Natural Language Processing
DSECL ZG565

Dr. Chetana Gavankar, Ph.D,
IIT Bombay-Monash University Australia
Chetana.gavankar@pilani.bits-pilani.ac.in

BITS Pilani
Pilani Campus

innovate achieve lead

BITS Pilani
Pilani Campus

Session 3

These slides are prepared by the instructor, with grateful acknowledgement of Jurafsky and Martin and many others who made their course materials freely available online.

Session Content

Vector Semantics and Word Embedding (Chapter 6 Jurafsky and Martin Book)

- Lexical semantics
- Vector semantics
- Word and Vectors
- TFIDF
- Word2Vec
  - Skip gram
  - CBOW
- Glove
- Visualizing Embedding's

Lexical semantics

- **Structure** of words : **morphology**
- **Distribution** of words: **language modeling**
- **Meaning** of words: **lexical semantics**
- **Distributional hypothesis:** a way to identify words with similar meanings
  - Words that occur in **similar contexts** tend to have **similar meanings**
  - E.g. oculist and eye-doctor occur near words like eye or examined
  - Amount of meaning difference between two words "corresponding roughly to the amount of difference in their environments (Context)"
  - two words that occur in very similar distributions (whose neighboring words are similar) have similar meanings.
  - **Basic idea: Measure the semantic similarity of words in terms of the similarity of the contexts in which they appear**

Applications

Question answering:

Q: "How ***tall*** is Mt. Everest?"

Ans: "The official ***height*** of Mount Everest is 29029 feet"

"tall"* is similar to *"height"

Plagiarism detection

MAINFRAMES

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

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

MAINFRAMES

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

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

What do words mean?

- N-gram or text classification methods we've seen so far
  - Words are just strings (or indices w_i in a vocabulary list)
  - That's not very satisfactory!
- Introductory logic classes:
  - The meaning of "dog" is DOG; cat is CAT

    $\forall x\ DOG(x) \longrightarrow MAMMAL(x)$
- Old linguistics joke by Barbara Partee in 1967:
  - Q: What's the meaning of life?
  - A: LIFE
- That seems hardly better!

Lemmas and senses

lemma

mouse (N)

sense

1. any of numerous small rodents...
2. a hand-operated device that controls a cursor...

Modified from the online thesaurus WordNet

A sense or "concept" is the meaning component of a word

Lemmas can be polysemous (have multiple senses)

Lexical semantics

- lexical semantics, the linguistic **study of word meaning**
- Concepts or **word senses**
  - Have a complex many-to-many association with words
- Have relations with each other
  - Synonymy
  - Similarity
  - Relatedness: semantic field and frame
  - Antonymy
  - Connotation
- **More generally, a model of word meaning should allow us to draw inferences to address meaning-related tasks like question-answering or dialogue.**

Relations between senses: **Synonymy**

- Synonyms have the same meaning in some or all contexts.
  - Same word sense, substitutable for one another, same propositional meaning, truth preserving
  - E.g couch/sofa, vomit/throw up, car/automobile

- water/H20 : "H20" in a surfing guide?
- big/large: my big sister != my large sister

- In practice, the word synonym is therefore used to describe a relationship of approximate or rough synonymy,

Relation: **Similarity**

- Words with similar meanings. Not synonyms, but sharing some element of meaning (Cat is not a synonym of dog, but cats and dogs are certainly similar words)
  - car, bicycle ; cow, horse
- The notion of word similarity is very useful in larger semantic tasks.
  - Help in computing how similar the meaning of two phrases or sentences are
  - Applications : question answering, paraphrasing, and summarization.
- Humans to judge how similar one word is to another

word1	word2	similarity
vanish	disappear	9.8
behave	obey	7.3
belief	impression	5.95
muscle	bone	3.65
modest	flexible	0.98
hole	agreement	0.3

SimLex-999 dataset (Hill et al., 2015)

Relation: Word relatedness

- Also called "word association"
- Words can be related in any way, perhaps via a semantic frame or field

  - `coffee, tea:` **similar**
  - `coffee, cup:` **related**, not similar

Semantic field

- Words that
  - cover a particular semantic domain
  - bear structured relations with each other.

hospitals

surgeon, scalpel, nurse, anaesthetic, hospital

restaurants

waiter, menu, plate, food, menu, chef

houses

door, roof, kitchen, family, bed

Relation: Antonymy

- Senses that are opposites with respect to only one feature of meaning
- Otherwise, they are very similar!

`dark/light` `short/long` `fast/slow` `rise/fall`
`hot/cold` `up/down` `in/out`

- More formally: antonyms can
  - define a binary opposition or be at opposite ends of a scale
    - `long/short, fast/slow`
  - Be *reversives*:
    - `rise/fall, up/down`

Connotation (sentiment)

- Words have affective meanings
  - Positive connotations (happy)
  - Negative connotations (sad)
- Evaluation (sentiment!)
  - Positive evaluation (great, love)
  - Negative evaluation (terrible, hate)

Connotation

Words seem to vary along 3 affective dimensions:

- **valence**: the pleasantness of the stimulus
- **arousal**: the intensity of emotion provoked by the stimulus
- **dominance**: the degree of control exerted by the stimulus

Osgood et al. (1957)

	Word	Score		Word	Score
Valence	love	1.000		toxic	0.008
	happy	1.000		nightmare	0.005
Arousal	elated	0.960		mellow	0.069
	frenzy	0.965		napping	0.046
Dominance	powerful	0.991		weak	0.045
	leadership	0.983		empty	0.081

Values from NRC VAD Lexicon (Mohammad 2018)

Computational models of word meaning

Can we build a theory of how to represent word meaning?

We'll introduce **vector semantics**

- The standard model in language processing!
- Handles many of our goals!

Let's define words by their usages

One way to define "usage":

words are defined by their environments (the words around them)

Zellig Harris (1954):

If A and B have almost identical environments we say that they are synonyms.

What does recent English borrowing *ongchoi* mean?

- Suppose you see these sentences:
  - Ong choi is delicious **sautéed with garlic**.
  - Ong choi is superb **over rice**
  - Ong choi **leaves** with salty sauces
- And you've also seen these:
  - …spinach **sautéed with garlic over rice**
  - Chard stems and **leaves** are **delicious**
  - Collard greens and other **salty** leafy greens
- Conclusion:
  - Ongchoi is a leafy green like spinach, chard, or collard greens
    - We could conclude this based on words like "leaves" and "delicious" and "sauteed"

Vector Semantics

Idea 1: Defining meaning by linguistic distribution

Idea 2: Meaning as a point in multidimensional space

Word embeddings

- Vectors for representing words are called embeddings
- Each word = a vector (not just "good" or "w_{45}")
- Defining meaning as a point in space based on distribution
- Similar words are "**nearby in semantic space**"
- **Every modern NLP algorithm uses embeddings as the representation of word meaning**

two-dimensional (t-SNE) projection of embeddings

We define meaning of a word as a vector

- Called an "embedding" because it's embedded into a space
- The standard way to represent meaning in NLP
- **Every modern NLP algorithm uses embeddings as the representation of word meaning**
- Fine-grained model of meaning for similarity

Intuition: why vectors?

- Consider sentiment analysis:
  - With **words**, a feature is a word identity
    - Feature 5: 'The previous word was "terrible"'
    - requires **exact same word** to be in training and test
  - With **embeddings**:
    - Feature is a word vector
    - '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!!!

Word embeddings: types

1. Frequency based Embedding
   - A common baseline model
   - **Sparse** long vectors
   - Words are represented by (a simple function of) the **counts** of nearby words
   - **dimensions** corresponding to **words** in the vocabulary or **documents** in a collection
   - **Count Vector**
   - **TF-IDF Vector**
   - **Co-Occurrence Vector**
2. Prediction based Embedding
   - **Dense** vectors
   - Representation is created by training a classifier to **predict** whether a word is likely to appear nearby
   - **Word2vec: Skip-gram and CBOW**
   - **GloVe**

Vectors are the basis of information retrieval

Each document is represented by a vector of words

	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	1	0	7	13
good	114	80	62	89
fool	36	58	1	4
wit	20	15	2	3

Vectors are similar for the two comedies

But comedies are different than the other two

Comedies have more *fools* and *wit* and fewer *battles*.

Visualizing document vectors

Idea for word meaning: Words can be vectors too!!!

	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	1	0	7	13
good	114	80	62	89
fool	36	58	1	4
wit	20	15	2	3

battle is "the kind of word that occurs in Julius Caesar and Henry V"

fool is "the kind of word that occurs in comedies, especially Twelfth Night"

More common: word-word matrix (or "term-context matrix")

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

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	...

computer
4000
3000
2000
1000
information
[3982,3325]
digital
[1683,1670]
1000
2000
3000
4000
data

Computing word similarity: Dot product and cosine

- The dot product between two vectors is a scalar:

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

- The dot product tends to be high when the two vectors have large values in the same dimensions
- Dot product can thus be a useful similarity metric between vectors

Problem with raw dot-product

- Dot product favors long vectors
- Dot product is higher if a vector is longer (has higher values in many dimension)
- Vector length:

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

- Frequent words (of, the, you) have long vectors (since they occur many times with other words).
- So dot product overly favors frequent words

Alternative: cosine for computing word similarity

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

Based on the definition of the dot product between two vectors a and b

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

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

Cosine as a similarity metric

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

- But since raw frequency values are non-negative, the cosine for term-term matrix vectors ranges from 0–1

Cosine examples

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

	pie	data	computer
cherry	442	8	2
digital	5	1683	1670
information	5	3982	3325

$$\cos(\text{cherry},\text{information}) =$$

$$\frac{442*5+8*3982+2*3325}{\sqrt{442^2+8^2+2^2}\sqrt{5^2+3982^2+3325^2}} = .017$$

$$\cos(\text{digital},\text{information}) =$$

$$\frac{5*5+1683*3982+1670*3325}{\sqrt{5^2+1683^2+1670^2}\sqrt{5^2+3982^2+3325^2}} = .996$$

Visualizing cosines (well, angles)

Tf-idf

- Raw frequency is a bad representation
- The co-occurrence matrices we have seen represent each cell by word frequencies.
- Frequency is clearly useful; if *sugar* appears a lot near *apricot*, that's useful information.
- But overly frequent words like *the*, *it,* or *they* are not very informative about the context
- It's a paradox! How can we balance these two conflicting constraints?

- when the dimensions are document word weighting using tf-idf
  - **tf-idf:** tf-idf value for word t in document d:

$$\boxed{w_{t,d} = \mathrm{tf}_{t,d} \times \mathrm{idf}_t}$$

  - Words like "the" or "it" have very low idf

Term frequency (tf)

$$\mathbf{tf}_{t,d} = \mathbf{count}(t,d)$$

Instead of using raw count, we squash a bit:

$$\mathbf{tf}_{t,d} = \log_{10}(\mathbf{count}(t,d)+1)$$

Document frequency (df)

- The second factor in ***tf-idf*** is used to give a higher weight to words that occur only in a few documents
- df_t is the number of documents t occurs in.
- (note this is not collection frequency: total count across all documents)
- "Romeo" is very distinctive for one Shakespeare play:

	Collection Frequency	Document Frequency
Romeo	113	1
action	113	31

Inverse document frequency (idf)

$$\text{idf}_t = \log_{10}\left(\frac{N}{\text{df}_t}\right)$$

N is the total number of documents in the collection

The fewer documents in which a term occurs, the higher this weight

Word	df	idf
Romeo	1	1.57
salad	2	1.27
Falstaff	4	0.967
forest	12	0.489
battle	21	0.246
wit	34	0.037
fool	36	0.012
good	37	0
sweet	37	0

Final tf-idf weighted value for a word

Raw counts:

	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	1	0	7	13
good	114	80	62	89
fool	36	58	1	4
wit	20	15	2	3

tf-idf: $w_{t,d} = \text{tf}_{t,d} \times \text{idf}_t$

Word: *wit*

tf = $\log_{10}(20+1)$ = 1:322, idf = $\log_{10}(37/34)$=0.037, tf-idf value=0.049

Word: *good*

appears in every document so tf-idf = 0

	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	0.074	0	0.22	0.28
good	0	0	0	0
fool	0.019	0.021	0.0036	0.0083
wit	0.049	0.044	0.018	0.022

Sparse versus dense vectors

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

Common methods for getting short dense vectors

- "Neural Language Model"-inspired models
  - Word2vec (skipgram, CBOW), GloVe

- **Alternative to these "static embeddings":**
  - Contextual Embeddings (ELMo, BERT)
  - Compute distinct embeddings for a word in its context
  - Separate embeddings for each token of a word

Word2vec

- Instead of **counting** how often each word *w* occurs near "*apricot*"
  - Train a classifier on a binary **prediction** task:
    - Is *w* likely to show up near "*apricot*"?
- We don't actually care about this task
    - But we'll take the learned classifier weights as the word embeddings
- Big idea: **self-supervision**:
    - A word c that occurs near apricot in the corpus cats as the gold "correct answer" for supervised learning
    - No need for human labels

Word2vec

Popular embedding method

Very fast to train

Code available on the web

Idea: **predict** rather than **count**

Word2vec provides various options. We'll do:

skip-gram with negative sampling (SGNS)

Simple static embeddings you can download!

- Word2vec (Mikolov et al)
- https://code.google.com/archive/p/word2vec/

- GloVe (Pennington, Socher, Manning)
- http://nlp.stanford.edu/projects/glove/

Basic word representation

V = [a, aaron, …, zulu, <UNK>]

1-hot representation

Man (5391)	Woman (9853)	King (4914)	Queen (7157)	Apple (456)	Orange (6257)
$\begin{bmatrix}0\\0\\0\\0\\\vdots\\1\\\vdots\\0\\0\end{bmatrix}$	$\begin{bmatrix}0\\0\\0\\0\\\vdots\\1\\\vdots\\0\end{bmatrix}$	$\begin{bmatrix}0\\0\\0\\\vdots\\1\\\vdots\\0\\0\\0\end{bmatrix}$	$\begin{bmatrix}0\\0\\0\\0\\\vdots\\1\\\vdots\\0\end{bmatrix}$	$\begin{bmatrix}0\\\vdots\\1\\\vdots\\0\\0\\0\\0\\0\end{bmatrix}$	$\begin{bmatrix}0\\0\\0\\0\\\vdots\\1\\\vdots\\0\end{bmatrix}$

I want a glass of orange _____.

I want a glass of apple _____.

Featurized representation: word embedding

	Man (5391)	Woman (9853)	King (4914)	Queen (7157)	Apple (456)	Orange (6257)
Gender	−1	1	-0.95	0.97	0.00	0.01
Royal	0.01	0.02	0.93	0.95	-0.01	0.00
Age	0.03	0.02	0.70	0.69	0.03	-0.02
Food	0.09	0.01	0.02	0.01	0.95	0.97

I want a glass of orange I want a glass of apple

[Mikolov et. al., 2013, Linguistic regularities in continuous space word representations]

Word embeddings

Embedding matrix

In practice, use specialized function to look up an embedding.

→ Embedding

BITS Pilani, Pilani Campus

Word2vec

- Instead of **counting** how often each word *w* occurs near "*apricot*"
  - Train a classifier on a binary **prediction** task:
    - Is *w* likely to show up near "*apricot*"?
- We don't actually care about this task
    - But we'll take the learned classifier weights as the word embeddings
- Big idea: **self-supervision**:
    - A word c that occurs near apricot in the corpus cats as the gold "correct answer" for supervised learning
    - No need for human labels
    - Bengio et al. (2003); Collobert et al. (2011)

Word2vec

- An unsupervised NLP method developed by Google in 2013 (Mikolov et al. 2013)
- Quantify the relationship between words
- Framework for learning word vectors Idea:
  - We have a large corpus (“body”) of text: a long list of words
  - Every word in a fixed vocabulary is represented by a vector
  - Go through each position t in the text, which has a center word c and context (“outside”) words o
  - Use the similarity of the word vectors for c and o to calculate the probability of o given c (or vice versa)
  - Keep adjusting the word vectors to maximize this probability

http://web.stanford.edu/class/cs224n/slides/cs224n-2022-lecture01-wordvecs1.pdf

Approach: predict if candidate word *c* is a "neighbor"

1. Treat the target word *t* and a neighboring context word *c* as **positive examples**.
2. Randomly sample other words in the lexicon to get negative examples
3. Use logistic regression to train a classifier to distinguish those two cases
4. Use the learned weights as the embeddings

Skip-Gram Training Data

- Assume a +/- 2 word window, given training sentence:

[target]

…lemon, a [tablespoon of apricot jam, a] pinch…

- c1 c2 c3 c4

Skip-Gram Classifier

- (assuming a +/- 2 word window)

…lemon, a [tablespoon of apricot jam, a] pinch…

c1 c2 [target] c3 c4

Goal: train a classifier that is given a candidate (**w**ord, **c**ontext) pair

(apricot, jam)

(apricot, aardvark)

…

And assigns each pair a probability:

$P(+|w, c)$

$P(-|w, c) = 1 - P(+|w, c)$

Similarity is computed from dot product

- Remember: two vectors are similar if they have a high dot product
  - Cosine is just a normalized dot product
- So:
  - Similarity(w,c) $\propto w \cdot c$
- We'll need to normalize to get a probability
  - (cosine isn't a probability either)

Turning dot products into probabilities

- Sim(w,c) ≈ w · c
- To turn this into a probability
- We'll use the sigmoid from logistic regression:

$$
\begin{aligned}
P(+|w,c) &= \sigma(c \cdot w) = \frac{1}{1+\exp(-c \cdot w)} \\
P(-|w,c) &= 1 - P(+|w,c) \\
&= \sigma(-c \cdot w) = \frac{1}{1+\exp(c \cdot w)}
\end{aligned}
$$

How Skip-Gram Classifier computes $P(+|w, c)$

$$P(+|w,c) = \sigma(c \cdot w) = \frac{1}{1+\exp(-c \cdot w)}$$

This is for one context word, but we have lots of context words.
We'll assume independence and just multiply them:

$$P(+|w, c_{1:L}) = \prod_{i=1}^{L} \sigma(c_i \cdot w)$$

$$\log P(+|w, c_{1:L}) = \sum_{i=1}^{L} \log \sigma(c_i \cdot w)$$

Skip-gram classifier: summary

- A probabilistic classifier, given
  - a test target word w
  - its context window of L words $c_{1:L}$
- Estimates probability that w occurs in this window based on similarity of w (embeddings) to $c_{1:L}$ (embeddings).

- To compute this, we just need embeddings for all the words.

These embeddings we'll need: a set for w, a set for c

$$\boldsymbol{\theta} =$$

Skip-Gram Training data

- For every + example select k negative examples

…lemon, a [tablespoon of apricot jam, a] pinch…

c1 c2 [target] c3 c4

positive examples +

t	c
apricot	tablespoon
apricot	of
apricot	jam
apricot	a

The paper (Mikolov et al., 2013) says that K=2 ~ 5 works for large data sets, and K=5 ~ 20 for small data sets.

Skip-Gram Training data

…lemon, a [tablespoon of apricot jam, a] pinch…

c1 c2 [target] c3 c4

positive examples +

t	c
apricot	tablespoon
apricot	of
apricot	jam
apricot	a

For each positive example we'll grab k negative examples, sampling by frequency

Skip-Gram Training data

…lemon, a [tablespoon of apricot jam, a] pinch…

c1 c2 [target] c3 c4

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

Skip-gram parameters

- SGNS learns two sets of embeddings
  - Target embeddings matrix W
  - Context embedding matrix C
- It's common to just add them together, representing word i as the vector $w_i + c_i$
- Thus the parameters we need to learn are two matrices W and C, each containing an embedding for every one of the |V| words in the vocabulary |V|

Word2vec training : how to learn vectors

- Given the set of positive and negative training instances, and an initial set of embedding vectors
- The goal of learning is to adjust those word vectors such that we:
  - **Maximize** the similarity of the target word, context word pairs (w, c_{pos}) drawn from the positive data
  - **Minimize** the similarity of the (w, c_{neg}) pairs drawn from the negative data.

Log Loss function

$$
\begin{aligned}
L_{CE} &= -\log\left[P(+|w,c_{pos})\prod_{i=1}^{k}P(-|w,c_{neg_i})\right] \\
&= -\left[\log P(+|w,c_{pos}) + \sum_{i=1}^{k}\log P(-|w,c_{neg_i})\right] \\
&= -\left[\log P(+|w,c_{pos}) + \sum_{i=1}^{k}\log\left(1-P(+|w,c_{neg_i})\right)\right] \\
&= -\left[\log \sigma(c_{pos}\cdot w) + \sum_{i=1}^{k}\log \sigma(-c_{neg_i}\cdot w)\right]
\end{aligned}
$$

Loss function for one *w with* c_{pos} , c_{neg1} ...c_{negk}

- Maximize the similarity of the target with the actual context words, and minimize the similarity of the target with the *k* negative sampled non-neighbor words.

$$
\begin{aligned}
L_{CE} &= -\log\left[P(+|w,c_{pos})\prod_{i=1}^{k}P(-|w,c_{neg_i})\right] \\
&= -\left[\log P(+|w,c_{pos}) + \sum_{i=1}^{k}\log P(-|w,c_{neg_i})\right] \\
&= -\left[\log P(+|w,c_{pos}) + \sum_{i=1}^{k}\log\left(1-P(+|w,c_{neg_i})\right)\right] \\
&= -\left[\log \sigma(c_{pos}\cdot w) + \sum_{i=1}^{k}\log \sigma(-c_{neg_i}\cdot w)\right]
\end{aligned}
$$

Learning the classifier

- How to learn?
  - Stochastic gradient descent!

- We'll adjust the word weights to
  - make the positive pairs more likely
  - and the negative pairs less likely,
  - over the entire training set.

Intuition of one step of gradient descent

- Tries to shift embeddings
- So the target embeddings (here for apricot) are closer to (have a higher dot product with) context embeddings for nearby words (here jam)
- And further from (lower dot product with) context embeddings for noise words that don't occur nearby (here Tolstoy and matrix).

Reminder: gradient descent

- At each step
  - Direction: We move in the reverse direction from the gradient of the loss function
  - Magnitude: we move the value of this gradient $\frac{d}{dw}L(f(x;w),y)$ weighted by a **learning rate** η
  - Higher learning rate means move *w* faster

$$w^{t+1} = w^{t} - h\frac{d}{dw}L(f(x;w),y)$$

The derivatives of the loss function

$$L_{CE} = -\left[\log\sigma(c_{pos}\cdot w) + \sum_{i=1}^{k}\log\sigma(-c_{neg_i}\cdot w)\right]$$

$$\frac{\partial L_{CE}}{\partial c_{pos}} = [\sigma(c_{pos}\cdot w) - 1]w$$

$$\frac{\partial L_{CE}}{\partial c_{neg}} = [\sigma(c_{neg}\cdot w)]w$$

$$\frac{\partial L_{CE}}{\partial w} = [\sigma(c_{pos}\cdot w) - 1]c_{pos} + \sum_{i=1}^{k}[\sigma(c_{\mathrm{neg}_i}\cdot w)]c_{neg_i}$$

Update equation in SGD

Start with randomly initialized C and W matrices, then incrementally do updates

$$
\begin{aligned}
c_{pos}^{t+1} &= c_{pos}^{t} - \eta[\sigma(c_{pos}^{t} \cdot w^{t}) - 1]w^{t} \\
c_{neg}^{t+1} &= c_{neg}^{t} - \eta[\sigma(c_{neg}^{t} \cdot w^{t})]w^{t} \\
w^{t+1} &= w^{t} - \eta\left[[\sigma(c_{pos} \cdot w^{t}) - 1]c_{pos} + \sum_{i=1}^{k}[\sigma(c_{\mathrm{neg}_i} \cdot w^{t})]c_{neg_i}\right]
\end{aligned}
$$

Skip-gram training example

input word	output word	target	input • output	sigmoid()	Error
not	thou	1	0.2	0.55	0.45
not	aaron	0	-1.11	0.25	-0.25
not	taco	0	0.74	0.68	-0.68

$$P(+|w,c) = \sigma(c \cdot w) = \frac{1}{1+\exp(-c \cdot w)}$$

$$\boxed{\begin{aligned} P(-|w,c) &= 1 - P(+|w,c) \\ &= \sigma(-c \cdot w) = \frac{1}{1+\exp(c \cdot w)} \end{aligned}}$$

Update Model Parameters

https://jalammar.github.io/illustrated-word2vec/

Two sets of embeddings

SGNS learns two sets of embeddings

Target embeddings matrix W

Context embedding matrix C

It's common to just add them together, representing word *i* as the vector $w_i + c_i$

Summary: How to learn word2vec (skip-gram) embeddings

- Start with V random d-dimensional vectors as initial embeddings
- Train a classifier based on embedding similarity
  - Take a corpus and take pairs of words that co-occur as positive examples
  - Take pairs of words that don't co-occur as negative examples
  - Train the classifier to distinguish these by slowly adjusting all the embeddings to improve the classifier performance
  - Throw away the classifier code and keep the embeddings.

Word Embeddings

Figure 7.12 Selecting the embedding vector for word V_5 by multiplying the embedding matrix **E** with a one-hot vector with a 1 in index 5.

Example

Corpus = "Ned Stark is the most honourable man"

Current word-context pair = ("Ned", "Stark")

Current negative words = "pimples", "zebra", "idiot"

	one hot vector				
ned	1	0	0	0	0
stark	0	1	0	0	0
pimples	0	0	1	0	0
zebra	0	0	0	1	0
idiot	0	0	0	0	1

initial embedding matrix			
ned	-0.018	0.404	-0.317
stark	0.204	-0.007	-0.733
pimples	-0.706	0.745	0.002
zebra	-0.492	-0.709	0.133
idiot	-0.628	-0.073	0.502

initial context matrix			
ned	-0.572	-0.588	-0.501
stark	0.116	0.723	-0.689
pimples	-0.94	0.601	0.146
zebra	-0.622	0.811	0.64
idiot	-0.077	-0.375	-0.056

Initial embeddings

Current word-context pair = ***(" Ned ", " Stark ")***

Current negative words = ***"pimples", "zebra", "idiot"***

#	Token	x
0	honorable	0
1	is	0
2	man	0
3	most	0
4	**ned**	**1**
5	**stark**	**0**
6	the	0
	pimples	0
	zebra	0
	idiot	0
Neg.	coins	0
	donkey	0
	machine	0

$\times$

W_{input}

-0.014	0.135	0.109
-0.665	0.669	0.309
0.702	0.621	-0.981
0.214	-0.813	-0.561
-0.018	0.404	-0.317
0.204	-0.007	-0.733
-0.652	-0.097	0.499
-0.706	0.745	0.002
-0.492	-0.709	0.133
-0.628	-0.073	0.502
0.456	0.767	-0.629
0.348	0.544	-0.740
-0.627	0.487	0.466

$=$

h

-0.018
0.404
-0.317

aegis4048.github.io

	input embeddings initial		
ned	-0.018	0.404	-0.317
stark	0.204	-0.007	-0.733
pimples	-0.706	0.745	0.002
zebra	-0.492	-0.709	0.133
idiot	-0.628	-0.073	0.502

	context embeddings initial		
ned	-0.572	-0.588	-0.501
stark	0.116	0.723	-0.689
pimples	-0.94	0.601	0.146
zebra	-0.622	0.811	0.64
idiot	-0.077	-0.375	-0.056

Forward Propagation: Sigmoid output layer

W_{output}		
-0.351	-0.812	-0.082
0.743	-0.394	-0.824
0.472	-0.873	-0.564
-0.627	-0.678	0.895
-0.572	-0.588	-0.501
0.116	0.723	-0.689
-0.165	-0.506	-0.341
-0.940	0.601	0.146
-0.622	0.811	0.640
-0.077	-0.375	-0.056
-0.028	-0.244	0.018
-0.411	-0.893	0.518
-0.691	0.236	0.363

$\times$

h
-0.018
0.404
-0.317

$=$

Token	$c_j \cdot h$	sigmoid → $\sigma(c_j \cdot h)$
honorable		
is		
man		
most		
ned		
stark	0.508	0.624
the		
pimples	0.213	0.553
zebra	0.136	0.534
idiot	-0.132	0.467
coins		
donkey		
machine		

aegis4048.github.io

$$P(+|w,c) = \sigma(\mathbf{c}\cdot\mathbf{w}) = \frac{1}{1+\exp(-\mathbf{c}\cdot\mathbf{w})}$$

	old context embeddings			old i/p word embedding	dot product	sigmoid()
stark	0.116	0.723	-0.689	-0.018	0.508	0.624
pimples	-0.94	0.601	0.146	0.404	0.213	0.553
zebra	-0.622	0.811	0.64	-0.317	0.136	0.534
idiot	-0.077	-0.375	-0.056		-0.132	0.467

Backward Propagation: Prediction Error

our current positive word is Stark, t_j=1 for Stark and t_j=0 for other negative words (pimples, zebra, idiot).

Token	$\sigma(c_j \cdot h)$		t_j		$\sigma(c_j \cdot h) - t_j$
honorable					
is					
man					
most					
ned					
stark	0.624		1		-0.376
the					
pimples	0.553		0		0.553
zebra	0.534		0		0.534
idiot	0.467		0		0.467
coins					
donkey					
machine					

aegis4048.github.io
aegis4048.github.io

Sigmoid()	t	sig-t (pred error)
0.624	1	-0.376
0.553	0	0.553
0.534	0	0.534
0.467	0	0.467

Derivative of loss w.r.t input word embeddings

$$\frac{\partial L_{CE}}{\partial w} = [\sigma(\mathbf{c}_{pos} \cdot \mathbf{w}) - 1]\mathbf{c}_{pos} + \sum_{i=1}^{k} [\sigma(\mathbf{c}_{neg_i} \cdot \mathbf{w})]\mathbf{c}_{neg_i}$$

	context embeddings initial		
ned	-0.572	-0.588	-0.501
stark	0.116	0.723	-0.689
pimples	-0.94	0.601	0.146
zebra	-0.622	0.811	0.64
idiot	-0.077	-0.375	-0.056

sig-t(pred error)	C*(sig-t)			derivative of loss w.r.t input word embeddings		
-0.376	-0.04362	-0.271848	0.259064	-0.93154	0.318454	0.65541
0.553	-0.51982	0.332353	0.080738			
0.534	-0.33215	0.433074	0.34176			
0.467	-0.03596	-0.175125	-0.02615			

Input word embedding update

$$\mathbf{w}^{t+1} = \mathbf{w}^{t} - \eta \left[[\sigma(\mathbf{c}_{pos} \cdot \mathbf{w}^{t}) - 1]\mathbf{c}_{pos} + \sum_{i=1}^{k} [\sigma(\mathbf{c}_{neg_i} \cdot \mathbf{w}^{t})]\mathbf{c}_{neg_i} \right]$$

Token	$W_{input}^{(old)}$				η		∇W_{input}				$W_{input}^{(new)}$		
honorable	-0.014	0.135	0.109	−	0.05	×				=	-0.014	0.135	0.109
is	-0.665	0.669	0.309								-0.665	0.669	0.309
man	0.702	0.621	-0.981								0.702	0.621	-0.981
most	0.214	-0.813	-0.561								0.214	-0.813	-0.561
ned	-0.018	0.404	-0.317				-0.932	0.319	0.655		**0.029**	**0.388**	**-0.350**
stark	0.204	-0.007	-0.733								0.204	-0.007	-0.733
the	-0.652	-0.097	0.499								-0.652	-0.097	0.499
pimples	-0.706	0.745	0.002								-0.706	0.745	0.002
zebra	-0.492	-0.709	0.133								-0.492	-0.709	0.133
idiot	-0.628	-0.073	0.502								-0.628	-0.073	0.502
coins	0.456	0.767	-0.629								0.456	0.767	-0.629
donkey	0.348	0.544	-0.740								0.348	0.544	-0.740
machine	-0.627	0.487	0.466								-0.627	0.487	0.466

aegis4048.github.io
aegis4048.github.io

word	old input word embedding			LR	LR* derivative w.r.t input			new i/p word embedding		
ned	-0.018	0.404	-0.317	0.05	-0.04658	0.015923	0.032771	0.029	0.388	-0.35

Derivative of loss w.r.t context word embeddings

Token	$\sigma(c_j \cdot h) - t_j$
honorable	
is	
man	
most	
ned	
stark	-0.376
the	
pimples	0.553
zebra	0.534
idiot	0.467
coins	
donkey	
machine	

$\times$

h
-0.018
0.404
-0.317

$=$

Token	∇W_{ouput}		
honorable			
is			
man			
most			
ned			
stark	0.007	-0.152	0.119
the			
pimples	-0.010	0.223	-0.175
zebra	-0.010	0.216	-0.169
idiot	-0.008	0.189	-0.148
coins			
donkey			
machine			

aegis4048.github.io

$$\frac{\partial L_{CE}}{\partial c_{pos}} = [\sigma(\mathbf{c}_{pos} \cdot \mathbf{w}) - 1]\mathbf{w}$$

$$\frac{\partial L_{CE}}{\partial c_{neg}} = [\sigma(\mathbf{c}_{neg} \cdot \mathbf{w})]\mathbf{w}$$

sigmoid-t (pred error)	i/p word embedding	derivative of loss w.r.t context word embeddings		
-0.376	-0.018	0.007	-0.152	0.119
0.553	0.404	-0.01	0.223	-0.175
0.534	-0.317	-0.01	0.216	-0.169
0.467		-0.008	0.189	-0.148

Context word embedding update

$$c_{pos}^{t+1} = c_{pos}^{t} - \eta\,[\sigma(c_{pos}^{t} \cdot w^{t}) - 1]\,w^{t}$$

$$c_{neg}^{t+1} = c_{neg}^{t} - \eta\,[\sigma(c_{neg}^{t} \cdot w^{t})]\,w^{t}$$

Token	$W_{output}^{(old)}$				η		∇W_{ouput}				$W_{output}^{(new)}$		
honorable	-0.351	-0.812	-0.082	−	0.05	×				=	-0.351	-0.812	-0.082
is	0.743	-0.394	-0.824								0.743	-0.394	-0.824
man	0.472	-0.873	-0.564								0.472	-0.873	-0.564
most	-0.627	-0.678	0.895								-0.627	-0.678	0.895
ned	-0.572	-0.588	-0.501								-0.572	-0.588	-0.501
stark	0.116	0.723	-0.689				0.007	-0.152	0.119		**0.116**	**0.731**	**-0.695**
the	-0.165	-0.506	-0.341								-0.165	-0.506	-0.341
pimples	-0.94	0.601	0.146				-0.010	0.223	-0.175		**-0.940**	**0.590**	**0.155**
zebra	-0.622	0.811	0.64				-0.010	0.216	-0.169		**-0.622**	**0.800**	**0.648**
idiot	-0.077	-0.375	-0.056				-0.008	0.189	-0.148		**-0.077**	**-0.384**	**-0.049**
coins	-0.028	-0.244	0.018								-0.028	-0.244	0.018
donkey	-0.411	-0.893	0.518								-0.411	-0.893	0.518
machine	-0.691	0.236	0.363								-0.691	0.236	0.363

aegis4048.github.io
aegis4048.github.io

word	old contextt word embedding			LR	LR* derivative w.r.t context			new context word embedd		
stark	0.116	0.723	-0.689	0.05	0.00035	-0.0076	0.00595	0.116	0.731	-0.695
pimples	-0.94	0.601	0.146	0.05	-0.0005	0.01115	-0.00875	-0.94	0.59	0.155
zebra	-0.622	0.811	0.64	0.05	-0.0005	0.0108	-0.00845	-0.622	0.8	0.648
idiot	-0.077	-0.375	-0.056	0.05	-0.0004	0.00945	-0.0074	-0.077	-0.384	-0.049

References

- https://aegis4048.github.io/optimize_computational_efficiency_of_skip-gram_with_negative_sampling#pred_error

- Ch-6 : Speech and Language Processing Daniel Jurafsky and James H. Martin

The kinds of neighbors depend on window size

- **Small windows** (C= +/- 2) : nearest words are syntactically similar words in same taxonomy
  - *Hogwarts* nearest neighbors are other fictional schools
    - *Sunnydale, Evernight, Blandings*
- **Large windows** (C= +/- 5) : nearest words are related words in same semantic field
  - *Hogwarts* nearest neighbors are Harry Potter world:
    - *Dumbledore, half-blood, Malfoy*

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

Historical embedding as a tool to study cultural biases

Garg, N., Schiebinger, L., Jurafsky, D., and Zou, J. (2018). Word embeddings quantify 100 years of gender and ethnic stereotypes. Proceedings of the National Academy of Sciences 115(16), E3635–E3644.

- Compute a **gender or ethnic bias** for each adjective: e.g., how much closer the adjective is to "woman" synonyms than "man" synonyms, or names of particular ethnicities
  - Embeddings for **competence** adjective (*smart, wise, brilliant, resourceful, thoughtful, logical)* are biased toward men, a bias slowly decreasing 1960-1990
  - Embeddings for **dehumanizing** adjectives (barbaric, monstrous, bizarre) were biased toward Asians in the 1930s, bias decreasing over the 20th century.
- These match the results of old surveys done in the 1930s

CBOW vs Skip-gram

Figure 1: New model architectures. The CBOW architecture predicts the current word based on the context, and the Skip-gram predicts surrounding words given the current word.

Original word2vec paper: https://arxiv.org/pdf/1301.3781.pdf

CBOW

Corpus
Transformation
Embedding method
"I think [???] I am"
CBOW
"therefore"
Word embeddings

center word
I am happy because I am learning
context words
C = 2
context half-size
window
window size = 5

From corpus to training

From corpus to training

From corpus to training

credits: DeepLearning.AI

CBOW (Continuous Bag of Words)

Skip gram versus CBOW

Skip-gram

– works well with a small amount of the training data,

– represents well even rare words or phrases.

CBOW

– several times faster to train than the skip-gram,

– slightly better accuracy for the frequent words.

Visualizing word embeddings

[van der Maaten and Hinton., 2008. Visualizing data using t-SNE]

Analogical relations

The classic parallelogram model of analogical reasoning

To solve: *"apple is to tree as grape is to ____"*

Figure 6.15 The parallelogram model for analogy problems (Rumelhart and Abrahamson, 1973): the location of $\overrightarrow{\text{vine}}$ can be found by subtracting $\overrightarrow{\text{apple}}$ from $\overrightarrow{\text{tree}}$ and adding $\overrightarrow{\text{grape}}$.

Analogical relations via parallelogram

- The parallelogram method can solve analogies with both sparse and dense embeddings (Turney and Littman 2005, Mikolov et al. 2013b)

  king – man + woman is close to queen

  Paris – France + Italy is close to Rome

- For a problem a:a*::b:b*, the parallelogram method is:

$$\hat{b}^* = \underset{x}{\operatorname{argmax}}\ \operatorname{distance}(x, a^* - a + b)$$

GloVe

- GloVe stands for global vectors for word representation.
- Unsupervised learning algorithm developed by Stanford for generating word embeddings
- GloVe captures both global statistics and local statistics of a corpus, in order to come up with word vectors
- Aggregates global word-word co-occurrence matrix from a corpus
- Idea is learning should be with ratios of co-occurrence probabilities rather than the probabilities themselves.
- The advantage of GloVe is that, unlike Word2vec, GloVe does not rely just on local statistics (local context information of words), but incorporates global statistics (word co-occurrence) to obtain word vectors.

Matrix of word-word co-occurrence counts

- 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

- let the matrix of word-word co-occurrence counts be denoted by X,
- whose entries X_{ij} tabulate the number of times word j occurs in the context of word i
- let $X_i = \sum_k X_{ik}$ be the number of times any word appears in the context of word i
- let $P_{ij} = P(i|j) = X_{ij}/X_i$ be the probability that word j appear in the context of word i

Intuition behind GloVe

ratio of conditional probabilities represents the word meanings

Probability and Ratio	$k = solid$	$k = gas$	$k = water$	$k = fashion$
$P(k\|ice)$	1.9×10^{-4}	6.6×10^{-5}	3.0×10^{-3}	1.7×10^{-5}
$P(k\|steam)$	2.2×10^{-5}	7.8×10^{-4}	2.2×10^{-3}	1.8×10^{-5}
$P(k\|ice)/P(k\|steam)$	8.9	8.5×10^{-2}	1.36	0.96

Very small or large:

solid is related to ice but not steam, or
gas is related to steam but not ice

close to 1:

water is highly related to ice and steam, or
fashion is not related to ice or steam.

- Given a prob word, the ratio can be small, large or equal to 1 depends on their correlations.
- For example, if the ratio is large, the probe word is related to w_i but not w_j.
- This ratio gives us hints on the relations between three different words

References

- Ch-6 : Speech and Language Processing Daniel Jurafsky and James H. Martin
- https://jalammar.github.io/illustrated-word2vec/
- http://web.stanford.edu/class/cs224n/slides/cs224n-2022-lecture02-wordvecs2.pdf
- https://arxiv.org/pdf/1301.3781.pdf
- https://nlp.stanford.edu/pubs/glove.pdf
- https://jonathan-hui.medium.com/nlp-word-embedding-glove-5e7f523999f6
- www.deeplearning.ai

References

Word embedding

- https://www.youtube.com/watch?v=ERibwqs9p38
- https://www.youtube.com/watch?v=EsfNYiLVtHI&t=10s
- https://www.youtube.com/watch?v=lrPxo-92GC0
- https://www.coursera.org/lecture/nlp-sequence-models/
- https://www.youtube.com/watch?v=UqRCEmrv1gQ
- https://www.youtube.com/watch?v=g7wEfamF0Eg
- https://www.youtube.com/watch?v=Sho-b4-9ODE