Vector Similarity

Adam Meyers
New York University

Outline

- Vector-based Document Similarity
  - Vectors represent documents, blocks of text or phrases
    - Based on words contained in those textual units
  - Use words to classify documents and blocks of text
    - Information Retrieval, Question Answering
      - Retrieving documents that are “similar” to query
    - Document Classification, Sentiment Analysis
      - Using similarity to group “similar” documents
- Lexical Semantics
  - Word Senses and WordNet
  - Embeddings = Vectors representing words
    - Use documents to classify the words contained by the document
- Logistic Regression & Deep Learning for creating embeddings

Part I: Documents & Chunks of Text

Re-occurring Themes

- Document → Vector representing its parts
- What are its parts? Words, N-grams, Chunks?
- How do we measure each part's importance?
- Vector Similarity measures similarity between
  - 2 documents
  - A question and a document;
  - Documents within a genre
  - Documents with the same “sentiment”
- Evaluation Methodology

Ad Hoc Information Retrieval (IR)

- Given:
  - a collection of documents: a set of recipes
  - a query: “Chicken soup with noodles”
- Find documents that “best” match the query.
- Assumptions:
  - Each document is a “bag of terms”
  - Each query is a “bag of terms”
- Bag of terms = unordered set of words (“chicken”) or word sequences (“ice cream”, “frying pan”)

Ad Hoc IR – More Details

- Model of document = unordered set (bag) of ***terms*** contained in that document
  - Term = word, bigram (2 consec words), trigram (3 consec words, noun group (sequence of adjectives & nouns), other units
- Query = user input, typically a set of terms
- Collection = set of documents
- Goal: find documents in collection "closest" to query
- Web search has IR component
  - IR component determines relevance of page to query
  - Other component (Google's PageRank) determines prominence on web (how many links to page)
  - Other components: question answering, etc.

Vector Representations

- Represent query and documents as vectors
  - each value represents a "score" or "weight" for a word.
- Query: "**Chicken Soup with Noodles**"

bean	chicken	lemon	noodle	stew	onion	soup	rice	coconut	sugar
0	5.4	0	2.7	0	0	5.1	0	0	0

- Recipe for chicken rice stew

bean	chicken	lemon	noodle	stew	onion	soup	rice	coconut	sugar
0	16.2	0	0	3.1	2.7	0	10.5	0	0

- **Recipe for chicken noodle soup with coconut**

bean	chicken	lemon	noodle	stew	onion	soup	rice	coconut	sugar
0	10.8	0	27	1.1	2.7	10.2	0	14.1	5.1

- Recipe for black bean soup with onion

bean	chicken	lemon	noodle	stew	onion	soup	rice	coconut	sugar
9.5	0	0	0	1.1	5.4	10.2	0	0	0

Intuitions About Previous Slide

- Query is "close" to chicken noodle soup with coconut recipe
  - Mostly the same positions with zero values
  - Smaller difference between non-zero values
  - More "important" positions tend to be similar
- How should we determine the scores?
- How should we measure similarity?

TFIDF = Common Weight for Vector

- Term Frequency – number of times term ***t*** occurs in document (alternative: number of terms divided by length of document)
- Inverse Document Frequency: Reciprocal of portion of large document set that contain term ***t***, normalized with log function:

$$\log\left(\frac{NumberOfDocuments}{NumberOfDocumentsContaining(t)}\right)$$

- TFIDF(t) = TF(t) × IDF(t)
  - Scores terms highly that occur frequently in a document or query
  - Scores terms highly that are infrequent in collection
- **TFIDF(t)** is high if **t** is more frequent in document **d** than **t** is in most documents, i.e., if **t is characteristic of the documen**t d

TF-IDF Normalization

- TF and IDF are 2 factors
- IDF is “normalized” by taking the log
  - Log normalization changes the rate of growth
  - There are other ways of normalizing, e.g., we may normalize F linearly by dividing by number of words in document.
- Other measures on a logarithmic scale:
  - Photography: Shutter speeds double (1/250, 1/125, 1/60, etc.)
  - Photography: F-stops (1.4, 2, 2.8, 4, 5.6, …) double the area
  - Music: 1 octave doubles the frequency of sound waves
  - Richter Scale: 8.0 earthquakes are 10 X 7.0 earthquakes, etc.

Example: **noodle** vs. ***tablespoon***

- ***noodle***
  - occurs ~ 3 times in chicken noodle soup with coconut recipe
    - Term frequency = 3
  - occurs in 4 out of 10,000 documents in collection
  - inverse document frequency = log(10000/4) = log(2500) = 7.82
  - TFIDF = 3 × 7.82 = 23.46
- ***tablespoon***
  - occurs 4 times in chicken and noodle soup with coconut recipe
    - Term frequency = 4
  - occurs in 1200 out of 10,000 documents in corpus
  - inverse document frequency = log(10000/1200) = log(8.33) = 2.12
  - TFIDF = 4 × 2.12 = 8.48
- **noodle** is more highly weighted for recipes than ***tablespoon***
- Note: Suitability of query term may depend on the nature of the collection
  - Is this a collection of recipes? – ***tablespoon*** not good query term
  - Is collection diverse: instructions, news, …? – ***tablespoon*** may be good query term

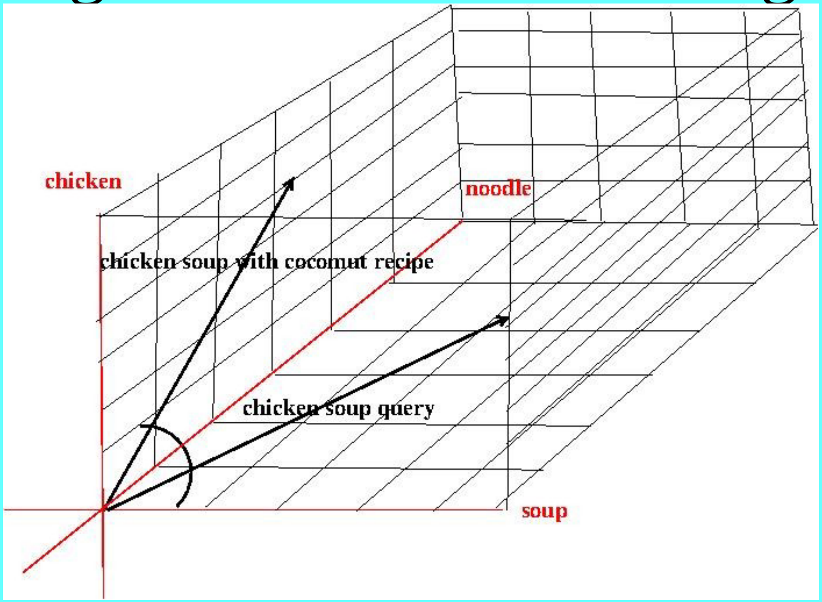
Cosine Similarity: Common Similarity Score

$$Similarity(A,B) = \frac{\sum_i a_i \times b_i}{\sqrt{\sum_i a_i^2 \times \sum_i b_i^2}}$$

- Cosine of the Angle Between the Vectors
- Numerator = Dot Product
- Denominator = Square root of product of squares
- If a query is A and a document is B
  - Cosine similarity high if values of a and b are similar
  - Maximum = 1, if a = b (i.e., numerator = denominator)

Cosine Similarity: Measures Closeness of Vectors

- Vectors are represented graphically as dimensions
  - 3 words → 3d space, 10 words → 10d space, ...
- 2 Vectors form an angle
- High Cosine if smaller angle (more similar)



Example

- Vector dimensions correspond to terms:
  - potato chip, chicken, sesame seed, coconut milk, ground beef
- 2 Queries
  - Q1 chicken, coconut milk: (0,5,0,5,0)
  - Q2 potato chip, ground beef: (4,0,0,0,7)
- 2 Documents
  - D1 Chicken and Coconut Soup Recipe: (0,7,0,9,0)
  - D2 Hamburger Recipe: (3,0,2,0,9)
- Cosine similarities

$$Similarity(Q1,D1)=\frac{(0+35+0+45+0)}{\sqrt{(5^2+0^2+5^2+0^2)\times(0^2+7^2+0^2+9^2+0^2)}}=.992$$

	Q1	Q2
D1	.992	0
D2	0	.959

Other Factors

- Many terms (possibly thousands) represented in each vector
- Other similarity measures and weight functions
- Lists of "stop words", e.g., ***the, a, in, to, does***, …
- Stemming procedures used to make equivalence classes
  - *[cat, cats]* → *cat*
  - *[analyze, analyzes, analyzed, analysis, analyse,...]* → *analyze*
- Identifying other similar words, e.g., synonyms
  - query expansion, term clustering, ...
- Systems identify word sequences as terms: N-grams or chunking
- Methods for removing dimensions, e.g., Latent Dirichlet Allocation
- Other methods for deriving vectors, e.g., Deep Learning

Jaccard Similarity
an Alternative to Cosine Similarity

- Formula for Jaccard Similarity: $Jaccard(A,B) = \frac{|A \cap B|}{|A \cup B|}$
- Example:
  - Jaccard (“green eggs and ham”, “blue eggs and chocolate”)
  - Intersection = 2 words
  - Union = 8 words
  - Score = 2/8 or .25

Evaluation Metrics: Precision, Recall, F-measure

- System Output = answers from a system
- Answer Key = correct answers from humans
- Correct = length (System Output ∩ Answer Key)
- Precision = Correct ÷ length(System Output)
- Recall = Correct ÷ length(Answer Key)
- F-measure = $\dfrac{2}{\dfrac{1}{precision}+\dfrac{1}{recall}}$
- Example:
  - System: 1, 2, 4, 5, 7
  - Answer Key 1, 2, 3, 4
  - Correct = 1, 2, 4
  - Precision = 3/5 = .6
  - Recall = ¾ = .75
  - F-measure = 2 ÷ (1.33 + 1.67) = .67

Tasks can Favor Precision or Recall

- Voice Recognition of Helicoptor Commands
  - Precision is much more important than Recall
  - An error is a disaster, better to produce no output and let pilot keep the control
- Finding Lost Children
  - Recall is much more important than Precision
  - Missing a child is a disaster, better to question some children who are not lost

Interannotator Agreement

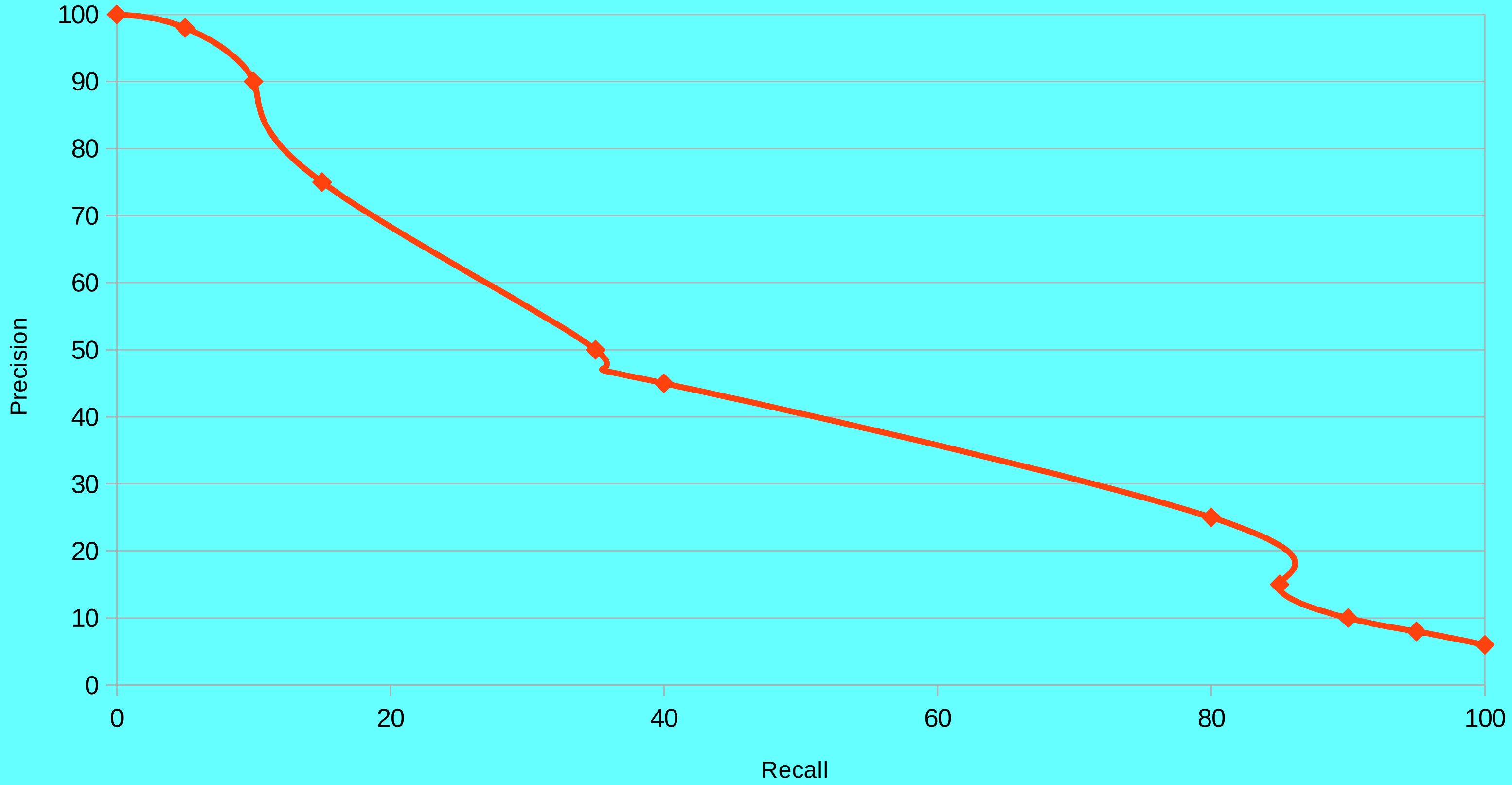
- Example: the word ***to*** has 2 uses:
  - A **preposition** – ***I walked to the store***
  - An **auxilliary** verb-like element – ***I want to fly***
- 2 Annotators tagged 100 cases for these 2 categories
- Annotator1 tagged 70 prepositions and 30 auxilliaries
- Annotator 2 tagged 60 prepositions and 40 auxilliaries
- They agreed on 50 out of the 100 cases.
- Use these numbers to calculate Kappa

$$\boldsymbol{Kappa} = \frac{\mathbf{Observed\_Agreement - Chance\_Agreement}}{\mathbf{1 - Chance\_Agreement}}$$

Example

- Observed Agreement = 50/100 = 50%
- Chance Agreement = (.7 * .6) + (.3 * .4) = 54%
- Kappa = (.5-.54)/.46 = -.09 (not so good)
- A better result would be 80% agreement
  – Kappa = (.8-.54)/.46 = .57
- And so on

Precision/Recall Curve: Parameters may trade precision for recall



Evaluating Top N Documents in Ranking

- Output of IR = A Ranked List of Documents
  - Ranking makes relevant/irrelevant distinction subtle
  - Error in high-ranked documents “worse” than error in low-ranked
- Precision/Recall tradeoff curves based on number of docs considered
  - Precision goes down and recall goes up for more documents
  - Measure using area under curve: higher area → better system
- Too Expensive to Create Gold Standard Manually
  - Collections can be millions or billions of documents
  - Precision can be approximated by taking samples of the text or evaluating the top N ranked terms manually.
  - Recall can also be approximated by some sort of sampling, e.g., only manually evaluating a subset of the collection

Mean **A**verage **P**recision is used to Score IR output

- Problem: How do we score a ranked list?
- MAP = compute precision at several intervals and average.
  - If high ranked items in list tend to be better, the score is higher
- Example
  - Answer Key: 20 correct answers
  - Different precision assuming progressively higher recall
    - 2/first 5 correct (2/5 precision at 10% recall)
    - 2/next 10 correct (4/15 precision at 20% recall)
    - 2/next 7 correct (6/22 precision at 30% recall)
    - …
    - 20 correct/top 200 (20/200 precision at 100% recall)
  - MAP = average of these 10 intervals
  - **.195 ≈ (2/5 + 4/15 + 6/22 + 8/40 + 10/55 + 12/85 + 14/105 + 16/120 + 18/150 + 20/200)/10**

Homework 4

- http://cs.nyu.edu/courses/fall23/CSCI-UA.0480-057/homework4.html
- Information Retrieval Task
- Due date: Evening of Class 6 (Graduate) or 11 (Under Graduate)

Question Answering Task can be Similar to IR

- Given a query and set of documents
- Find the paragraph (not a document) that is closest to the query.
- This assumes the answer to the question will be found in the paragraph
- Essentially same problem as IR, but looking for most similar paragraph, instead of most similar document

(Supervised) Document Classification

- Given:
  - N sets (categories, genres, topics, etc) of seed documents
  - A set of unclassified documents
- Goal
  - Automatically assign new documents to “closest” category

Is Doc 11 a Recipe or a Sports Document?

- Recipes

Doc	bean	chick	lime	salt	ball	stick	helmet	score
1	4.3	5.1	0	1.2	0	0	0	1
2	7.8	10.2	2.4	0.6	2.5	0	0	0
3	0	5.1	4.5	1.5	0	4.3	0	0
4	4.3	5.1	2.4	1.2	0	0	0	0
5	9.6	0	4	.6	0	4.3	0	0
Ave	5.2	5.1	2.7	2.1	0.5	1.7	0	.2

- Sports

Doc	bean	chick	lime	salt	ball	stick	helmet	score
6	0	1.1	0	1.2	9.2	0	7.8	5.8
7	1.5	1.1	0	0	10.2	5.1	4.3	6.1
8	0	0	0	0	5.4	4.3	5.7	8.1
9	0	0	1	1.1	0	5.4	4.5	5.1
10	0	0	0	0	7.2	9.0	4,5	0
Ave	0.3	0.44	0.2	0.26	6.4	4.8	5.4	6.8

- Doc 11 (Unclassified)

Doc	bean	chick	lime	salt	ball	stick	helmet	score
11	0	0	2.0	0	5.1	0	10.1	4.3

Document Classification

- Each document → vector of TF-IDFs of list of terms
- Each category → average of category doc vectors
  - Each dimension d = average of scores for d
  - Average is an example – alternative combo functions (like max)
- Starting points
  - Supervised: Some documents classified already
  - Unsupervised: Each document starts out as its own category
- Steps:
  - Supervised: Category of New doc → “similar” category
  - Unsupervised: Iteratively merge similar categories
- Similarity scores may be ranked by best to worst fit
  - Allow each doc to be part of multiple categories?
  - Unsupervised needs a stopping point
    - e.g., minimum number of categories or cut-off similarity score

Examples of Document Classification

- Supervised classification of science papers by grade
  - CS /Education Major now getting her Masters at UPenn
  - Paper at 2017 School Science and Mathematics Association Conference
- Supervised classification of supreme court opinions by topic
  - Visiting CS student from Case applying for grad schools
  - Presenting CS paper at Fed CSIS 2018
- Other Examples:
  - Classify documents (tweets, news, etc.) by political persuasion (undergrad NLP students did this)
  - Classify newspaper articles by genre: sports, business, politics, etc.
  - Classify medical articles by subject matter: cancer, genetics, etc. (colleague did this)
  - Cluster home insurance claims by sources of damage (flood, fire, etc.)

Evaluating Document Clustering

- Set aside some test documents
  - already assigned classes by human beings
  - not seed documents
- Accuracy = correct ÷ number of documents
  - Precision and Recall are only used if the system can produce a different number of results
- Example: 50 correct out of 100 = 50% accuracy
- It is customary to set aside 2 sets of testing documents
  - dev set: use to design your system
  - test set: use to report final results

Unsupervised Document Clustering

- Goal: Partition set of unclassified documents
- Given:
  - Set of unclassified documents and their vectors
  - Each document is its own cluster
- Repeat until some "stopping point"
    - Merge the 2 most similar clusters
      - The vector for the new cluster is the average of the document vectors for the cluster
- Stopping points:
  - There are only N remaining clusters (e.g., N = 10)
  - The closest clusters have a similarity of less than X (e.g., X = .5)

More on Unsupervised Document Clustering

- Clusters are not guaranteed to be natural classes of documents (possible criterion for scoring results)
- Topic Modeling = A popular clustering technique
  - Methods to reduce the dimensions of the vectors
    - e.g., Latent Dirichlet allocation
  - Dimensions may not refer to particular words, but vectors represent topics (which are sets of words)
  - Topic is hierarchical and documents can be members of more than one cluster
  - Words representing documents are used to provide user with an idea of what each topic is "about"

Examples

- Automatically Group all final papers into 5 categories
  - Professor can grade similar papers together and have basis of comparison
- Divide 1000 news articles automatically into 10 topics
  - Based on sets of words associated with each topic, try to manually assign categories

Sentiment Analysis

- Document Classification
  - Classes based on opinions or sentiments
- Examples:
  - Positive vs Negative vs Neutral views about
    - Companies or their stock prices
    - Political Candidates
    - Products
  - Star Ratings or Scales from 1 to 5
    - Reviews of movies, products, etc.
  - Happy vs Sad vs Angry vs … – Really difficult
- Sentiment Task Available: predict stars from reviews:
  - https://www.kaggle.com/c/sentiment-analysis-on-movie-reviews
- **Warning: Students sometimes choose sentiment analysis for a new domain without really thinking through the evaluation process. Pre-defined tasks makes the evaluation much easier and allows you to focus on the system.**

Sentiment: Special Considerations

- Negation: not, no, never, doesn't, ….
  - Bag of words model problematic
  - “I don't like Senator Smith”
    - Negative sentiment in spite of “like”
- Same terms linked to positive/negative in different contexts
  - low, high, small, large, thin, thick, visible, loud, soft, …
    - *high/low quality, high/low interest, high/low resolution*

Final Projects on Document Classification

- Always cite related work
- Separate training, development, test documents
- Evaluation plan should be clear
- New vs Old task
  - If you use a pre-created task, you can concentrate on the methodology
  - If you make your own task, most of your work will be setting up your task – only a small part will be making your system to do the task

Part II: Lexical Semantics

Outline

- Basics of Lexical Semantics
- Word Senses and WordNet
- Word Similarity using Word Embeddings
- Logistic Regression and Deep Learning

Lemmas and Wordforms

- Lemma: basic word form (paired with POS) used in lexicon
  - base form representing a set of inflected forms
    - Singular nouns: ***book*** → ***book, books***
    - Bare infinitive verbs: ***be*** → ***be, being, been, am, is, are, were, was***
    - Base adjective: ***angry*** → ***angry, angrier, angriest***
- Word form: word how it actually occurs
  - a single wordform can be related to multiple lemmas:
    - ***bases*** is the plural of ***basis*** and ***base***
    - ***leaves*** is the plural of ***leaf*** and ***leave*** (as in ***leaves of absense***)
      - also present-tense 3rd Pers Sing form (VBZ) of the verb ***leave***
  - A word form can be defined phonologically (different homophones)
    - **/tu/** has 3 lemmas corresponding to ***two***, ***to*** and ***too***
  - A word form can be defined orthographically (different homographs)
    - Thus ***does*** corresponds to 2 lemmas with different pronunciations: (1) present 3rd person singular of the verb ***do*** and (2) plural of the noun ***doe***

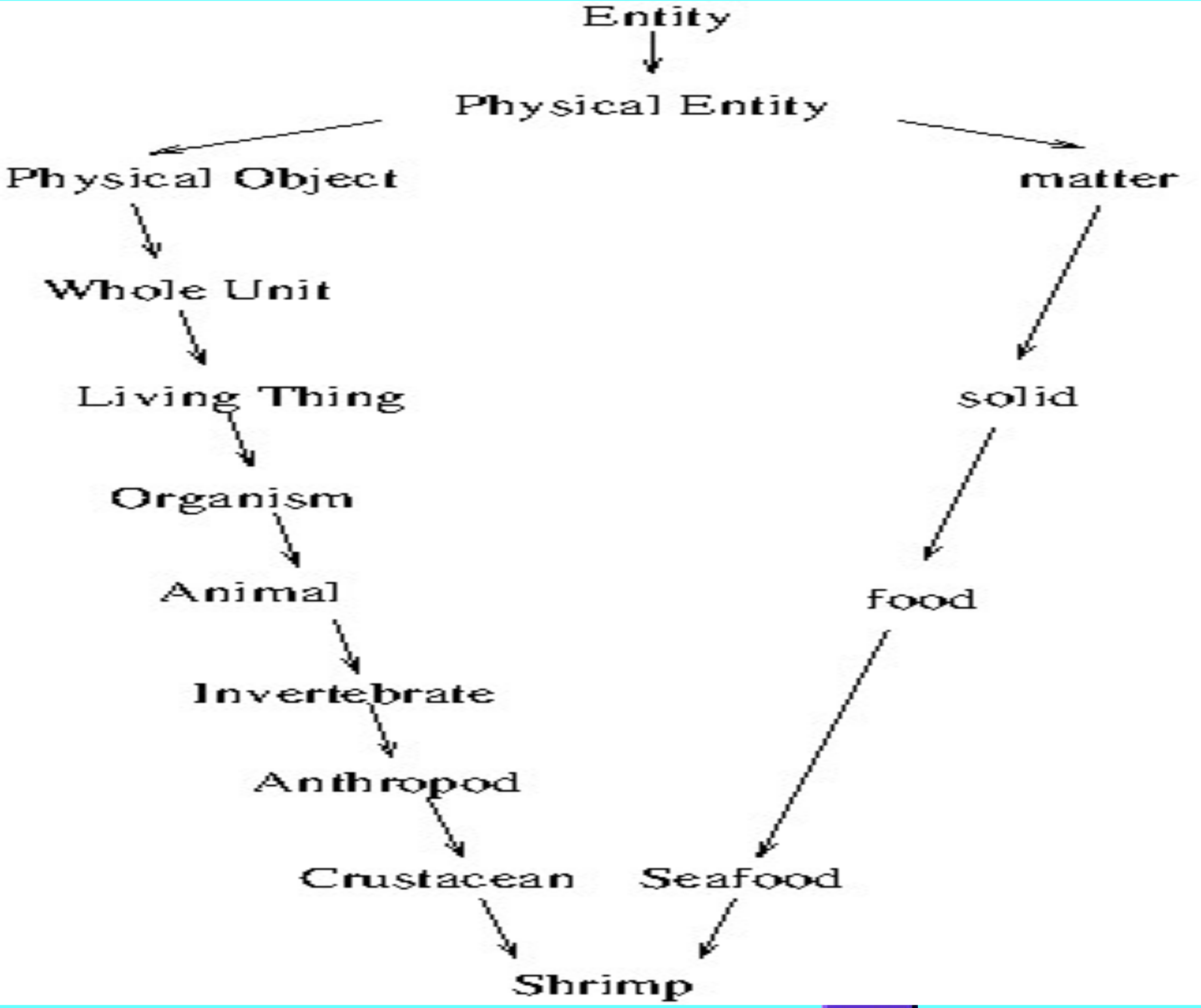
Senses

- Conventional Dictionaries and Thesauri map lemmas to sets of different meanings called senses
- Granularity of senses: a standard problem in lexicography
  - **merge** 2 senses together or **split** one sense into 2?
- Lets lookup ***bank*** in WordNet (Version 3), a thesaurus/dictionary that we will be featuring
  - Compare definitions 1 and 2
  - Compare definitions 2 and 9
    - Is it possible that these definitions should be merged together?
    - All organization names can stand in for buildings that house them
      - Thus 9 is predictable from 2 (by metonymy)
    - A single instance can have both “senses” at once:
      - ***The bank on the corner hired 3 security guards***

WordNet Senses

- Conventional Dictionaries
  - Senses are defined informally as text
  - The relations between senses is not represented
- WordNet
  - a sense is defined by a *Synset*, the set of words that share that meaning. Intuition = define properties by extensions
  - Formal semantic definitions are often *extensional*
    - Both Frege and Russel's set-theoretic definitions of natural numbers
      - a number ***N*** is defined as the set of sets with cardinality ***N***
    - Many other definitions of meaning used in (computational) linguistics
  - Senses are hierarchical (connected by graph, edges represent IS-A)
    - Senses have hyponyms (sub-senses) and hypernyms (super-senses)
      - ***furniture*** $\supset$ ***seat*** $\supset$ ***chair*** $\supset$ ***recliner***
    - The full graph allows multiple inheritance
      - and even cycles such as : ***restrain*** $\supset$ ***inhibit*** $\supset$ ***restrain*** $\supset$ ***inhibit ...***

The Super Classes of *shrimp*



Applications of WordNet

- Sense disambiguation (multi-sense words only):
  - Given some text tagged with WN senses
  - Train a classifier that can automatically tag raw data
  - Issue: fine-grained senses are difficult to tag (manually or automatically)
    - Most approaches collapse several WordNet senses together
- Calculating Semantic Similarity
  - Some models of similarity of meaning are based on distance in the sense hierarchy
  - Issue: same length paths do not reflect same similarity
    - distinctions are based on available information, not calculated similarities
    - Alternatives: cruder relative path distances; path distances combined with other similarity scores (e.g., vector-based cosine similarity)

Distributional Methods for Sense Disambiguation and Word Similarity

- ML with features based on N-grams (sequences of neighboring tokens), dependencies (relations with words in parse trees), etc.
- Word Embeddings
  - Represent each word by a large vector representing the words that occur in any of the same sentences in some corpus.
  - Similarity calculated using similarity between vectors
    - e.g., cosine similarity
    - similarity between other words measures word similarity
    - similarity between instance and automatically or manually annotated synset instance can indicate sense
  - Subject of most of the remaining slides

Word Embeddings using TF-IDF

- Information Retrieval – Classifies documents (**columns**) based on the TF-IDFs of words in those documents
  –

	Doc 1	Doc 2	Doc 3	Doc 4
Chicken	23.46	0	7.82	0
Sugar	2.4	4.8	0	9.6
Noodle	12.4	0	4.1	0
Table Spoon	.5	1	.5	1.5

  – Doc 1 & Doc 3 are similar; Doc 2 and Doc 4 are similar
- Word Embeddings can classify words based on the documents they occur in (**rows, rather than the columns**)
  – "Chicken" & "Noodle" are similar – but what does that mean?

What does word similarity mean?

- (Approximate) Synonymy
  - ***coat* ≈ *jacket*** **–** For fashion experts, coats are longer than jackets
- Sample types of related meanings
  - Hypernym: ***chair* ⊒ *furniture***
  - Hyponym: ***furniture* ⊑ *chair***
  - Related by common Hypernym: ***chair, table***
    - both hyponyms of furniture
  - Same semantic field: ***check, stalemate, pawn***
    - all chess terms
- Embeddings are NOT designed to differentiate among types of similarity
  - Example:
    - ***oval*** and ***ellipse*** are synonyms
    - But the embeddings for ***oval*** & ***rectangle*** may be more similar than ***oval*** & ***ellipse*** because ***oval*** occurs in fewer math texts
- Famous example: King – Male + Female = Queen (by vector arithmetic)
  - Attributed to "Efficient Estimation of Word Representations in Vector Space" (Mikolov et al., 2013)

Context for Word Embeddings

- Information Retrieval Example:
  - Context = Document
- Usually Smaller Context for Word Similarity
  - Same Document
  - Same Paragraph
  - Same Sentence
  - Within a window of +/-N words
    - Example if N = 2
    - $\boldsymbol{Word_{n-2}}$ $\boldsymbol{Word_{n-1}}$ $\boldsymbol{Word_{n}}$ $\boldsymbol{Word_{n+1}}$ $\boldsymbol{Word_{n+2}}$

Word Word Matrix Using Pointwise Mutual Information

- Word Word Matrix (aka ***word embedding***)
  - Rows represent word_R
  - Columns (aka ***dimensions***) represent words co-occurring with word_C
  - Can be generalized to multi-words (n-grams, phrases, …)
    - multi-word to multi-word
  - Context can be defined other ways, e.g., proximity in syntactic tree
- Approximate meaning: “A word is defined by the company it keeps” (Firth)
- Scores in Matrix
  - How related is word_R to word_C represented by column C
  - Pointwise Mutual Information:

$$PMI = \log\left(\frac{prob(word_C \mid word_R)}{prob(word_R) \times prob(word_C)}\right)$$

  - Positive PMI often used: **PPMI = max(PMI(w,r),0)** – negaitve PMI is unreliable
  - LaPlace smoothing – add constant (e.g., 1) to all counts to adjust for low frequencies

Sample Word Embedding 1

- Assume a “bag of words” approach
  - Order of words don't matter
  - Other embeddings (e.g., skip grams) model word order as well
  - Assume that words are stemmed
- Use words in a window of K words before and K words after $word_R$
- Let's assume K = 5 (for this example)
- Eliminate stop words and high frequency (low IDF) words
- Use numeric scores in vectors
  - We are using integers for example
  - Actual scores usually between 0 and 1

Sample Word Embedding 2
From Hypothetical Recipe Corpus

- Rows = words being classified
- Columns = words in context
- Higher number indicates more likely to be in window of +/- 5 from word labeling row

	cup	ounce	taste	chicken	stir	bake	chocolate
beef	1	4	1	0	4	5	0
cabbage	3	0	0	0	0	5	0
lemon	3	3	4	2	2	0	1
parsley	2	1	4	2	1	2	0
pepper	0	4	4	3	0	5	0
salt	1	3	4	4	0	5	1
sugar	5	1	4	0	1	2	5

Cosine similarity for Word Vectors from Previous Slide

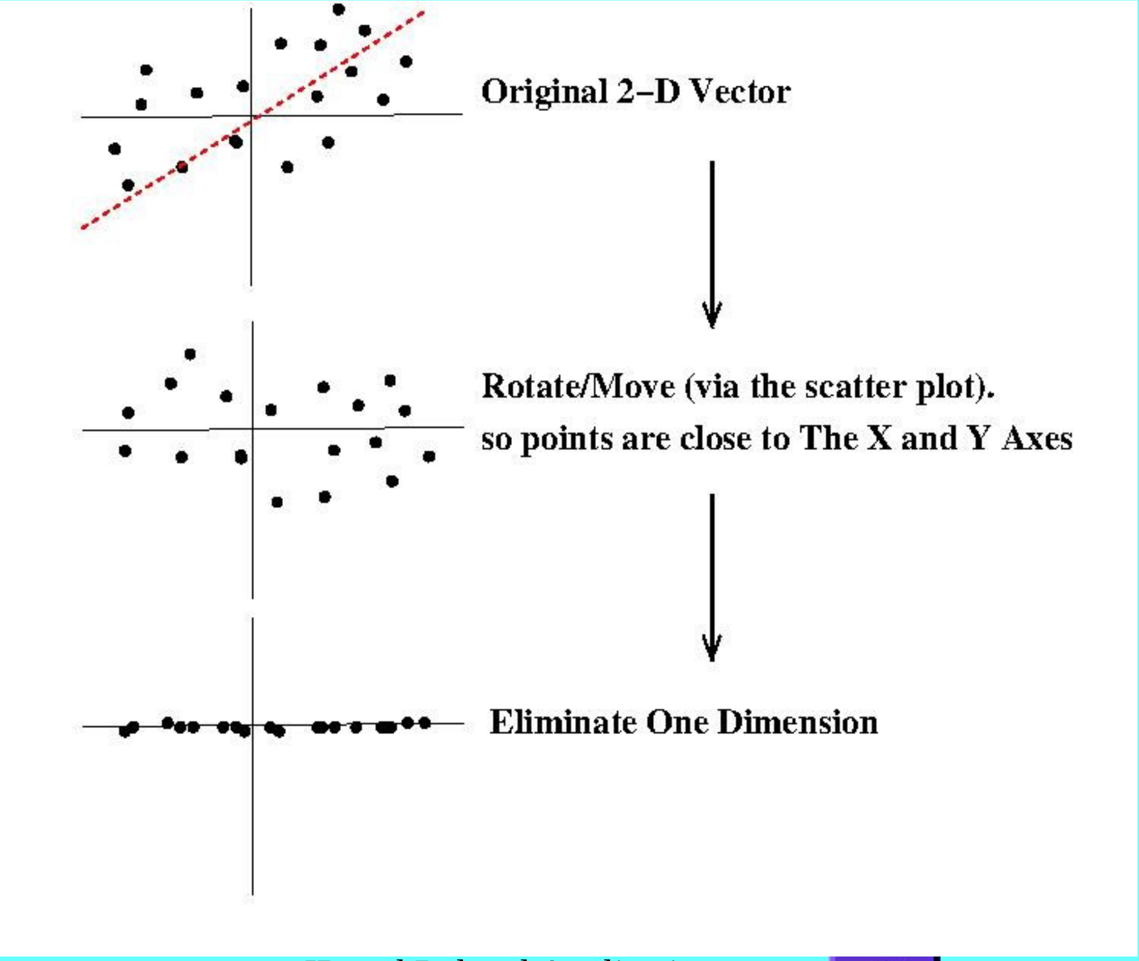
	beef	cabbage	lemon	parsley	pepper	salt	sugar
beef	1	.63	.54	.57	.72	.66	.41
cabbage	.63	1	.25	.51	.53	.58	.51
lemon	.54	.25	1	.86	.64	.68	.74
parsley	.57	.51	.86	1	.81	.86	.69
pepper	.72	.53	.64	.81	1	.97	.44
salt	.66	.58	.68	.86	.97	1	.56
sugar	.41	.51	.74	.69	.44	.56	1

“The Curse of Dimensionality”

- To account for all word word co-occurances, we may need 100s of thousands of dimensions
- Millions if we include n-grams
- These vectors are “sparse”, i.e., lots of zeros
- As a result, some problems will not scale well
- Solutions involve smaller, denser vectors

Latent Semantic Analysis: Reducing Dimensions

- Repeatedly remove dimensions as follows:



Logistic Regression

- Start with a function from input to output with randomly assigned weights
- Apply function to random subsets of input for which the output is known
- If the function makes the correct prediction, increase the weights
- If the function makes the incorrect prediction, decrease the weights
- Repeat this process until some stopping point.

Word Embeddings Using Logistic Regression

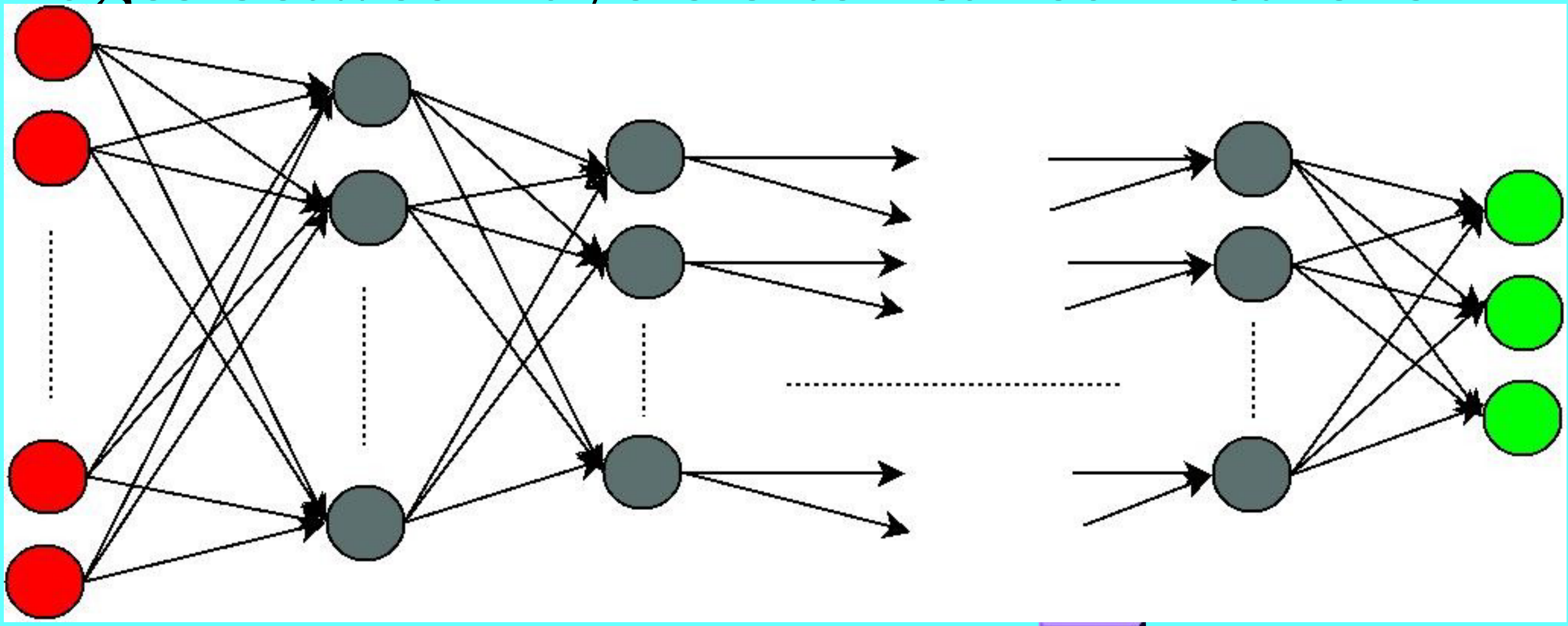
- Randomly fill numbers into a vector with some fixed number of dimensions, e.g., 100
- Start with an "objective" function that predicts the occurance of words. The function uses the vector's values as weights.
- Repeat until some stopping point:
  - Randomly select sample from large input text
  - For each $word_R$ in sample, determine if context predicts that the word is likely (context = words in ±k window)
    - Increase or decrease the dimension values accordingly
  - Combine individual probabilities of words in context (e.g., k =2):
    - $P(Word_n \mid Word_{n-2})$, $P(Word_n \mid Word_{n-1})$, $P(Word_n \mid Word_{n+1})$, $P(Word_n \mid Word_{n+2})$

"Deep Learning"

- Logistic Regression Methods for manipulating vectors
- Vectors are weights for some objective function
- Methods described in this talk have a single "hidden layer"
  - The word embeddings that are learned
- Other Deep Learning Methods may have multiple hidden layers
- Deep Learning, aka, Neural Networks, are inspired by models of the human brain by psychologists
- Successful models unlikely to be models of human thinking
  - Cynical view: allusion to human brain is just marketing

Deep Learning Network

- Red = Input, Green = Output, Grey = “hidden” layers
- When acquiring word embeddings, **there is typically only one hidden layer**, containing numbers between 0 and 1. This hidden layer is the word embedding itself.
- Edges between layers often called “neurons”



Word2Vec SkipGram with Negative Sampling

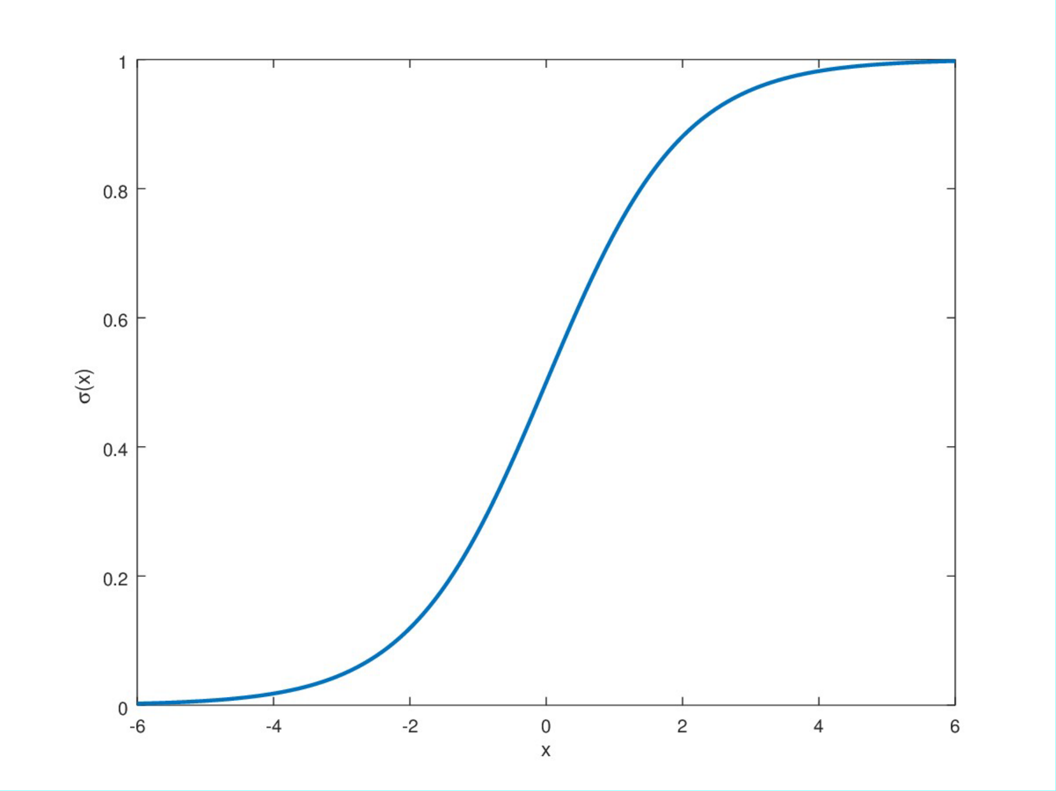
- For each word t in a sample
  - P(+|t,c) = probability that context word c occurs near t (within some window, e.g., 2 word before/after)
  - P(-|t,c) = probability that c is not near t = 1 – P(+t,c)
- Assumption:
  - probability is based on idea that embeddings of nearby words should be "similar"
  - Given the vectors (embeddings) we are assuming:
    - Probability predictable from distribution of similarity scores between embeddings of context words (words close to target word)
    - Sigmoid functions (below) model a set of similarities among embeddings as probability scores (between 0 and 1) [Sigmoid is an example of a "softmax" function"]
    - In equations below: t & c are embeddings for target and context words

$$P(+|t,c)=\sigma(+|t,c)=\frac{1}{1+e^{-\mathrm{Sim}(t,c)}}$$

$$P(-|t,c)=\sigma(-|t,c)=1-\frac{1}{1-e^{-\mathrm{Sim}(t,c)}}=\frac{e^{-\mathrm{Sim}(t,c)}}{1+e^{-\mathrm{Sim}(t,c)}}$$

Sigmoid Function: between 0 & 1

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



Word2Vec: Probability of multiple context words

- Product of prob of k context words, $w_1, \ldots, w_k$:

$$P(+|t,c^{1:k}) = \prod_{1}^{k} \frac{1}{1+e^{-\mathrm{Sim}(t,c^i)}}$$

  – For window of ±2, k = 4
  – P (Probability) modelled as **σ** (sigmoid function)
  – Multiplication implies that probabilities of context words are independent of each other

- Logarithm version – Addition instead of Multiplication:

$$\log P(+|t,c^{1:k}) = \sum_{1}^{k} \frac{1}{1+e^{-\mathrm{Sim}(t,c^i)}}$$

- Negative Probability of noise words

$$\log P(-|(t,n^{1:k})) = \sum_{i=1}^{k} \log \frac{1}{1+e^{\mathrm{Sim}(n^i,t)}}$$

Train Word2Vec embeddings from text samples

- Objective Function: function to maximize/minimize, e.g., by adjusting parameters (layers or embeddings)
- Goal: Identify an “Objective” function that given a target word t in context c, will:
  - Maximize P(+|t,c): similarity to context (nearby words)
  - Minimize P(-|t,n) similarity to noise (random words)
- Objective function for each word/context pair (t,c) with k noise words $n_{1...k}$
- $$OBJ(\theta) = \sum_{(t,c)\in +} \log \frac{1}{1+e^{-\mathrm{Sim}(t,c^i)}} + \sum_{i=1}^{k} \log \frac{1}{1+e^{\mathrm{Sim}(n^i,t)}}$$
- 1^{st} term = combo of similarity of t to context
- 2^{nd} term = combo of negative similarity to noise

Training Word2Vec: Apply Gradient Descent

- Each word has 2 embeddings: word embedding and target embedding (combo of targets)
- All embeddings initially have random values, but are tuned by logistic regression
  - For each word **w** in each sample:
    - Calculate (cosine) similarity between word embedding & context embedding
    - Calculate similarity between word embedding & noise embedding
    - Modify all embeddings so similarity increases for context and decreases for noise
- Keep sampling text and recalculating word vectors until stopping point
  - e.g., when similarity score between word & context is already very high and/or the similarity between word & noise is already very low

Logistic Regresson Embeddings Frameworks

- Popular Frameworks:
  - Word2Vec, FastText, GLOVE, BERT, ELMo
- Pretrained Embeddings Available
  - Enormous amount of training data
  - Often work better than small individual efforts
- Tune to special domains, e.g., Twitter, Legal, Games, Recipes
  - More effective to combine with large pre-trained embeddings than to simply retrain
  - Example:
    - Train on in-domain data, but …
    - initialize as pre-trained embeddings instead of random

Application: Find similar words

- Demo: http://demo.patrickpantel.com/demos/lexsem/thesaurus.htm

Application: Word Sense Disambiguation

- Demo of A word sense disambiguator demo
  - http://ltbev.informatik.uni-hamburg.de/wsd/single-word
    - (May not be working)
- Shared tasks include Semcor
  - http://web.eecs.umich.edu/~mihalcea/downloads.html#semcor
- Using Word Vectors for Word Sense Disambiguation
  - Vectors represent word senses rather than words
    - Need sense annotated corpus
  - Create vectors for words in new text
  - Compute similarity of words in new text with sense vectors and choose most similar sense

Paraphrase and Entailment

- SemEval Text Similarity Task:
  - 2016 (Task 1)
  - http://alt.qcri.org/semeval2014/task1/ (webpage)
  - https://aclweb.org/anthology/S/S16/S16-1081.pdf (write-up)
- Input pairs of text “snippets”
  - English/English (like previous year tasks)
  - Spanish/English pairs (innovation for 2016)
    - previous snippets, with one member of pair translated
- System produces score from 0 to 5 indicating similarity
- Manually tagged data (test, dev, training sets)
- Data collection of snippets based on heuristics and manually annotate
  - One heuristic is based on word embedding similarity – embedding of sentence = sum of the embeddings of words

Summary

- WordNet: human created ontology and sense annotation
- Vector characterization of words (word embeddings)
  – Dimensions represent words in context within a window
  – Related words/word-senses/translations/etc. have similar embeddings
- Dimensions can be derived
  – using transparent metrics like TF-IDF or PMI
  – Using logistic regression based methods that, among other advantages, require fewer dimensions
- Dimensions can be reduced by methods like LDA
- Similarity is calculated with Cosine Similarity, Jaccard similarity, …
- "Deep Learning" methods
  – Use logistic regression with several "hidden layers"
  – Are less transparent, but typically higher scoring than previously discussed Machine Learning Methods
-

Readings on WordNet and Word Similarity

- J & M: Chapter 19.1, 19.2 and 19.3
- WordNet
  - Read the first 2 papers found here:
    - http://wordnetcode.princeton.edu/5papers.pdf
  - Read NLTK section 2.5 and try the NLTK WordNet module

Word Embeddings References

- Jurafsky and Martin 3rd Edition: https://web.stanford.edu/~jurafsky/slp3/
  - Chapter 6 – Good Overview
- Pretrained Embeddings: https://www.aclweb.org/anthology/L18-1008.pdf
- Word2Vec
  - https://www.tensorflow.org/versions/r0.12/tutorials/word2vec/index.html
  - https://deeplearning4j.org/word2vec
  - https://github.com/dav/word2vec
- BERT
  - https://arxiv.org/pdf/1810.04805.pdf
- Domain Adapation using pre-trained embeddings:
  - https://www.aclweb.org/anthology/D19-1433.pdf

Deep Learning Documentation (Reading Optional) and Code

- Intro to Deep Learning that I found helpful:
  - Taylor M. (2017). Neural Networks Math: A Visual Introduction for Beginners. Blue Mills Media

Deep Learning at NYU

- Machine Translation
  - Prof. Kyunghyun Cho (http://www.kyunghyuncho.me/)
- Natural Language Semantics
  - Prof. Sam Bowman (https://www.nyu.edu/projects/bowman/)
- My research group:
  - Undavia etl al. (2018) – Document classification
  - Ortega et. al. (2018) – Machine Translation
  - Nguyen et. a. (2016) – ACE event extaction
- And Others

Embeddings and N-grams

- During a future lecture, I will be discussing various ways of representing word sequences including:
  - Phrase structure rules, e.g.,
    - [S [NP They]
      [VP [VBD laughed]
      [PP [IN about]
      [NP [NN stuff]]]]]
  - N-Grams – probability based analysis in terms of frequencies that words occur next to each other
- Then we will talk about some ML approaches that combine phrase structure and probability
- At the end of that unit, we will re-introduce embeddings as a possible replacement for n-grams, in connection with the noun predicate argument structure task.