Neural Word Embedding as Implicit Matrix Factorization
Levy & Goldberg, 2014

Geneviève Chafouleas & David Ferland

March 23, 2020

Intro

This paper shows that the objective function of the Word2Vec Skip-gram with negative sampling (SGNS) is an implicit weighted matrix factorization of a shifted PMI matrix.

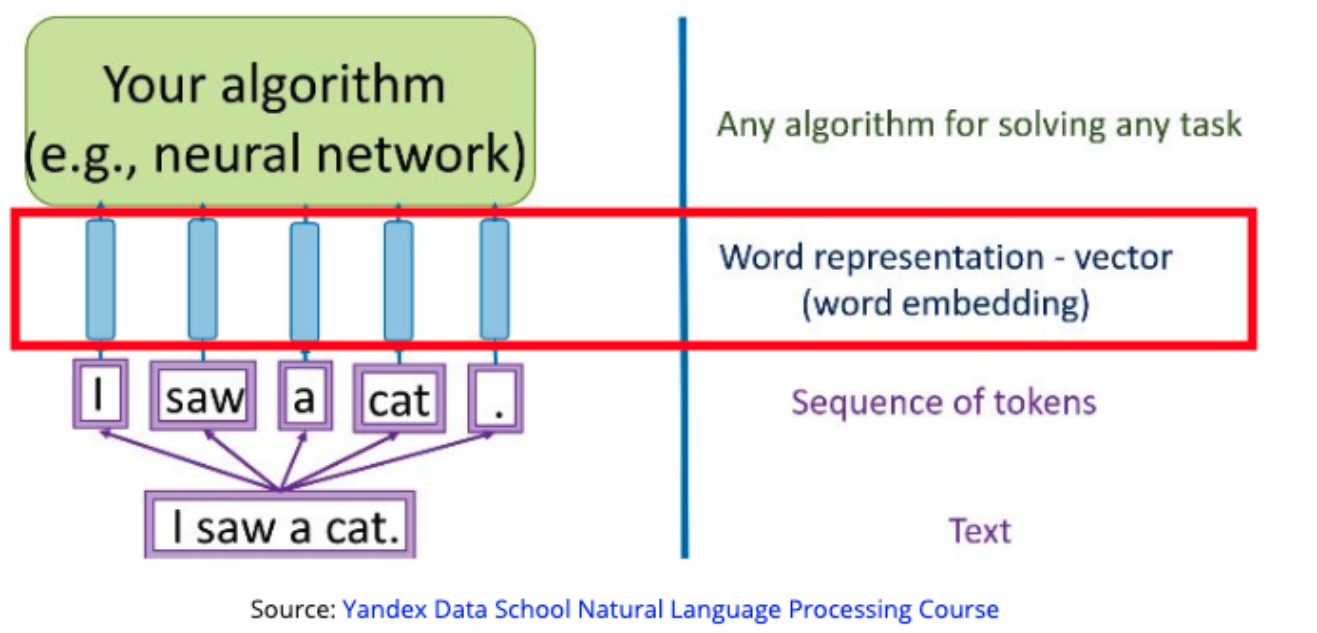
They propose using SVD decomposition of the shifted PPMI matrix as an alternative word embedding technique.

Outline

- Context and Motivation
- Word-context Matrix
- Review Word2Vec Skip-gram with negative sampling(SGNS)
- Implicit matrix factorization
- Proposed Alternative Word representations
- Empirical Results

Context - Word Representations

- NLP/NLU tasks generally require a **word representation**
- String token $=>$ numeric vector

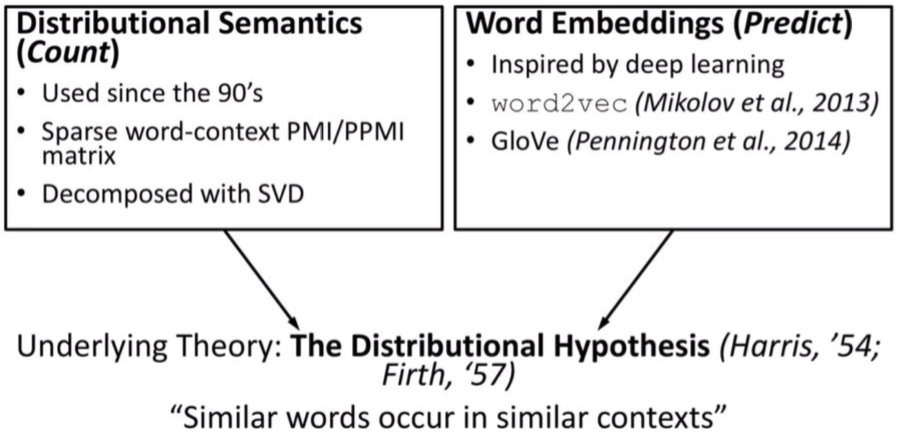


Source: Yandex Data School Natural Language Processing Course

Context - Distributional Hypothesis

- Simple representations treats individual words as unique symbols (e.g. one-hot encoding, bag of words) => **do not consider context**

- But many tasks benefit from capturing semantic or meaning-related relationship between words is key => **consider context**
  - Common paradigm: The Distributional Hypothesis (Harris, Firth)
  - "You shall know a word by the company it keeps" (Firth)



Distributed word representations

Count-based

- Based on matrix $\mathbf{M} \in \mathbb{R}^{|V_w| \times |V_c|}$
- Rows are sparse vectors
- **PMI (point-mutual information)**
- PPMI (positive PMI)

Prediction-based (neural/word embedding)

- Learned $\mathbf{W} \in \mathbb{R}^{|V_w| \times d}$, $\mathbf{C} \in \mathbb{R}^{|V_c| \times d}$
- Rows are dense vectors
- word2vec: CBOW, Skip-Gram
  - Skip-gram Negative Sampling **(SGNS)**

Main goal

Show that SGNS can be cast as a **weighted factorization of the shifted PMI matrix**

Distributed word representations

Count-based

- Based on matrix $\mathbf{M} \in \mathbb{R}^{|V_w| \times |V_c|}$
- Rows are sparse vectors
- **PMI (point-mutual information)**
- PPMI (positive PMI)

Prediction-based (neural/word embedding)

- Learned $\mathbf{W} \in \mathbb{R}^{|V_w| \times d}$, $\mathbf{C} \in \mathbb{R}^{|V_c| \times d}$
- Rows are dense vectors
- word2vec: CBOW, Skip-Gram
- SGNS

PMI Matrix

- Word-Context matrix: $\mathbf{M} \in \mathbb{R}^{|V_w| \times |V_c|}$
  - row_i: $w_i \in V_w$
  - $column_j$: $c_j \in V_c$
  - $\mathbf{M}_{i,j} = f(w_i, c_j)$: measure of association
- **Co-occurrence** matrix: $f(w, c) = P(w, c)$
- **Pointwise Mutual Information** (PMI) matrix:

$$f(w, c) = PMI(w, c) = \log\left(\frac{P(w, c)}{P(w)P(c)}\right)$$

Intuition on PMI

How much more/less likely is the co-occurrence of (w, c) than observing them independently.

(P)PMI Matrix

For $w \in V_W$ and $c \in V_C$ and (w, c) word-context pairs observed in D.

- **Empirical PMI**:

$$P(w,c) = \frac{\#(w,c)}{|D|},\ P(w) = \frac{\#(w)}{|D|},\ P(c) = \frac{\#(c)}{|D|}$$

$$PMI(w,c) = \log\left(\frac{\#(w,c) \cdot |D|}{\#(w) \cdot \#(c))}\right)$$

  - Issue for unseen (w,c) pairs:

$$PMI(w,c) = \log 0 = -\infty$$

- **Alternative: PPMI**

$$PPMI(w,c) = max(PMI(w,c), 0)$$

Distributed word representations

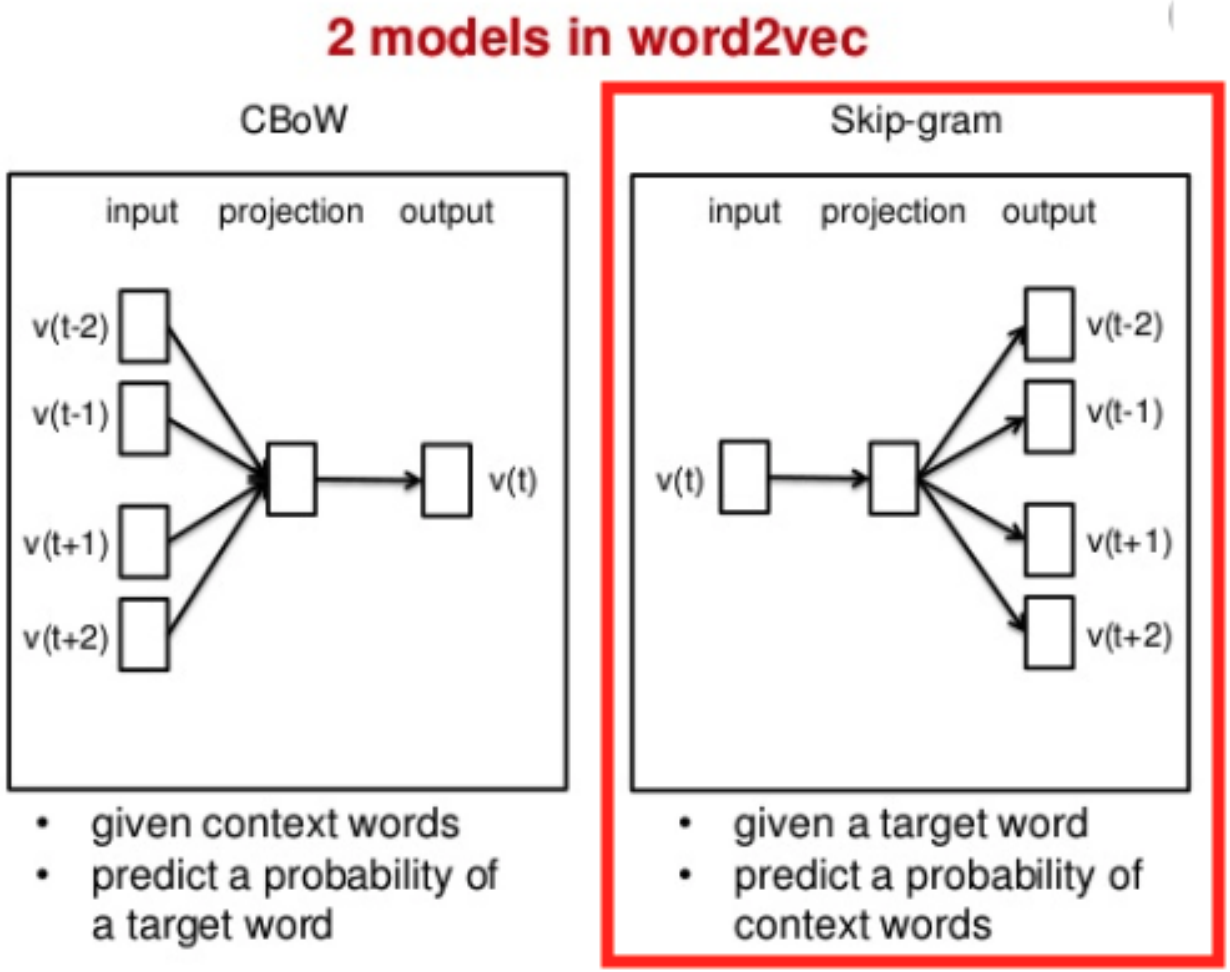
Count-based

- Based on matrix $\mathbf{M} \in \mathbb{R}^{|V_w| \times |V_c|}$
- Rows are sparse vectors
- PMI (point-mutual information)
- PPMI (positive PMI)

Prediction-based (neural/word embedding)

- Learned $\mathbf{W} \in \mathbb{R}^{|V_w| \times d}$, $\mathbf{C} \in \mathbb{R}^{|V_c| \times d}$
- Rows are dense vectors
- word2vec: CBOW, Skip-Gram
- **SGNS**

Word2Vec



- given context words
- predict a probability of a target word

- given a target word
- predict a probability of context words

Word2Vec - Skip-Gram Notation

Notation:

- D $\equiv$ collection of observed (w,c) pairs
- Each $w \in V_W$ is associated with a vector $\vec{w} \in \mathbb{R}^d$
- Each $c \in V_C$ is associated with a vector $\vec{c} \in \mathbb{R}^d$
- Expressing these vectors as matrices: $\mathbf{W} \in \mathbb{R}^{|V_w| \times d}$, $\mathbf{C} \in \mathbb{R}^{|V_C| \times d}$
  - $V_c = V_w$
- Output layer: Hierarchical Softmax or Negative Sampling

Skip-Gram Negative Sampling (SGNS)

- **Softmax:**
  - For each context word c_i to predict, we have:

$$p(c_i|w_{center}) = \frac{\exp(\vec{c_i} \cdot \vec{w}_{center})}{\sum_{j=1}^{|V_c|} \exp(\vec{c_j} \cdot \vec{w}_{center})}$$

  - Costly to train due to large $|V_c|$ (must update all voc. weights)

- **Alternative:** Skip-Gram with **Negative Sampling**
  - For each training sample: 1 positive and k random negative samples
  - k+1 binary classifications using Logistic Regression

$$\Rightarrow \text{ Only k+1 weight updates for each training sample}$$

Word2Vec - SGNS Objective

$P_{D|w,c}(w, c)$ modeled as:

- $P(D = 1|w, c) = \sigma(\vec{w} \cdot \vec{c}) = \frac{\exp(\vec{w} \cdot \vec{c})}{1+\exp(\vec{w} \cdot \vec{c})}$
- $P(D = 0|w, c) = 1 - \sigma(\vec{w} \cdot \vec{c}) = \sigma(-\vec{w} \cdot \vec{c})$

SGNS objective for a given (w, c) pair

$$\log \sigma(\vec{w} \cdot \vec{c}) + k \cdot \mathbf{E}_{c_N \sim P_D}[\log \sigma(-\vec{w} \cdot \vec{c_N})]$$

where c_N is drawn from $P_D(c) = \frac{\#(c)}{|D|}$.

$$tot.loss = l = \sum_{(w,c) \in D} \#(w, c)(\log \sigma(\vec{w} \cdot \vec{c}) + k \cdot \mathbf{E}_{c_N \sim P_D}[\log \sigma(-\vec{w} \cdot \vec{c_N})]) \quad (1)$$

SGNS as Implicit Matrix Factoriztion

- SGNS embeds words and contexts into matrices $\mathbf{W}$ and $\mathbf{C}$
- Consider $\mathbf{M} = \mathbf{W} \cdot \mathbf{C}^T$
- $\mathbf{M}_{ij} = \vec{w_i} \cdot \vec{c_j}$
  - represents an *implicit* association measure $f(w_i, c_j)$

What is the matrix M that Word2vec implicitly factorizes?

Characterizing the Implicit Matrix

$$tot.loss = \sum_{(w,c)\in D} \#(w,c)(\log \sigma(\vec{w}\cdot\vec{c}) + k \cdot \mathbf{E}_{c_N\sim P_D}[\log \sigma(-\vec{w}\cdot\vec{c_N})])$$

For a specific (w, c) pair:

$$l(w,c) = \underbrace{\#(w,c)}_{\text{positive obs. weight}} \log \sigma(\vec{w}\cdot\vec{c}) + \underbrace{k\cdot\#(w)\cdot\frac{\#(c)}{|D|}}_{\text{negative obs. weight}} \log \sigma(-\vec{w}\cdot\vec{c})$$

We take the derivative and solve for $\vec{w}\cdot\vec{c}$:

$$\vec{w}\cdot\vec{c} = \log\left(\frac{\#(w,c)\cdot|D|}{\#(w)\cdot\#(c)}\cdot\frac{1}{k}\right) = \log\left(\frac{\#(w,c)\cdot|D|}{\#(w)\cdot\#(c)}\right) - \log(k)$$

SGNS is factorizing implicitly:

$$\mathbf{M}_{ij}^{SGNS} = \vec{w}_i\cdot\vec{c}_j = PMI(w_i,c_j) - \log k$$

Alternative Word Representation

Shifted PPMI

Shifted PPMI

$$M^{SPPMI_k} = SPPMI_k(w, c) = max(PMI(w, c) - \log k, 0)$$

- where k is a hyperparameter

- Solves the issue of having cell value equal to $\log(0) = -\infty$
- M^{SPPMI_k} is a a sparse matrix, can apply SVD efficiently.

SVD over Shifted PPMI

Truncated SVD

Given a matrix $\mathbf{M}$, we have $\mathbf{M}_d = \mathbf{U}_d \cdot \mathbf{\Sigma}_d \cdot \mathbf{V}_d^T$

- $\mathbf{M_d}$ that best approximates $\mathbf{M}$ under L_2.

$$\mathbf{M}_d = argmin_{Rank(\mathbf{M}')=d} ||\mathbf{M}' - \mathbf{M}||_2$$

A popular approach in NLP is factorizing $\mathbf{M}^{PPMI}$ with SVD:

$$\mathbf{W}^{SVD} = \mathbf{U}_d \cdot \mathbf{\Sigma}_d, \mathbf{C}^{SVD} = \mathbf{V}_d$$

Symetric SVD of $\mathbf{M}^{SPPMI}$

$$\mathbf{W}^{SVD_{1/2}} = \mathbf{U}_d \cdot \sqrt{\mathbf{\Sigma}_d}, \mathbf{C}^{SVD_{1/2}} = \mathbf{V}_d \cdot \sqrt{\mathbf{\Sigma}_d}$$

SVD versus SGNS

SVD over shifted PPMI matrix

Advantages

- No hyperparameter tuning.
- easily applied on count-agg. data (i.e $\{(w, c, (w, c))\}$).
- More efficient for large corpas.

Disadvantages

- Un-weighted L2 loss when solving for best SVD, objective does not distinguish between unobserved and observed pairs.
- Must define arbitrarily $\mathbf{W}$ from the decomposed matrices

SGNS

Advantages

- The objective weights different (w, c) pairs differently.
- Trained over observed pairs and learns embedding $\mathbf{W}$ directly

Disadvantages

- Requires hyperparameter tuning.
- Requires each observation (w, c) to be presented separately in training.

Experimental Setup

- Trained on English Wikepedia.
- Trained SGNS models and word representation alternatives.

Optimizing the Objective

Method	PMI$-\log k$	SPPMI	SVD			SGNS		
			$d = 100$	$d = 500$	$d = 1000$	$d = 100$	$d = 500$	$d = 1000$
$k = 1$	0%	0.00009%	26.1%	25.2%	24.2%	31.4%	29.4%	7.40%
$k = 5$	0%	0.00004%	95.8%	95.1%	94.9%	39.3%	36.0%	7.13%
$k = 15$	0%	0.00002%	266%	266%	265%	7.80%	6.37%	5.97%

Table 1: Percentage of deviation from the optimal objective value (lower values are better). See 5.1 for details.

- Deviation is calculated $\left(\frac{\ell - \ell_{opt}}{\ell_{opt}}\right)$
- Optimal objective: $PMI - \log k$

Performance of Word Representations on Linguistic Tasks

WS353 (WordSim) [13]			MEN (WordSim) [4]			Mixed Analogies [20]			Synt. Analogies [22]		
Representation		Corr.	Representation		Corr.	Representation		Acc.	Representation		Acc.
SVD	(k=5)	0.691	SVD	(k=1)	0.735	SPPMI	(k=1)	0.655	SGNS	(k=15)	0.627
SPPMI	(k=15)	0.687	SVD	(k=5)	0.734	SPPMI	(k=5)	0.644	SGNS	(k=5)	0.619
SPPMI	(k=5)	0.670	SPPMI	(k=5)	0.721	SGNS	(k=15)	0.619	SGNS	(k=1)	0.59
SGNS	(k=15)	0.666	SPPMI	(k=15)	0.719	SGNS	(k=5)	0.616	SPPMI	(k=5)	0.466
SVD	(k=15)	0.661	SGNS	(k=15)	0.716	SPPMI	(k=15)	0.571	SVD	(k=1)	0.448
SVD	(k=1)	0.652	SGNS	(k=5)	0.708	SVD	(k=1)	0.567	SPPMI	(k=1)	0.445
SGNS	(k=5)	0.644	SVD	(k=15)	0.694	SGNS	(k=1)	0.540	SPPMI	(k=15)	0.353
SGNS	(k=1)	0.633	SGNS	(k=1)	0.690	SVD	(k=5)	0.472	SVD	(k=5)	0.337
SPPMI	(k=1)	0.605	SPPMI	(k=1)	0.688	SVD	(k=15)	0.341	SVD	(k=15)	0.208

Table 2: A comparison of word representations on various linguistic tasks. The different representations were created by three algorithms (SPPMI, SVD, SGNS) with $d = 1000$ and different values of k.

Conclusion

- SGNS implicitly factorizing the (shifted) word-context PMI matrix.
- Presented SPPMI as word representation.
- Presentated matrix factorization of SPPMI as word representation.

References

[1] https://papers.nips.cc/paper/5477-neural-word-embedding-as-implicit-matrix-factorization.pdf
[2] https://medium.com/radix-ai-blog/unifying-word-embeddings-and-matrix-factorization-part-1-cb3984e95141
[3] https://medium.com/radix-ai-blog/unifying-word-embeddings-and-matrix-factorization-part-2-a0174ace78b8
[4] https://medium.com/radix-ai-blog/unifying-word-embeddings-and-matrix-factorization-part-3-4269d9a07470

The End