Word Representation

Palacode Narayana Iyer Anantharaman

narayana.anantharaman@gmail.com

6th Oct 2018

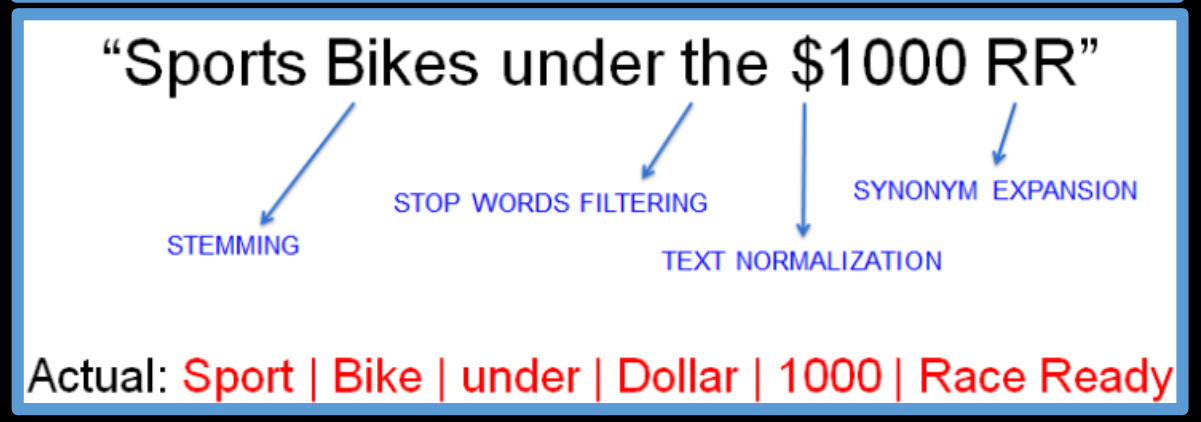
Words and Sentences

- Text documents are made up of discourses constituted by paragraphs, sentences, phrases and words
- The words provide us the meaning
- Each class of documents have some common words and also some distinguishing words.
  - Likewise terms like: comedy, hero, climax etc occur often in film reviews.
- Word level analysis can help us predict the document the words came from.
- Notion of: term frequency (tf), inverse document frequency (idf) and tf-idf

Content Category	
/Adult	/Hobbies & Leisure
/Arts & Entertainment	/Hobbies & Leisure/Clubs & Organizations
/Arts & Entertainment/Celebrities & Entertainment News	/Hobbies & Leisure/Clubs & Organizations/Youth Organizations & Resources
/Arts & Entertainment/Comics & Animation	/Hobbies & Leisure/Crafts
/Arts & Entertainment/Comics & Animation/Anime & Manga	/Hobbies & Leisure/Crafts/Fiber & Textile Arts
/Arts & Entertainment/Comics & Animation/Cartoons	/Hobbies & Leisure/Merit Prizes & Contests
/Arts & Entertainment/Comics & Animation/Comics	/Hobbies & Leisure/Outdoors
/Arts & Entertainment/Entertainment Industry	/Hobbies & Leisure/Outdoors/Fishing
/Arts & Entertainment/Entertainment Industry/Film & TV Industry	/Hobbies & Leisure/Outdoors/Hiking & Camping
/Arts & Entertainment/Entertainment Industry/Recording Industry	/Hobbies & Leisure/Paintball
/Arts & Entertainment/Events & Listings	/Hobbies & Leisure/Radio Control & Modeling
/Arts & Entertainment/Events & Listings/Bars, Clubs & Nightlife	/Hobbies & Leisure/Radio Control & Modeling/Model Trains & Railroads

Some basic concepts to get started

- Concepts
  - Word and Sentence segmentation
  - Pre-processing the text and Normalization: stop words removal, stemming, lemmatization
  - Term Frequency (tf)
  - Inverse document Frequency (idf)
  - Tfidf
  - Bag of words (BOW)
  - Vector Space models
  - Cosine Similarity
- Language Models



Text Normalization: Basic Steps

- Case conversion
- Tokenization

Vocabulary – ref: Dan Jurafsky and Christopher Manning

N = number of tokens

V = vocabulary = set of types

$|V|$ is the size of the vocabulary

Church and Gale (1990): $|V| > O(N^{½})$

	Tokens = N	**Types = |V|**
Switchboard phone conversations	2.4 million	20 thousand
Shakespeare	884,000	31 thousand
Google N-grams	1 trillion	13 million

Issues in tokenization - Dan Jurafsky and Christopher Manning

- `Finland's capital` → `Finland Finlands Finland's ?`
- `what're, I'm, isn't` → `What are, I am, is not`
- `Hewlett-Packard` → `Hewlett Packard ?`
- `state-of-the-art` → `state of the art ?`
- `Lowercase` → `lower-case lowercase lower case ?`
- `San Francisco` → one token or two?
- m.p.h., PhD. → ??

- **Social Media texts form a major chunk of public data**
- **Text normalization is challenging when used with social media data**

A motivating example: Pre-processing and Text Normalization challenges from tweet data

- Consider the following tweets:
  - GOD's been very kind as HE didn't create FRIENDS with price tags.If He did, I really couldn't afford U **#HappyFriendshipDay** @KimdeLeon
  - Why are wishing friendship day today? Aren't we sppsd to be friends for all the 365 days?**#HappyFriendshipDay**
  - eVrYthin ChnGZ & nthin StaYs D SMe bt aS v grOw uP 1 thnG vl reMain i ws wT u b4& vL b tiLl D end **#HappyFriendshipDay** pic.twitter.com/Ym3ZAnFiFn

- How do we tokenize, normalize the above text?
  - Often the answer to the above is application dependent

Words

- How do we identify word boundaries and tokenize the words?
    - Don’t, would’ve, b4 (if we have a rule/logic where we consider valid English words to consist only of alphabets [a-zA-Z], then the word boundary would break at the letter 4)
- Normalizing lowercase, uppercase
    - Some news headlines may be capitalized: “PM Modi Announces US $1 Billion Concessional Line of Credit to Nepal”
    - Some tweets may be in all capital letters
- Normalization to same word expressed with differing syntax
    - This is needed for applications like information retrieval so that the query string can be more appropriately matched with the content being searched - E.g, U.K and UK may be resolved to one term, similarly Mr and Mr., Ph.D, PhD etc
- How do we handle words that are distorted versions of a valid English word?
    - eVrYthin, ChnGZ, D, etc – if we have a vocabulary of a given corpus that has the terms everything, changes, the would we count the abbreviated terms same as the valid ones or would we consider them different?
    - “I am tooooooooo happppppppppppy” – If we correct these in to: “I am too happy” we may lose the implied and hidden emotion of the strength of happiness expressed in the phrase.
- How do we handle new words not part of the language?
    - Tweeple, selfie, bestie etc – consider a spell checker application that we would like to build. Would we correct these? Are these candidate correction words of some one’s misspells, eg, Selfy instead of selfie?

Sentence Segmentation

- Sentence segmentation is concerned about splitting the given input text in to sentences.
- Characters like !, ?, . may indicate sentence boundaries
- However, there could be ambiguity, for instance:
  - The quarterly results of Yahoo! showed a promising trend
  - Microsoft announced updates to .NET framework
  - Indian rupee appreciated by 0.5% against USD during today's trade
  - Who moved my cheese? is a great read.
- The ambiguity may also arise due to spelling mistakes
- Possible ways to resolve:
  - Traditional rule based regular expressions
  - Decision trees encoding some rules
  - Classifiers

Challenges with social data

1	i	17,071,657
2	-i	4,254
3	ijust	1,446
4	dontcha	872
5	ireally	705
6	d'you	527
7	whaddya	511
8	istill	477
9	//i	438
10	ijus	424
11	i-i	393
12	inever	362
13	#uever	347

1	haven't	170,431
2	havent	38,303
3	shoulda	14,114
4	would've	13,043
5	should've	12,997
6	hadn't	11,899
7	woulda	10,061
8	could've	7,435
9	coulda	6,009
10	havnt	5,227
11	shouldve	3,972
12	wouldve	3,752
13	must've	3,164
14	musta	2,159
15	couldve	2,142
16	haven't	

1	love	1,908,093
2	luv	58,486
3	lovee	9,927
4	envy	9,668
5	salute	5,629
6	loveee	5,107
7	lov	3,625
8	dread	3,188
9	looove	3,109
10	cba	2,975
11	loveeee	2,932
12	loooove	2,740
13	loove	1,664
14	looooove	1,584
15	loveeeee	1,435
16	lurve	868
17	loooooove	849
18	l0ve	712
19	loveeeeee	667
20	luff	626
21	l.o.v.e	585
22	lovelovelove	583
23	luvv	558

Some key challenges are:

- How do we tokenize words like: ireally, l0ve, #unever etc.
- How to handle tagging problems like: POS tagging, NE tagging and so on
- More broadly: How to address the traditional NLP core tasks and applicati

Tokenization - Tools

Tweet NLP

ARK Carnegie Mellon

We provide a tokenizer, a part-of-speech tagger, hierarchical word clusters, and a dependency parser for tweets, along with annotated corpora and web-based annotation tools.

Contributors: Archna Bhatia, Dipanjan Das, Chris Dyer, Jacob Eisenstein, Jeffrey Flanigan, Kevin Gimpel, Michael Heilman, Lingpeng Kong, Daniel Mills, Brendan O'Connor, Olutobi Owoputi, Nathan Schneider, Noah Smith, Swabha Swayamdipta and Dani Yogatama.

Quick Links

- Part-of-speech Tagger and POS annotated data - also Twokenizer: tokenizer software (part of tagger package) and Tagging Models -- Download Link
- Tweeboparser and Tweebank: Dependency parser software and dependency annotated data -- Download Link
- Documentation, annotation guidelines, and papers describing this work
- Hierarchical Twitter Word Clusters

```
>>> zen = TextBlob("Beautiful is better than ugly. "
...                "Explicit is better than implicit. "
...                "Simple is better than complex.")
>>> zen.words
WordList(['Beautiful', 'is', 'better', 'than', 'ugly', 'Explicit', 'is', 'better',
>>> zen.sentences
[Sentence("Beautiful is better than ugly."), Sentence("Explicit is better than imp
```

NLTK 3.0 documentation

PREVIOUS | MODULES | INDEX

nltk.tokenize package

Stanford University NLP Natural Language Processing

The Stanford Natural Language Processing Group

home · people · teaching · research · publications · software · events · local

Stanford Tokenizer

About | Obtaining | Usage | Questions | Mailing Lists

About

A tokenizer divides text into a sequence of tokens, which roughly correspond to "words". We provide a class suitable for tokenization of English, called PTBTokenizer. It was initially designed to largely mimic Penn Treebank 3 (PTB) tokenization hence its name, though over time the tokenizer has added quite a few options and a fair amount of Unicode compatibility, so in general it will work well over text encoded in the Unicode Basic Multilingual Plane that does not require word segmentation (such as writing systems that do not put spaces between words) or more exotic language-particular rules (such as writing systems that use : or ? as a character inside words, etc.). An ancillary tool uses this tokenization to provide the ability to split text into sentences. PTBTokenizer mainly targets formal English writing rather than SMS-speak.

PTBTokenizer is a an efficient, fast, deterministic tokenizer. (For the more technically inclined, it is implemented as a finite automaton, produced by JFlex.) On a 2015 laptop computer, it will tokenize text at a rate of about 1,000,000 tokens per

Term Frequency, Inverse Document Frequency

- Term frequency denotes the count of occurrences of a given term in the given document:
  - Term frequency of a term t in a document d: tf(t, d)
  - Term Frequency signifies the importance of the given term t in a given document d

- Suppose we have N documents and suppose we are counting the occurrence of a term t across all the N documents. The Inverse Document Frequency (idf) is the log of ratio of number of documents to those that contain the term t.
  - IDF of a term t across N documents: **idf(t, D) = log(N / D)** where D = set of all documents containing t
  - IDF indicates whether the given term t is a common term found in a large number of documents in the corpus or it is a rare term found only in a small subset of documents
  - Thus, IDF is a measure of how much information the given term provides. A common term provides very less information while terms that are unique to a given document help distinguish the documents.

tfidf

Tfidf is the product of tf and idf

- tfidf(t, d, D) = tf(t, d) * idf(t, D)
- Tfidf assigns a weight to term t in a document d, where d belongs to a corpus of N documents
- Terms with relatively high tfidf help discriminate one document from the other
- Given a query term and document d, we can compute a score: $\text{Score}(q, d) = \sum_{t \in q} \text{tf-idf}_{t,d}$

Algorithm: Computing document ranks with respect to a query

Input:

- Query String
- Corpus of N documents

Output:

- Rank list : [(di, si), (dj, sj), …] # di is the top ranked document, dj has rank 2 etc

1. Word tokenize the query to a set of words: q1, q2…, qk
2. Set ranks to an empty list # ranks = [ ] : output will be produced here
3. Compute for each document di in the document corpus D:
    1. Initialize score of di = 0
    2. Compute for each query term t in q:
        1. n = Tf(t, di) # compute term frequency
        2. m = idf(t, D) # compute IDF for t across documents: **NOTE: If needed add 1 smoothing**
        3. tfidf = n * m
        4. Add tfidf to score of di
    3. Add the tuple (index of di, score for di) to ranks # (di, si)
4. Sort the ranks list
5. Return ranks

Rank ordering the 2 documents

- Consider the two documents shown in the next slide
- Let the query string be: q = “Python Web Server”
- The above query is made up of 3 words: (Python, Web, Server)
- We are required to rank the documents in terms of match for this query
  - Compute_ranks(q, documents) : returns the rank of the documents

Illustration of tfidf computation

HOWTO Use Python in the web

Author: Marek Kubica

Document 1: From Python official web site

Abstract

This document shows how Python fits into the web. It presents some ways to integrate Python with a web server, and general practices useful for developing web sites.

Programming for the Web has become a hot topic since the rise of "Web 2.0", which focuses on user-generated content on web sites. It has always been possible to use Python for creating web sites, but it was a rather tedious task. Therefore, many frameworks and helper tools have been created to assist developers in creating faster and more robust sites. This HOWTO describes some of the methods used to combine Python with a web server to create dynamic content. It is not meant as a complete introduction, as this topic is far too broad to be covered in one single document. However, a short overview of the most popular libraries is provided.

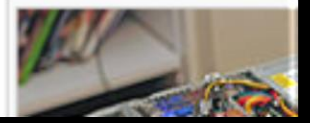
Web server

From Wikipedia, the free encyclopedia

Document 2: From Wikipedia

This article **needs additional citations for verification**. Please help improve this article by adding citations to reliable sources. Unsourced material may be challenged and removed. *(March 2009)*

The term **web server**, also written as **Web server**, can refer to either the hardware (the computer) or the software (the computer application) that helps to deliver web content that can be accessed through the Internet.[1]



Tfidf computation and rank ordering

Query term	Tf(t, d1)	Tf(t, d2)	Tfidf(t, D)	Tfidf(t, D)
Python	4	0	4 * log 2 = 1.2	0
Web	9	4	0	0
Server	2	3	0	0

Note:
- As this is just an illustration, the tf values above are just counts and not normalized. Since the document sizes can vary, we may need to normalize in some applications
- This illustration uses log with base 10
- We have not used smoothing approach. If a query term is not found in a document, it is excluded from computing tfidf score.
- Tfidf = 0 for the terms web and server because both documents contain these terms and hence idf = log 1, which is zero. Hence tf * idf = 0

As per our algorithm, score(q, d1) = 1.2 and score(q, d2) = 0

Hence the output is: ranks = [(d1, 1.2), (d2, 0)]

Vector Space Representation of Sentences

Text Normalization

Text Normalization – Wiki Definition

Text normalization

Text normalization is the process of transforming **text** into a single canonical form that it might not have had before. **Normalizing text** before storing or processing it allows for separation of concerns, since input is guaranteed to be consistent before operations are performed on it.

Text normalization - Wikipedia, the free encyclopedia
en.wikipedia.org/wiki/**Text_normalization** ▾

Text Normalization

- The goal of text normalization is to transform the text in to a form that makes it suitable for further processing.
- The kind of processing that we might do on the normalized text depends on the goals of the application.
- Hence it is imperative that there may not be one single best way to normalize the text.
- Natural language text generated in social media adds to the challenges of NLP. Social media texts may contain:
  - Misspelt words
    - Peter Norvig reports an accuracy of 98.9% for his spell correction program that computes edit distance of 2 on a standard text corpus (http://norvig.com/spell-correct.html). But some of the tweets in social media may have words that can not be corrected with an edit distance of 2. eg, “vL” corresponds to “we will” in some tweets!
  - New words and terminologies
  - URLs, emoticons, hashtags, @names etc as part of the text
  - Named entities that contain non alphabet symbols as a part of the name: eg, Yahoo!
  - Sentence and word boundaries not well marked
    - For example, tokenizing words for “vL b” yields 2 word tokens while “we will be” yields 3 tokens

Text tokenization and normalization: Motivating example

Consider 2 problems:

- **Suppose we need to pick the tweets from the list below that have distinct information (as against having same information expressed in different forms), which ones we would pick?**
- **Suppose we need to draw a tag cloud of terms from the tweets, which words are to be shown?**
- **Key questions:**
  - **What is the contribution of twitter ids, hash tags, URLs etc?**
  - **Which words contribute most? Least?**
  - **What kind of text processing may bring best results?**

Sample Tweets (time stamp: 8 Aug 2014, 5:20 pm IST):

1. RT @IndianCricNews: Pujara was dismissed for a duck for the first time in his Test career. 1st duck in FC cricket since November 2008 http:…
2. 4th Test Match' England Vs India' Eng 160-5 over 48 Root 9* M Ali 10* Lead by 8 runs' *GEO PAKISTAN*
3. RT @CricketAus: If you're playing at home, #EngvInd day two is starting. Can India fight back? Follow LIVE http://t.co/4FFMDOcJBV http://t.…
4. Virat Kohli going through worst patch of his career - Zee News http://t.co/MHUll6zi6a
5. India vs England Live Score: 4th Test, Day 2 - IBNLive - IBNLiveIndia vs England Live Score: 4th Test, Day 2IBNLiv... http://t.co/ZIUTD5Y94j
6. England v India: Fourth Test, Old Trafford, day twoLive - BBC Sport http://t.co/OHOEiRj9QX
7. RT @cricbuzz: .@ECB_cricket move into the lead now! Follow here: http://t.co/GdlOLvNgmb #EngvInd

Text normalization

- Text normalization is required for most NLP applications
- Consider the problem of generating the index of a corpus of text, where the index contains the word and the locations in the text where the word is found.
- This involves:
  - Splitting the text in to sentences
  - Word tokenization that yields (word, location of the word)
- Word types and tokens
  - Similar to the token types and token values that we encounter during the lexical analysis of programming languages

Lemmatisation and Stemming (ref: Wikipedia definition)

- **Lemmatisation** is the process of grouping together the different inflected forms of a word so they can be analysed as a single item
  - Lemmatization is sensitive to POS: E.g: Meeting (noun), meeting (verb)

- **Stemming**: In linguistic morphology and information retrieval, **stemming** is the process for reducing inflected (or sometimes derived) words to their stem, base or root form—generally a written word form.

Porter's algorithm for stemming is widely used

```
>>> p.stem('happier')
'happier'
>>> p.stem('happiest')
'happiest'
>>> p.stem('better')
'better'
>>> p.stem('going')
'go'
>>> p.stem('walking')
'walk'
>>> p.stem('loci')
'loci'
>>> p.stem('foci')
'foci'
>>> print(lemmatizer.lemmatize('running', pos['noun']))
running
>>> print(lemmatizer.lemmatize('running', pos['verb']))
Run
>>> print(lemmatizer.lemmatize('holding', pos['verb']))
hold
```

Stemming and Lemmatization Differences

- Both lemmatization and stemming attempt to bring a canonical form for a set of related word forms.
- Lemmatization takes the part of speech in to consideration. For example, the term ‘meeting’ may either be returned as ‘meeting’ or as ‘meet’ depending on the part of speech.
- Lemmatization often uses a tagged vocabulary (such as Wordnet) and can perform more sophisticated normalization. E.g. transforming mice to mouse or foci to focus.
- Stemming implementations, such as the Porter’s stemmer, use heuristics that truncates or transforms the end letters of the words with the goal of producing a normalized form. Since this is algorithm based, there is no requirement of a vocabulary.
- Some stemming implementations may combine a vocabulary along with the algorithm. Such an approach for example convert ‘cars’ to ‘automobile’ or even ‘Honda City’, ‘Mercedes Benz’ to a common word ‘automobile’
- A stem produced by typical stemmers may not be a word that is part of a language vocabulary but lemmatizer transform the given word forms to a valid lemma.

Bag of words and Vector Space Model

9.1 Broad to Rahane, no run, touch short, outside off and Rahane fiddles at it, stuck in the crease

9.2 Broad to Rahane, no run, snorting bouncer from Broad, flying through at throat height, Rahane limbos for his life underneath it

- Bag of words is a representation of a document as an unordered set of words with the respective frequency
- This approach of modelling a document by the list of words it contains, without regard to the order, is adequate for several applications (such as those that use Naïve Bayes Classifier or query matching) but is a severe limitation for certain other applications.
- Let us model the above 2 statements in the example as a bag of words

  Vocabulary size: |V| = |V1 U V2|

  We have 29 unique words – hence 29 dimensional vector space

a = (1,1,2,1,1,1,1,1,1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0)

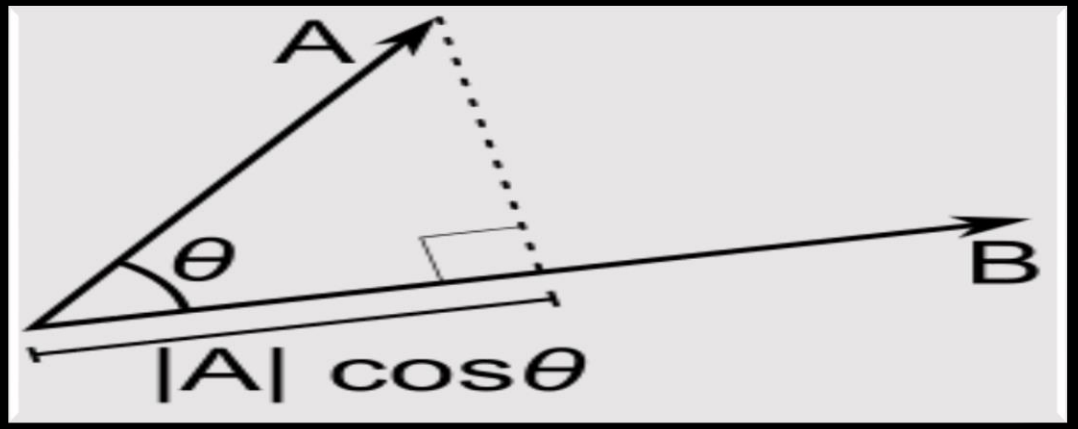
b = (2,1,2,1,1,0,0,0,0,0,0,1,0,0,0,0,0,1,1,1,1,1,1,1,1,1,1,1)

Word	Count 1	Count 2
Broad	1	2
to	1	1
Rahane	2	2
no	1	1
run	1	1
touch	1	0
Short	1	0
Outside	1	0
Off	1	0
And	1	0
fiddles	1	0
At	1	1
It	1	0
Stuck	1	0
In	1	0
The	1	0
crease	1	0

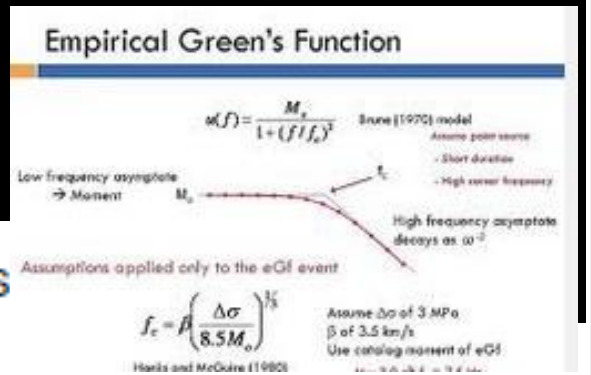
Word	Count 1	Count 2
snorting	0	1
bouncer	0	1
from	0	1
flying	0	1
through	0	1
throat	0	1
height	0	1
limbos	0	1
for	0	1
his	0	1
life	0	1
underneath	0	1

Cosine similarity

- We can map each document in a corpus to a n-dimensional vector, where n is the size of the vocabulary.
  - Here, we represent each unique word as a dimension and the magnitude along this dimension is the count of that word in the document.
- Given such vectors a, b, …, we can compute the vector dot product and cosine of the angle between them.
- The angle is a measure of alignment between 2 vectors and hence similarity.
- An example of its use in information retrieval is to: Vectorize both the query string and the documents and find similarity(q, di) for all i from 1 to n.



What does the word “Green” mean to you?



Green nod for Noida projects, relief for buyers

Green

From Wikipedia, the free encyclopedia

This article is about the color. For other uses, see Green (disambiguation).

Green is the color between blue and yellow on the spectrum of visible light. It is evoked by light with a predominant wavelength of roughly 495–570 nm. In the subtractive color system, used in painting and color printing, it is created by a combination of yellow and blue, or yellow and cyan; in the RGB color model, used on television and computer screens, it is one of the additive primary colors, along with red and blue, which are mixed in different combinations to create all other colors.

1 Etymology and linguistic definitions
- 1.1 Languages where green and blue are one c
- 1.2 Shades and varieties

2 In science
- 2.1 Color vision and colorimetry
- 2.2 Lasers
- 2.3 Minerals and chemistry
- 2.4 Biology
- 2.5 Green eyes

3 In history and art
- 3.1 In the ancient world
- 3.2 In the Middle Ages and Renaissance
- 3.3 The 18th and 19th century
- 3.4 20th and 21st Century

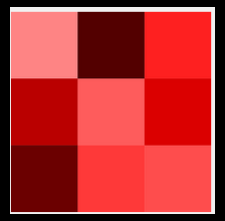
4 Symbolism and associations
- 4.1 Nature, vivacity, and life
- 4.2 Springtime, freshness, and hope
- 4.3 Youth and inexperience
- 4.4 Calm, tolerance, and the agreeable
- 4.5 Jealousy and envy
- 4.6 Love and sexuality
- 4.7 Fairies, dragons, monsters, and devils
- 4.8 Poison, sickness, and misfortune
- 4.9 Safety and permission
- 4.10 Social status, prosperity and the dollar

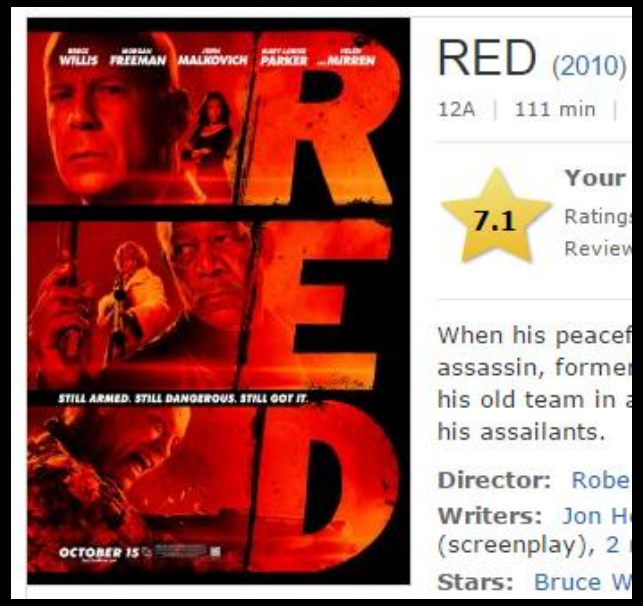
5 On flags

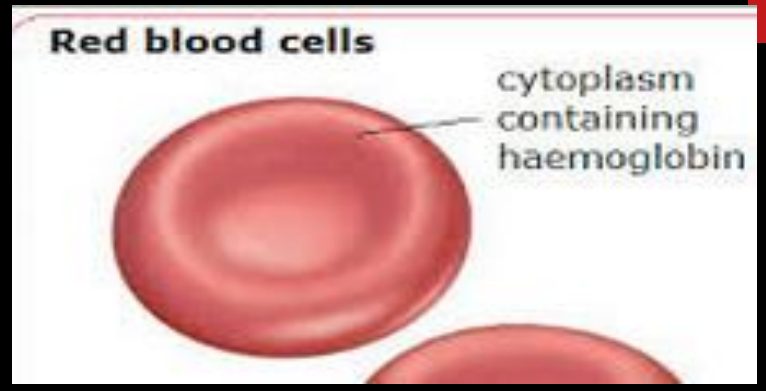
6 In politics

7 In religion

How about “Red”?







Observation#1

- A given word (red or green in our example) can assume different meanings and interpretations based on the context
  - In a medical condition the term “critical” refers to a patient’s health condition and hence connotes to an undesirable situation
  - In the context of a nuclear fission reactor the term critical means that the reactor is able to sustain the chain reaction and hence has the capability to generate power – an event that the nuclear scientists would celebrate!

- How to represent a word that can lend itself to be interpreted differently?

Associations of the words

(Green, regularization), (red, Tendulkar), (red, green)

About 16,10,000 results (0.77 seconds)

Images for red tendulkar — Report imag

More images for red tendulkar

Crab in Red Sauce, Sachin Tendulkar's favourite - TripAdvisor
www.tripadvisor.in › ... › Betalbatim › Betalbatim Restaurants
Martin's Corner, Betalbatim Picture: Crab in **Red** Sauce, Sachin **Tendulkar's** favourite
Check out TripAdvisor members' 1442 candid photos and videos.

When a red-faced Tendulkar shouted at hotel manager ...
www.thatscricket.com › News
Nov 10, 2014 - When a **red**-faced Sachin **Tendulkar** shouted at hotel manager in Sydney during India's tour to Australia in 2008.

Crab in Red Sauce, Sachin Tendulkar's favourite - TripAdvisor
www.tripadvisor.com › Asia › India › Goa › Colva
Colva, Goa Picture: Crab in **Red** Sauce, Sachin **Tendulkar's** favourite - Check out TripAdvisor members' 2677 candid photos and videos of Colva.

About 9,18,000 results (0.48 seconds)

Regularization (physics) - Wikipedia, the free encyclopedia
https://en.wikipedia.org/wiki/**Regularization**_(physics)
In physics, especially quantum field theory, **regularization** is a method of dealing with ...
Regularization method results in **regularized** n-point **Green's** functions ...

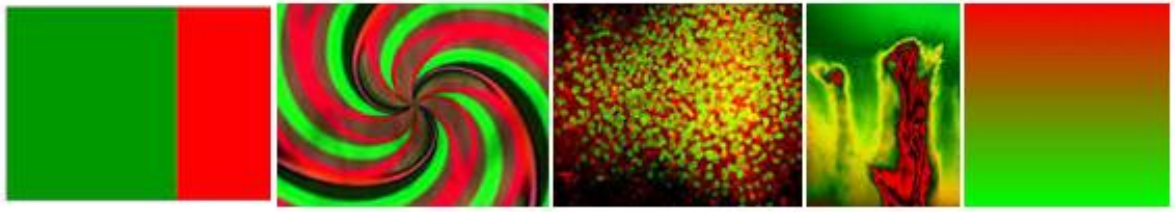
Operator regular
link.aps.org/doi/10.1
by DGC McKeon - 198
Operator **regularizatic**
perturbative expansion

[PDF] A Learning F
users.cis.fiu.edu/~tac
by C Ding - 2007 - Cite
Aug 15, 2007 - **Regula**
ABSTRACT. **Green's**

[PDF] The connecti
citeseerx.ist.psu.edu
by AJ Smola - 1998 - C
In this paper a corresp
that the **Green's** Funct

About 6,34,00,00,000 results (0.63 seconds)

Images for green red — Report images



More images for green red

what does green plus red make? | Yahoo Answers
https://answers.yahoo.com/question/index?qid...
Sep 12, 2008 - Best Answer: **green** and **red** are complimentary colors, like yellow and purple, or blue and orange. any time you combine two complimentary ...

Color Mixing - Enchanted Learning Software
www.enchantedlearning.com/crafts/Colormixing.shtml
The six tertiary colors (**red**-orange, **red**-violet, yellow-**green**, yellow-orange, blue-**green** and blue-violet) are made by mixing a primary color with an adjacent ...
Label Color Wheel Print-out - Label Color Complements ... - Colors

RGB color model - Wikipedia, the free encyclopedia
https://en.wikipedia.org/wiki/RGB_color_model

Observation#2

- A given word may be related to another word in more than 1 dimension. In a given dimension the words may be quite similar and in another dimension they may be totally dissimilar.
  - Red and Green are both basic colors and similar in that perspective
  - If you are waiting at a traffic signal, red implies stop and green let's you go. Here the 2 words are dissimilar.

- The associations can be learnt from a corpus. But for us to learn all possible ways in which the words can be associated we need a huge corpus. In general, larger the corpus, better will be the word representation and its ability to model semantics

What should a good Word Representation system satisfy?

As obvious from the examples discussed, we can state a couple of criteria that a good word representation system should satisfy:

1. The representation should capture the different ways the word is semantically interpreted
   - For instance: “Red letter day” gives positive semantic to the word red, “The progress is slow due to red tape” associates red with a negative sentiment

2. From the representation it should be possible to infer similarities in the semantics between any words belonging to a vocabulary
   - For instance: Red, Blue, Green should have smaller mutual distance compared to the terms that are totally unrelated to them.

Atomic Vs Distributed Representation

Foundation

- Represent the word by its symbol
  - LMs count the occurrence of words as n-grams. The word is represented by its symbol.
- Represent the word by its index in a table of vocabulary
  - Can be used form one hot vector of words
  - Can be used to represent sentences
- See the illustration of how the word “red” and its semantics are represented. You observe 7 synsets and their corresponding definitions. This representation is quite inflexible as:
  - This doesn’t capture a large number of other semantics of red (e.g. Red Tape)
  - No easy way to find similarities to other words (red, Tendulkar)
  - We can find hypernyms and deduce that red and green are similar – but the dimensions of comparisons and resolving metaphoric references are limited
  - Wordnet is a static database – vocabulary is limited, learning new words is not automatic
  - Intense human effort from domain experts to build such semantic databases

Discrete/Atomic representations

```
>>> from textblob import Word
>>> word = Word("red")
>>> for syn in word.synsets:
        print syn


Synset('red.n.01')
Synset('red.n.02')
Synset('bolshevik.n.01')
Synset('loss.n.06')
Synset('red.s.01')
Synset('crimson.s.02')
Synset('crimson.s.03')
>>> for d in word.definitions:
        print d


red color or pigment; the chromatic color resembling the hue of blood
a tributary of the Mississippi River that flows eastward from Texas along the southern boundary of Oklahoma and through L
ouisiana
emotionally charged terms used to refer to extreme radicals or revolutionaries
the amount by which the cost of a business exceeds its revenue
of a color at the end of the color spectrum (next to orange); resembling the color of blood or cherries or tomatoes or ru
bies
characterized by violence or bloodshed
(especially of the face) reddened or suffused with or as if with blood from emotion or exertion
```

Problems with discrete representation

- Wordnet or other such databases mayn’t exist for all languages
- Such databases may not be updated frequently and hence new words might get missed
- It may not capture domain specific semantics
- Atomic symbol
- Semantic similarity and associations not captured in the representation

One hot vector representation

- Let V be the vocabulary and |V| be its size
- Suppose if a given sentence has words: $w_1 w_2 \ldots w_n$
- We represent any word w_i by a one hot vector where the element that corresponds to w_i is set to 1 and all other elements of the vector are set to zero
  - X_i = [0, 0, 0, ….., 1, 0, 0, …., 0]
- The sentence is represented by concatenating all these vectors in the respective order
  - If the sentence has 10 words and each word is represented by a vector of dimension |V|, the total length of input is 10 |V|

Problems with 1-h vectors

- The 1-h vectors are sparse
- Very high dimensionality, as |V| is typically very large
- They don’t capture semantic similarities
  - E.g: Home and Residence are similar and so should be close to each other. But in English dictionary organized in alphabetical order, they are far away.

Ref: R Socher CS224d, lecture 2

Distributional similarity based representations

You can get a lot of value by representing a word by means of its neighbors

"You shall know a word by the company it keeps"

(J. R. Firth 1957: 11)

One of the most successful ideas of modern statistical NLP

government debt problems turning into **banking** crises as has happened in

saying that Europe needs unified **banking** regulation to replace the hodgepodge

↖ These words will represent *banking* ↗

Window based cooccurence matrix

- Example corpus:
  - I like deep learning.
  - I like NLP.
  - I enjoy flying.

counts	I	like	enjoy	deep	learning	NLP	flying	.
I	0	2	1	0	0	0	0	0
like	2	0	0	1	0	1	0	0
enjoy	1	0	0	0	0	0	1	0
deep	0	1	0	0	1	0	0	0
learning	0	0	0	1	0	0	0	1
NLP	0	1	0	0	0	0	0	1
flying	0	0	1	0	0	0	0	1
.	0	0	0	0	1	1	1	0

8

Issues with cooccurence matrix

- The matrix size increases substantially with size. With 2 words, this is square of |V|
- Very high dimensional (needs huge storage)
- Sparsity issues
- Less robust models

Singular Value Decomposition (SVD)

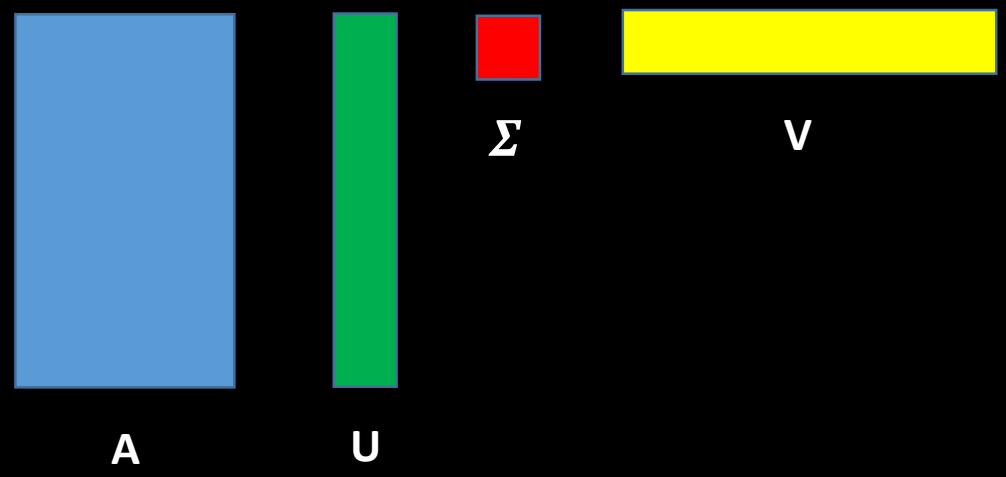
- SVD allows us to factorize a matrix as a product of 3 matrices

$$A_{m\times n} = U_{m\times r}\,\Sigma_{r\times r}\,(V_{n\times r})^T$$

- Terminology:
  - A is the input matrix
  - U has the left singular vectors
  - Σ has the singular values
  - V has the right singular vectors

SVD: Theorem

- It is always possible to decompose a matrix $A_{m\times n}$ as a product of $U_{m\times r}, \Sigma_{r\times r}, V_{n\times r}$
- U, Σ, V are unique for a given A
- U, V are made up of orthonormal column vectors
- $U^T U = I \; and \; V^T V = I$



Application Example for SVD

- Suppose there are m people and n movies and each person is rating a movie between 0 to 5.
- Rating 0 is for “not watched”, 1 is the lowest rating and 5 is the highest rating.
- We can form a matrix of people and movies and each matrix element shows a rating
- Our goal is to determine the “concepts” that are implicit in this matrix

	M1	M2	M3	M4	M5	M6
P1						
P2						
P3						
P4						
P5						
P6						
P7						

Application Example (Contd)

- The concepts behind the movies could be the genres of the movie such as: sci-fi movies, suspense thrillers, comedy, etc
- It could also be based on who acted in the film. E.g. if a group of users have watched movies like Arunachalam, Basha, Chandramukhi, Endiran and Kabali, it is easy for us to see that this group is a Rajinikanth fan club ☺
- The key point is that the number of concepts are typically much less than the number of items that map to a concept.
- By mapping individual items (e.g movies) to the concepts (e.g Rajini movies, * Khan movies, etc) we can perform dimensionality reduction

Application Example (Contd)

The SVD of the people versus movie matrix provides an excellent mechanism to determine the concepts as below

- The U matrix captures the person to concept association
- The Σ matrix captures the strength of each concept
- V represents the movie to concept similarity matrix

Latent Semantic Analysis

- We can apply the SVD as a tool to determine concepts in a text document.
- We can form a matrix of occurrence of terms versus documents in a corpus.
- Our goal is to map the terms to concepts and also documents to concepts
- The matrix elements could be bag of words counts or it could also be tf-idf value.
- Once we form such a matrix, we can obtain U, V and determine respective associations to the concepts

Word Representation

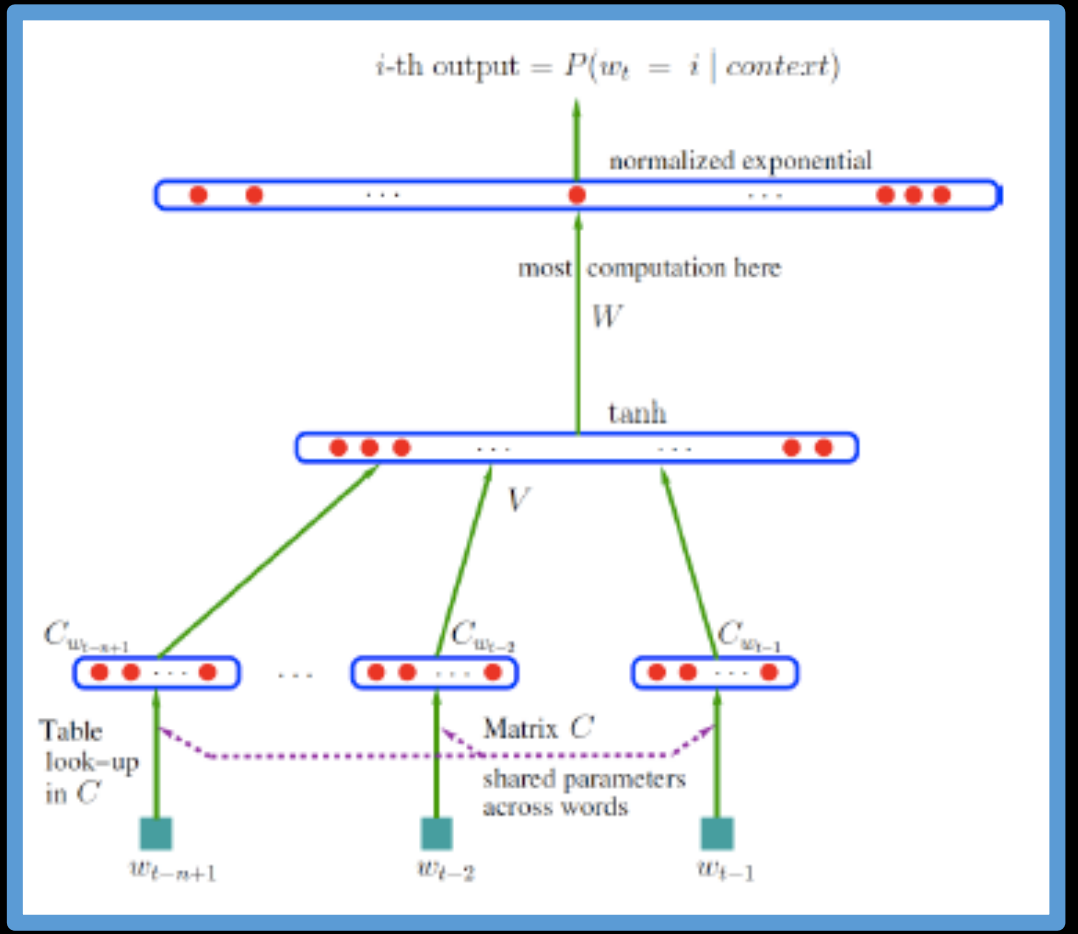
- One hot representation
  - Words as atomic symbols
  - In terms of Vector Space, this is a vector with a 1 and rest dimensions 0
  - Dimensionality: 20K (Speech), 500K (standard dictionaries), 13M (Google)

- Problems:
  - Data Sparsity
  - Ignores semantic associations between words
    - Hello how are you?
    - Hey, how are you?
    - Hi how do you do?

Neural Word Embeddings as a distributed representations

- A word is represented as a dense vector
  - E.g. hello = [0.398, -0.679, 0.112, 0.825, -0.566]

- Vector subtraction is shown to yield meaningful results using DL based representations
  - king – queen = man – woman
  - India – Delhi = France – Paris
  - Cricket – Sachin = Soccer - Peale

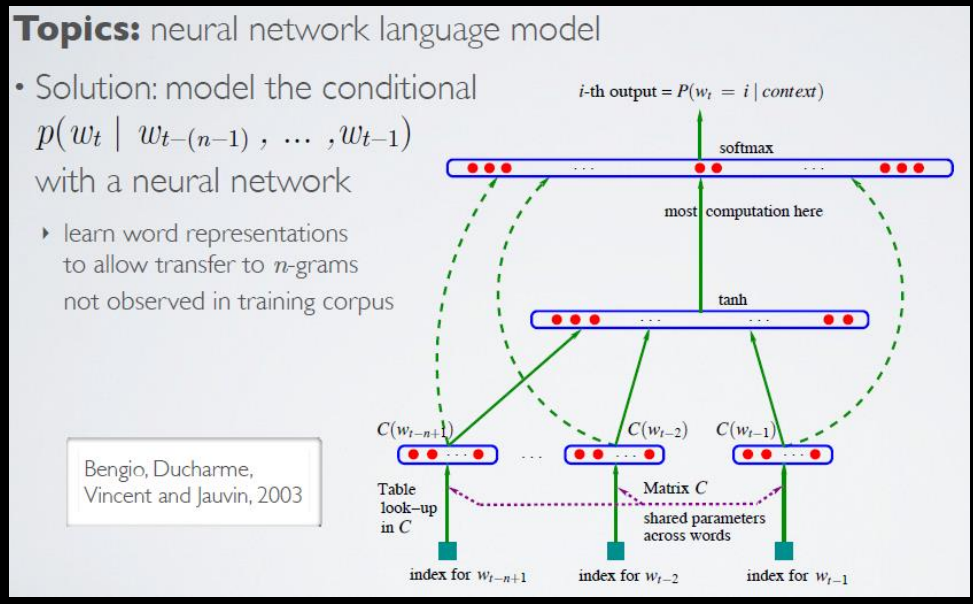


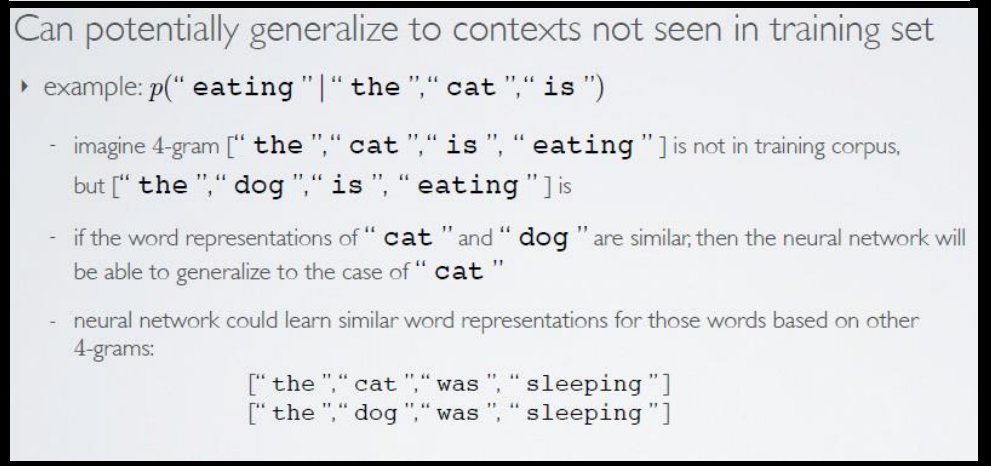
Word Representation with Neural Networks

- Fig from ref: Hugo

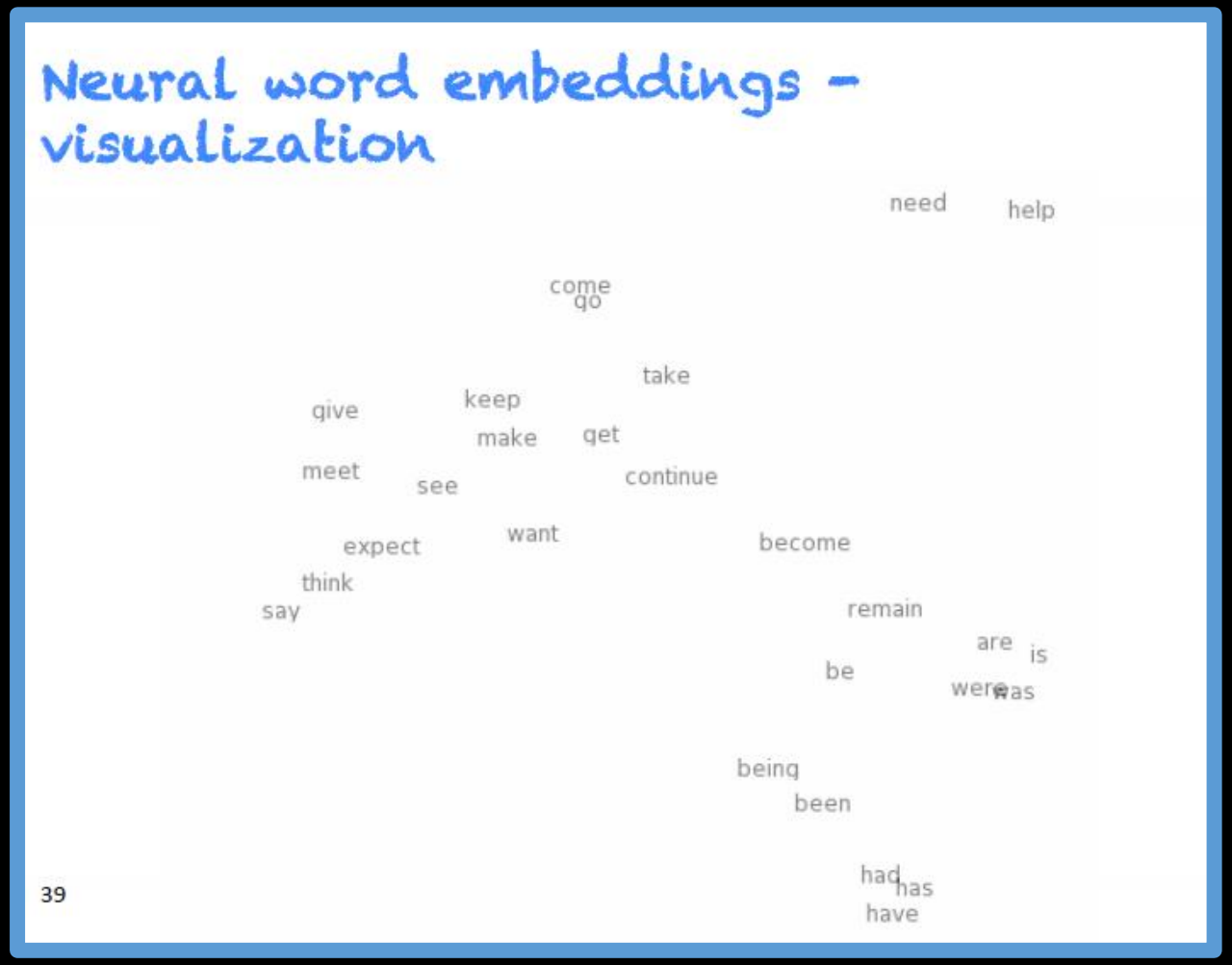
- Bengio proposed a neural network based LM (see prev slide)

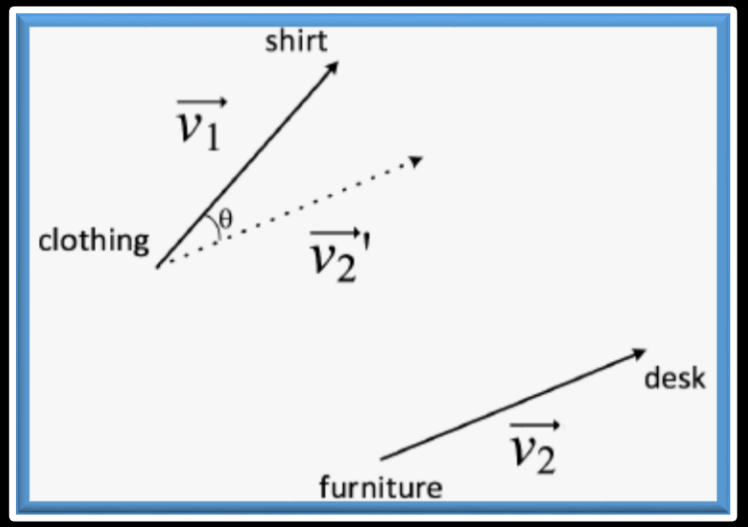
- The word vectors are learnt as a part of training for LM prediction





Visualization – Ref: R Socher and C Manning, Stanford University





Towards low dimensional vectors: word2vec

- Singular Value Decomposition (SVD)
  - SVD is one way to reduce a high dimensional vector space in to low dimensional.
  - Computational cost shoots up quadratically for an n x m matrix
  - Hard to incorporate new words

- Skip grams techniques used in Word2vec
  - Language Model techniques, CBOW models predict a word given its context
  - Skip gram models as used in word2vec do the reverse – predict the context surrounding the given word.
  - The context is specified by the window length
  - A window length say 5 usually is implemented as the given word w being the center word, the context comprising of 2 words occurring prior to w and 2 words after w

Unsupervised Word Vector Learning

- Key idea:
  - A sentence that is likely in a text is a positive example
    - Today is our republic day!
    - The lecture is informative
  - A sentence that is formed by inserting random words in otherwise a correct sentence is a negative example
    - Today is our neural day!
    - The lecture is home
- Evaluate: Score(s1) > Score(s2)
- We use a neural network to compute the score

Word2vec under the hood

- Key idea:
  - Start with an initial assignment of word vector(s) for each word. Predict surrounding words in a window of length c of every word. Compute the cost and adjust the word vector to minimize the cost.
- Hypothesis formulation

For $p(w_{t+j}|w_t)$ the simplest first formulation is

$$p(w_O|w_I) = \frac{\exp\left({v'_{w_O}}^{\top} v_{w_I}\right)}{\sum_{w=1}^{W} \exp\left({v'_w}^{\top} v_{w_I}\right)}$$

where v and v′ are “input” and “output” vector representations of w (so every word has two vectors!)

- Objective function

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

Reference: Stanford CS 224D R Socher (2015)

- Go through each word of the whole corpus
- Predict surrounding words of each (window's center) word

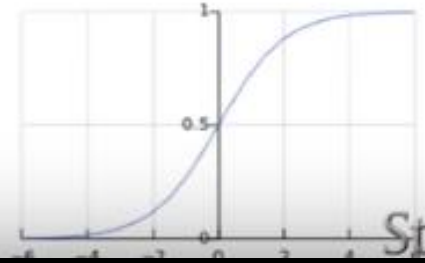
$$p(o|c) = \frac{\exp(u_o^T v_c)}{\sum_{w=1}^{V} \exp(u_w^T v_c)}$$

- Then take gradients at each such window for SGD

- From paper: "Distributed Representations of Words and Phrases and their Compositionality" (Mikolov et al. 2013)
- Overall objective function: $J(\theta) = \frac{1}{T}\sum_{t=1}^{T} J_t(\theta)$

$$J_t(\theta) = \log \sigma\left(u_o^T v_c\right) + \sum_{i=1}^{k} \mathbb{E}_{j\sim P(w)} \left[\log \sigma\left(-u_j^T v_c\right)\right]$$

- The sigmoid function! $\sigma(x) = \frac{1}{1+e^{-x}}$ (we'll become good friends soon)
- So we maximize the probability of two words co-occurring in first log



Stanfo

$$\sum_{t=1}^{T} \left[\sum_{c\in\mathcal{C}_t} \ell(s(w_t, w_c)) + \sum_{n\in\mathcal{N}_{t,c}} \ell(-s(w_t, n)) \right]$$

Updating the vectors

- Compute all gradients: That is all outside vectors and the inside vector for the center word.
- Produce the new vectors for each word
- Iterate

FastText: Using sub-words to enrich the representation

- Using a vector representation directly at the word level ignores morphological information : that is, the way the words themselves are composed of.

$$p(w_c \mid w_t) = \frac{e^{s(w_t,\, w_c)}}{\sum_{j=1}^{W} e^{s(w_t,\, j)}}.$$

- We can improve the representation by taking in to account sub word information: ie: form n-grams of the word, learn representation for each of the n-gram, sum them up to get the vector for word
  - Example: Let the word be: where
  - N-grams are: <wh, whe, her, ere, re> and <where>

$$\sum_{t=1}^{T} \left[\sum_{c \in \mathcal{C}_t} \ell(s(w_t,\, w_c)) + \sum_{n \in \mathcal{N}_{t,c}} \ell(-s(w_t,\, n)) \right]$$

Suppose that you are given a dictionary of n-grams of size G. Given a word w, let us denote by $\mathcal{G}_w \subset \{1, \ldots, G\}$ the set of n-grams appearing in w. We associate a vector representation $\mathbf{z}_g$ to each n-gram g. We represent a word by the sum of the vector representations of its n-grams. We thus obtain the scoring function:

$$s(w, c) = \sum_{g \in \mathcal{G}_w} \mathbf{z}_g^\top \mathbf{v}_c.$$

This simple model allows sharing the representations across words, thus allowing to learn reliable representation for rare words.

Summary of Word2Vec

- Go through each word in the corpus
- Predict the context words
- This captures co-occurrence of words one at a time
- The co-occurrence matrix performs the same function but with a sparse matrix

Ref: R Socher 2018

Count based vs direct prediction

- LSA, HAL (Lund & Burgess),
- COALS, Hellinger-PCA (Rohde et al, Lebret & Collobert)

- Fast training
- Efficient usage of statistics
- Primarily used to capture word similarity
- Disproportionate importance given to large counts

- Skip-gram/CBOW (Mikolov et al)
- NNLM, HLBL, RNN (Bengio et al; Collobert & Weston; Huang et al; Mnih & Hinton)

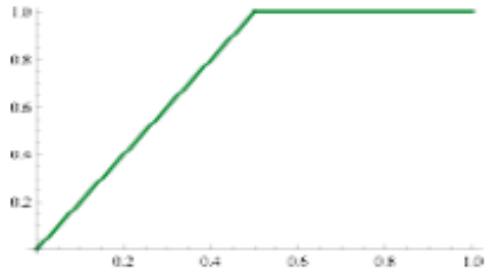
- Scale with corpus size
- Inefficient usage of statistics
- Generate improved performance on other tasks
- Can capture complex patterns beyond word similarity

Slide Credits: R Socher 2018

Combining the best of both worlds: GloVe

$$J(\theta) = \frac{1}{2} \sum_{i,j=1}^{W} f(P_{ij})(u_i^T v_j - \log P_{ij})^2$$

- Fast training
- Scalable to huge corpora

$f \sim$ 

- Good performance even with small corpus, and small vectors
- By Pennington, Socher, Manning (2014)

Evaluating Word Vectors

- Intrinsic
  - King – Queen = Man – Woman
- Extrinsic
  - LM tasks
  - MT tasks

Evaluation Metrics (Source: Wikipedia)

In the field of information retrieval, precision is the fraction of retrieved documents that are relevant to the query:

$$\text{precision} = \frac{|\{\text{relevant documents}\} \cap \{\text{retrieved documents}\}|}{|\{\text{retrieved documents}\}|}$$

For example, for a text search on a set of documents, precision is the number of correct results divided by the number of all returned results.

Precision takes all retrieved documents into account, but it can also be evaluated at a given cut-off rank, considering only the topmost results returned by the system. This measure is called *precision at n* or *P@n*.

Precision is used with recall, the percent of *all* relevant documents that is returned by the search. The two measures are sometimes used together in the F1 Score (or f-measure) to provide a single measurement for a system.

Note that the meaning and usage of "precision" in the field of information retrieval differs from the definition of accuracy and precision within other branches of science and technology.

Recall [edit]

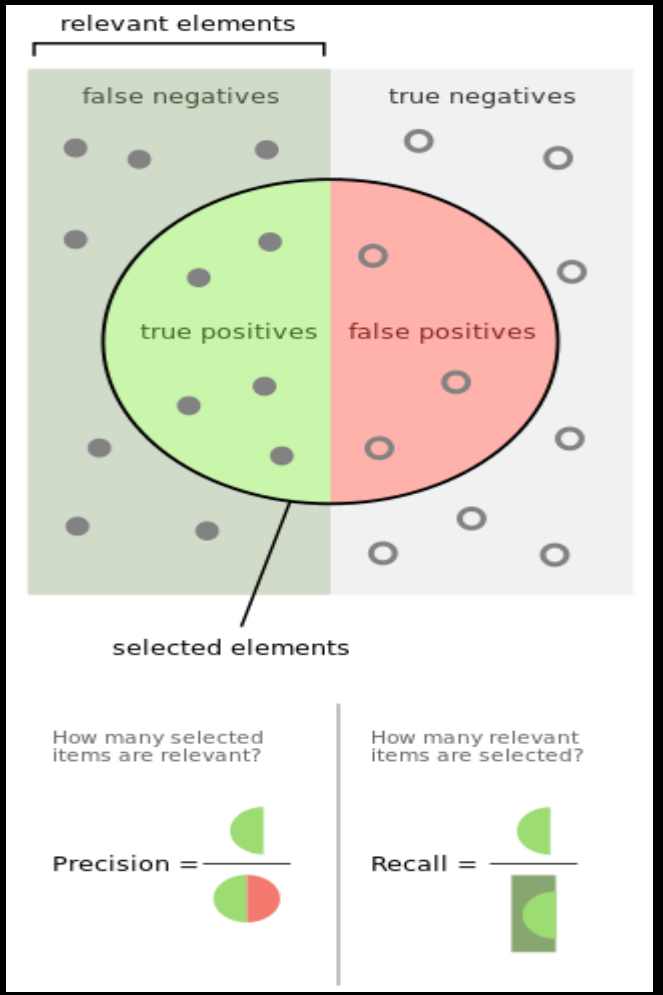
In information retrieval, recall is the fraction of the relevant documents that are successfully retrieved.

$$\text{recall} = \frac{|\{\text{relevant documents}\} \cap \{\text{retrieved documents}\}|}{|\{\text{relevant documents}\}|}$$

For example, for a text search on a set of documents, recall is the number of correct results divided by the number of results that should have been returned.

In binary classification, recall is called sensitivity. It can be viewed as the probability that a relevant document is retrieved by the query.

It is trivial to achieve recall of 100% by returning all documents in response to any query. Therefore, recall alone is not enough but one needs to measure the number of non-relevant documents also, for example by also computing the precision.



Review

- Sentence and Word Tokenization
- Text normalization: Lower casing, Stop words removal, stemming, lemmatization
- Tf, idf, tf-idf
- Atomic representations: wordnet, one hot vector, cosine similarity
- Distributed representations: Co occurrence matrix, Word2Vec, glove
- Evaluation metrics: Intrinsic, extrinsic