Introduction to

Information Retrieval

Information Retrieval and Web Search

Distributed Word Representations for Information Retrieval

How can we more robustly match a user's search intent?

We want to **understand** a query, not just do String equals()

- If user searches for [Dell notebook battery size], we would like to match documents discussing "Dell laptop battery capacity"
- If user searches for [Seattle motel], we would like to match documents containing "Seattle hotel"

A pure keyword-matching IR system does nothing to help….

Simple facilities that we have already discussed do a bit to help

- Spelling correction
- Stemming / case folding

But we'd like to better **understand** when query/document match

How can we more robustly match a user's search intent?

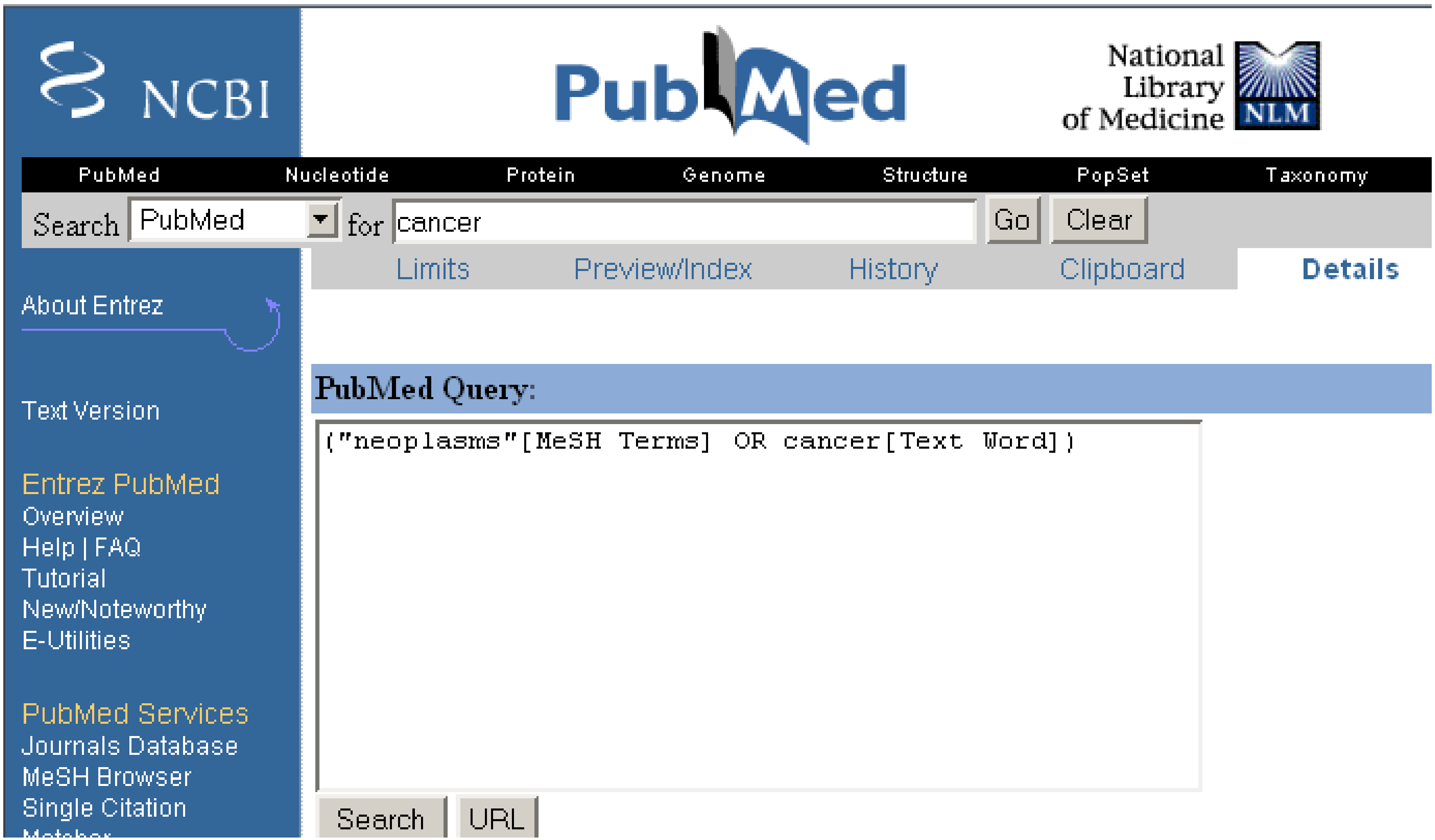
Query expansion:

- **Relevance feedback** could allow us to capture this if we get near enough to matching documents with these words
- We can also use information on **word similarities**:
  - A manual **thesaurus** of synonyms for query expansion
  - A **measure of word similarity**
    - Calculated from a big document collection
    - Calculated by query log mining (common on the web)

Document expansion:

- Use of **anchor text** may solve this by providing human authored synonyms, but not for new or less popular web pages, or non-hyperlinked collections

Example of manual thesaurus



Search log query expansion

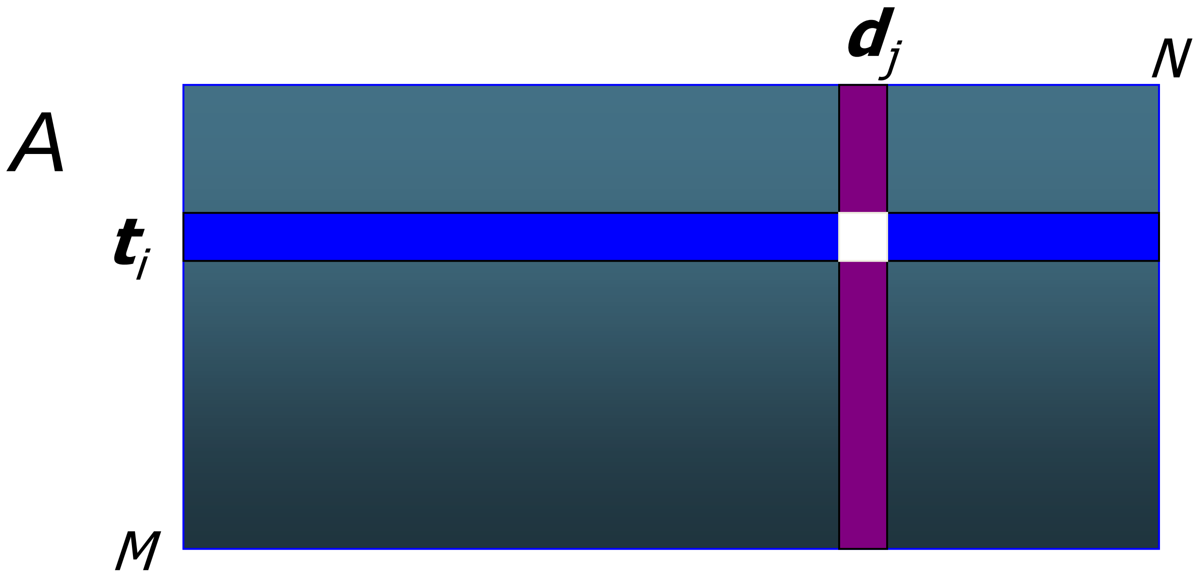
- Context-free query expansion ends up problematic
  - [wet ground] ≈ [wet earth]
  - So expand [ground] ⇒ [ground earth]
  - But [ground coffee] ≠ [earth coffee]
- You can learn query context-specific rewritings from search logs by attempting to identify the same user making a second attempt at the same user need
  - [Hinton word vector]
  - [Hinton word embedding]
- In this context, [vector] ≈ [embedding]
  - But not when talking about a *disease vector* or C++!

Automatic Thesaurus Generation

- Attempt to generate a thesaurus automatically by analyzing a collection of documents
- Fundamental notion: similarity between two words
- Definition 1: Two words are similar if they co-occur with similar words.
- Definition 2: Two words are similar if they occur in a given grammatical relation with the same words.
- You can harvest, peel, eat, prepare, etc. apples and pears, so apples and pears must be similar.
- Co-occurrence based is more robust, grammatical relations are more accurate.

Simple Co-occurrence Thesaurus

- Simplest way to compute one is based on term-term similarities in $C = AA^T$ where A is term-document matrix.
- $w_{i,j}$ = (normalized) weight for (t_i ,$\mathbf{d}_j$)



What does C contain if A is a term-doc incidence (0/1) matrix?

- For each t_i, pick terms with high values in C

Automatic thesaurus generation example … sort of works

Word	Nearest neighbors
absolutely	absurd, whatsoever, totally, exactly, nothing
bottomed	dip, copper, drops, topped, slide, trimmed
captivating	shimmer, stunningly, superbly, plucky, witty
doghouse	dog, porch, crawling, beside, downstairs
makeup	repellent, lotion, glossy, sunscreen, skin, gel
mediating	reconciliation, negotiate, cease, conciliation
keeping	hoping, bring, wiping, could, some, would
lithographs	drawings, Picasso, Dali, sculptures, Gauguin
pathogens	toxins, bacteria, organisms, bacterial, parasites
senses	grasp, psyche, truly, clumsy, naïve, innate

Too little data (10s of millions of words) treated by **too sparse method**.
100,000 words = 10^{10} entries in *C*.

How can we represent term relations?

- With the standard symbolic encoding of terms, each term is a dimension
- Different terms have no inherent similarity
- motel $[0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 1\ 0\ 0\ 0\ 0]^T$
  hotel $[0\ 0\ 0\ 0\ 0\ 0\ 0\ 3\ 0\ 0\ 0\ 0\ 0\ 0\ 0] = 0$
- If query on *hotel* and document has *motel*, then our query and document vectors are **orthogonal**

Can you directly learn term relations?

- Basic IR is scoring on $q^T d$
- No treatment of synonyms; no machine learning
- Can we learn parameters W to rank via $q^T W d$?

"search ranking" "information retrieval ranking"

$$
\underbrace{\begin{array}{ccccc} \multicolumn{5}{c}{q^T} \\ (1 & 0 & 0 & 1 & 0) \\ se & in & re & ra & or \end{array}}
\overset{W}{\begin{pmatrix} 1 & 0.7 & 0.5 & 0 & 0 \\ 0.3 & 1 & 0.2 & 0 & 0 \\ 0.5 & 0.2 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0.7 \\ 0 & 0 & 0 & 0.7 & 1 \end{pmatrix}}
\overset{d}{\begin{pmatrix} 0 \\ 1 \\ 1 \\ 1 \\ 0 \end{pmatrix}}
\begin{array}{l} se \\ in \\ re = \\ ra \\ or(dering) \end{array}
$$

- Cf. Query translation models: Berger and Lafferty (1999)
- Problem is again sparsity – W is huge $> 10^{10}$

Is there a better way?

- Idea:
  - Can we learn a dense low-dimensional representation of a word in $\mathbb{R}^d$ such that dot products $u^T v$ express word similarity?
  - We could still if we want to include a “translation” matrix between vocabularies (e.g., cross-language): $u^T W v$
    - But **now *W* is small**!

- But we’ll develop direct similarity in this class

Distributional similarity based representations

- You can get a lot of value by representing a word by means of its neighbors
- "You shall know a word by the company it keeps"
  - (J. R. Firth 1957: 11)

- One of the most successful ideas of modern statistical NLP

…government debt problems turning into	***banking***	*crises as happened in 2009…*
…saying that Europe needs unified	***banking***	*regulation to replace the hodgepodge…*
…India has just given its	***banking***	*system a shot in the arm…*

↖ These words will represent *banking* ↗

Solution: Low dimensional vectors

- The number of topics that people talk about is small (in some sense)
  - Clothes, movies, politics, …
- Idea: store "most" of the important information in a fixed, small number of dimensions: a dense vector
- Usually 25 – 1000 dimensions

- How to reduce the dimensionality?
  - Go from big, sparse co-occurrence count vector to low dimensional "word embedding"

Traditional Way: Latent Semantic Indexing/Analysis

- Use Singular Value Decomposition (SVD) – kind of like Principal Components Analysis (PCA) for an arbitrary rectangular matrix – or just random projection to find a low-dimensional basis or orthogonal vectors
- Theory is that similarity is preserved as much as possible
- You can actually gain in IR (slightly) by doing LSA, as "noise" of term variation gets replaced by semantic "concepts"
- Somewhat popular in the 1990s [Deerwester et al. 1990, etc.]
  - But **results were always somewhat iffy** (… it worked sometimes)
  - Hard to implement efficiently in an IR system (dense vectors!)
- Discussed in *IIR* chapter 18, but not discussed further here
  - Not on the exam (!!!)

"NEURAL EMBEDDINGS"

Word meaning is defined in terms of vectors

- We will build a dense vector for each word type, chosen so that it is good at predicting other words appearing in its context

… those other words also being represented by vectors … it all gets a bit recursive

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

Neural word embeddings - visualization

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
- $\mathrm{p}(\text{context}|w_t) = \ldots$
- which has a loss function, e.g.,
- $J = 1 - \mathrm{p}(w_{-t}\,|w_t)$
- We look at many positions t in a big language corpus
- We keep adjusting the vector representations of words to minimize this loss

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

- Old idea: 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
- Per-token representations: Deep contextual word representations: ELMo, ULMfit, **BERT**

Non-linear and slow

Fast bilinear models

Current state of the art

Word2vec is a family of algorithms

[Mikolov et al. 2013]

Predict between every word and its context words!

Two algorithms

1. **Skip-grams (SG)**

   Predict context words given target (position independent)

2. Continuous Bag of Words (CBOW)

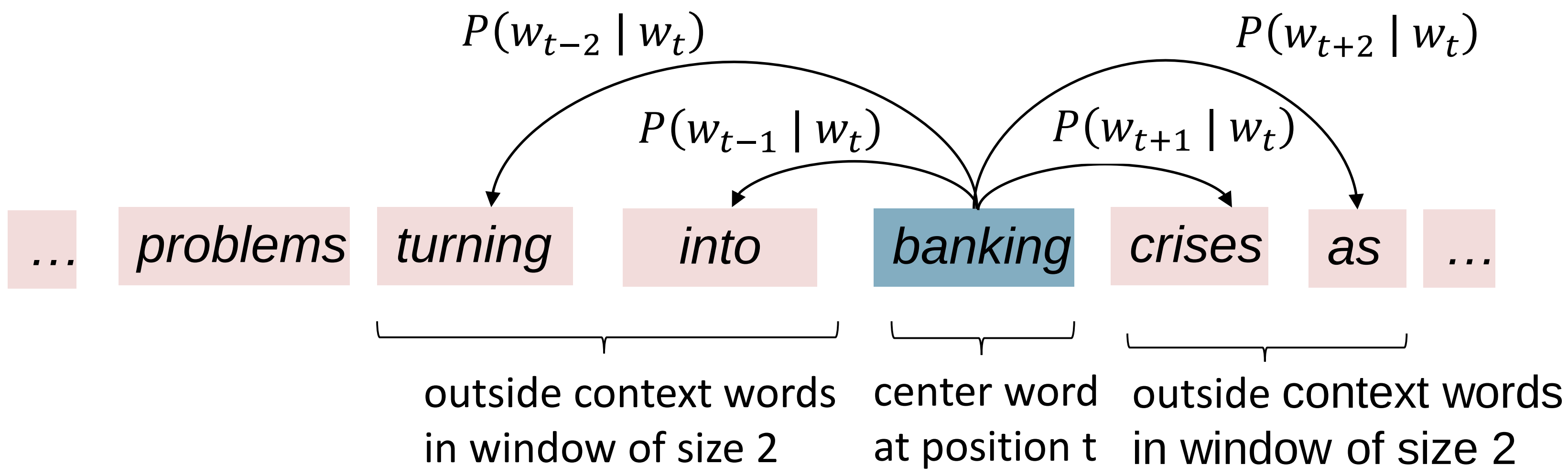
   Predict target word from bag-of-words context

Two (moderately efficient) training methods

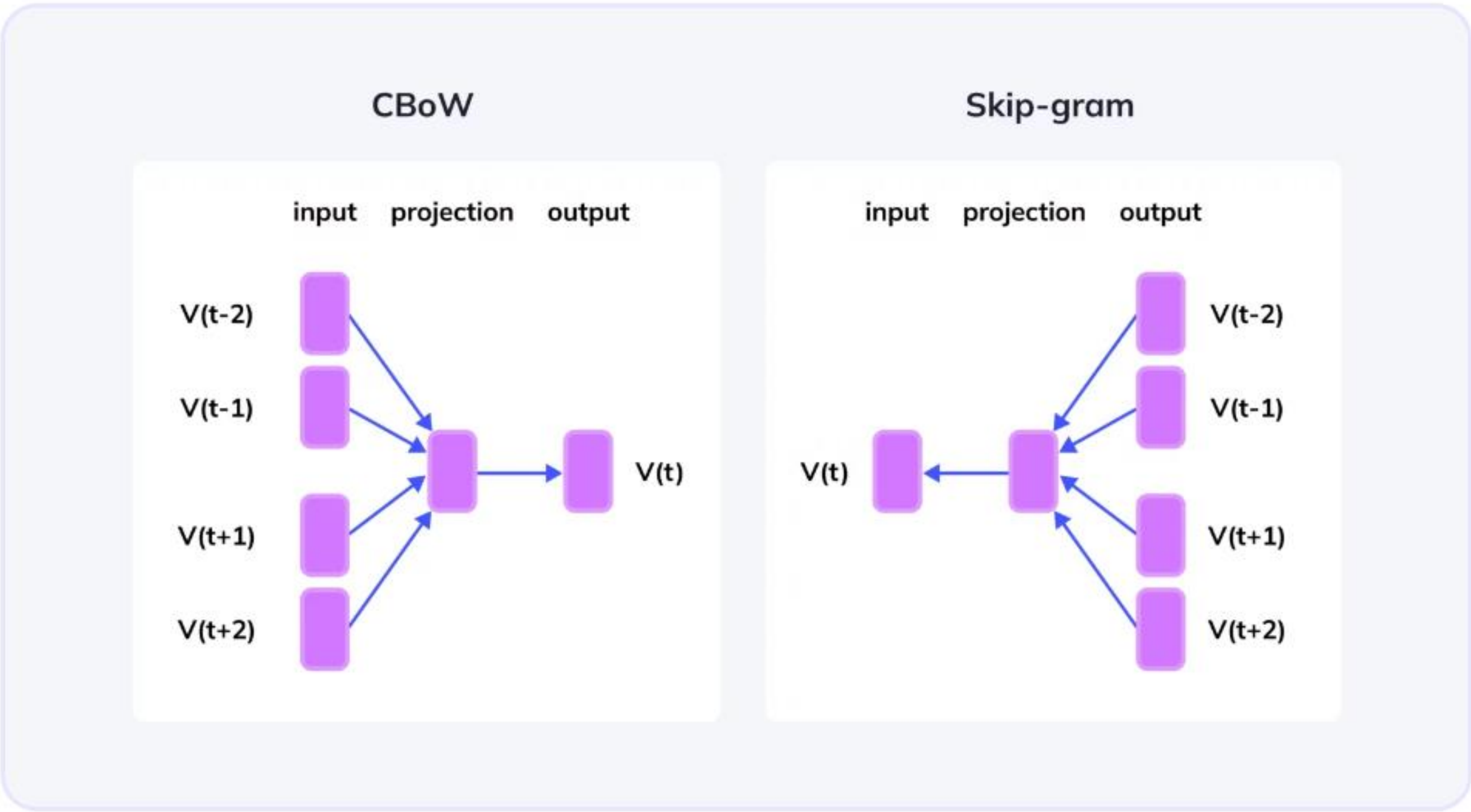
1. Hierarchical softmax
2. Negative sampling
3. **Naïve softmax**

Word2Vec Skip-gram Overview

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



Word2vec: Skip-grams vs CBOW



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.

Likelihood =

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

θ is all variables to be optimized

Sometimes called *cost* or *loss* function

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

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

Minimizing objective function $\Leftrightarrow$ Maximizing predictive accuracy

Word2vec: objective function

- We want to minimize the objective function:

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

- <u>Question:</u> How to calculate $P\left(w_{t+j} \mid w_t; \theta\right)$?
- <u>Answer:</u> 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*:

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

Word2vec: prediction function

Exponentiation makes anything positive

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

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

Normalize over entire vocabulary to give probability distribution

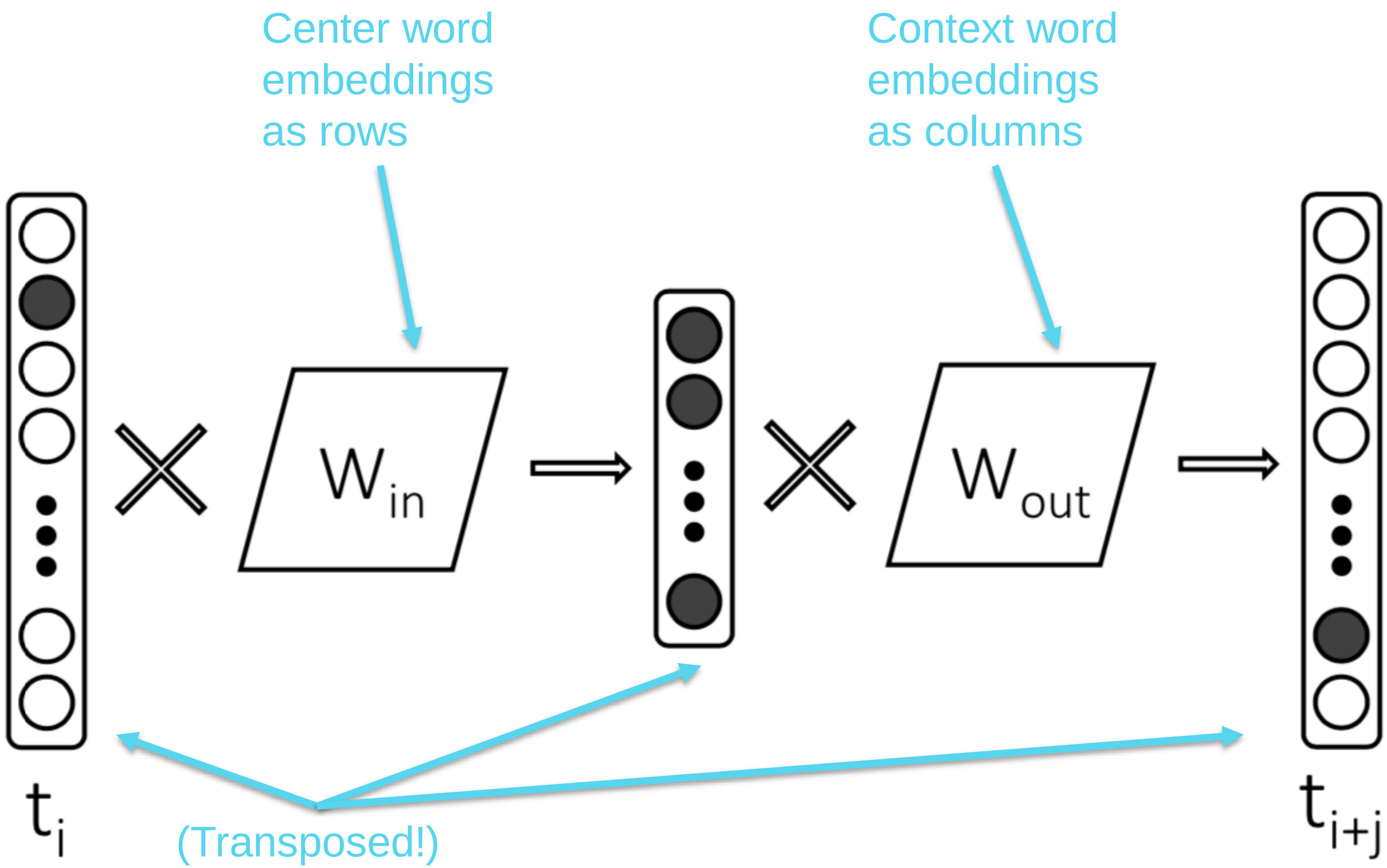
- This is an example of the **softmax function** $\mathbb{R}^n \rightarrow (0,1)^n$

Open region

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

- 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 neural networks/Deep Learning

Word2vec: 2 matrices of parameters



To learn good word vectors: Compute all vector gradients!

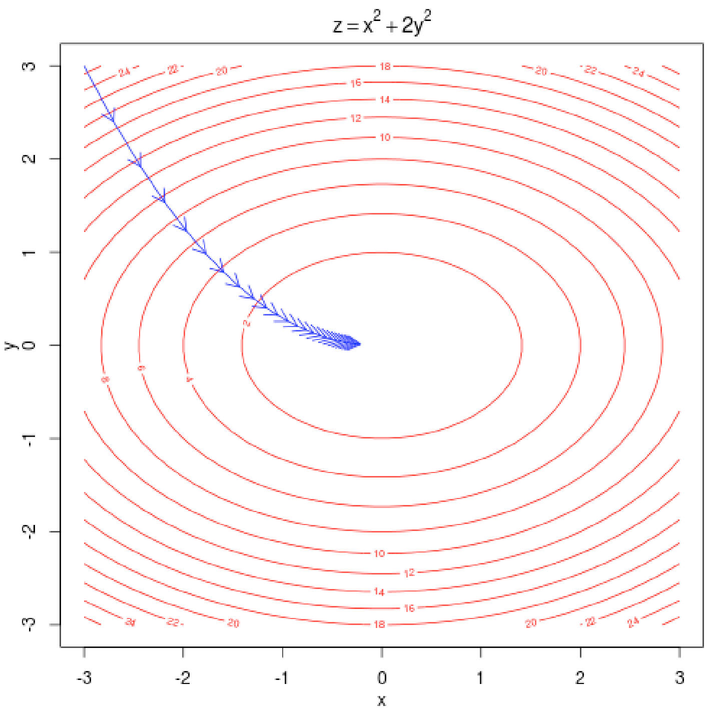
- We often define the set of **all** parameters in a model in terms of one long vector θ
- In our case with *d*-dimensional vectors and *V* many words:
- We then optimize these parameters

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

Note: Every word has two vectors! Makes it simpler!

Intuition of how to minimize loss for a simple function over two parameters

We start at a random point and walk in the steepest direction, which is given by the derivative of the function

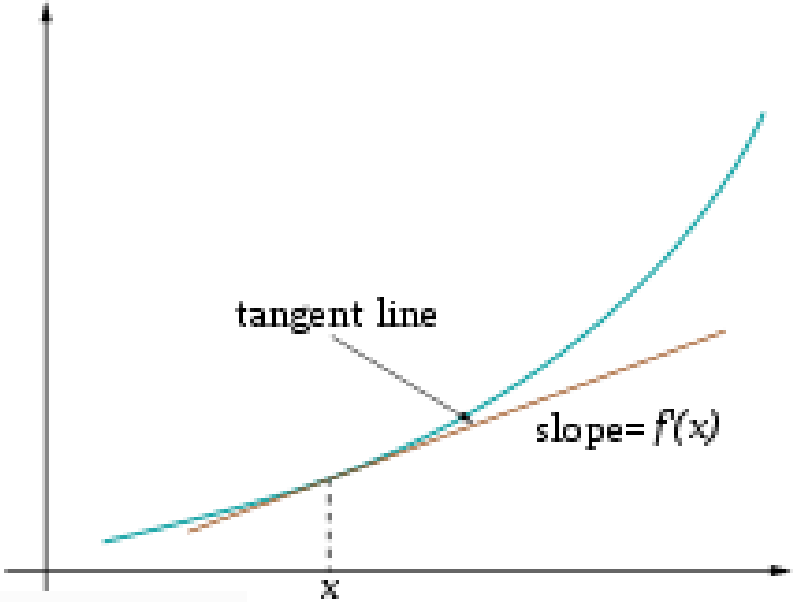


Contour lines show points of equal value of objective function

Descending by using derivatives

We will minimize a cost function by gradient descent

Trivial example: (from Wikipedia)
Find a local minimum of the function
$f(x) = x^4 - 3x^3 + 2$,
with derivative $f'(x) = 4x^3 - 9x^2$




```
x_old = 0
x_new = 6 # The algorithm starts at x=6
eps = 0.01 # step size
precision = 0.00001

def f_derivative(x):
    return 4 * x**3 - 9 * x**2

while abs(x_new - x_old) > precision:
    x_old = x_new
    x_new = x_old - eps * f_derivative(x_old)

print("Local minimum occurs at", x_new)
```

Subtracting a fraction of the gradient moves you towards the minimum!

Vanilla Gradient Descent Code

$$\theta^{new} = \theta^{old} - \alpha \nabla_{\theta} J(\theta)$$

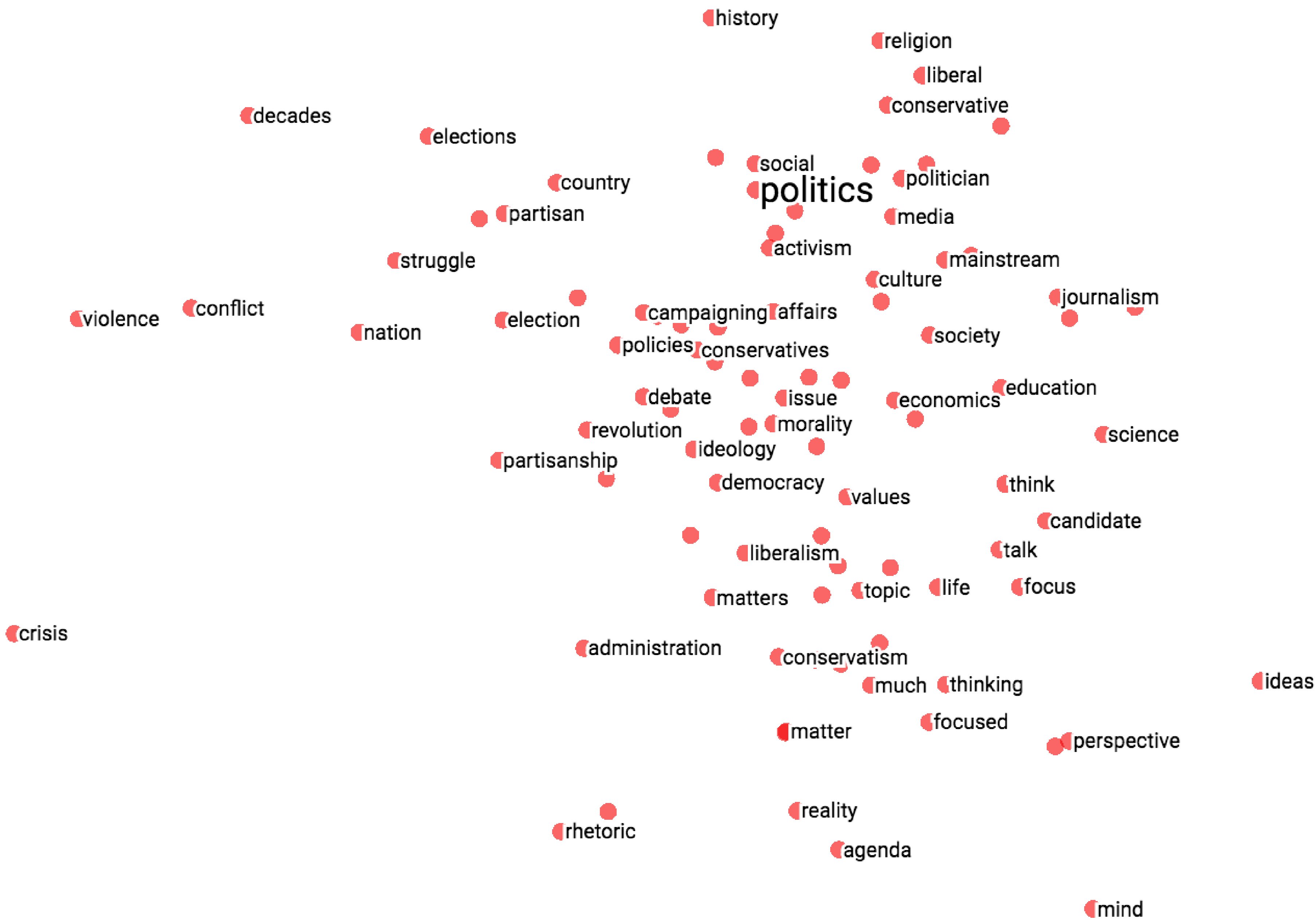
```
while True:
    theta_grad = evaluate_gradient(J,corpus,theta)
    theta = theta - alpha * theta_grad
```

Stochastic Gradient Descent

- But Corpus may have 40B tokens and windows
- You would wait a very long time before making a single update!
- **Very** bad idea for pretty much all neural nets!
- Instead: We update parameters after each window *t*
  → Stochastic gradient descent (SGD)

$$\theta^{new} = \theta^{old} - \alpha \nabla_{\theta} J_t(\theta)$$

```
while True:
    window = sample_window(corpus)
    theta_grad = evaluate_gradient(J,window,theta)
    theta = theta - alpha * theta_grad
```


history
religion
liberal
conservative
decades
elections
social
politics
politician
country
partisan
media
activism
struggle
mainstream
culture
journalism
violence
conflict
nation
election
campaigning
affairs
society
policies
conservatives
debate
issue
economics
education
revolution
morality
ideology
science
partisanship
democracy
values
think
candidate
liberalism
talk
matters
topic
life
focus
crisis
administration
conservatism
much
thinking
ideas
matter
focused
perspective
reality
rhetoric
agenda
mind

Linear Relationships in word2vec

These representations are *very good* at encoding similarity and dimensions of similarity!

- Analogies testing dimensions of similarity can be solved quite well just by doing vector subtraction in the embedding space

  Syntactically

  - $x_{apple} - x_{apples} \approx x_{car} - x_{cars} \approx x_{family} - x_{families}$
  - Similarly for verb and adjective morphological forms

  Semantically (Semeval 2012 task 2)

  - $x_{shirt} - x_{clothing} \approx x_{chair} - x_{furniture}$
  - $x_{king} - x_{man} \approx x_{queen} - x_{woman}$

Word Analogies

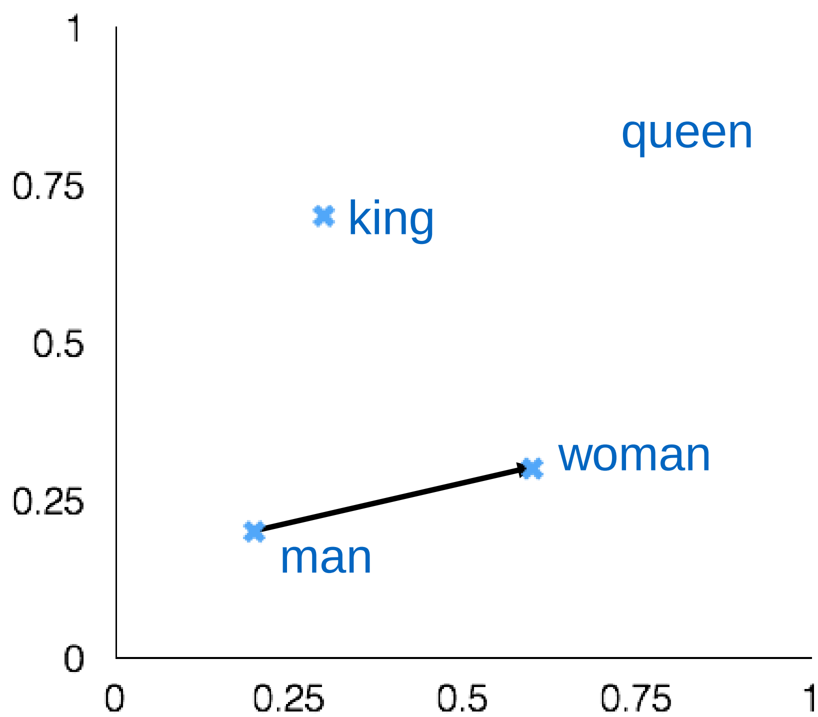
Test for linear relationships, examined by Mikolov et al.

a:b :: c:?

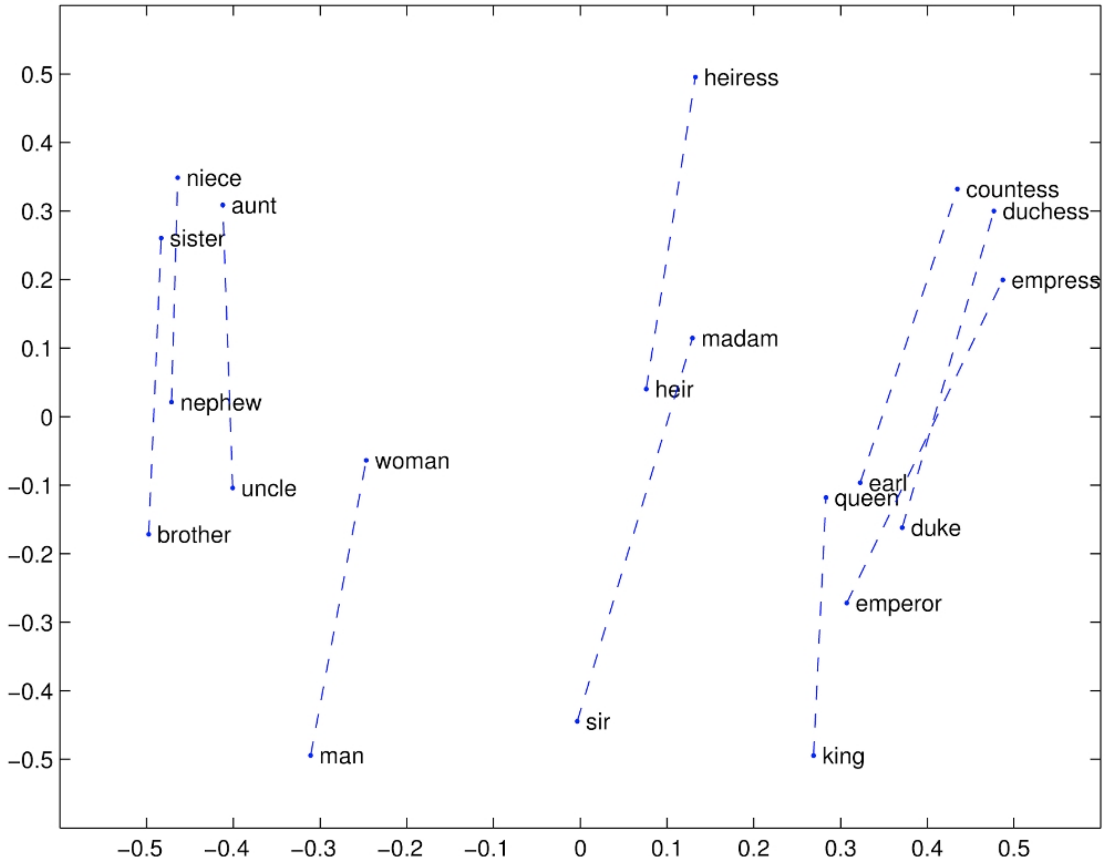
$$d = \arg\max_x \frac{(w_b - w_a + w_c)^T w_x}{||w_b - w_a + w_c||}$$

man:woman :: king:?

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

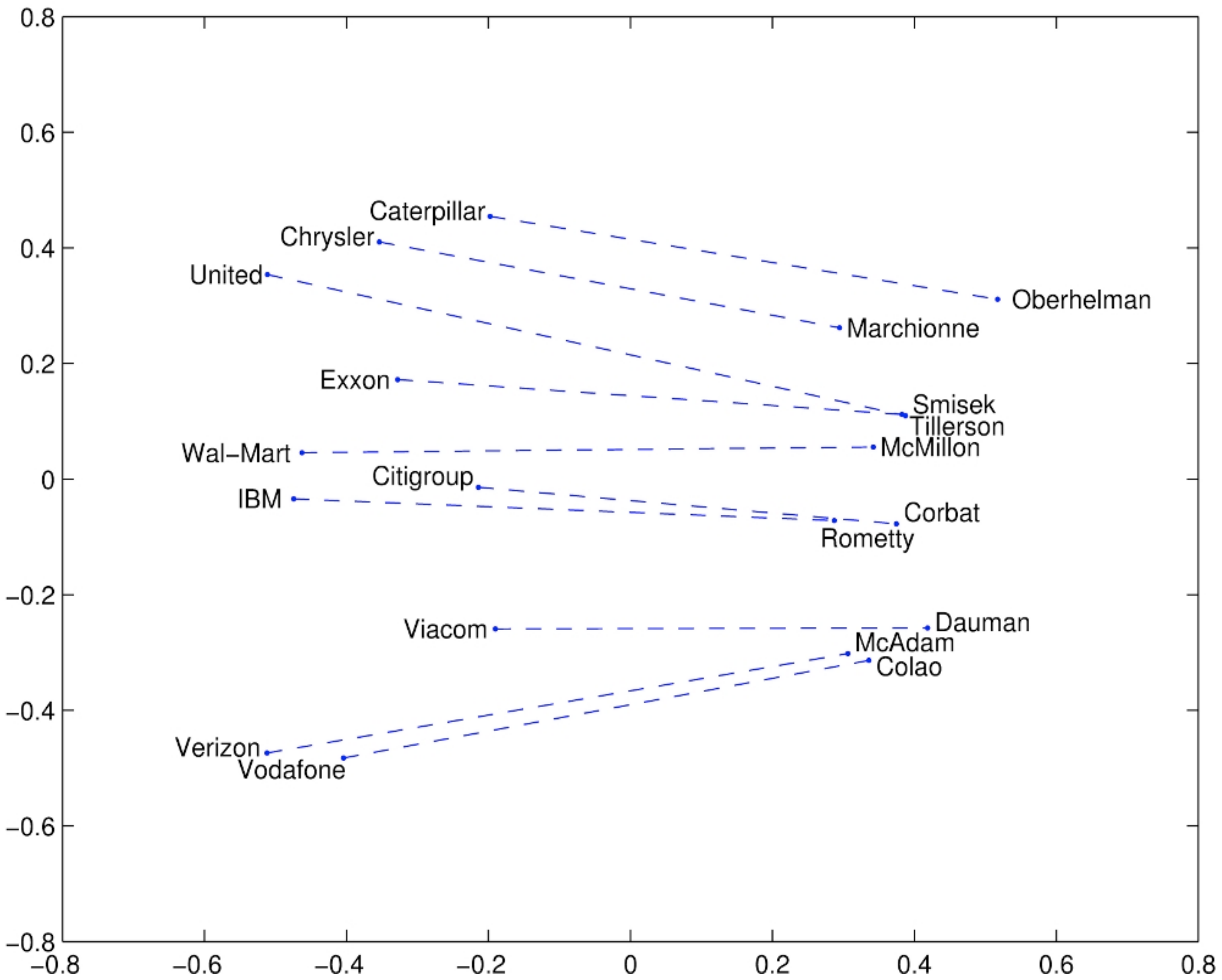


GloVe Visualizations

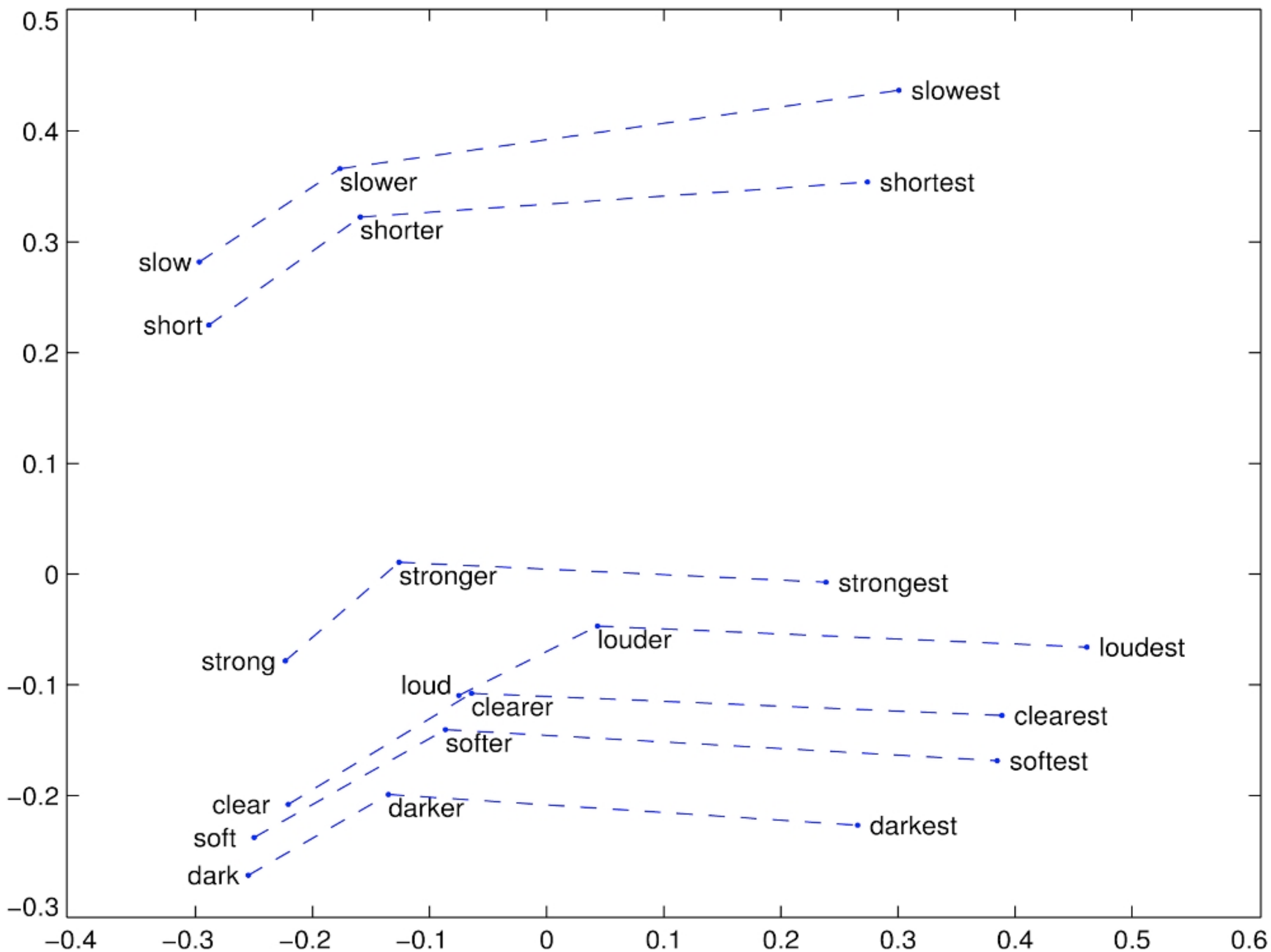


http://nlp.stanford.edu/projects/glove/

Glove Visualizations: Company - CEO



Glove Visualizations: Superlatives



Application to Information Retrieval

Application is just beginning – we're "at the end of the early years"

- Google's RankBrain – little is publicly known
  - Bloomberg article by Jack Clark (Oct 26, 2015): http://www.bloomberg.com/news/articles/2015-10-26/google-turning-its-lucrative-web-search-over-to-ai-machines
  - A result reranking system. "3rd most valuable ranking signal"
  - But note: more of the potential value is in the tail?

An application to information retrieval

Nalisnick, Mitra, Craswell & Caruana. 2016. Improving Document Ranking with Dual Word Embeddings. *WWW 2016 Companion.* http://research.microsoft.com/pubs/260867/pp1291-Nalisnick.pdf

Mitra, Nalisnick, Craswell & Caruana. 2016. A Dual Embedding Space Model for Document Ranking. **arXiv:1602.01137 [cs.IR]**

Builds on BM25 model idea of "aboutness"

- Not just term repetition indicating aboutness
- Relationship between query terms and *all* terms in the document indicates aboutness (BM25 uses only query terms)

Makes clever argument for different use of word and context vectors in word2vec's CBOW/SGNS or GloVe

Modeling document aboutness: Results from a search for Albuquerque

d_1

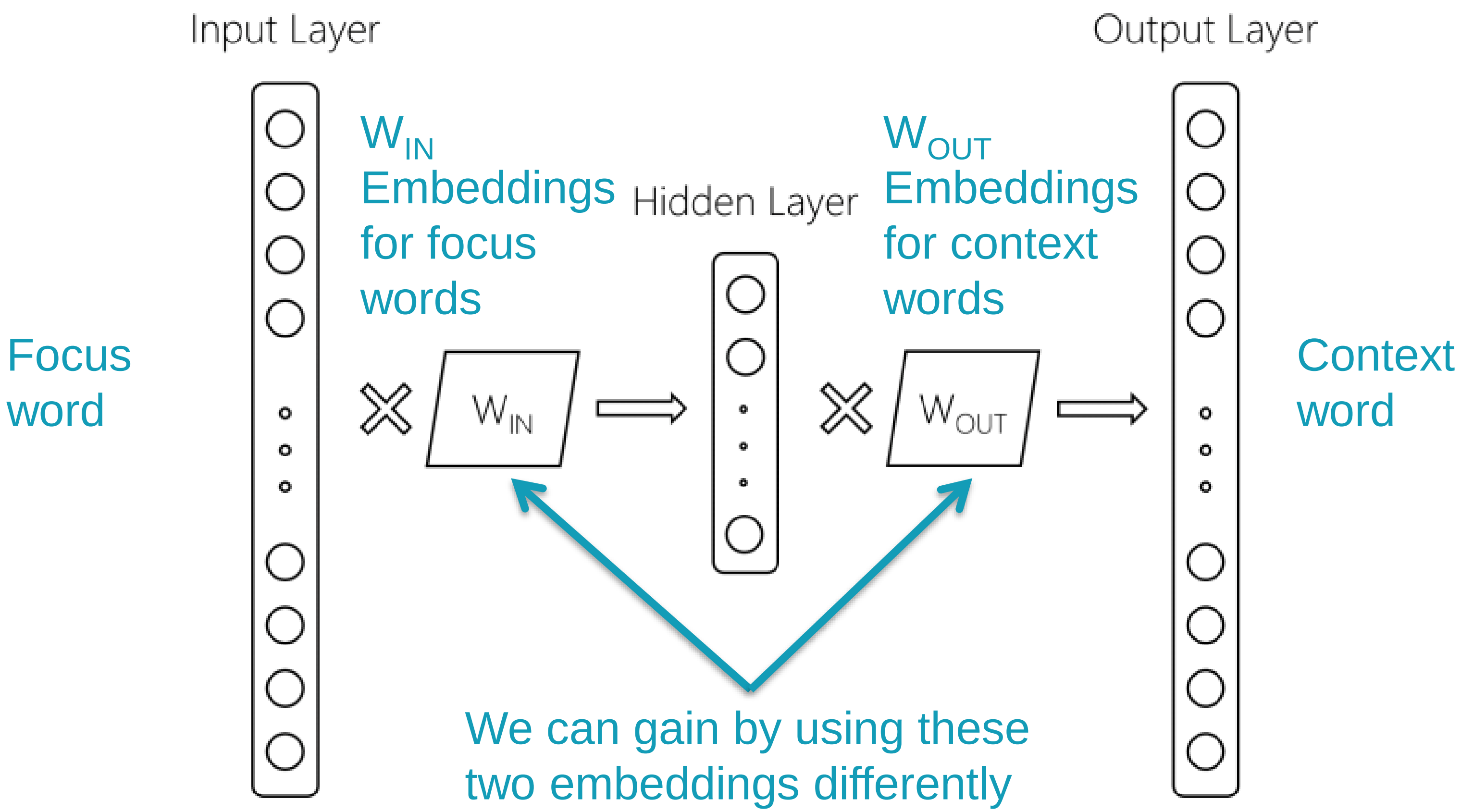
Allen suggested that they could program a BASIC interpreter for the device; after a call from Gates claiming to have a working interpreter, MITS requested a demonstration. Since they didn't actually have one, Allen worked on a simulator for the Altair while Gates developed the interpreter. Although they developed the interpreter on a simulator and not the actual device, the interpreter worked flawlessly when they demonstrated the interpreter to MITS in Albuquerque, New Mexico in March 1975; MITS agreed to distribute it, marketing it as Altair BASIC.

d_2

Albuquerque is the most populous city in the U.S. state of New Mexico. The high-altitude city serves as the county seat of Bernalillo County, and it is situated in the central part of the state, straddling the Rio Grande. The city population is 557,169 as of the July 1, 2014, population estimate from the United States Census Bureau, and ranks as the 32nd-largest city in the U.S. The Metropolitan Statistical Area (or MSA) has a population of 902,797 according to the United States Census Bureau's most recently available estimate for July 1, 2013.

Using 2 word embeddings

word2vec model with 1 word of context



Dual Embedding Space Model (DESM)

- Simple model
- A document is represented by the centroid of its word vectors

$$\overline{\mathbf{D}} = \frac{1}{|D|} \sum_{\mathbf{d}_j \in D} \frac{\mathbf{d}_j}{\|\mathbf{d}_j\|}$$

- Query-document similarity is average over query words of cosine similarity

$$DESM(Q, D) = \frac{1}{|Q|} \sum_{q_i \in Q} \frac{\mathbf{q}_i^T \overline{\mathbf{D}}}{\|\mathbf{q}_i\| \|\overline{\mathbf{D}}\|}$$

Dual Embedding Space Model (DESM)

- What works best is to use the OUT vectors for the document and the IN vectors for the query

$$DESM_{IN-OUT}(Q,D) = \frac{1}{|Q|} \sum_{q_i \in Q} \frac{q_{IN,i}^T \overline{D_{OUT}}}{\|q_{IN,i}\| \|\overline{D_{OUT}}\|}$$

- This way similarity measures *aboutness* – words that appear with this word – which is more useful in this context than *(distributional) semantic similarity*

Experiments

- Train word2vec from either
  - 600 million Bing queries
  - 342 million web document sentences
- Test on 7,741 randomly sampled Bing queries
  - 5 level eval (Perfect, Excellent, Good, Fair, Bad)
- Two approaches
  1. Use DESM model to rerank top results from BM25
  2. Use DESM alone or a mixture model of it and BM25

$$MM(Q,D) = \alpha DESM(Q,D) + (1-\alpha)BM25(Q,D)$$
$$\alpha \in \mathbb{R}, 0 \leq \alpha \leq 1$$

Results – reranking *k*-best list

	Explicitly Judged Test Set		
	NDCG@1	NDCG@3	NDCG@10
BM25	23.69	29.14	44.77
LSA	22.41*	28.25*	44.24*
DESM (IN-IN, trained on body text)	23.59	29.59	45.51*
DESM (IN-IN, trained on queries)	23.75	29.72	46.36*
DESM (IN-OUT, trained on body text)	24.06	30.32*	46.57*
DESM (IN-OUT, trained on queries)	**25.02***	**31.14***	**47.89***

Pretty decent gains – e.g., 2% for NDCG@3

Gains are bigger for model trained on queries than docs

Results – whole ranking system

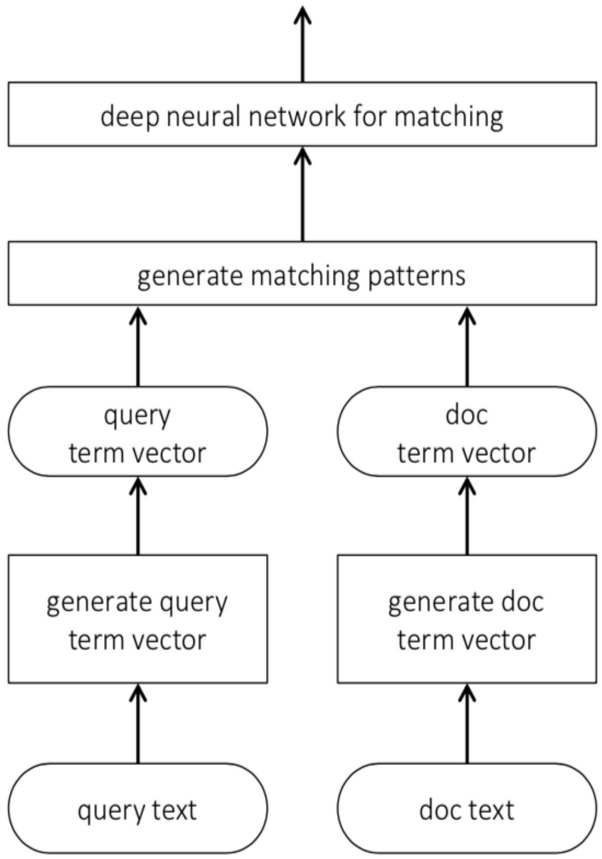
	Explicitly Judged Test Set		
	NDCG@1	NDCG@3	NDCG@10
BM25	21.44	26.09	37.53
LSA	04.61*	04.63*	04.83*
DESM (IN-IN, trained on body text)	06.69*	06.80*	07.39*
DESM (IN-IN, trained on queries)	05.56*	05.59*	06.03*
DESM (IN-OUT, trained on body text)	01.01*	01.16*	01.58*
DESM (IN-OUT, trained on queries)	00.62*	00.58*	00.81*
BM25 + DESM (IN-IN, trained on body text)	21.53	26.16	37.48
BM25 + DESM (IN-IN, trained on queries)	**21.58**	26.20	37.62
BM25 + DESM (IN-OUT, trained on body text)	21.47	26.18	37.55
BM25 + DESM (IN-OUT, trained on queries)	21.54	**26.42***	**37.86***

DESM conclusions

- DESM is a weak ranker but effective at finding subtler similarities/aboutness
- It is effective at, but only at, reranking at least somewhat relevant documents
  - For example, DESM can confuse Oxford and Cambridge
  - Bing rarely makes an Oxford/Cambridge mistake!

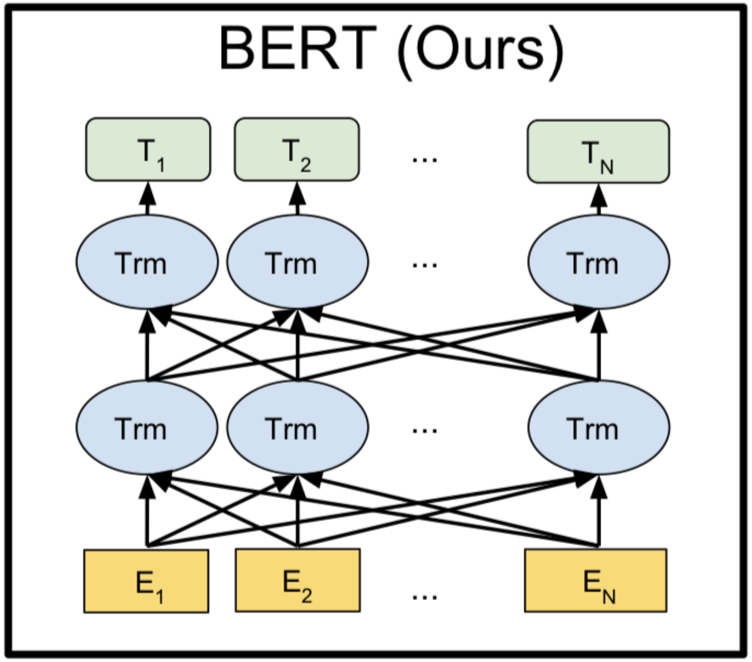
What else can neural nets do in IR?

- Use a neural network as a supervised reranker
- Assume a query and document embedding network (as we have discussed)
- Assume you have (q,d,rel) relevance data
- Learn a neural network (with supervised learning) to predict relevance of (q,d) pair

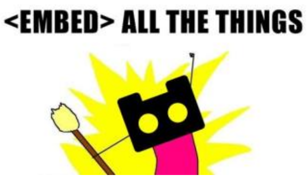


What else can neural nets do in IR?

- BERT (*Bidirectional Encoder Representations*
- *from Transformers)*: Devlin, Chang, Lee, Toutanova (2018)
- A deep transformer-based neural network
- Builds per-token (in context) representations
- Produces a query/document representation as well
- Or jointly embed query and document and ask for a retrieval score
- Incredibly effective!
- https://arxiv.org/abs/1810.04805

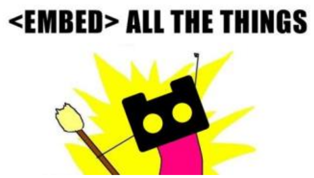


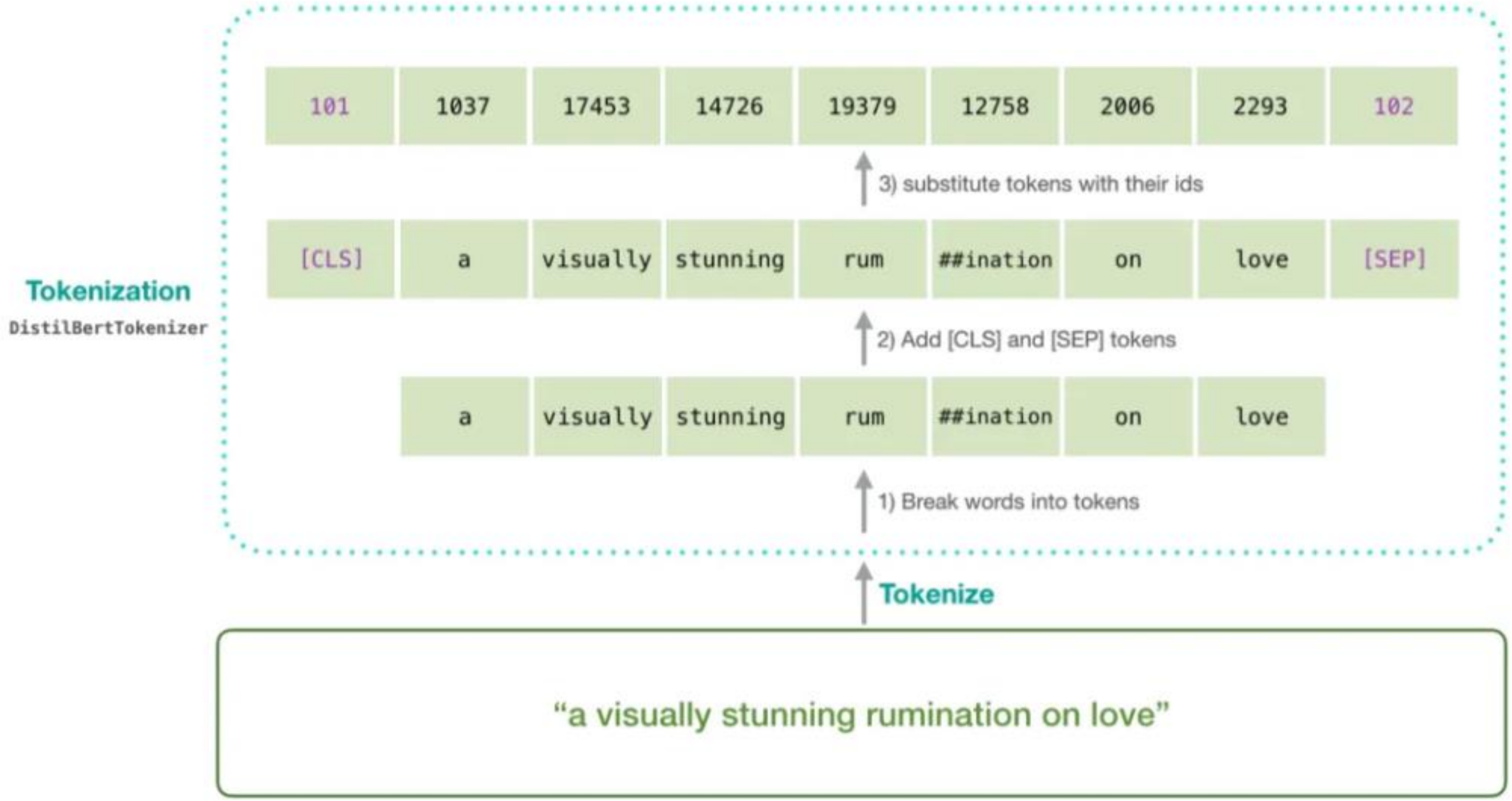
BERT



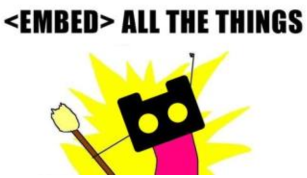
- A massive dataset of 3.3 Billion words has contributed to BERT's continued success.
- BERT was specifically trained on Wikipedia (~2.5B words) and Google's BooksCorpus (~800M words).
- Masked Language Model
- Next Sentence Prediction

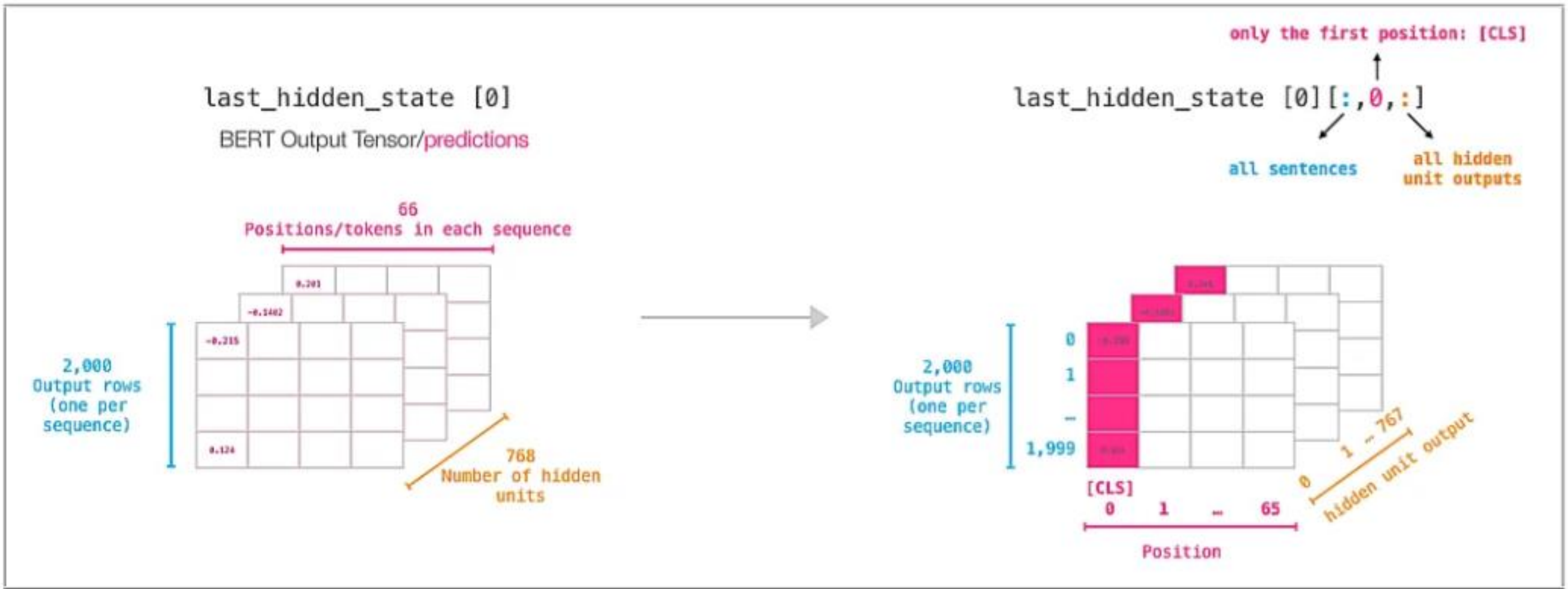
BERT


<EMBED> ALL THE THINGS


Tokenization
DistilBertTokenizer
101
1037
17453
14726
19379
12758
2006
2293
102
3) substitute tokens with their ids
[CLS]
a
visually
stunning
rum
##ination
on
love
[SEP]
2) Add [CLS] and [SEP] tokens
a
visually
stunning
rum
##ination
on
love
1) Break words into tokens
Tokenize
"a visually stunning rumination on love"

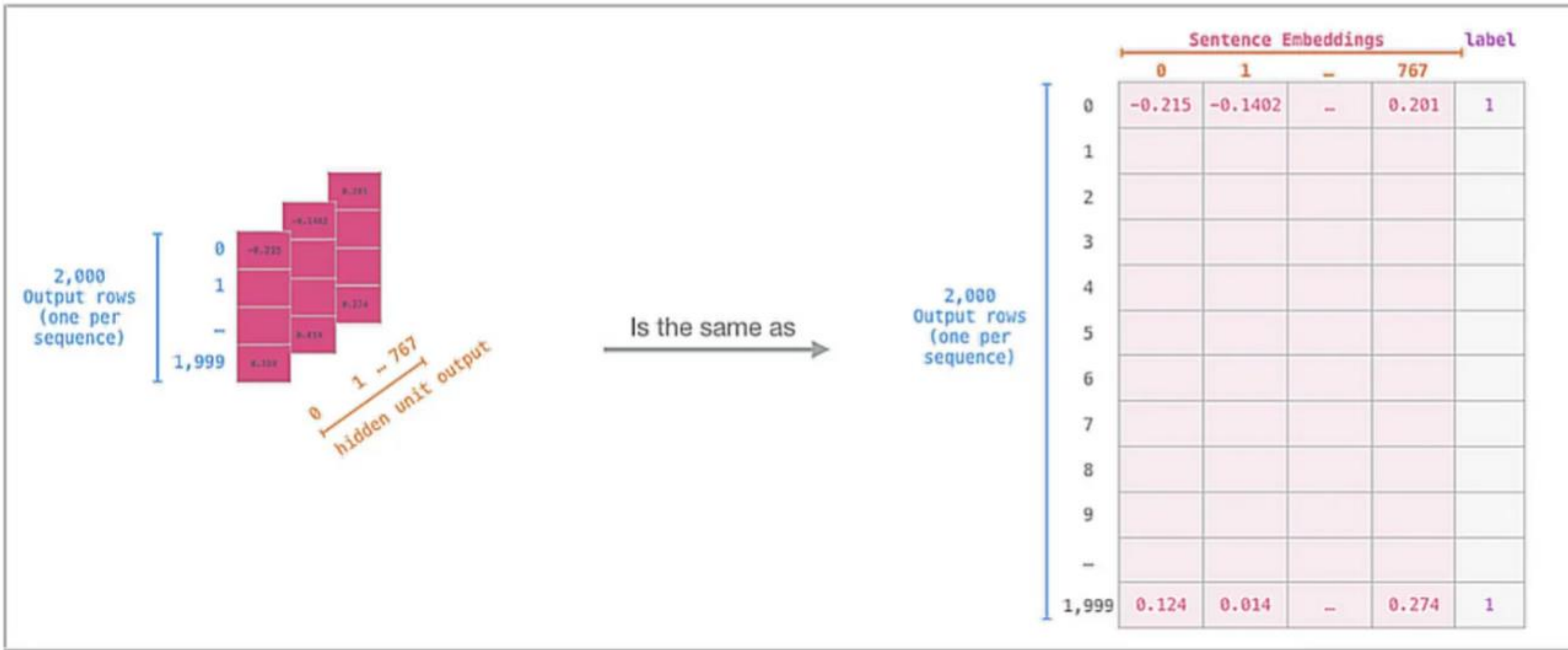
BERT


<EMBED> ALL THE THINGS

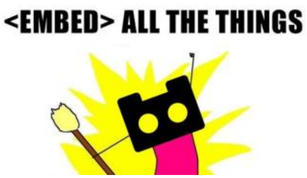

last_hidden_state [0]
BERT Output Tensor/predictions
66
Positions/tokens in each sequence
2,000
Output rows
(one per
sequence)
768
Number of hidden
units
only the first position: [CLS]
last_hidden_state [0][:,0,:]
all sentences
all hidden
unit outputs
2,000
Output rows
(one per
sequence)
0
1
1,999
[CLS]
0
1
65
Position
0
1 … 767
hidden unit output


<EMBED> ALL THE THINGS

BERT


2,000
Output rows
(one per
sequence)
0
1
…
1,999
0
1 … 767
hidden unit output
Is the same as
2,000
Output rows
(one per
sequence)
Sentence Embeddings
label
0
1
…
767
0 -0.215 -0.1402 … 0.201 1
1
2
3
4
5
6
7
8
9
…
1,999 0.124 0.014 … 0.274 1

Summary: Embed all the things!



Word embeddings are the hot new technology (again!)

Lots of applications wherever knowing word context or similarity helps prediction:

- Synonym handling in search
- Document aboutness
- Ad serving
- Language models: from spelling correction to email response
- Machine translation
- Sentiment analysis
- …