Word Representation

Tanmoy Chakraborty
Associate Professor, IIT Delhi
https://tanmoychak.com/

Introduction to Large Language Models

‘Meaning’ of a Word

To perform language modelling effectively, it is essential for the model to somehow capture the **meaning** of each word.

Definition: meaning (Webster dictionary)

- The idea that is represented by a word, phrase, etc.
- The idea that a person wants to express by using words, signs, etc.
- The idea that is expressed in a work of writing, art, etc.

Need for Word Representation

For language modeling:

- We need **effective representation** of words
  - The representation must somehow encapsulate the word meaning

Representing Words as Discrete Symbols

In traditional NLP, we regard words as discrete symbols:

hotel, conference, motel – a localist representation

Means one 1, the rest 0s

Such symbols for words can be represented by one-hot vectors:

motel = [0 0 0 0 0 0 0 0 0 0 1 0 0 0 0]

hotel = [0 0 0 0 0 0 0 1 0 0 0 0 0 0 0]

Vector dimension = number of words in vocabulary (e.g., 500,000+)

Problem with Words as Discrete Symbols

Example: in web search, if a user searches for "Delhi motel", we would also like to match documents containing "Delhi hotel"

But:

motel = [0 0 0 0 0 0 0 0 0 0 1 0 0 0 0]

hotel = [0 0 0 0 0 0 0 1 0 0 0 0 0 0 0]

- These two vectors are orthogonal
- There is no natural notion of **similarity** for one-hot vectors!
- **Solution:**
  - Could try to rely on WordNet's list of synonyms to get similarity?

Use Existing Thesauri or Ontologies like WordNet

WordNet 3.0

- A hierarchically organized lexical database
- Online thesaurus + aspects of a dictionary
  - Some other languages available or under development
    - (Arabic, Finnish, German, Portuguese...)

Category	Unique Strings
Noun	117,798
Verb	11,529
Adjective	22,479
Adverb	4,481

Adapted from NLP Lectures by Daniel Jurafsky

Use Existing Thesauri or Ontologies like WordNet

How is "sense" defined in WordNet?

- Using the **synset (synonym set)**, the set of near-synonyms, instantiates a sense or concept, with a gloss.
- Example:
  - *chump* as a noun with the gloss:

    "a person who is gullible and easy to take advantage of"
  - This sense of "chump" is shared by 9 words:

    chump^{1}, fool^{2}, gull^{1}, mark^{9}, patsy^{1}, fall guy^{1}, sucker^{1}, soft touch^{1}, mug^{2}
  - Each of **these** senses have this same gloss
    - (Not **every** sense; sense 2 of gull is the aquatic bird)

Adapted from NLP Lectures by Daniel Jurafsky

Use Existing Thesauri or Ontologies like WordNet

- <u>Example:</u>

Senses of ‘bass’:

Key: "S:" = Show Synset (semantic) relations, "W:" = Show Word (lexical) relations
Display options for sense: (gloss) "an example sentence"

Noun

- S: (n) **bass** (the lowest part of the musical range)
- S: (n) **bass**, bass part (the lowest part in polyphonic music)
- S: (n) **bass**, basso (an adult male singer with the lowest voice)
- S: (n) sea bass, **bass** (the lean flesh of a saltwater fish of the family Serranidae)
- S: (n) freshwater bass, **bass** (any of various North American freshwater fish with lean flesh (especially of the genus Micropterus))
- S: (n) **bass**, bass voice, basso (the lowest adult male singing voice)
- S: (n) **bass** (the member with the lowest range of a family of musical instruments)
- S: (n) **bass** (nontechnical name for any of numerous edible marine and freshwater spiny-finned fishes)

Adjective

- S: (adj) **bass**, deep (having or denoting a low vocal or instrumental range) *"a deep voice"; "a bass voice is lower than a baritone voice"; "a bass clarinet"*

Adapted from NLP Lectures by Daniel Jurafsky

Drawbacks of Thesaurus-based Approaches

- A useful resource but missing nuance
  - e.g., “proficient” is listed as a synonym for “good”: this is only correct in some contexts
  - Also, WordNet lists offensive synonyms in some synonym sets without any coverage of the connotations or appropriateness of words
- Missing new meanings of words
  - e.g., wicked, badass, nifty, wizard, genius, ninja, bombest
  - Impossible to keep up-to-date!
- Subjective
- Requires human labor to create and adapt

Representing Words by Their Context

Distributional semantics: A word's meaning is given by the words that frequently appear close-by.

"You shall know a word by the company it keeps" (J. R. Firth 1957: 11)

- When a word *w* appears in a text, its context is the set of words that appear nearby (within a fixed-size window).
  - We can have many contexts of *w* to build up a representation of *w*
    - …government debt problems turning into banking crises as happened in 2009…
    - …saying that Europe needs unified banking regulation to replace the hodgepodge…
    - …India has just given its banking system a shot in the arm…
  - These context words will represent banking

Count-based Methods

Use Co-occurrences for Word Similarity

The Term-Context matrix (or, word-word matrix)

- Each cell: number of times the row (target) word and the column (context) word co-occur in some context in the corpus
  - Generally, smaller contexts are used, like:
    - Paragraph
    - Window of 10 words
- Each word is a count vector in $\mathbb{N}^V$: a row below (V: size of vocabulary)

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

Adapted from NLP Lectures by Daniel Jurafsky

Sample Contexts: 20 words (Brown corpus)

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

Adapted from NLP Lectures by Daniel Jurafsky

Use Co-occurrences for Word Similarity

The Term-Context matrix (or, word-word matrix)

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

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

Adapted from NLP Lectures by Daniel Jurafsky

Should We Use Raw Counts?

- Raw word frequency is not a great measure of association between words
  - It's very skewed
    - "the" and "of" are very frequent, but maybe not the most discriminative
- We'd rather have a measure that asks whether a context word is **particularly informative** about the target word.
- For the term-document matrix:
  - We generally use tf-idf instead of raw term counts.

Term Frequency (tf)

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

Instead of using raw count, we squash a bit:

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

Document Frequency (df)

df_t *is* the number of documents *t* occurs in.

(note this is NOT *collection frequency*: *total count across all documents*)

Example: "*Romeo*" is very distinctive for one Shakespeare play:

	Collection Frequency	Document Frequency
Romeo	113	1
action	113	31

Inverse Document Frequency (idf)

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

N is the total number of documents in the collection

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

What is a Document?

- Could be a play or a Wikipedia article

- But for the purposes of tf-idf, documents can be **anything**; we often call each paragraph a document!

Final *tf-idf* Weighted Value for a Word

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

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

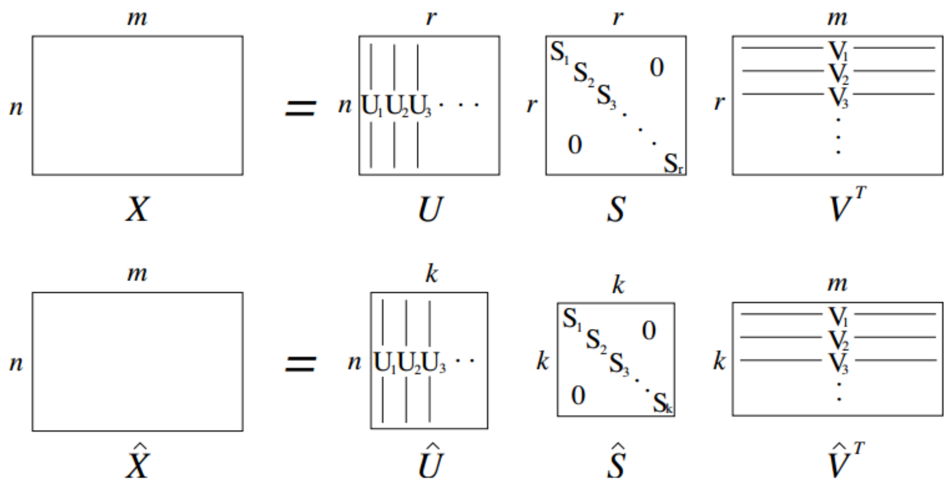
tf-idf	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	0.074	0	0.22	0.28
good	0	0	0	0
fool	0.019	0.021	0.0036	0.0083
wit	0.049	0.044	0.018	0.022

Drawbacks of Co-occurrence Matrix Approach

- Quadratic space needed
- Relative position and order of words not considered

Low Dimensional Vectors

- Store only "important" information in fixed, low dimensional vector.
- Singular Value Decomposition (SVD) on co-occurrence matrix
  - $\hat{X}$ is the best rank k approximation to X, in terms of least squares
  - Motel = [0.286, 0.792, -0.177, -0.107, 0.109, -0.542, 0.349, 0.271]



Drawbacks of SVD-based Approach

- Computational cost scales quadratically for n x m matrix: $O(mn^2)$ flops (when n<m)
- Hard to incorporate new words or documents
- Does not consider order of words

Prediction-based Methods

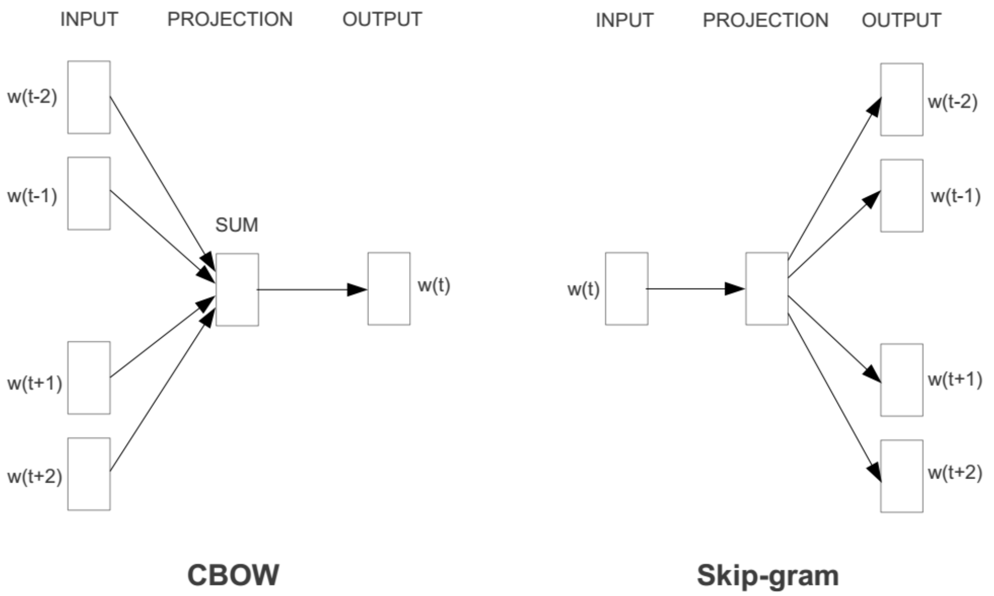
Word Embedding

- Dense vector
- Helps in learning less parameters
- May generalize better
- Can capture synonyms better
  - ***car*** and ***automobile*** are synonyms; but have distinct dimensions
    - A word with ***car*** as a neighbor and a word with ***automobile*** as a neighbor should be similar, but are not

Represent The Meaning of Word: Word2vec

Two basic neural network models:

- **Continuous Bag of Word (CBOW):** use a window of word to **predict the middle word**
- **Skip-gram (SG):** use a word to **predict the surrounding words in window**



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)

Adapted from NLP Lectures by Daniel Jurafsky

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:

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

c_1 c_2 target c_3 c_4

Skip-Gram Classifier

(assuming a +/- 2 word window)

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

c_1 c_2 target c_3 c_4

- **Goal:** Train a classifier, that, **given a candidate (word, context) pair**

  (apricot, jam)
  (apricot, aardvark)
  …

  assigns each pair a probability:

  $P(+ \mid w, c)$

  $P(- \mid w, c) = 1 - P(+ \mid w, c)$

Similarity is Computed Using Dot Product

- **Remember:** *Two vectors are similar if they have a high dot product*
  - Cosine is just a normalized dot product
- **Similarity(w , c) $\propto$ w · 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 function, as in logistic regression:

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

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

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

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:

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

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

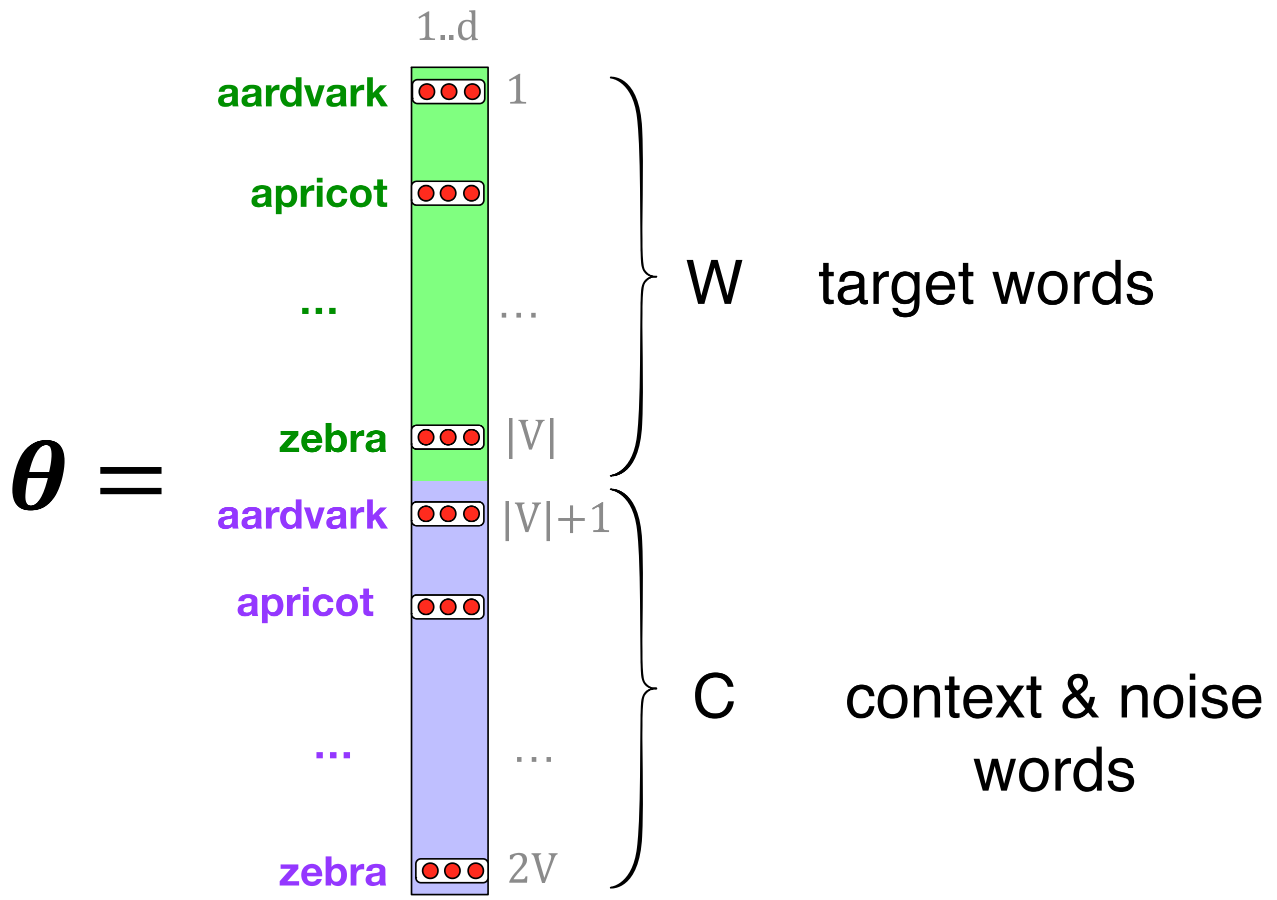
Skip-gram Classifier: Summary

- 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.

The Embeddings We'll Need: A Set for *w*, A Set for *c*



Word2vec: Learning the Embeddings

Skip-Gram Training Data

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

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

c_1 c_2 target c_3 c_4

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

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

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

c_1 c_2 target c_3 c_4

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

Choosing Negative Examples

$$P_\alpha(w) = \frac{count(w)^\alpha}{\sum_{w'} count(w')^\alpha}$$

Setting $\alpha = .75$ gives better performance because it gives rare noise words slightly higher probability: for rare words, $P_\alpha(w) > P(w)$.

$$P(a) = .99 \qquad P_\alpha(a) = \frac{.99^{.75}}{.99^{.75} + .01^{.75}} = .97$$

$$P(b) = .01 \qquad P_\alpha(b) = \frac{.01^{.75}}{.99^{.75} + .01^{.75}} = .03$$

Word2vec: How to Learn Word 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.

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.

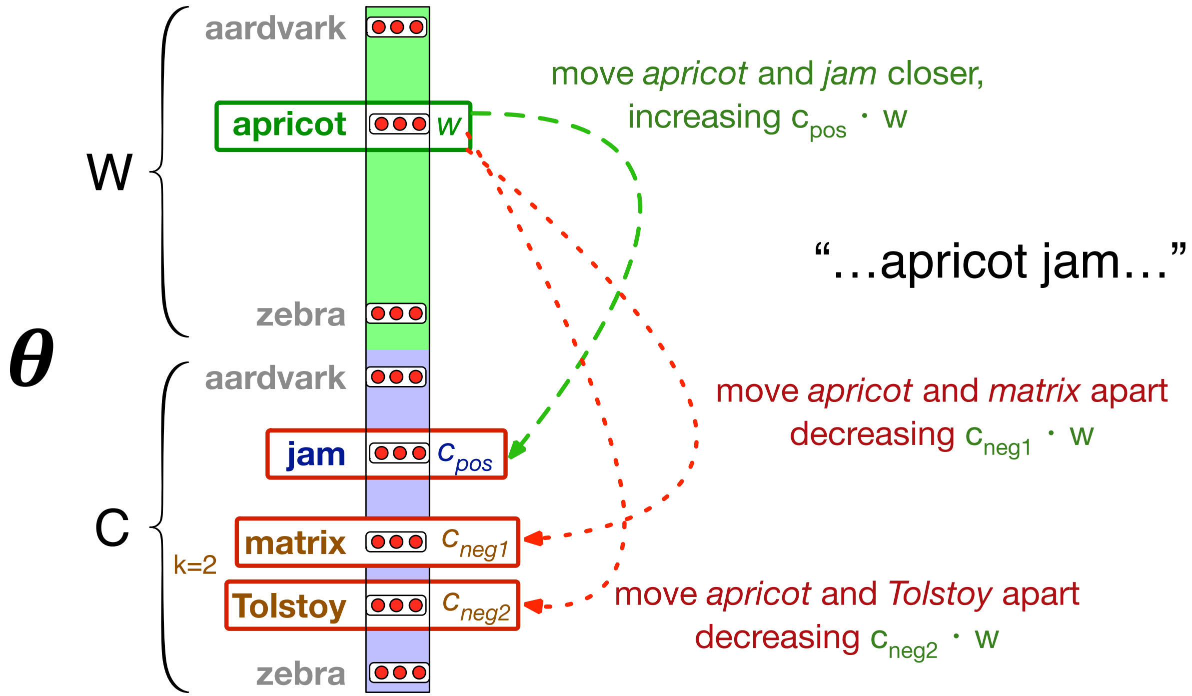
$$\begin{aligned} L_{\text{CE}} &= -\log\left[P(+|w,c_{pos})\prod_{i=1}^{k}P(-|w,c_{neg_i})\right] \\ &= -\left[\log P(+|w,c_{pos})+\sum_{i=1}^{k}\log P(-|w,c_{neg_i})\right] \\ &= -\left[\log P(+|w,c_{pos})+\sum_{i=1}^{k}\log\left(1-P(+|w,c_{neg_i})\right)\right] \\ &= -\left[\log\sigma(c_{pos}\cdot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

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.



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

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

The Derivatives of The Loss Function

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

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

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

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

Update Equation in SGD

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

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

Two Sets of Embeddings

Skip-gram learns **two sets of embeddings**:

1. Target embeddings matrix W
2. Context embedding matrix C

It's common to just add them together, representing **i-th word** 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

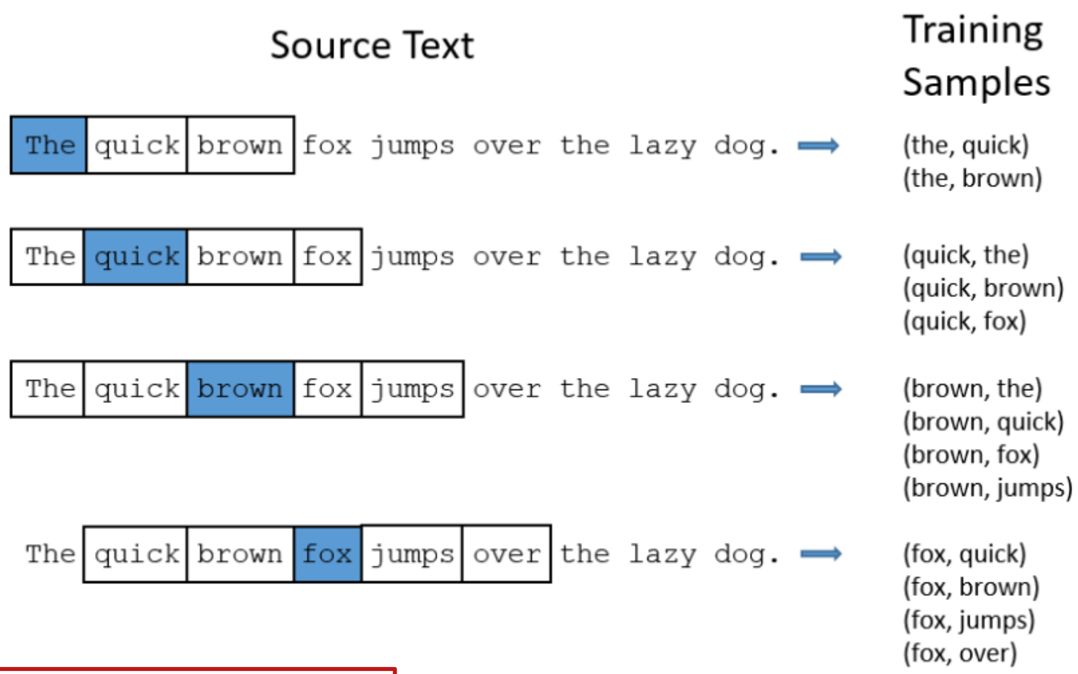
Some Tricks

- **Sub-sampling Frequent Words**

There are two “problems” with common words like “the”:

1. When looking at word pairs, (“fox”, “the”) doesn’t tell us much about the meaning of “fox”. “the” appears in the context of pretty much every word.

2. We will have many more samples of (“the”, …) than we need to learn a good vector for “the”.



$P(w_i)$ is the probability of *keeping* the word:

$$P(w_i) = \left(\sqrt{\frac{z(w_i)}{0.001}} + 1\right) \cdot \frac{0.001}{z(w_i)}$$

http://mccormickml.com/2017/01/11/word2vec-tutorial-part-2-negative-sampling/

Some Tricks

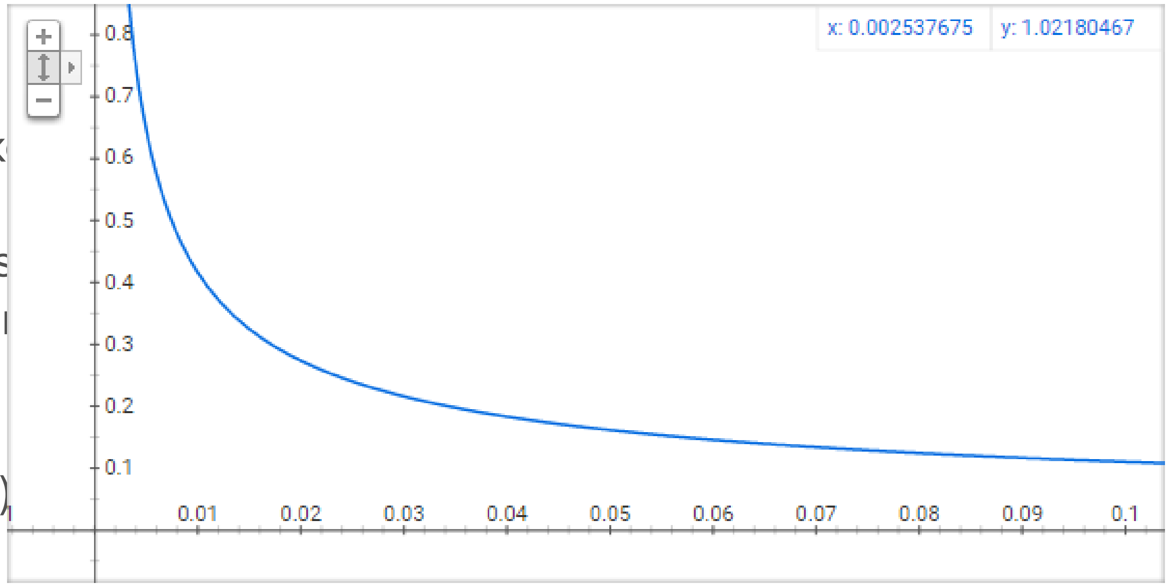
- **Sub-sampling Frequent Words**

There are two “problems” with common words lik

1. When looking at word pairs, (“fox”, “the”) does much about the meaning of “fox”. “the” appea context of pretty much every word.

2. We will have many more samples of (“the”, …) need to learn a good vector for “the”.

Graph for (sqrt(x/0.001)+1)*0.001/x



Training Samples

(the, quick)
(the, brown)

(quick, the)
(quick, brown)
(quick, fox)

(brown, the)
(brown, quick)
(brown, fox)
(brown, jumps)

(fox, quick)
(fox, brown)
(fox, jumps)
(fox, over)

$P(w_i)$ is the probability of *keeping* the word:

$$P(w_i) = \left(\sqrt{\frac{z(w_i)}{0.001}} + 1\right) \cdot \frac{0.001}{z(w_i)}$$

http://mccormickml.com/2017/01/11/word2vec-tutorial-part-2-negative-sampling/

Sub-sampling Frequent Words

- If we have a window size of 10, and we remove a specific instance of "the" from our text:
  - As we train on the remaining words, "the" will not appear in any of their context windows.
  - We'll have 10 fewer training samples where "the" is the input word.

Here are some interesting points in this function (again this is using the default sample value of 0.001).

- $P(w_i) = 1.0$ (100% chance of being kept) when $z(w_i) <= 0.0026$.
  - This means that only words which represent less than 0.26% of the total words will be subsampled.
- $P(w_i) = 0.5$ (50% chance of being kept) when $z(w_i) = 0.00746$.
- $P(w_i) = 0.033$ (3.3% chance of being kept) when $z(w_i) = 1.0$.
  - That is, if the corpus consisted entirely of word w_i, which of course is ridiculous.

Some Interesting Results

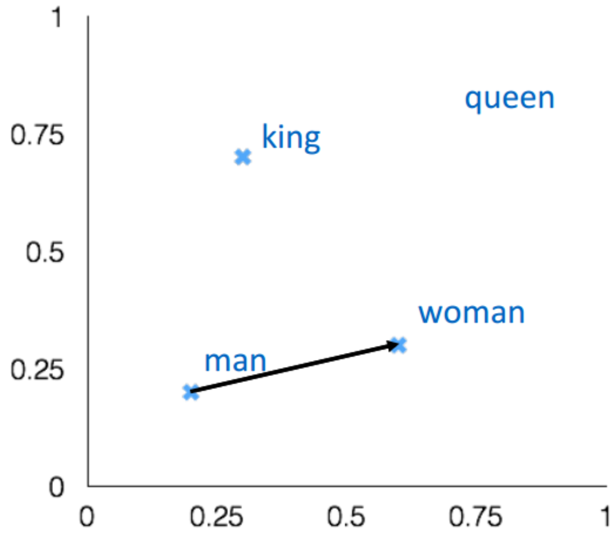
Word Analogies

Test for linear relationships, examined by Mikolov et al. (2014)

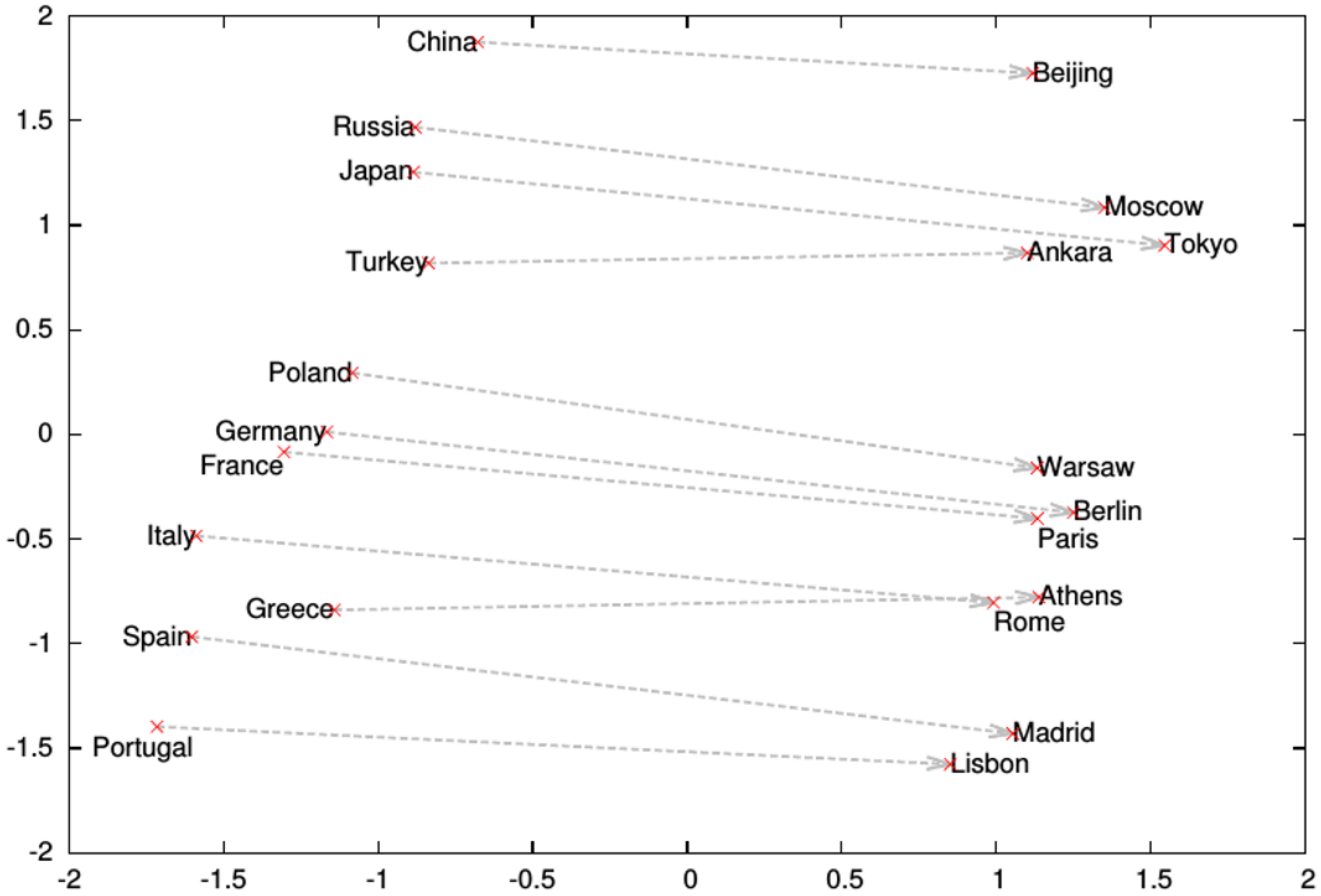
a:b :: c:? → $d = \arg\max_x \frac{(w_b - w_a + w_c)^T w_x}{||w_b - w_a + w_c||}$

man:woman :: king:?

+	king	[0.30 0.70]
-	man	[0.20 0.20]
+	woman	[0.60 0.30]
	queen	[0.70 0.80]



Word Analogies



Problems of Word2vec

The **cat sat on** ***the*** **mat**

Word2vec can't capture the information like:

- Is "The" a special context of the words "cat" and "mat"?

  Or

- Is "The" just a stopword?

Problems of Word2vec

- Word2Vec can’t handle **unknown words** – words appearing in a test corpus but were unseen in the training corpus

fasttext embedding – Subword embedding

- Each word is represented by itself plus a bag of constituent *n*-grams, with special boundary symbols '<' and '>' added to each word.

- For example, with n = 3 the word ***where*** would be represented by the sequence plus the character n-grams:

where, <wh, whe, her, ere, re>

- Skip-gram is learned for each constituent *n*-gram
- ***where*** is represented by the sum of all of the embeddings of its constituent n-grams.
- Unknown words can then be presented only by the sum of the constituent n-grams