香港中文大學（深圳）
The Chinese University of Hong Kong, Shenzhen

CSC6052/DDA6307/MDS6002: Large Language Model

Lecture 3: Word Representation and Language Modeling

Spring 2024
Benyou Wang
School of Data Science

Recap

- What is linguistics?
- Linguistic structure
  - Character
  - Word
  - Sentence
  - Discourse (篇章)

- More about desturcture and scaling
  - Inductive bias
  - Inductive bias in NLP during many decades
  - Rethinking Empiricism vs. Rationalism

- (Next) From linguistics to computing linguistics

How modern NN perceives structure

- ❑ Bag of words
- ❑ Word sequence
- ❑ Injected structure
  - ❑ syntax or dependency tree(Recursive NN)
  - ❑ with local connections (Convolution NN)
  - ❑ with a recurrent bias (Recurrent NN)

Transformer： bag-of-words models with position embeddings

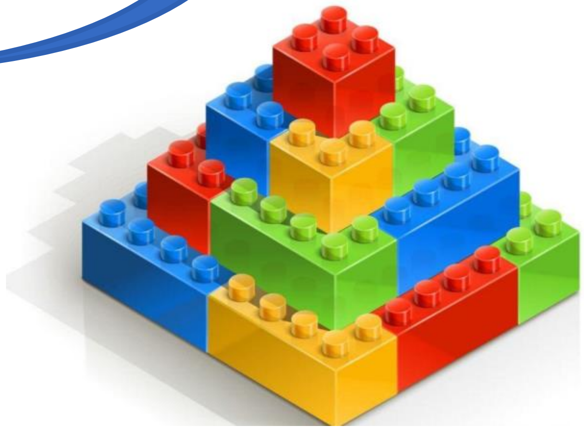
Structure is learned in a data-driven way thanks to free attention.

For NLP

- Building blocks
  - Words/subwords/tokens

- Building strategies
  - Neural network
  - Pre-trained objective
    - Language modeling

Side product

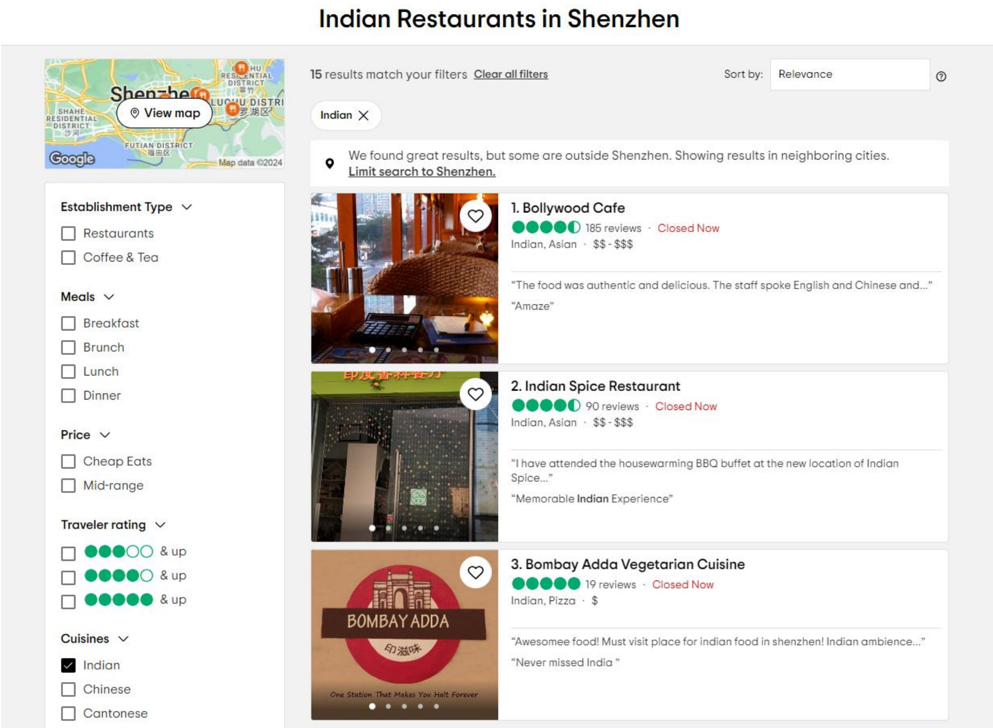


Part 1:

Understanding Words from the perspective of **Information retrieval**

Application 1: How to find a book in library?

Application 2: How to search?



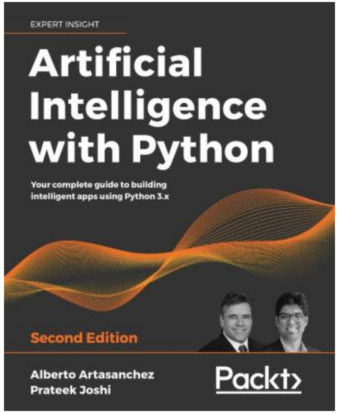
https://www.tripadvisor.com/Restaurants-g297415-c24-Shenzhen_Guangdong.html

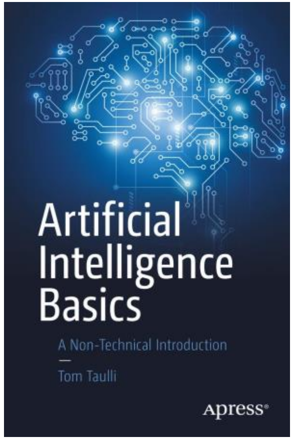
Application 3： talk to your personal assistant

First, we need to know how to represent words

A simple practice: term matching

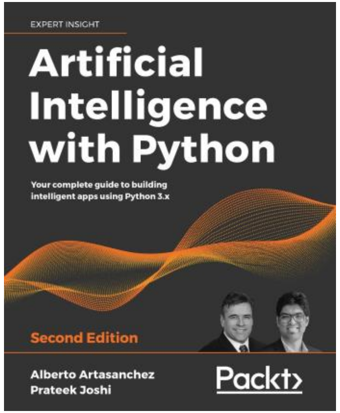
- Find books with AI in the title

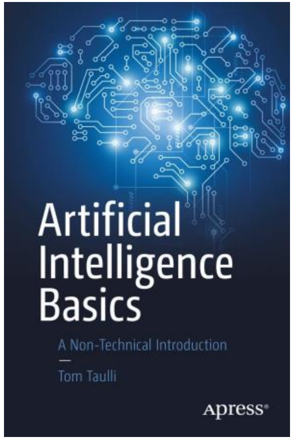




Have the word or not?

Which book is more relevant?



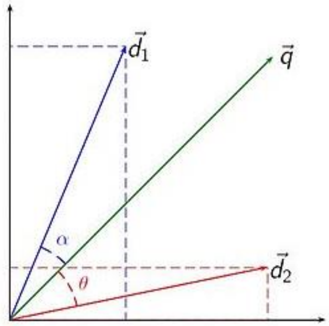


Which one do you prefer if all book have “AI” in their titles

Vector Space Model:

Representation:
Documents and queries are represented as vectors in some space.
-- What are the dimensions?
-- How to map documents and queries to this space?

Scoring Function:
Rank documents by a measure of “closeness” of query and document vectors
-- Distance, Cosine of the angle etc.



$$score(d,q) = \frac{\vec{d} \cdot \vec{q}}{||\vec{d}||\ ||\vec{q}||}$$

Image from: http://en.wikipedia.org/wiki/Vector_space_model

Vector Space Model: TF-IDF Model

Dimensions: Every word becomes a dimension.

$$D = <x_1, x_2, \ldots, x_{|V|}>$$

Term Frequency:

A term that occurs many times in a document is important.

$$tf(t, D) = \log\left(1 + \#(t, D)\right)$$

Inverse Document Frequency:

A term that occurs in many documents is unimportant.

$$idf(t) = \log\left(\frac{N}{|D_t|}\right)$$

TF-IDF:

Term-frequency weighted by inverse document frequency.

$$tfidf(t, D) = tf(t, D) \times idf(t)$$

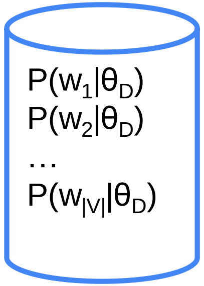
Vector Space Models: Enhancements

- Efficient scoring
  - Inverted indices – compression, skip lists, etc.
  - Top-k documents without scoring all documents entirely.
- Scoring methods
  - Document length normalizations (e.g., pivoted)
  - BM25 and various other heuristics
- Dimensionality reduction
  - Stemming, phrase representations etc.
  - LSI

Probabilistic Retrieval Models: Language Modeling

Generative Assumption

Assume that each document is generated by a probabilistic process specified by a **multinomial distribution.**



Document model parameterized by θ

Query-likelihood Model

If a document D is relevant to a query, then the query should be a high probability sample from the document multinomial.

Later termed as “uni-gram” language model.

Probabilistic Retrieval Models: Query-likelihood Model

Rank by the probability that the document model generated the query.

$$P(Q \mid D) = \prod_{q \in Q} P(q \mid \theta_D)$$

$P(q|\theta)$ → Query model

θ_D → Document model

Probabilistic Retrieval Models: Estimation

Estimate document model via maximum-likelihood

-- Assume document is a sample of the underlying distribution.

$$\theta_D(w) = \frac{\#(w, D)}{\sum_{w_i \in D} \#(w_i, D)}$$

Zero probability issue!

Latent Semantic Analysis

Premise

Using one-dimension per word is problematic.

Doesn’t scale.
(millions of words in large collections)

Doesn’t handle synonymy
(dog vs. canine)

Idea

Project documents and queries into a lower-dimensional space.
Instead of a |V| dimensional vector each word will be represented by a **k << |V|** dimensional vector.

Latent Semantic Analvsis

Original term-by-document matrix

	D1	D2	D3	D4	D5	D6	Q1
rock	2	1	0	2	0	1	1
granite	1	0	1	0	0	0	0
marble	1	2	0	0	0	0	1
music	0	0	0	1	2	0	0
song	0	0	0	1	0	2	0
band	0	0	0	0	1	0	0

Documents projected into 2D semantic space

	D1	D2	D3	D4	D5	D6	Q1
Dim. 1	-0.888	-0.759	-0.615	-0.961	-0.388	-0.851	-0.845
Dim. 2	0.460	0.652	0.789	-0.276	-0.922	-0.525	0.534

1) Represent co-occurrences as a term-document TD matrix (X).

2) Use singular-value decomposition to factorize TD.

$$X = U \Sigma V^T$$

3) Sigma is a diagonal matrix. Take the top k *singular values.*

4) Project query and documents using the top-k *singular* values.

$$\hat{d} = \Sigma_k^{-1} U_k^T d \quad \hat{q} = \Sigma_k^{-1} U_k^T q$$

5) Cosine similarity to score documents.

Figures stolen from: http://www.cs.cmu.edu/~nasmith/LS2/gimpel.06.pdf

A bigger difference

$$X = U\Sigma V^T$$

Original term-by-document matrix

	D1	D2	D3	D4	D5	D6	Q1
rock	2	1	0	2	0	1	1
granite	1	0	1	0	0	0	0
marble	1	2	0	0	0	0	1
music	0	0	0	1	2	0	0
song	0	0	0	1	0	2	0
band	0	0	0	0	1	0	0

Documents projected into 2D semantic space

	D1	D2	D3	D4	D5	D6	Q1
Dim. 1	-0.888	-0.759	-0.615	-0.961	-0.388	-0.851	-0.845
Dim. 2	0.460	0.652	0.789	-0.276	-0.922	-0.525	0.534

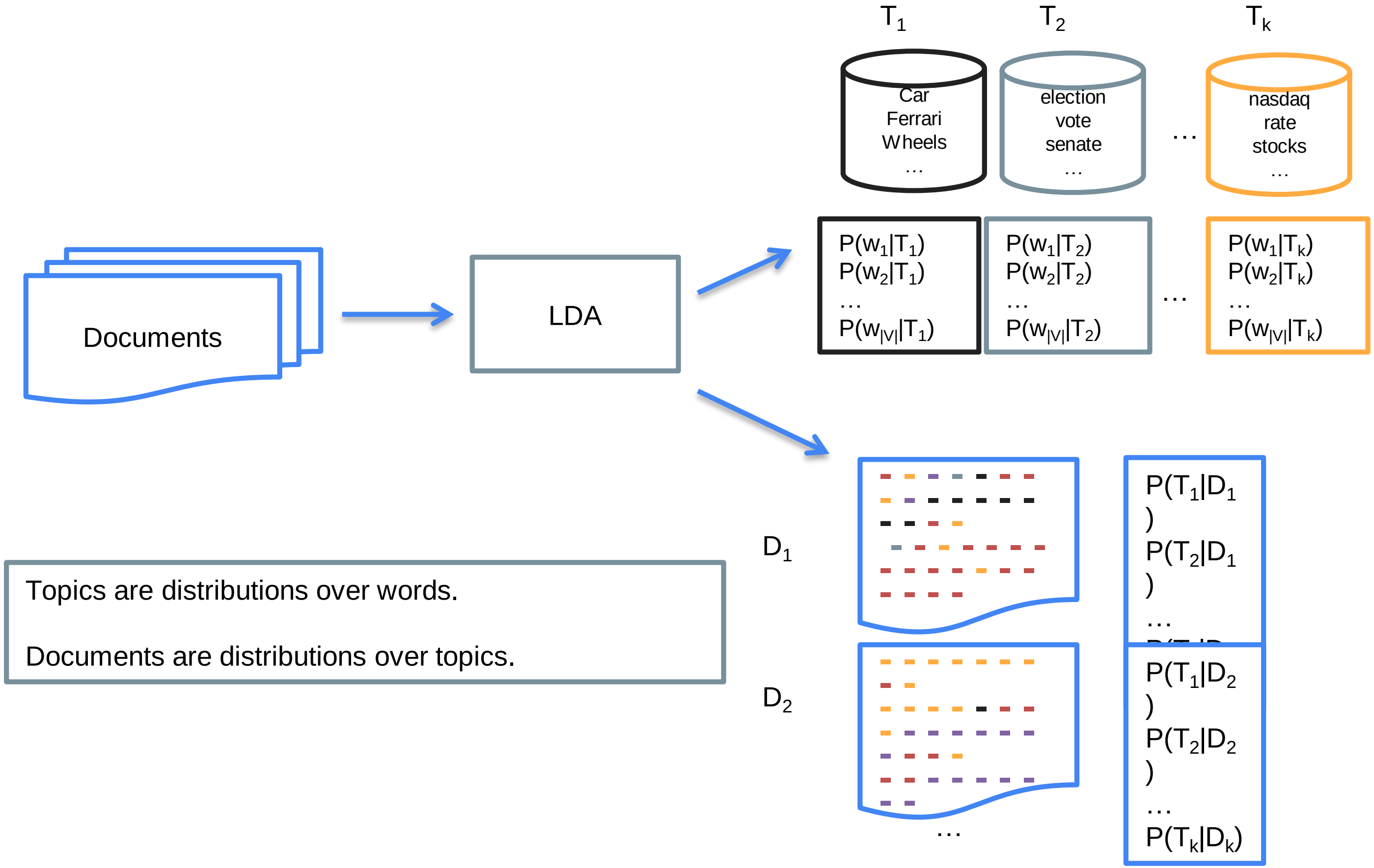
It does not rely on term match;
Even two documents could be relevant even they do not share any common words

V^T *implies* a set word vectors, later we will explain it.

Latent Semantic Analysis

- Benefits
  - Addresses synonymy
  - Dimensionality reduction
- Issues
  - Efficient SVD implementations necessary.
    - Many implementations available.
  - New documents need to be handled in a special way.
  - Disconnect w/ retrieval performance.
  - Rely on **direct counting**
- Extensions
  - Probabilistic Latent Semantic Analysis (pre-cursor to LDA).
  - Hierarchical (H-PLSA)

Topic Models using LDA



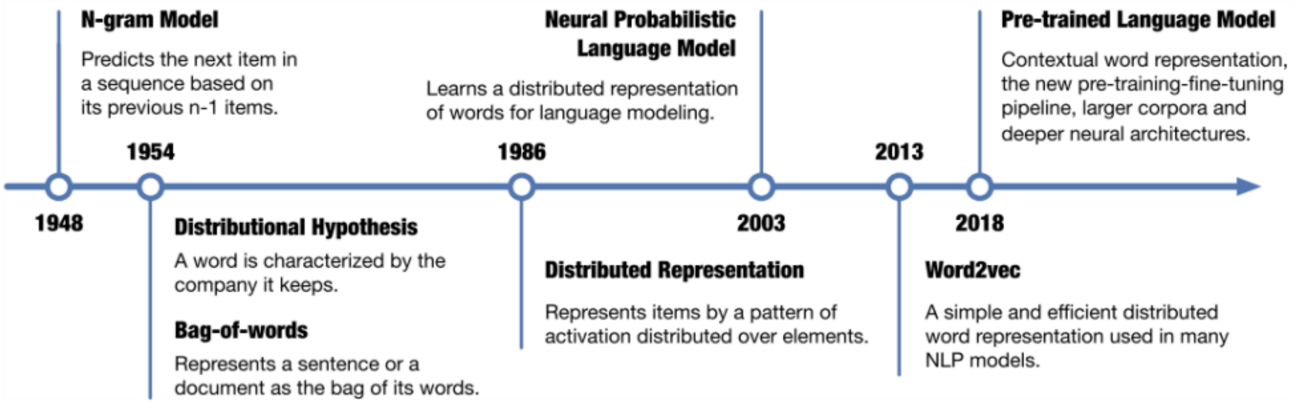
Topics are distributions over words.

Documents are distributions over topics.

Part 2:
Statistical language models to **neural** language models

Background

- language model



Liu et al., Representation Learning for Natural Language Processing, Springer, 2020

What is language modeling?

A **language model** assigns a probability to a N-gram

$$f : V^n \rightarrow R^+$$

What is language modeling?

A **language model** assigns a probability to a N-gram

$$f: V^n \rightarrow R^+$$

Sfklkljf fskjhfkjsh kjfs fs kjhkjhs fsjhfkshkjfh

Low probability

ChatGPT is all you need

high probability

What is language modeling?

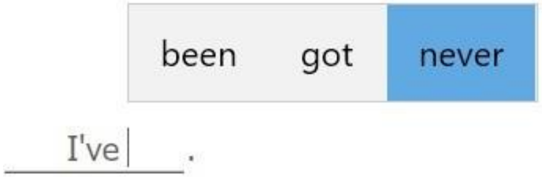
A **language model** assigns a probability to a N-gram

$$f: V^n \rightarrow R^+$$

A **conditional language model** assigns a probability of a word given some conditioning context

$$g: (V^{n-1}, V) \rightarrow R^+$$

And $\boldsymbol{p(w_n | w_1 \cdots w_{n-1})} = g(w_1 \cdots w_{n-1}, w) = \frac{f(w_1 \cdots w_n)}{f(w_1 \cdots w_{n-1})}$



What is language modeling?

A **language model** assigns a probability to a N-gram

$$f: V^n \rightarrow R^+$$

A **conditional language model** assigns a probability of a word given some conditioning context

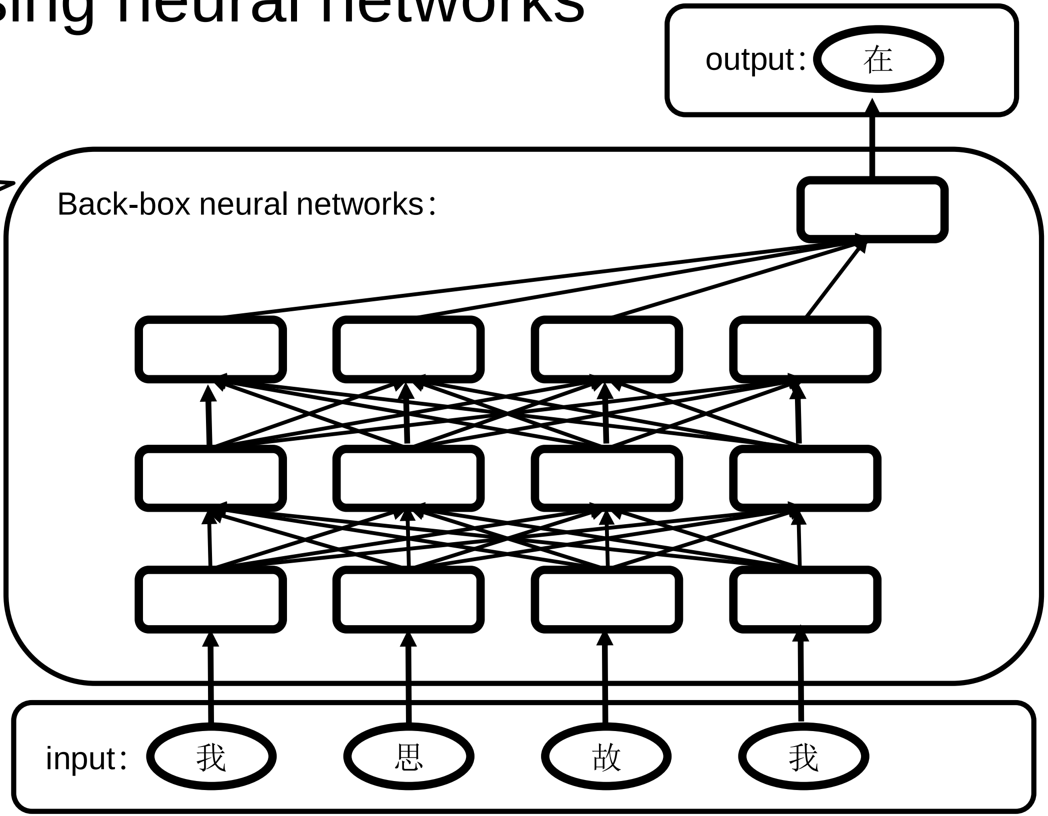
$$g: (V^{n-1}, V) \rightarrow R^+$$

And $\boldsymbol{p(w_n|w_1 \cdots w_{n-1})} = g(w_1 \cdots w_{n-1}, w) = \frac{f(w_1 \cdots w_n)}{f(w_1 \cdots w_{n-1})}$

$\boldsymbol{p(w_n|w_1 \cdots w_{n-1})}$ is the foundation of **modern large language models** (GPT, ChatGPT, etc.)

Language model using neural networks

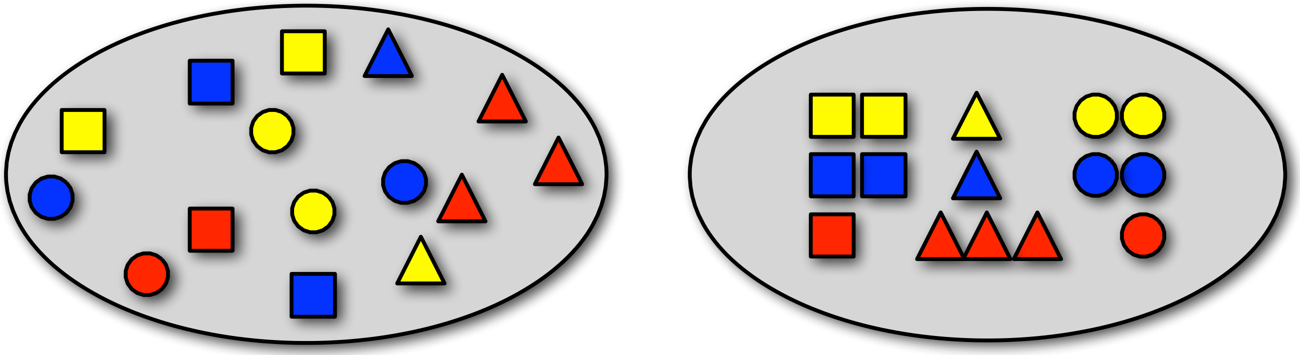
GPT-3/ChatGPT/GPT4 have 175B+ parameters
Humans have 100B+ neurons



Recap:
Basic Probability Theory

Sampling with replacement

Pick a random shape, then put it back in the bag.



P(■) = 2/15

P(blue) = 5/15

P(blue | □) = 2/5

P(■) = 1/15

P(red) = 5/15

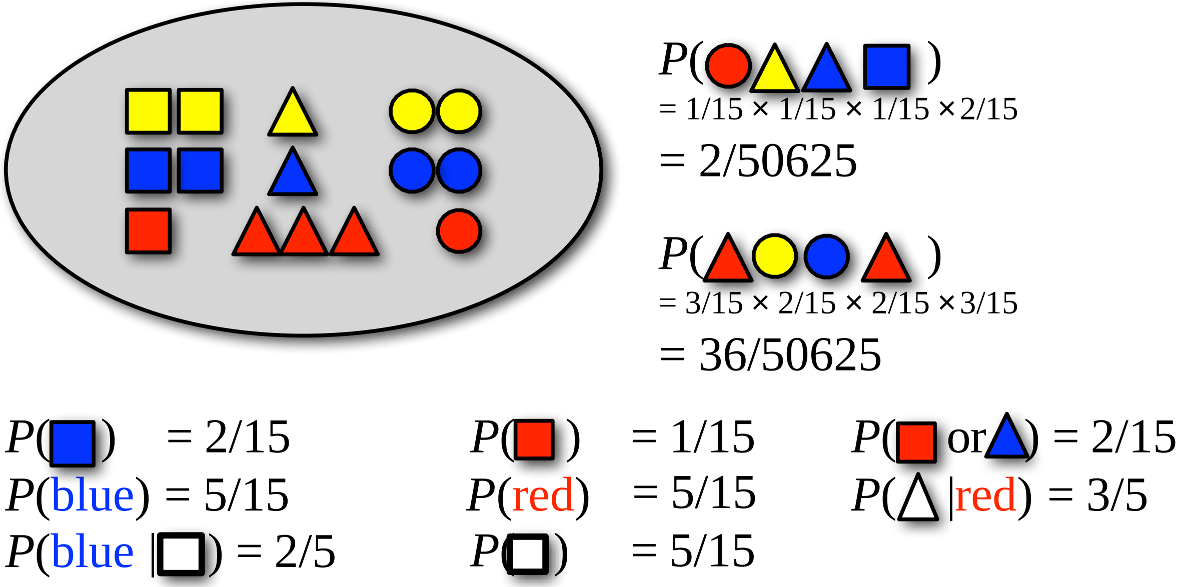
P(□) = 5/15

P(■ or ▲) = 2/15

P(△ | red) = 3/5

Sampling with replacement

Pick a random shape, then put it back in the bag.
What sequence of shapes will you draw?



Sampling with replacement

Alice was beginning to get very tired of sitting by her sister on the bank, and of having nothing to do: once or twice she had peeped into the book her sister was reading, but it had no pictures or conversations in it, 'and what is the use of a book,' thought Alice 'without pictures or conversation?'

$P($of$) = 3/66$

$P($Alice$) = 2/66$

$P($was$) = 2/66$

$P($to$) = 2/66$

$P($her$) = 2/66$

$P($sister$) = 2/66$

$P(,) = 4/66$

$P(') = 4/66$

Sampling with replacement

```
beginning by, very Alice but was and?
reading no tired of to into sitting
sister the, bank, and thought of without
her nothing: having conversations Alice
once do or on she it get the book her had
peeped was conversation it pictures or
sister in, 'what is the use had twice of
a book''pictures or' to
```

$P(\texttt{of}) = 3/66$ $P(\texttt{her}) = 2/66$

$P(\texttt{Alice}) = 2/66$ $P(\texttt{sister}) = 2/66$

$P(\texttt{was}) = 2/66$ $P(\texttt{,}) = 4/66$

$P(\texttt{to}) = 2/66$ $P(\texttt{'}) = 4/66$

In this model, $P(\textit{English sentence}) = P(\textit{word salad})$

Probability theory: terminology

Trial (aka "experiment")

Picking a shape, predicting a word

Sample space Ω:

The **set of all possible outcomes**
(all shapes; all words in *Alice in Wonderland*)

Event $\omega \subseteq \Omega$:

An **actual outcome** (a subset of Ω)
(predicting '*the*', picking a triangle)

Random variable $X: \Omega \rightarrow T$

A function from the sample space (often the identity function)
Provides a 'measurement of interest' from a trial/experiment
(Did we pick 'Alice'/a noun/a word starting with "x"/…?)

What is a probability distribution?

$P(\omega)$ defines a **distribution** over Ω iff

1) Every event ω has a probability $P(\omega)$ between 0 and 1:

$$0 \leq P(\omega \subseteq \Omega) \leq 1$$

2) The *null* event ∅ has probability $P(\emptyset) = 0$:

$$P(\emptyset) = 0$$

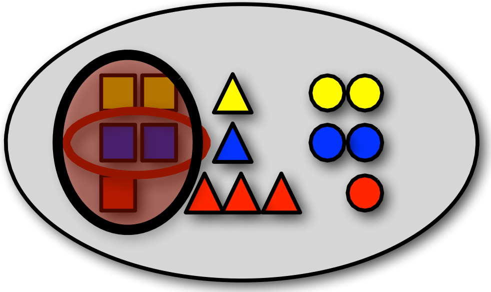
3) And the probability of all *disjoint* events sums to 1.

$$\sum_{\omega_i \subseteq \Omega} P(\omega_i) = 1 \quad \text{if } \forall j \neq i : \omega_i \cap \omega_j = \emptyset$$
$$\text{and } \bigcup_i \omega_i = \Omega$$

Joint and Conditional Probability

The conditional probability of X given Y, $P(X \mid Y)$, is defined in terms of the probability of Y, $P(Y)$, and the joint probability of X and Y, $P(X,Y)$:

$$P(X \mid Y) = \frac{P(X, Y)}{P(Y)}$$



$P($blue $\mid$ ■ $) = 2/5$

The chain rule

The joint probability $P(X,Y)$ can also be expressed in terms of the conditional probability $P(X \mid Y)$

$$P(X, Y) = P(X \mid Y)P(Y)$$

This leads to the so-called chain rule

$$\begin{aligned} P(X_1, X_2, \ldots, X_n) &= P(X_1)P(X_2|X_1)P(X_3|X_2, X_1)....P(X_n|X_1, ...X_{n-1}) \\ &= P(X_1)\prod_{i=2}^{n} P(X_i|X_1 \ldots X_{i-1}) \end{aligned}$$

Independence

Two random variables X and Y are independent if

$$P(X, Y) = P(X)P(Y)$$

If X and Y are independent, then $P(X \mid Y) = P(X)$:

$$
\begin{aligned}
P(X \mid Y) &= \frac{P(X, Y)}{P(Y)} \\
&= \frac{P(X)P(Y)}{P(Y)} \quad (X, Y \text{ independent}) \\
&= P(X)
\end{aligned}
$$

Probability models

Building a probability model consists of two steps:
1. Defining the model
2. Estimating the model's parameters
   (= training/learning)

Models (almost) always make
independence *assumptions.*

That is, even though X and Y are not actually independent, our model may treat them as independent.

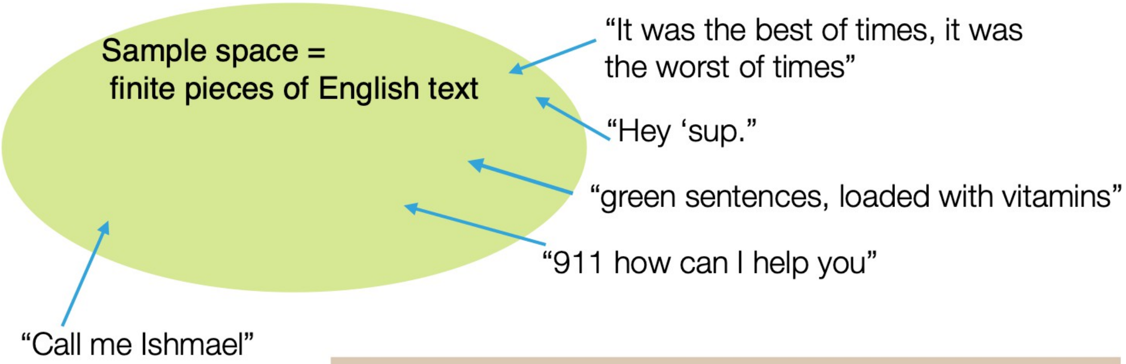
This reduces the number of model parameters that we need to estimate (e.g. from n^2 to $2n$)

language modeling

What is a language model?

- A probabilistic model of a sequence of words
- Joint probability distribution of words $w_1, w_2, \ldots, w_n$:

$$P(w_1, w_2, w_3, \ldots, w_n)$$



(i.e., $\Pr[w_1 w_2 w_2 \ldots w_n]$ associated with every finite word sequence $w_1 w_2 w_2 \ldots w_n$ (including nonsensical ones)

How likely is a given phrase, sentence, paragraph or even a document?

Chain rule

Conditional probability:
$p(w \mid w_1, w_2), \forall w \in V$

$$p(w_1, w_2, w_3, \ldots, w_n) =$$

$$p(w_1)p(w_2 \mid w_1)p(w_3 \mid w_1, w_2) ⇥ \cdots ⇥ p(w_n \mid w_1, w_2, \ldots, w_{n-1})$$

Sentence: “the cat sat on the mat”

$$\mathrm{P}(\text{the cat sat on the mat}) = \mathrm{P}(\text{the}) ⇤ \mathrm{P}(\text{cat}|\text{the}) ⇤ \mathrm{P}(\text{sat}|\text{the cat})$$
$$⇤\mathrm{P}(\text{on}|\text{the cat sat}) ⇤ \mathrm{P}(\text{the}|\text{the cat sat on})$$
$$⇤\mathrm{P}(\text{mat}|\text{the cat sat on the})$$

Implicit order

Language models: Narrow Sense

A probabilistic model that assigns a probability to every finite sequence (grammatical or not)

Sentence: “the cat sat on the mat”

$$
\begin{aligned}
P(\text{the cat sat on the mat}) = {} & P(\text{the}) * P(\text{cat}|\text{the}) * P(\text{sat}|\text{the cat}) \\
& * P(\text{on}|\text{the cat sat}) * P(\text{the}|\text{the cat sat on}) \\
& * P(\text{mat}|\text{the cat sat on the})
\end{aligned}
$$

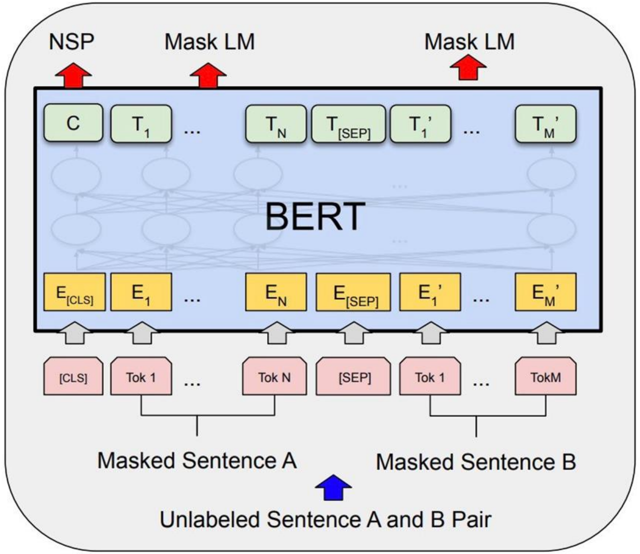
Implicit order

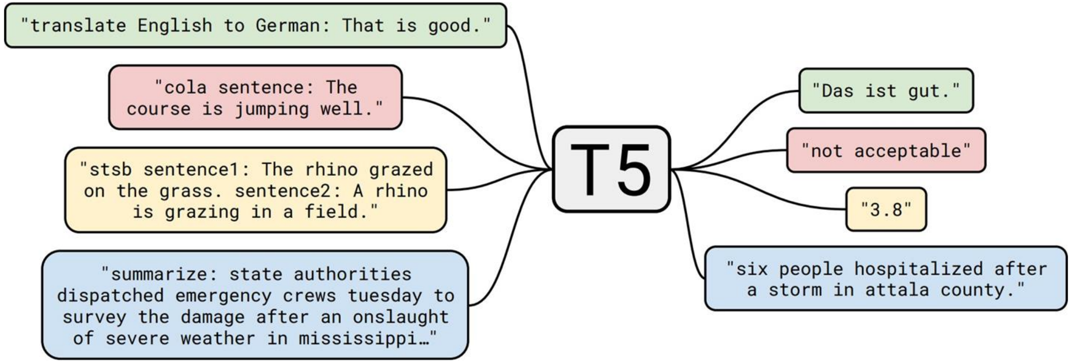
GPT-3 still acts in this way but the model is implemented as a very large neural network of 175-billion parameters!

Language models:Broad Sense

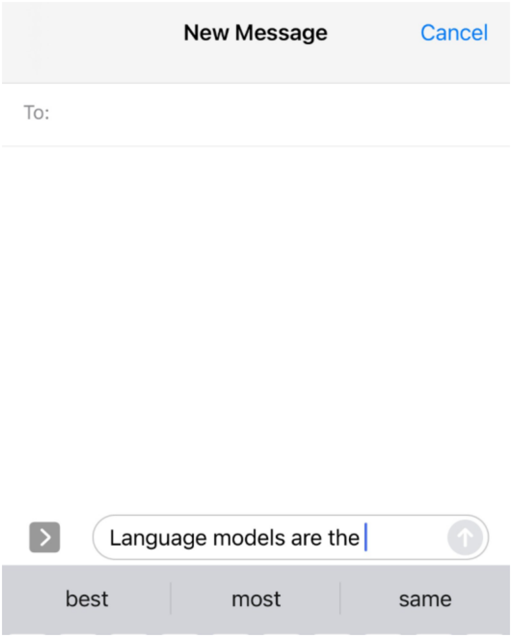
- ❖ Decoder-only models (GPT-x models)
- ❖ Encoder-only models (BERT, RoBERTa, ELECTRA)
- ❖ Encoder-decoder models (T5, BART)

The latter two usually involve a different **pre-training** objective.

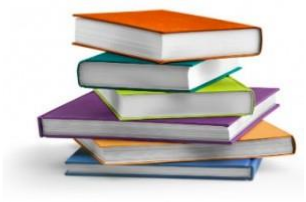




Language models are everywhere



Estimating probabilities



$$P(\text{sat}|\text{the cat}) = \frac{\text{count(the cat sat)}}{\text{count(the cat)}}$$

$$P(\text{on}|\text{the cat sat}) = \frac{\text{count(the cat sat on)}}{\text{count(the cat sat)}}$$

$$\vdots$$

trigram

bigram

Maximum likelihood estimate (MLE)

Assume we have a vocabulary of size V
how many sequences of length n do we have?

A) $n * V$

B) n^V

C) V^n

D) V/n

Estimating probabilities

$$P(\text{sat}|\text{the cat}) = \frac{\text{count(the cat sat)}}{\text{count(the cat)}}$$

$$P(\text{on}|\text{the cat sat}) = \frac{\text{count(the cat sat on)}}{\text{count(the cat sat)}}$$

$$\vdots$$

Maximum likelihood estimate (MLE)

- With a vocabulary of size V, # sequences of length $n = V^n$
- Typical English vocabulary ~ 40k words
  - Even sentences of length <= 11 results in more than 4 * 10^50 sequences. Too many to count! *(# of atoms in the earth ~ 10^50)*

Markov assumption

- Use only the recent past to predict the next word
- Reduces the number of estimated parameters in exchange for modeling capacity
- 1st order

$$P(\text{mat}|\text{the cat sat on the}) ⇡ P(\text{mat}|\text{the})$$

- 2nd order

$$P(\text{mat}|\text{the cat sat on the}) ⇡ P(\text{mat}|\text{on the})$$

Andrey Markov

k^{th} order Markov

Consider only the last *k* words (or less) for context

$$P(w_i \mid w_1 w_2 \ldots w_{i-1}) \approx P(w_i \mid w_{i-k} \ldots w_{i-1})$$

which implies the probability of a sequence is:

$$P(w_1 w_2 \ldots w_n) \approx \prod_i P(w_i \mid w_{i-k} \ldots w_{i-1})$$

(assume $w_j = \phi \quad \forall j < 0$)

Need to estimate counts for up to (k+1) grams

n-gram models

Unigram $$P(w_1, w_2, ... w_n) = \prod_{i=1}^{n} P(w_i)$$ e.g. P(the) P(cat) P(sat)

Bigram $$P(w_1, w_2, ... w_n) = \prod_{i=1}^{n} P(w_i | w_{i-1})$$ e.g. P(the) P(cat | the) P(sat | cat)

and Trigram, 4-gram, and so on.

Larger the n, more accurate and better the language model
(but also higher costs)

Caveat: Assuming infinite data!

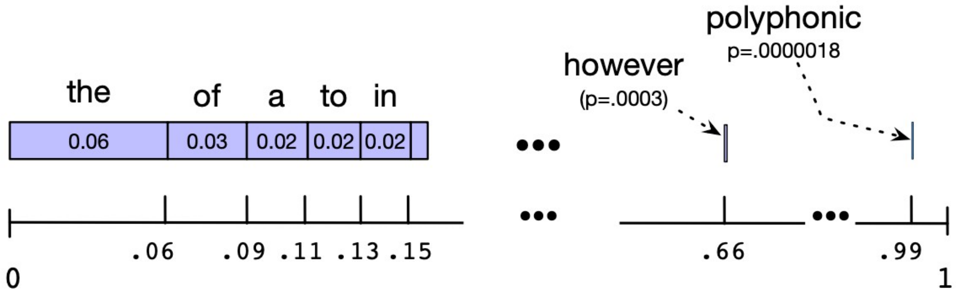
Generation using a language model

Generating from a language model

- Given a language model, how to generate a sequence?

Bigram $P(w_1, w_2, ...w_n) = \prod_{i=1}^{n} P(w_i|w_{i-1})$

- Generate the first word $w_1 \sim P(w)$
- Generate the second word $w_2 \sim P(w \mid w_1)$
- Generate the third word $w_3 \sim P(w \mid w_2)$
- …



Generating from a language model

- Given a language model, how to generate a sequence?

Trigram $$P(w_1, w_2, \ldots, w_n) = \prod_{i=1}^{n} P(w_i \mid w_{i-2}, w_{i-1})$$

- Generate the first word $w_1 \sim P(w)$
- Generate the second word $w_2 \sim P(w \mid w_1)$
- Generate the third word $w_3 \sim P(w \mid w_1, w_2)$
- Generate the fourth word $w_4 \sim P(w \mid w_2, w_3)$
- …

Generations

Unigram	*release millions See ABC accurate President of Donald Will cheat them a CNN megynkelly experience @ these word out- the*
Bigram	*Thank you believe that @ ABC news, Mississippi tonight and the false editorial I think the great people Bill Clinton . "*
Trigram	*We are going to MAKE AMERICA GREAT AGAIN! #MakeAmericaGreatAgain https: //t.co/DjkdAzT3WV*

Typical LMs are not sufficient to handle long-range dependencies

"Alice/Bob could not go to work that day because she/he had a doctor's appointment"

Generations

Example from a GPT-2 output:

prompt aka. conditional generation

With the start of the new academic year, Princeton has an opportunity to help provide a new generation of women with a diverse set of academic resources for higher education.
We are offering the resources of the Princeton-McGill program specifically to women with undergraduate degrees who would like to enhance their academic experience. Princeton-McGill offers a comprehensive suite of services for women and their families including a variety of graduate programs, support programs, and the opportunity to serve as leaders in their communities with a wide variety of programs, activities and services. For the upcoming fall, Princeton-McGill will also offer its Women's Center , which is located in a renovated women's dorm.
At Princeton, we are working with the Princeton-McGill community to develop a suite of programs that are designed to give new and returning students a strong foundation for a successful, rewarding graduate career. The Women's Center , the Princeton-McGill Women's Center provides a range of supports to address the specific needs of female doctoral degree graduates. Programs are tailored to meet the unique needs of women under the age of 28, women and families

https://talktotransformer.com/

$$P(w_1, w_2, \ldots, w_n) = \prod_{i=1}^{n} P(w_i \mid w_{i-1024}, \ldots, w_{i-2}, w_{i-1})$$

Modern LMs can handle much longer contexts!

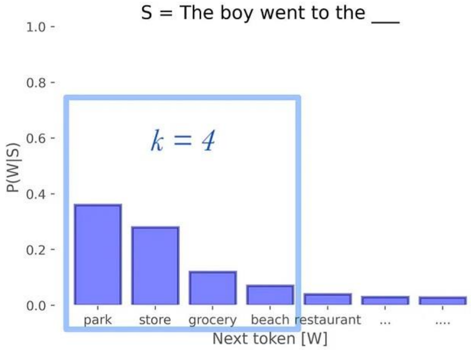
Generation methods (advanced)

- Greedy: choose the most likely word!

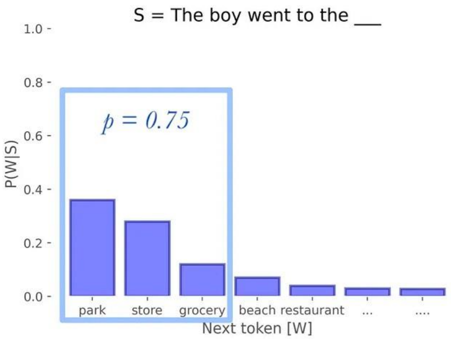
To predict the next word given a context of two words w_1, w_2:

$$w_3 = \arg\max_{w \in V} P(w \mid w_1, w_2)$$

- Top-k vs top-p sampling:



Top-k sampling

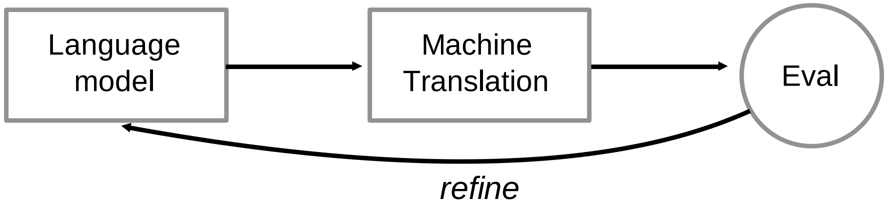


Top-p sampling

https://blog.allenai.org/a-guide-to-language-model-sampling-in-allennlp-3b1239274bc3

Evaluating a language model (perplexity)

Extrinsic evaluation



- Train LM apply to task observe accuracy
- Directly optimized for downstream applications
  - higher task accuracy better model
- Expensive, time consuming
- Hard to optimize downstream objective (indirect feedback)

Intrinsic evaluation of language models

Research process:

- Train parameters on a suitable training corpus
  - Assumption: observed sentences ~ good sentences
- Test on *different, unseen* corpus
  - If a language model assigns a higher probability to the test set, it is better
- Evaluation metric - perplexity!

Perplexity (ppl)

- Measure of how well a LM **predicts** the next word
- For a test corpus with words $w_1, w_2, \ldots w_n$

$$\text{Perplexity} = P(w_1, w_2, \ldots, w_n)^{-1/n}$$

$$\text{ppl}(S) = e^x \quad \text{where} \quad x = -\frac{1}{n}\log P(w_1, \ldots, w_n) = -\frac{1}{n}\sum_{i=1}^{n}\log P(w_i | w_1 \ldots w_{i-1})$$

- Unigram model: $x = -\frac{1}{n}\sum_{i=1}^{n}\log P(w_i)$ (since $P(w_j | w_1 \ldots w_{j-1}) \approx P(w_j)$)

- Minimizing perplexity ~ maximizing probability of corpus

Intuition on perplexity

If our k-gram model (with vocabulary V) has following probability:

$$P(w \mid w_{i-k}, \ldots w_{i-1}) = \frac{1}{|V|} \quad \forall w \in V$$

what is the perplexity of the test corpus?

A) $e^{|V|}$ B) $|V|$ C) $|V|^2$ D) $e^{-|V|}$

$$\text{ppl}(S) = e^{x} \quad \text{where} \quad x = -\frac{1}{n}\sum_{i=1}^{n} \log P(w_i \mid w_1 \ldots w_{i-1})$$

Intuition on perplexity

If our k-gram model (with vocabulary V) has following probability:

$$P(w \mid w_{i-k}, \ldots w_{i-1}) = \frac{1}{|V|} \quad \forall w \in V$$

$$\text{ppl}(S) = e^x \quad \text{where}$$
$$x = -\frac{1}{n} \sum_{i=1}^{n} \log P(w_i \mid w_1 \ldots w_{i-1})$$

what is the perplexity of the test corpus?

A) $e^{|V|}$ B) $|V|$ C) $|V|^2$ D) $e^{-|V|}$

$$\text{ppl} = e^{-\frac{1}{n} n \log(1/|V|)} = |V|$$

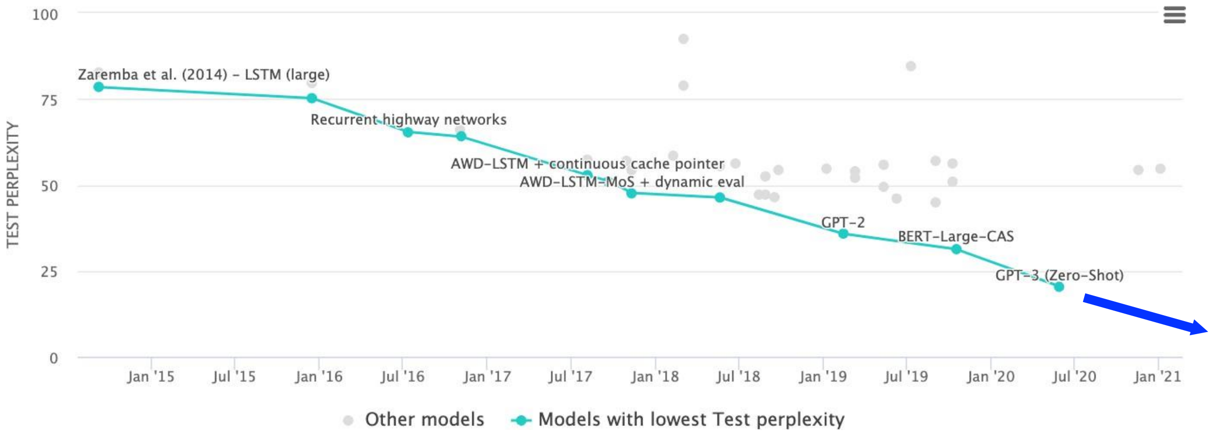
Measure of model's uncertainty about next word (aka `average branching factor')

branching factor = # of possible words following any word

Perplexity

Training corpus 38 million words, test corpus 1.5 million words, both **WSJ**

N-gram Order	Unigram	Bigram	Trigram
Perplexity (test)	962	170	109



GPT-3 175B:
ppl = 20.5

https://paperswithcode.com/sota/language-modelling-on-penn-treebank-word

Smoothing in language modeling

Generalization of n-grams

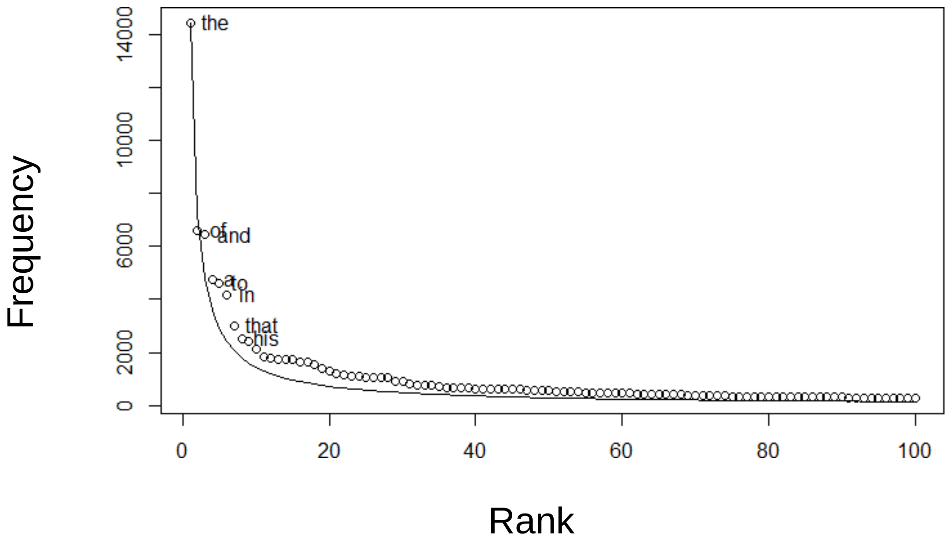
Any problems with n-gram models and their evaluation?

- Not all n-grams in the test set will be observed in training data
- Test corpus might have some that have zero probability under our model
  - **Training set**: *Google news*
  - **Test set**: *Shakespeare*
  - P(affray | voice doth us) = 0 $\Longrightarrow$ P(test corpus) = 0
  - Perplexity is not defined.

$$\text{ppl}(S) = e^x \quad \text{where}$$

$$x = -\frac{1}{n}\sum_{i=1}^{n} \log P(w_i | w_1 \ldots w_{i-1})$$

Sparsity in language



$$\text{freq} \propto \frac{1}{\text{rank}}$$

Zipf's Law

- Long tail of infrequent words
- Most finite-size corpora will have this problem.

Smoothing

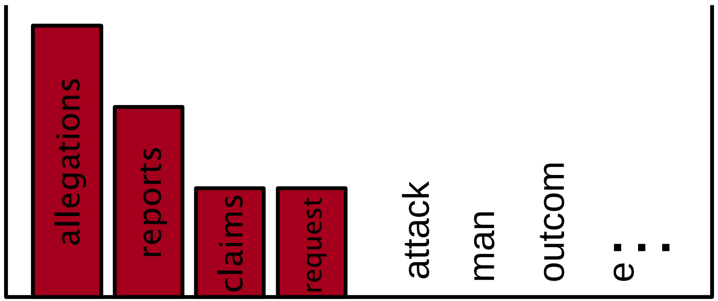
- Handle sparsity by making sure all probabilities are non-zero in our model
  - Additive: Add a small amount to all probabilities
  - Interpolation: Use a combination of different granularities of n-grams
  - Discounting: Redistribute probability mass from observed n-grams to unobserved ones

Smoothing intuition

When we have sparse statistics:

P(w | denied the)

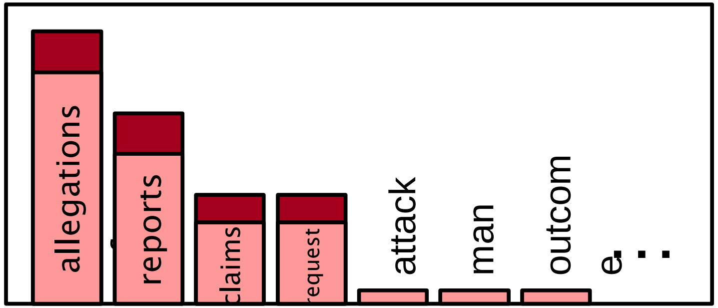
3 allegations
2 reports
1 claims
1 request
7 total



Steal probability mass to generalize better

P(w | denied the)

2.5 allegations
1.5 reports
0.5 claims
0.5 request
2 other
7 total



(Slide credit: Dan Klein)

Laplace smoothing

- Also known as add-alpha
- Simplest form of smoothing: Just add to all counts and renormalize!
- Max likelihood estimate for bigrams:

$$P(w_i|w_{i-1}) = \frac{C(w_{i-1}, w_i)}{C(w_{i-1})}$$

- After smoothing:

$$P(w_i|w_{i-1}) = \frac{C(w_{i-1}, w_i) + \alpha}{C(w_{i-1}) + \alpha|V|}$$

Raw bigram counts
(Berkeley restaurant corpus)

- Out of 9222 sentences

	i	want	to	eat	chinese	food	lunch	spend
i	5	827	0	9	0	0	0	2
want	2	0	608	1	6	6	5	1
to	2	0	4	686	2	0	6	211
eat	0	0	2	0	16	2	42	0
chinese	1	0	0	0	0	82	1	0
food	15	0	15	0	1	4	0	0
lunch	2	0	0	0	0	1	0	0
spend	1	0	1	0	0	0	0	0

(Slide credit: Dan Jurafsky)

Smoothed bigram counts

	i	want	to	eat	chinese	food	lunch	spend
i	6	828	1	10	1	1	1	3
want	3	1	609	2	7	7	6	2
to	3	1	5	687	3	1	7	212
eat	1	1	3	1	17	3	43	1
chinese	2	1	1	1	1	83	2	1
food	16	1	16	1	2	5	1	1
lunch	3	1	1	1	1	2	1	1
spend	2	1	2	1	1	1	1	1

Add 1 to all the entries in the matrix

(Slide credit: Dan Jurafsky)

Smoothed bigram probabilities

$$P(w_i|w_{i-1}) = \frac{C(w_{i-1}, w_i) + \alpha}{C(w_{i-1}) + \alpha|V|} \qquad \alpha = 1$$

	i	want	to	eat	chinese	food	lunch	spend
i	0.0015	0.21	0.00025	0.0025	0.00025	0.00025	0.00025	0.00075
want	0.0013	0.00042	0.26	0.00084	0.0029	0.0029	0.0025	0.00084
to	0.00078	0.00026	0.0013	0.18	0.00078	0.00026	0.0018	0.055
eat	0.00046	0.00046	0.0014	0.00046	0.0078	0.0014	0.02	0.00046
chinese	0.0012	0.00062	0.00062	0.00062	0.00062	0.052	0.0012	0.00062
food	0.0063	0.00039	0.0063	0.00039	0.00079	0.002	0.00039	0.00039
lunch	0.0017	0.00056	0.00056	0.00056	0.00056	0.0011	0.00056	0.00056
spend	0.0012	0.00058	0.0012	0.00058	0.00058	0.00058	0.00058	0.00058

(Credits: Dan Jurafsky)

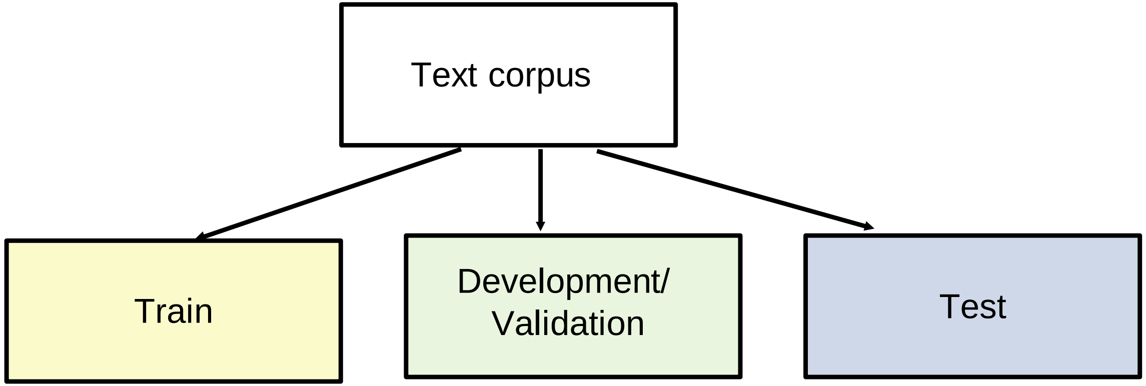
Linear Interpolation

$$\hat{P}(w_i \mid w_{i-2}, w_{i-1}) = \lambda_1 P(w_i \mid w_{i-2}, w_{i-1}) \quad \text{Trigram}$$
$$+\lambda_2 P(w_i | w_{i-1}) \quad \text{Bigram}$$
$$+\lambda_3 P(w_i) \quad \text{Unigram}$$

$$\sum_i \lambda_i = 1$$

- Use a combination of models to estimate probability
- Strong empirical performance

How can we choose lambdas?



- First, estimate n-gram prob. on training set
- Then, estimate lambdas (hyperparameters) to maximize probability on the held-out development/validation set
- Use best model from above to evaluate on test set

Discounting

- Determine some “mass” to remove from probability estimates
- More explicit method for redistributing mass among unseen n-grams
- Just choose an absolute value to discount (usually <1)

Absolute Discounting

- Define Count*(x) = Count(x) - 0.5
- Missing probability mass:

$$\alpha(w_{i-1}) = 1 - \sum_{w} \frac{\text{Count}^*(w_{i-1,w})}{\text{Count}(w_{i-1})}$$

$$\alpha(\text{the}) = 10 \times 0.5/48 = 5/48$$

- Divide this mass between words for which Count(the,) = 0

x	$\text{Count}(x)$	$\text{Count}^*(x)$	$\frac{\text{Count}^*(x)}{\text{Count}(x)}$
the	48		
the, dog	15	14.5	14.5/48
the, woman	11	10.5	10.5/48
the, man	10	9.5	9.5/48
the, park	5	4.5	4.5/48
the, job	2	1.5	1.5/48
the, telescope	1	0.5	0.5/48
the, manual	1	0.5	0.5/48
the, afternoon	1	0.5	0.5/48
the, country	1	0.5	0.5/48
the, street	1	0.5	0.5/48

Part 3:

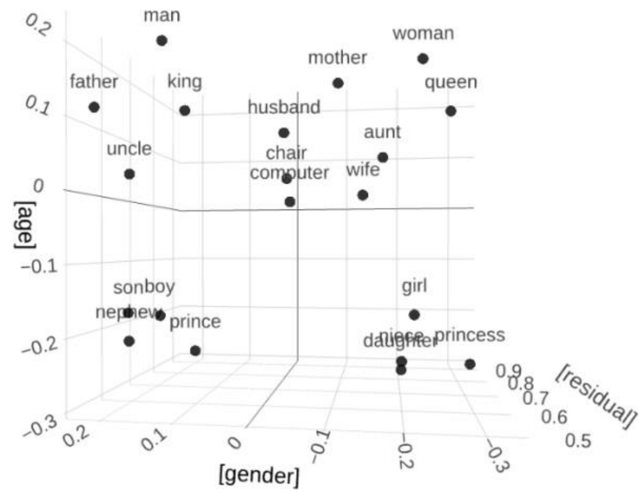
Word representation in modern NLP

The big idea: model of meaning focusing on similarity

$$v_{\text{cat}} = \begin{pmatrix} -0.224 \\ 0.130 \\ -0.290 \\ 0.276 \end{pmatrix} \qquad v_{\text{dog}} = \begin{pmatrix} -0.124 \\ 0.430 \\ -0.200 \\ 0.329 \end{pmatrix}$$

$$v_{\text{the}} = \begin{pmatrix} 0.234 \\ 0.266 \\ 0.239 \\ -0.199 \end{pmatrix} \qquad v_{\text{language}} = \begin{pmatrix} 0.290 \\ -0.441 \\ 0.762 \\ 0.982 \end{pmatrix}$$

Similar words are "**nearby in the vector space**"



(Bandyopadhyay et al. 2022)

How do we represent words in NLP models?

- n-gram models

$$P(w_1, w_2, ...w_n) = \prod_{i=1}^{n} P(w_i|w_{i-1})$$

$$P(w_i|w_{i-1}) = \frac{\boxed{C(w_{i-1}, w_i)} + \alpha}{C(w_{i-1}) + \alpha|V|}$$

- Naive Bayes

$$\hat{P}(w_i \mid c_j) = \frac{\boxed{\text{Count}(w_i, c_j)} + \alpha}{\sum_{w \in V} \text{Count}(w, c_j) + \alpha|V|}$$

Each word is just a string or indices w_i in the vocabulary list

cat = the 5th word in

dog = the 10th word in

cats = the 118th word in

How do we represent words in NLP models?

- Logistic regression

string match

Var	Definition	Value in Fig. 5.2
x_1	count(positive lexicon) $\in$ doc)	3
x_2	count(negative lexicon) $\in$ doc)	2
x_3	$\begin{cases} 1 & \text{if “no”} \in \text{doc} \\ 0 & \text{otherwise} \end{cases}$	1
x_4	count(1st and 2nd pronouns $\in$ doc)	3
x_5	$\begin{cases} 1 & \text{if “!”} \in \text{doc} \\ 0 & \text{otherwise} \end{cases}$	0
x_6	$\log(\text{word count of doc})$	$\ln(64) = 4.15$

What do words mean?

- **Synonyms**: couch/sofa, car/automobile, filbert/hazelnut
- **Antonyms**: dark/light, rise/fall, up/down
- Some words are not synonyms but they share some element of meaning
  - cat/dog, car/bicycle, cow/horse
- Some words are not similar but they are **related**
  - coffee/cup, house/door, chef/menu
- **Affective meanings** or **connotations**:

vanish	disappear	9.8
belief	impression	5.95
muscle	bone	3.65
modest	flexible	0.98
hole	agreement	0.3

SimLex-999

	Valence	Arousal	Dominance
courageous	8.05	5.5	7.38
music	7.67	5.57	6.5
heartbreak	2.45	5.65	3.58
cub	6.71	3.95	4.24

(Osgood et al., 1957)

valence: the pleasantness of the stimulus

arousal: the intensity of emotion provoked by the stimulus

dominance: the degree of control exerted by the stimulus

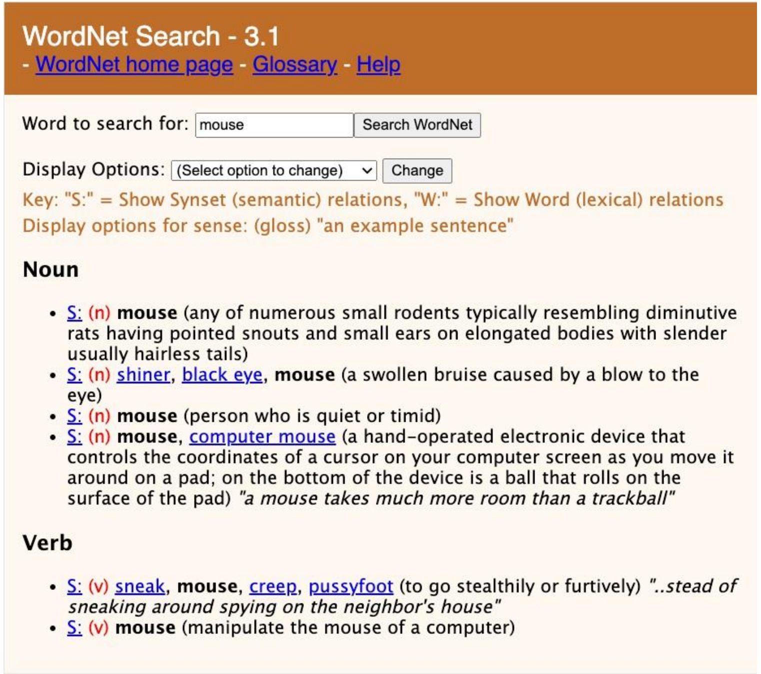
Why word meaning in NLP models?

- With words, a feature is a word identity (= string)
  - Feature 5: `The previous word was "terrible"'
  - Requires **exact same word** to be in the training and testing set

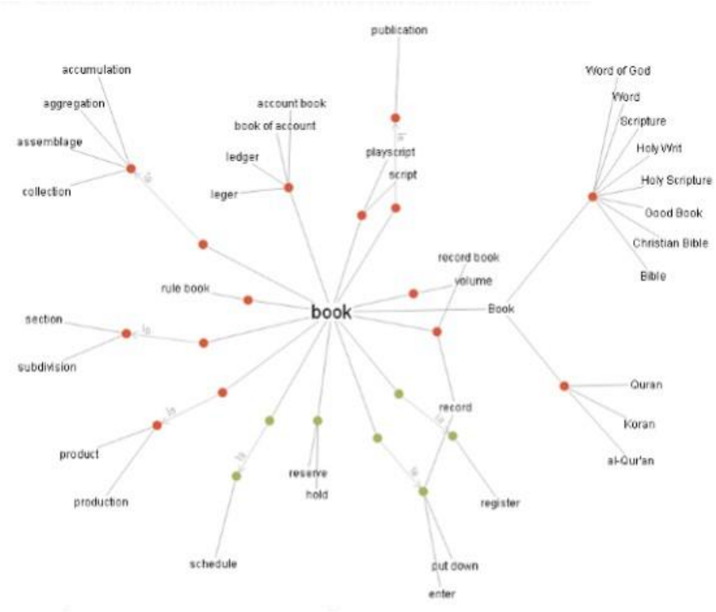
"terrible" ≠ "horrible"

- If we can represent word meaning in vectors:
  - The previous word was vector [35, 22, 17, …]
  - Now in the test set we might see a similar vector [34, 21, 14, …]
  - We can generalize to **similar but unseen** words!!!

Lexical resources



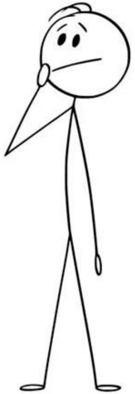
http://wordnetweb.princeton.edu/



(-) Huge amounts of human labor to create and maintain

Distributional hypothesis

Distributional hypothesis



- "The meaning of a word is its use in the language" [Wittgenstein PI 43]
- "If A and B have almost identical environments we say that they are synonyms." [Harris 1954]
- "You shall know a word by the company it keeps" [Firth 1957]

Distributional hypothesis

Distributional hypothesis: words that occur in similar **contexts** tend to have similar meanings

J.R.Firth 1957

- “You shall know a word by the company it keeps”
- One of the most successful ideas of modern statistical NLP!

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

…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 context words will represent “*banking”.*

Distributional hypothesis

“Ongchoi”

Ongchoi is delicious sautéed with garlic

Ongchoi is superb over rice

Ongchoi leaves with salty sauces

Distributional hypothesis

"Ongchoi"

Ongchoi is delicious sautéed with garlic

Ongchoi is superb over rice

Ongchoi leaves with salty sauces

Q: What do you think 'Ongchoi' means?

A) a savory snack
B) a green vegetable
C) an alcoholic beverage
D) a cooking sauce

Distributional hypothesis

“Ongchoi”

Ongchoi is delicious sautéed with garlic

Ongchoi is superb over rice

Ongchoi leaves with salty sauces

You may have seen these sentences before:

spinach **sautéed with garlic over rice**

chard stems and **leaves** are **delicious**

collard greens and other **salty** leafty greens

Distributional hypothesis

“Ongchoi”

Ongchoi is a leafty green like spinach, chard or collard greens

空心菜
kangkong
rau muống
...

How can do the same thing computationally?

- Count the words in the context of ongchoi
- See what other words occur in those contexts

We can represent a word's context using vectors!

Sparse vs dense vectors

Words and vectors

Q: What is the dimension of each such vector?

A: |V|

First solution: Let's use **word-word co-occurrence counts** to represent the meaning of words!

Each word is represented by the corresponding **row vector**

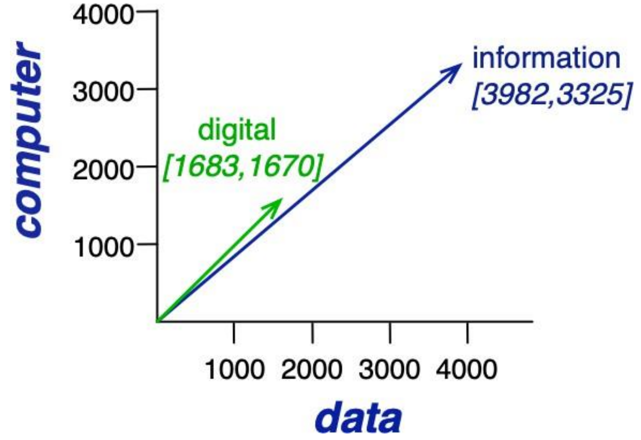
context words:
4 words to the left +
4 words to the right

is traditionally followed by	**cherry**	pie, a traditional dessert
often mixed, such as	**strawberry**	rhubarb pie. Apple pie
computer peripherals and personal	**digital**	assistants. These devices usually
a computer. This includes	**information**	available on the internet

	aardvark	...	computer	data	result	pie	sugar	...
cherry	0	...	2	8	9	442	25	...
strawberry	0	...	0	0	1	60	19	...
digital	0	...	1670	1683	85	5	4	...
information	0	...	3325	3982	378	5	13	...

Most entries are 0s $\Longrightarrow$ sparse vectors

Measuring similarity



A common similarity metric: **cosine** of the angle between the two vectors (the larger, the more similar the two vectors are)

$$\cos(\mathbf{u}, \mathbf{v}) = \frac{\mathbf{u} \cdot \mathbf{v}}{\|\mathbf{u}\|\|\mathbf{v}\|}$$

$$\cos(\mathbf{u}, \mathbf{v}) = \frac{\sum_{i=1}^{|V|} u_i v_i}{\sqrt{\sum_{i=1}^{|V|} u_i^2}\sqrt{\sum_{i=1}^{|V|} v_i^2}}$$

Q: Why cosine similarity instead of dot product $\mathbf{u} \cdot \mathbf{v}$?

What is the range of $\cos(u, v)$ if u, v are **count vectors**?

(a) [-1, 1]

(b) [0, 1]

(c) $[0, +\infty)$

(d) $(-\infty, +\infty)$

$$\cos(\mathbf{u}, \mathbf{v}) = \frac{\sum_{i=1}^{|V|} u_i v_i}{\sqrt{\sum_{i=1}^{|V|} u_i^2} \sqrt{\sum_{i=1}^{|V|} v_i^2}}$$

What is the range of $\cos(u, v)$ if u, v are **count vectors**?

(a) [-1, 1]

(b) [0, 1]

(c) $[0, +\infty)$

(d) $(-\infty, +\infty)$

$$\cos(\mathbf{u}, \mathbf{v}) = \frac{\sum_{i=1}^{|V|} u_i v_i}{\sqrt{\sum_{i=1}^{|V|} u_i^2}\sqrt{\sum_{i=1}^{|V|} v_i^2}}$$

The answer is (b). Cosine similarity ranges between -1 and 1 in general. In this model, all the values of u_i, v_i are non-negative.

Any issues with this model?

Raw frequency count is a bad representation!

- Frequency is clearly useful; if “pie” appears a lot near “cherry”, that's useful information.
- But overly frequent words like “the”, “it", or “they” also appear a lot near “cherry”. They are not very informative about the context.

Sparse vs dense vectors

- The vectors in the word-word occurrence matrix are
  - **Long**: vocabulary size
  - **Sparse**: most are 0's

- Alternative: we want to represent words as **short** (50-300 dimensional) & **dense** (real-valued) vectors

  - The basis for modern NLP systems

$$v_{\text{cat}} = \begin{pmatrix} -0.224 \\ 0.130 \\ -0.290 \\ 0.276 \end{pmatrix} \quad v_{\text{dog}} = \begin{pmatrix} -0.124 \\ 0.430 \\ -0.200 \\ 0.329 \end{pmatrix}$$

$$v_{\text{the}} = \begin{pmatrix} 0.234 \\ 0.266 \\ 0.239 \\ -0.199 \end{pmatrix} \quad v_{\text{language}} = \begin{pmatrix} 0.290 \\ -0.441 \\ 0.762 \\ 0.982 \end{pmatrix}$$

Why dense vectors?

- Short vectors are easier to use as **features** in ML systems
- Dense vectors generalize better than explicit counts (points in real space vs points in integer space)
- Sparse vectors can’t capture higher-order co-occurrence
  - w_1 co-occurs with “car”, w_2 co-occurs with “automobile”
  - They should be similar but they aren’t because “car” and “automobile” are distinct dimensions
- In practice, they work better!

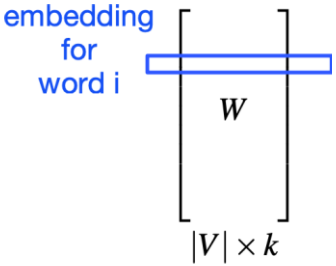
How to get short dense vectors?

- **Count-based methods**: Singular value decomposition (SVD) of count matrix

Singular value decomposition (SVD) of PPMI weighted co-occurrence matrix

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

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



We can approximate the full matrix by only keeping the top k (e.g., 100) singular values!

How to get short dense vectors?

- **Count-based methods**: Singular value decomposition (SVD) of count matrix

- **Prediction-based methods**:
  - Vectors are created by training a classifier to predict whether a word *c* ("pie") is likely to appear in the context of a word w ("cherry")
  - Examples: **word2vec** (Mikolov et al., 2013), **Glove** (Pennington et al., 2014), **FastText** (Bojanowski et al., 2017)

***Don't count, predict!* A systematic comparison of context-counting vs. context-predicting semantic vectors**

Marco Baroni and **Georgiana Dinu** and **Germán Kruszewski**
Center for Mind/Brain Sciences (University of Trento, Italy)

(Baroni et al., 2014)

Also called word **embeddings!**

Part 4:

Word2vec

Word embeddings

Goal: represent words as **short** (50-300 dimensional) & **dense** (real-valued) vectors

Count-based approaches

- Used since the 90s
- Sparse word-word co-occurrence PPMI matrix
- Decomposed with SVD

Prediction-based approaches

- Formulated as a machine learning problem
- Word2vec (Mikolov et al., 2013)
- GloVe (Pennington et al., 2014)

Underlying theory: Distributional Hypothesis *(Firth, '57)*

"Similar words occur in similar contexts"

Word embeddings: the learning problem

Learning vectors from text for representing words

- **Input**: a large text corpus, vocabulary *V*, vector dimension d (e.g., 300)
- **Output**: $f : V \to \mathbb{R}^d$

Each coordinate/dimension of the vector doesn't have a particular interpretation

$$v_{\text{cat}} = \begin{pmatrix} -0.224 \\ 0.130 \\ -0.290 \\ 0.276 \end{pmatrix} \qquad v_{\text{dog}} = \begin{pmatrix} -0.124 \\ 0.430 \\ -0.200 \\ 0.329 \end{pmatrix}$$

$$v_{\text{the}} = \begin{pmatrix} 0.234 \\ 0.266 \\ 0.239 \\ -0.199 \end{pmatrix} \qquad v_{\text{language}} = \begin{pmatrix} 0.290 \\ -0.441 \\ 0.762 \\ 0.982 \end{pmatrix}$$

Word embeddings

- Basic property: similar words have similar vectors

word w^*= “sweden”

$$\arg\max_{w \in V} \cos(e(w), e(w^*))$$

Word	Cosine distance
norway	0.760124
denmark	0.715460
finland	0.620022
switzerland	0.588132
belgium	0.585835
netherlands	0.574631
iceland	0.562368
estonia	0.547621
slovenia	0.531408

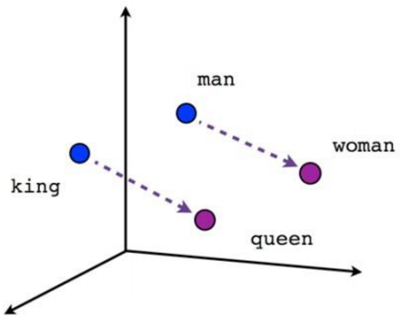
$\cos(u, v)$ ranges between -1 and 1

Word embeddings

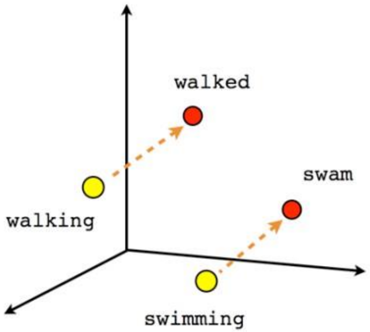
ACL'19 **Towards Understanding Linear Word Analogies**

Kawin Ethayarajh, David Duvenaud†**, Graeme Hirst**
University of Toronto
†Vector Institute
{kawin, duvenaud, gh}@cs.toronto.edu

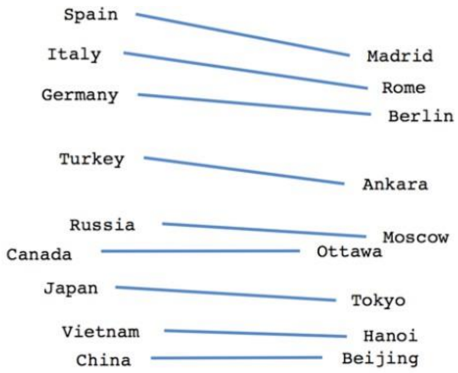
- They have some other nice properties too!



Male-Female



Verb tense



Country-Capital

$$v_{\text{man}} - v_{\text{woman}} \approx v_{\text{king}} - v_{\text{queen}}$$

$$v_{\text{Paris}} - v_{\text{France}} \approx v_{\text{Rome}} - v_{\text{Italy}}$$

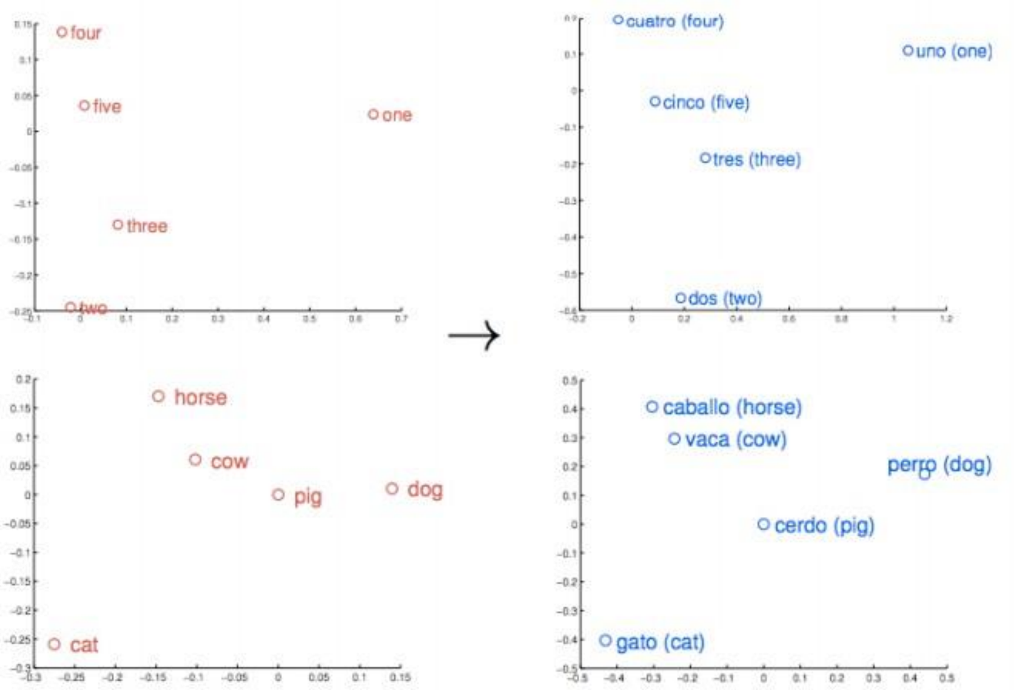
Word analogy test: $a : a^* :: b : b^*$

$$b^* = \arg\max_{w \in V} \cos(e(w), e(a^*) - e(a) + e(b))$$

Word embeddings

- They have some other nice properties too!

$$v(\text{cuatro}) \approx W v(\text{four})$$

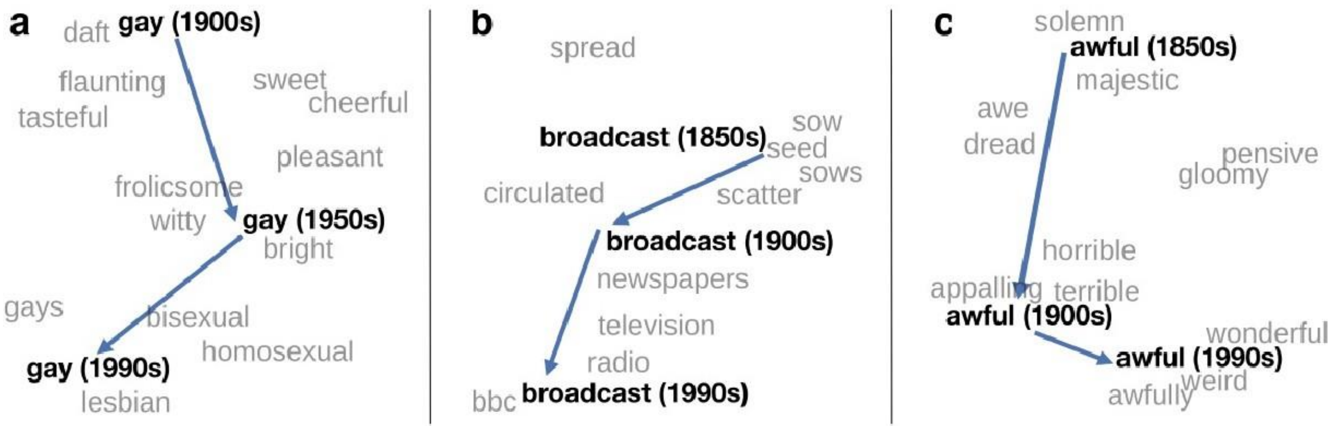


(Mikolov et al, 2013): Exploiting Similarities among Languages for Machine Translation

Embeddings as a window onto historical semantics

Train embeddings on different decades of historical text to see meanings shift

~30 million books, 1850-1990, Google Books data



William L. Hamilton, Jure Leskovec, and Dan Jurafsky. 2016. Diachronic Word Embeddings Reveal Statistical Laws of Semantic Change. Proceedings of ACL.

Embeddings reflect cultural bias!

Bolukbasi, Tolga, Kai-Wei Chang, James Y. Zou, Venkatesh Saligrama, and Adam T. Kalai. "Man is to computer programmer as woman is to homemaker? debiasing word embeddings." In *NeurIPS*, pp. 4349-4357. 2016.

Ask "Paris : France :: Tokyo : x"

- x = Japan

Ask "father : doctor :: mother : x"

- x = nurse

Ask "man : computer programmer :: woman : x"

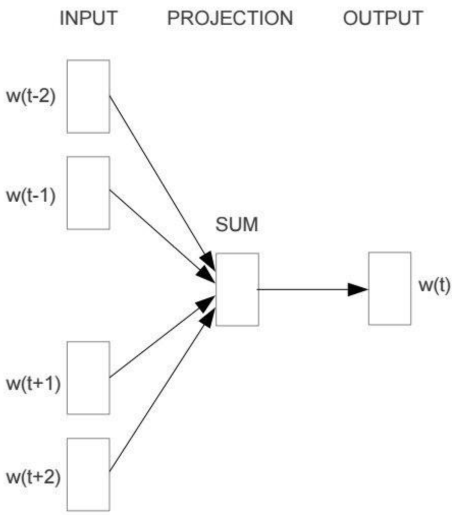
- x = homemaker

Algorithms that use embeddings as part of e.g., hiring searches for programmers, might lead to bias in hiring

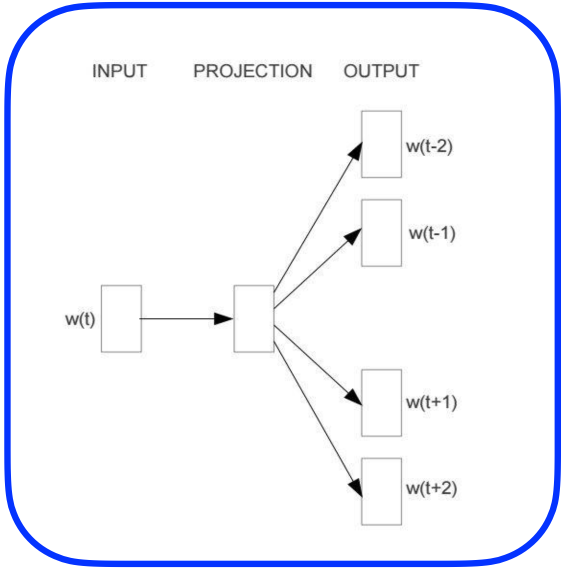
word2vec

- (Mikolov et al 2013a): Efficient Estimation of Word Representations in Vector Space
- (Mikolov et al 2013b): Distributed Representations of Words and Phrases and their Compositionality

Thomas
Mikolov



Continuous Bag of Words
(CBOW)

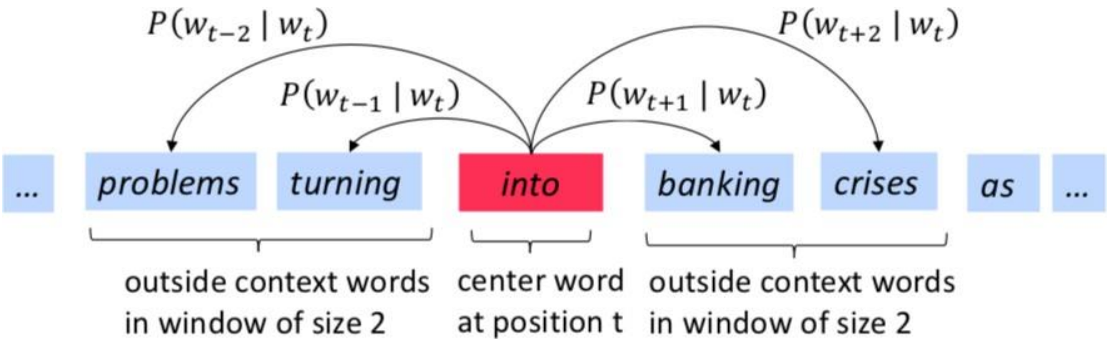


Skip-gram

Skip-gram

- Assume that we have a large corpus $w_1, w_2, \ldots, w_T \in V$
- **Key idea:** Use each word to **predict** other words in its context
- Context: a fixed window of size 2*m* (m = 2 in the example)

A classification problem!

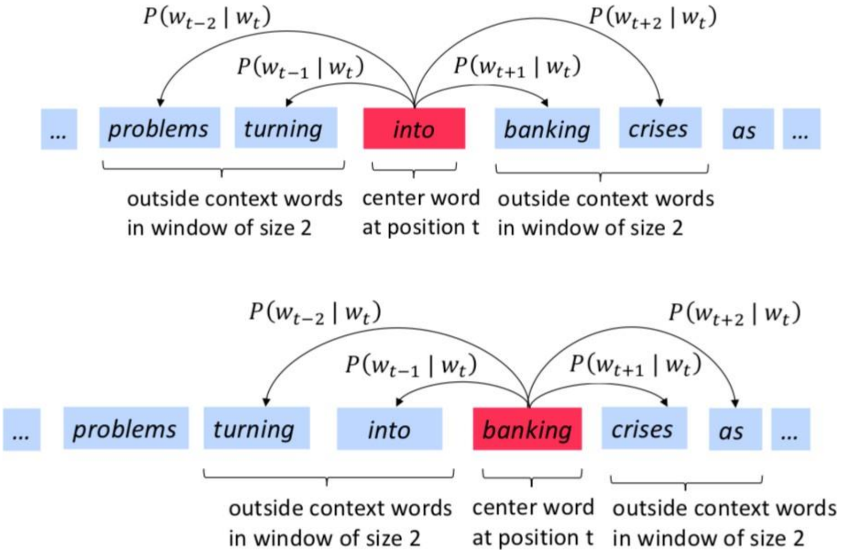


$P(b \mid a)$ = given the center word is a, what is the probability that b is a context word?

$P(\cdot \mid a)$ is a probability distribution defined over V: $\sum_{w \in V} P(w \mid a) = 1$

We are going to define this distribution soon!

Skip-gram



Convert the training data into:
(into, problems)
(into, turning)
(into, banking)
(into, crises)
(banking, turning)
(banking, into)
(banking, crises)
(banking, as)
…

Our goal is to find parameters that can maximize

P(problems | into) × P(turning | into) × P(banking | into) × P(crises | into) × P(turning | banking) × P(into | banking) × P(crises | banking) × P(as | banking)...

Skip-gram: objective function

- For each position t = $1,2,\ldots T$, predict context words within context size m, given center word w_t:

all the parameters to be optimized

$$\mathcal{L}(\theta) = \prod_{t=1}^{T} \prod_{-m \leq j \leq m, j \neq 0} P(w_{t+j} \mid w_t; \theta)$$

- It is equivalent as minimizing the (average) negative log likelihood:

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

How to define $P(w_{t+j} \mid w_t; \theta)$?

- Use two sets of vectors for each word in the vocabulary

$$\mathbf{u}_a \in \mathbb{R}^d\text{: vector for center word a , } \forall a \in V$$

$$\mathbf{v}_b \in \mathbb{R}^d\text{: vector for context word b , } \forall b \in V$$

- Use inner product $\mathbf{u}_a \cdot \mathbf{v}_b$ to measure how likely word *a* appears with context word *b*

Softmax we have seen in multinomial logistic regression!

$$P(w_{t+j} \mid w_t) = \frac{\exp(\mathbf{u}_{w_t} \cdot \mathbf{v}_{w_{t+j}})}{\sum_{k \in V} \exp(\mathbf{u}_{w_t} \cdot \mathbf{v}_k)}$$

Recall that $P(\cdot \mid a)$ is a probability distribution defined over V…

… vs multinominal logistic regression

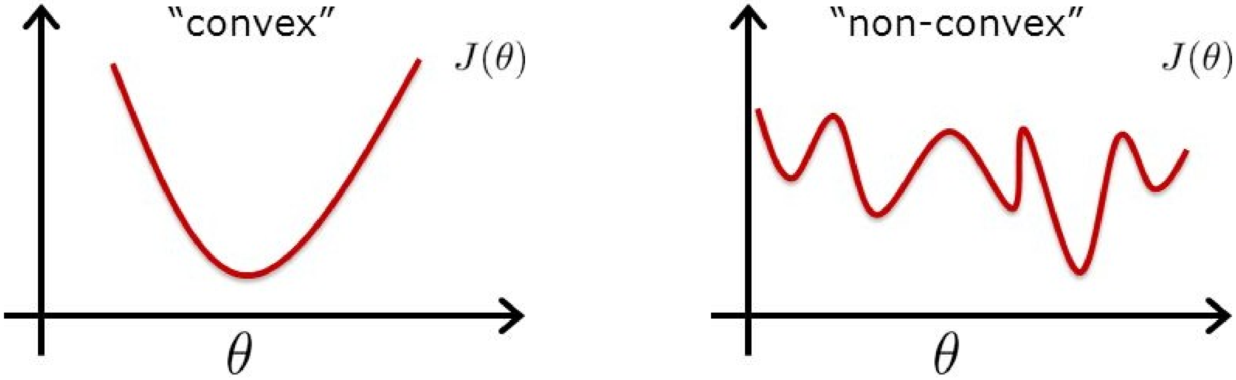
Multinomial logistic regression:

$$P(y = c \mid x) = \frac{\exp(\mathbf{w}_c \cdot \mathbf{x} + b_c)}{\sum_{j=1}^{m} \exp(\mathbf{w}_j \cdot \mathbf{x} + b_j)}$$

$$P(w_{t+j} \mid w_t) = \frac{\exp(\mathbf{u}_{w_t} \cdot \mathbf{v}_{w_{t+j}})}{\sum_{k \in V} \exp(\mathbf{u}_{w_t} \cdot \mathbf{v}_k)}$$

- Essentially a |V|-way classification problem
- If we fix $\mathbf{u}_{w_t}$, it is reduced to a multinomial logistic regression problem.
- However, since we have to learn both and together, the training objective is non-convex.

… vs multinominal logistic regression



- It is hard to find a global minimum
- But can still use stochastic gradient descent to optimize
  :

$$\theta^{(t+1)} = \theta^{(t)} - \eta \nabla_{\theta} J(\theta)$$

Important note

$$J(\theta) = -\frac{1}{T}\sum_{t=1}^{T}\sum_{-m\leq j\leq m, j\neq 0}\log\frac{\exp(\mathbf{u}_{w_t}\cdot\mathbf{v}_{w_{t+j}})}{\sum_{k\in V}\exp(\mathbf{u}_{w_t}\cdot\mathbf{v}_k)}$$

- In this formulation, we don't care about the classification task itself like we do for the logistic regression model we saw previously.

- The key point is that the *parameters* used to optimize this training objective—when the training corpus is large enough—can give us very good representations of words (following the principle of distributional hypothesis)!

How many parameters in this model?

$$J(\theta) = -\frac{1}{T}\sum_{t=1}^{T}\sum_{-m\leq j\leq m, j\neq 0}\log\frac{\exp(\mathbf{u}_{w_t}\cdot\mathbf{v}_{w_{t+j}})}{\sum_{k\in V}\exp(\mathbf{u}_{w_t}\cdot\mathbf{v}_k)}$$

How many parameters does this model have (i.e. what is size of)?

(a) $d|V|$

(b) $2d|V|$

(c) $2m|V|$

(d) $2md|\mathrm{V}|$

d = dimension of each vector

How many parameters in this model?

$$J(\theta) = -\frac{1}{T} \sum_{t=1}^{T} \sum_{-m \leq j \leq m, j \neq 0} \log \frac{\exp(\mathbf{u}_{w_t} \cdot \mathbf{v}_{w_{t+j}})}{\sum_{k \in V} \exp(\mathbf{u}_{w_t} \cdot \mathbf{v}_k)}$$

How many parameters does this model have (i.e. what is size of)?

(a) $d|V|$

(b) $2d|V|$

(c) $2m|V|$

(d) $2md|V|$

d = dimension of each vector

The answer is (b).

Each word has two d-dimensional vectors, so it is $2 \times |V| \times d$.

word2vec formulation

$$J(\theta) = -\frac{1}{T}\sum_{t=1}^{T}\sum_{-m\leq j\leq m, j\neq 0}\log\frac{\exp(\mathbf{u}_{w_t}\cdot\mathbf{v}_{w_{t+j}})}{\sum_{k\in V}\exp(\mathbf{u}_{w_t}\cdot\mathbf{v}_k)}$$

Q: Why do we need two vectors for each word instead of one?

A: because one word is not likely to appear in its own context window, e.g., P(dog | dog) should be low. If we use one set of vectors only, it essentially needs to minimize $\mathbf{u}_{\text{dog}} \cdot \mathbf{u}_{\text{dog}}$..

Q: Which set of vectors are used as word embeddings?

A: This is an empirical question. Typically just $\mathbf{u}_w$ but you can also concatenate the two vectors..

Word2vec and its variants

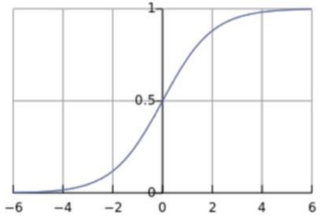
Skip-gram with negative sampling (SGNS)

Problem: every time you get one pair of (t, c), you need to update $\mathbf{v}_k$ with all the words in the vocabulary! This is very expensive computationally.

$$\frac{\partial y}{\partial \mathbf{u}_t} = -\mathbf{v}_c + \sum_{k \in V} P(k \mid t)\mathbf{v}_k \qquad \frac{\partial y}{\partial \mathbf{v}_k} = \begin{cases} (P(k \mid t) - 1)\,\mathbf{u}_t & k = c \\ P(k \mid t)\mathbf{u}_t & k \neq c \end{cases}$$

Negative sampling: instead of considering all the words in V, let's randomly sample K (5-20) negative examples.

$$\sigma(x) = \frac{1}{1 + \exp(-x)}$$



softmax: $$y = -\log\left(\frac{\exp(\mathbf{u}_t \cdot \mathbf{v}_c)}{\sum_{k \in V} \exp(\mathbf{u}_t \cdot \mathbf{v}_k)}\right)$$

Negative sampling: $$y = -\log(\sigma(\mathbf{u}_t \cdot \mathbf{v}_c)) - \sum_{i=1}^{K} \mathbb{E}_{j \sim P(w)} \log(\sigma(-\mathbf{u}_t \cdot \mathbf{v}_j))$$

Skip-gram with negative sampling (SGNS)

Key idea: Convert the $|V|$ -way classification into a set of binary classification tasks.

Every time we get a pair of words (*t*, *c*), we don't predict *c* among all the words in the vocabulary. Instead, we predict (*t*, *c*) is a positive pair, and (*t*, *c'*) is a negative pair for a small number of sampled *c'*.

positive examples +

t	c
apricot	tablespoon
apricot	of
apricot	jam
apricot	a

negative examples -

t	c	t	c
apricot	aardvark	apricot	seven
apricot	my	apricot	forever
apricot	where	apricot	dear
apricot	coaxial	apricot	if

$$y = -\log(\sigma(\mathbf{u}_t \cdot \mathbf{v}_c)) - \sum_{i=1}^{K} \mathbb{E}_{j \sim P(w)} \log(\sigma(-\mathbf{u}_t \cdot \mathbf{v}_j))$$

P(w): sampling according to the frequency of words

Similar to **binary logistic regression**, but we need to optimize and together.

$$P(y = 1 \mid t, c) = \sigma(\mathbf{u}_t \cdot \mathbf{v}_c) \qquad p(y = 0 \mid t, c') = 1 - \sigma(\mathbf{u}_t \cdot \mathbf{v}_{c'}) = \sigma(-\mathbf{u}_t \cdot \mathbf{v}_{c'})$$

Understanding SGNS

$$y = -\log(\sigma(\mathbf{u}_t \cdot \mathbf{v}_c)) - \sum_{i=1}^{K} \mathbb{E}_{j \sim P(w)} \log(\sigma(-\mathbf{u}_t \cdot \mathbf{v}_j))$$

In skip-gram with negative sampling (SGNS), how many parameters need to be updated in θ for every (t, c) pair?

(a) Kd

(b) $2Kd$

(c) $(K+1)d$

(d) $(K+2)d$

Understanding SGNS

$$y = -\log(\sigma(\mathbf{u}_t \cdot \mathbf{v}_c)) - \sum_{i=1}^{K} \mathbb{E}_{j \sim P(w)} \log(\sigma(-\mathbf{u}_t \cdot \mathbf{v}_j))$$

In skip-gram with negative sampling (SGNS), how many parameters need to be updated in θ for every (t, c) pair?

(a) Kd

(b) $2Kd$

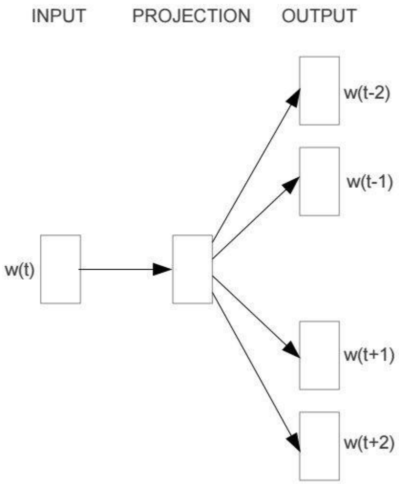
(c) $(K + 1)d$

(d) $(K + 2)d$

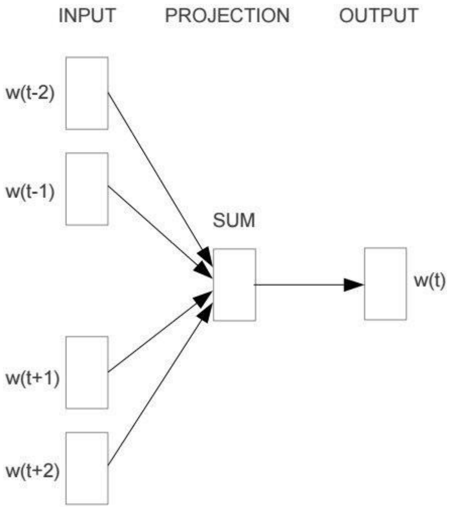
The answer is (d).

We need to calculate gradients with respect to $\mathbf{u}_t$ and (K + 1) $\mathbf{v}_i$ (one positive and K negatives).

Continuous Bag of Words (CBOW)



Skip-gram



Continuous Bag of Words (CBOW)

$$L(\theta) = \prod_{t=1}^{T} P\left(w_t \mid \{w_{t+j}\}, -m \leq j \leq m, j \neq 0\right)$$

$$\bar{\mathbf{v}}_t = \frac{1}{2m} \sum_{-m \leq j \leq m, j \neq 0} \mathbf{v}_{t+j}$$

$$P(w_t \mid \{w_{t+j}\}) = \frac{\exp(\mathbf{u}_{w_t} \cdot \bar{\mathbf{v}}_t)}{\sum_{k \in V} \exp(\mathbf{u}_k \cdot \bar{\mathbf{v}}_t)}$$

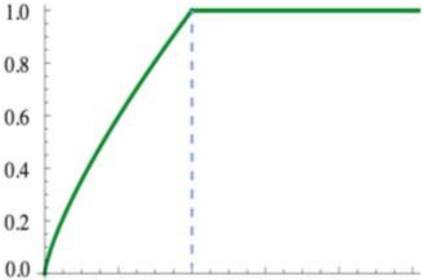
GloVe: **Glo**bal **Ve**ctors

- Key idea: let's approximate $\mathbf{u}_i \cdot \mathbf{v}_j$ using their co-occurrence counts directly
- Take the global co-occurrence statistics: $X_{i,j}$

$$J(\theta) = \sum_{i,j \in V} f(X_{i,j}) \left(\mathbf{u}_i \cdot \mathbf{v}_j + b_i + \tilde{b}_j - \log X_{i,j} \right)^2$$

- Training faster
- Scalable to very large corpora

$f \sim$



(Pennington et al, 2014): GloVe: Global Vectors for Word Representation

Trained word embeddings available

- word2vec: https://code.google.com/archive/p/word2vec/
- GloVe: https://nlp.stanford.edu/projects/glove/
- FastText: https://fasttext.cc/

Download pre-trained word vectors

- Pre-trained word vectors. This data is made available under the Public Domain Dedication and License v1.0 whose full text can be found at: http://www.opendatacommons.org/licenses/pddl/1.0/.
  - Wikipedia 2014 + Gigaword 5 (6B tokens, 400K vocab, uncased, 50d, 100d, 200d, & 300d vectors, 822 MB download): glove.6B.zip
  - Common Crawl (42B tokens, 1.9M vocab, uncased, 300d vectors, 1.75 GB download): glove.42B.300d.zip
  - Common Crawl (840B tokens, 2.2M vocab, cased, 300d vectors, 2.03 GB download): glove.840B.300d.zip
  - Twitter (2B tweets, 27B tokens, 1.2M vocab, uncased, 25d, 50d, 100d, & 200d vectors, 1.42 GB download): glove.twitter.27B.zip
- Ruby script for preprocessing Twitter data

Differ in algorithms, text corpora, dimensions, cased/uncased…

Applied to many other languages

Easy to use!

```python
from gensim.models import KeyedVectors
# Load vectors directly from the file
model = KeyedVectors.load_word2vec_format('data/GoogleGoogleNews-vectors-negative300.bin', binary=True)
# Access vectors for specific words with a keyed lookup:
vector = model['easy']
```

```
In [17]: model.similarity('straightforward', 'easy')
Out[17]: 0.5717043285477517

In [18]: model.similarity('simple', 'impossible')
Out[18]: 0.29156160264633707

In [19]: model.most_similar('simple')
Out[19]: [('straightforward', 0.7460169196128845),
          ('Simple', 0.7108174562454224),
          ('uncomplicated', 0.6297484636306763),
          ('simplest', 0.6171397566795349),
          ('easy', 0.5990299582481384),
          ('fairly_straightforward', 0.5893306732177734),
          ('deceptively_simple', 0.5743066072463989),
          ('simpler', 0.5537199378013611),
          ('simplistic', 0.5516539216041565),
          ('disarmingly_simple', 0.5365327000617981)]
```

Evaluating Word2vec

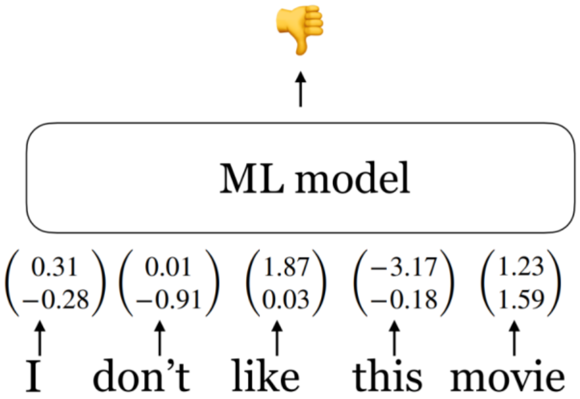
Extrinsic vs intrinsic evaluation

Extrinsic evaluation

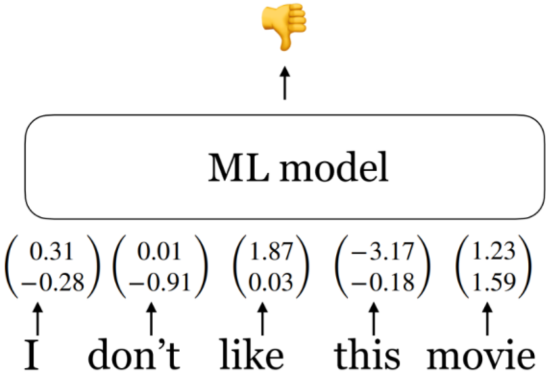
- Let's plug these word embeddings into a real NLP system and see whether this improves performance
- Could take a long time but still the most important evaluation metric

Intrinsic evaluation

- Evaluate on a specific/intermediate subtask
- Fast to compute
- Not clear if it really helps downstream tasks



Extrinsic evaluation



A straightforward solution: given an input sentence $x_1, x_2, \ldots, x_n$

Instead of using a bag-of-words model, we can compute $vec(x) = \mathbf{e}(x_1) + \mathbf{e}(x_2) + \ldots + \mathbf{e}(x_n)$

And then train a logistic regression classifier on $vec(x)$ as we did before!

There are much better ways to do this e.g., take word embeddings as input of neural networks

Intrinsic evaluation: word similarity

Word similarity

Example dataset: wordsim-353
353 pairs of words with human judgement

http://www.cs.technion.ac.il/~gabr/resources/data/wordsim353/

Word 1	Word 2	Human (mean)
tiger	cat	7.35
tiger	tiger	10
book	paper	7.46
computer	internet	7.58
plane	car	5.77
professor	doctor	6.62
stock	phone	1.62
stock	CD	1.31
stock	jaguar	0.92

Cosine similarity:

$$\cos(\boldsymbol{u}_i, \boldsymbol{u}_j) = \frac{\boldsymbol{u}_i \cdot \boldsymbol{u}_j}{||\boldsymbol{u}_i||_2 \times ||\boldsymbol{u}_j||_2}.$$

Metric: Spearman rank correlation

Intrinsic evaluation: word similarity

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	<u>72.7</u>	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	<u>58.1</u>	37.2
GloVe	6B	<u>65.8</u>	<u>72.7</u>	<u>77.8</u>	53.9	<u>38.1</u>
SVD-L	42B	74.0	76.4	74.1	58.3	39.9
GloVe	42B	**<u>75.9</u>**	**<u>83.6</u>**	**<u>82.9</u>**	**<u>59.6</u>**	**<u>47.8</u>**
CBOW*	100B	68.4	79.6	75.4	59.4	45.5

SG: Skip-gram

Intrinsic evaluation: word analogy

Word analogy test: $a : a^* :: b : b^*$

$$b^* = \arg\max_{w \in V} \cos(e(w), e(a^*) - e(a) + e(b))$$

semantic

Chicago:Illinois Philadelphia: ?

syntactic

bad:worst cool: ?

More examples at

http://download.tensorflow.org/data/questions-words.txt

Metric: accuracy

Intrinsic evaluation: word analogy

Model	Dim.	Size	Sem.	Syn.	Tot.
ivLBL	100	1.5B	55.9	50.1	53.2
HPCA	100	1.6B	4.2	16.4	10.8
GloVe	100	1.6B	67.5	54.3	60.3
SG	300	1B	61	61	61
CBOW	300	1.6B	16.1	52.6	36.1
vLBL	300	1.5B	54.2	64.8	60.0
ivLBL	300	1.5B	65.2	63.0	64.0
GloVe	300	1.6B	80.8	61.5	70.3
SVD	300	6B	6.3	8.1	7.3
SVD-S	300	6B	36.7	46.6	42.1
SVD-L	300	6B	56.6	63.0	60.1
CBOW†	300	6B	63.6	67.4	65.7
SG†	300	6B	73.0	66.0	69.1
GloVe	300	6B	77.4	67.0	71.7
CBOW	1000	6B	57.3	68.9	63.7
SG	1000	6B	66.1	65.1	65.6
SVD-L	300	42B	38.4	58.2	49.2
GloVe	300	42B	**81.9**	**69.3**	**75.0**

Thanks

How to train word2vec?

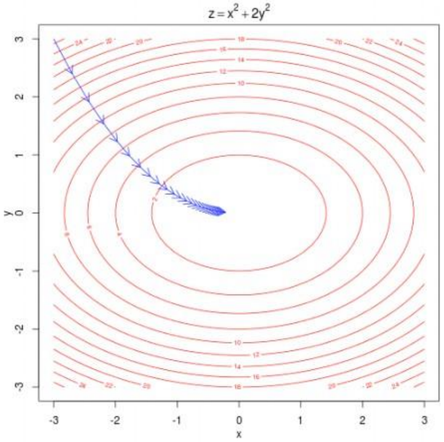
How to train this model?

$$J(\theta) = -\frac{1}{T}\sum_{t=1}^{T}\sum_{-m\leq j\leq m, j\neq 0}\log\frac{\exp(\mathbf{u}_{w_t}\cdot\mathbf{v}_{w_{t+j}})}{\sum_{k\in V}\exp(\mathbf{u}_{w_t}\cdot\mathbf{v}_k)}$$

- To train such a model, we need to compute the vector gradient $\nabla_\theta J(\theta) = ?$

- Again, θ represents all $2d\,|\,V\,|$ model parameters, in one vector.

$$\theta = \begin{bmatrix} v_{aardvark} \\ v_a \\ \vdots \\ v_{zebra} \\ u_{aardvark} \\ u_a \\ \vdots \\ u_{zebra} \end{bmatrix}$$



Vectorized gradients

$$f(\mathbf{x}) = \mathbf{x} \cdot \mathbf{a}$$

$$\mathbf{x}, \mathbf{a} \in \mathbb{R}^n$$

$$\frac{\partial f}{\partial \mathbf{x}} = \mathbf{a}$$

$$f = x_1 a_1 + x_2 a_2 + \ldots + x_n a_n$$

$$\frac{\partial f}{\partial \mathbf{x}} = [\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \ldots, \frac{\partial f}{\partial x_n}]$$

Vectorized gradients: exercises

Let $f = \exp(\mathbf{w} \cdot \mathbf{x})$, what is the value of $\frac{\partial f}{\partial \mathbf{x}}$? $\mathbf{w}, \mathbf{x} \in \mathbb{R}^n$

(a) $\mathbf{w}$

(b) $\exp(\mathbf{w} \cdot \mathbf{x})$

(c) $\exp(\mathbf{w} \cdot \mathbf{x})\mathbf{w}$

(d) $\mathbf{x}$

Vectorized gradients: exercises

Let $f = \exp(\mathbf{w} \cdot \mathbf{x})$, what is the value of $\dfrac{\partial f}{\partial \mathbf{x}}$? $\mathbf{w}, \mathbf{x} \in \mathbb{R}^n$

(a) $\mathbf{w}$

(b) $\exp(\mathbf{w} \cdot \mathbf{x})$

(c) $\exp(\mathbf{w} \cdot \mathbf{x})\mathbf{w}$

(d) $\mathbf{x}$

The answer is (c).

$$\frac{\partial}{\partial x_i} = \frac{\exp(\sum_{k=1}^{n} w_i x_i)}{\partial x_i} = \exp(\sum_{k=1}^{n} w_i x_i) w_i$$

Let's compute gradients for word2vec

$$J(\theta) = -\frac{1}{T}\sum_{t=1}^{T}\sum_{-m\leq j\leq m, j\neq 0}\log\frac{\exp(\mathbf{u}_{w_t}\cdot\mathbf{v}_{w_{t+j}})}{\sum_{k\in V}\exp(\mathbf{u}_{w_t}\cdot\mathbf{v}_k)}$$

Consider one pair of center/context words (t, c):

$$y = -\log\left(\frac{\exp(\mathbf{u}_t\cdot\mathbf{v}_c)}{\sum_{k\in V}\exp(\mathbf{u}_t\cdot\mathbf{v}_k)}\right)$$

We need to compute the gradient of with respect to

$\mathbf{u}_t$ and $\mathbf{v}_k, \forall k \in V$

Let’s compute gradients for word2vec

$$y = -\log\left(\frac{\exp(\mathbf{u}_t \cdot \mathbf{v}_c)}{\sum_{k\in V}\exp(\mathbf{u}_t \cdot \mathbf{v}_k)}\right)$$

$$\begin{aligned} y &= -\log(\exp(\mathbf{u}_t \cdot \mathbf{v}_c)) + \log(\sum_{k\in V}\exp(\mathbf{u}_t \cdot \mathbf{v}_k)) \\ &= -\mathbf{u}_t \cdot \mathbf{v}_c + \log(\sum_{k\in V}\exp(\mathbf{u}_t \cdot \mathbf{v}_k)) \end{aligned}$$

Let's compute gradients for word2vec

$$y = -\log\left(\frac{\exp(\mathbf{u}_t \cdot \mathbf{v}_c)}{\sum_{k\in V}\exp(\mathbf{u}_t \cdot \mathbf{v}_k)}\right)$$

$$y = -\log(\exp(\mathbf{u}_t \cdot \mathbf{v}_c)) + \log(\sum_{k\in V}\exp(\mathbf{u}_t \cdot \mathbf{v}_k))$$
$$= -\mathbf{u}_t \cdot \mathbf{v}_c + \log(\sum_{k\in V}\exp(\mathbf{u}_t \cdot \mathbf{v}_k))$$

$$\frac{\partial y}{\partial \mathbf{u}_t} = \frac{\partial(-\mathbf{u}_t \cdot \mathbf{v}_c)}{\partial \mathbf{u}_t} + \frac{\partial(\log\sum_{k\in V}\exp(\mathbf{u}_t \cdot \mathbf{v}_k))}{\partial \mathbf{u}_t}$$
$$= -\mathbf{v}_c + \frac{\frac{\partial\sum_{k\in V}\exp(\mathbf{u}_t \cdot \mathbf{v}_k)}{\partial \mathbf{u}_t}}{\sum_{k\in V}\exp(\mathbf{u}_t \cdot \mathbf{v}_k)}$$

$$= -\mathbf{v}_c + \frac{\sum_{k\in V}\exp(\mathbf{u}_t \cdot \mathbf{v}_k)\cdot \mathbf{v}_k}{\sum_{k\in V}\exp(\mathbf{u}_t \cdot \mathbf{v}_k)}$$

Let's compute gradients for word2vec

$$y = -\log\left(\frac{\exp(\mathbf{u}_t \cdot \mathbf{v}_c)}{\sum_{k \in V} \exp(\mathbf{u}_t \cdot \mathbf{v}_k)}\right)$$

$$y = -\log(\exp(\mathbf{u}_t \cdot \mathbf{v}_c)) + \log(\sum_{k \in V} \exp(\mathbf{u}_t \cdot \mathbf{v}_k))$$
$$= -\mathbf{u}_t \cdot \mathbf{v}_c + \log(\sum_{k \in V} \exp(\mathbf{u}_t \cdot \mathbf{v}_k))$$

$$\frac{\partial y}{\partial \mathbf{u}_t} = \frac{\partial(-\mathbf{u}_t \cdot \mathbf{v}_c)}{\partial \mathbf{u}_t} + \frac{\partial(\log \sum_{k \in V} \exp(\mathbf{u}_t \cdot \mathbf{v}_k))}{\partial \mathbf{u}_t}$$
$$= -\mathbf{v}_c + \frac{\frac{\partial \sum_{k \in V} \exp(\mathbf{u}_t \cdot \mathbf{v}_k)}{\partial \mathbf{u}_t}}{\sum_{k \in V} \exp(\mathbf{u}_t \cdot \mathbf{v}_k)}$$
$$= -\mathbf{v}_c + \frac{\sum_{k \in V} \exp(\mathbf{u}_t \cdot \mathbf{v}_k) \cdot \mathbf{v}_k}{\sum_{k \in V} \exp(\mathbf{u}_t \cdot \mathbf{v}_k)}$$

Recall that

$$P(w_{t+j} \mid w_t) = \frac{\exp(\mathbf{u}_{w_t} \cdot \mathbf{v}_{w_{t+j}})}{\sum_{k \in V} \exp(\mathbf{u}_{w_t} \cdot \mathbf{v}_k)}$$

Let's compute gradients for word2vec

$$y = -\log\left(\frac{\exp(\mathbf{u}_t \cdot \mathbf{v}_c)}{\sum_{k \in V} \exp(\mathbf{u}_t \cdot \mathbf{v}_k)}\right)$$

$$y = -\log(\exp(\mathbf{u}_t \cdot \mathbf{v}_c)) + \log(\sum_{k \in V} \exp(\mathbf{u}_t \cdot \mathbf{v}_k))$$

$$= -\mathbf{u}_t \cdot \mathbf{v}_c + \log(\sum_{k \in V} \exp(\mathbf{u}_t \cdot \mathbf{v}_k))$$

Recall that

$$P(w_{t+j} \mid w_t) = \frac{\exp(\mathbf{u}_{w_t} \cdot \mathbf{v}_{w_{t+j}})}{\sum_{k \in V} \exp(\mathbf{u}_{w_t} \cdot \mathbf{v}_k)}$$

$$\frac{\partial y}{\partial \mathbf{u}_t} = \frac{\partial(-\mathbf{u}_t \cdot \mathbf{v}_c)}{\partial \mathbf{u}_t} + \frac{\partial(\log \sum_{k \in V} \exp(\mathbf{u}_t \cdot \mathbf{v}_k))}{\partial \mathbf{u}_t}$$

$$= -\mathbf{v}_c + \frac{\frac{\partial \sum_{k \in V} \exp(\mathbf{u}_t \cdot \mathbf{v}_k)}{\partial \mathbf{u}_t}}{\sum_{k \in V} \exp(\mathbf{u}_t \cdot \mathbf{v}_k)}$$

$$= -\mathbf{v}_c + \frac{\sum_{k \in V} \exp(\mathbf{u}_t \cdot \mathbf{v}_k) \cdot \mathbf{v}_k}{\sum_{k \in V} \exp(\mathbf{u}_t \cdot \mathbf{v}_k)}$$

$$= -\mathbf{v}_c + \sum_{k \in V} \frac{\exp(\mathbf{u}_t \cdot \mathbf{v}_k)}{\sum_{k' \in V} \exp(\mathbf{u}_t \cdot \mathbf{v}_{k'})} \mathbf{v}_k$$

$$= -\mathbf{v}_c + \sum_{k \in V} P(k \mid t) \mathbf{v}_k$$

Overall algorithm

- Input: text corpus, embedding size d, vocabulary V , **context size m**
- Initialize $\mathbf{u}_i, \mathbf{v}_i$ randomly $\forall i \in V$
- Run through the training corpus and for each training instance (t, c):
  - Update $\mathbf{u}_t \leftarrow \mathbf{u}_t - \eta \frac{\partial y}{\partial \mathbf{u_t}}$ $\quad \frac{\partial y}{\partial \mathbf{u}_t} = -\mathbf{v}_c + \sum_{k \in V} P(k \mid t)\mathbf{v}_k$
  - Update $\mathbf{v}_k \leftarrow \mathbf{v}_k - \eta \frac{\partial y}{\partial \mathbf{v_k}}, \forall k \in V$ $\quad \frac{\partial y}{\partial \mathbf{v}_k} = \begin{cases} (P(k \mid t) - 1)\,\mathbf{u}_t & k = c \\ P(k \mid t)\mathbf{u}_t & k \neq c \end{cases}$

Convert the training data into:
(into, problems)
(into, turning)
(into, banking)
(into, crises)
(banking, turning)
(banking, into)
(banking, crises)
(banking, as)
…

Overall algorithm

- Input: text corpus, embedding size d, vocabulary V , **context size m**
- Initialize $\mathbf{u}_i, \mathbf{v}_i$ randomly $\forall i \in V$
- Run through the training corpus and for each training instance (t, c):
  - Update $\mathbf{u}_t \leftarrow \mathbf{u}_t - \eta \frac{\partial y}{\partial \mathbf{u_t}}$ $\quad \frac{\partial y}{\partial \mathbf{u}_t} = -\mathbf{v}_c + \sum_{k \in V} P(k \mid t)\mathbf{v}_k$
  - Update $\mathbf{v}_k \leftarrow \mathbf{v}_k - \eta \frac{\partial y}{\partial \mathbf{v_k}}, \forall k \in V$ $\quad \frac{\partial y}{\partial \mathbf{v}_k} = \begin{cases} (P(k \mid t) - 1)\,\mathbf{u}_t & k = c \\ P(k \mid t)\mathbf{u}_t & k \neq c \end{cases}$

Convert the training data into:
(into, problems)
(into, turning)
(into, banking)
(into, crises)
(banking, turning)
(banking, into)
(banking, crises)
(banking, as)
…

Q: Can you think of any issues with this algorithm?