Information Retrieval

PANDARUS:
Alas, I think he shall be come approached and the day
When little srain would be attain'd into being never fed,
And who is but a chain and subjects of his death,
I should not sleep.

Second Senator:
They are away this miseries, produced upon my soul,
Breaking and strongly should be buried, when I perish
The earth and thoughts of many states.

DUKE VINCENTIO:
Well, your wit is in the care of side and that.

Second Lord:
They would be ruled after this chamber, and
my fair nues begun out of the fact, to be conveyed,
Whose noble souls I'll have the heart of the wars.

Clown:
Come, sir, I will make did behold your worship.

VIOLA:
I'll drink it.

What is NLP?

- Roughly divided between Language understanding and Language generation.
- Based on machine learning techniques but also linguistics.

Language generation: wikipedia

```
Naturalism and decision for the majority of Arab countries' capitalide was grounded
by the Irish language by [[John Clair]], [[An Imperial Japanese Revolt]], associated
with Guangzham's sovereignty. His generals were the powerful ruler of the Portugal
in the [[Protestant Immineners]], which could be said to be directly in Cantonese
Communication, which followed a ceremony and set inspired prison, training. The
emperor travelled back to [[Antioch, Perth, October 25|21]] to note, the Kingdom
of Costa Rica, unsuccessful fashioned the [[Thrales]], [[Cynth's Dajoard]], known
in western [[Scotland]], near Italy to the conquest of India with the conflict.
Copyright was the succession of independence in the slop of Syrian influence that
was a famous German movement based on a more popular servicious, non-doctrinal
and sexual power post. Many governments recognize the military housing of the
[[Civil Liberalization and Infantry Resolution 265 National Party in Hungary]],
that is sympathetic to be to the [[Punjab Resolution]]
(PJS)[http://www.humah.yahoo.com/guardian.
cfm/7754800786d17551963s89.htm Official economics Adjoint for the Nazism, Montgomery
was swear to advance to the resources for those Socialism's rule,
was starting to signing a major tripad of aid exile.]]
```

INT
המרכז הבינלאומי
ללימודי הייטק וחדשנות

Proof. Omitted. □

Lemma 0.1. *Let $\mathcal{C}$ be a set of the construction.*

Let $\mathcal{C}$ be a gerber covering. Let $\mathcal{F}$ be a quasi-coherent sheaves of $\mathcal{O}$-modules. We have to show that

$$\mathcal{O}_{\mathcal{O}_X} = \mathcal{O}_X(\mathcal{L})$$

.

Proof. This is an algebraic space with the composition of sheaves $\mathcal{F}$ on $X_{étale}$ we have

$$\mathcal{O}_X(\mathcal{F}) = \{morph_1 \times_{\mathcal{O}_X} (\mathcal{G}, \mathcal{F})\}$$

where $\mathcal{G}$ defines an isomorphism $\mathcal{F} \to \mathcal{F}$ of $\mathcal{O}$-modules. □

Lemma 0.2. *This is an integer $\mathcal{Z}$ is injective.*

Proof. See Spaces, Lemma ??. □

Lemma 0.3. *Let S be a scheme. Let X be a scheme and X is an affine open covering. Let $\mathcal{U} \subset \mathcal{X}$ be a canonical and locally of finite type. Let X be a scheme. Let X be a scheme which is equal to the formal complex.*

The following to the construction of the lemma follows.

Let X be a scheme. Let X be a scheme covering. Let

$$b : X \to Y' \to Y \to Y \to Y' \times_X Y \to X.$$

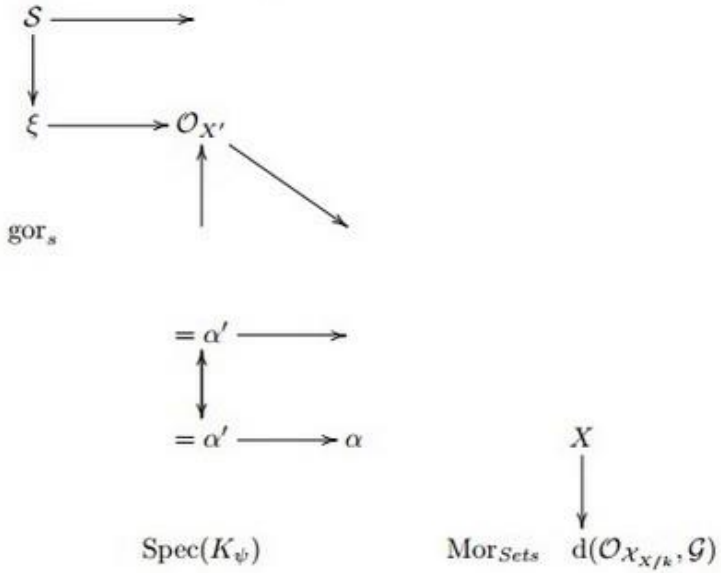
be a morphism of algebraic spaces over S and Y.

Proof. Let X be a nonzero scheme of X. Let X be an algebraic space. Let $\mathcal{F}$ be a quasi-coherent sheaf of $\mathcal{O}_X$-modules. The following are equivalent

(1) $\mathcal{F}$ is an algebraic space over S.
(2) If X is an affine open covering.

Consider a common structure on X and X the functor $\mathcal{O}_X(U)$ which is locally of finite type. □

This since $\mathcal{F} \in \mathcal{F}$ and $x \in \mathcal{G}$ the diagram



is a limit. Then $\mathcal{G}$ is a finite type and assume S is a flat and $\mathcal{F}$ and $\mathcal{G}$ is a finite type f_*. This is of finite type diagrams, and

- the composition of $\mathcal{G}$ is a regular sequence,
- $\mathcal{O}_{X'}$ is a sheaf of rings.

□

Proof. We have see that $X = \mathrm{Spec}(R)$ and $\mathcal{F}$ is a finite type representable by algebraic space. The property $\mathcal{F}$ is a finite morphism of algebraic stacks. Then the cohomology of X is an open neighbourhood of U. □

Proof. This is clear that $\mathcal{G}$ is a finite presentation, see Lemmas ??.
A *reduced above* we conclude that U is an open covering of $\mathcal{C}$. The functor $\mathcal{F}$ is a "field

$$\mathcal{O}_{X,x} \longrightarrow \mathcal{F}_{\overline{x}} \quad \text{-1}(\mathcal{O}_{X_{étale}}) \longrightarrow \mathcal{O}_{X_\ell}^{-1}\mathcal{O}_{X_\lambda}(\mathcal{O}_{X_\eta}^{\overline{v}})$$

is an isomorphism of covering of $\mathcal{O}_{X_i}$. If $\mathcal{F}$ is the unique element of $\mathcal{F}$ such that X is an isomorphism.
The property $\mathcal{F}$ is a disjoint union of Proposition ?? and we can filtered set of presentations of a scheme $\mathcal{O}_X$-algebra with $\mathcal{F}$ are opens of finite type over S.
If $\mathcal{F}$ is a scheme theoretic image points. □

If $\mathcal{F}$ is a finite direct sum $\mathcal{O}_{X_\lambda}$ is a closed immersion, see Lemma ??. This is a sequence of $\mathcal{F}$ is a similar morphism.

```
/*
 * Increment the size file of the new incorrect UI_FILTER group information
 * of the size generatively.
 */
static int indicate_policy(void)
{
  int error;
  if (fd == MARN_EPT) {
    /*
     * The kernel blank will coeld it to userspace.
     */
    if (ss->segment < mem_total)
      unblock_graph_and_set_blocked();
    else
      ret = 1;
    goto bail;
  }
  segaddr = in_SB(in.addr);
  selector = seg / 16;
  setup_works = true;
  for (i = 0; i < blocks; i++) {
    seq = buf[i++];
    bpf = bd->bd.next + i * search;
    if (fd) {
      current = blocked;
    }
  }
  rw->name = "Getjbbregs";
  bprm_self_clearl(&iv->version);
  regs->new = blocks[(BPF_STATS << info->historidac)] | PFMR_CLOBATHINC_SECONDS << 12;
  return segtable;
}
```

More applications / challenges

- Question answering, (bots).
- Machine translation.
- Spam detection.
- Information retrieval.

Used in Supervised, Unsupervised, Reinforcement and Semi-supervised learning

Basic steps to process text

1. Tokenization
2. Stopwords
3. Stemming or lemmatization

More advanced pre-processing

1. Parts of speech (POS)
2. Named entity recognition (NER)

Tokenization

The process of segmenting a text into words/sentences before an analysis.

Naturally, before any real text processing is to be done, text needs to be segmented into linguistic units such as **words**, **punctuation**, **numbers**, **alpha-numerics**, etc. This process is called tokenization.

Words can be often separated by blanks (white space), but not all white space is equal.
Both “Los Angeles” and “rock 'n' roll” present a single idea despite the fact that they contain multiple words and spaces.

We may also want to separate single words like “I'm” into separate words “I” and “am”.

Tokenization steps:

1. Segmenting Text into Words
2. Handling Abbreviations
3. Handling Hyphenated Words
4. Numerical and special expressions

Common Acronyms with Punctuation
1. I.O.U.
2. M.D.
3. N.B.
4. P.O.
5. U.K.
6. U.S.
7. U.S.A.
8. P.S.

Common Words containing Periods
1. .c
2. mr.
3. mrs.
4. .com
5. dr.
6. .sh
7. .java
8. st.

INT
המרכז הבינלאומי
ללימודי הייטק וחדשנות

Tokenization steps:

1.Segmenting Text into Words
2. Handling Abbreviations
3. Handling Hyphenated Words
4. Numerical and special expressions

Hyphenated segments present a case of ambiguity for a tokenizer:
sometimes a hyphen is part of a token: “self-assessment”, “F-15”, “forty-two”
sometimes it is not: “Los Angeles-based”

Tokenization steps:

the New York-based co-operative was fine- tuning forty-two K-9-like models.	
Token	Type
New York-based	**Sentential**
co-operative	**Lexical**
fine-tuning	**End-of-Line**, but could also be considered a **Lexical** hyphen based on the author's stylistic preferences.
Forty-two	**Lexical**
K-9-like	**Lexical** and **Sentential**

INT
המרכז הבינלאומי
ללימודי הייטק וחדשנות

Tokenization steps:

1. Segmenting Text into Words
2. Handling Abbreviations
3. Handling Hyphenated Words
4. Numerical and special expressions

1. 123-456-7890
2. (123)-456-7890
3. 123.456.7890
4. (123) 456-7890

Examples:

1. Email addresses
2. URLs
3. Complex enumeration of items
4. Telephone Numbers
5. Dates
6. Time
7. Measures
8. Vehicle Licence Numbers
9. Paper and book citations
10. etc

Date/Time Formats:

- 8th-Feb
- 8-Feb-2013
- 02/08/13
- February 8th, 2013
- Feb 8th

"I said, 'what're you? Crazy?'" said Sandowsky. "I can't afford to do that."

	Naïve Whitespace Parser	Apache Open NLP 1.5.2 (using en-token.bin)	Stanford 2.0.3	Custom	Hypothetical Tokenizer (Ideal Tokenization)
1		“	“	“	“
2	“i	i	i	i	i
3	said,	said	said	said	said
4		,	,	,	,
5	'what're	'what	`	'	'
6			what	what're	what
7		're	're		are
8	you?	you	you	you	you
9		?	?	?	?
10	crazy?'"	crazy	crazy	crazy	crazy
11		?	?	?	?
12		'	'	'	'
13	said	said	said	said	said
14	sandowsky.	sandowsky	sandowsky	sandowsky	sandowsky
15		.	.	.	.
16		'	'	'	“
17	'i	i	i	i	i
18	can't	ca	ca	can't	can
19		n't	n't		not
20	afford	afford	afford	afford	afford
21	to	to	to	to	to
22	do	do	do	do	do
23	that.'	that	that	that	that
24		.	.	.	.
25		'	'	'	'

Stopwords

a	an	and	are	as	at	be	by	for	from
has	he	in	is	it	its	of	on	that	the
to	was	were	will	with					

Usually, contain no information and can mislead processing by being the strongest terms in vectorizing techniques.

Stemming

- When searching for the word shoe, would you be interested in finding also shoes? So, should we just chop off ‘s’ from words?
- But how a about informative and information? Adjustable and adjustment?
- Would you turn Caress to Cares ?

Stemming ‘stems’ words to a basic form (not necessarily a valid word) to be common among different variants of the same meaning. Done by a lookup table, statistically or a set of rules.

Example: Porter stemmer.

Lemmatization

- When you search for **‘go’** would you be interested in ‘**went’**? How about ‘**better**’ and ‘**good’**?
- Would you want to notice the difference in the use of the word ‘**meeting’** (stems to ‘**meet**’) between **‘I am meeting Amnon today’** and **‘we have a meeting today’**?

A lemma is the base form of a word. It is a valid word.
Lemmatization is more challenging and requires dictionary and in some cases the context of the sentence.

Text to features

After preprocessing the text we need to decide how to turn the text into features.

Bag of words

Advantage:
Simple

Disadvantage:
No positioning context

```
(1) John likes to watch movies. Mary likes movies too.
```

```
(2) John also likes to watch football games.
```

Based on these two text documents, a list is constructed as follows:

```
[
    "John",
    "likes",
    "to",
    "watch",
    "movies",
    "also",
    "football",
    "games",
    "Mary",
    "too"
]
```

```
(1) [1, 2, 1, 1, 2, 0, 0, 0, 1, 1]
(2) [1, 1, 1, 1, 0, 1, 1, 1, 0, 0]
```

N-grams

```
[
    "John likes",
    "likes to",
    "to watch",
    "watch movies",
    "Mary likes",
    "likes movies",
    "movies too",
]
```

$$P(w_1, \ldots, w_m) = \prod_{i=1}^{m} P(w_i \mid w_1, \ldots, w_{i-1}) \approx \prod_{i=1}^{m} P(w_i \mid w_{i-(n-1)}, \ldots, w_{i-1})$$

$$= \frac{\mathrm{count}(w_{i-(n-1)}, \ldots, w_{i-1}, w_i)}{\mathrm{count}(w_{i-(n-1)}, \ldots, w_{i-1})}$$

In a bigram (n = 2) language model, the probability of the sentence *I saw the red house* is approximated as

$$\begin{aligned} & P(\mathrm{I}, \mathrm{saw}, \mathrm{the}, \mathrm{red}, \mathrm{house}) \\ \approx\ & P(\mathrm{I} \mid \langle s \rangle) P(\mathrm{saw} \mid \mathrm{I}) P(\mathrm{the} \mid \mathrm{saw}) P(\mathrm{red} \mid \mathrm{the}) P(\mathrm{house} \mid \mathrm{red}) P(\langle /s \rangle \mid \mathrm{house}) \end{aligned}$$

whereas in a trigram (n = 3) language model, the approximation is

$$\begin{aligned} & P(\mathrm{I}, \mathrm{saw}, \mathrm{the}, \mathrm{red}, \mathrm{house}) \\ \approx\ & P(\mathrm{I} \mid \langle s \rangle, \langle s \rangle) P(\mathrm{saw} \mid \langle s \rangle, I) P(\mathrm{the} \mid \mathrm{I}, \mathrm{saw}) P(\mathrm{red} \mid \mathrm{saw}, \mathrm{the}) P(\mathrm{house} \mid \mathrm{the}, \mathrm{red}) P(\langle /s \rangle \mid \mathrm{red}, \mathrm{house}) \end{aligned}$$

N-grams

```
[
    "John likes",
    "likes to",
    "to watch",
    "watch movies",
    "Mary likes",
    "likes movies",
    "movies too",
]
```

$$P(w_1, \ldots, w_m) = \prod_{i=1}^{m} P(w_i \mid w_1, \ldots, w_{i-1}) \approx \prod_{i=1}^{m} P(w_i \mid w_{i-(n-1)}, \ldots, w_{i-1})$$

$$= \frac{\mathrm{count}(w_{i-(n-1)}, \ldots, w_{i-1}, w_i)}{\mathrm{count}(w_{i-(n-1)}, \ldots, w_{i-1})}$$

In a bigram (n = 2) language model, the probability of the sentence *I saw the red house* is approximated as

$$P(\mathrm{I}, \mathrm{saw}, \mathrm{the}, \mathrm{red}, \mathrm{house})$$
$$\approx P(\mathrm{I} \mid \langle s \rangle) P(\mathrm{saw} \mid \mathrm{I}) P(\mathrm{the} \mid \mathrm{saw}) P(\mathrm{red} \mid \mathrm{the}) P(\mathrm{house} \mid \mathrm{red}) P(\langle /s \rangle \mid \mathrm{house})$$

whereas in a trigram (n = 3) language model, the approximation is

$$P(\mathrm{I}, \mathrm{saw}, \mathrm{the}, \mathrm{red}, \mathrm{house})$$
$$\approx P(\mathrm{I} \mid \langle s \rangle, \langle s \rangle) P(\mathrm{saw} \mid \langle s \rangle, I) P(\mathrm{the} \mid \mathrm{I}, \mathrm{saw}) P(\mathrm{red} \mid \mathrm{saw}, \mathrm{the}) P(\mathrm{house} \mid \mathrm{the}, \mathrm{red}) P(\langle /s \rangle \mid \mathrm{red}, \mathrm{house})$$

Zipf Law

- Zipf’s Law was proposed by linguist George Kingsley Zipf, describing the analogous pattern that appears in language. This law was proposed related to patterns seen in natural language corpus’s many times.

Zipf Law

- Given a large corpus of natural language occurrences, thefrequency of any word is inversely proportional to its ranking frequency table.

The most appearing word in a corpus suppose has frequency f

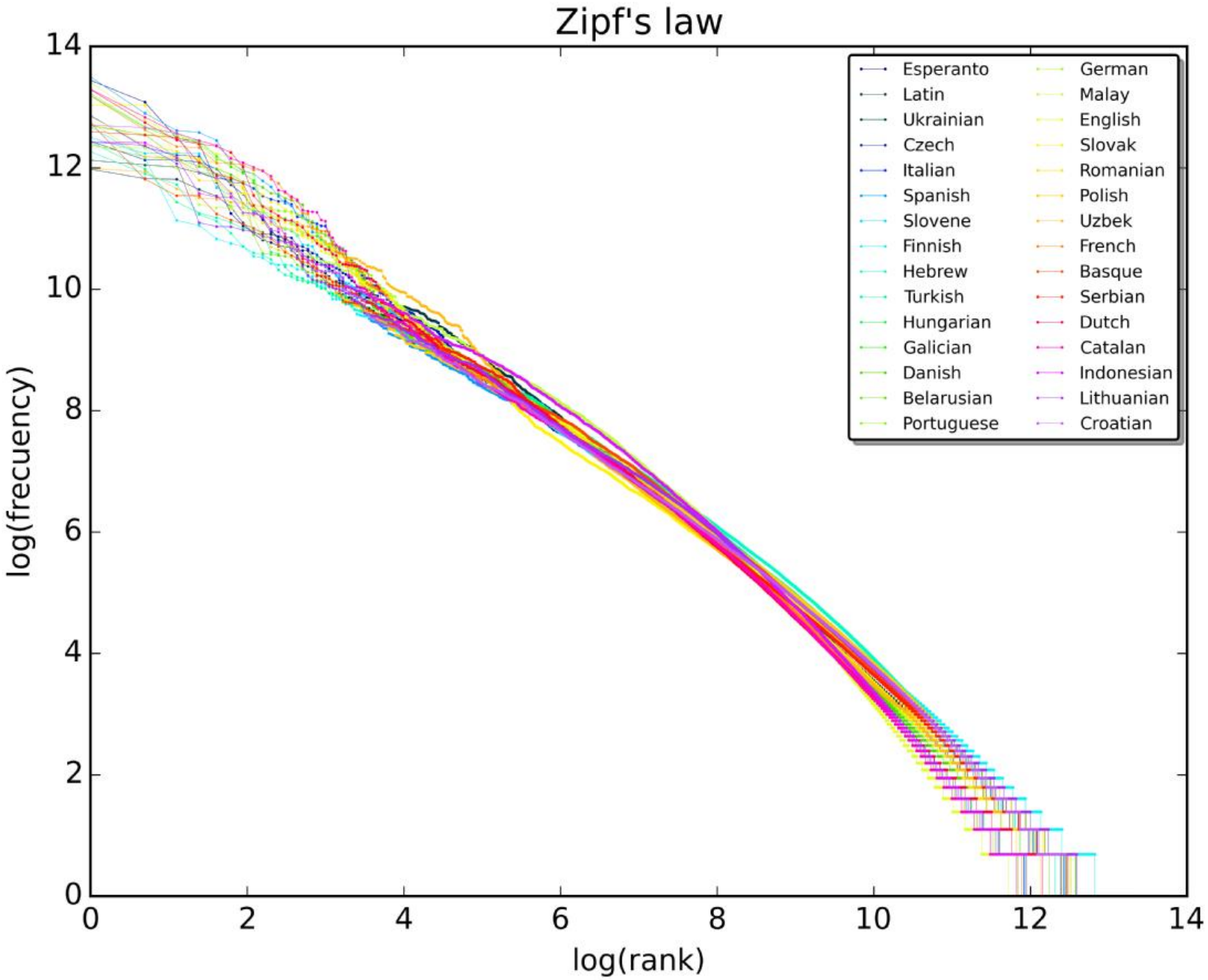
The second most appearing word would have frequency roughly f/2

Then the third most appearing word would have frequency roughly f/3

Then the fourth most appearing word would have frequency roughly f/4

In general, Frequency = 1/Rank.

What is astonishing is that this law holds true for almost all huge natural language corpus's out there.



Term-document count matrices

- Consider the number of occurrences of a term in a document:
  - Each document is a count vector in $\mathbb{N}^v$: a column below

	Antony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth
Antony	157	73	0	0	0	0
Brutus	4	157	0	1	0	0
Caesar	232	227	0	2	1	1
Calpurnia	0	10	0	0	0	0
Cleopatra	57	0	0	0	0	0
mercy	2	0	3	5	5	1
worser	2	0	1	1	1	0

Term frequency tf

- The term frequency $tf_{t,d}$ of term *t* in document *d* is defined as the number of times that *t* occurs in *d*.
- We want to use tf when computing query-document match scores. But how?
- Raw term frequency is not what we want:
  - A document with 10 occurrences of the term is more relevant than a document with 1 occurrence of the term.
  - But not 10 times more relevant.
- Relevance does not increase proportionally with term frequency.

Log-frequency weighting

- The log frequency weight of term t in d is

$$w_{t,d} = \begin{cases} 1 + \log_{10} \mathrm{tf}_{t,d}, & \text{if } \mathrm{tf}_{t,d} > 0 \\ 0, & \text{otherwise} \end{cases}$$

- 0 → 0, 1 → 1, 2 → 1.3, 10 → 2, 1000 → 4, etc.
- Score for a document-query pair: sum over terms *t* in both *q* and *d*:
- score

$$= \sum_{t \in q \cap d} (1 + \log \mathrm{tf}_{t,d})$$

- The score is 0 if none of the query terms is present in the document.

Document frequency

- Rare terms are more informative than frequent terms
  - Recall stop words
- Consider a term in the query that is rare in the collection (e.g., *arachnocentric*)
- A document containing this term is very likely to be relevant to the query *arachnocentric*
- We want a high weight for rare terms like *arachnocentric*.

Document frequency

- Frequent terms are less informative than rare terms
- Consider a query term that is frequent in the collection (e.g., *high, increase, line*)
- A document containing such a term is more likely to be relevant than a document that doesn't
- But it's not a sure indicator of relevance.
- → For frequent terms, we want high positive weights for words like *high, increase, and line*
- But lower weights than for rare terms.
- We will use document frequency (df) to capture this.

idf weight

- df_t is the document frequency of t: the number of documents that contain t
  - df_t is an inverse measure of the informativeness of t
  - $df_t \leq N$
- We define the idf (inverse document frequency) of t by

$$idf_t = \log_{10}(N/df_t)$$

  - We use log (N/df_t) instead of N/df_t to "dampen" the effect of idf.

Will turn out the base of the log is immaterial.

idf example, suppose N = 1 million

term	df_t	idf_t
calpurnia	1	
animal	100	
sunday	1,000	
fly	10,000	
under	100,000	
the	1,000,000	

$$\text{idf}_t = \log_{10}(N/\text{df}_t)$$

There is one idf value for each term t in a collection.

tf-idf weighting

- The tf-idf weight of a term is the product of its tf weight and its idf weight.

$$w_{t,d} = \log(1 + \mathrm{tf}_{t,d}) \times \log_{10}(N / \mathrm{df}_t)$$

- Best known weighting scheme in information retrieval
- Increases with the number of occurrences within a document
- Increases with the rarity of the term in the collection

Score for a document given a query

$$\text{Score}(q,d) = \sum_{t \in q \cap d} \text{tf.idf}_{t,d}$$

- Many variants
  - How “tf” is computed (with/without logs)
  - Whether the terms in the query are also weighted
  - …

weight matrix

	Antony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth
Antony	5.25	3.18	0	0	0	0.35
Brutus	1.21	6.1	0	1	0	0
Caesar	8.59	2.54	0	1.51	0.25	0
Calpurnia	0	1.54	0	0	0	0
Cleopatra	2.85	0	0	0	0	0
mercy	1.51	0	1.9	0.12	5.25	0.88
worser	1.37	0	0.11	4.15	0.25	1.95

Each document is now represented by a real-valued vector of tf-idf weights $\in \mathbb{R}^{|V|}$

INT
המרכז הבינלאומי
ללימודי הייטק וחדשנות

Documents as vectors

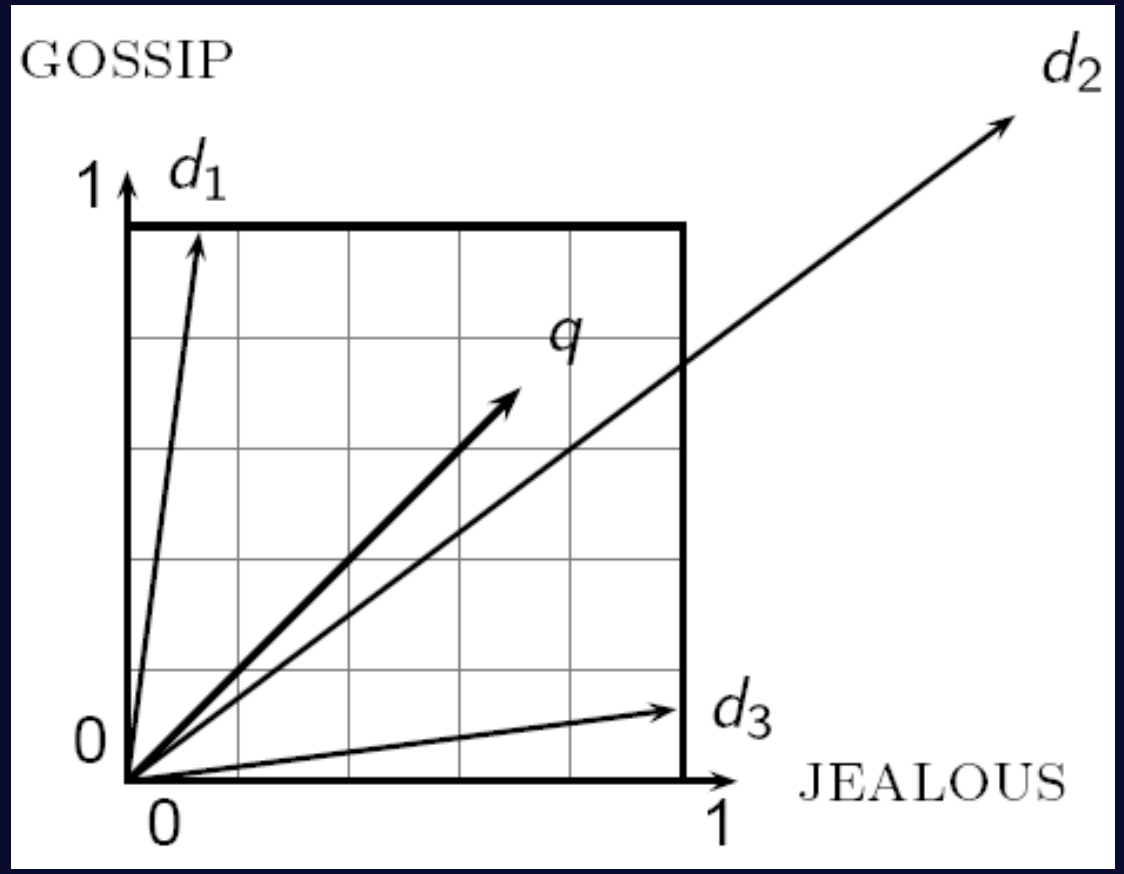
- So we have a |V|-dimensional vector space
- Terms are axes of the space
- Documents are points or vectors in this space
- Very high-dimensional: tens of millions of dimensions when you apply this to a web search engine
- These are very sparse vectors - most entries are zero.

Vector space proximity

- First cut: distance between two points
  - (= distance between the end points of the two vectors)
- Euclidean distance?
- Euclidean distance is a bad idea . . .
- . . . because Euclidean distance is large for vectors of different lengths.

Why distance is a bad idea

The Euclidean distance between $\vec{q}$ and $\vec{d_2}$ is large even though the distribution of terms in the query $\vec{q}$ and the distribution of terms in the document $\vec{d_2}$ are very similar.



Use angle instead of distance

- Thought experiment: take a document *d* and append it to itself. Call this document *d*′.
- “Semantically” d and d’ have the same content
- The Euclidean distance between the two documents can be quite large
- The angle between the two documents is 0, corresponding to maximal similarity.

- Key idea: Rank documents according to angle with query.

From angles to cosines

- The following two notions are equivalent.
  - Rank documents in decreasing order of the angle between query and document
  - Rank documents in increasing order of cosine(query,document)
- Cosine is a monotonically decreasing function for the interval [0^o, 180^o]

Length normalization

- A vector can be (length-) normalized by dividing each of its components by its length – for this we use the L_2 norm:

$$\|\vec{x}\|_2 = \sqrt{\sum_i x_i^2}$$

- Dividing a vector by its L_2 norm makes it a unit (length) vector (on surface of unit hypersphere)
- Effect on the two documents d and d′ (d appended to itself) from earlier slide: they have identical vectors after length-normalization.
  - Long and short documents now have comparable weights

cosine(query,document)

Dot product

Unit vectors

$$\cos(\vec{q},\vec{d}) = \frac{\vec{q}\bullet\vec{d}}{|\vec{q}||\vec{d}|} = \frac{\vec{q}}{|\vec{q}|}\bullet\frac{\vec{d}}{|\vec{d}|} = \frac{\sum_{i=1}^{|V|} q_i d_i}{\sqrt{\sum_{i=1}^{|V|} q_i^2}\sqrt{\sum_{i=1}^{|V|} d_i^2}}$$

q_i is the tf-idf weight of term i in the query
d_i is the tf-idf weight of term i in the document

$\cos(\vec{q},\vec{d})$ is the cosine similarity of $\vec{q}$ and $\vec{d}$ … or, equivalently, the cosine of the angle between $\vec{q}$ and $\vec{d}$.

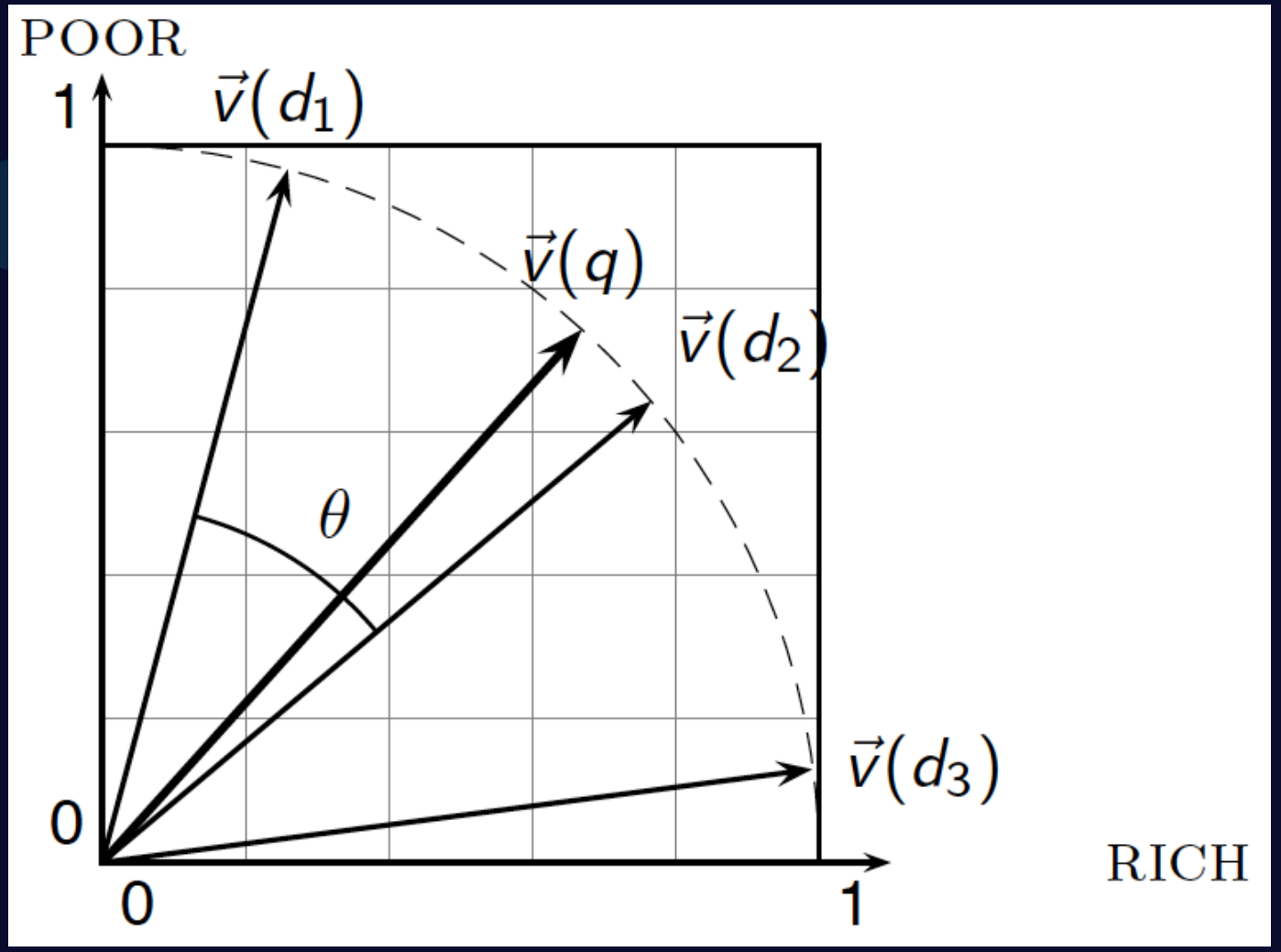
Cosine for length-normalized vectors

- For length-normalized vectors, cosine similarity is simply the dot product (or scalar product):

$$\cos(\vec{q},\vec{d}) = \vec{q} \bullet \vec{d} = \sum_{i=1}^{|V|} q_i d_i$$

for q, d length-normalized.

Cosine similarity illustrated



Cosine similarity amongst 3 documents

How similar are the novels SaS: *Sense and Sensibility* PaP: *Pride and Prejudice*, and WH: *Wuthering Heights*?

term	SaS	PaP	WH
affection	115	58	20
jealous	10	7	11
gossip	2	0	6
wuthering	0	0	38

Term frequencies (counts)

Note: To simplify this example, we don't do idf weighting.

3 documents example contd.

Log frequency weighting

term	SaS	PaP	WH
affection	3.06	2.76	2.30
jealous	2.00	1.85	2.04
gossip	1.30	0	1.78
wuthering	0	0	2.58

After length normalization

term	SaS	PaP	WH
affection	0.789	0.832	0.524
jealous	0.515	0.555	0.465
gossip	0.335	0	0.405
wuthering	0	0	0.588

cos(SaS,PaP) ≈
0.789 × 0.832 + 0.515 × 0.555 + 0.335 × 0.0 + 0.0 × 0.0
≈ 0.94
cos(SaS,WH) ≈ 0.79
cos(PaP,WH) ≈ 0.69

tf-idf weighting has many variants

Term frequency		Document frequency		Normalization	
n (natural)	$\mathrm{tf}_{t,d}$	n (no)	1	n (none)	1
l (logarithm)	$1+\log(\mathrm{tf}_{t,d})$	t (idf)	$\log \frac{N}{\mathrm{df}_t}$	c (cosine)	$\frac{1}{\sqrt{w_1^2+w_2^2+\ldots+w_M^2}}$
a (augmented)	$0.5+\frac{0.5\times\mathrm{tf}_{t,d}}{\max_t(\mathrm{tf}_{t,d})}$	p (prob idf)	$\max\{0, \log \frac{N-\mathrm{df}_t}{\mathrm{df}_t}\}$	u (pivoted unique)	$1/u$
b (boolean)	$\begin{cases}1 & \text{if } \mathrm{tf}_{t,d}>0\\ 0 & \text{otherwise}\end{cases}$			b (byte size)	$1/CharLength^{\alpha}$, $\alpha<1$
L (log ave)	$\frac{1+\log(\mathrm{tf}_{t,d})}{1+\log(\mathrm{ave}_{t\in d}(\mathrm{tf}_{t,d}))}$				

Columns headed ‘n’ are acronyms for weight schemes.

Why is the base of the log in idf immaterial?

Summary – vector space ranking

- Represent the query as a weighted tf-idf vector
- Represent each document as a weighted tf-idf vector
- Compute the cosine similarity score for the query vector and each document vector
- Rank documents with respect to the query by score
- Return the top *K* (e.g., *K* = 10) to the user

Word Embedding

Bag of words

Advantage:
Simple

Disadvantage:
No positioning context

```
(1) John likes to watch movies. Mary likes movies too.
```

```
(2) John also likes to watch football games.
```

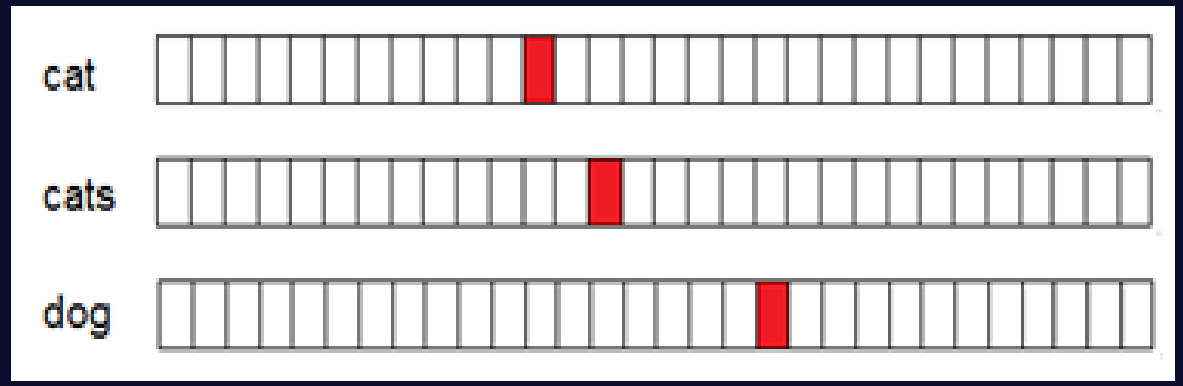
Based on these two text documents, a list is constructed as follows:

```
[
    "John",
    "likes",
    "to",
    "watch",
    "movies",
    "also",
    "football",
    "games",
    "Mary",
    "too"
]
```

```
(1) [1, 2, 1, 1, 2, 0, 0, 0, 1, 1]
(2) [1, 1, 1, 1, 0, 1, 1, 1, 0, 0]
```

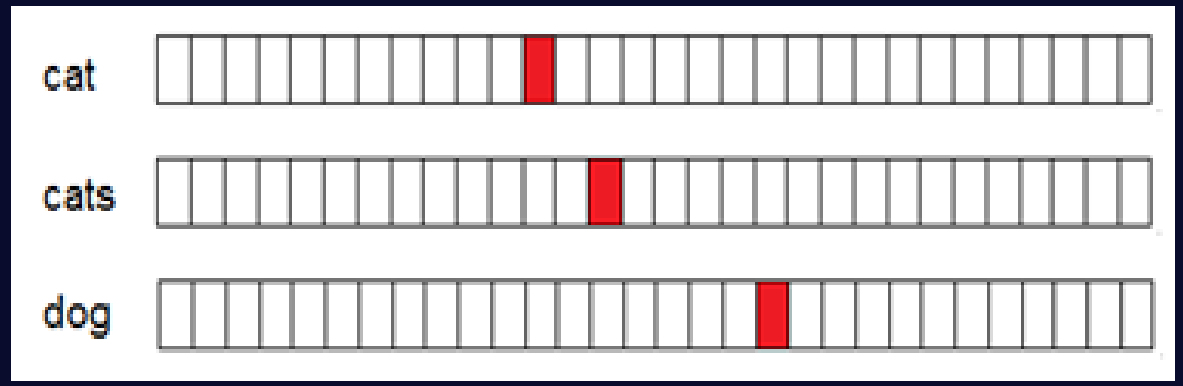
One hot vectors

One hot



One hot vectors

One hot



~100,000 free parameters

no semantics

Joint distribution

1. I enjoy flying.
2. I like NLP.
3. I like deep learning.

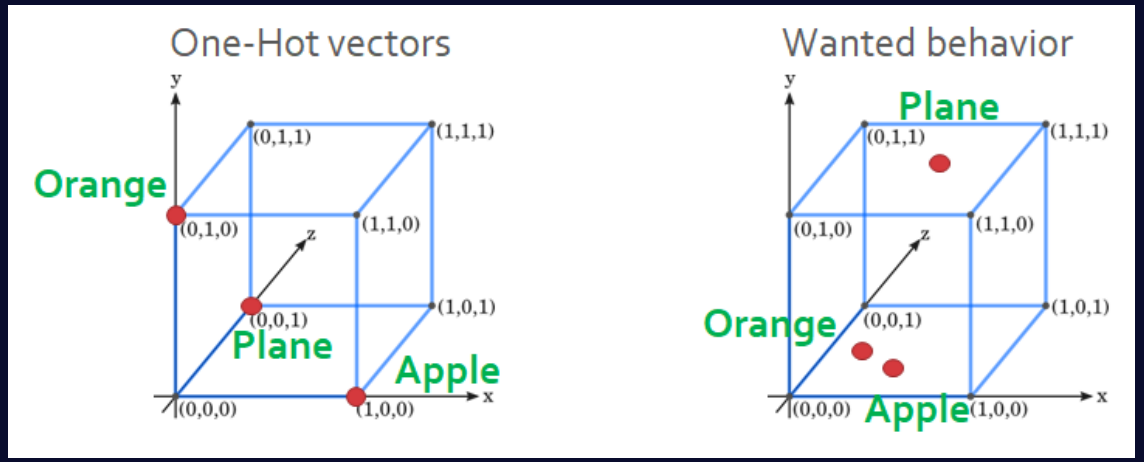
$$X = \begin{array}{c} \\ I \\ like \\ enjoy \\ deep \\ learning \\ NLP \\ flying \\ . \end{array} \begin{array}{c} \begin{array}{cccccccc} I & like & enjoy & deep & learning & NLP & flying & . \end{array} \\ \begin{bmatrix} 0 & 2 & 1 & 0 & 0 & 0 & 0 & 0 \\ 2 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \end{bmatrix} \end{array}$$

~$100{,}000^{10}$ free parameters

Joint distribution

- Similar words - close embedding
- Distant words - far embeddings

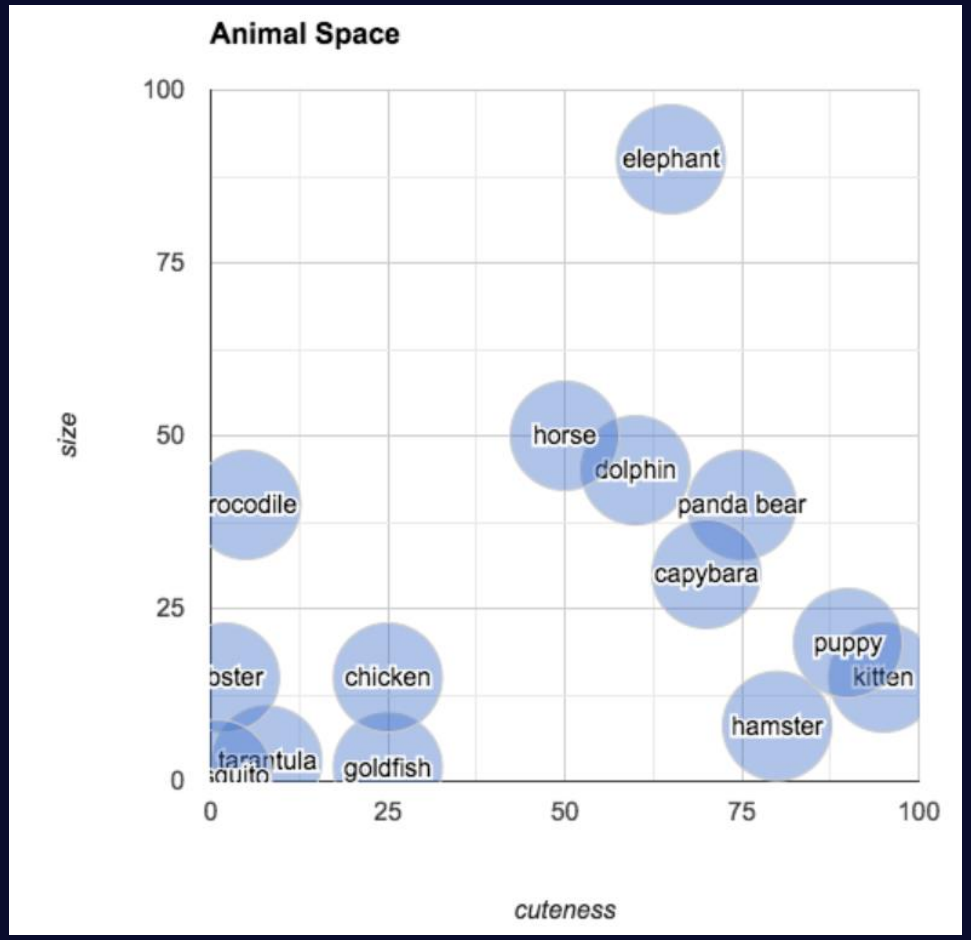
One hot



Word Embeddings

	cuteness (0–100)	size (0–100)
kitten	95	15
hamster	80	8
tarantula	8	3
puppy	90	20
crocodile	5	40
dolphin	60	45
panda bear	75	40
lobster	2	15
capybara	70	30
elephant	65	90
mosquito	1	1
goldfish	25	2
horse	50	50
chicken	25	15

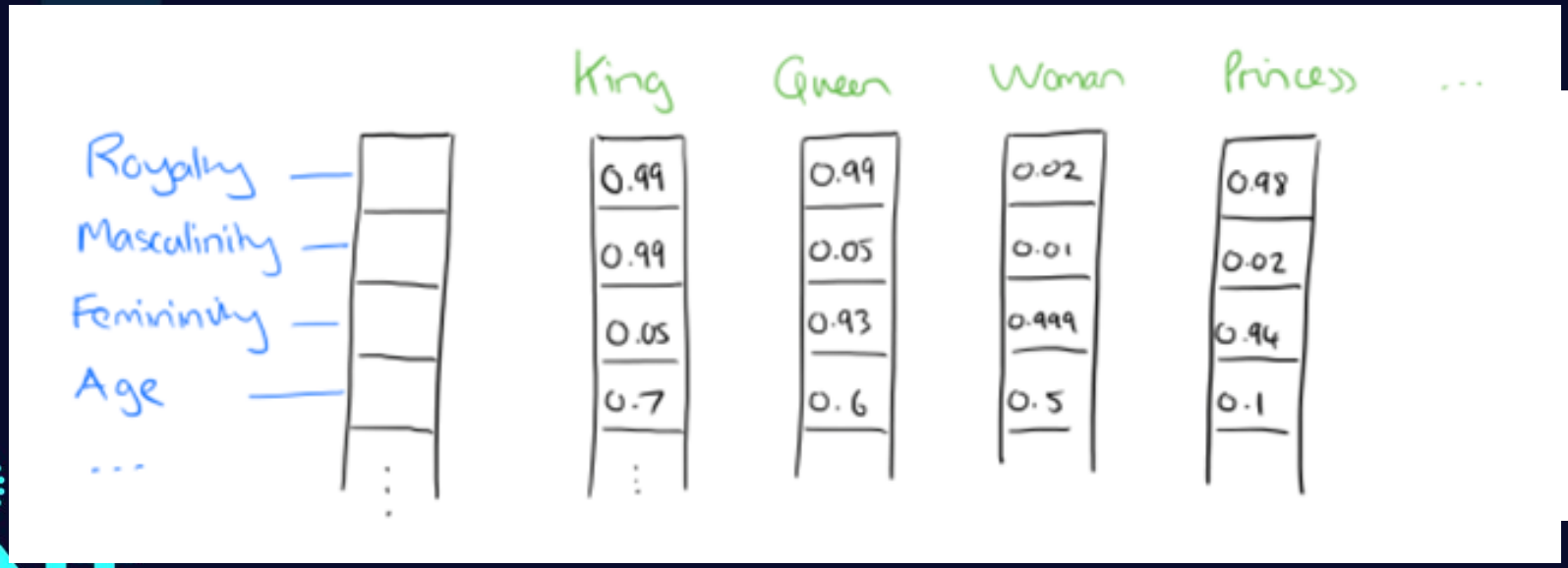
Word Embeddings

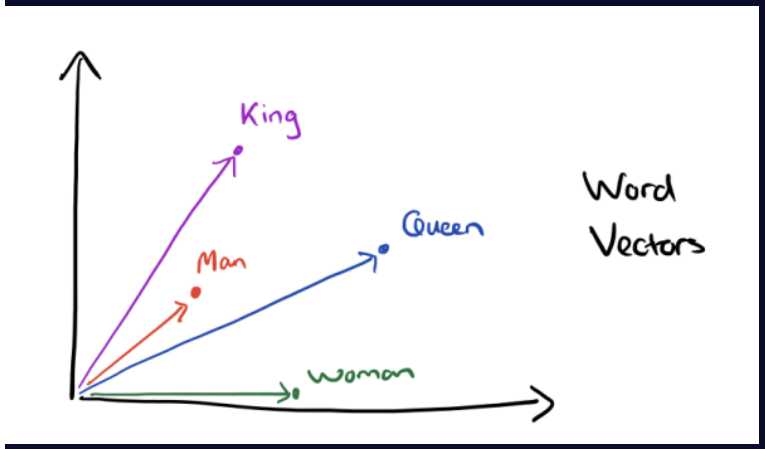


המרכז הבינלאומי
ללימודי הייטק וחדשנות

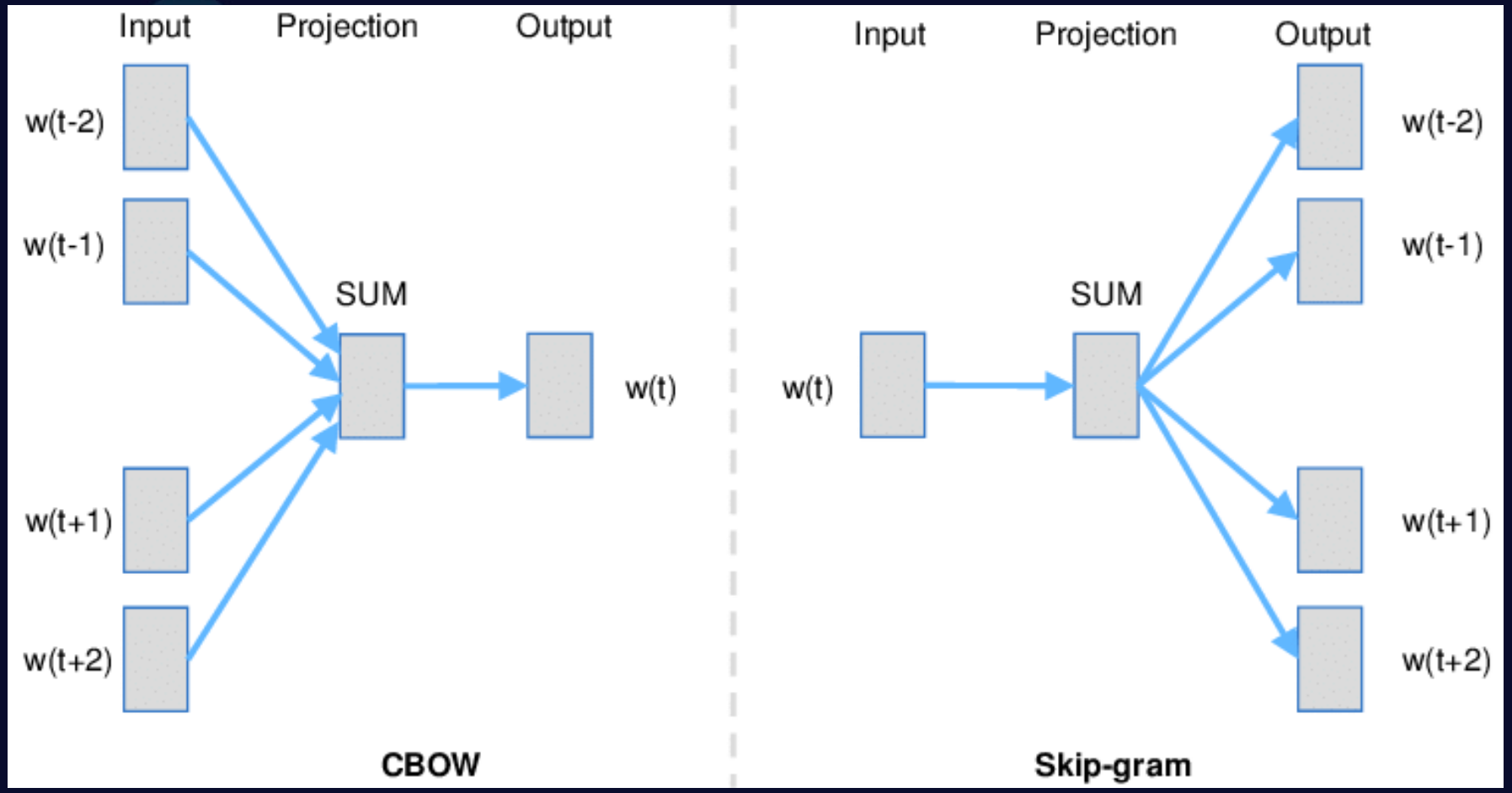
Word Embeddings

- Words are represented by real valued dense vectors of significantly smaller dimensions (e.g. 100 – 1000).

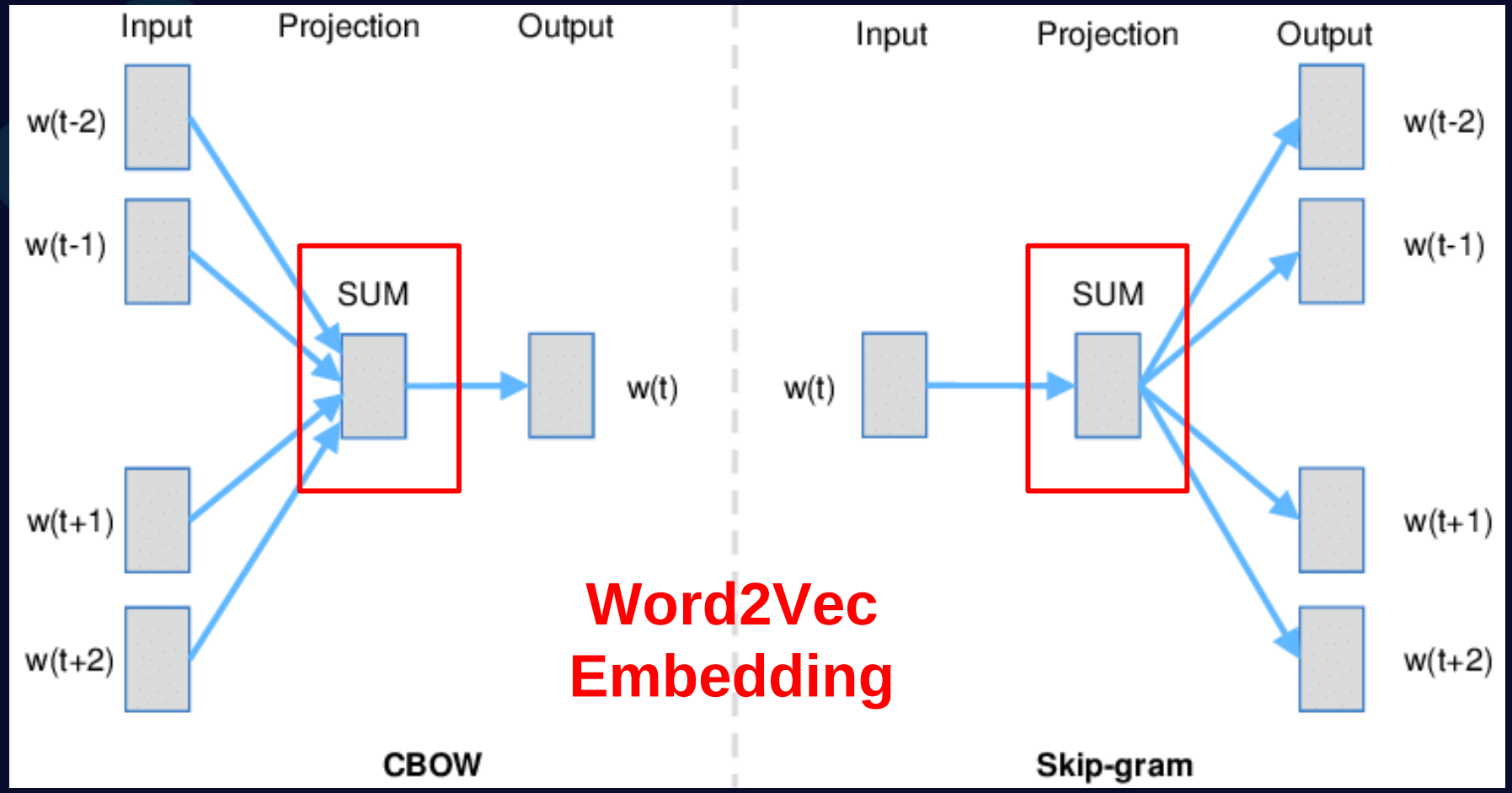




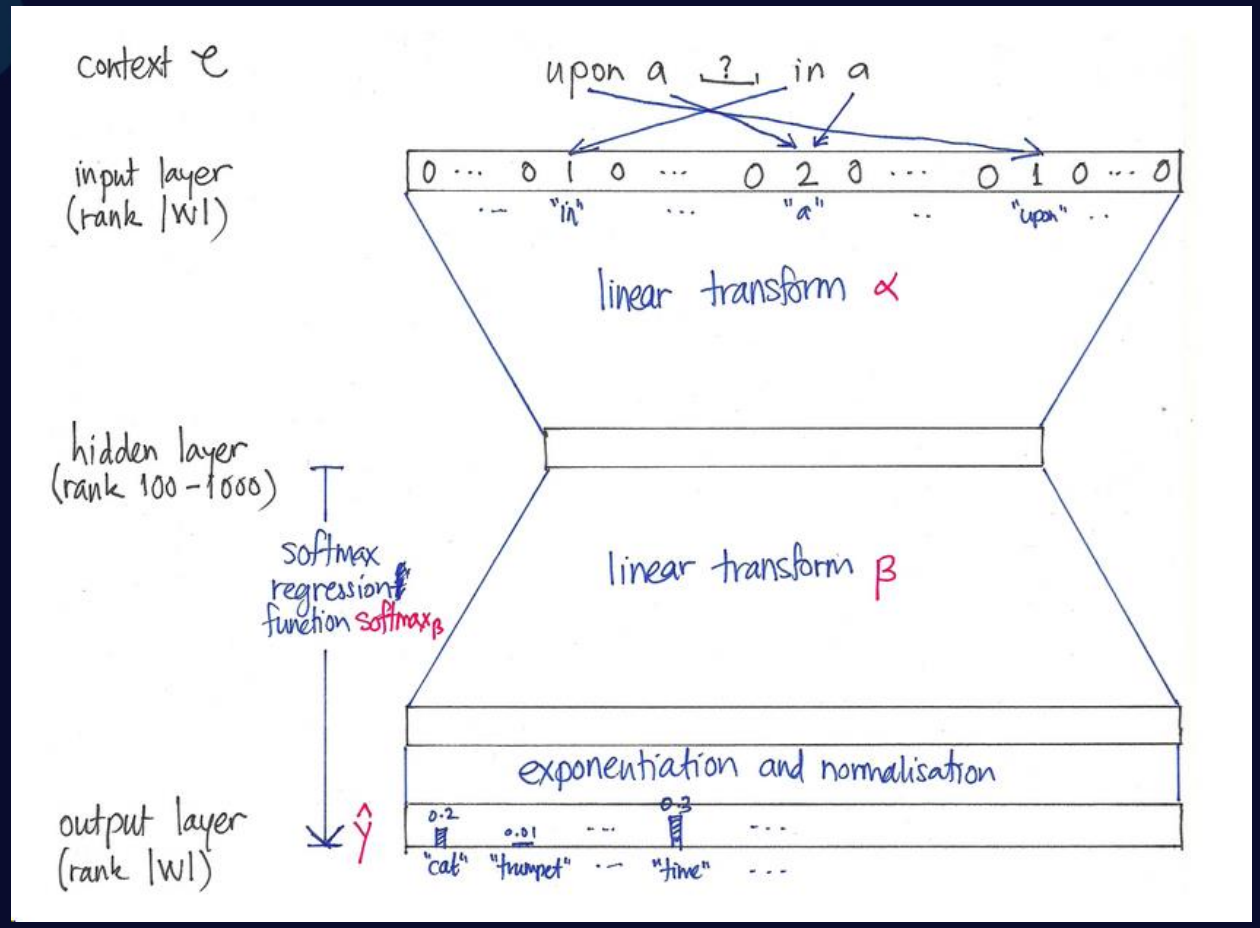
Continues Bag of Words (CBOW) and Skip-Gram



Continues Bag of Words (CBOW) and Skip-Gram

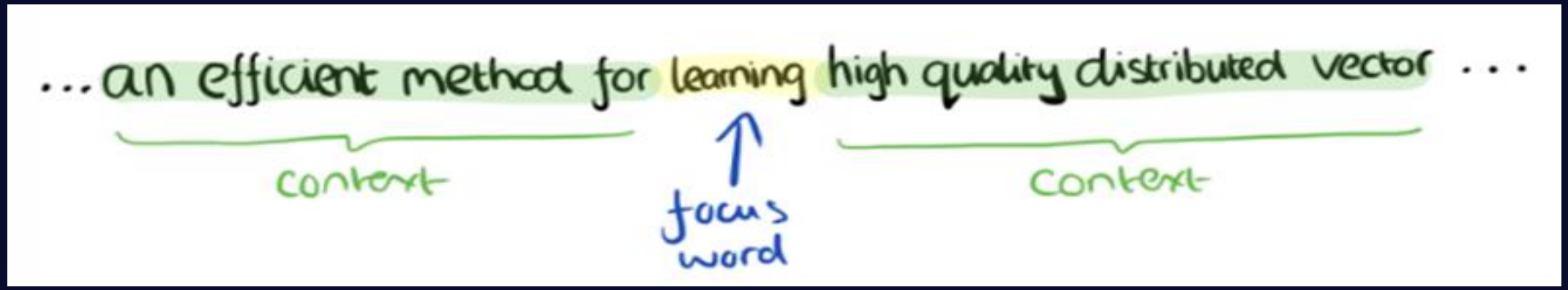


Continues Bag of Words (CBOW)



Continues Bag of Words (CBOW)

- Real value embedding
- Word Vector size: 100-1000 dimensions
- Words appearing together correlate



Loss – Skip Gram

- Skip- Gram maximizes the log - likelihood:

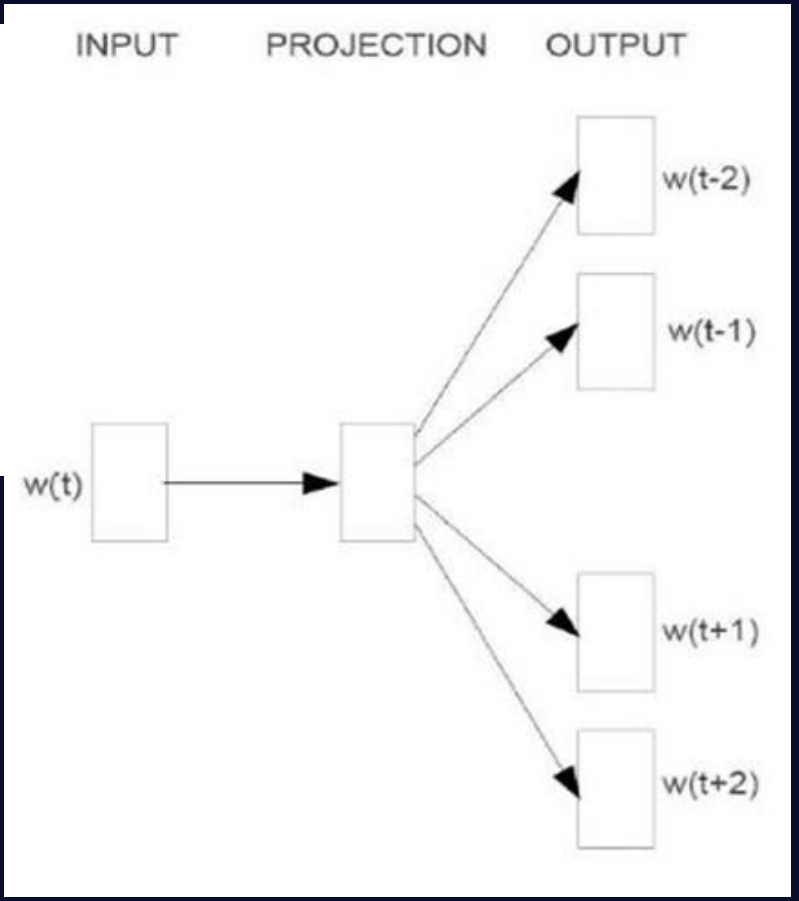
$$\frac{1}{T}\sum_{t=1}^{T}\sum_{-c\leq j\leq c, j\neq 0}\log p(w_{t+j}|w_t)$$

- Where:
  - T – # of words in the corpus.
  - c – unidirectional window size of the context.

Skip Gram

- For example consider the following sentence:
  "If a dog chews shoes, whose shoes does he choose?"

- Input word: *shoes*
- Window size: 2



CBOW Vs Skip-Gram

- Skip-gram: works well with 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

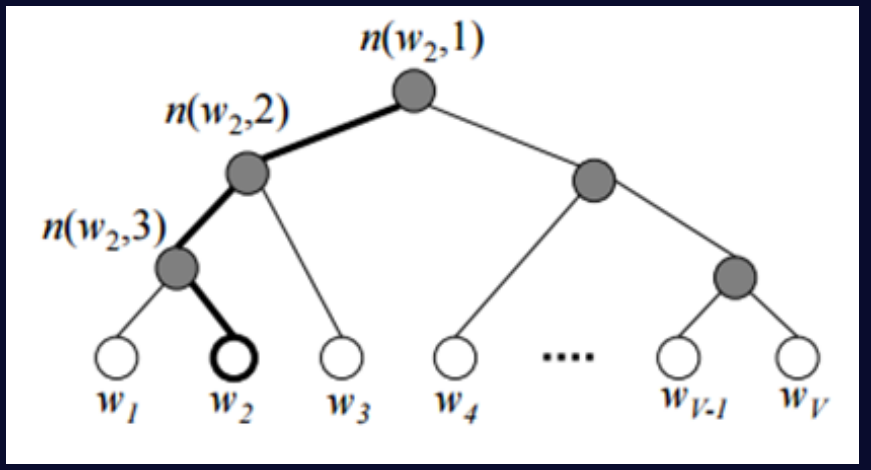
The Softmax Problem

Applying softmax on output layer is expensive!

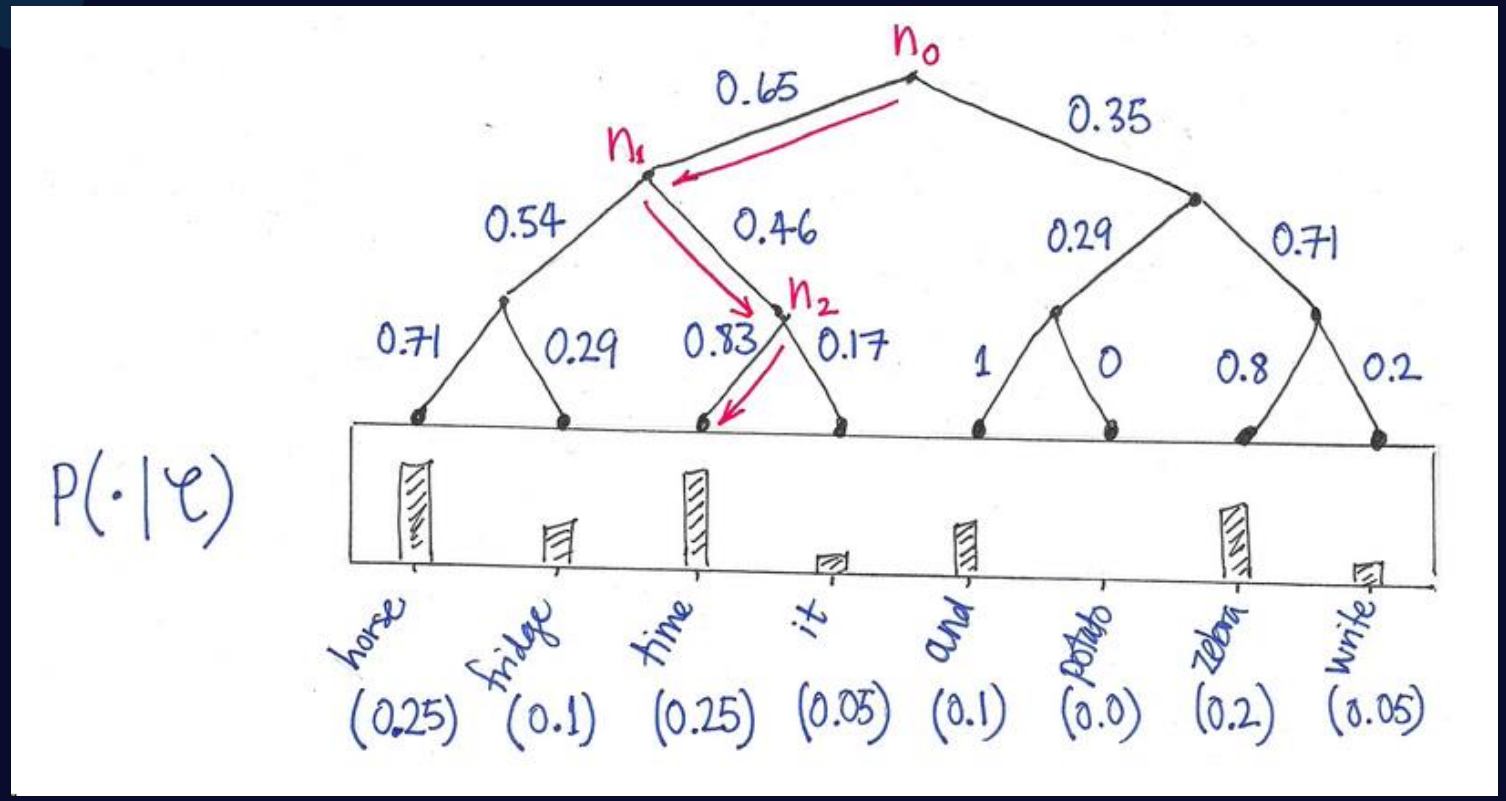
$$p(c|w) = \frac{\exp(v'_c{}^T v_w)}{\sum_{i=1}^{W} \exp(v'_i{}^T v_w)}$$

Possible Solutions:

- Hierarchical Softmax
- Negative Sampling

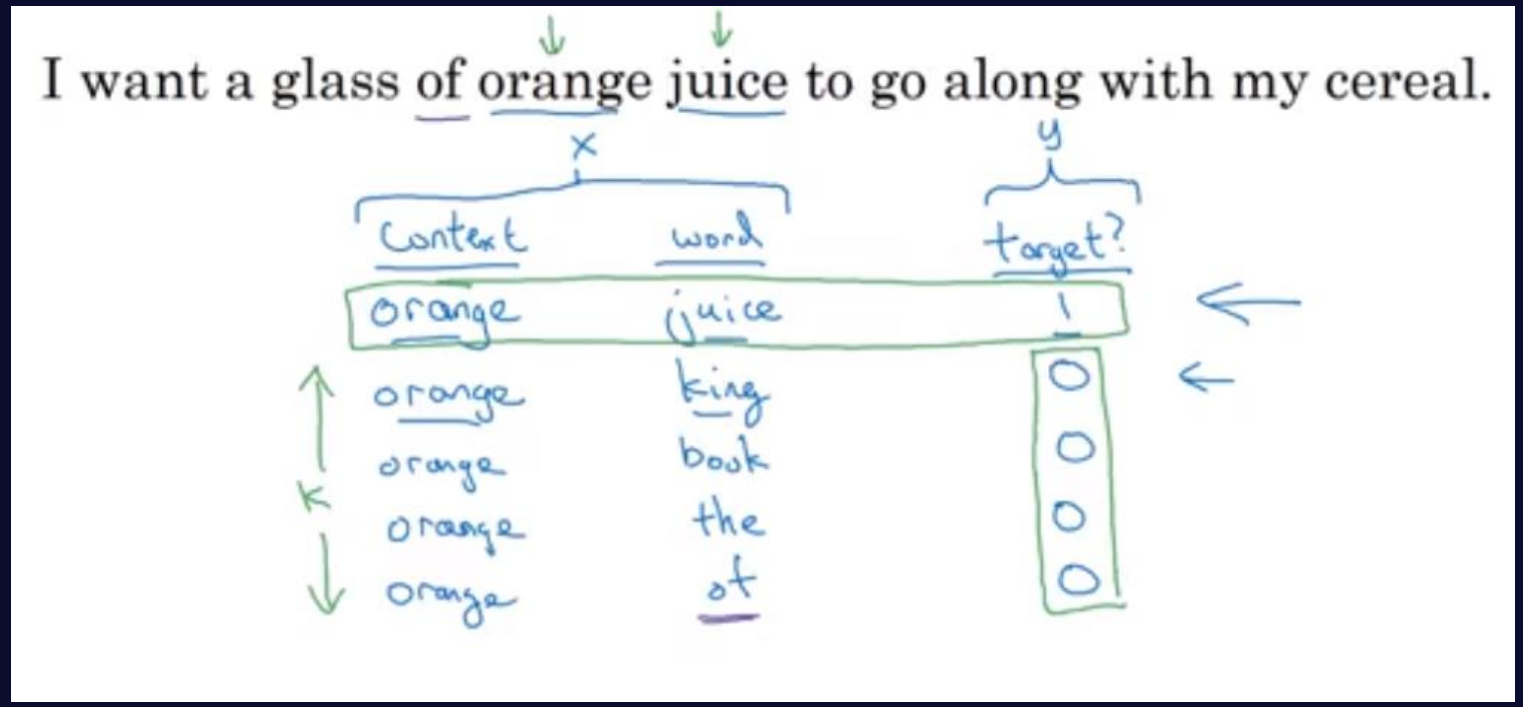


The Softmax Problem



Negative Sampling

Train ~10,000 Binary logistic regression classifiers (one for each output word)



Negative Sampling

Train ~10,000 Binary logistic regression classifiers (one for each output word)

$\mathrm{p}(D = 1|w, c)$ is the probability that $(w, c) \in D$.

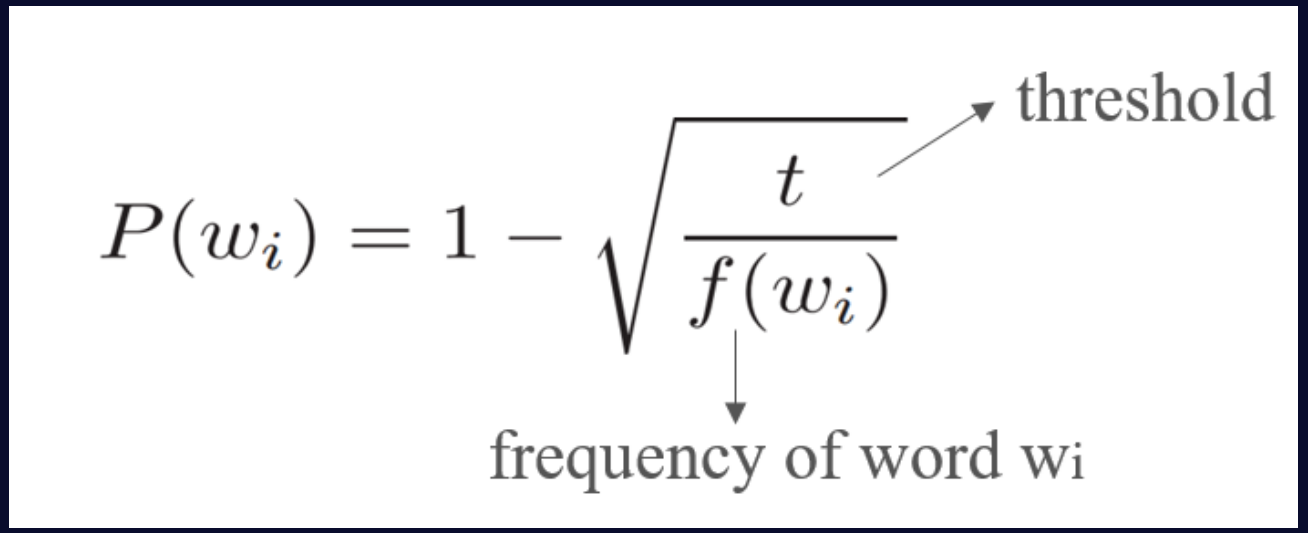
$\mathrm{p}(D = 0|w, c) = 1 - \mathrm{p}(D = 1|w, c)$ is the probability that $(w, c) \notin \mathrm{D}$.

$p(D = 1|w, c)$ must be low $\Rightarrow$ $p(D = 0|w, c)$ will be high.

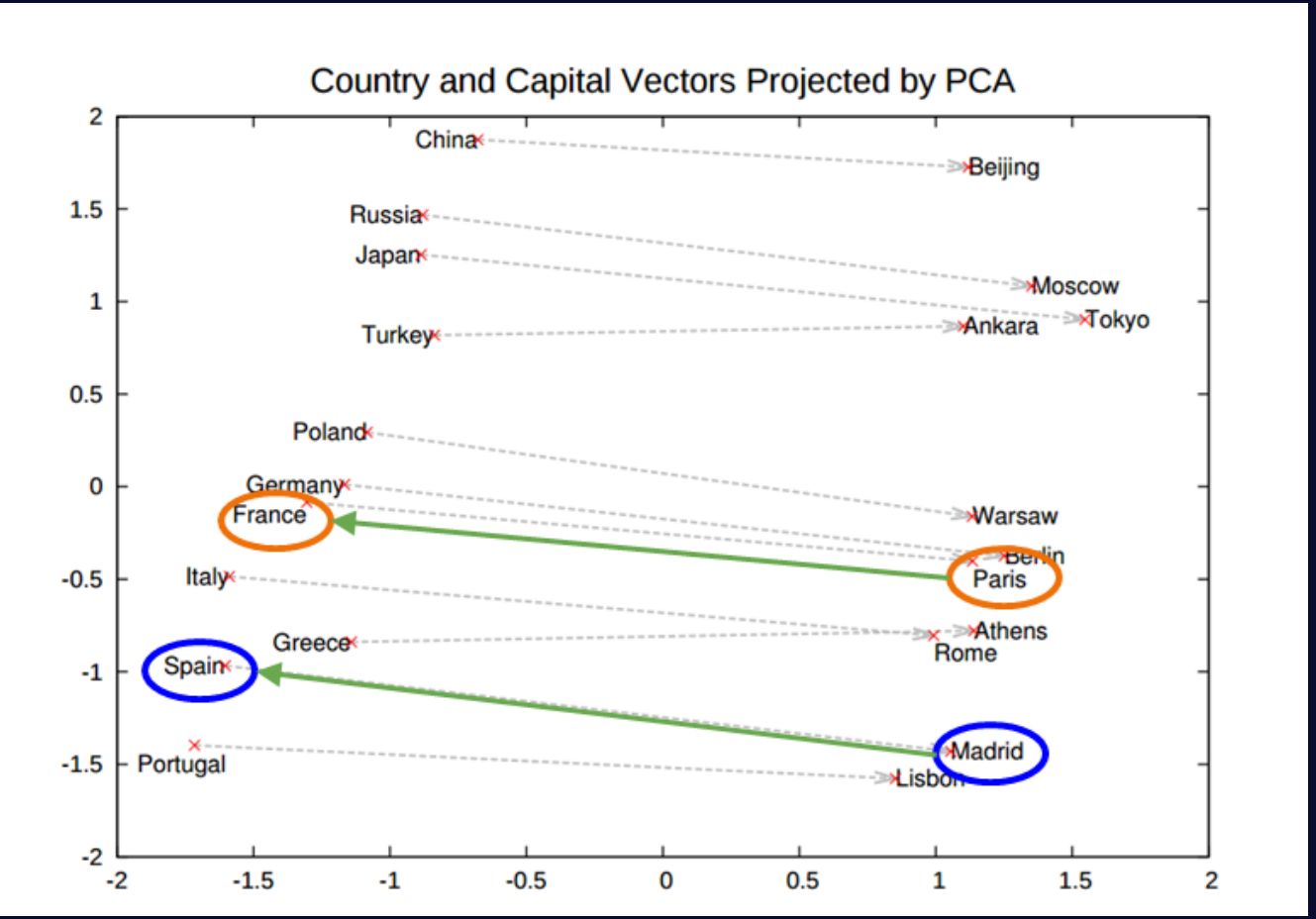
Negative Sampling

Remove frequent words (provide less information)

Practically enlarges the context field of view



Negative Sampling

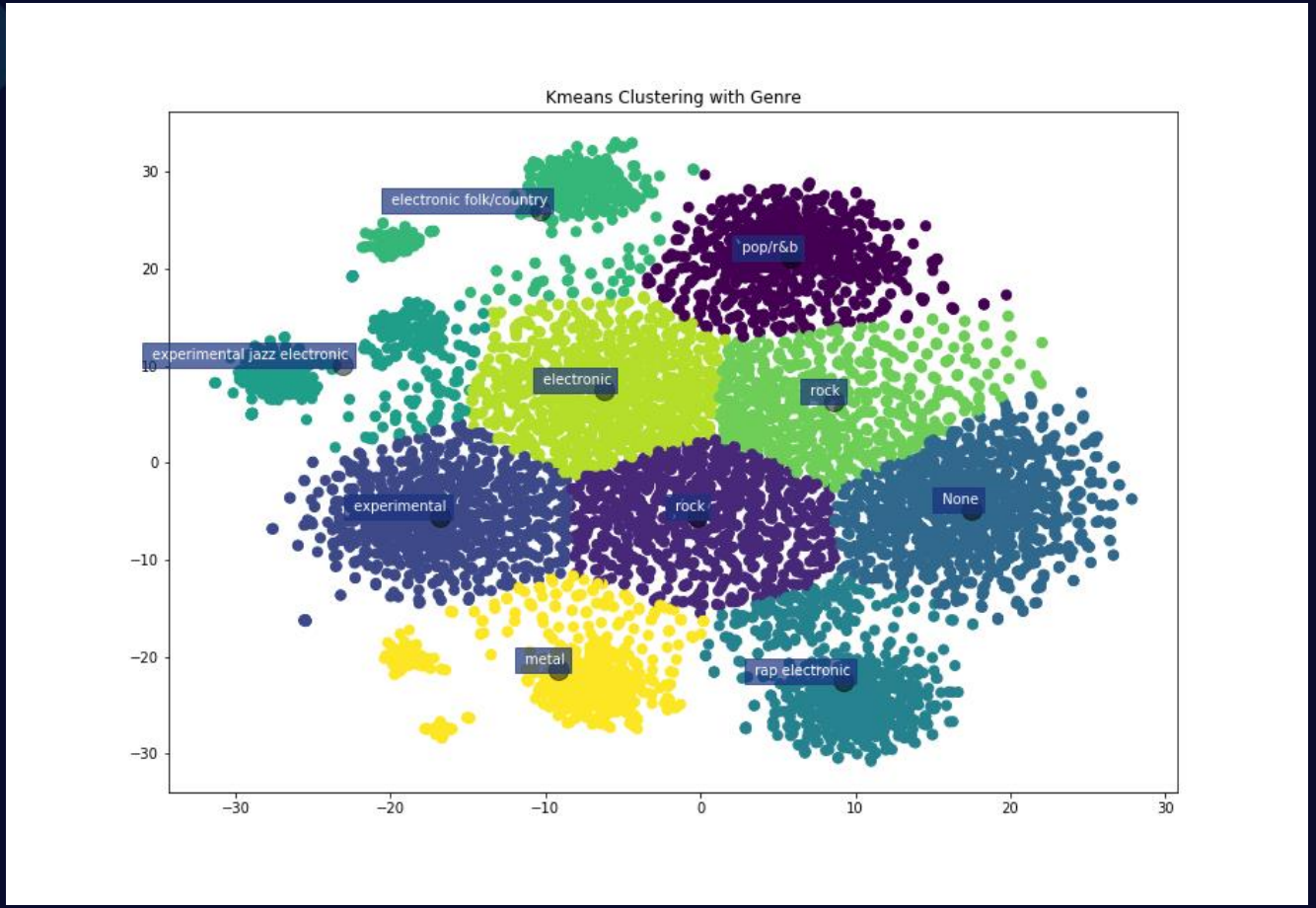


Negative Sampling

German + airlines
airline Lufthansa
carrier Lufthansa
flag carrier Lufthansa
Lufthansa

המרכז הבינלאומי
ללימודי הייטק וחדשנות

Negative Sampling



$$\mathrm{tf}(\text{"this"}, d_1) = \frac{1}{5} = 0.2$$
$$\mathrm{tf}(\text{"this"}, d_2) = \frac{1}{7} \approx 0.14$$

An idf is constant per corpus, and **accounts** for the ratio of documents that include the word "this". In this case, we have a corpus of two documents and all of them include the word "this".

$$\mathrm{idf}(\text{"this"}, D) = \log\left(\frac{2}{2}\right) = 0$$

So tf–idf is zero for the word "this", which implies that the word is not very informative as it appears in all documents.

$$\mathrm{tfidf}(\text{"this"}, d_1) = 0.2 \times 0 = 0$$
$$\mathrm{tfidf}(\text{"this"}, d_2) = 0.14 \times 0 = 0$$

A slightly more interesting example arises from the word "example", which occurs three times but only in the second document:

$$\mathrm{tf}(\text{"example"}, d_1) = \frac{0}{5} = 0$$
$$\mathrm{tf}(\text{"example"}, d_2) = \frac{3}{7} \approx 0.429$$
$$\mathrm{idf}(\text{"example"}, D) = \log\left(\frac{2}{1}\right) = 0.301$$

Document 1

Term	Term Count
this	1
is	1
a	2
sample	1

Document 2

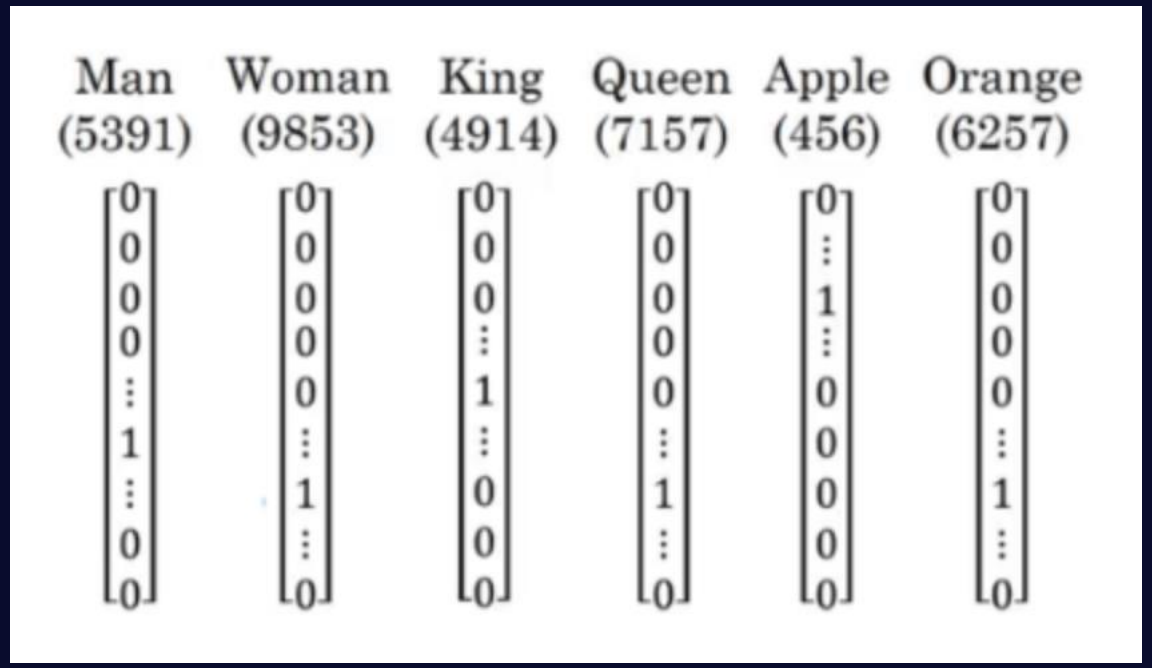
Term	Term Count
this	1
is	1
another	2
example	3

Finally,

$$\mathrm{tfidf}(\text{"example"}, d_1) = \mathrm{tf}(\text{"example"}, d_1) \times \mathrm{idf}(\text{"example"}, D) = 0 \times 0.301 = 0$$
$$\mathrm{tfidf}(\text{"example"}, d_2) = \mathrm{tf}(\text{"example"}, d_2) \times \mathrm{idf}(\text{"example"}, D) = 0.429 \times 0.301 \approx 0.13$$

Classical Word Embeddings

One-hot encoding (sparse vectors)

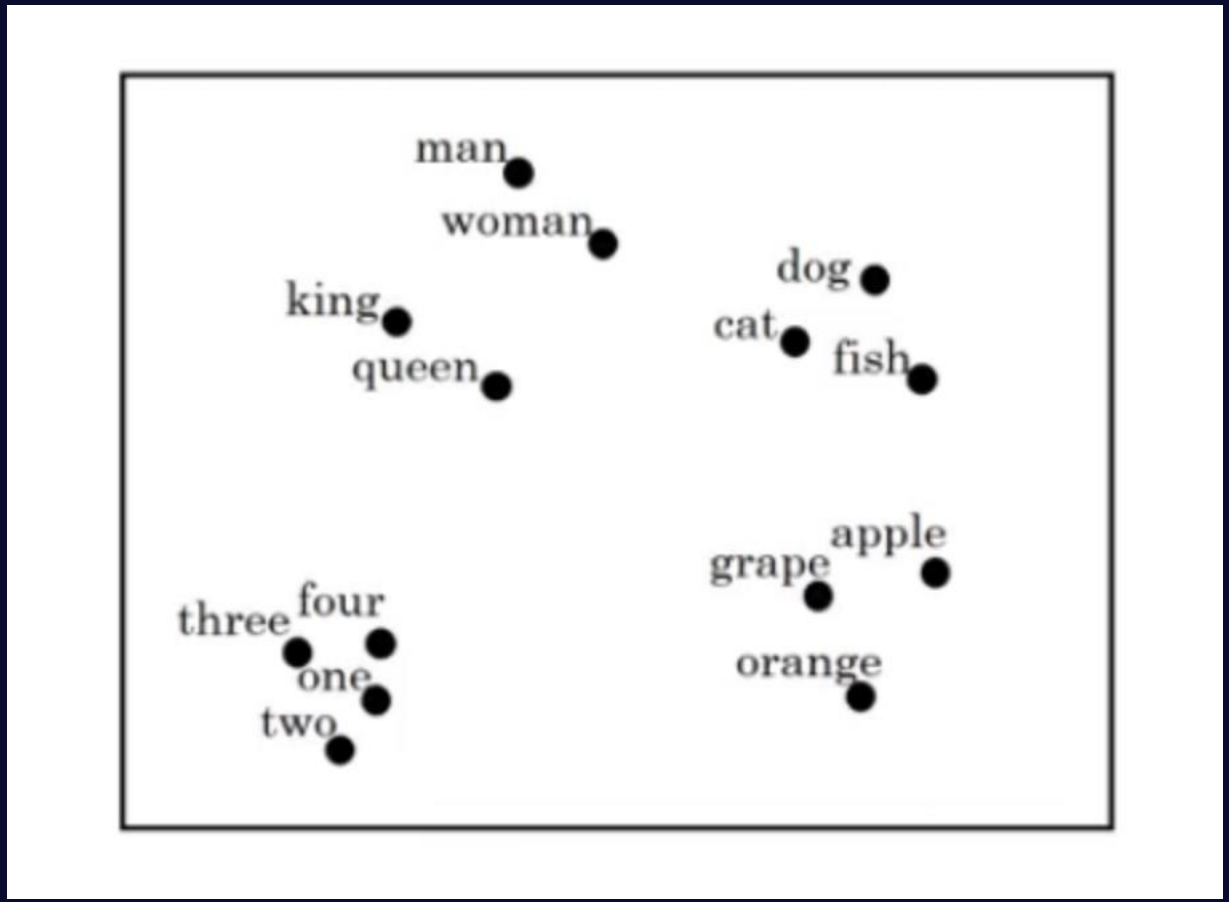


Word embeddings

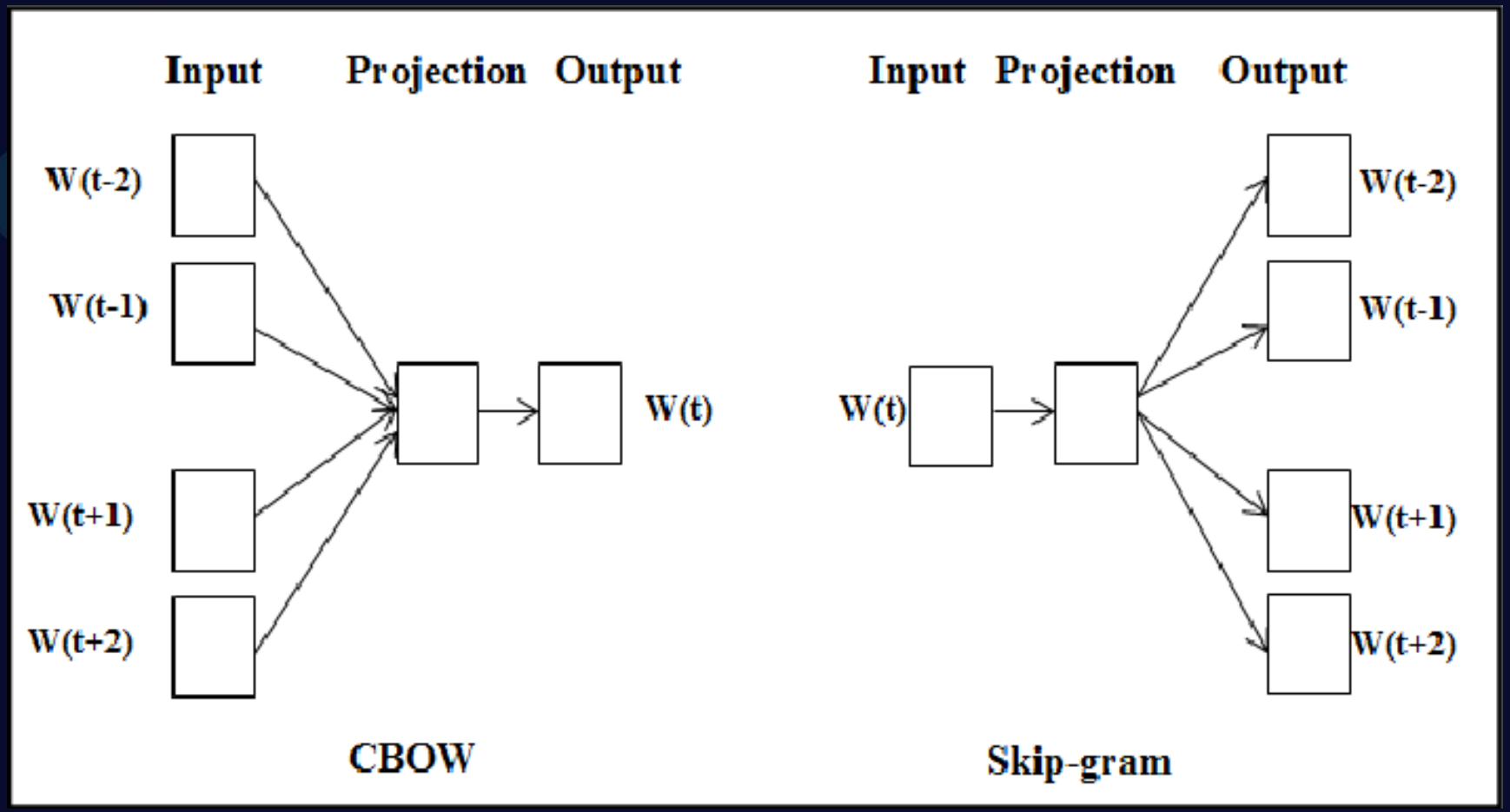
	Man (5391)	Woman (9853)	King (4914)	Queen (7157)	Apple (456)	Orange (6257)
Gender	-1	1	-0.95	0.97	0.00	0.01
Royal	0.01	0.02	0.93	0.95	-0.01	0.00
Age	0.03	0.02	0.7	0.69	0.03	-0.02
Food	0.09	0.01	0.02	0.01	0.95	0.97

INT
המרכז הבינלאומי
ללימודי הייטק וחדשנות

Word embeddings



Word embeddings

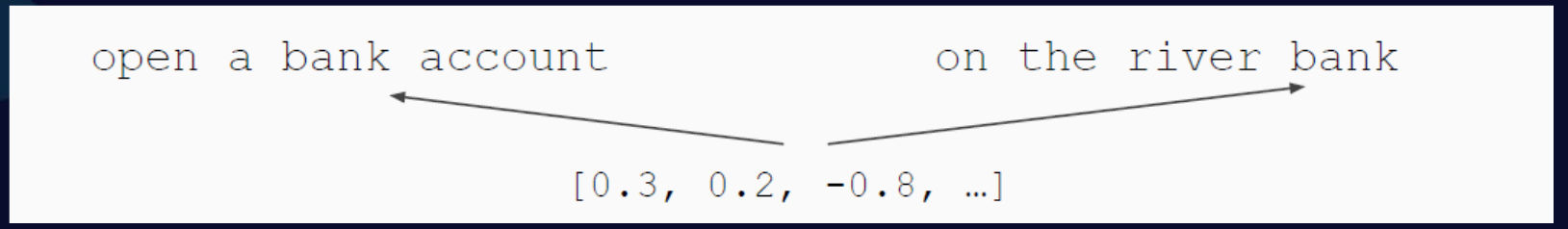


Word embeddings

- Word embeddings are very useful. However they are not perfect.
- Can you think of a problem with fixed word vectors?

Contextual word-Embeddings

- **Problem**: Word embeddings are applied in a context free manner



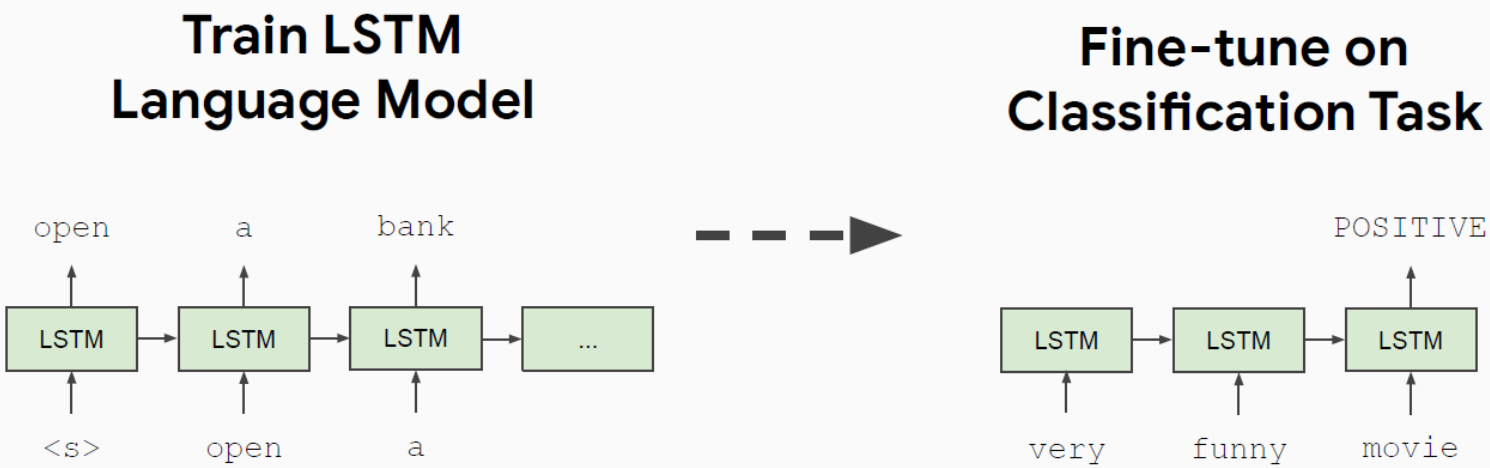
- **Solution**: Train *contextual* representations on text corpus

INT
המרכז הבינלאומי
ללימודי הייטק וחדשנות

Contextual word-Embeddings

- **First attempt:**

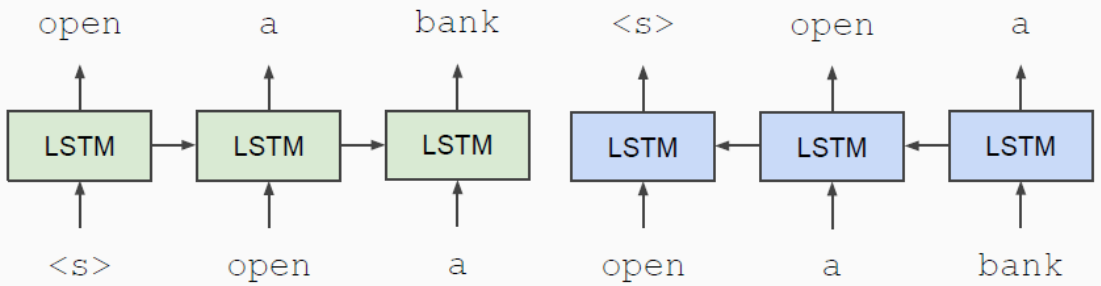
- *Semi-Supervised Sequence Learning*, Google, 2015



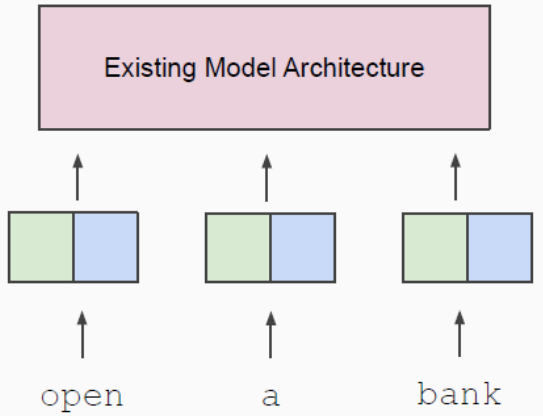
Contextual word-Embeddings

- *ELMo: Deep Contextual Word Embeddings*, AI2 & University of Washington, 2017

Train Separate Left-to-Right and Right-to-Left LMs



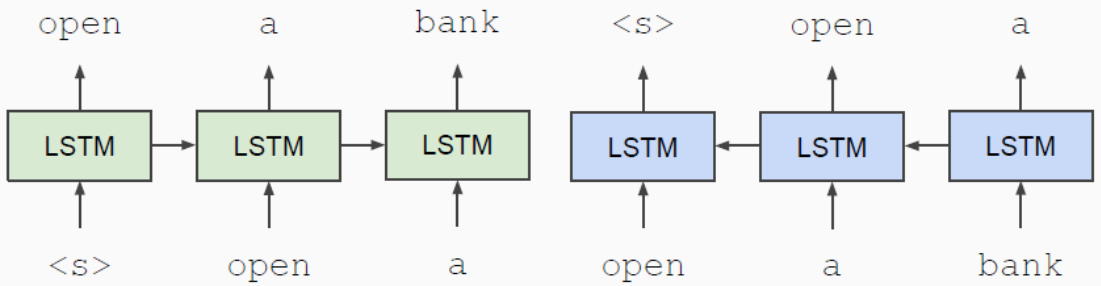
Apply as "Pre-trained Embeddings"



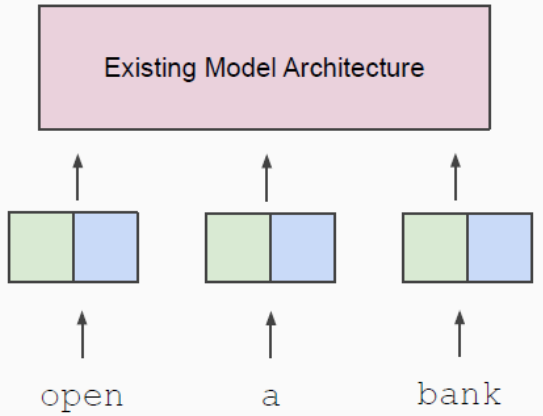
Contextual word-Embeddings

- *ELMo: Deep Contextual Word Embeddings*, AI2 & University of Washington, 2017

Train Separate Left-to-Right and Right-to-Left LMs

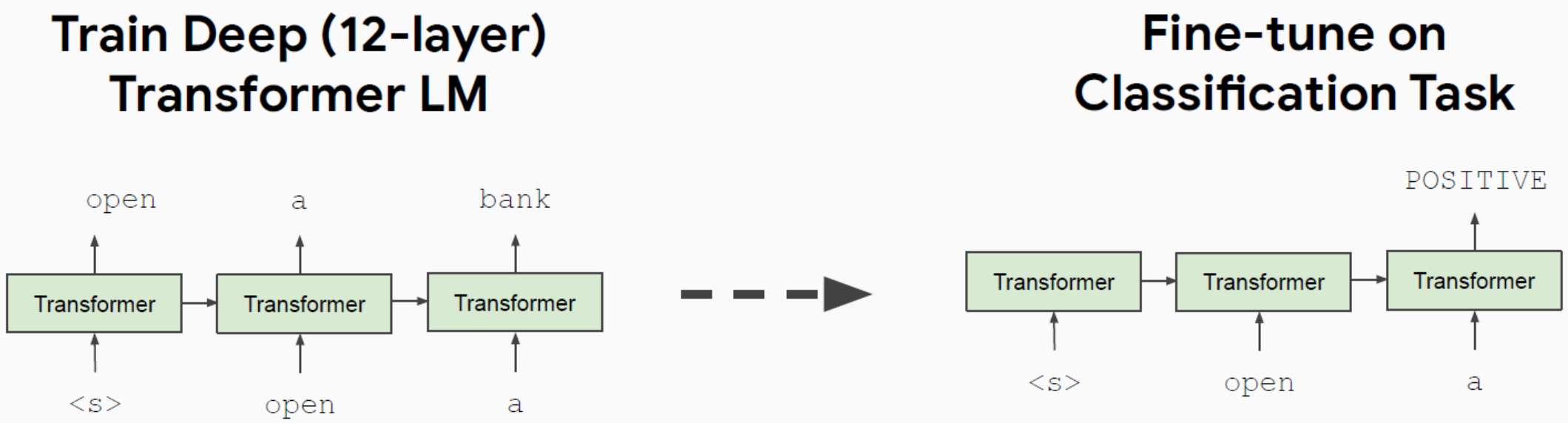


Apply as "Pre-trained Embeddings"



Contextual word-Embeddings

- *Improving Language Understanding by Generative Pre-Training*, OpenAI, 2018



Contextual word-Embeddings

Why is this advance so exciting?

A significant progress in solving the challenge of ambiguous words.

Significant Improvements in down-stream tasks:

1. QA about content in a given paragraph (9% relative error reduction on the SQuAD benchmark)
2. Label the semantic arguments of verbs (16% relative error reduction on the Ontonotes semantic role labeling benchmark)
3. Label expressions in text that refer to people, organizations, and other named entities (4% relative error reduction on the CoNLL 2003 benchmark)
4. Resolve which referring expressions refer to the same entities (10% relative error reduction on the Ontonotes benchmark).

INT
המרכז הבינלאומי
ללימודי הייטק וחדשנות

Contextual word-Embeddings

The Language models that we saw on the first lectures can be used as word embeddings as well. The difference now that every word does not have a fixed vector.

Over the past lectures we learned several types of word embeddings but which one is better? How do we choose between the different options?

Evaluation

Evaluation

- Evaluation is a crucial part in any ML task.
- In many NLP task the evaluation process is not trivial and not decisive.
- Many NLP tasks can be evaluated using a predefined score or using human evaluation.

Evaluation

We can evaluate word embeddings in two of the following ways:

Intrinsic or Extrinsic

Intrinsic Evaluation

Intrinsic evaluation of word vectors is the evaluation of a set of word vectors generated by an embedding technique on specific intermediate subtasks (such as analogy completion).

These subtasks are typically simple and fast to compute and thereby allow us to help understand the system used to generate the word vectors.

Intrinsic Evaluation - Word Vector Analogies

A popular choice for intrinsic evaluation of word vectors is its performance in completing word vector analogies.

In a word vector analogy, we are given an incomplete analogy of the form:

a : b : : c : ?

Israel: Jerusalem::France:

Intrinsic Evaluation - Word Vector Analogies

The intrinsic evaluation system then identifies the word vector which maximizes the cosine similarity:

$$d = \arg\max_i \frac{(x_b - x_a + x_c)^\top x_i}{||x_b - x_a + x_c||}$$

This metric has an intuitive interpretation. Ideally, we want $x_b - x_a = x_d - x_c$ (For instance, queen – king = actress – actor). This implies that we want $x_b - x_a + x_c = x_d$.

Thus we identify the vector x_d which maximizes the normalized dot-product between the two word vectors (i.e. cosine similarity).

Intrinsic Evaluation - Word Vector Analogies

City 1 : State containing City 1 : : City 2 : State containing City 2

Input	Result Produced
Chicago : Illinois : : Houston	Texas
Chicago : Illinois : : Philadelphia	Pennsylvania
Chicago : Illinois : : Phoenix	Arizona
Chicago : Illinois : : Dallas	Texas
Chicago : Illinois : : Jacksonville	Florida
Chicago : Illinois : : Indianapolis	Indiana
Chicago : Illinois : : Austin	Texas
Chicago : Illinois : : Detroit	Michigan
Chicago : Illinois : : Memphis	Tennessee
Chicago : Illinois : : Boston	Massachusetts

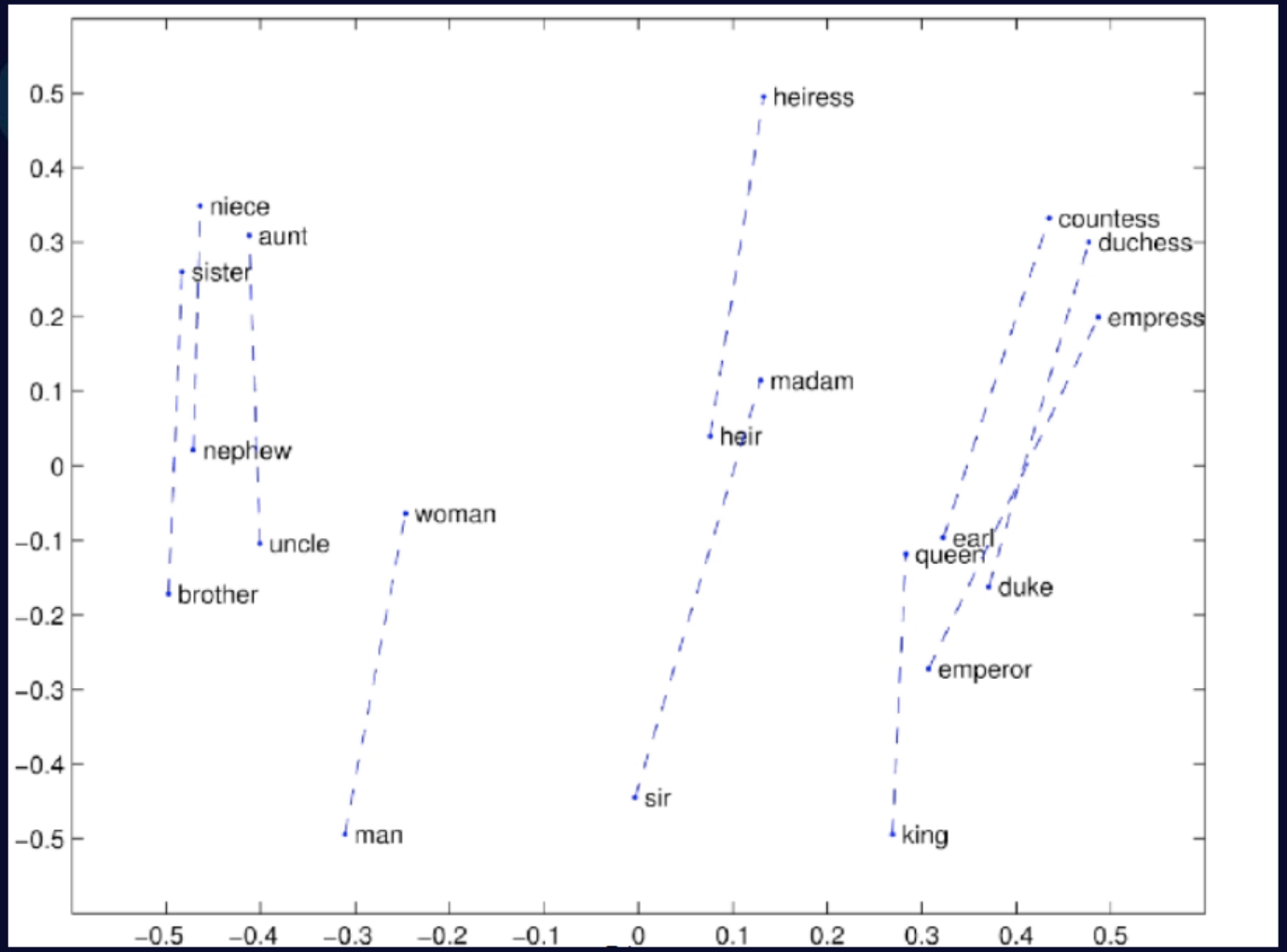
Intrinsic Evaluation - Word Vector Analogies

Input	Result Produced
bad : worst : : big	biggest
bad : worst : : bright	brightest
bad : worst : : cold	coldest
bad : worst : : cool	coolest
bad : worst : : dark	darkest
bad : worst : : easy	easiest
bad : worst : : fast	fastest
bad : worst : : good	best
bad : worst : : great	greatest

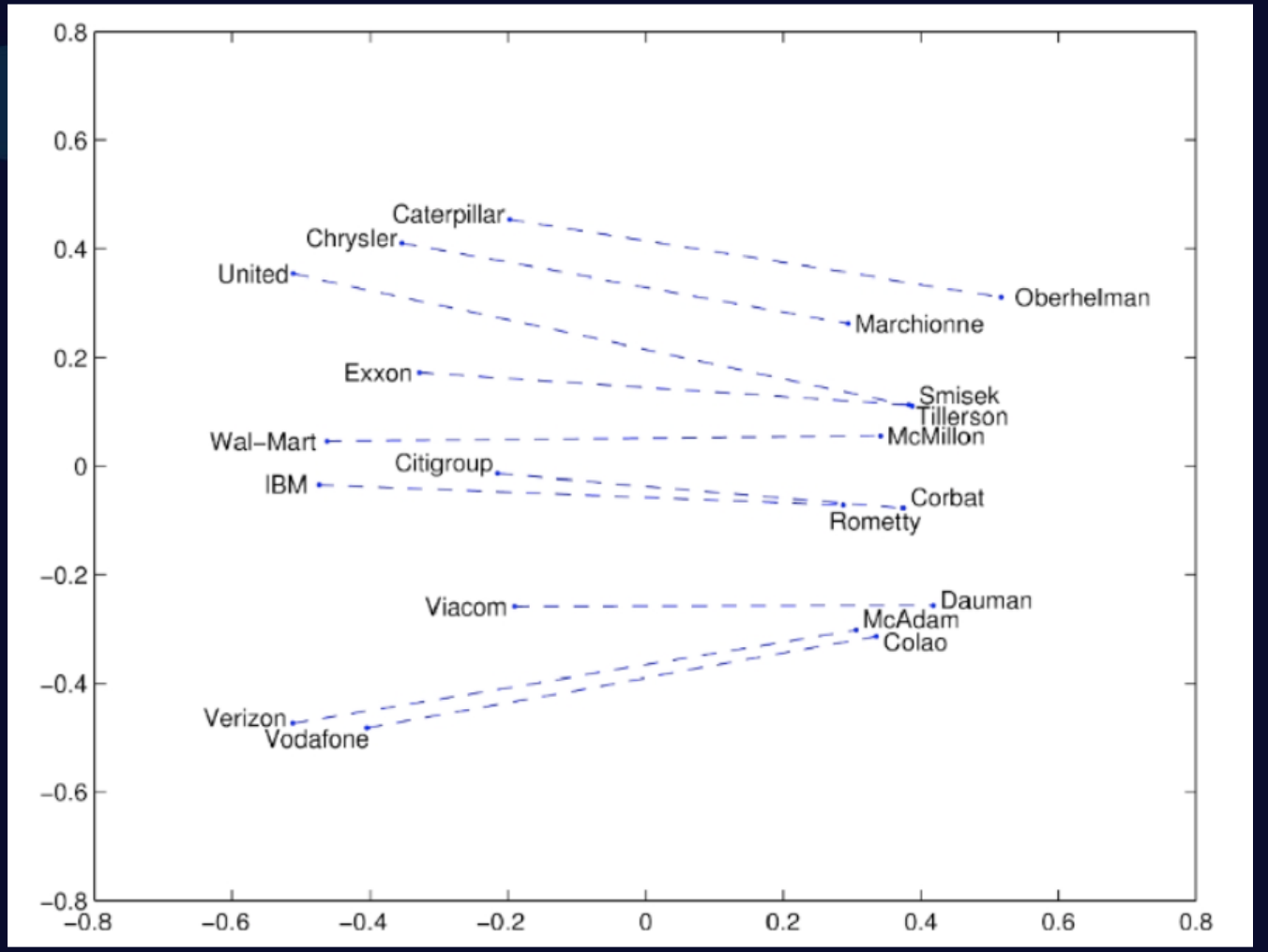
Intrinsic Evaluation - Word Vector Analogies

Input	Result Produced
dancing : danced : : decreasing	decreased
dancing : danced : : describing	described
dancing : danced : : enhancing	enhanced
dancing : danced : : falling	fell
dancing : danced : : feeding	fed
dancing : danced : : flying	flew
dancing : danced : : generating	generated
dancing : danced : : going	went
dancing : danced : : hiding	hid
dancing : danced : : hitting	hit

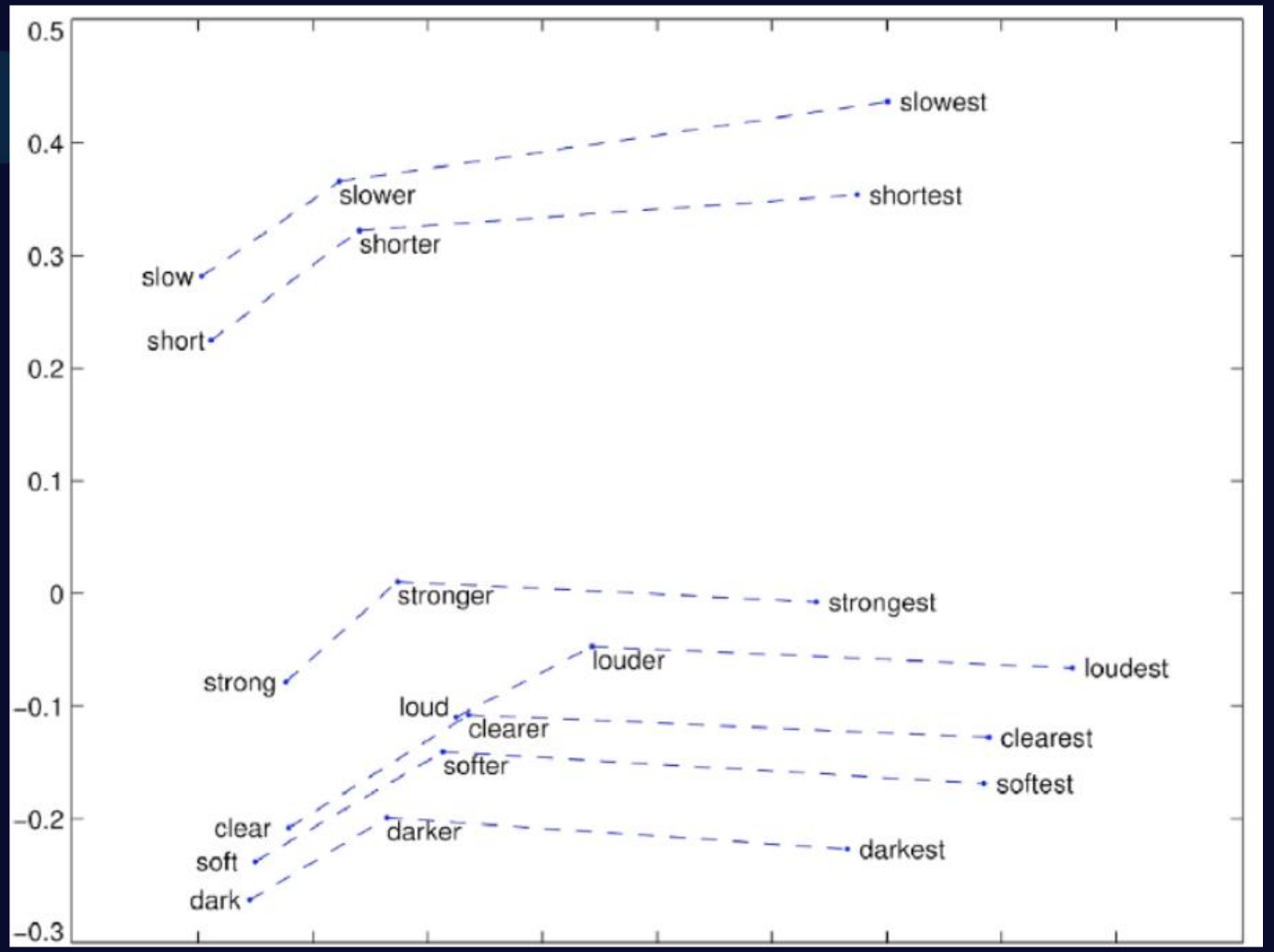
Intrinsic Evaluation



Intrinsic Evaluation



Intrinsic Evaluation



Intrinsic Evaluation Correlation Evaluation

Another simple way to evaluate the quality of word vectors is by asking humans to assess the similarity between two words on a fixed scale (say 0-10) and then comparing this with the cosine similarity between the corresponding word vectors.

This has been done on various datasets that contain human judgement survey data.

Intrinsic Evaluation

word 1	word 2	human judgement
tiger	cat	7.35
book	paper	7.46
computer	internet	7.58
plane	car	5.77
stock	phone	1.62
stock	CD	1.31
stock	jaguar	0.92

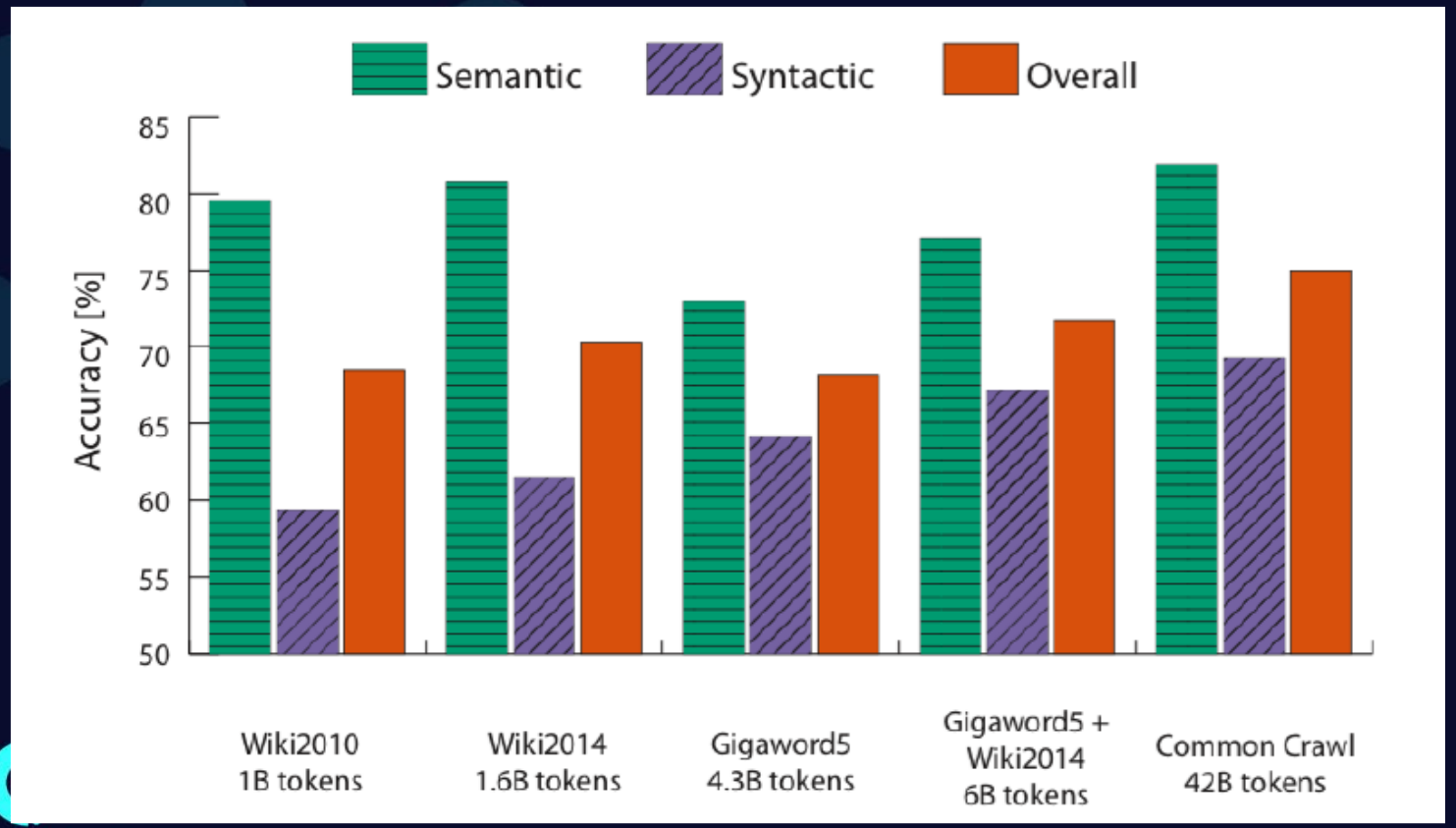
Intrinsic Evaluation Correlation Evaluation

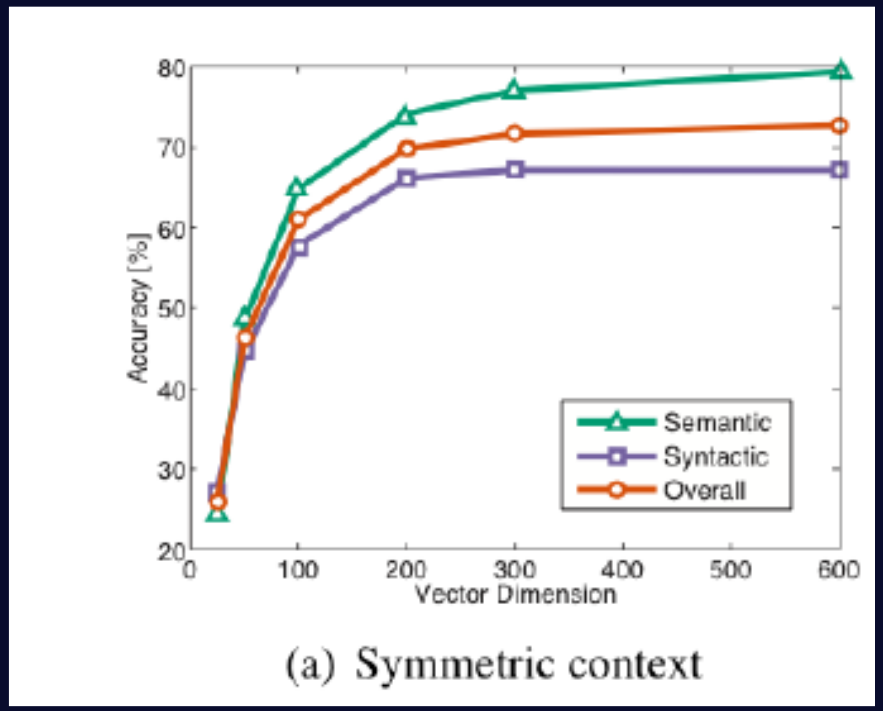
Model	Size	WS353	MC	RG	SCWS	RW
SVD	6B	35.3	35.1	42.5	38.3	25.6
SVD-S	6B	56.5	71.5	71.0	53.6	34.7
SVD-L	6B	65.7	72.7	75.1	56.5	37.0
CBOW	6B	57.2	65.6	68.2	57.0	32.5
SG	6B	62.8	65.2	69.7	58.1	37.2
GloVe	6B	65.8	72.7	77.8	53.9	38.1
SVD-L	42B	74.0	76.4	74.1	58.3	39.9
GloVe	42B	75.9	83.6	82.9	59.6	47.8
CBOW	100B	68.4	79.6	75.4	59.4	45.5

Intrinsic Evaluation

- Performance is heavily dependent on the model used for word embedding
- Performance increases with larger corpus sizes
- Performance is lower for extremely low as well as for extremely high dimensional word vectors

Intrinsic Evaluation - Word Vector Analogies





(a) Symmetric context

Extrinsic evaluation

We have so far focused on intrinsic tasks and emphasized their importance in developing a good word embedding technique.

Of course, the end goal of most real-world problems is to use the resulting word vectors for some other extrinsic task.

Extrinsic evaluation of word vectors is the evaluation of a set of word vectors generated by an embedding technique on the real task at hand.

Extrinsic evaluation

Task: named entity recognition. Find mentions of person, location, organization in text.

Model	Dev	Test	ACE	MUC7
Discrete	91.0	85.4	77.4	73.4
SVD	90.8	85.7	77.3	73.7
SVD-S	91.0	85.5	77.6	74.3
SVD-L	90.5	84.8	73.6	71.5
HPCA	92.6	**88.7**	81.7	80.7
HSMN	90.5	85.7	78.7	74.7
CW	92.2	87.4	81.7	80.2
CBOW	93.1	88.2	82.2	81.1
GloVe	**93.2**	88.3	**82.9**	**82.2**

Extrinsic evaluation – GLUE / SUPERGLUE

- The General Language Understanding Evaluation (GLUE): A benchmark of nine sentence- or sentence-pair language understanding tasks built on established existing datasets and selected to cover a diverse range of dataset sizes, text genres, and degrees of difficulty.

- Super-GLUE

Caution

- Word vectors are biased -

  Like any engineered artifact, a computer program is likely to reflect the perspective of its builders. Computer programs that are built from data will reflect what's in the data—in our case, a text corpus.

  If the text corpus signals associations between concepts that reflect cultural biases, these associations should be expected to persist in the word vectors and any system that uses them.

Caution

- Language is a lot more than words -

  Effective understanding and production of language is about more than knowing word meanings; it requires knowing how words are put together to form more complicated concepts, propositions, and more.

Caution

- Natural language processing is not a single problem -

  It is important to remember that existing benchmarks reflect only a handful of benchmarks that have emerged in the research community.

  These benchmarks are, to varying degrees, controversial, and are always subject to debate. No one who has spent any serious amount of time studying NLP believes they are “complete” in any interesting sense.

  NLP can only make progress if we have ways of objectively measuring progress, but we also need continued progress on the design of the benchmarks and the scores we use for comparisons.

Sentence Embeddings

Sentence Embeddings

- We now move from word embedding to sentence embedding.

- We will discuss two issues:
  - Evaluation
  - Models

Sentence Embeddings

- Sentence similarity
  - Two sentences With Similar Meanings Should Have High Cosine similarities
  - Correlation between algorithm and human judgments.

- Text classification
  - Use sentence embedding for classification and take average over several tasks.

Sentence Embeddings

a man with a jersey is dunking the ball at a basketball game

the ball is being dunked by a man with a jersey at a basketball game

---

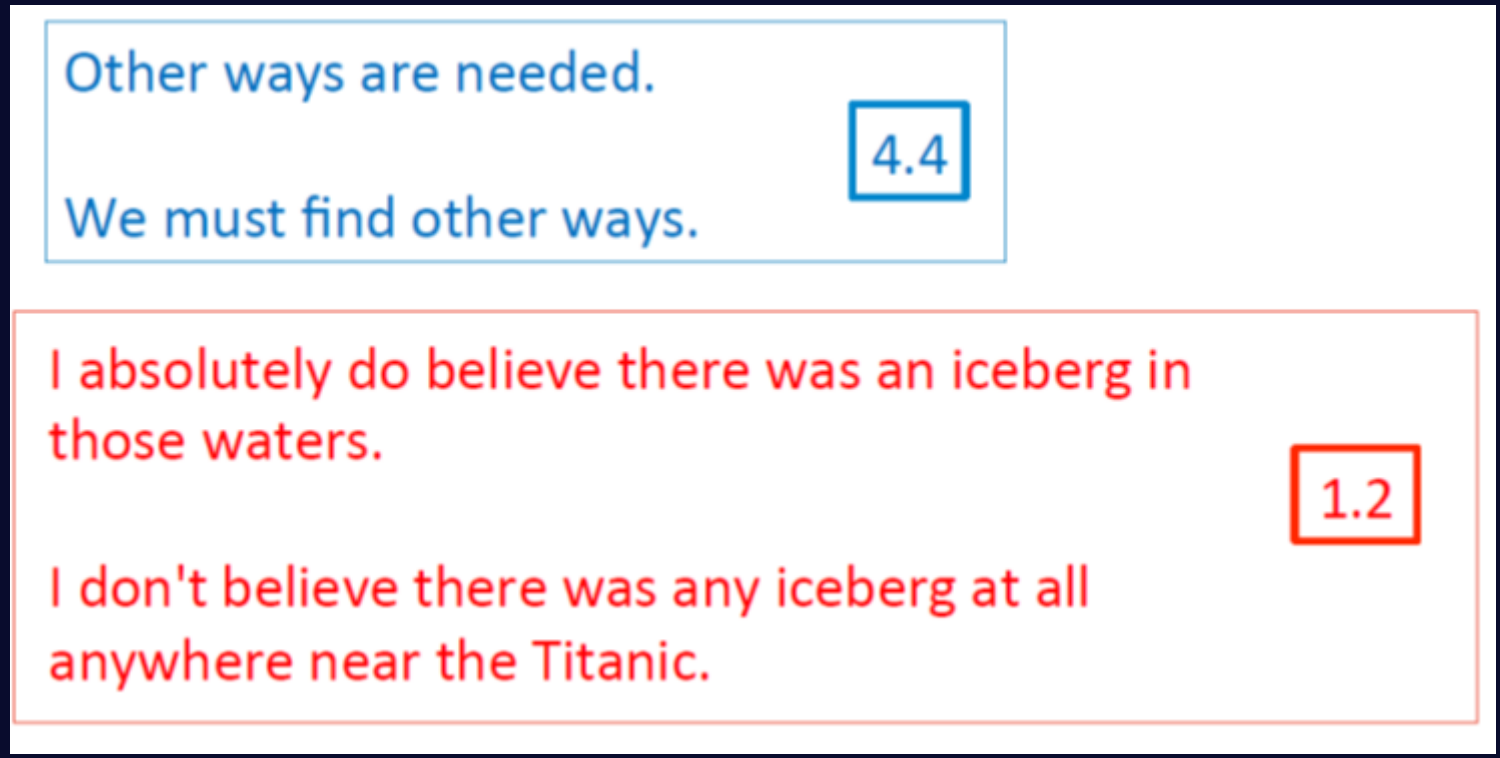
people wearing costumes are gathering in a forest and are looking in the same direction

a little girl in costume looks like a woman

5	*The two sentences are completely equivalent, as they mean the same thing.*
	The bird is bathing in the sink. Birdie is washing itself in the water basin.
4	*The two sentences are mostly equivalent, but some unimportant details differ.*
	Two boys on a couch are playing video games. Two boys are playing a video game.
3	*The two sentences are roughly equivalent, but some important information differs/missing.*
	John said he is considered a witness but not a suspect. “He is not a suspect anymore.” John said.
2	*The two sentences are not equivalent, but share some details.*
	They flew out of the nest in groups. They flew into the nest together.
1	*The two sentences are not equivalent, but are on the same topic.*
	The woman is playing the violin. The young lady enjoys listening to the guitar.
0	*The two sentences are completely dissimilar.*
	The black dog is running through the snow. A race car driver is driving his car through the mud.

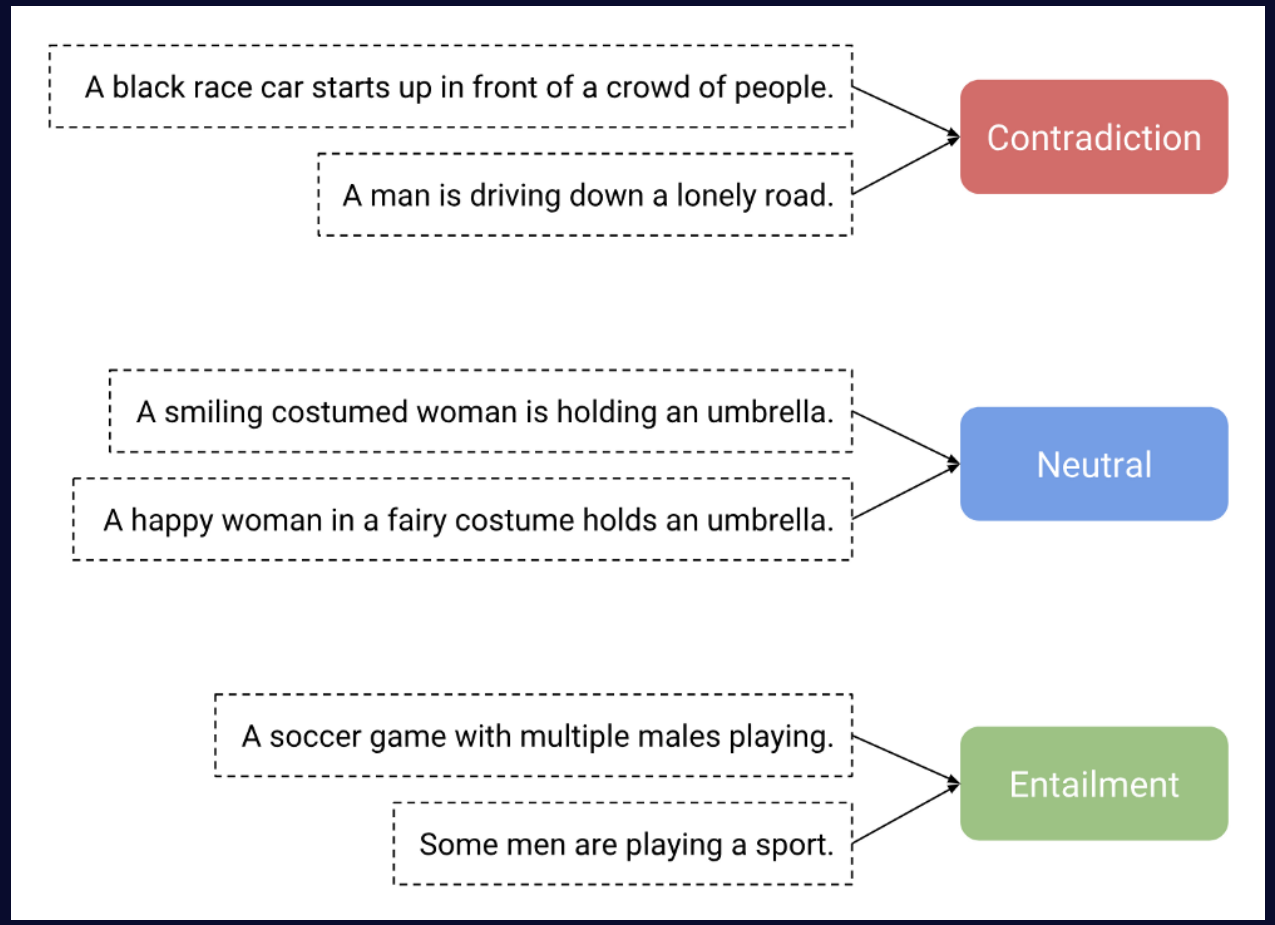
Sentence Embeddings

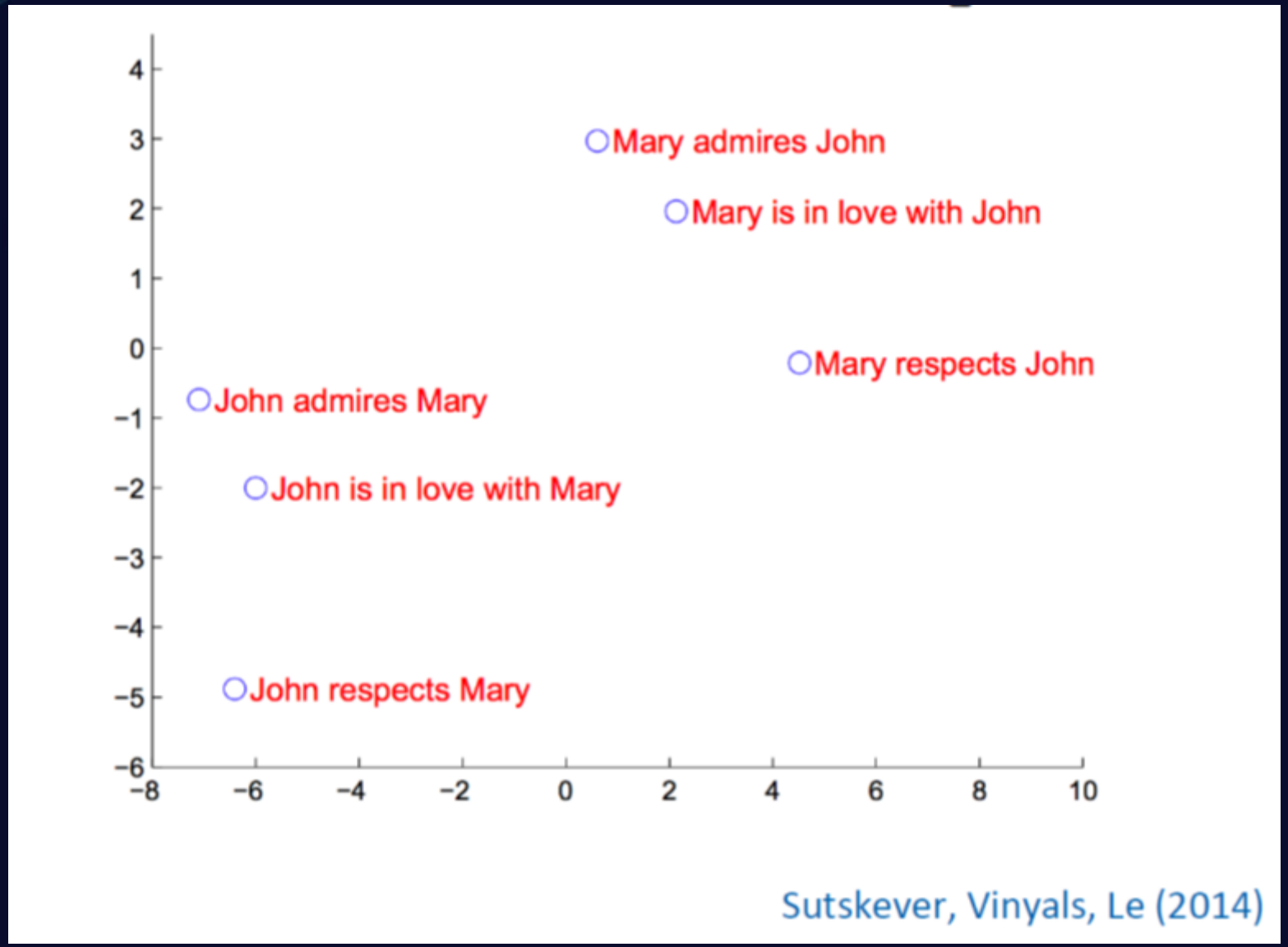
- Semantic Textual Similarity (STS)



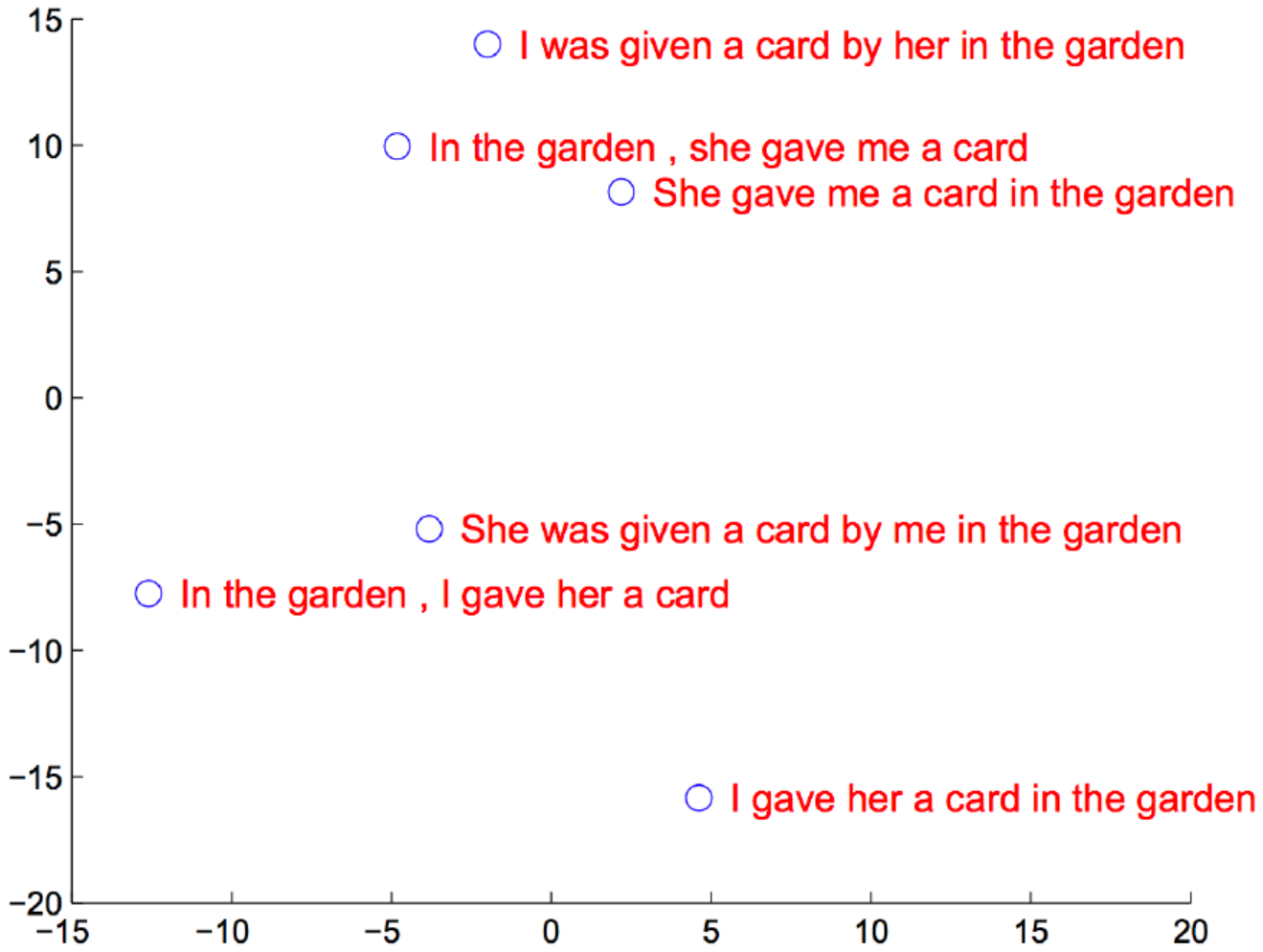
INT
המרכז הבינלאומי
ללימודי הייטק וחדשנות

Sentence Embeddings





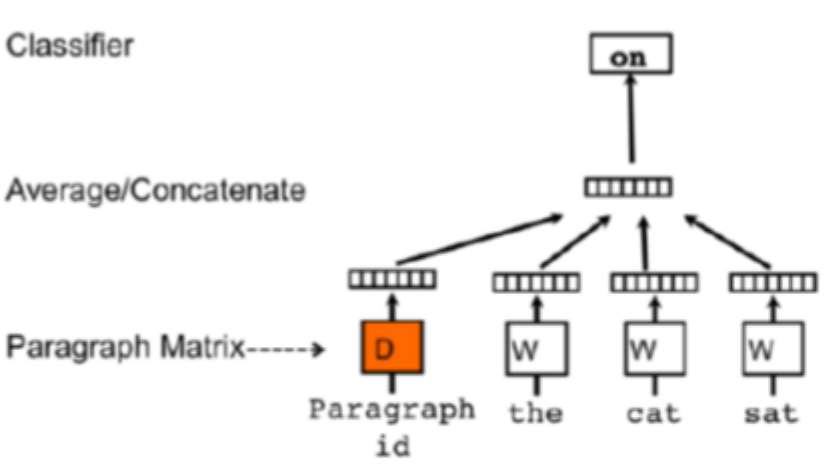
Sutskever, Vinyals, Le (2014)

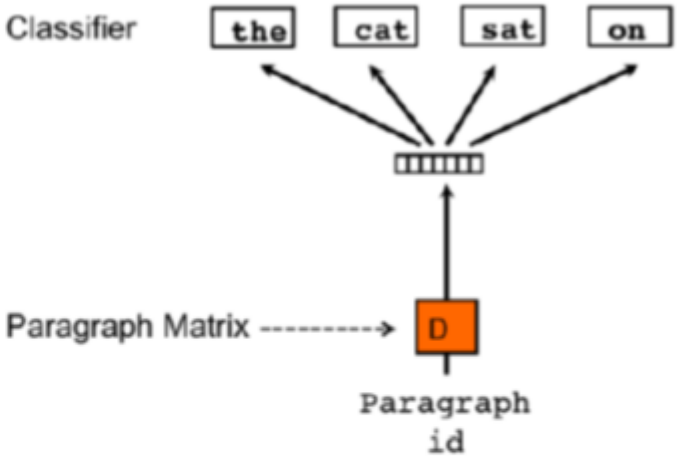


Sutskever, Vinyals, Le (2014)

Sentence Embeddings – Paragraph Vectors

- Represent sentence by predicting its own words or context words.



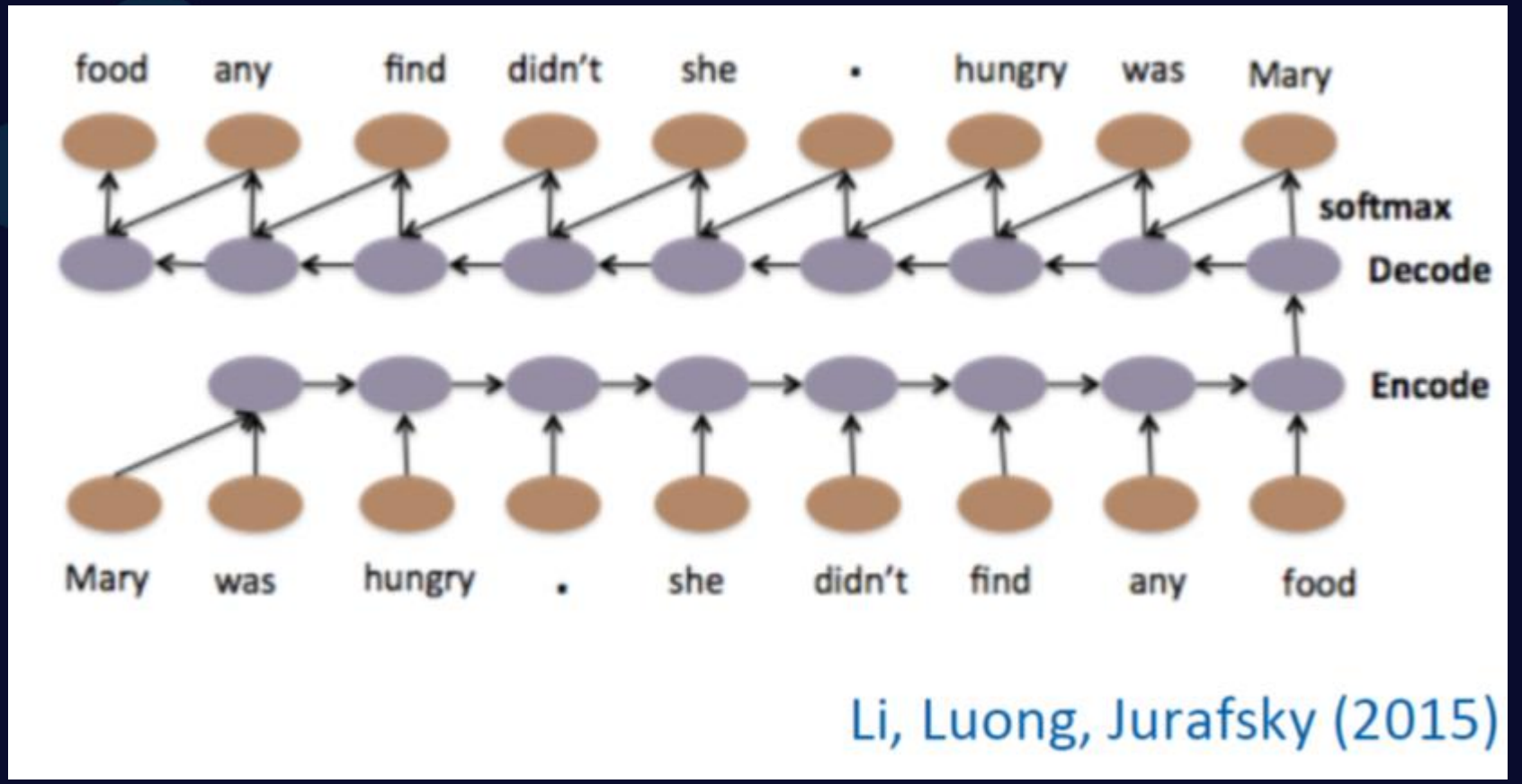


Le & Mikolov (2014)

Sentence Embeddings – Sentence Autoencoders

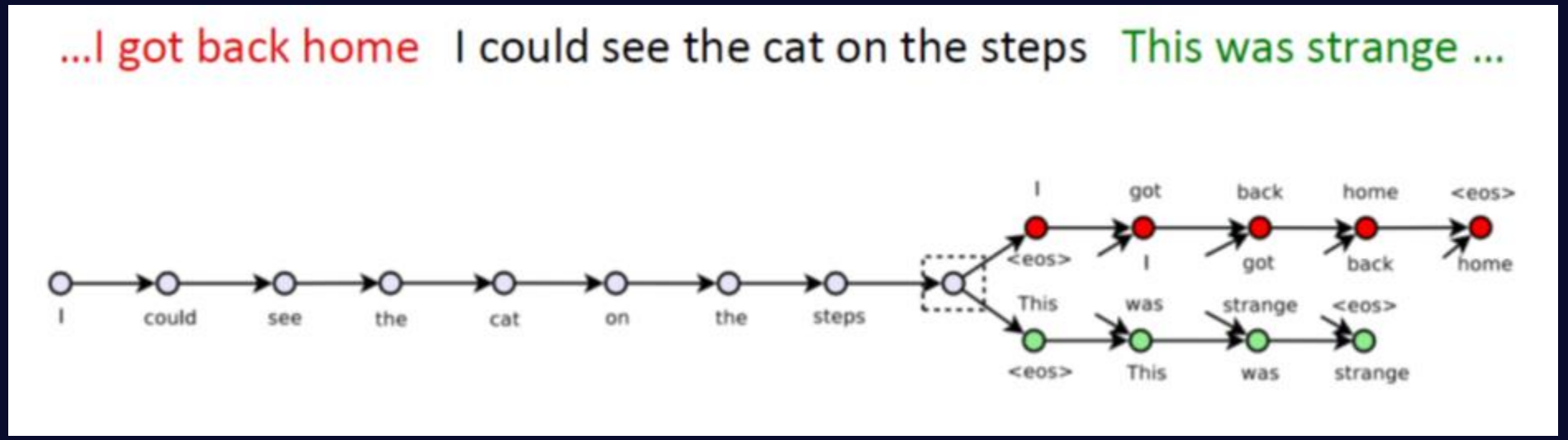
- Encode sentence as vector, then decode it.
- Minimize reconstruction error (MSE or cross entropy)

Sentence Embeddings – LSTM Autoencoders



Sentence Embeddings – Skip Thoughts

- Encode sentence using an RNN
- Decode two neighboring sentences
- Use different RNNs for previous and next sentences



Sentence Embeddings – Skip Thoughts

query sentence:

im sure youll have a glamorous evening , she said , giving an exaggerated wink .

nearest neighbor:

im really glad you came to the party tonight , he said , turning to her .

Sentence Embeddings – Skip Thoughts

query sentence:

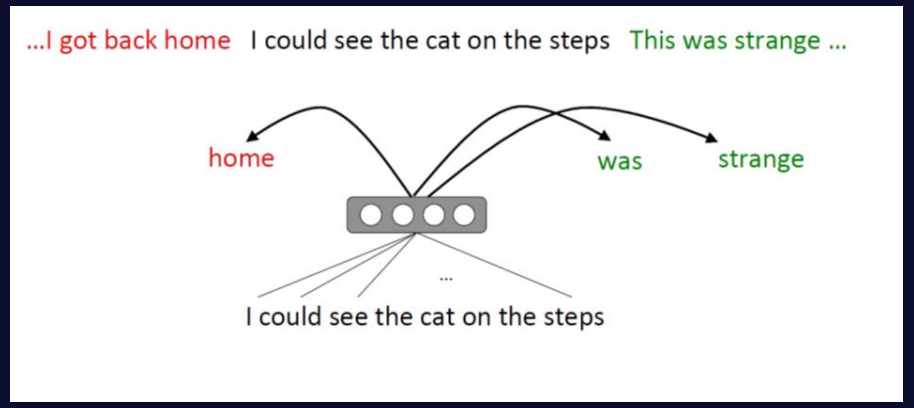
if he had a weapon , he could maybe take out their last imp , and then beat up errol and vanessa .

nearest neighbor:

if he could ram them from behind , send them sailing over the far side of the levee , he had a chance of stopping them .

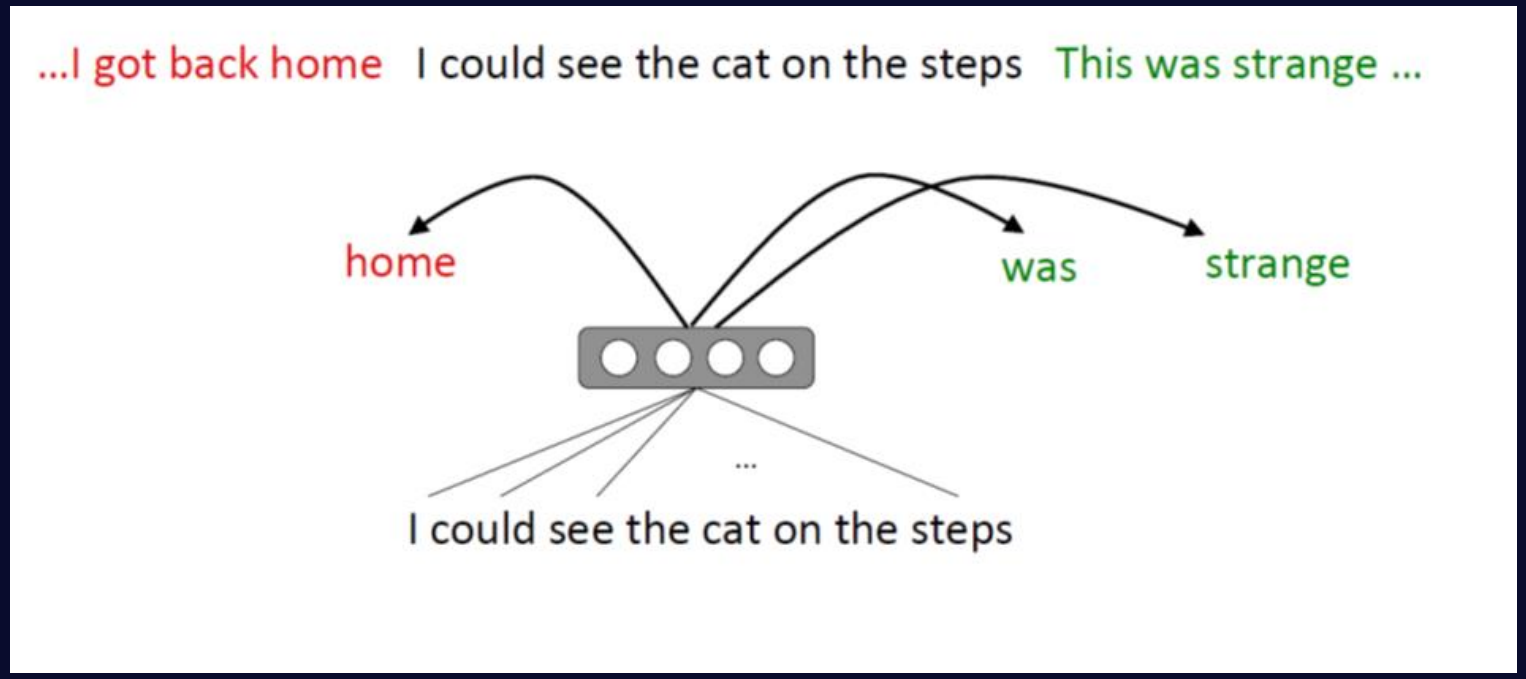
Sentence Embeddings – Fast-Sent

- Skip-thought vectors are slow to train.
- Preserving Skip-Thoughts insight: predicting surrounding sentences is a powerful way to obtain distributed representations.
- Fast-Sent represents sentences as the simple sum of its word embeddings, making training efficient.



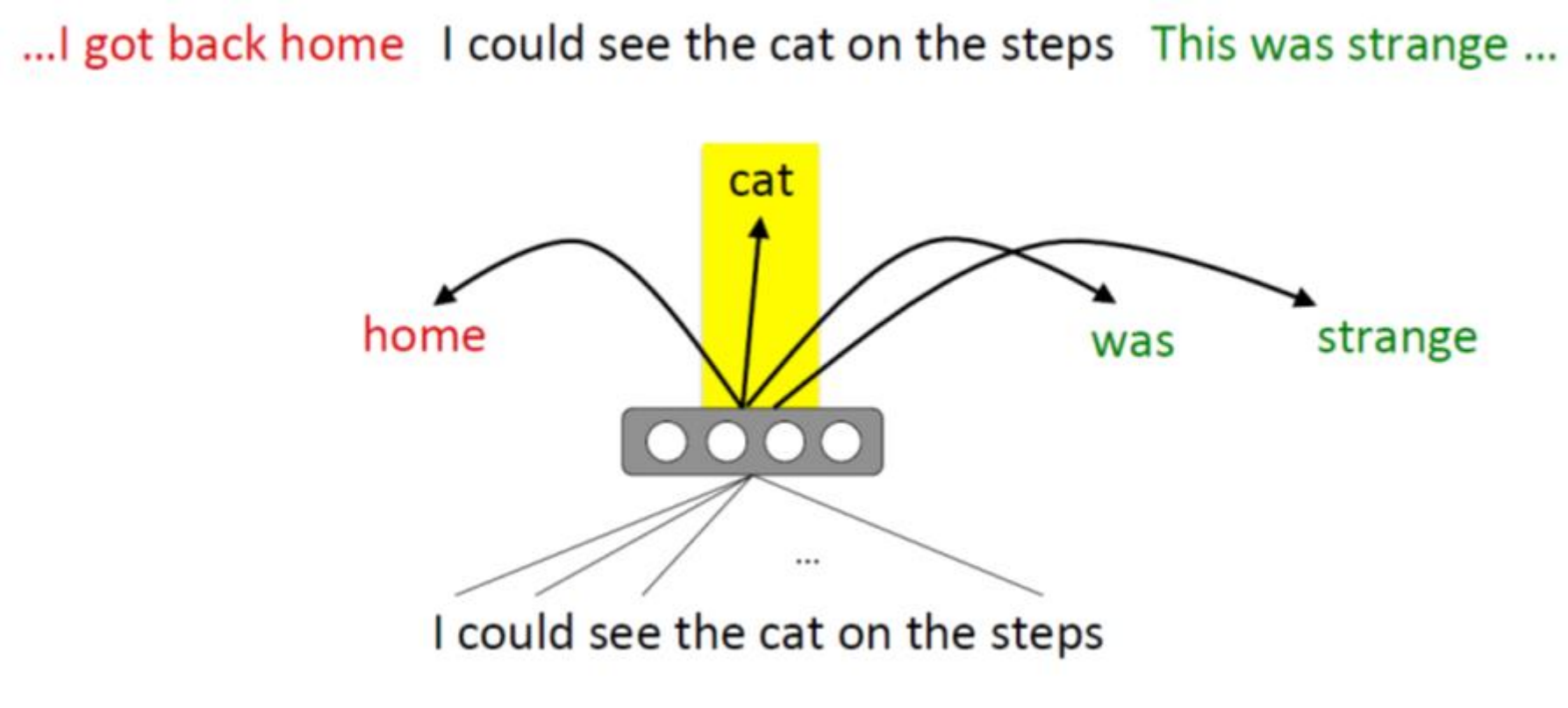
Sentence Embeddings – Fast-Sent

- Encode center sentence using sum.
- Decode to predict words in neighboring sentences.



Sentence Embeddings – Fast-Sent + AE

- Encode center sentence using sum.
- Decode to predict words in current and neighboring sentences.



Sentence Embeddings

- Fancy methods…..

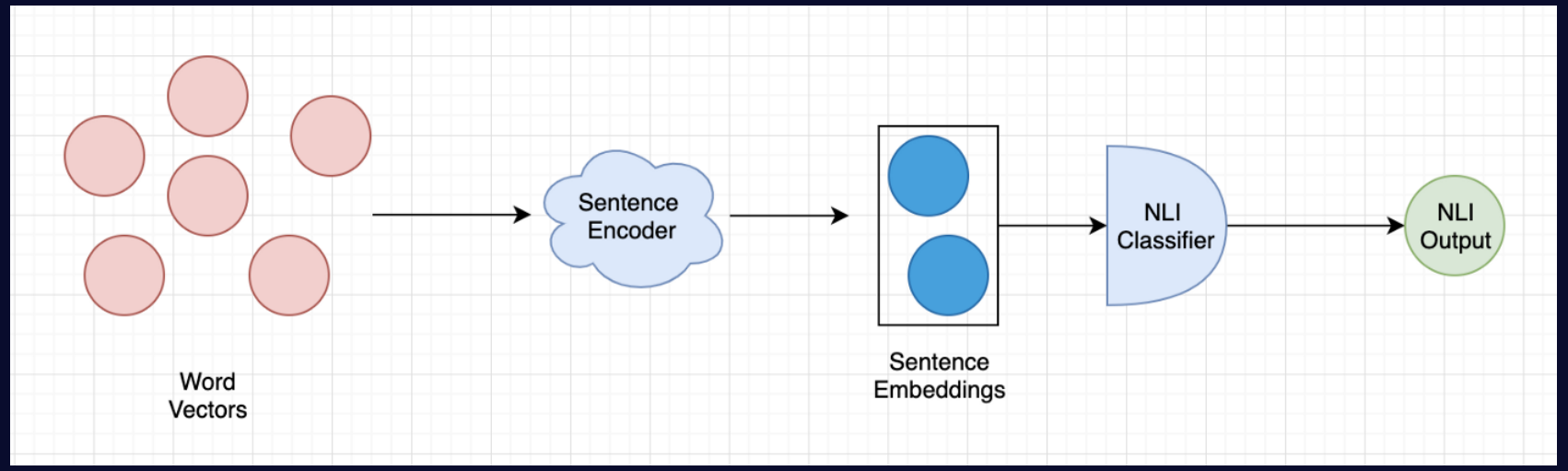
- How about just taking the average of your favorite word embedding method?

Sentence Embeddings - Evaluation

Sentence Embedding Model	STS	Classification
Paragraph Vector	44	70.4
LSTM Autoencoder	43	75.9
LSTM Denoising Autoencoder	38	78.9
Neural MT Encoder	42	76.9
Skip Thought	31	85.3
FastSent	64	79.3
FastSent + Autoencoder	62	79.7
C-PHRASE	67	81.7
Avg. pretrained word embeddings	65	80.6

Sentence Embeddings - InferSent

- The architecture consists of 2 parts:
- One is the sentence encoder that takes word vectors and encodes sentences into vectors.
- Two, an NLI classifier that takes the encoded vectors in and outputs a class among **entailment, contradiction** and **neutral**.



Sentence Embeddings - Universal Sentence Embedding

- Can be used with your favorite DNN.
- Universal Sentence Embedding is trained on a variety of data sources and a variety of tasks with the aim of dynamically accommodating a wide variety of natural language understanding tasks.

Sentence Embeddings - Evaluation

- Ok how about using BERT/ELMO etc.. ?

Sentence Embeddings - Evaluation

Model	STS12	STS13	STS14	STS15	STS16	STSb	SICK-R	Avg.
Avg. GloVe embeddings	55.14	70.66	59.73	68.25	63.66	58.02	53.76	61.32
Avg. BERT embeddings	38.78	57.98	57.98	63.15	61.06	46.35	58.40	54.81
BERT CLS-vector	20.16	30.01	20.09	36.88	38.08	16.50	42.63	29.19
InferSent - Glove	52.86	66.75	62.15	72.77	66.87	68.03	65.65	65.01
Universal Sentence Encoder	64.49	67.80	64.61	76.83	73.18	74.92	**76.69**	71.22

Table 1: Spearman rank correlation ρ between the cosine similarity of sentence representations and the gold labels for various Textual Similarity (STS) tasks. Performance is reported by convention as $\rho \times 100$. STS12-STS16: SemEval 2012-2016, STSb: STSbenchmark, SICK-R: SICK relatedness dataset.

Sentence Embeddings - Evaluation

Ok so you showed us that Bert is not good enough…

- So how can we utilize SOTA language models?

Sentence Bert

- Fine-tune BERT sentence embeddings on NLI datasets.

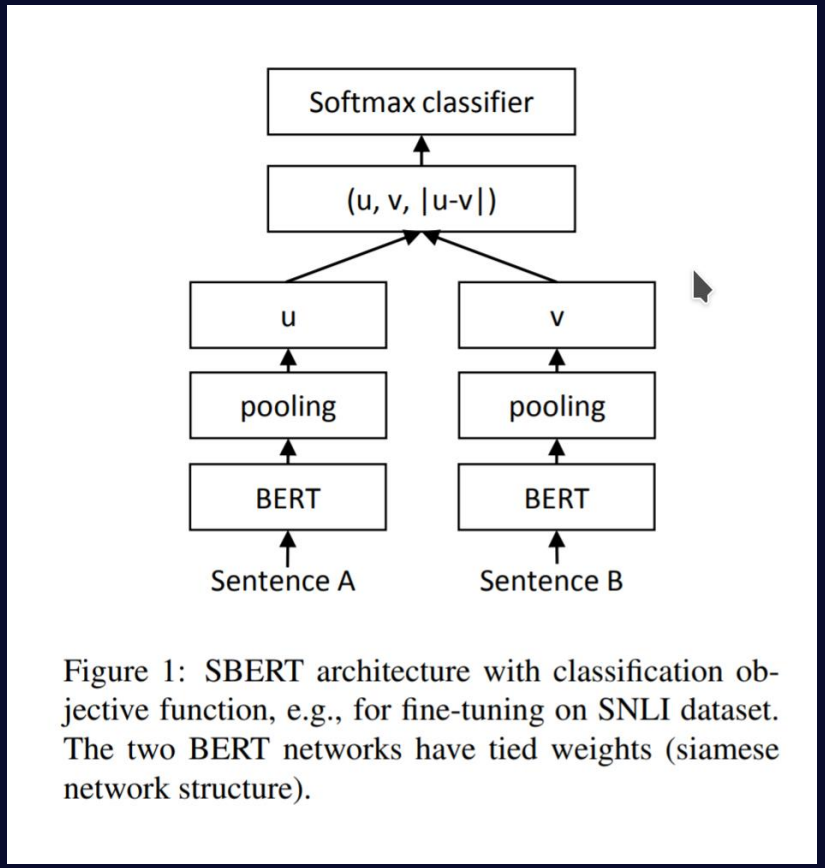


Figure 1: SBERT architecture with classification objective function, e.g., for fine-tuning on SNLI dataset. The two BERT networks have tied weights (siamese network structure).

Sentence Bert

Model	STS12	STS13	STS14	STS15	STS16	STSb	SICK-R	Avg.
Avg. GloVe embeddings	55.14	70.66	59.73	68.25	63.66	58.02	53.76	61.32
Avg. BERT embeddings	38.78	57.98	57.98	63.15	61.06	46.35	58.40	54.81
BERT CLS-vector	20.16	30.01	20.09	36.88	38.08	16.50	42.63	29.19
InferSent - Glove	52.86	66.75	62.15	72.77	66.87	68.03	65.65	65.01
Universal Sentence Encoder	64.49	67.80	64.61	76.83	73.18	74.92	**76.69**	71.22
SBERT-NLI-base	70.97	76.53	73.19	79.09	74.30	77.03	72.91	74.89
SBERT-NLI-large	72.27	**78.46**	**74.90**	80.99	76.25	**79.23**	73.75	76.55
SRoBERTa-NLI-base	71.54	72.49	70.80	78.74	73.69	77.77	74.46	74.21
SRoBERTa-NLI-large	**74.53**	77.00	73.18	**81.85**	**76.82**	79.10	74.29	**76.68**

Table 1: Spearman rank correlation ρ between the cosine similarity of sentence representations and the gold labels for various Textual Similarity (STS) tasks. Performance is reported by convention as $\rho \times 100$. STS12-STS16: SemEval 2012-2016, STSb: STSbenchmark, SICK-R: SICK relatedness dataset.

Sentence Bert

Model	Spearman
Not trained for STS	
Avg. GloVe embeddings	58.02
Avg. BERT embeddings	46.35
InferSent - GloVe	68.03
Universal Sentence Encoder	74.92
SBERT-NLI-base	77.03
SBERT-NLI-large	79.23
Trained on STS benchmark dataset	
BERT-STSb-base	84.30 ± 0.76
SBERT-STSb-base	84.67 ± 0.19
SRoBERTa-STSb-base	**84.92** ± 0.34
BERT-STSb-large	**85.64** ± 0.81
SBERT-STSb-large	84.45 ± 0.43
SRoBERTa-STSb-large	85.02 ± 0.76
Trained on NLI data + STS benchmark data	
BERT-NLI-STSb-base	**88.33** ± 0.19
SBERT-NLI-STSb-base	85.35 ± 0.17
SRoBERTa-NLI-STSb-base	84.79 ± 0.38
BERT-NLI-STSb-large	**88.77** ± 0.46
SBERT-NLI-STSb-large	86.10 ± 0.13
SRoBERTa-NLI-STSb-large	86.15 ± 0.35

Table 2: Evaluation on the STS benchmark test set. BERT systems were trained with 10 random seeds and 4 epochs. SBERT was fine-tuned on the STSb dataset, SBERT-NLI was pretrained on the NLI datasets, then fine-tuned on the STSb dataset.

INT
המרכז הבינלאומי
ללימודי הייטק וחדשנות