Distributional Semantics and Word Embeddings

Chase Geigle

2016-10-14

What is a word?

“dog”

What is a word?

“dog” “canine”

What is a word?

“dog”

“canine”

3

What is a word?

“dog”	“canine”
3	399,999

What is a word?

"dog"	"canine"
3	399,999

$$\begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \\ \vdots \\ 0 \\ 0 \end{pmatrix}$$

What is a word?

"dog"	"canine"
3	399,999
$\begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \\ \vdots \\ 0 \\ 0 \end{pmatrix}$	$\begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ \vdots \\ 1 \\ 0 \end{pmatrix}$

What is a word?

“dog”	“canine”
3	399,999

$$\begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \\ \vdots \\ 0 \\ 0 \end{pmatrix} \qquad \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ \vdots \\ 1 \\ 0 \end{pmatrix}$$

Sparsity!

The Scourge of Sparsity

In the real world, **sparsity** hurts us:

- Low vocabulary coverage in training $\rightarrow$ high OOV rate in applications (poor generalization)
- “one-hot” representations $\rightarrow$ *huge* parameter counts
- information about one word *completely unutilized* for another!

Thus, a good representation must:

- reduce parameter space,
- improve generalization, and
- somehow “transfer” or “share” knowledge between words

The Distributional Hypothesis

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

Words with **high similarity** occur in the **same contexts** as one another.

The Distributional Hypothesis

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

Words with **high similarity** occur in the **same contexts** as one another.

A bit of foreshadowing:

- A **word** ought to be able to **predict its context** (word2vec Skip-Gram)
- A **context** ought to be able to **predict its missing word** (word2vec CBOW)

Brown Clustering

Language Models

A language model is a

Language Models

A language model is a **probability distribution** over

Language Models

A language model is a **probability distribution** over **word sequences**.

Language Models

A language model is a **probability distribution** over **word sequences**.

- Unigram language model $p(w \mid \theta)$:

$$p(\mathbf{W} \mid \theta) = \prod_{i=1}^{N} p(w_i \mid \theta)$$

Language Models

A language model is a **probability distribution** over **word sequences**.

- Unigram language model $p(w \mid \theta)$:

$$p(\mathbf{W} \mid \theta) = \prod_{i=1}^{N} p(w_i \mid \theta)$$

- Bigram language model $p(w_i \mid w_{i-1}, \theta)$:

$$p(\mathbf{W} \mid \theta) = \prod_{i=1}^{N} p(w_i \mid w_{i-1}, \theta)$$

Language Models

A language model is a **probability distribution** over **word sequences**.

- Unigram language model $p(w \mid \theta)$:

$$p(\mathbf{W} \mid \theta) = \prod_{i=1}^{N} p(w_i \mid \theta)$$

- Bigram language model $p(w_i \mid w_{i-1}, \theta)$:

$$p(\mathbf{W} \mid \theta) = \prod_{i=1}^{N} p(w_i \mid w_{i-1}, \theta)$$

- n-gram language model: $p(w_n \mid w_1^{n-1}, \theta)$

$$p(\mathbf{W} \mid \theta) = \prod_{i=1}^{N} p(w_i \mid w_{i-n+1}^{i-1}, \theta)$$

A Class-Based Language Model[1]

Main Idea: cluster words into a fixed number of clusters C and use their cluster assignments as their identity instead (reducing sparsity)

If $\pi : V \to C$ is a mapping function from a word type to a cluster (or class), we want to find

$$\pi^* = \arg\max p(\mathbf{W} \mid \pi)$$

where

$$p(\mathbf{W} \mid \pi) = \prod_{i=1}^{N} p(c_i \mid c_{i-1}) p(w_i \mid c_i).$$

[1] Peter F. Brown et al. "Class-based N-gram Models of Natural Language". In: *Comput. Linguist.* 18.4 (Dec. 1992), pp. 467–479.

Finding the best partition

$$\pi^* = \arg\max_{\pi} P(\mathbf{W} \mid \pi) = \arg\max_{\pi} \log P(\mathbf{W} \mid \pi)$$

One can derive[2] $L(\pi)$ to be

$$L(\pi) = \underbrace{\sum_{w} n_w \log n_w}_{\substack{\text{(nearly) unigram entropy} \\ \text{(fixed w.r.t. } \pi)}} + \underbrace{\sum_{c_i, c_j} n_{c_i, c_j} \log \frac{n_{c_i, c_j}}{n_{c_i} \cdot n_{c_j}}}_{\substack{\text{(nearly) mutual information} \\ \text{(varies with } \pi)}}$$

[2] Sven Martin, Jörg Liermann, and Hermann Ney. "Algorithms for Bigram and Trigram Word Clustering". In: *Speech Commun.* 24.1 (Apr. 1998), pp. 19–37.

Finding the best partition

$$L(\pi) = \underbrace{\sum_{w} n_w \log n_w}_{\substack{\text{(nearly) unigram entropy} \\ \text{(fixed w.r.t. } \pi\text{)}}} + \underbrace{\sum_{c_i, c_j} n_{c_i, c_j} \log \frac{n_{c_i, c_j}}{n_{c_i} \cdot n_{c_j}}}_{\substack{\text{(nearly) mutual information} \\ \text{(varies with } \pi\text{)}}}$$

What does maximizing this mean?

Recall

$$MI(c, c') = \sum_{c_i, c_j} p(c_i, c_j) \log \frac{p(c_i, c_j)}{p(c_i) p(c_j)}$$

which can be shown to be

$$MI(c, c') = \underbrace{\frac{1}{N}}_{\text{(constant)}} \sum_{c_i, c_j} n_{c_i, c_j} \log \frac{n_{c_i, c_j}}{n_{c_i} \cdot n_{c_j}} + \underbrace{\log N}_{\text{(constant)}}$$

and thus **maximizing MI of adjacent classes** selects the best π.

Finding the best partition (cont'd)

$$\pi^* = \arg\max_{\pi} \underbrace{\sum_{w} n_w \log n_w}_{\substack{\text{(nearly) unigram entropy} \\ \text{(fixed w.r.t. } \pi\text{)}}} + \underbrace{\sum_{c_i, c_j} n_{c_i, c_j} \log \frac{n_{c_i, c_j}}{n_{c_i} \cdot n_{c_j}}}_{\substack{\text{(nearly) mutual information} \\ \text{(varies with } \pi\text{)}}}$$

Direct maximization is **intractable!** Thus, agglomerative (bottom-up) clustering is used as a greedy heuristic.

The best merge is determined by the lowest loss in average mutual information.

Agglomerative Clustering

Do they work? (Yes.)

Named entity recognition:

- Scott Miller, Jethran Guinness, and Alex Zamanian. "Name Tagging with Word Clusters and Discriminative Training." In: *HLT-NAACL*. 2004, pp. 337–342
- Lev Ratinov and Dan Roth. "Design Challenges and Misconceptions in Named Entity Recognition". In: *Proc. CoNLL*. 2009, pp. 147–155

Dependency parsing:

- Terry Koo, Xavier Carreras, and Michael Collins. "Simple Semi-supervised Dependency Parsing". In: *Proc. ACL: HLT*. June 2008, pp. 595–603
- Jun Suzuki and Hideki Isozaki. "Semi-Supervised Sequential Labeling and Segmentation Using Giga-Word Scale Unlabeled Data". In: *Proc. ACL: HLT*. June 2008, pp. 665–673

Constituency parsing:

- Marie Candito and Benoît Crabbé. "Improving Generative Statistical Parsing with Semi-supervised Word Clustering". In: IWPT. 2009, pp. 138–141
- Muhua Zhu et al. "Fast and Accurate Shift-Reduce Constituent Parsing". In: *ACL*. Aug. 2013, pp. 434–443

Vector Spaces for Word Representation

Toward Vector Spaces

Brown clusters are nice, but limiting:

- Arbitrary cut-off point $\rightarrow$ two words may be assigned different classes arbitrarily because of our chosen cutoff
- Definition of "similarity" is limited to only bigrams of classes[3]
- Completely misses lots of **regularity** that's present in language

[3] There *are* adaptations of Brown clustering that move past this, but what we'll see today is still even better.

What do I mean by “regularity”?

woman is to **sister** as **man** is to

What do I mean by “regularity”?

woman is to **sister** as **man** is to **brother**

What do I mean by “regularity”?

woman	is to	**sister**	as	**man**	is to	**brother**
summer	is to	**rain**	as	**winter**	is to	

What do I mean by “regularity”?

woman	is to	**sister**	as	**man**	is to	**brother**
summer	is to	**rain**	as	**winter**	is to	**snow**

What do I mean by “regularity”?

woman	is to	**sister**	as	**man**	is to	**brother**
summer	is to	**rain**	as	**winter**	is to	**snow**
man	is to	**king**	as	**woman**	is to	

What do I mean by “regularity”?

woman	is to	**sister**	as	**man**	is to	**brother**
summer	is to	**rain**	as	**winter**	is to	**snow**
man	is to	**king**	as	**woman**	is to	**queen**

What do I mean by “regularity”?

woman	is to	**sister**	as	**man**	is to	**brother**
summer	is to	**rain**	as	**winter**	is to	**snow**
man	is to	**king**	as	**woman**	is to	**queen**
fell	is to	**fallen**	as	**ate**	is to	

What do I mean by “regularity”?

woman	is to	**sister**	as	**man**	is to	**brother**
summer	is to	**rain**	as	**winter**	is to	**snow**
man	is to	**king**	as	**woman**	is to	**queen**
fell	is to	**fallen**	as	**ate**	is to	**eaten**

What do I mean by “regularity”?

woman	is to	**sister**	as	**man**	is to	**brother**
summer	is to	**rain**	as	**winter**	is to	**snow**
man	is to	**king**	as	**woman**	is to	**queen**
fell	is to	**fallen**	as	**ate**	is to	**eaten**
running	is to	**ran**	as	**crying**	is to	

What do I mean by “regularity”?

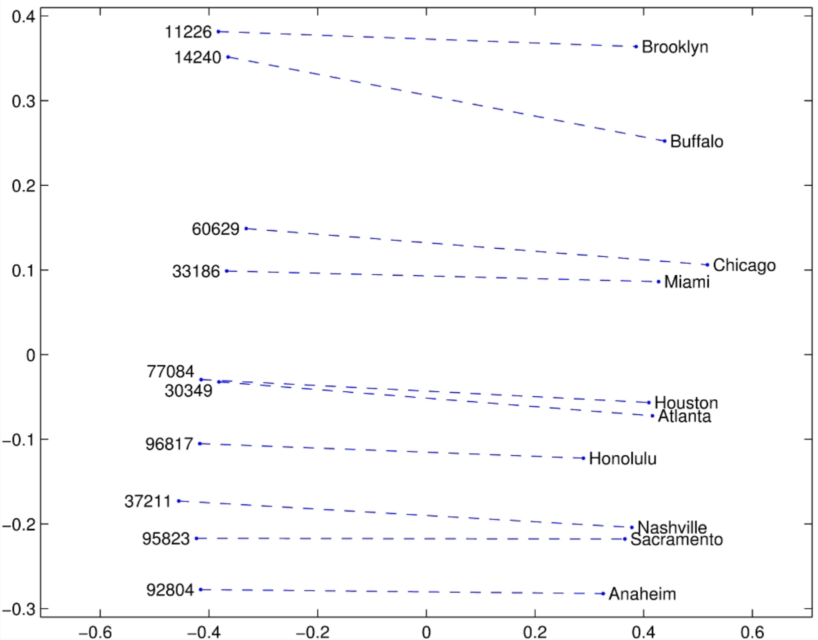
woman	is to	**sister**	as	**man**	is to	**brother**
summer	is to	**rain**	as	**winter**	is to	**snow**
man	is to	**king**	as	**woman**	is to	**queen**
fell	is to	**fallen**	as	**ate**	is to	**eaten**
running	is to	**ran**	as	**crying**	is to	**cried**

What do I mean by "regularity"?

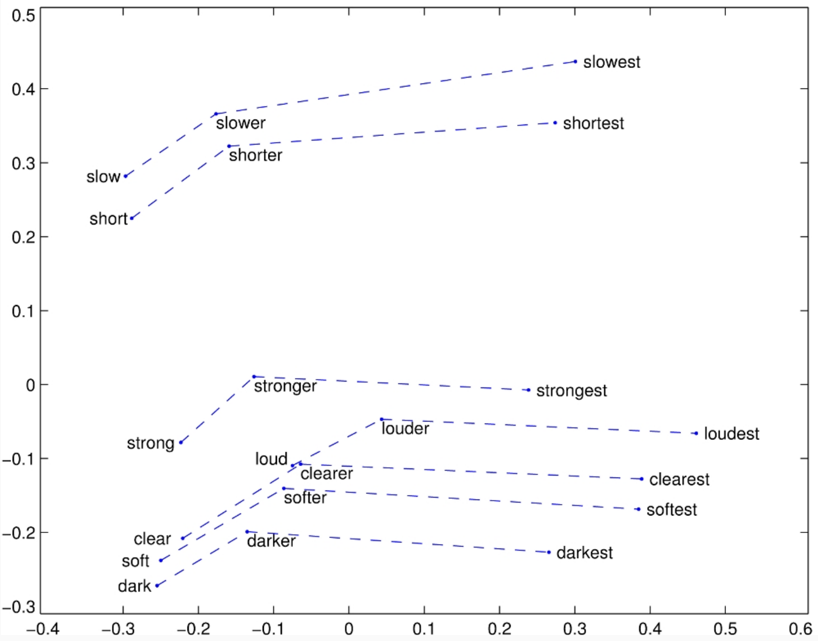
woman	is to	**sister**	as	**man**	is to	**brother**
summer	is to	**rain**	as	**winter**	is to	**snow**
man	is to	**king**	as	**woman**	is to	**queen**
fell	is to	**fallen**	as	**ate**	is to	**eaten**
running	is to	**ran**	as	**crying**	is to	**cried**

The **differences** between each **pair of words** are similar.

Can our word representations capture this? (demo)



0.4
0.3
0.2
0.1
0
−0.1
−0.2
−0.3
−0.6
−0.4
−0.2
0
0.2
0.4
0.6
11226
Brooklyn
14240
Buffalo
60629
Chicago
33186
Miami
77084
30349
Houston
Atlanta
96817
Honolulu
37211
Nashville
95823
Sacramento
92804
Anaheim


slowest
slower
shortest
shorter
slow
short
stronger
strongest
louder
loudest
strong
loud
clearer
clearest
softer
softest
clear
darker
darkest
soft
dark
0.5
0.4
0.3
0.2
0.1
0
−0.1
−0.2
−0.3
−0.4
−0.3
−0.2
−0.1
0
0.1
0.2
0.3
0.4
0.5
0.6

Neural Word Embeddings

Associate a low-dimensional, dense vector $\vec{w}$ with each word $w \in V$ so that similar words (in a distributional sense) share a similar vector representation.

If I only had a vector: The Analogy Problem

To solve analogy problems of the form "w_a is to w_b as w_c is to what?", we can simply compute a query vector

$$\mathbf{q} = \mathbf{w}_b - \mathbf{w}_a + \mathbf{w}_c$$

and find the most similar word vector $\mathbf{v} \in \mathbf{W}$ to $\mathbf{q}$. If we normalize $\mathbf{q}$ to unit-length

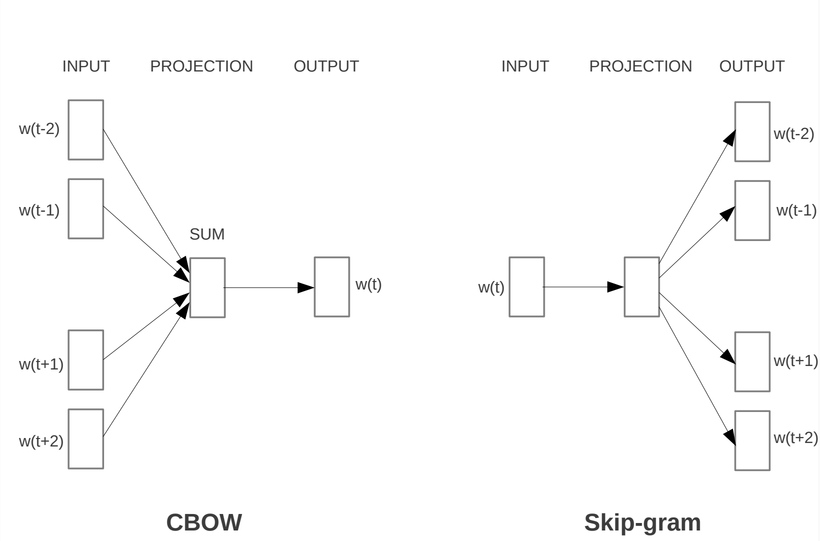
$$\hat{\mathbf{q}} = \frac{\mathbf{q}}{||\mathbf{q}||}$$

and assume each vector in $\mathbf{W}$ is also unit-length, this reduces to computing

$$\arg\max_{\mathbf{v} \in \mathbf{W}} \mathbf{v} \cdot \hat{\mathbf{q}}$$

and returning the associated word v.

The CBOW and Skip-Gram Models[4] (word2vec)



[4] Tomas Mikolov et al. "Efficient estimation of word representations in vector space". In: *ICLR Workshop* (2013).

Skip-Gram, Mathematically

Training corpus $w_1, w_2, \ldots, w_N$ (with N typically in the billions) from a fixed vocabulary V.

Goal is to maximize the average log probability:

$$\frac{1}{N} \sum_{i=1}^{N} \sum_{-L \leq k \leq L; k \neq 0} \log p(w_{i+k} \mid w_i)$$

Associate with each $w \in V$ an "input vector" $\mathbf{w} \in \mathbb{R}^d$ and an "output vector" $\tilde{\mathbf{w}} \in \mathbb{R}^d$. Model context probabilities as

$$p(c \mid w) = \frac{\exp(\mathbf{w} \cdot \tilde{\mathbf{c}})}{\sum_{c' \in V} \exp(\mathbf{w} \cdot \tilde{\mathbf{c}}')}.$$

The problem? V is *huge*! $\nabla \log p(c \mid w)$ takes time $O(|V|)$ to compute!

Negative Sampling[5]

Given a pair (w, c), can we determine if this came from our corpus or not? Model probabilistically as

$$p(D = 1 \mid w, c) = \sigma(\mathbf{w} \cdot \tilde{\mathbf{c}}) = \frac{1}{1 + \exp(-\mathbf{w} \cdot \tilde{\mathbf{c}})}.$$

Goal: maximize $p(D = 1 \mid w, c)$ for pairs (w, c) that **occur in the data**.

Also maximize $p(D = 0 \mid w, c_N)$ for (w, c_N) pairs where c_N is **sampled randomly** from the empirical unigram distribution.

[5] Tomas Mikolov et al. "Distributed Representations of Words and Phrases and their Compositionality". In: *Advances in Neural Information Processing Systems 26*. 2013, pp. 3111–3119.

Skip-Gram Negative Sampling Objective

Locally,

$$\ell = \log \sigma(\mathbf{w} \cdot \tilde{\mathbf{c}}) + k \cdot \mathbb{E}_{c_N \sim p_D(c_N)} \left[\log \sigma(-\mathbf{w} \cdot \tilde{\mathbf{c_N}})\right]$$

and thus globally

$$\mathcal{L} = \sum_{w \in V} \sum_{c \in V} (n_{w,c}) \left(\log \sigma(\mathbf{w} \cdot \tilde{\mathbf{c}}) + k \cdot \mathbb{E}_{c_N \sim P_n(c_N)} \left[\log \sigma(-\mathbf{w} \cdot \tilde{\mathbf{c_N}})\right]\right)$$

- k: number of negative samples to take (hyperparameter)
- $n_{w,c}$: number of times (w, c) was seen in the data
- $\mathbb{E}_{c_N \sim P_n(c_N)}$ indicates an expectation taken with respect to the noise distribution $P_n(c_N)$.

Implementation Details (Read: Useful Heuristics)

1. What is the noise distribution?

$$P_n(c_N) = \frac{(n_{c_N}/N)^{3/4}}{Z}$$

   (which is the empirical unigram distribution raised to the 3/4 power).

2. Frequent words can dominate the loss. Throw away word w_i in the training data according to

$$P(w_i) = 1 - \sqrt{\frac{t}{n_{w_i}}}$$

   where t is some threshold (like 10^{-5}).

Both are essentially unexplained by Mikolov et al.

What does this *actually do*?

It turns out this is nothing that new! Levy and Goldberg[6] show that SGNS is **implicitly factorizing** the matrix

$$M_{i,j} = \mathbf{w_i} \cdot \tilde{\mathbf{c}}_\mathbf{j} = PMI(w_i, c_j) - \log k = \log \frac{p(w_i, c_j)}{p(w_i)p(c_j)} - \log k$$

using an objective that weighs deviations in more frequent (w, c) pairs more strongly than less frequent ones.

Thus...

[6] Omer Levy and Yoav Goldberg. “Neural Word Embedding as Implicit Matrix Factorization”. In: *Advances in Neural Information Processing Systems 27*. 2014, pp. 2177–2185.

Matrix Factorization Methods for Word Embeddings

Why not just SVD?

Since SGNS is just factorizing a shifted version of the PMI matrix, **why not just perform a rank-k SVD** on the PMI matrix directly?

Assignment 6!

With a little TLC, this method can actually work and does surprisingly well.

GloVe: Global Vectors for Word Representation[7]

How can we capture words related to **ice**, but not to **steam**?

Prob. or Ratio	w_k = **solid**	w_k = **gas**	w_k = **water**	w_k = **fashion**
$P(w_k \mid \text{ice})$	1.9×10^{-4}	6.6×10^{-5}	3.0×10^{-3}	1.7×10^{-5}
$P(w_k \mid \text{steam})$	2.2×10^{-5}	7.8×10^{-4}	2.2×10^{-3}	1.8×10^{-5}
$\frac{P(w_k \mid \text{ice})}{P(w_k \mid \text{steam})}$	8.9	8.5×10^{-2}	1.36	0.96

Probability ratios are most informative:

- **solid** is related to **ice** but not **steam**
- **gas** is related to **steam** but not **ice**
- **water** and **fashion** do not discriminate between **ice** or **steam** (ratios close to 1)

[7] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. "GloVe: Global Vectors for Word Representation". In: *EMNLP*. 2014, pp. 1532–1543.

Building a model: intuition

We would like for the vectors $\mathbf{w}_i$, $\mathbf{w}_j$, and $\mathbf{w}_k$ to be able to capture the information present in the probability ratio. If we set

$$\mathbf{w}_i^\intercal \mathbf{w}_k = \log P(w_k \mid w_i)$$

then we can easily see that

$$(\mathbf{w}_i - \mathbf{w}_j)^\intercal \mathbf{w}_k = \log P(w_k \mid w_i) - \log P(w_k \mid w_j) = \log \frac{P(w_k \mid w_i)}{P(w_k \mid w_j)},$$

ensuring that the information present in the co-occurrence probability ratio is expressed in the vector space.

GloVe Objective

$$\mathbf{w}_i^\intercal \tilde{\mathbf{w}}_j + b_i + \tilde{b}_j = \log X_{ij}$$

Again: two sets of vectors: "target" vectors $\mathbf{w}$ and "context" vectors $\tilde{\mathbf{w}}$.

$X_{ij} = n_{w_i, w_j}$ is the number of times w_j appears in the context of w_i.

Problem: all co-occurrences are weighted equally

$$J = \sum_{i,j=1}^{V} f(X_{ij})(\mathbf{w}_i^\intercal \tilde{\mathbf{w}}_j + b_i + \tilde{b}_j - \log X_{ij})^2$$

Final objective is just **weighted least-squares**. $f(X_{ij})$ serves as a "dampener", lessening the weight of the rare co-occurrences.

$$f(x) = \begin{cases} \left(\frac{x}{x_{max}}\right)^\alpha & \text{if } x < x_{max} \\ 1 & \text{otherwise} \end{cases}$$

where α's default is (you guessed it) 3/4, and x_{max}'s default is 100.

Relation to word2vec

GloVe Objective:

$$J = \sum_{i,j=1}^{V} f(X_{ij})(\mathbf{w}_i^\intercal \tilde{\mathbf{w}}_j + b_i + \tilde{b}_j - \log X_{ij})^2$$

word2vec (Skip-Gram) Objective (after rewriting):

$$J = -\sum_{i=1}^{V} X_i \sum_{j=1}^{V} P(w_j \mid w_i) \log Q(w_j \mid w_i)$$

where $X_i = \sum_k X_{ik}$ and P and Q are the empirical co-occurrence and model co-occurrence distributions, respectively. **Weighted cross-entropy error!**

Authors show that **replacing cross-entropy with least-squares nearly re-derives GloVe**:

$$\hat{J} = \sum_{i,j} X_i (w_i^\intercal \tilde{w}_j - \log X_{ij})^2$$

Model Complexity

word2vec: $O(|C|)$, linear in corpus size

GloVe naïve estimate: $O(|V|^2)$, square of vocab size

But it actually only depends on the number of non-zero entries in $\mathbf{X}$. If co-occurrences are modeled via a power law, then we have

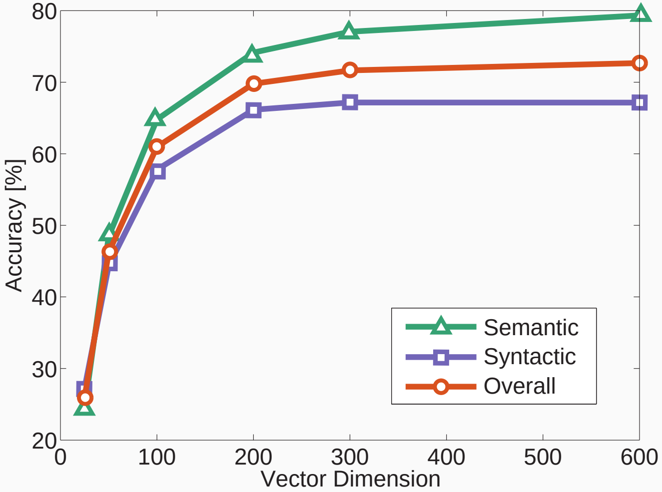
$$X_{ij} = \frac{k}{(r_{ij})^\alpha}.$$

Modeling $|C|$ under this assumption, the authors eventually arrive at

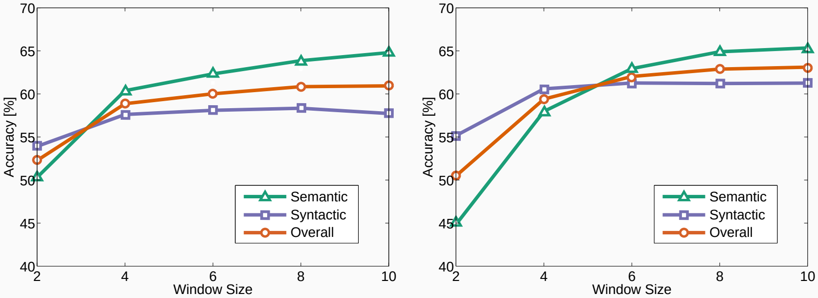
$$|\mathbf{X}| = \begin{cases} O(|C|) & \text{if } \alpha < 1 \\ O(|C|^{1/\alpha}) & \text{otherwise} \end{cases}$$

where the corpora studied in the paper were well modeled with $\alpha = 1.25$, leading to $O(|C|^{0.8})$. In practice, it's faster than word2vec.

Accuracy vs Vector Size

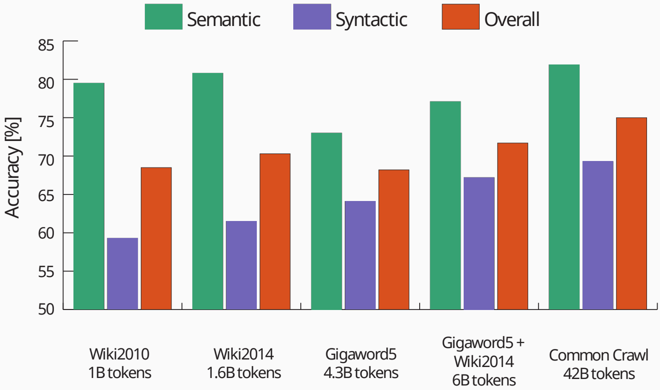


Accuracy vs Window Size

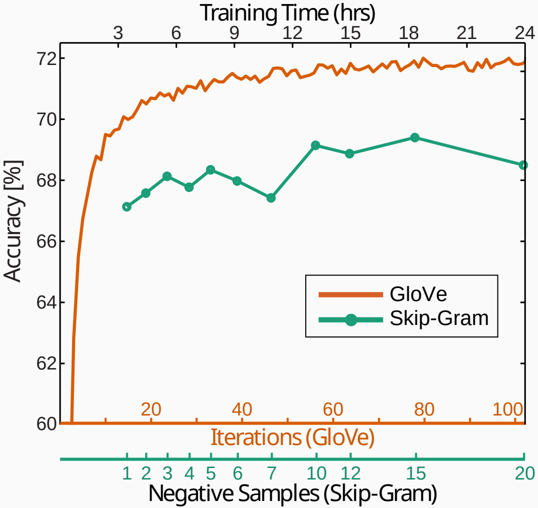


Left: Symmetric window, **Right:** Asymmetric window

Accuracy vs Corpus Choice



Training Time



This evaluation is **not quite fair**!

But wait…

(Referring to `word2vec`)

> *…the code is designed only for a single training epoch…*
> *…specifies a learning schedule specific to one pass through the data, making a modification for multiple passes a non-trivial task.*

But wait...

(Referring to `word2vec`)

> *...the code is designed only for a single training epoch...*
> *...specifies a learning schedule specific to one pass through the data, making a modification for multiple passes a non-trivial task.*

This is **false**, and a **bad excuse**.

How could you modify `word2vec` to support multiple “epochs” with zero effort?

But wait...

...we choose to use the sum $\mathbf{W} + \tilde{\mathbf{W}}$ *as our word vectors.*

But wait...

> *...we choose to use the sum* $\mathbf{W} + \tilde{\mathbf{W}}$ *as our word vectors.*

`word2vec` *also* learns $\tilde{\mathbf{W}}$! Why not do the same for both?

Embedding performance is very much a function of your hyperparameters!

Investigating Hyperparameters[8]

Levy, Goldberg, and Dagan perform a **systematic evaluation** of `word2vec`, SVD, and GloVe where **hyperparameters are controlled for** and **optimized across different methods** for a variety of tasks.

Interesting results:

> *MSR's analogy dataset is the only case where SGNS and GloVe substantially outperform PPMI and SVD.*

And:

> *...SGNS outperforms GloVe in **every** task.*

[8] Omer Levy, Yoav Goldberg, and Ido Dagan. "Improving Distributional Similarity with Lessons Learned from Word Embeddings". In: *Transactions of the Association for Computational Linguistics* 3 (2015), pp. 211–225.

But wait... (mark II)

Both GloVe and `word2vec` use **context window weighting**:

- `word2vec`: **samples a window size** $\ell \in [1, L]$ for each token before extracting counts
- GloVe: weighs contexts by their **distance from the target word** using the harmonic function $\frac{1}{d}$

What is the impact of using the different context window strategies between methods?

Levy, Goldberg, and Dagan **did not** investigate this, instead using `word2vec`'s method across all of their evaluations.

Evaluating things properly is non-obvious and often very difficult!

What You Should Know

What You Should Know

- What is the **sparsity problem** in NLP?
- What is the **distributional hypothesis**?
- How does **Brown clustering** work? What is it trying to maximize?
- What is **agglomerative clustering**?
- What is a **word embedding**?
- What is the difference between the **CBOW** and **Skip-Gram** models?
- What is **negative sampling** and why is it useful?
- What is the **learning objective** for **SGNS**?
- What is the matrix that **SGNS is implicitly factorizing**?
- What is the **learning objective** for **GloVe**?
- What are some examples of **hyperparameters for word embedding methods**?

Implementations

- `word2vec`:
  - The **original implementation**: `https://code.google.com/archive/p/word2vec/`
  - Yoav Goldberg's `word2vecf` **modification (multiple epochs, arbitrary context features)**: `https://bitbucket.org/yoavgo/word2vecf`
  - **Python implementation** in `gensim`: `https://radimrehurek.com/gensim/models/word2vec.html`
- GloVe:
  - The **original implementation**: `http://nlp.stanford.edu/projects/glove/`
  - MeTA: `https://meta-toolkit.org`
  - **R implementation** `text2vec`: `http://text2vec.org/`

Thanks!