自然语言处理

week-6

凌震华

2024年3月28日

◘Vector Semantics (语义学)

Chapter 6 “Vector Semantics”, Speech and Language Processing (3rd ed. draft), https://web.stanford.edu/~jurafsky/slp3/

Overview

- Semantics is about meaning
- Meaning maps strings/utterances to the "real" world
  - Physical world: objects, people, places
  - Mental world: concepts, feelings, ideas
- Meaning of words and sentences
  - The use of vectors
  - Processing pipeline: Syntax → Semantics

What Counts as Understanding?

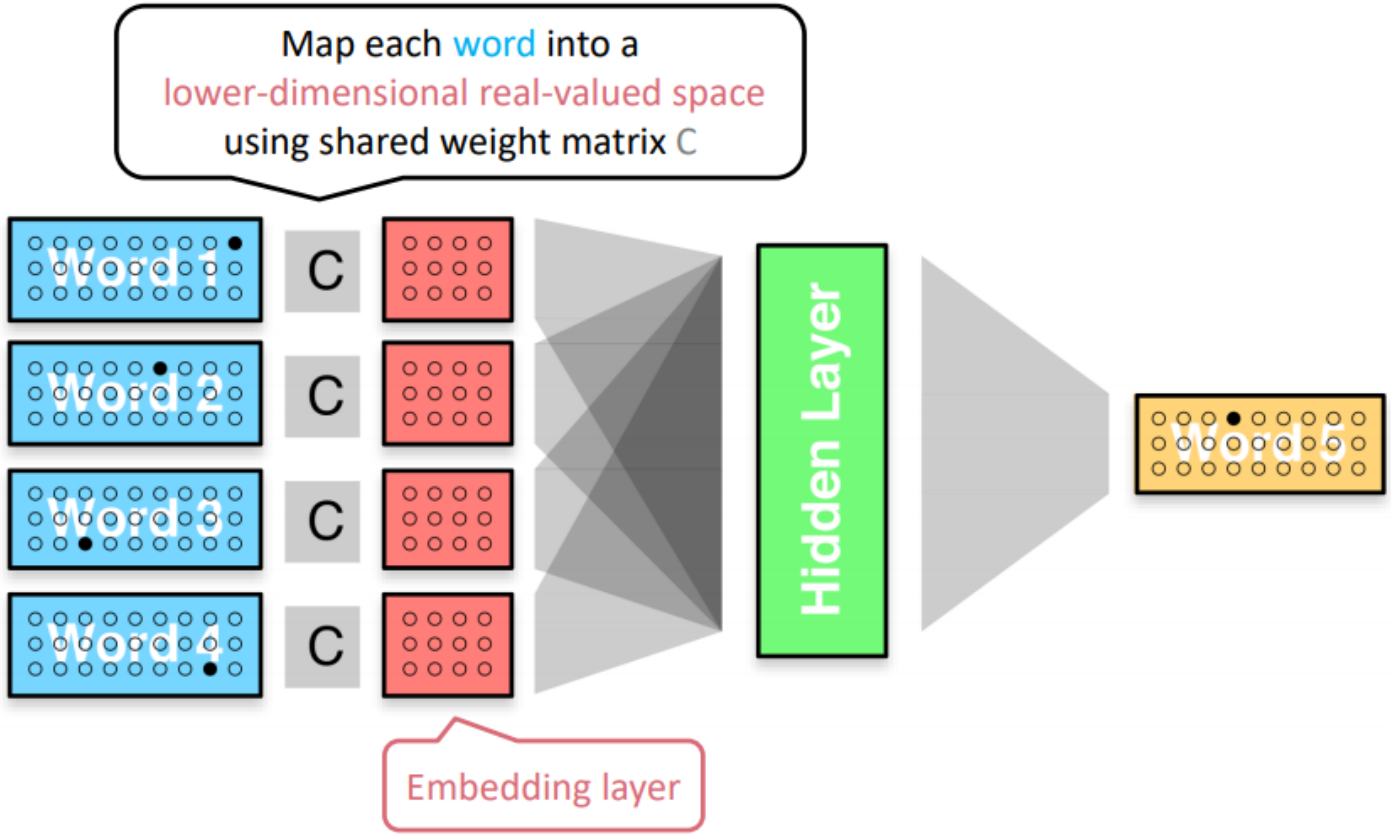
- A somewhat difficult philosophical question
- We understand if we can respond appropriately
  - "throw axe at dwarf"
- We understand statement if we can determine its truth
- We understand statement if we can use it to answer questions [similar to above – requires reasoning]
  - **Easy:** John ate pizza. What was eaten by John?
- Understanding is the ability to translate
  - English to Chinese? requires deep understanding?
  - English to **logic**? deepest - one definition we' ll use later!
    - all humans are mortal = $\forall x$ [human(x) $\Rightarrow$mortal(x)]
- We can represent truth in a model
- We assume we have logic-manipulating rules to tell us how to act, draw conclusions, answer questions …
- "Meaning is relative" : understand one thing relative to another (another definition that we will use)

Vector Semantics

Intuitions:

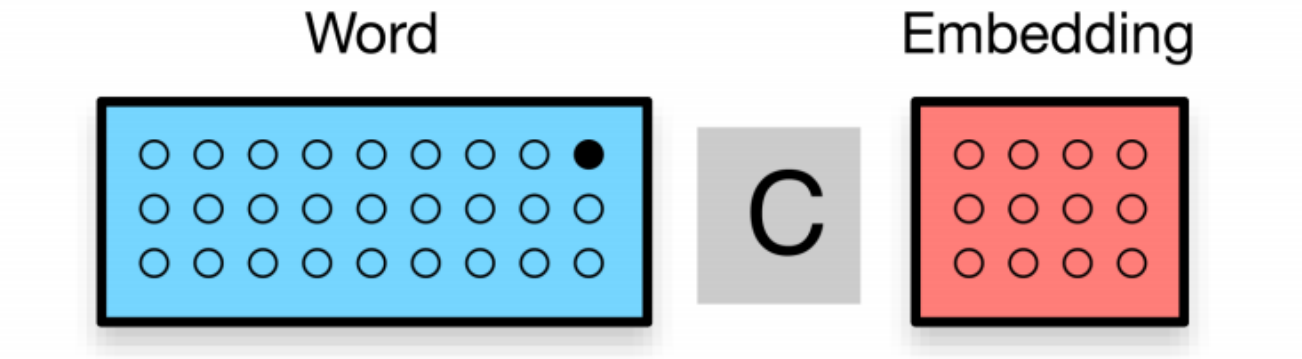
- Zellig Harris (1954):
  - "oculist and eye-doctor … occur in almost the same environments"
  - "If A and B have almost identical environments we say that they are synonyms."
- Firth (1957):
  - "You shall know a word by the company it keeps!"

Recall: Neural Language Model



Bengio et al. 2003

Word Embeddings



- Neural language models produce word embeddings as a by product
- Words that occurs in similar contexts tend to have similar embeddings
- Embeddings are useful features in many NLP tasks

Word Embeddings Illustrated

Similarity between Words

"**fast**" is similar to "**rapid**"

"**tall**" is similar to "**height**"

Question answering:

- *Q: "How **tall** is Mt. Everest?"*
- *Candidate A: "The official **height** of Mount Everest is 29029 feet"*

Word similarity for plagiarism detection

MAINFRAMES

Mainframes are primarily referred to large computers with rapid, advanced processing capabilities that can execute and perform tasks equivalent to many Personal Computers (PCs) machines networked together. It is characterized with high quantity Random Access Memory (RAM), very large secondary storage devices, and high-speed processors to cater for the needs of the computers under its service.

Consisting of advanced components, mainframes have the capability of running multiple large applications required by many and most enterprises and organizations. This is one of its advantages. Mainframes are also suitable to cater for those applications (programs) or files that are of very high demand by its users (clients). Examples of such organizations and enterprises using mainframes are online shopping websites such as Ebay, Amazon, and computing-giant

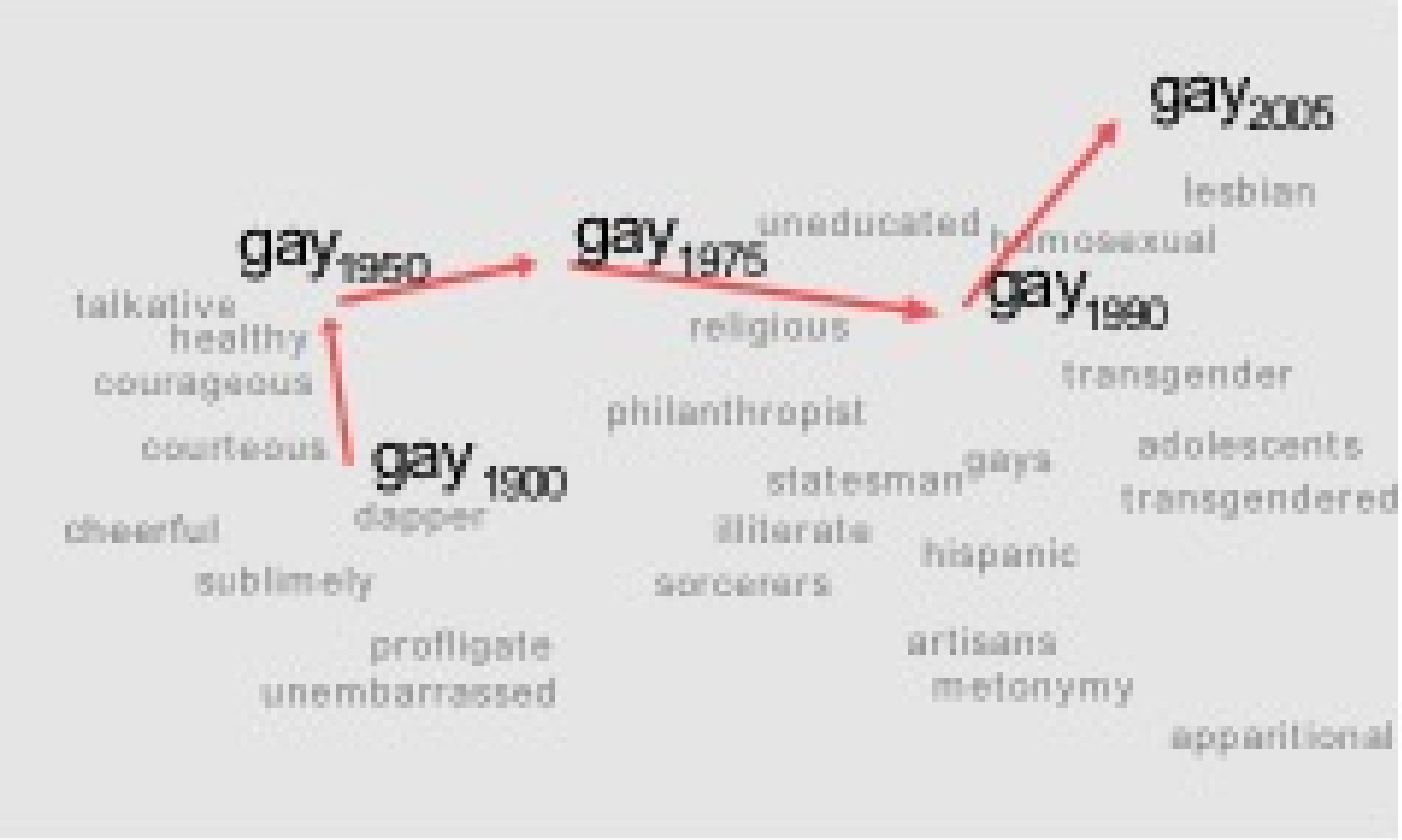
MAINFRAMES

Mainframes usually are referred those computers with fast, advanced processing capabilities that could perform by itself tasks that may require a lot of Personal Computers (PC) Machines. Usually mainframes would have lots of RAMs, very large secondary storage devices, and very fast processors to cater for the needs of those computers under its service.

Due to the advanced components mainframes have, these computers have the capability of running multiple large applications required by most enterprises, which is one of its advantage. Mainframes are also suitable to cater for those applications or files that are of very large demand by its users (clients). Examples of these include the large online shopping websites -i.e. : Ebay, Amazon, Microsoft, etc.

Semantic change over time

Kulkarni, Al-Rfou, Perozzi, Skiena 2015



Distributional word similarity

- Example:

```
A bottle of tesgüino is on the table
Everybody likes tesgüino
Tesgüino makes you drunk
We make tesgüino out of corn.
```

- From context words humans can guess *tesgüino* means
  - an alcoholic beverage like **beer**
- Intuition for algorithm:
  - Two words are similar if they have similar word contexts.

Three kinds of vector models

Sparse vector representations

1. Mutual-information (互信息) weighted word co-occurrence matrices

Dense vector representations:

2. Singular value decomposition (and Latent Semantic Analysis 潜在语义分析)
3. Neural-network-inspired models (skip-grams, CBOW, BERT) (word embeddings seen earlier)

Shared intuition

- Model the meaning of a word by “embedding” in a vector space
- The meaning of a word is a vector of numbers
  - Vector models are also called “**embeddings**”
- Contrast: word meaning is represented in many computational linguistic applications by a vocabulary index (“word number 545”)

Term-document (词项-文档) matrix

- Each cell: count of term t in a document d: $\mathrm{tf}_{t,d}$
  - Each document is a count vector in $\mathbb{N}^{|V|}$: a column below

	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	1	1	8	15
soldier	2	2	12	36
fool	37	58	1	5
clown	6	117	0	0

Term-document matrix

- Two documents are similar if their vectors are similar

	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	1	1	8	15
soldier	2	2	12	36
fool	37	58	1	5
clown	6	117	0	0

The words in a term-document matrix

- Each word is a count vector in $\mathbb{N}^D$: a row below

	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	1	1	8	15
soldier	2	2	12	36
fool	37	58	1	5
clown	6	117	0	0

The words in a term-document matrix

- Two **words** are similar if their vectors are similar

	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	1	1	8	15
soldier	2	2	12	36
fool	37	58	1	5
clown	6	117	0	0

Term-context matrix for word similarity

- Two **words** are similar in meaning if their context vectors are similar

	aardvark	computer	data	pinch	result	sugar	…
apricot	0	0	0	1	0	1	
pineapple	0	0	0	1	0	1	
digital	0	2	1	0	1	0	
information	0	1	6	0	4	0	

The word-word or word-context matrix

- Instead of entire documents, use smaller contexts
  - Paragraph
  - Window of $\pm$ 4 words
- A word is now defined by a vector over counts of context words
- Instead of each vector being of length D
- Each vector is now of length |V|
- The word-word matrix is |V|x|V|

Word-word matrix
Sample contexts ± 7 words

sugar, a sliced lemon, a tablespoonful of **apricot** preserve or jam, a pinch each of,
their enjoyment. Cautiously she sampled her first **pineapple** and another fruit whose taste she likened
well suited to programming on the digital **computer**. In finding the optimal R-stage policy from
for the purpose of gathering data and **information** necessary for the study authorized in the

	aardvark	computer	data	pinch	result	sugar	…
apricot	0	0	0	1	0	1	
pineapple	0	0	0	1	0	1	
digital	0	2	1	0	1	0	
information	0	1	6	0	4	0	
…	…						

Word-word matrix

- We showed only 4x6, but the real matrix is 50,000 x 50,000
  - So it's very **sparse**
    - Most values are 0.
  - That's OK, since there are lots of efficient algorithms for sparse matrices.
- The size of windows depends on your goals
  - The shorter the windows , the more **syntactic** the representation
    ± 1-3 very syntactic
  - The longer the windows, the more **semantic** the representation
    ± 4-10 more semantic

2 kinds of co-occurrence between 2 words

(Schütze and Pedersen, 1993)

- First-order co-occurrence (**syntagmatic** 组合 **association**):
  - They are typically nearby each other.
  - *wrote* is a first-order associate of *book* or *poem*.
- Second-order co-occurrence (**paradigmatic** 聚合 **association**):
  - They have similar neighbors.
  - *wrote* is a second- order associate of words like *said* or *remarked*.

Problem with raw counts

- Raw word frequency is not a great measure of association between words
  - It's very skewed
    - "the" and "of" are very frequent, but maybe not the most discriminative
- We'd rather have a measure that asks whether a context word is **particularly informative** about the target word.
  - Positive Pointwise Mutual Information (PPMI) 点间互信息

Pointwise Mutual Information

Pointwise mutual information:

Do events x and y co-occur more than if they were independent?

$$\mathrm{PMI}(X,Y) = \log_2 \frac{P(x,y)}{P(x)P(y)}$$

PMI between two words: (Church & Hanks 1989)

Do words x and y co-occur more than if they were independent?

$$\mathrm{PMI}(word_1, word_2) = \log_2 \frac{P(word_1, word_2)}{P(word_1)P(word_2)}$$

Positive Pointwise Mutual Information

- PMI ranges from $-\infty$ to $+\infty$
- But the negative values are problematic
  - Things are co-occurring **less than** we expect by chance
  - Unreliable without enormous corpora
    - Imagine w1 and w2 whose probability is each 10^{-6}
    - Hard to be sure p(w1,w2) is significantly different than 10^{-12}
- So we just replace negative PMI values by 0
- Positive PMI (PPMI) between word1 and word2:

$$\text{PPMI}(word_1, word_2) = \max\left(\log_2 \frac{P(word_1, word_2)}{P(word_1)P(word_2)}, 0\right)$$

Computing PPMI on a term-context matrix

- Matrix F with W rows (words) and C columns (contexts)
- f_{ij} is # of times w_i occurs in context c_j

$$p_{ij} = \frac{f_{ij}}{\sum_{i=1}^{W}\sum_{j=1}^{C} f_{ij}} \qquad p_{i*} = \frac{\sum_{j=1}^{C} f_{ij}}{\sum_{i=1}^{W}\sum_{j=1}^{C} f_{ij}} \qquad p_{*j} = \frac{\sum_{i=1}^{W} f_{ij}}{\sum_{i=1}^{W}\sum_{j=1}^{C} f_{ij}}$$

	aardvark	computer	data	pinch	result	sugar
apricot	0	0	0	1	0	1
pineapple	0	0	0	1	0	1
digital	0	2	1	0	1	0
information	0	1	6	0	4	0

$$pmi_{ij} = \log_2 \frac{p_{ij}}{p_{i*}p_{*j}} \qquad ppmi_{ij} = \begin{cases} pmi_{ij} & \text{if } pmi_{ij} > 0 \\ 0 & \text{otherwise} \end{cases}$$

$$p_{ij} = \frac{f_{ij}}{\sum_{i=1}^{W}\sum_{j=1}^{C} f_{ij}}$$

Count(w,context)

	computer	data	pinch	result	sugar
apricot	0	0	1	0	1
pineapple	0	0	1	0	1
digital	2	1	0	1	0
information	1	6	0	4	0

p(w=information,c=data) = 6/19 = .32

p(w=information) = 11/19 = .58

p(c=data) = 7/19 = .37

$$p(w_i) = \frac{\sum_{j=1}^{C} f_{ij}}{N} \qquad p(c_j) = \frac{\sum_{i=1}^{W} f_{ij}}{N}$$

	p(w,context)					p(w)
	computer	data	pinch	result	sugar	
apricot	0.00	0.00	0.05	0.00	0.05	0.11
pineapple	0.00	0.00	0.05	0.00	0.05	0.11
digital	0.11	0.05	0.00	0.05	0.00	0.21
information	0.05	0.32	0.00	0.21	0.00	0.58
p(context)	0.16	0.37	0.11	0.26	0.11	

$$pmi_{ij} = \log_2 \frac{p_{ij}}{p_{i*}p_{*j}}$$

	p(w,context)					p(w)
	computer	data	pinch	result	sugar	
apricot	0.00	0.00	0.05	0.00	0.05	0.11
pineapple	0.00	0.00	0.05	0.00	0.05	0.11
digital	0.11	0.05	0.00	0.05	0.00	0.21
information	0.05	0.32	0.00	0.21	0.00	0.58
p(context)	0.16	0.37	0.11	0.26	0.11	

- pmi(information,data) = $\log_2$ (.32 /(.37*.58))= .57

PPMI(w,context)

	computer	data	pinch	result	sugar
apricot	-	-	2.25	-	2.25
pineapple	-	-	2.25	-	2.25
digital	1.66	0.00	-	0.00	-
information	0.00	0.57	-	0.47	-

Weighting PMI

- PMI is biased toward infrequent events
  - Very rare words have very high PMI values
- Two solutions:
  - Give rare words slightly higher probabilities
  - Use add-one smoothing (which has a similar effect)

Giving rare context words slightly higher probability

- Raise the context probabilities to $\alpha = 0.75$:

$$\text{PPMI}_\alpha(w,c) = \max(\log_2 \frac{P(w,c)}{P(w)P_\alpha(c)}, 0)$$

$$P_\alpha(c) = \frac{count(c)^\alpha}{\sum_c count(c)^\alpha}$$

- This helps because $P_\alpha(c) > P(c)$ for rare c
- Consider two events, P(a) = .99 and P(b)=.01
- $P_\alpha(a) = \frac{.99^{.75}}{.99^{.75}+.01^{.75}} = .97$ $P_\alpha(b) = \frac{.01^{.75}}{.01^{.75}+.01^{.75}} = .03$

Use Laplace (add-1) smoothing

- General concept is to add a small number
- Add-2 is quite popular

Add-2 Smoothed Count(w,context

	computer	data	pinch	result	sugar
apricot	2	2	3	2	3
pineapple	2	2	3	2	3
digital	4	3	2	3	2
information	3	8	2	6	2

p(w,context) [add-2] **p(w)**

	computer	data	pinch	result	sugar	
apricot	0.03	0.03	0.05	0.03	0.05	0.20
pineapple	0.03	0.03	0.05	0.03	0.05	0.20
digital	0.07	0.05	0.03	0.05	0.03	0.24
information	0.05	0.14	0.03	0.10	0.03	0.36
p(context)	0.19	0.25	0.17	0.22	0.17	

PPMI versus add-2 smoothed PPMI

PPMI(w,context)

	computer	data	pinch	result	sugar
apricot	-	-	2.25	-	2.25
pineapple	-	-	2.25	-	2.25
digital	1.66	0.00	-	0.00	-
information	0.00	0.57	-	0.47	-

PPMI(w,context) [add-2]

	computer	data	pinch	result	sugar
apricot	0.00	0.00	0.56	0.00	0.56
pineapple	0.00	0.00	0.56	0.00	0.56
digital	0.62	0.00	0.00	0.00	0.00
information	0.00	0.58	0.00	0.37	0.00

Measuring similarity

- Given 2 target words *v* and *w*
- We'll need a way to measure their similarity.
- Most measure of vectors similarity are based on the:
- **Dot product** (点积) or **inner product** (内积) from linear algebra

$$\text{dot-product}(\vec{v},\vec{w}) = \vec{v}\cdot\vec{w} = \sum_{i=1}^{N} v_i w_i = v_1 w_1 + v_2 w_2 + \ldots + v_N w_N$$

  - High when two vectors have large values in same dimensions.
  - Low (in fact 0) for **orthogonal** (正交) **vectors** with zeros in complementary distribution

Problem with dot product

$$\text{dot-product}(\vec{v},\vec{w}) = \vec{v}\cdot\vec{w} = \sum_{i=1}^{N} v_i w_i = v_1 w_1 + v_2 w_2 + \ldots + v_N w_N$$

- Dot product is longer if the vector is longer. Vector length:

$$|\vec{v}| = \sqrt{\sum_{i=1}^{N} v_i^2}$$

- Vectors are longer if they have higher values in each dimension
- That means more frequent words will have higher dot products
- That's bad: we don't want a similarity metric to be sensitive to word frequency

Solution: cosine

- Just divide the dot product by the length of the two vectors!

$$\frac{\vec{a} \cdot \vec{b}}{|\vec{a}||\vec{b}|}$$

- This turns out to be the cosine of the angle between them!

$$\vec{a} \cdot \vec{b} = |\vec{a}||\vec{b}| \cos\theta$$

$$\frac{\vec{a} \cdot \vec{b}}{|\vec{a}||\vec{b}|} = \cos\theta$$

Cosine for computing similarity

Dot product

Unit vectors

$$\cos(\vec{v},\vec{w}) = \frac{\vec{v}\bullet\vec{w}}{|\vec{v}||\vec{w}|} = \frac{\vec{v}}{|\vec{v}|}\bullet\frac{\vec{w}}{|\vec{w}|} = \frac{\sum_{i=1}^{N} v_i w_i}{\sqrt{\sum_{i=1}^{N} v_i^2}\sqrt{\sum_{i=1}^{N} w_i^2}}$$

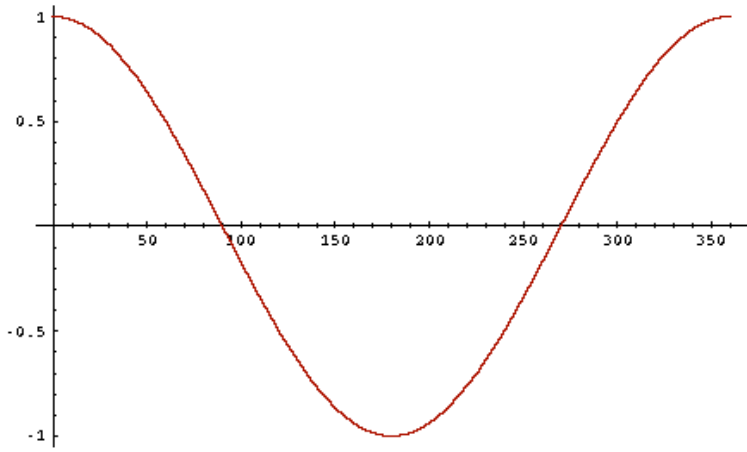
v_i is the PPMI value for word v in context i

w_i is the PPMI value for word w in context i

$\cos(\vec{v},\vec{w})$ is the cosine similarity of $\vec{v}$ and $\vec{w}$

Cosine as a similarity metric

- -1: vectors point in opposite directions
- +1: vectors point in same directions
- 0: vectors are orthogonal



- Raw frequency or PPMI are non-negative, so cosine range 0-1

$$\cos(\vec{v},\vec{w}) = \frac{\vec{v}\bullet\vec{w}}{|\vec{v}||\vec{w}|} = \frac{\vec{v}}{|\vec{v}|}\bullet\frac{\vec{w}}{|\vec{w}|} = \frac{\sum_{i=1}^{N} v_i w_i}{\sqrt{\sum_{i=1}^{N} v_i^2}\sqrt{\sum_{i=1}^{N} w_i^2}}$$

	large	data	computer
apricot	2	0	0
digital	0	1	2
information	1	6	1

Which pair of words is more similar?

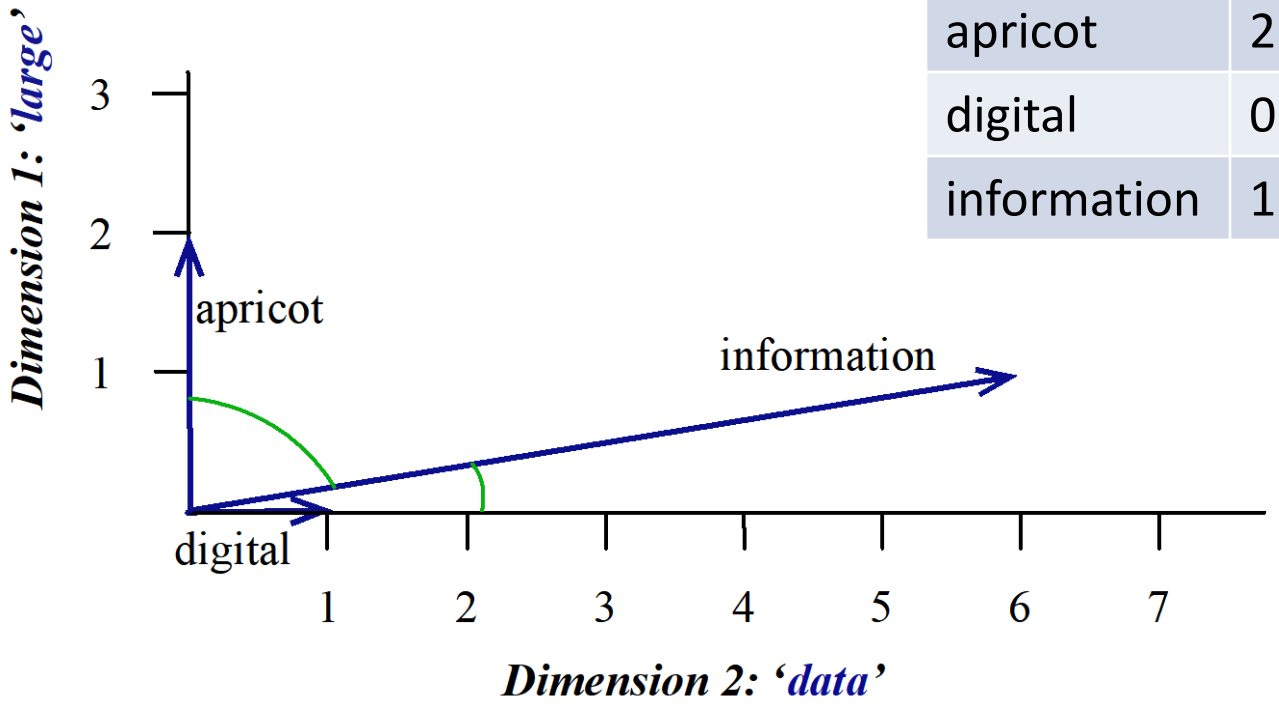
cosine(apricot,information) = $\frac{2+0+0}{\sqrt{2+0+0}\sqrt{1+36+1}} = \frac{2}{\sqrt{2}\sqrt{38}} = .23$

cosine(digital,information) = $\frac{0+6+2}{\sqrt{0+1+4}\sqrt{1+36+1}} = \frac{8}{\sqrt{38}\sqrt{5}} = .58$

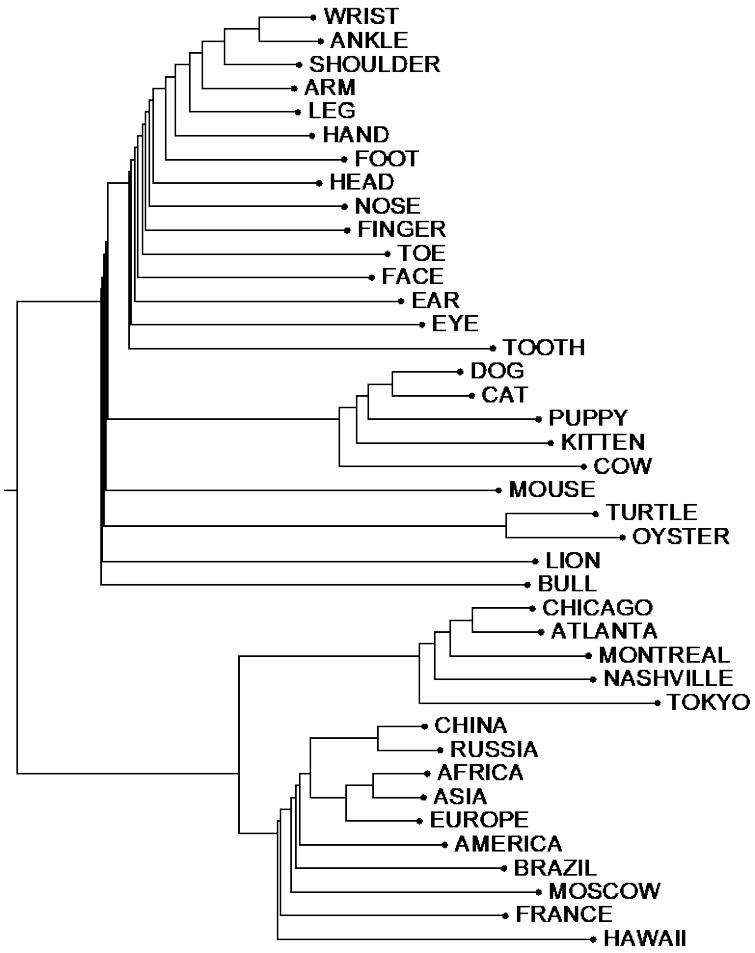
cosine(apricot,digital) = $\frac{0+0+0}{\sqrt{1+0+0}\sqrt{0+1+4}} = 0$

Visualizing vectors and angles

	large	data
apricot	2	0
digital	0	1
information	1	6



Clustering vectors to visualize similarity in co-occurrence matrices



Rohde et al. (2006)

Other possible similarity measures

$$\mathrm{sim}_{\mathrm{cosine}}(\vec{v},\vec{w}) = \frac{\vec{v}\cdot\vec{w}}{|\vec{v}||\vec{w}|} = \frac{\sum_{i=1}^{N} v_i \times w_i}{\sqrt{\sum_{i=1}^{N} v_i^2}\sqrt{\sum_{i=1}^{N} w_i^2}}$$

$$\mathrm{sim}_{\mathrm{Jaccard}}(\vec{v},\vec{w}) = \frac{\sum_{i=1}^{N} \min(v_i,w_i)}{\sum_{i=1}^{N} \max(v_i,w_i)}$$

$$\mathrm{sim}_{\mathrm{Dice}}(\vec{v},\vec{w}) = \frac{2\times\sum_{i=1}^{N} \min(v_i,w_i)}{\sum_{i=1}^{N} (v_i+w_i)}$$

$$\mathrm{sim}_{\mathrm{JS}}(\vec{v}||\vec{w}) = D(\vec{v}|\frac{\vec{v}+\vec{w}}{2}) + D(\vec{w}|\frac{\vec{v}+\vec{w}}{2})$$

Using syntax to define a word's context

- Zellig Harris (1968)

  "The meaning of entities, and the meaning of grammatical relations among them, is related to the restriction of combinations of these entities relative to other entities"

- **Two words are similar if they have similar syntactic contexts**
  - **Duty** and **responsibility** have similar syntactic distribution:

Modified by adjectives	additional, administrative, assumed, collective, congressional, constitutional …
Objects of verbs	assert, assign, assume, attend to, avoid, become, breach..

Co-occurrence vectors based on syntactic dependencies

Dekang Lin, 1998 “Automatic Retrieval and Clustering of Similar Words”

- Each dimension: a context word in one of R grammatical relations
  - Subject-of- “absorb”
- Instead of a vector of */V/* features, a vector of *R/V/*
- Example: counts for the word *cell* :

	subj-of, absorb	subj-of, adapt	subj-of, behave	...	pobj-of, inside	pobj-of, into	...	nmod-of, abnormality	nmod-of, anemia	nmod-of, architecture	...	obj-of, attack	obj-of, call	obj-of, come from	obj-of, decorate	...	nmod, bacteria	nmod, body	nmod, bone marrow
cell	1	1	1		16	30		3	8	1		6	11	3	2		3	2	2

Alternative to PPMI for measuring association

- **tf-idf** (that's a hyphen not a minus sign)
- The combination of two factors
  - **Term frequency** (Luhn 1957): frequency of the word (can be logged)
  - **Inverse document frequency** (IDF) (Sparck Jones 1972)
    - N is the total number of documents
    - df_i = "document frequency of word *i*"
    - = # of documents with word *I*
  - w_{ij} = *word* i *in document* j

$$w_{ij}=tf_{ij}\,idf_i$$

$$\text{idf}_i = \log\left(\frac{N}{df_i}\right)$$

tf-idf not generally used for word-word similarity

- But is by far the most common weighting when we are considering the relationship of words to documents

Evaluating similarity

- Extrinsic (task-based, end-to-end) Evaluation:
  - Question Answering
  - Spell Checking
  - Essay grading
- Intrinsic Evaluation:
  - Correlation between algorithm and human word similarity ratings
    - Wordsim353: 353 noun pairs rated 0-10. *sim(plane,car)=5.77*
  - Taking TOEFL multiple-choice vocabulary tests
    - Levied is closest in meaning to:

imposed, believed, requested, correlated

Sparse versus dense vectors

- PPMI vectors are
  - **long** (length |V|= 20,000 to 50,000)
  - **sparse** (most elements are zero)
- Alternative: learn vectors which are
  - **short** (length 200-1000)
  - **dense** (most elements are non-zero)

Why Dense Vectors?

- Short vectors may be easier to use as features in machine learning (less weights to tune)
- Dense vectors may generalize better than storing explicit counts
- They may do better at capturing synonymy (同义):
  - *car* and *automobile* are synonyms; but are represented as distinct dimensions; this fails to capture similarity between a word with *car* as a neighbor and a word with *automobile* as a neighbor

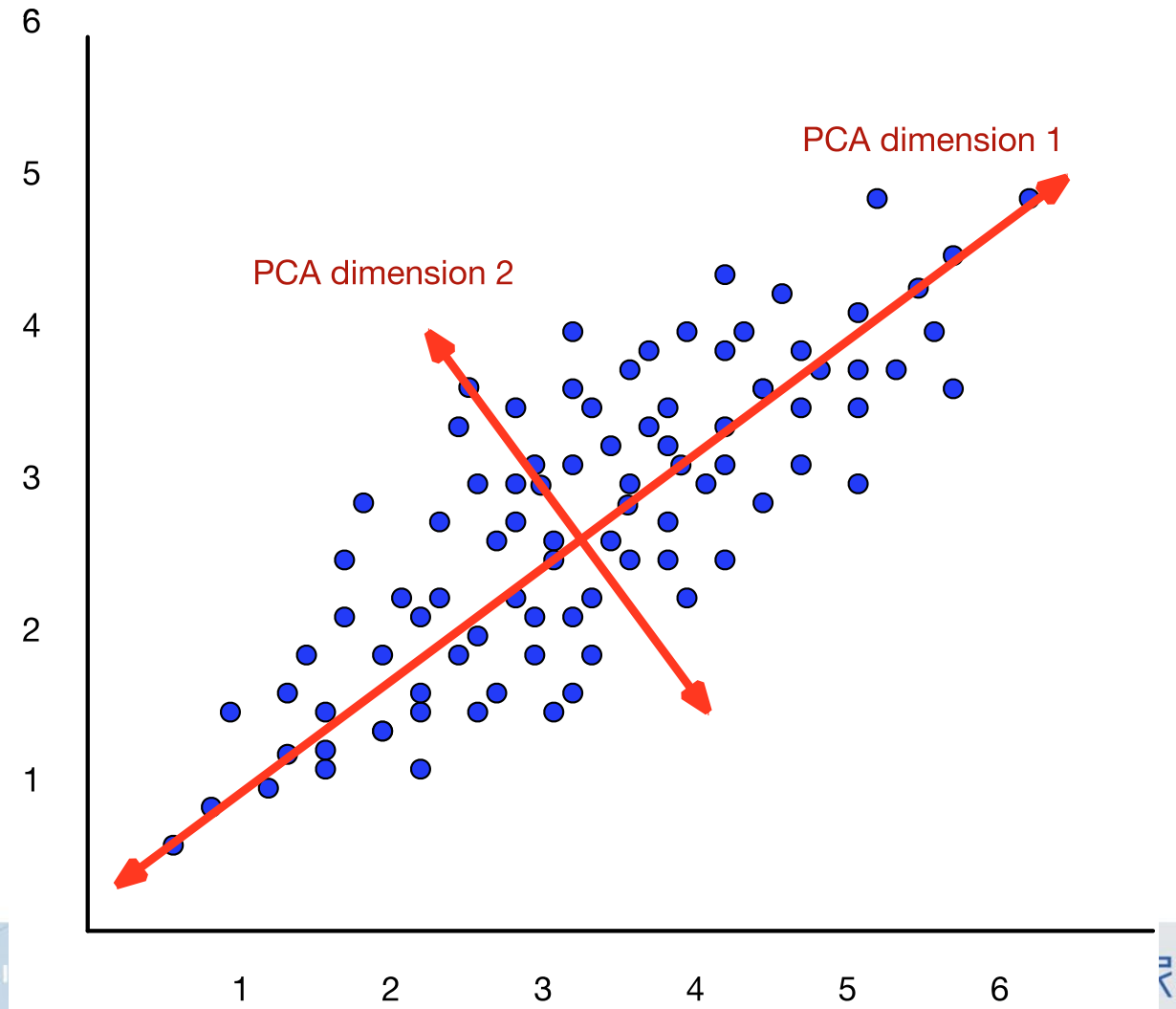
Two methods for getting short dense vectors

- Singular Value Decomposition (SVD)
  - A special case of this is called LSA – Latent Semantic Analysis
- "Neural Language Model"-inspired predictive models
  - skip-grams and CBOW
  - Pre-trained language models, e.g., BERT

Dense Vectors via SVD

- Approximate an N-dimensional dataset using fewer dimensions
- By first rotating the axes into a new space
- In which the highest order dimension captures the most variance in the original dataset
- And the next dimension captures the next most variance, etc.
- Many such (related) methods:
  - PCA – principle components analysis
  - Factor Analysis
  - SVD

Dimensionality reduction



Singular Value Decomposition

Any rectangular w x c matrix X equals the product of 3 matrices:

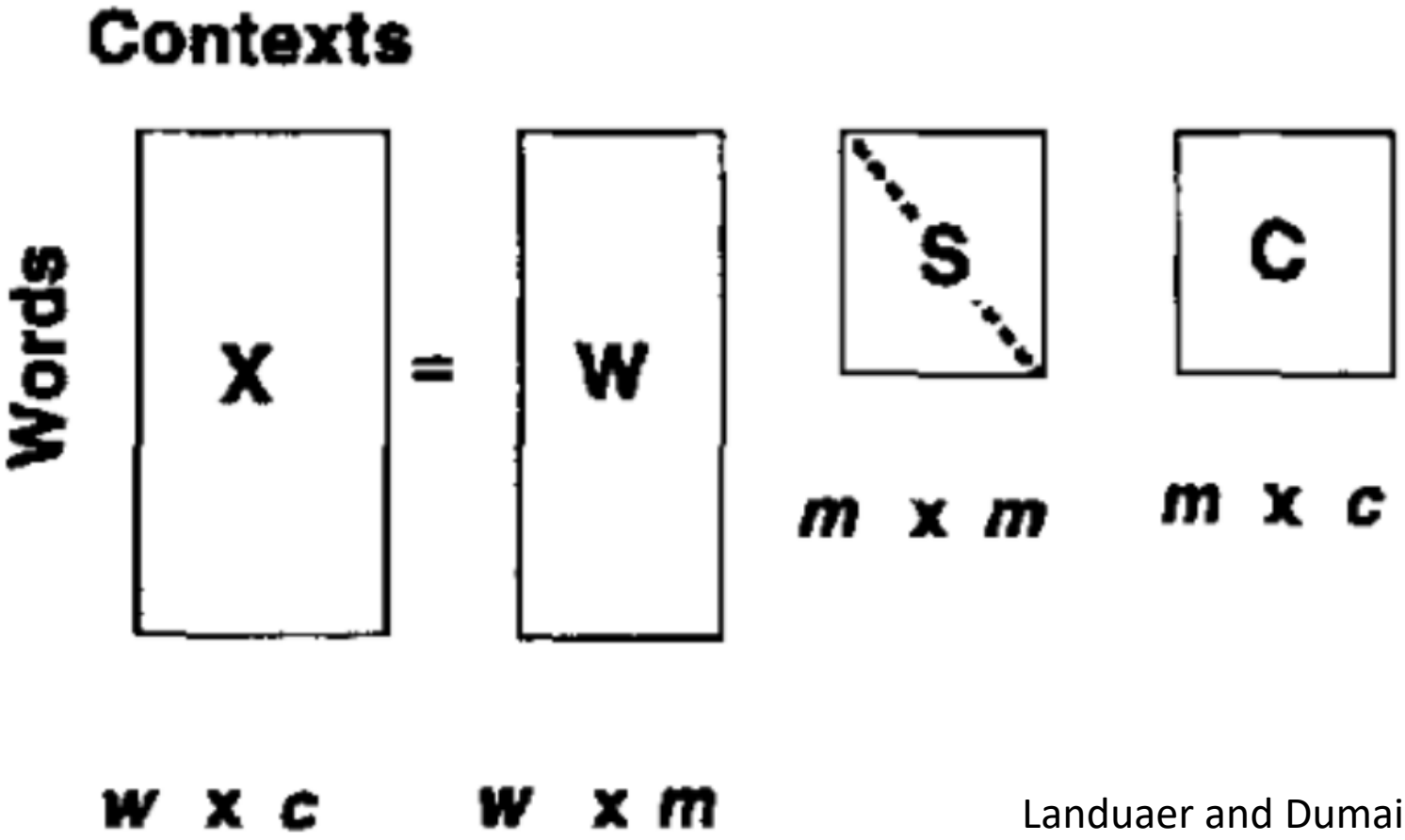
W: rows corresponding to original but m columns represents a dimension in a new latent space, such that

- M column vectors are orthogonal to each other
- Columns are ordered by the amount of variance in the dataset each new dimension accounts for

S: diagonal *m* x *m* matrix of **singular values** expressing the importance of each dimension.

C: columns corresponding to original but m rows corresponding to singular values

Singular Value Decomposition

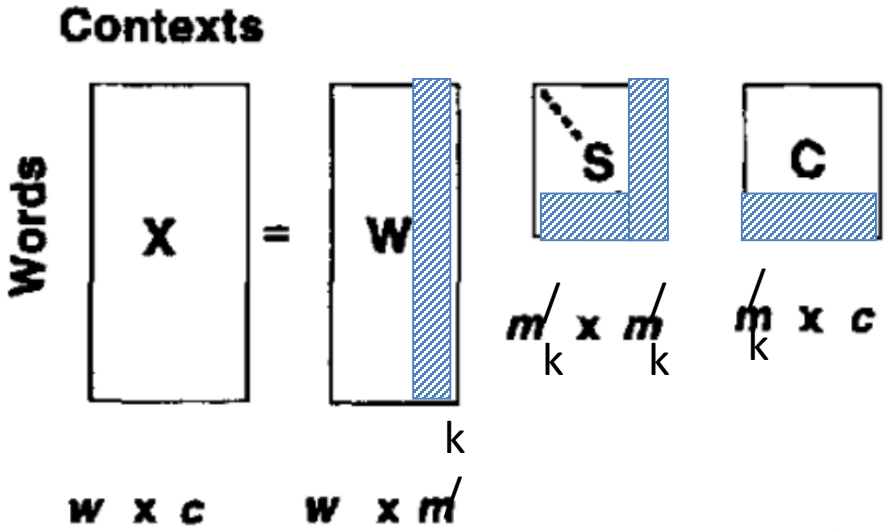


Landuaer and Dumais 1997

SVD to Term-Doc Matrix: Latent Semantic Analysis

Deerwester et al (1988)

- If instead of keeping all m dimensions, we just keep the top k singular values. Let's say 300.
- The result is a least-squares approximation to the original X
- But instead of multiplying, we'll just make use of W.
- Each row of W:
  - A k-dimensional vector
  - Representing word W



Landauer T K, Dumais S T. A solution to Plato's problem: The latent semantic analysis theory of acquisition, induction, and representation of knowledge[J]. Psychological review, 1997, 104(2): 211.

LSA more details

- 300 dimensions are commonly used
- The cells are commonly weighted by a product of two weights
  - Local weight: Log term frequency
  - Global weight: either idf or an entropy measure

Let's return to PPMI word-word matrices

- Can we apply SVD to them?

SVD applied to term-term matrix

$$\underset{|V|\times|V|}{\begin{bmatrix} X \end{bmatrix}} = \underset{|V|\times|V|}{\begin{bmatrix} W \end{bmatrix}} \underset{|V|\times|V|}{\begin{bmatrix} \sigma_1 & 0 & 0 & \dots & 0 \\ 0 & \sigma_2 & 0 & \dots & 0 \\ 0 & 0 & \sigma_3 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & \sigma_V \end{bmatrix}} \underset{|V|\times|V|}{\begin{bmatrix} C \end{bmatrix}}$$

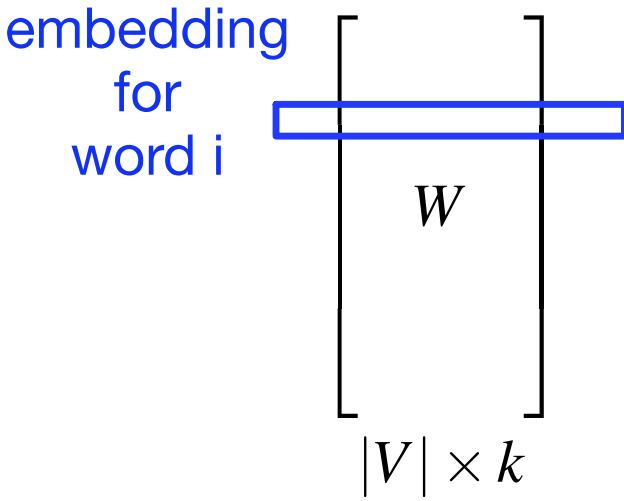
(I'm simplifying here by assuming the matrix has rank |V|)

Truncated SVD on term-term matrix

$$\underset{|V|\times|V|}{\begin{bmatrix} \\ X \\ \\ \end{bmatrix}} = \underset{|V|\times k}{\begin{bmatrix} \\ W \\ \\ \end{bmatrix}} \underset{k\times k}{\begin{bmatrix} \sigma_1 & 0 & 0 & \dots & 0 \\ 0 & \sigma_2 & 0 & \dots & 0 \\ 0 & 0 & \sigma_3 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & \sigma_k \end{bmatrix}} \underset{k\times|V|}{\begin{bmatrix} \\ C \\ \\ \end{bmatrix}}$$

Truncated SVD produces embeddings

- Each row of W matrix is a k-dimensional representation of each word *w*
- K might range from 50 to 1000
- Generally we keep the top k dimensions, but some experiments suggest that getting rid of the top 1 dimension or even the top 50 dimensions is helpful (Lapesa and Evert 2014)



Embeddings versus sparse vectors

- Dense SVD embeddings sometimes work better than sparse PPMI matrices at tasks like word similarity
  - Denoising: low-order dimensions may represent unimportant information
  - Truncation may help the models generalize better to unseen data.
  - Having a smaller number of dimensions may make it easier for classifiers to properly weight the dimensions for the task.
  - Dense models may do better at capturing higher order co-occurrence.

Prediction-based models: Another way to get dense vectors

- **Skip-gram** (Mikolov et al. 2013a) **CBOW** (Mikolov et al. 2013b)
- Learn embeddings as part of the process of word prediction.
- Train a neural network to predict neighboring words
  - Inspired by **neural net language models**.
  - In so doing, learn dense embeddings for the words in the training corpus.
- Advantages:
  - Fast, easy to train (much faster than SVD)
  - Available online in the `word2vec` package
  - Including sets of pretrained embeddings!

Skip-grams

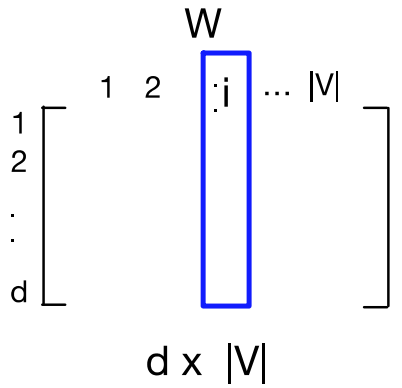
- Predict each neighboring word
  - in a context window of $2C$ words
  - from the current word.
- So for C=2, we are given word w_t and predicting these 4 words:

$$[w_{t-2}, w_{t-1}, w_{t+1}, w_{t+2}]$$

Skip-grams learn 2 embeddings for each w

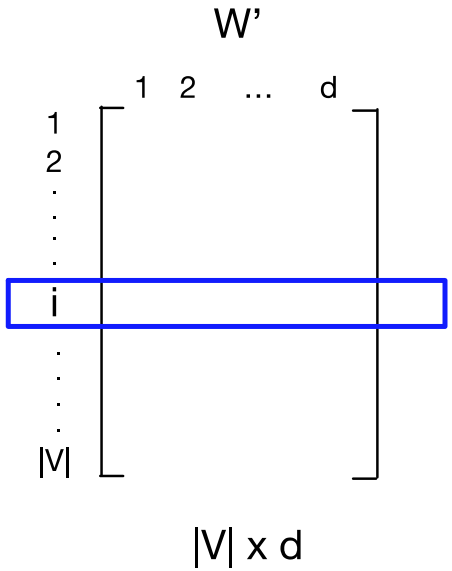
input embedding v_i in the input matrix W

- Column i of the input matrix W is the $1 \times d$ embedding v_i for word i in the vocabulary.



output embedding v', in output matrix W'

- Row i of the output matrix W is a $d \times 1$ vector embedding v'_i for word i in the vocabulary.

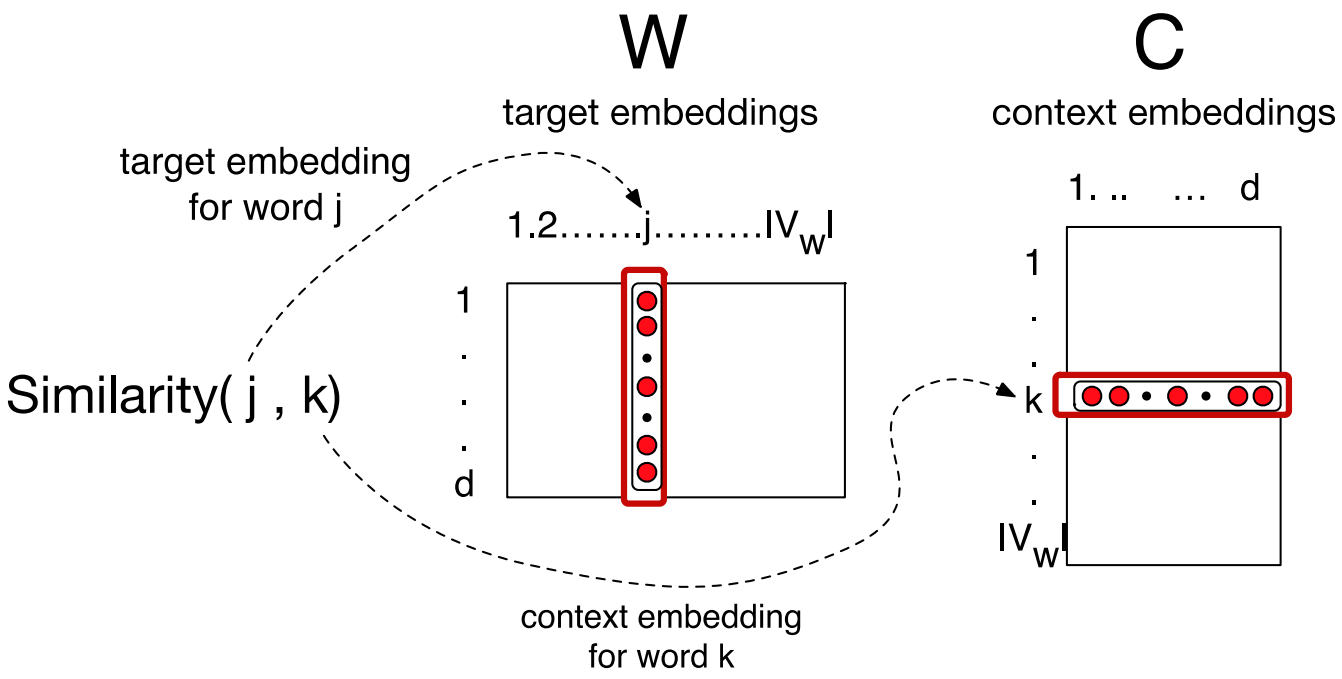


Setup

- Walking through corpus pointing at word $w(t)$, whose index in the vocabulary is j, so we'll call it w_j ($1 < j < |V|$).

- Let's predict $w(t+1)$, whose index in the vocabulary is k ($1 < k < |V|$). Hence our task is to compute $P(w_k|w_j)$.

Intuition: similarity as dot-product between a target vector and context vector



Similarity is computed from dot product

- Remember: two vectors are similar if they have a high dot product
  - Cosine is just a normalized dot product
- So:
  - Similarity(j,k) $\propto c_k \cdot v_j$
- We'll need to normalize to get a probability

Turning dot products into probabilities

- Similarity(j, k) $= c_k \cdot v_j$

- We use softmax to turn into probabilities

$$p(w_k|w_j) = \frac{exp(c_k \cdot v_j)}{\sum_{i \in |V|} exp(c_i \cdot v_j)}$$

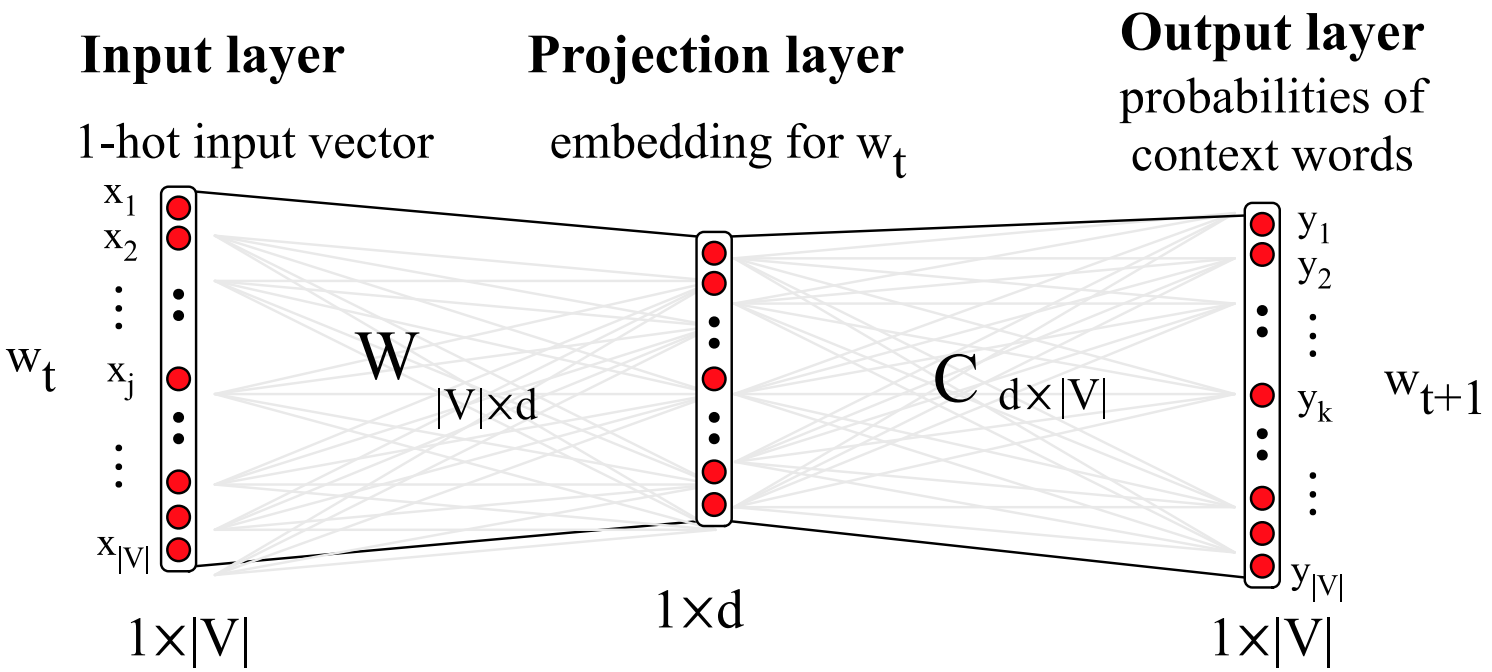
Embeddings from W and W’

- Since we have two embeddings, v_j and c_j for each word w_j
- We can either:
  - Just use v_j
  - Sum them
  - Concatenate them to make a double-length embedding

Learning

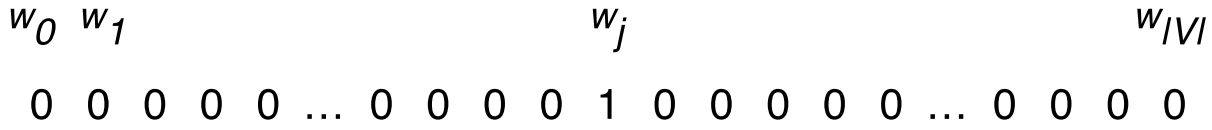
- Start with some initial embeddings (e.g., random)
- iteratively make the embeddings for a word
  - more like the embeddings of its neighbors
  - less like the embeddings of other words

Visualizing W and C as a network for doing error backprop



One-hot vectors

- A vector of length |V|
- 1 for the target word and 0 for other words
- So if “popsicle” is vocabulary word 5
- The **one-hot vector** is
- [0,0,0,0,1,0,0,0,0…….0]



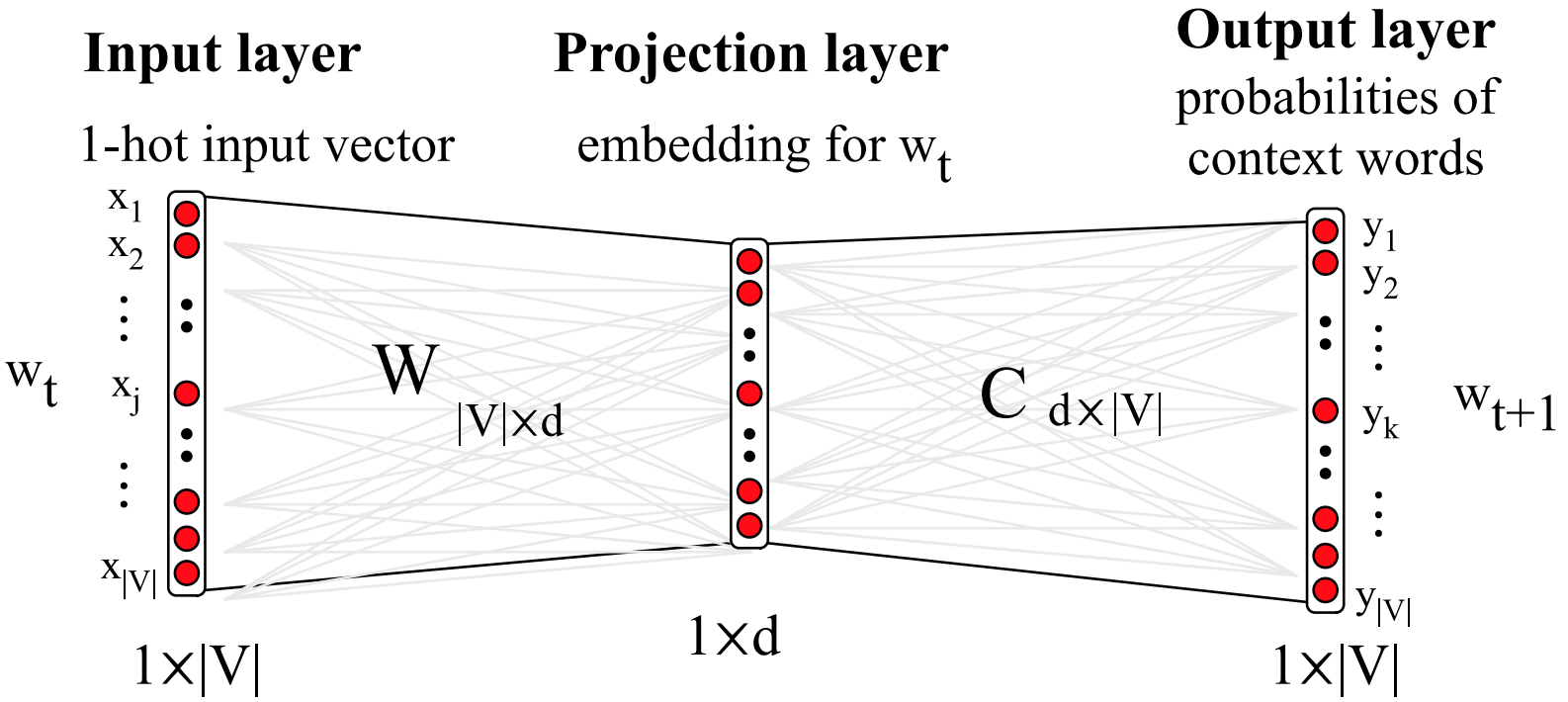
Skip-gram

$o = Ch$

$o_k = c_k h$

$h = v_j$

$o_k = c_k \cdot v_j$



Problem with the softmax

- The denominator: have to compute over every word in vocab

$$p(w_k|w_j) = \frac{exp(c_k \cdot v_j)}{\sum_{i \in |V|} exp(c_i \cdot v_j)}$$

- Instead: just sample a few of those negative words

Goal in learning

- Make the word like the context words

```
lemon,  a [tablespoon of apricot preserves or] jam
           c1         c2    w     c3       c4
```

$$\sigma(x) = \frac{1}{1+e^{x}}$$

- We want this to be high:

$$\sigma(c1 \cdot w) + \sigma(c2 \cdot w) + \sigma(c3 \cdot w) + \sigma(c4 \cdot w)$$

- And not like *k* randomly selected "noise words"

```
[cement metaphysical dear coaxial    apricot attendant whence forever puddle]
 n1     n2           n3   n4              n5        n6      n7       n8
```

- We want this to be low:

$$\sigma(n1 \cdot w) + \sigma(n2 \cdot w) + \ldots + \sigma(n8 \cdot w)$$

Skipgram with negative sampling: Loss function

$$\log \sigma(c \cdot w) + \sum_{i=1}^{\kappa} \mathbb{E}_{w_i \sim p(w)} \left[\log \sigma(-w_i \cdot w) \right]$$

Relation between skipgrams and PMI!

- If we multiply WW^T
- We get a |V|x|V| matrix M, each entry m_{ij} corresponding to some association between input word i and output word j
- Levy and Goldberg (2014b) show that skip-gram reaches its optimum just when this matrix is a shifted version of PMI:

$$WW^T = M^{\text{PMI}} - \log k$$

- So skip-gram is implicitly factoring a shifted version of the PMI matrix into the two embedding matrices.

Properties of embeddings

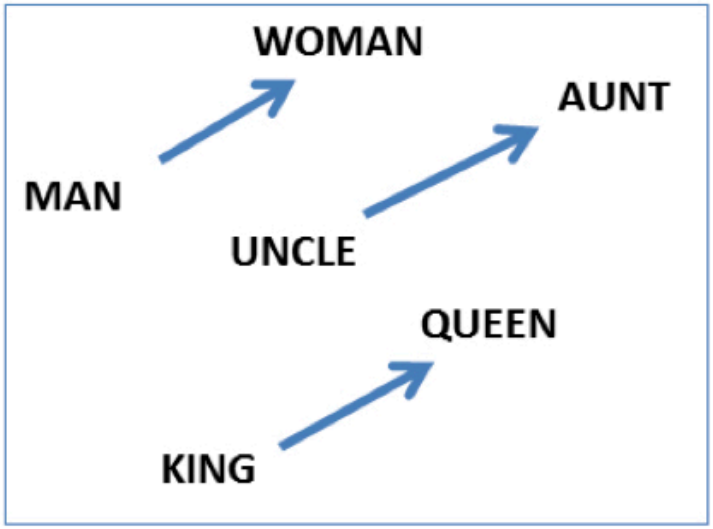
- Nearest words to some embeddings (Mikolov et al. 2013)

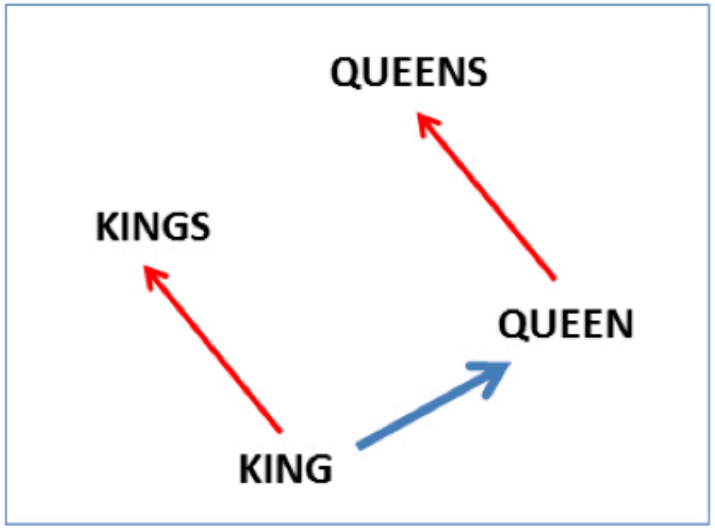
target:	Redmond	Havel	ninjutsu	graffiti	capitulate
	Redmond Wash.	Vaclav Havel	ninja	spray paint	capitulation
	Redmond Washington	president Vaclav Havel	martial arts	grafitti	capitulated
	Microsoft	Velvet Revolution	swordsmanship	taggers	capitulating

Embeddings capture relational meaning!

vector(*'king'*) - vector(*'man'*) + vector(*'woman'*) ≈ vector('queen')

vector(*'Paris'*) - vector(*'France'*) + vector(*'Italy'*) ≈ vector('Rome')





Contextual Representations

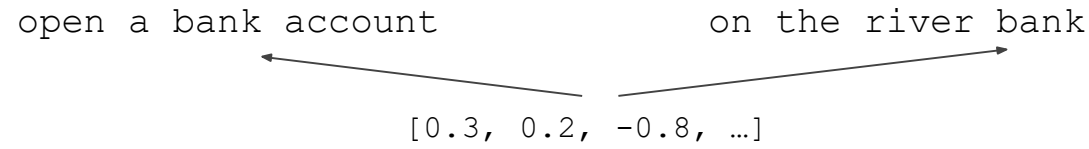
- Word embeddings are the basis of deep learning for NLP

- Word embeddings (`word2vec`, `GloVe`) are often *pre-trained* on text corpus from co-occurrence statistics

Slides from Jacob Devlin

Contextual Representations

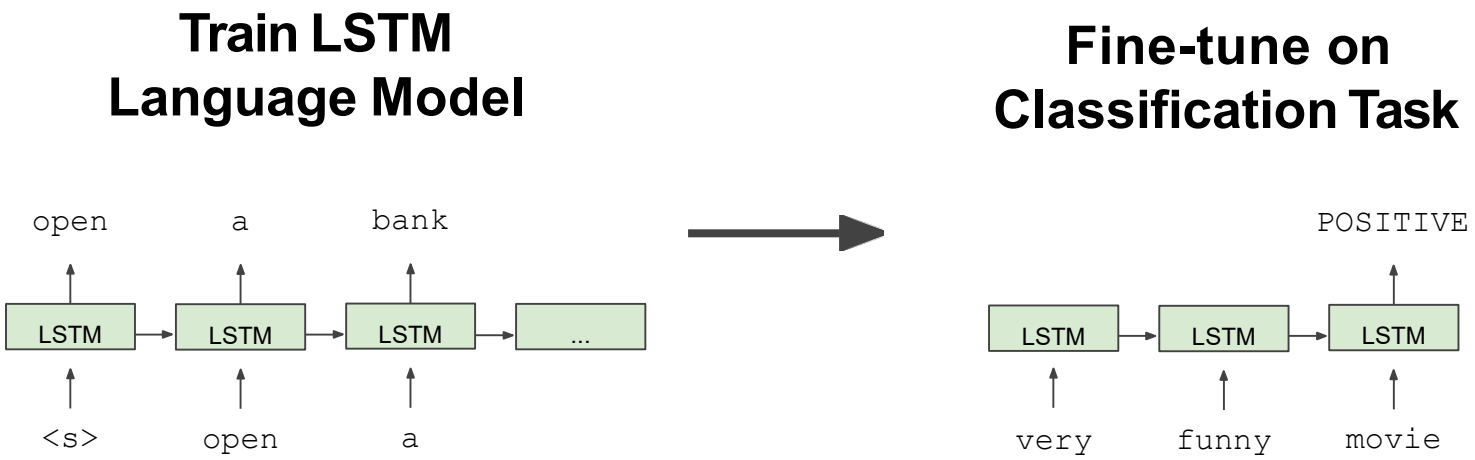
- **Problem**: Word embeddings are applied in a context free manner



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

History of Contextual Representations

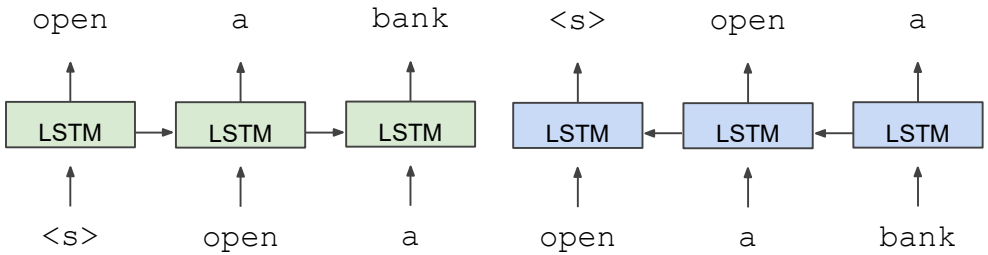
- *Semi-Supervised Sequence Learning*, Google, 2015



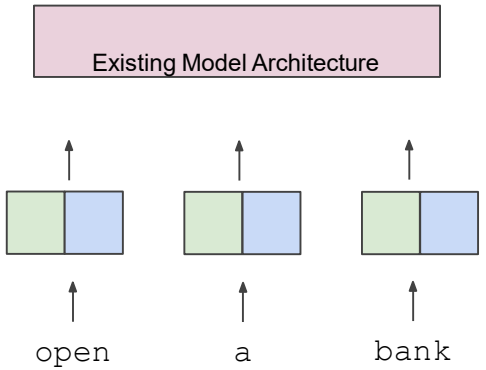
History of Contextual Representations

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

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

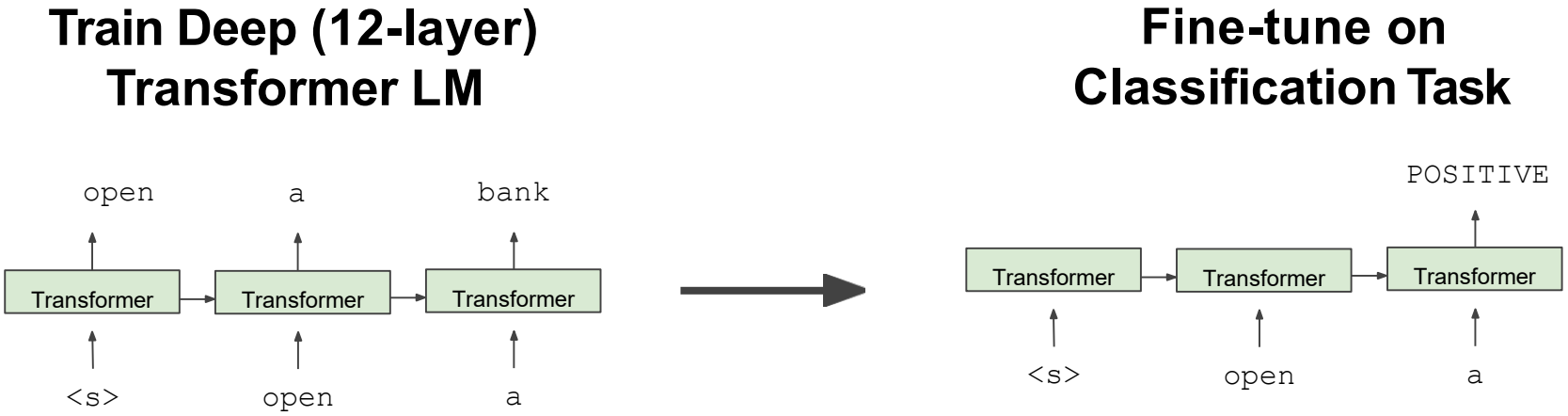


Apply as "Pre-trained Embeddings"



History of Contextual Representations

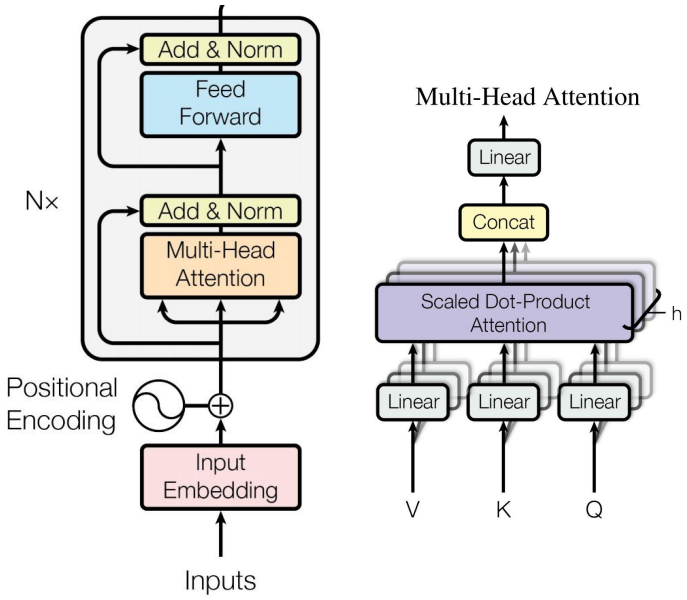
- *Improving Language Understanding by Generative Pre-Training*, OpenAI, 2018



Model Architecture

Transformer encoder

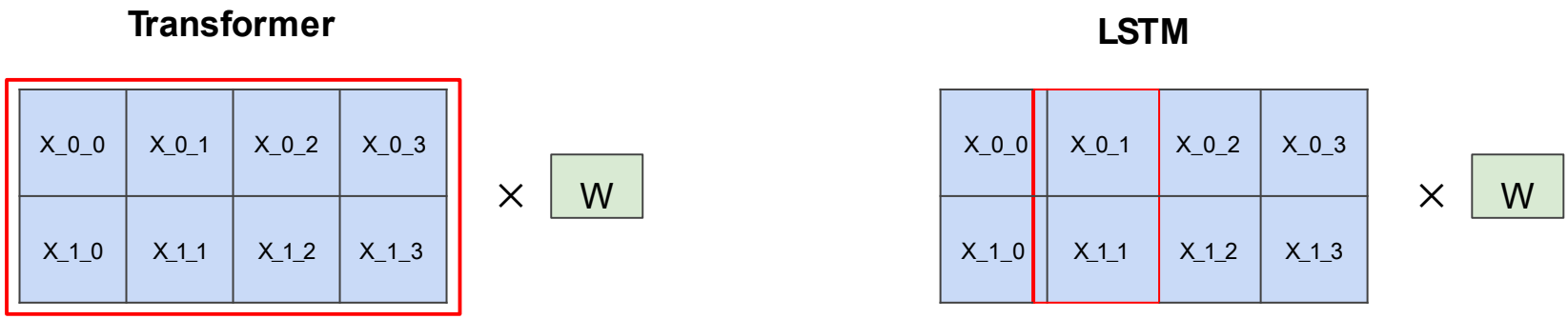
- Multi-headed self attention
  - Models context
- Feed-forward layers
  - Computes non-linear hierarchical features
- Layer norm and residuals
  - Makes training deep networks healthy
- Positional embeddings
  - Allows model to learn relative positioning



Model Architecture

- Empirical advantages of Transformer vs. LSTM:

1. Self-attention == no locality bias
   - Long-distance context has “equal opportunity”
2. Single multiplication per layer == efficiency on TPU
   - Effective batch size is number of *words*, not *sequences*



Problem with Previous Methods

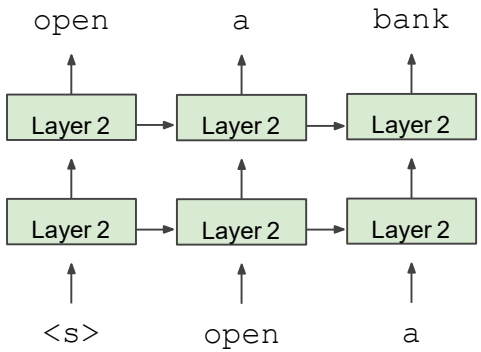
- **Problem**: Language models only use left context *or* right context, but language understanding is bidirectional.
- Why are LMs unidirectional?
- Reason 1: Directionality is needed to generate a well-formed probability distribution.
  - We don't care about this.
- Reason 2: Words can "see themselves" in a bidirectional encoder.

->BERT (Bidirectional Encoder Representations from Transformers)

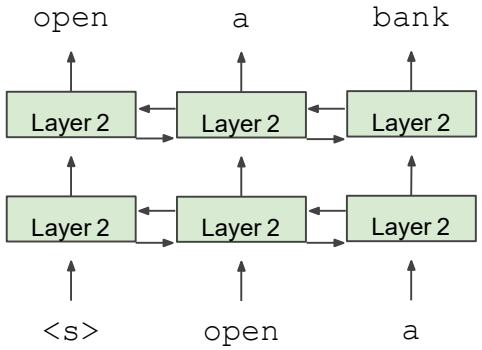
Google, 2018

Unidirectional vs. Bidirectional Models

Unidirectional context
Build representation incrementally

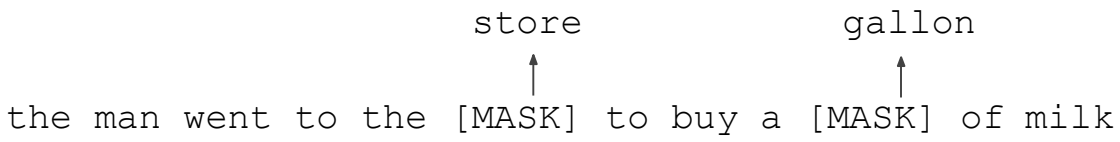


Bidirectional context
Words can “see themselves”



Masked LM

- **Solution**: Mask out *k*% of the input words, and then predict the masked words
  - We always use *k* = 15%



- Too little masking: Too expensive to train
- Too much masking: Not enough context

Masked LM

- Problem: Mask token never seen at fine-tuning
- Solution: 15% of the words to predict, but don't replace with `[MASK]` 100% of the time. Instead:
- 80% of the time, replace with `[MASK]`

  `went to the store → went to the [MASK]`
- 10% of the time, replace random word

  `went to the store → went to the running`
- 10% of the time, keep same

  `went to the store → went to the store`

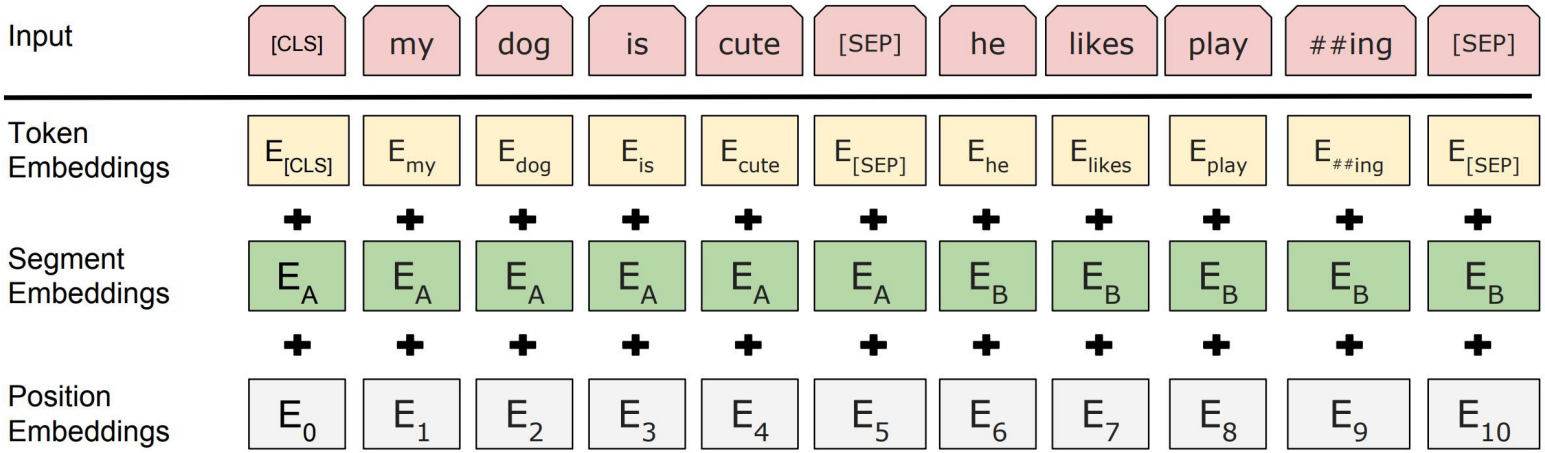
Next Sentence Prediction

- To learn *relationships* between sentences, predict whether Sentence B is actual sentence that proceeds Sentence A, or a random sentence

Sentence A = `The man went to the store.`
Sentence B = `He bought a gallon of milk.`
Label = `IsNextSentence`

Sentence A = `The man went to the store.`
Sentence B = `Penguins are flightless.`
Label = `NotNextSentence`

Input Representation

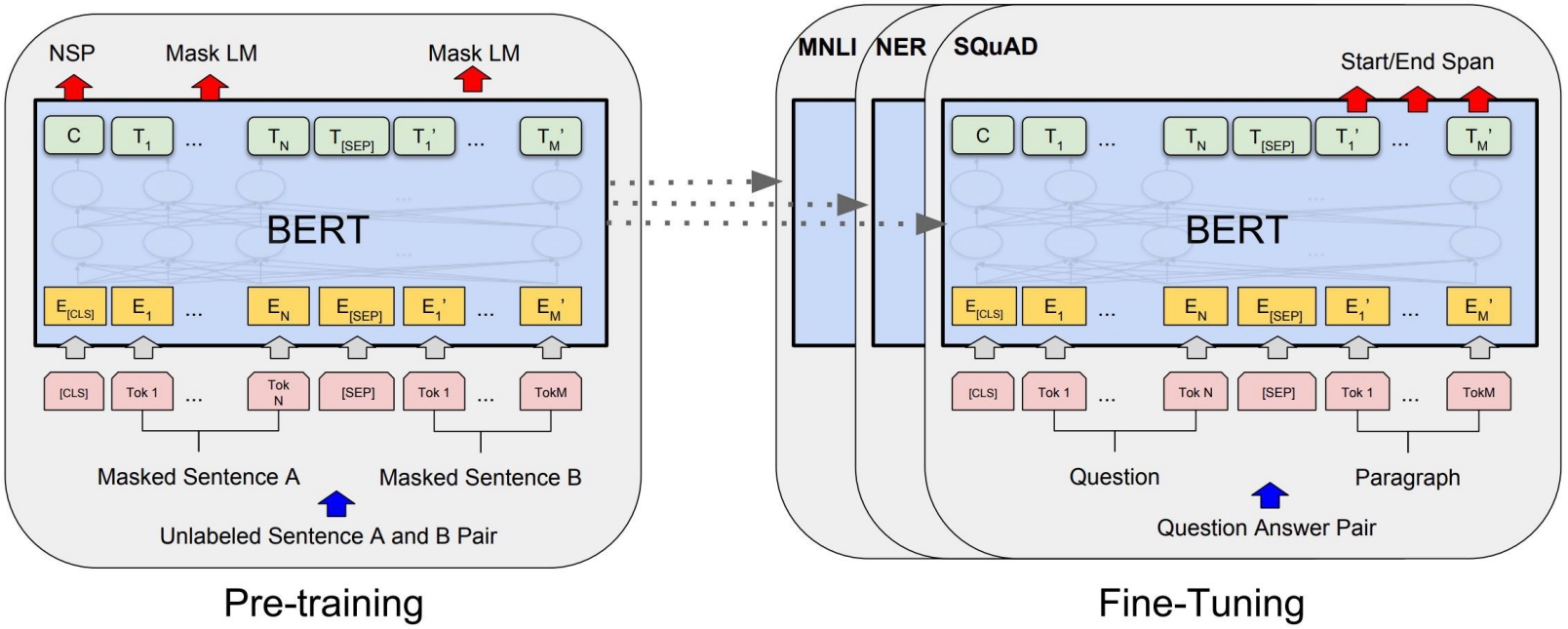


- Use 30,000 WordPiece vocabulary on input.
- Each token is sum of three embeddings
- Single sequence is much more efficient.

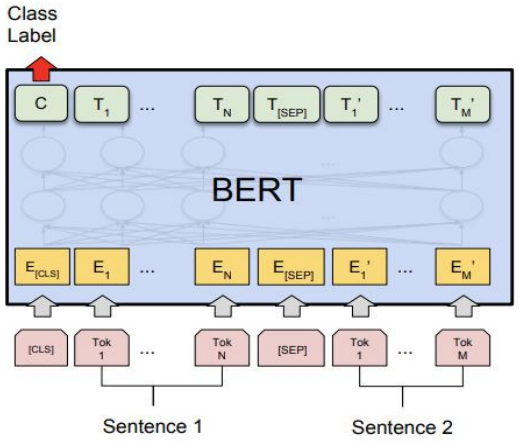
Model Details

- Data: Wikipedia (2.5B words) + BookCorpus (800M words)
- Batch Size: 131,072 words (1024 sequences * 128 length or 256 sequences * 512 length)
- Training Time: 1M steps (~40 epochs)
- Optimizer: AdamW, 1e-4 learning rate, linear decay
- `BERT-Base`: 12-layer, 768-hidden, 12-head
- `BERT-Large`: 24-layer, 1024-hidden, 16-head
- Trained on 4x4 or 8x8 TPU slice for 4 days

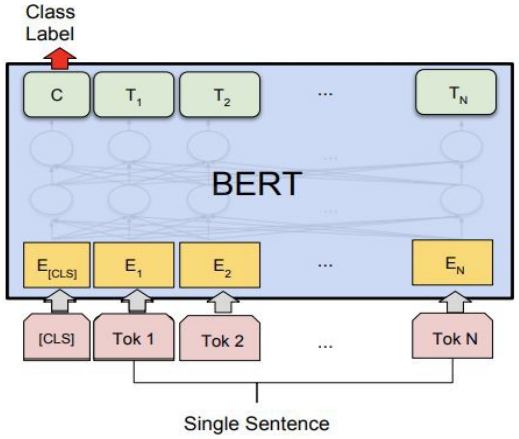
Fine-Tuning Procedure



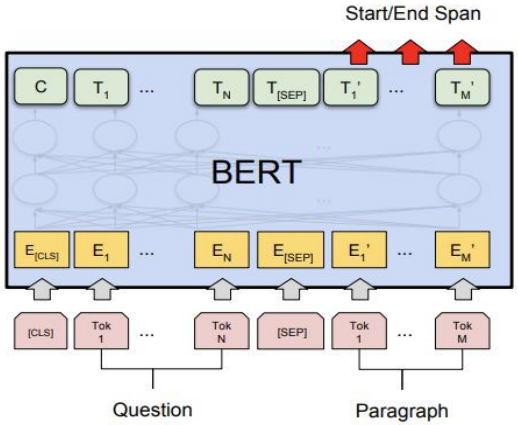
Fine-Tuning Procedure



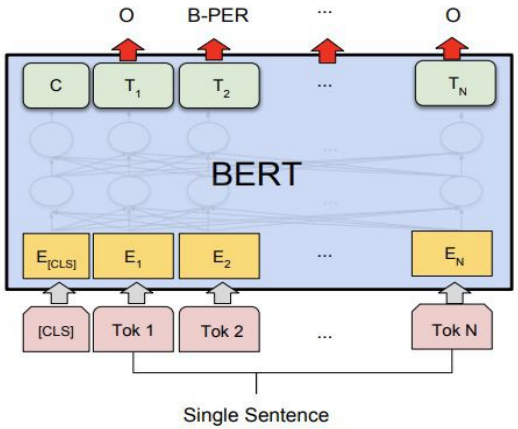
(a) Sentence Pair Classification Tasks: MNLI, QQP, QNLI, STS-B, MRPC, RTE, SWAG



(b) Single Sentence Classification Tasks: SST-2, CoLA



(c) Question Answering Tasks: SQuAD v1.1



(d) Single Sentence Tagging Tasks: CoNLL-2003 NER

GLUE Results

System	MNLI-(m/mm) 392k	QQP 363k	QNLI 108k	SST-2 67k	CoLA 8.5k	STS-B 5.7k	MRPC 3.5k	RTE 2.5k	**Average** -
Pre-OpenAI SOTA	80.6/80.1	66.1	82.3	93.2	35.0	81.0	86.0	61.7	74.0
BiLSTM+ELMo+Attn	76.4/76.1	64.8	79.9	90.4	36.0	73.3	84.9	56.8	71.0
OpenAI GPT	82.1/81.4	70.3	88.1	91.3	45.4	80.0	82.3	56.0	75.2
$\text{BERT}_{\text{BASE}}$	84.6/83.4	71.2	90.1	93.5	52.1	85.8	88.9	66.4	79.6
$\text{BERT}_{\text{LARGE}}$	**86.7/85.9**	**72.1**	**91.1**	**94.9**	**60.5**	**86.5**	**89.3**	**70.1**	**81.9**

MultiNLI

Premise: Hills and mountains are especially sanctified in Jainism.
Hypothesis: Jainism hates nature.
Label: Contradiction

CoLa

Sentence: The wagon rumbled down the road.
Label: Acceptable

Sentence: The car honked down the road.
Label: Unacceptable

SQuAD 2.0

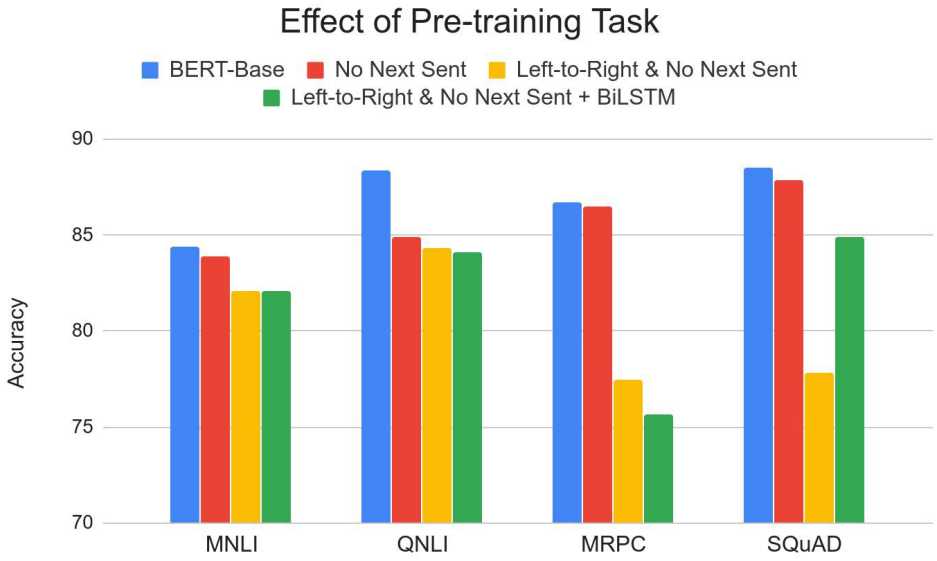
What action did the US begin that started the second oil shock?
Ground Truth Answers: <No Answer>
Prediction: <No Answer>

The 1973 oil crisis began in October 1973 when the members of the Organization of Arab Petroleum Exporting Countries (OAPEC, consisting of the Arab members of OPEC plus Egypt and Syria) proclaimed an oil embargo. By the end of the embargo in March 1974, the price of oil had risen from US$3 per barrel to nearly $12 globally; US prices were significantly higher. The embargo caused an oil crisis, or "shock", with many short- and long-term effects on global politics and the global economy. It was later called the "first oil shock", followed by the 1979 oil crisis, termed the "second oil shock."

Rank	Model	EM	F1
	Human Performance *Stanford University* (Rajpurkar & Jia et al. '18)	86.831	89.452
12 Nov 08, 2018	BERT (single model) *Google AI Language*	80.005	83.061
20 Sep 13, 2018	nlnet (single model) *Microsoft Research Asia*	74.272	77.052

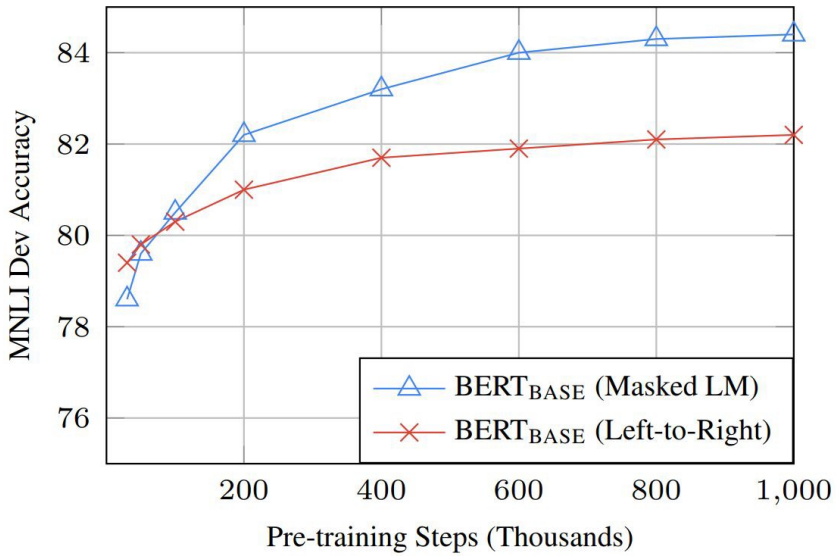
- Use token 0 ([CLS]) to emit logit for “no answer”.
- “No answer” directly competes with answer span.
- Threshold is optimized on dev set.

Effect of Pre-training Task



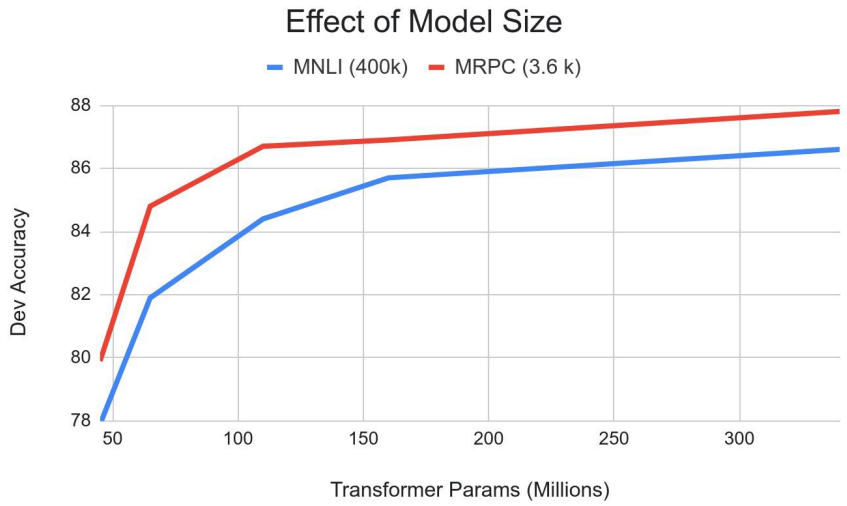
- Masked LM (compared to left-to-right LM) is very important on some tasks, Next Sentence Prediction is important on other tasks.
- Left-to-right model does very poorly on word-level task (SQuAD), although this is mitigated by BiLSTM

Effect of Directionality and Training Time



- Masked LM takes slightly longer to converge because we only predict 15% instead of 100%
- But absolute results are much better almost immediately

Effect of Model Size



- Big models help *a lot*
- Going from 110M -> 340M params helps even on datasets with 3,600 labeled examples
- Improvements have *not* asymptoted

Open Source Release

- One reason for BERT's success was the open source release
  - Minimal release (not part of a larger codebase)
  - No dependencies but TensorFlow (or PyTorch)
  - Abstracted so people could including a single file to use model
  - End-to-end push-button examples to train SOTA models
  - Thorough README
  - Idiomatic code
  - Well-documented code
  - Good support (for the first few months)

RoBERTA

- *RoBERTa: A Robustly Optimized BERT Pretraining Approach* (Liu et al, University of Washington and Facebook, 2019)
- Trained BERT for more epochs and/or on more data
  - Showed that more epochs alone helps, even on same data
  - More data also helps
- Improved masking and pre-training data slightly

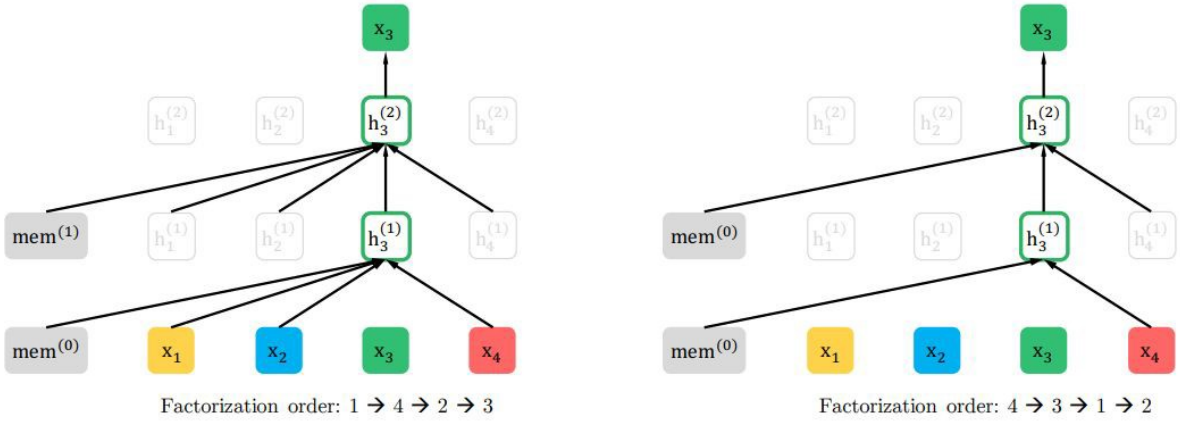
	MNLI	QNLI	QQP	RTE	SST	MRPC	CoLA	STS	WNLI	Avg
Single-task single models on dev										
$\text{BERT}_{\text{LARGE}}$	86.6/-	92.3	91.3	70.4	93.2	88.0	60.6	90.0	-	-
$\text{XLNet}_{\text{LARGE}}$	89.8/-	93.9	91.8	83.8	95.6	89.2	63.6	91.8	-	-
RoBERTa	**90.2/90.2**	**94.7**	**92.2**	**86.6**	**96.4**	**90.9**	**68.0**	**92.4**	**91.3**	-

XLNet

- *XLNet: Generalized Autoregressive Pretraining for Language Understanding* (Yang et al, CMU and Google, 2019)
- Innovation #1: Relative position embeddings
  - Sentence: `John ate a hot dog`
  - Absolute attention: “How much should `dog` attend to `hot`(in any position), and how much should `dog` in position 4 attend to the word in position 3? (Or 508 attend to 507, …)”
  - Relative attention: “How much should `dog` attend to `hot` (in any position) and how much should `dog` attend to the previous word?”

XLNet

- Innovation #2: Permutation Language Modeling
  - In a left-to-right language model, every word is predicted based on all of the words to its left
  - Instead: Randomly permute the order for *every training sentence*
  - Equivalent to masking, but many more predictions per sentence
  - Can be done efficiently with Transformers



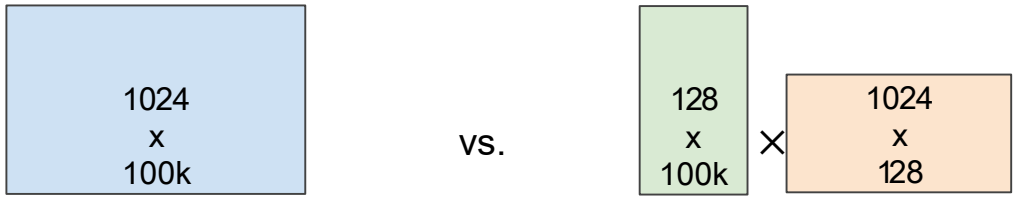
XLNet

- Also used more data and bigger models, but showed that innovations improved on BERT even with same data and model size
- XLNet results:

Model	MNLI	QNLI	QQP	RTE	SST-2	MRPC	CoLA	STS-B
Single-task single models on dev								
BERT [2]	86.6/-	92.3	91.3	70.4	93.2	88.0	60.6	90.0
RoBERTa [21]	90.2/90.2	94.7	92.2	**86.6**	96.4	**90.9**	68.0	92.4
XLNet	**90.8/90.8**	**94.9**	**92.3**	85.9	**97.0**	90.8	**69.0**	**92.5**

ALBERT

- *ALBERT: A Lite BERT for Self-supervised Learning of Language Representations* (Lan et al, Google and TTI Chicago, 2019)
- Innovation #1: Factorized embedding parameterization
  - Use small embedding size (e.g., 128) and then project it to Transformer hidden size (e.g., 1024) with parameter matrix



ALBERT

- Innovation #2: Cross-layer parameter sharing
  - Share all parameters between Transformer layers
- Results:

Models	MNLI	QNLI	QQP	RTE	SST	MRPC	CoLA	STS
Single-task single models on dev								
BERT-large	86.6	92.3	91.3	70.4	93.2	88.0	60.6	90.0
XLNet-large	89.8	93.9	91.8	83.8	95.6	89.2	63.6	91.8
RoBERTa-large	90.2	94.7	**92.2**	86.6	96.4	**90.9**	68.0	92.4
ALBERT (1M)	90.4	95.2	92.0	88.1	96.8	90.2	68.7	92.7
ALBERT (1.5M)	**90.8**	**95.3**	**92.2**	**89.2**	**96.9**	**90.9**	**71.4**	**93.0**

- ALBERT is light in terms of *parameters*, not *speed*

Model		Parameters	SQuAD1.1	SQuAD2.0	MNLI	SST-2	RACE	Avg	Speedup
BERT	base	108M	90.4/83.2	80.4/77.6	84.5	92.8	68.2	82.3	4.7x
	large	334M	92.2/85.5	85.0/82.2	86.6	93.0	73.9	85.2	1.0
ALBERT	base	12M	89.3/82.3	80.0/77.1	81.6	90.3	64.0	80.1	5.6x
	large	18M	90.6/83.9	82.3/79.4	83.5	91.7	68.5	82.4	1.7x
	xlarge	60M	92.5/86.1	86.1/83.1	86.4	92.4	74.8	85.5	0.6x
	xxlarge	235M	**94.1/88.3**	**88.1/85.1**	**88.0**	**95.2**	**82.3**	**88.7**	0.3x

T5

- *Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer* (Raffel et al, Google, 2019)
- Ablated many aspects of pre-training:
  - Model size
  - Amount of training data
  - Domain/cleanness of training data
  - Pre-training objective details (e.g., span length of masked text)
  - Ensembling
  - Finetuning recipe (e.g., only allowing certain layers to finetune)
  - Multi-task training

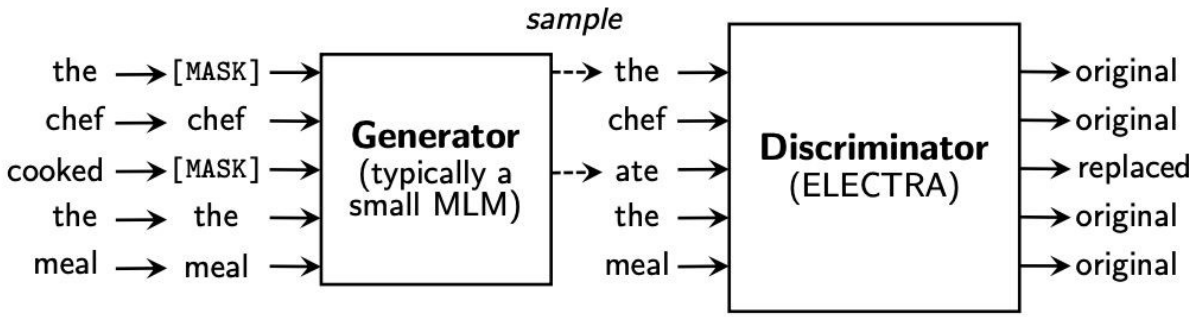
T5

- Conclusions:
  - Scaling up model size and amount of training data helps a lot
  - Best model is 11B parameters (BERT-Large is 330M), trained on 120B words of cleaned common crawl text
  - Exact masking/corruptions strategy doesn't matter that much
  - Mostly negative results for better finetuning and multi-task strategies
- T5 results:

	Rank	Name	Model	URL	Score	BoolQ	CB	COPA	MultiRC	ReCoRD	RTE	WiC	WSC	AX-b	AX-g
	1	SuperGLUE Human Baselines	SuperGLUE Human Baselines		89.8	89.0	95.8/98.9	100.0	81.8/51.9	91.7/91.3	93.6	80.0	100.0	76.6	99.3/99.7
+	2	T5 Team - Google	T5		89.3	91.2	93.9/96.8	94.8	88.1/63.3	94.1/93.4	92.5	76.9	93.8	65.6	92.7/91.9
	3	Zhuiyi Technology	RoBERTa-mtl-adv		85.7	87.1	92.4/95.6	91.2	85.1/54.3	91.7/91.3	88.1	72.1	91.8	58.5	91.0/78.1
	4	Facebook AI	RoBERTa		84.6	87.1	90.5/95.2	90.6	84.4/52.5	90.6/90.0	88.2	69.9	89.0	57.9	91.0/78.1
	5	IBM Research AI	BERT-mtl		73.5	84.8	89.6/94.0	73.8	73.2/30.5	74.6/74.0	84.1	66.2	61.0	29.6	97.8/57.3
	6	SuperGLUE Baselines	BERT++		71.5	79.0	84.8/90.4	73.8	70.0/24.1	72.0/71.3	79.0	69.6	64.4	38.0	99.4/51.4
			BERT		69.0	77.4	75.7/83.6	70.6	70.0/24.1	72.0/71.3	71.7	69.6	64.4	23.0	97.8/51.7

ELECTRA

- *ELECTRA: Pre-training Text Encoders as Discriminators Rather Than Generators* (Clark et al, 2020)
- Train model to discriminate locally plausible text from real text

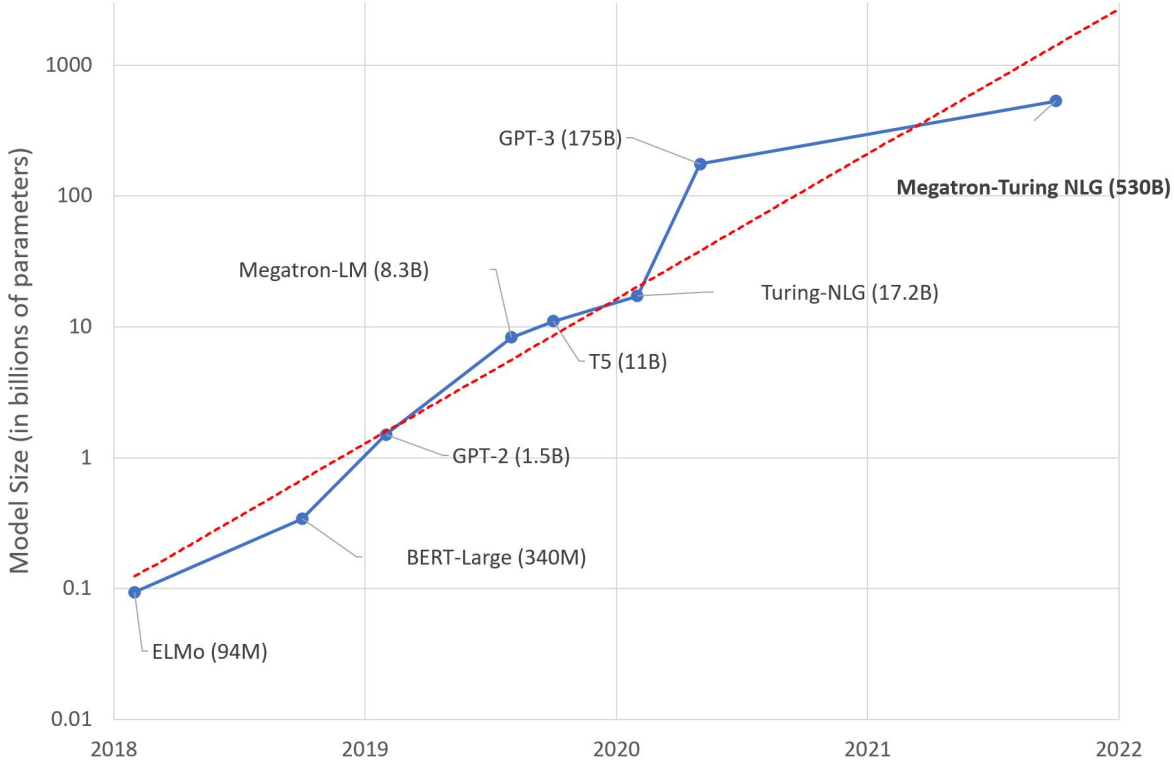


ELECTRA

- Difficult to match SOTA results with less compute

Model	Train FLOPs	Params	SQuAD 1.1		SQuAD 2.0	
			EM	F1	EM	F1
BERT-Base	6.4e19 (0.09x)	110M	80.8	88.5	–	–
BERT	1.9e20 (0.27x)	335M	84.1	90.9	79.0	81.8
SpanBERT	7.1e20 (1x)	335M	88.8	94.6	85.7	88.7
XLNet-Base	6.6e19 (0.09x)	117M	81.3	–	78.5	–
XLNet	3.9e21 (5.4x)	360M	**89.7**	**95.1**	87.9	**90.6**
RoBERTa-100K	6.4e20 (0.90x)	356M	–	94.0	–	87.7
RoBERTa-500K	3.2e21 (4.5x)	356M	88.9	94.6	86.5	89.4
ALBERT	3.1e22 (44x)	235M	89.3	94.8	87.4	90.2
BERT (ours)	7.1e20 (1x)	335M	88.0	93.7	84.7	87.5
ELECTRA-Base	6.4e19 (0.09x)	110M	84.5	90.8	80.5	83.3
ELECTRA-400K	7.1e20 (1x)	335M	88.7	94.2	86.9	89.6
ELECTRA-1.75M	3.1e21 (4.4x)	335M	**89.7**	94.9	**88.1**	**90.6**

Large-Scale Language Models



- Megatron-Turing NLG (2021.10.11)
  - 105-layer, Transformer-based, 530B model parameters
  - 339B training tokens
  - 560 servers * 8 NVIDIA A100 80GB Tensor Core GPUs
  - Strong performance on zero-, one-, and few-shot learning tasks