brown **fox** jumps over the lazy dog.	(fox, quick) (fox, brown) (fox, jumps) (fox, over)



Input Vector
Hidden Layer Linear Neurons
Output Layer Softmax Classifier
0
1
A '1' in the position corresponding to the word "ants"
10,000 positions
300 neurons
10,000 neurons
Probability that the word at a randomly chosen, nearby position is "abandon"
... "ability"
... "able"
... "zone"

Size of vocabulary

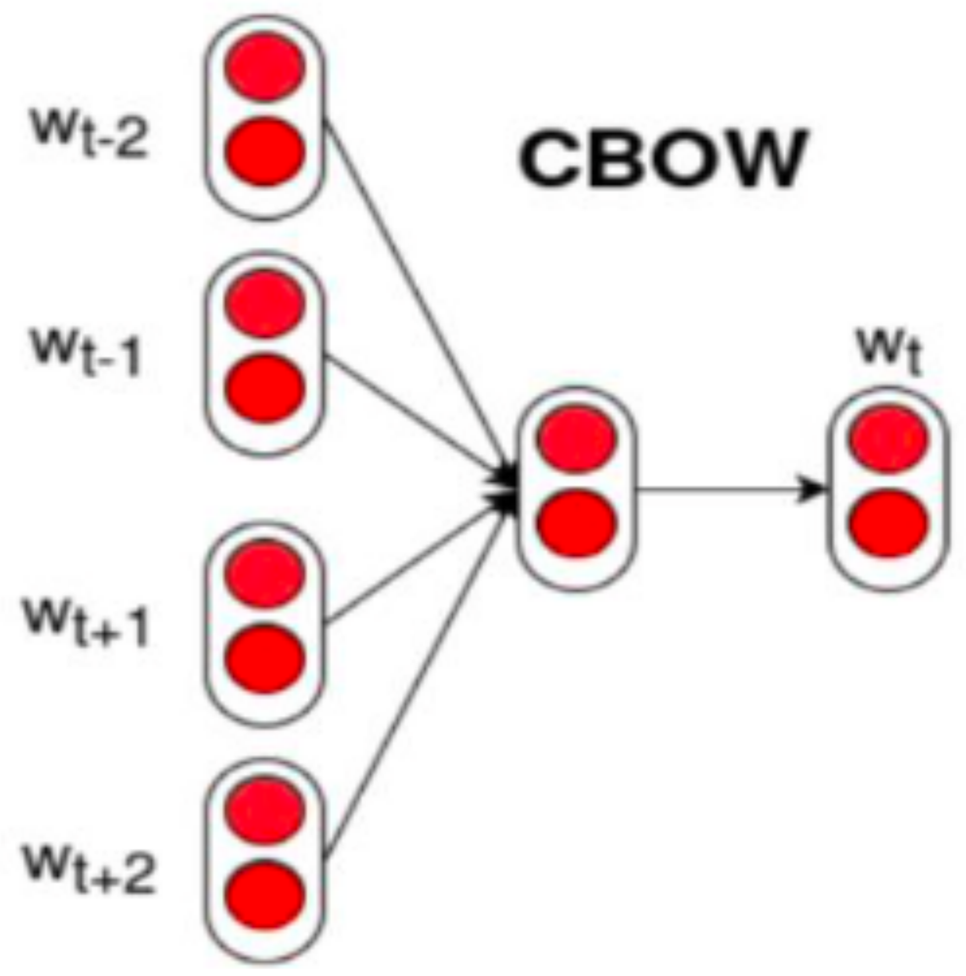
http://mccormickml.com/2016/04/19/word2vec-tutorial-the-skip-gram-model/

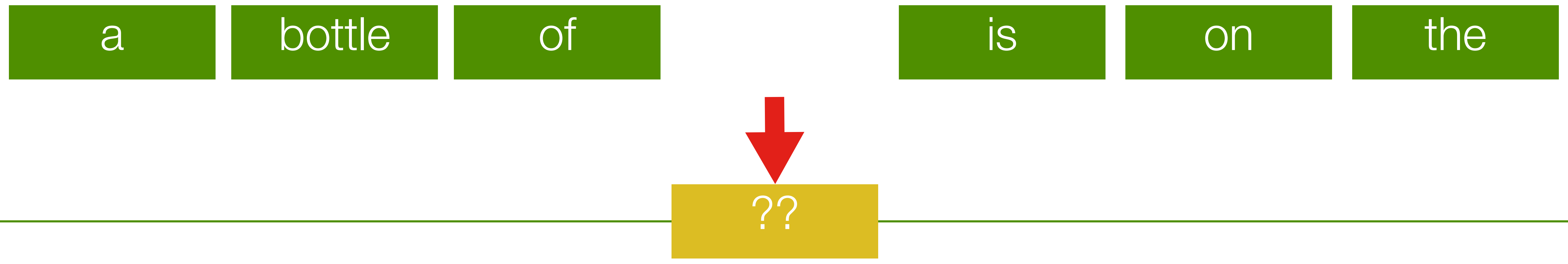
CBOW - Continous Bag of Word

- Predict a centre word from the surrounding context in terms of word vectors
  - Bag-of-words model: because the order of the context words does not matter
  - Continuous: condition on a continuous vector constructed from the word embeddings

- **Input**: multiple one-hot vectors (one per context word)
- **Output**: a single vector, for every word the probability that a word is selected to be the right one for the context
- The dimension of the hidden layer is the same as for SKIPGRAM

- **SKIPGRAM**: works well with a small amount of the training data, represents well even rare words or phrases.
- **CBOW**: several times faster to train than the skip-gram, slightly better accuracy for the frequent words.





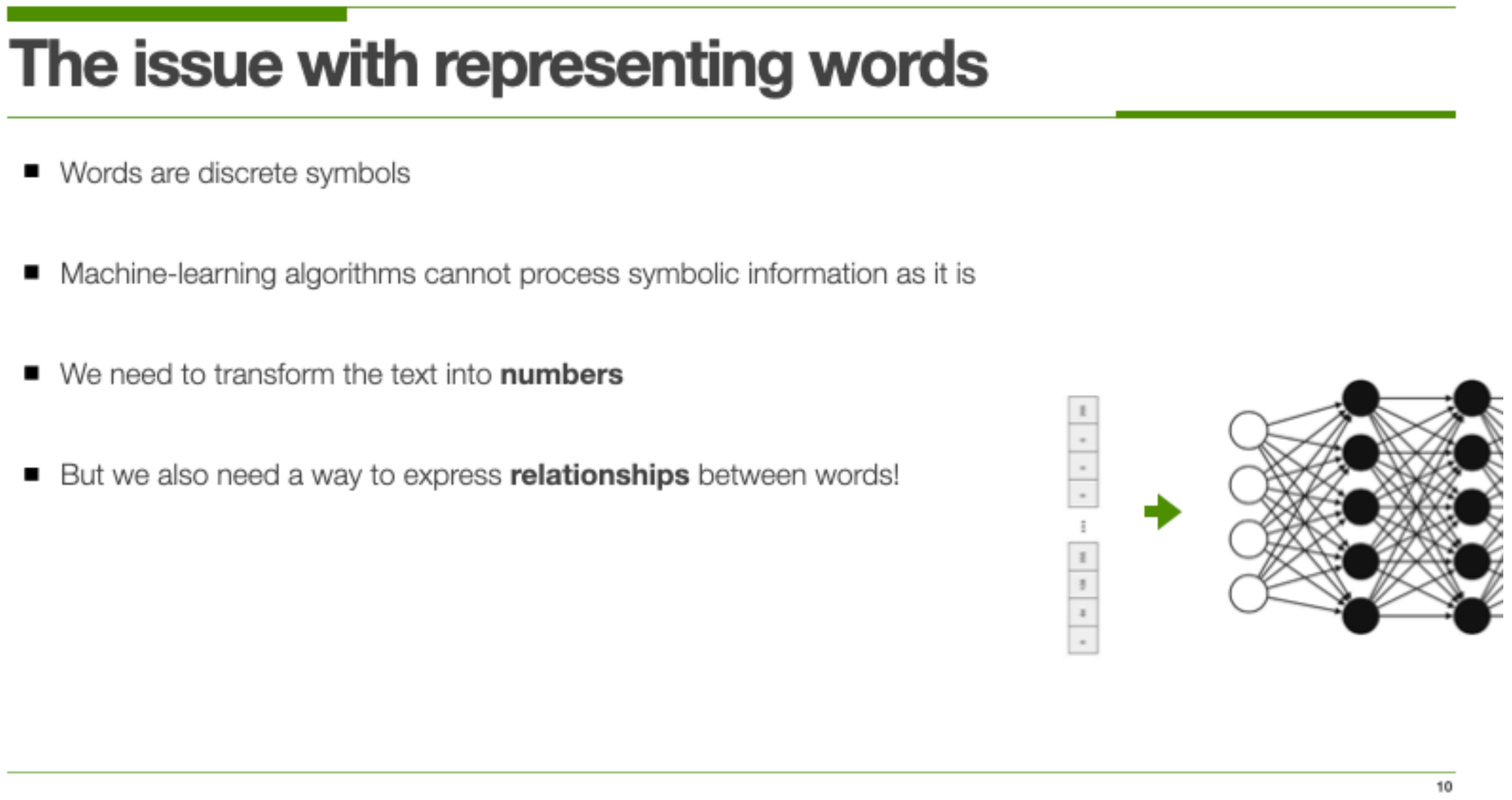
Issues

- Results are in general impressive, but:
- **Multi-sense** words (e.g. bank)
  - Possible solution: multi-sense word embeddings
- **Fixed-size** vocabulary: new words are not learned
  - **Out of Vocabulary** words are represented with the same dense vector
- No information about sub-word structure: morphology is completely ignored
  - Possible solution: character-based word representation
    - e.g. Facebook’s FastText (https://fasttext.cc)

Using Word Embeddings

Why are embeddings important

- They are essential for using neural networks to solve NLP tasks
- They bridge the symbolic (discrete) world of ***words*** with the numerical (continuous) world of neural networks



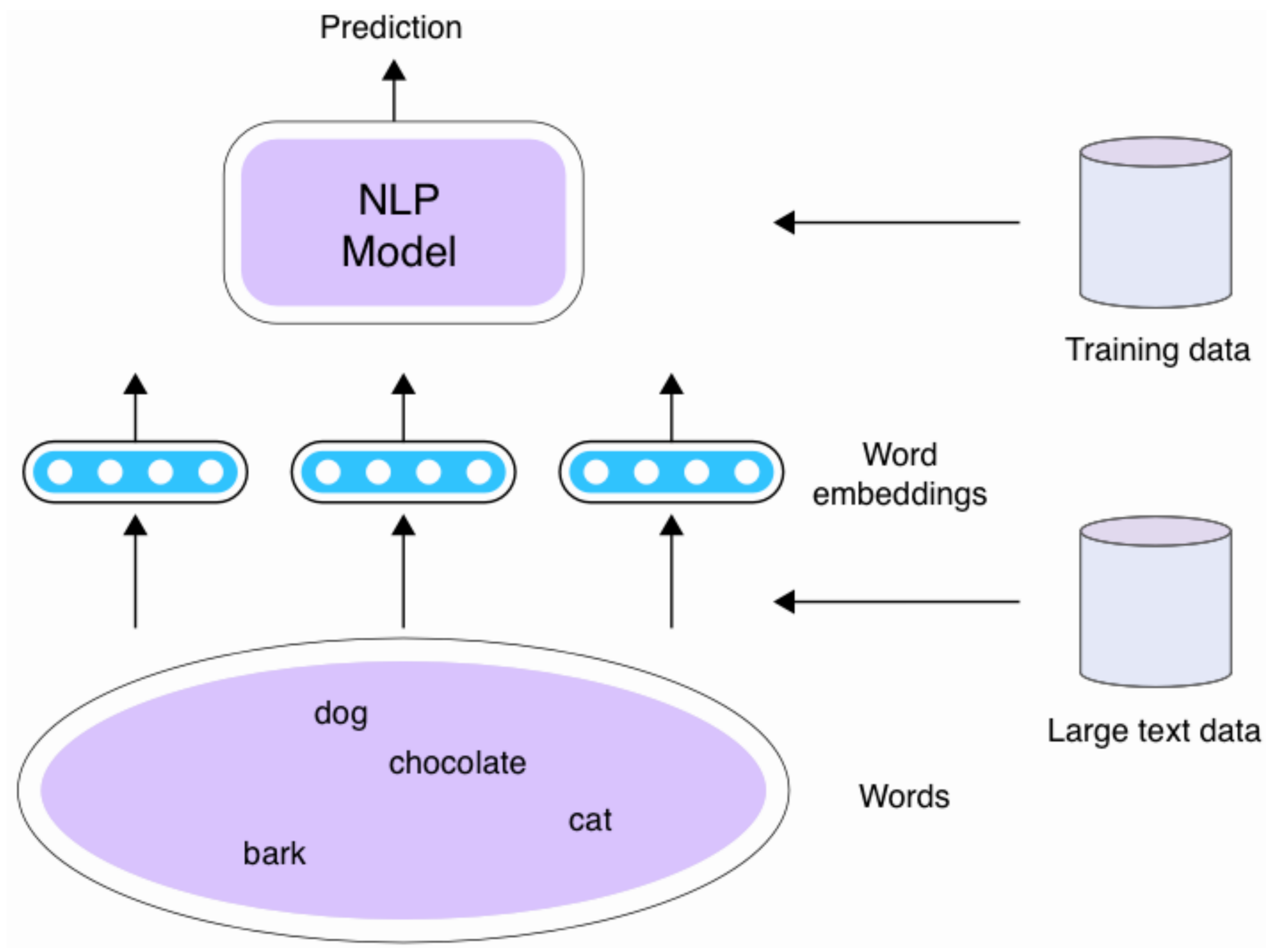
The issue with representing words

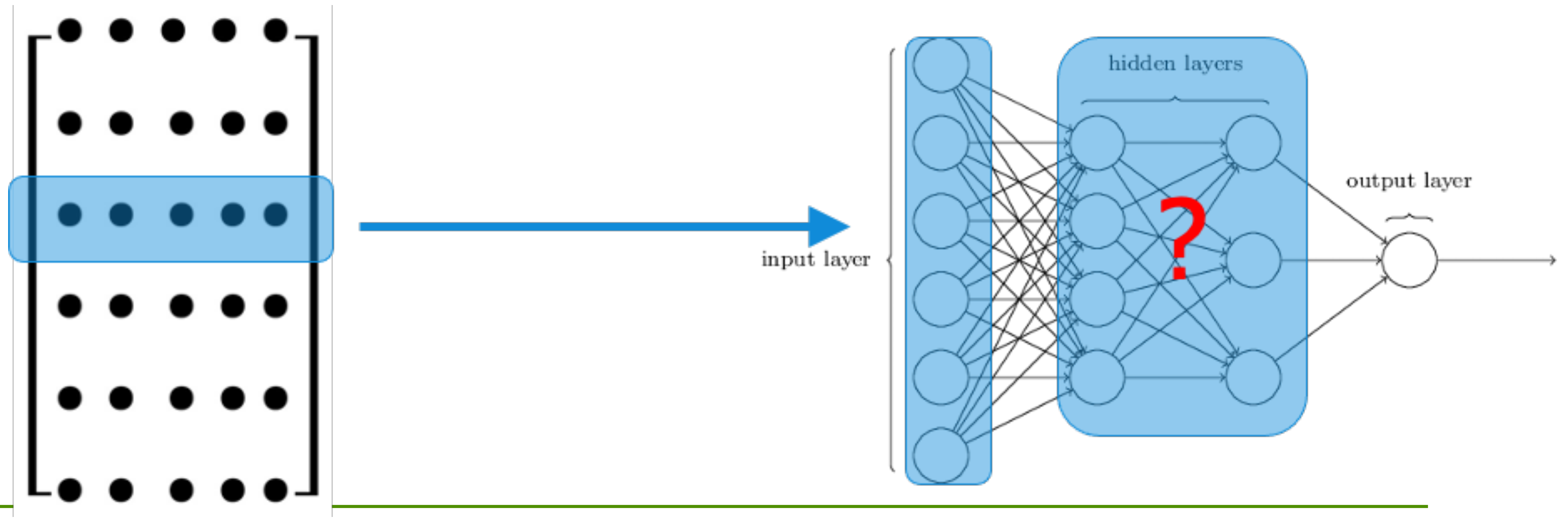
- Words are discrete symbols
- Machine-learning algorithms cannot process symbolic information as it is
- We need to transform the text into **numbers**
- But we also need a way to express **relationships** between words!

10

How can embeddings be used with NLP Models?

- Word embeddings can be trained, but sometimes you just want to reuse them
- Three scenarios
  - **Scenario 1**: Train word embeddings and your model at the same time using the train set for your task
  - **Scenario 2**: initialize your model using the pre-trained word embeddings, and train them (fine-tune) and your model at the same time using the train set for your task
    - A large amount of plain text data (e.g. Wikipedia dumps), which are usually more readily available than the train datasets for your task
    - This is an example of *transfer learning*
  - **Scenario 3**: Same as Scenario 2, except you fix word embeddings while you train your model





Use Word2vec in your work

- Easiest way to use it is via the Gensim library for Python (tends to be slowish, even though it tries to use C optimizations like Cython, NumPy)
  - https://radimrehurek.com/gensim/models/word2vec.html

- Original word2vec C code by Google
  - https://code.google.com/archive/p/word2vec/

- **Use pre-trained word vectors whenever possible**
  - Glove: https://nlp.stanford.edu/projects/glove/
  - fastText: https://fasttext.cc/docs/en/english-vectors.html

Evaluating Word Embeddings

How to evaluate word vectors?

- Related to a general evaluation in NLP: Intrinsic vs. extrinsic

- **Intrinsic**: evaluation on a specific/intermediate subtask [analogy]
  - Fast to compute
  - Helps to understand that system
  - Not clear if really helpful unless correlation to the real task is established

- **Extrinsic**: evaluation on a real task
  - Can take a long time to compute the accuracy
  - Unclear if the subsystem is the problem or it is an interaction with other subsystems

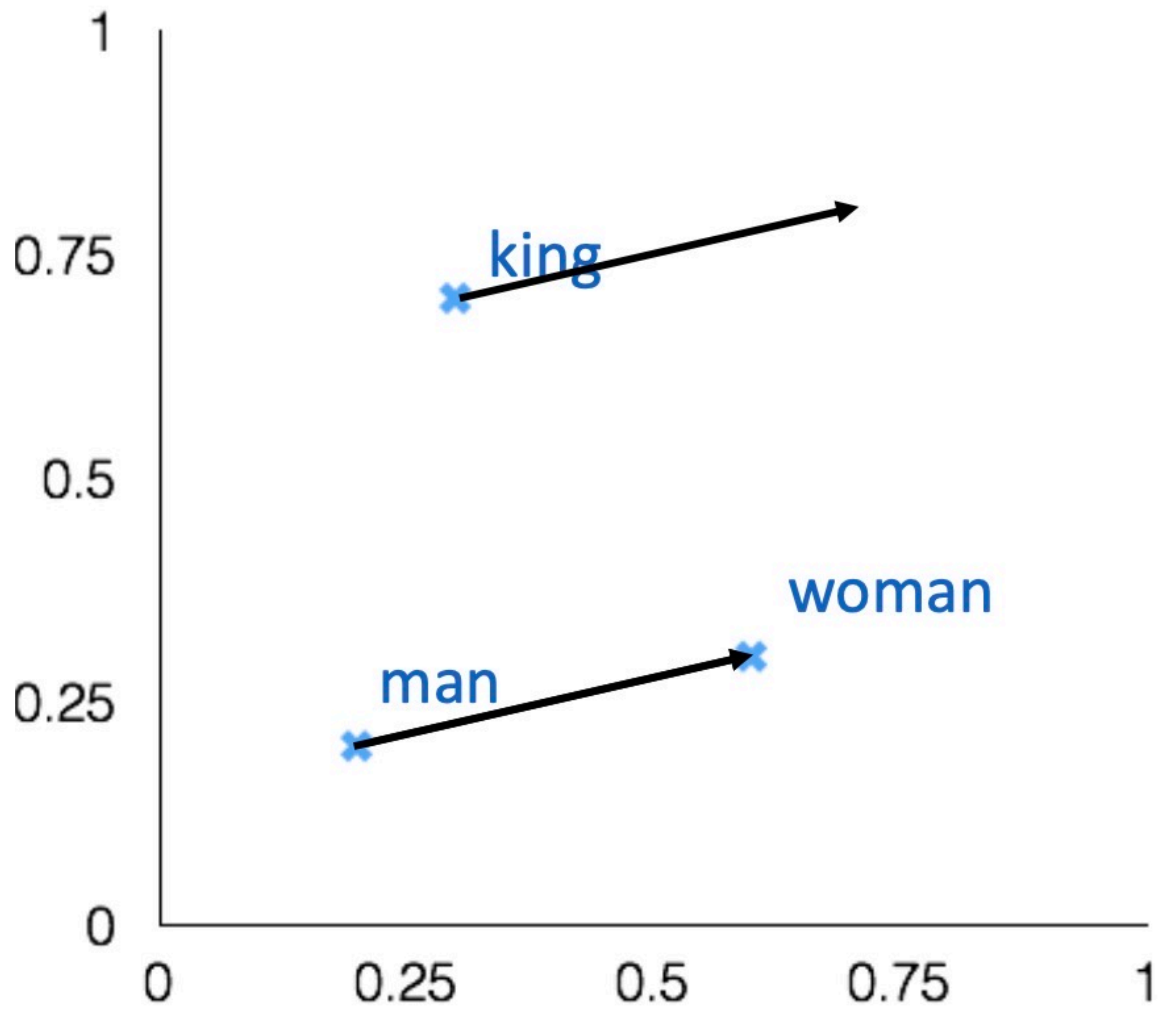
Intrinsic word evaluation

- Word vector **analogies**

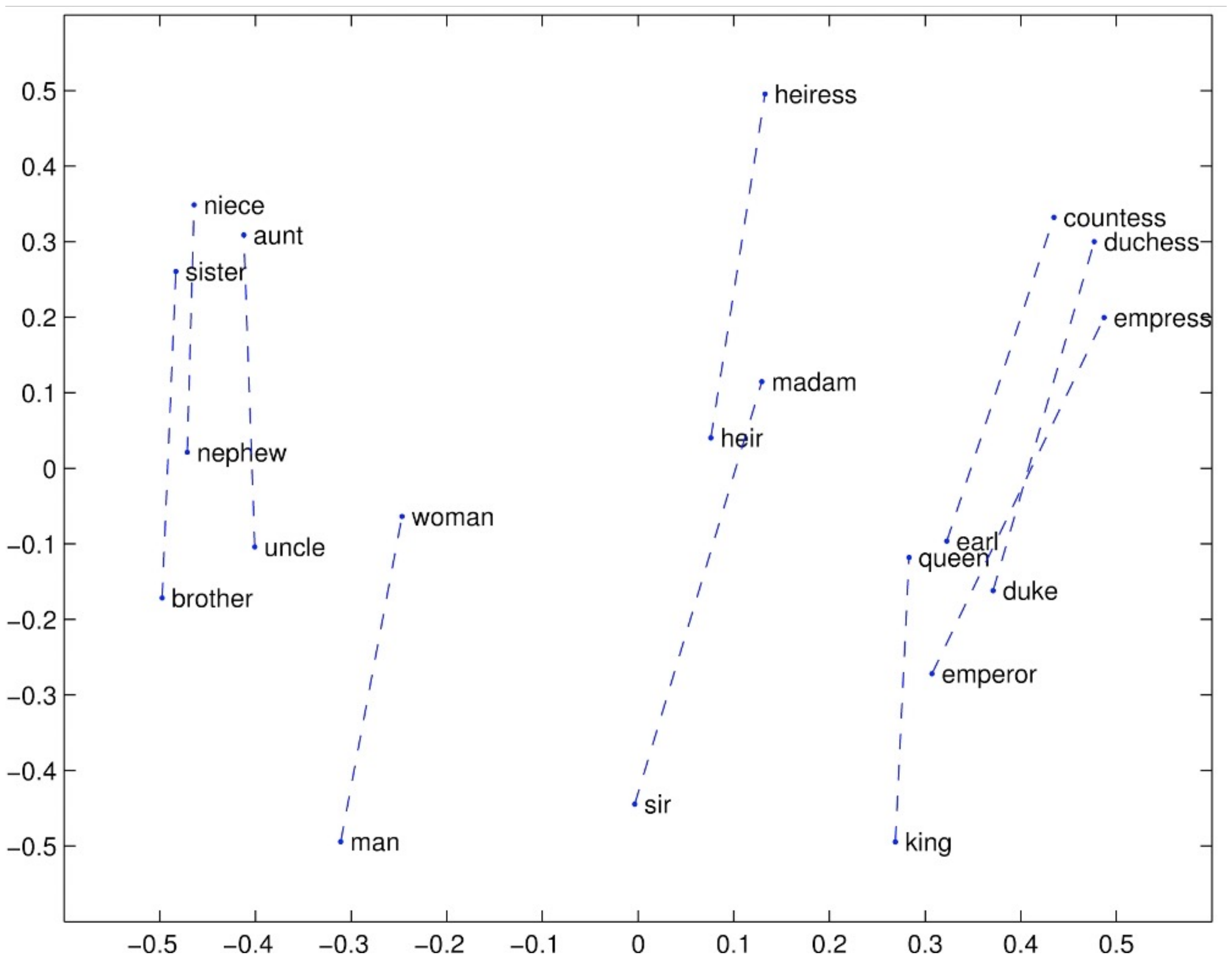
a:b = c:?

man:woman = king: ?

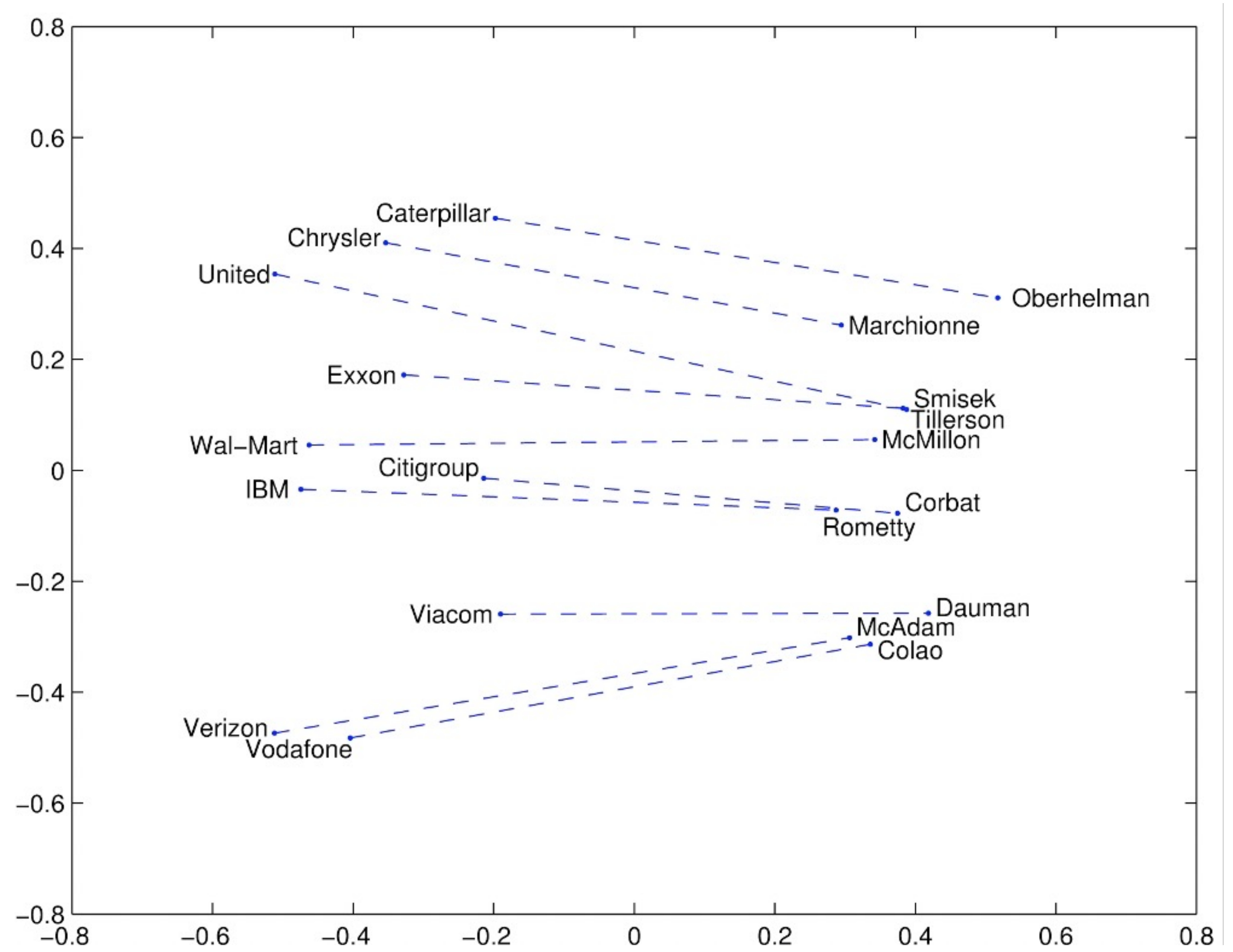
- Evaluation: find a word such that the vector is closest to *vec[man]-vec[woman]+vec[king]* according to the cosine similarity
  - Correct if the word found is *queen*
- Can be applied to test for syntactic analogy as well
  - *Quick*:*quickly* = *slow*:*slowly*



Gender relation

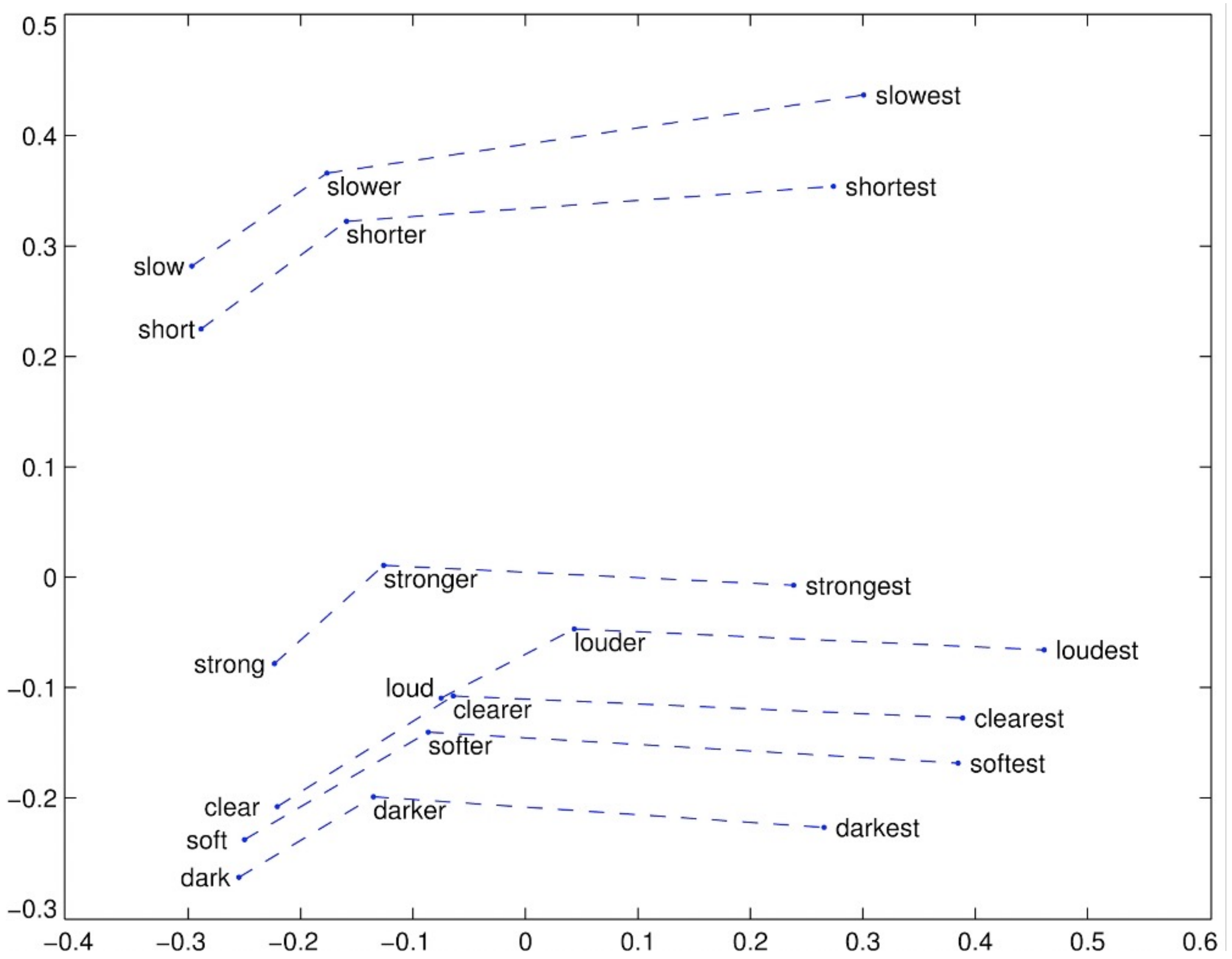


Company - CEO

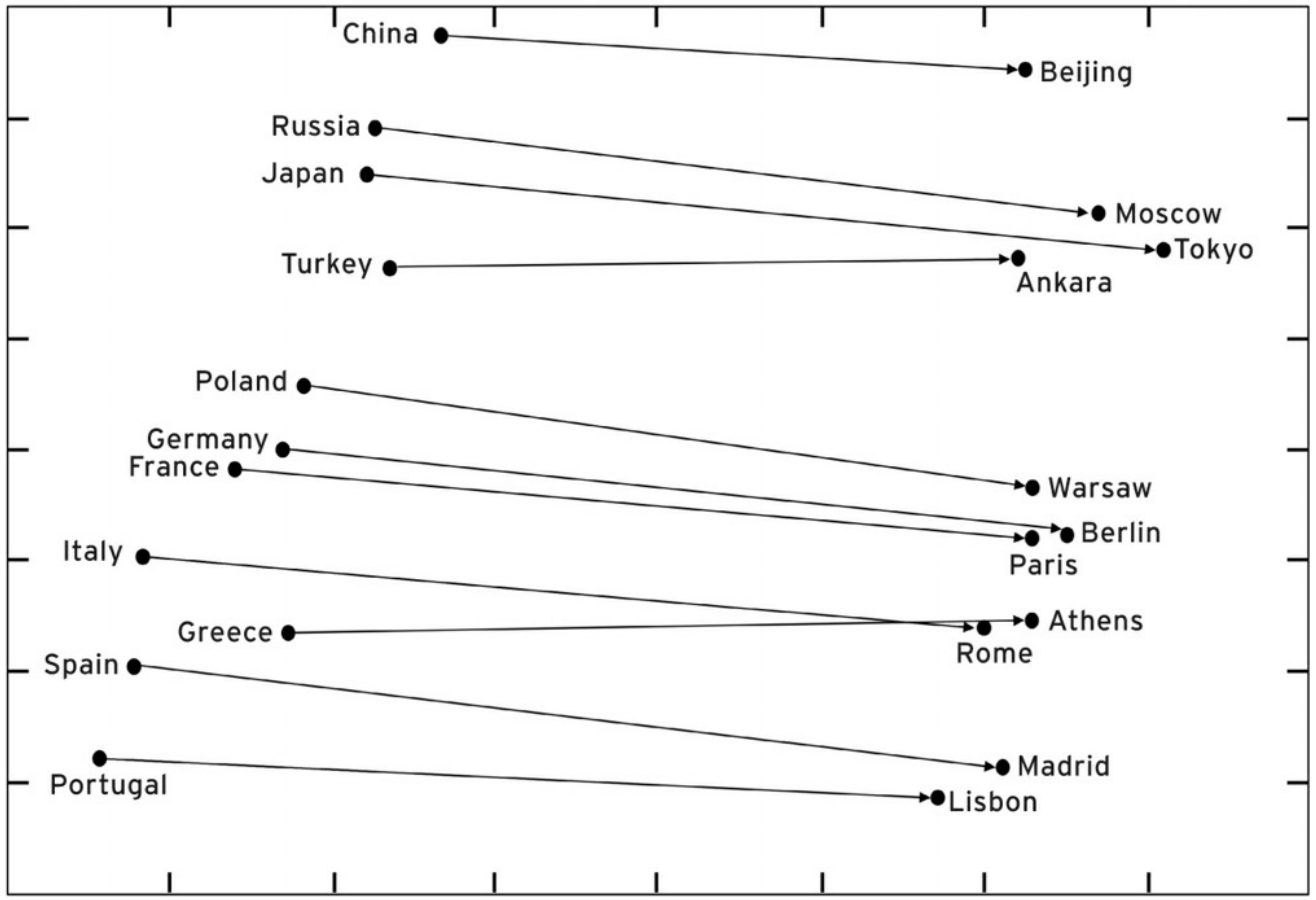


Caterpillar
Chrysler
United
Oberhelman
Marchionne
Exxon
Smisek
Tillerson
Wal–Mart
McMillon
Citigroup
IBM
Corbat
Rometty
Viacom
Dauman
McAdam
Colao
Verizon
Vodafone
0.8
0.6
0.4
0.2
0
−0.2
−0.4
−0.6
−0.8
−0.8
−0.6
−0.4
−0.2
0
0.2
0.4
0.6
0.8

Comparatives and Superlatives



Countries and their capital



But are word embeddings that good?

- By exploring the semantic space, you can also find analogies like
  - *Thirsty* is to *drink* as *tired* is to *drunk*
  - *Fish* is to *water* as *bird* is to *hydrant*

But are word embeddings that good?

- By exploring the semantic space, you can also find analogies like
  - *Thirsty* is to *drink* as *tired* is to *drunk*
  - *Fish* is to *water* as *bird* is to *hydrant*

  - *Man* is to *woman* as *computer programmer* is to ______
  - *Woman* is to *man* as *computer programmer* is to ______
  - *Man* is to *genius* as *woman* is to______
  - *Woman* is to *genius* as man is to ______

But are word embeddings that good?

- By exploring the semantic space, you can also find analogies like
  - *Thirsty* is to *drink* as *tired* is to *drunk*
  - *Fish* is to *water* as *bird* is to *hydrant*

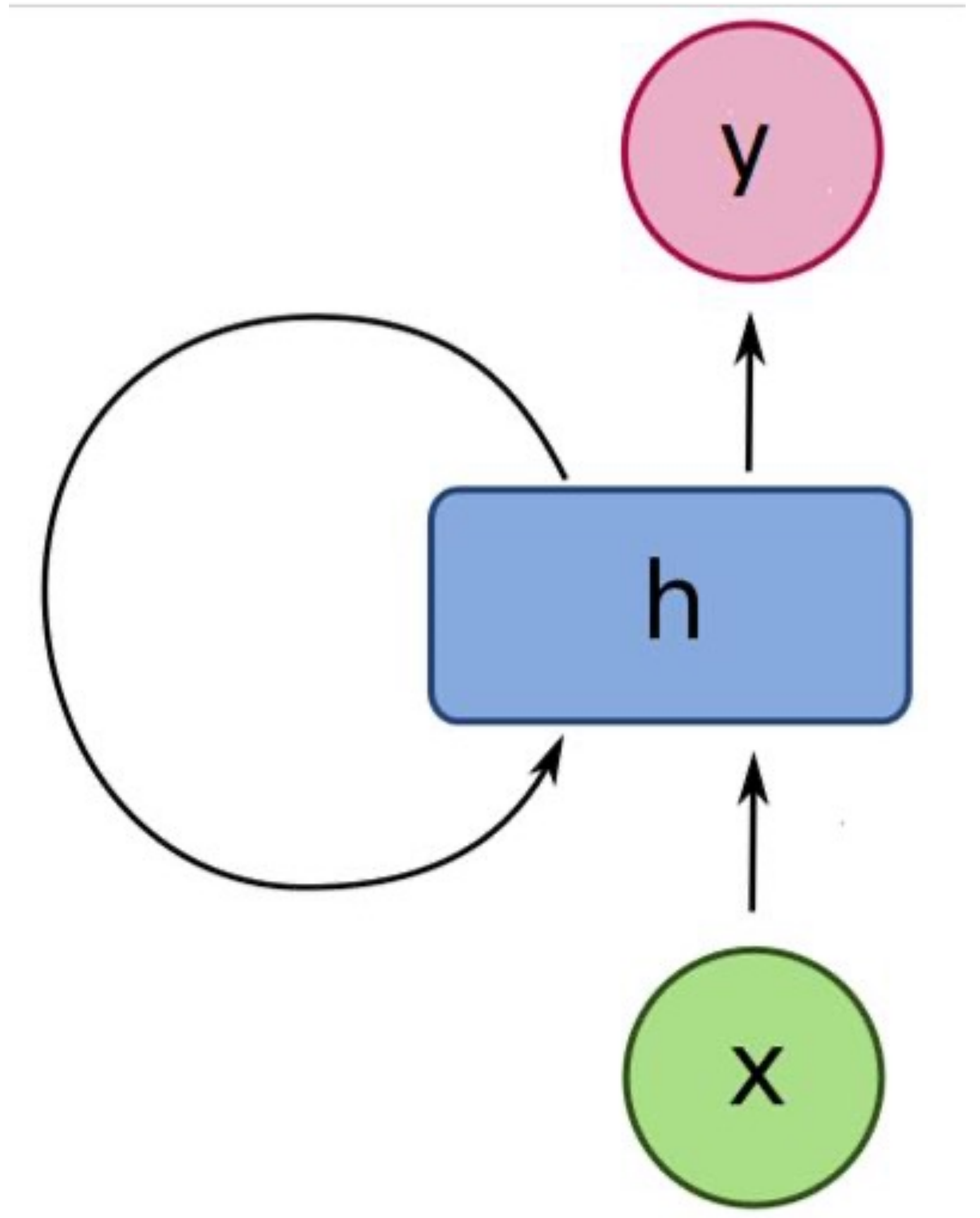
  - *Man* is to *woman* as *computer programmer* is to ______
  - *Woman* is to *man* as *computer programmer* is to ______
  - *Man* is to *genius* as *woman* is to ______
  - *Woman* is to *genius* as man is to ______

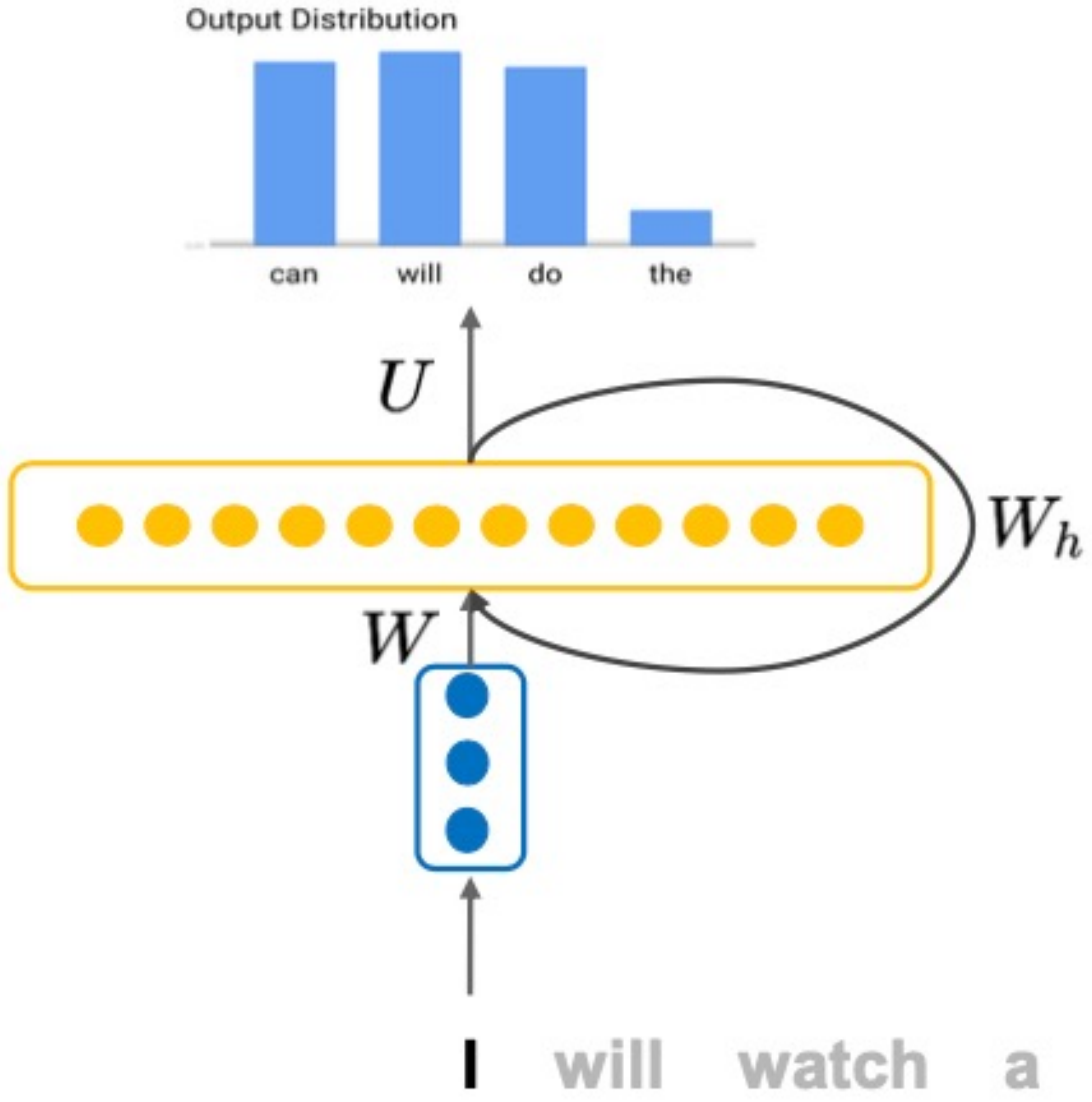
- Biases in word vectors might seep through to produce unexpected, hard-to-predict biases in widely used NLP applications

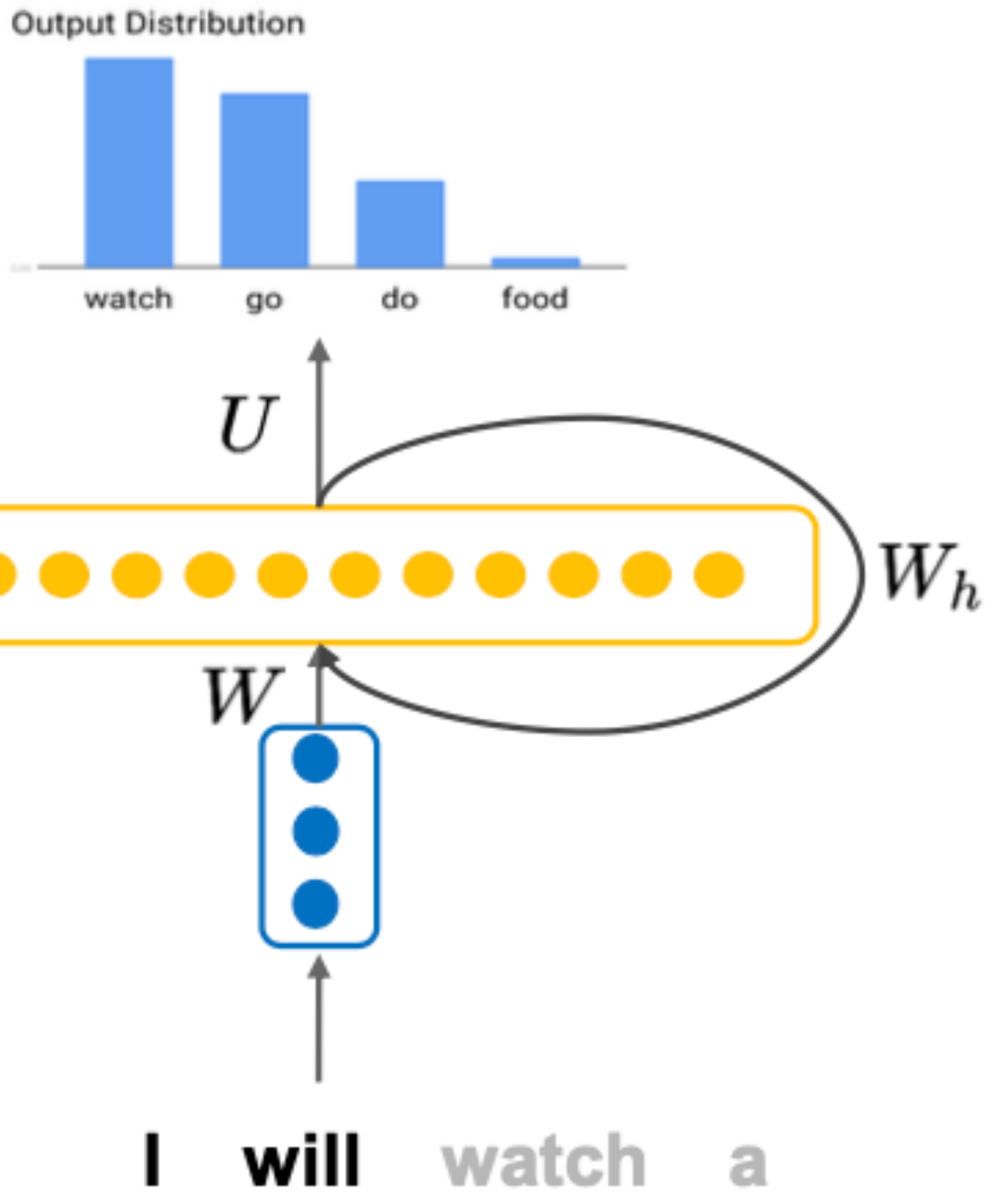
Extra: Recurrent Neural Networks

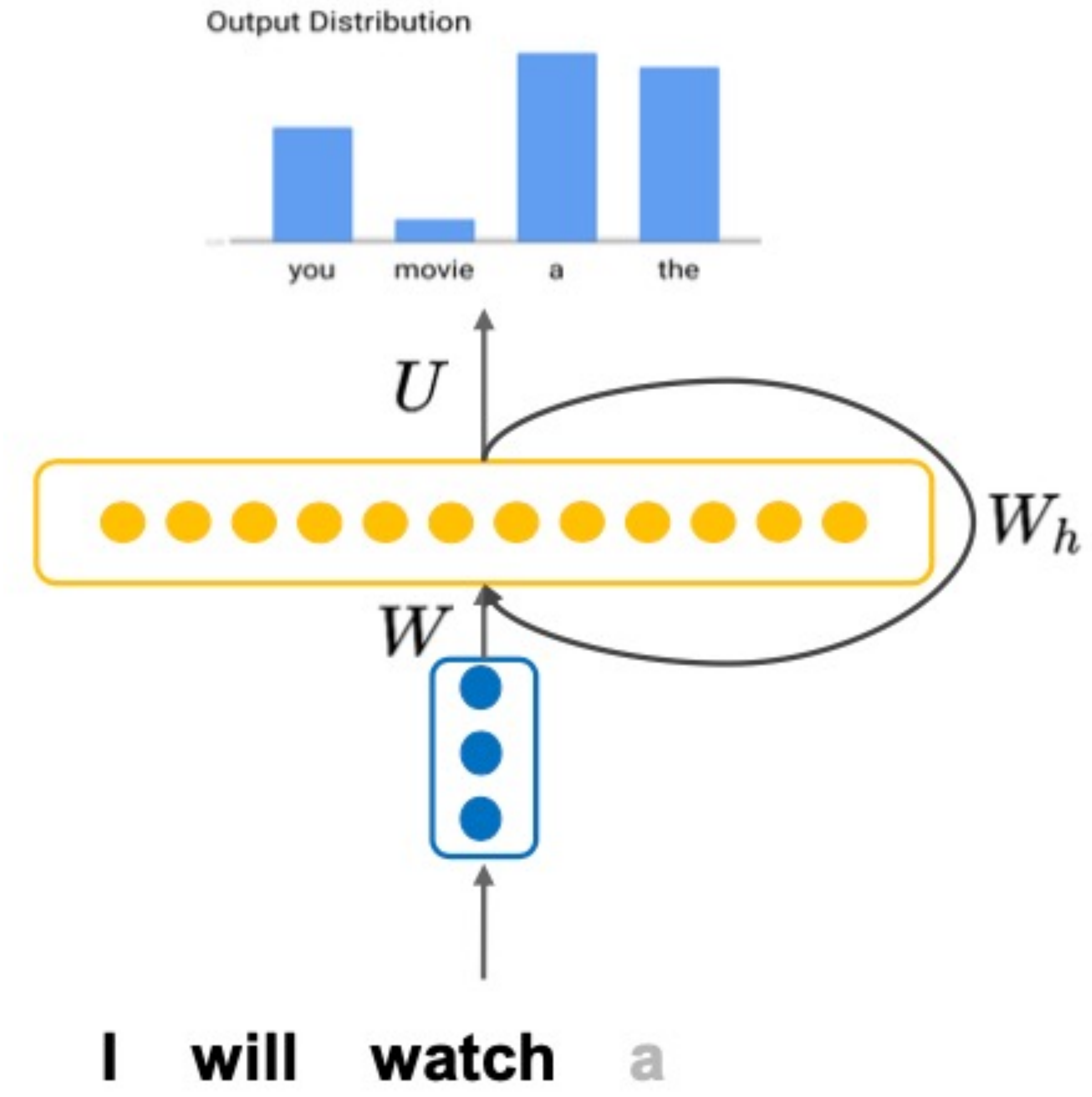
Recurrent Neural Network

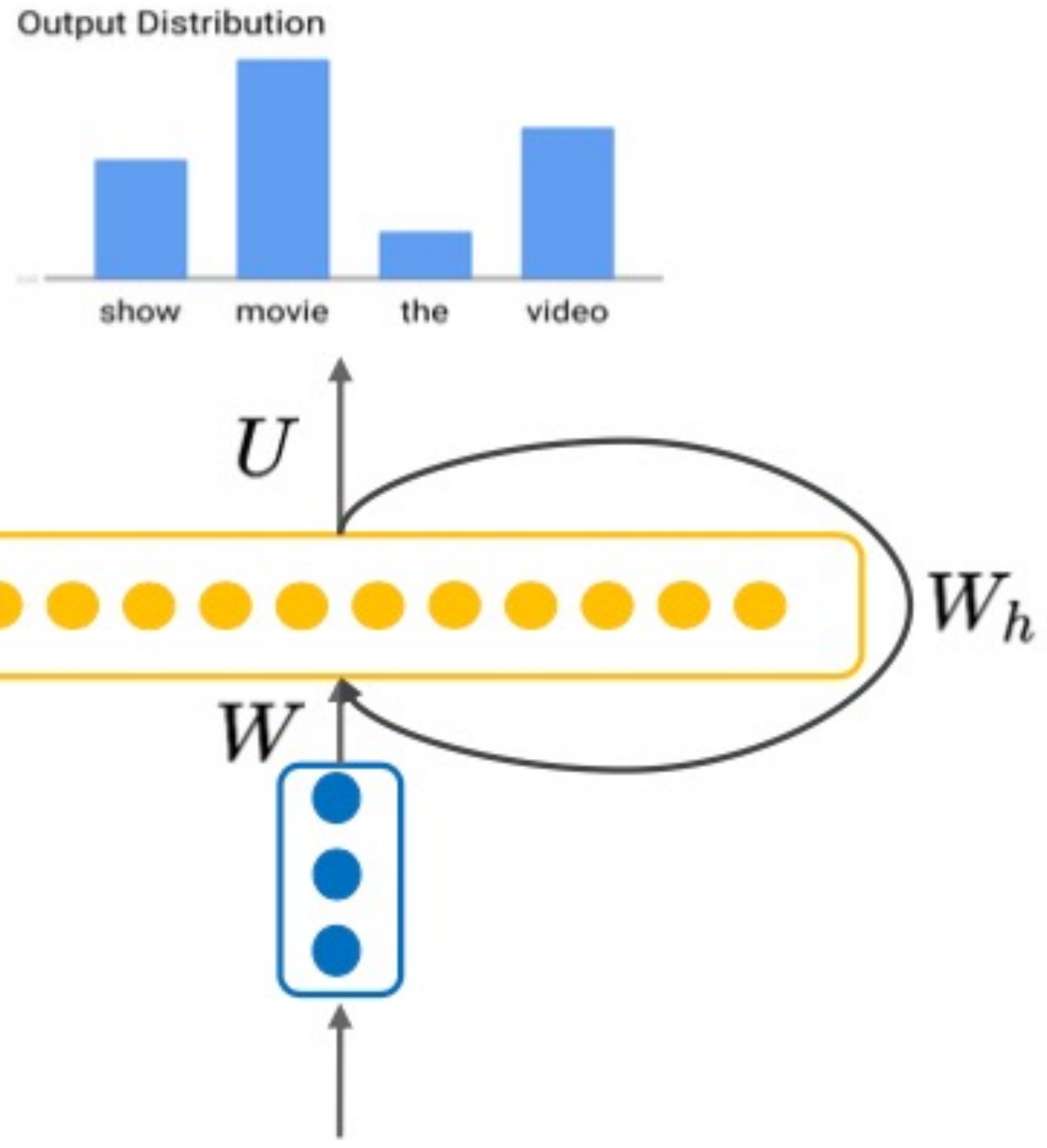
- Traditional neural networks can consider only a finite window of previous words
  - Also, the behaviour does not depend on the order in which inputs are presented
- Recurrent Neural Networks are capable of conditioning the model on ALL previous words
  - inspired by ideas on how the brain interprets sequences
- The hidden state has feedback connections that pass information about the past to the next input
  - Output can be produced at any step or only at the end of the sequence
- How to train an RNN?
  - feedback connections create loops, which is a problem since the update of weight depends on itself at the previous time step.
  - Solution: a recurrent neural network processing a sequence of length **T** is equivalent to a feedforward network obtained by the unfolding of the RNN **T** times
  - The unfolded network is trained with standard backpropagation with weight sharing




Output Distribution
can
will
do
the
U
W_h
W
I
will
watch
a


Output Distribution
watch
go
do
food
U
W_h
W
I
will
watch
a


Output Distribution
you
movie
a
the
U
W_h
W
I will watch a



I will watch a

What are RNNs for?

- Recurrent Neural Networks can be used in a variety of scenarios depending on how the inputs are fed and the outputs are interpreted

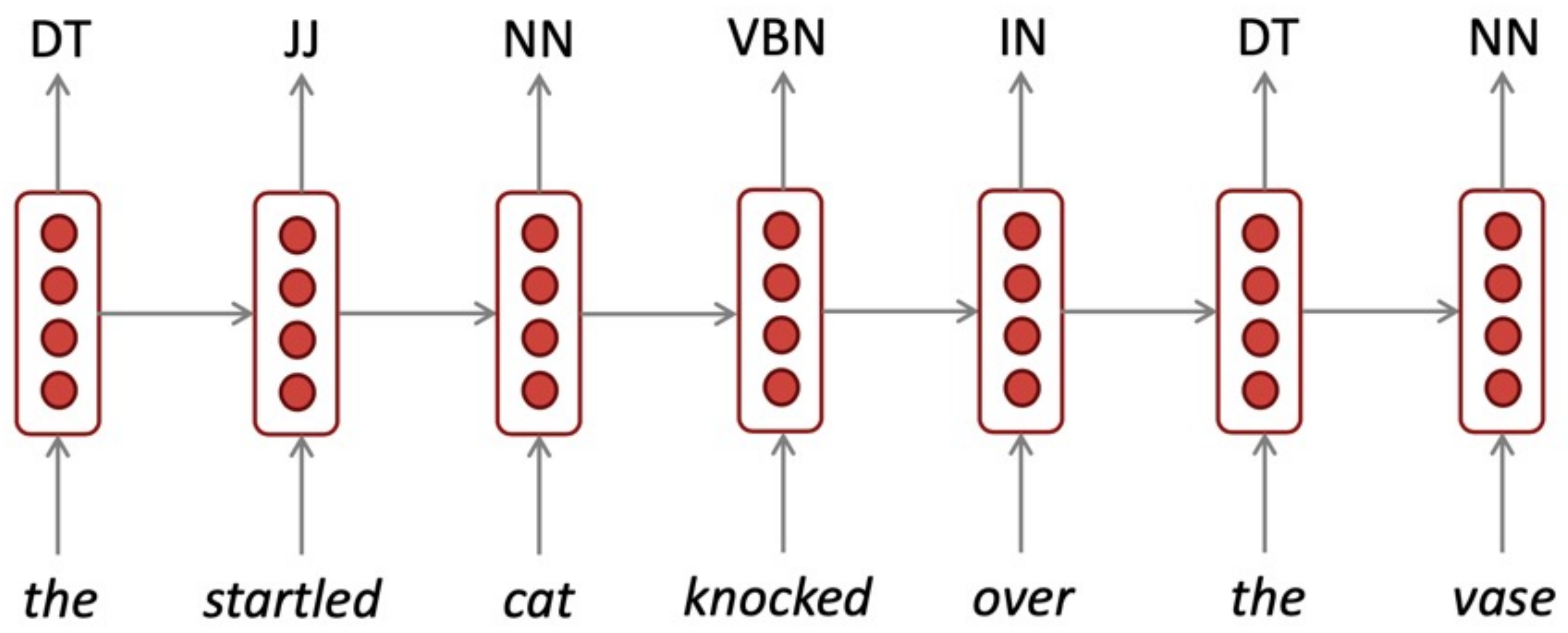
- Sequential input to sequential output
  - Machine translation / part-of-speech tagging and language modelling tasks lie within this class

- Sequential input to single output.
  - e.g sentiment analysis, in which we fed a sentence and we want to classify it as positive, neutral or negative

- Single input to sequential output
  - e.g. image captioning: where we fed a picture to the RNN and want to generate a description of it

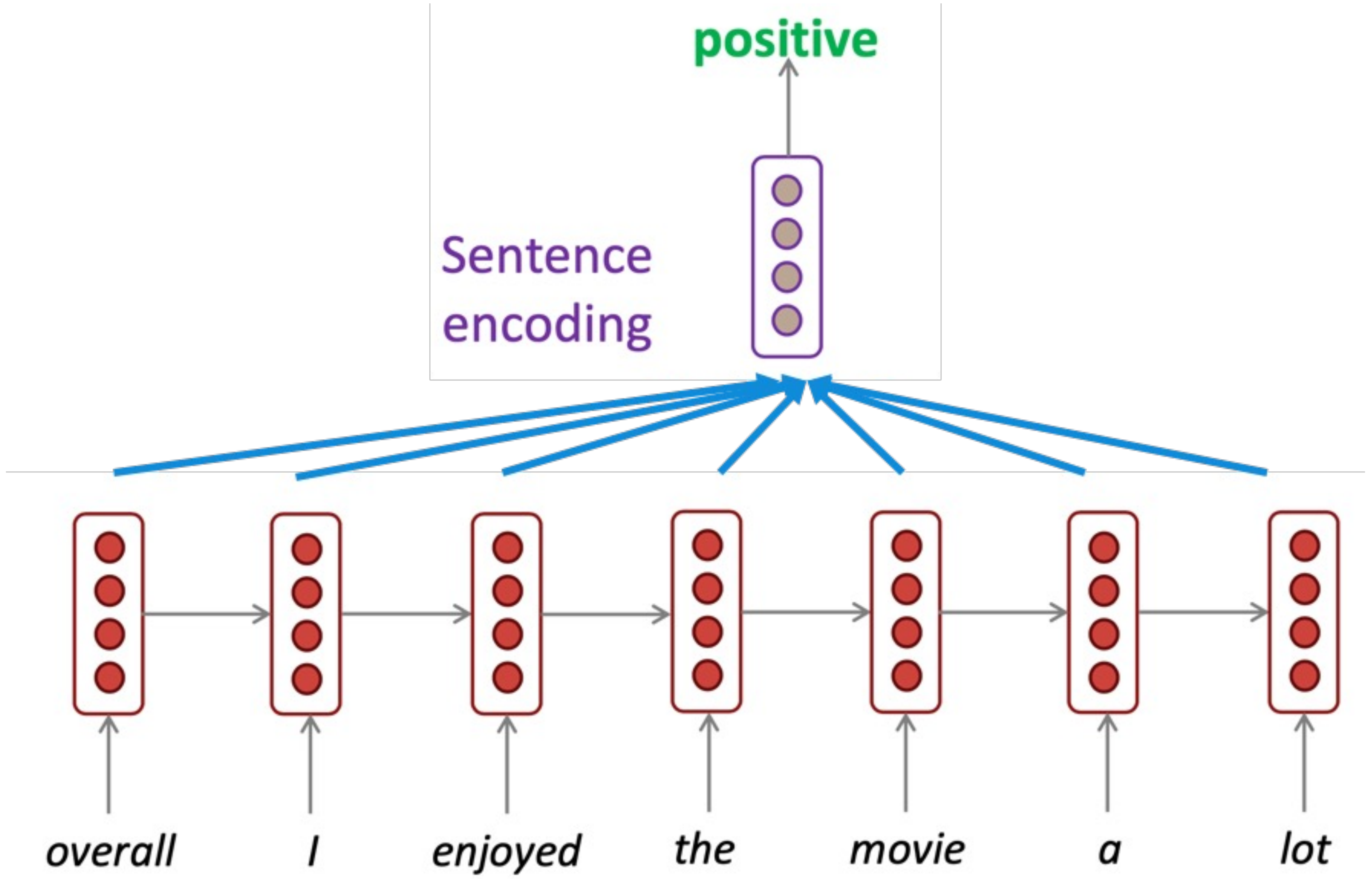
RNNs can be used for tagging

- e.g., part-of-speech tagging, named entity recognition



Sentence Classification

- e.g., sentiment classification



Pros and cons

- RNN Advantages:
  - Can process any length input
  - Computation for step t can (in theory) use information from many steps back
  - Model size doesn’t increase for longer input context

- RNN Disadvantages:
  - Recurrent computation is slow
  - In practice, difficult to access information from many steps back (gradient vanishing problem)

http://neuralnetworksanddeeplearning.com/chap5.html#the_vanishing_gradient_problem

Admin

Advanced Machine Learning For Design

Lecture 3 - Machine Learning and Natural Language Processing / Part 2

Evangelos Niforatos

05/10/2022

aml4d-ide@tudelft.nl
https://aml4design.github.io/

Sources

- CIS 419/519 Applied Machine Learning. Eric Eaton, Dinesh Jayaraman. https://www.seas.upenn.edu/~cis519/spring2020/
- EECS498: Conversational AI. Kevin Leach. https://dijkstra.eecs.umich.edu/eecs498/
- CS 4650/7650: Natural Language Processing. Diyi Yang. https://www.cc.gatech.edu/classes/AY2020/cs7650_spring/
- Natural Language Processing. Alan W Black and David Mortensen. http://demo.clab.cs.cmu.edu/NLP/
- IN4325 Information Retrieval. Jie Yang.
- Speech and Language Processing, An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition. Third Edition. Daniel Jurafsky, James H. Martin.
- Natural Language Processing, Jacob Eisenstein, 2018.