Improving Distributional Similarity with Lessons Learned from Word Embeddings

Authors: Omar Levy, Yoav Goldberg, Ido Dagan

Presentation: Collin Gress

Motivation

- We want to do NLP tasks. How do we represent words?

Motivation

- We want to do NLP tasks. How do we represent words?
- We generally want vectors. Think neural networks

Motivation

- We want to do NLP tasks. How do we represent words?
- We generally want vectors. Think neural networks
- What are some ways to get vector representations of words?

Motivation

- We want to do NLP tasks. How do we represent words?
- We generally want vectors. Think neural networks
- What are some ways to get vector representations of words?
- Distributional hypothesis: "words that are used and occur in the same contexts tend to purport similar meanings" - Wikipedia

Vector representations of words and their surrounding contexts

- Word2vec [1]

Vector representations of words and their surrounding contexts

- Word2vec [1]
- Glove [2]

Vector representations of words and their surrounding contexts

- Word2vec [1]
- Glove [2]
- PMI – Pointwise mutual Information

Vector representations of words and their surrounding contexts

- Word2vec [1]
- Glove [2]
- PMI – Pointwise mutual information
- SVD of PMI – Singular value decomposition of PMI matrix

Very briefly: Word2vec

- This paper focuses on Skip-Gram negative sampling (SGNS) which predicts context words based off of a target word

Very briefly: Word2vec

- This paper focuses on Skip-Gram negative sampling (SGNS) which predicts context words based off of a target word
- Optimization problem solvable by gradient descent. Want to maximize $\vec{w} \cdot \vec{c}$ for word context pairs that exist in the dataset, and minimize it for "hallucinated" word context pairs [0].

Very briefly: Word2vec

- This paper focuses on Skip-Gram negative sampling (SGNS) which predicts context words based off of a target word
- Optimization problem solvable by gradient descent. Want to maximize $\vec{w} \cdot \vec{c}$ for word context pairs that exist in the dataset, and minimize it for "hallucinated" word context pairs [0].
- For every real word-context pair in dataset, hallucinate k word-context pairs. That is, given some word target, draw k contexts from $p_D(c) = \frac{count(c)}{\sum_{c'} count(c')}$

Very briefly: Word2vec

- This paper focuses on Skip-Gram negative sampling (SGNS) which predicts context words based off of a target word
- Optimization problem solvable by gradient descent. Want to maximize $\vec{w} \cdot \vec{c}$ for word context pairs that exist in the dataset, and minimize it for "hallucinated" word context pairs [0].
- For every real word-context pair in dataset, hallucinate k word-context pairs. That is, given some word target, draw k contexts from $p_D(c) = \frac{count(c)}{\sum_{c'} count(c')}$
- End up with a set of vectors, $\vec{w}_i \in \mathbb{R}^d$ for every word in dataset. Similarly, set of vectors, $\vec{c}_i \in \mathbb{R}^d$ for each context in the dataset.

See Mikolov paper [1] for details

Very briefly: Glove

- Learn d-dimensional vectors $\vec{w}$ and $\vec{c}$ as well as word and context specific scalars, b_w and b_c such that $\vec{w} \cdot \vec{c} + b_w + b_c = \log(\text{count}(w, c))$ for all word context pairs in data set [0]

Very briefly: Glove

- Learn d-dimensional vectors $\vec{w}$ and $\vec{c}$ as well as word and context specific scalars, b_w and b_c such that $\vec{w} \cdot \vec{c} + b_w + b_c = \log(\text{count}(w, c))$ for all word context pairs in data set [0]
- Objective "solved" by factorization of the log count matrix, $M^{\log(count(w,c))}$, $W \cdot C^T + b^{\vec{w}} + b^{\vec{c}}$

Very briefly: Pointwise mutual information (PMI)

- $PMI(w, c) = \log(\frac{p(w,c)}{p(w)p(c)})$

PMI: example

word 1	word 2	count word 1	count word 2	count of co-occurrences	PMI
puerto	rico	1938	1311	1159	10.0349081703
hong	kong	2438	2694	2205	9.72831972408
los	angeles	3501	2808	2791	9.56067615065
carbon	dioxide	4265	1353	1032	9.09852946116
prize	laureate	5131	1676	1210	8.85870710982
san	francisco	5237	2477	1779	8.83305176711
nobel	prize	4098	5131	2498	8.68948811416
ice	hockey	5607	3002	1933	8.6555759741
star	trek	8264	1594	1489	8.63974676575
car	driver	5578	2749	1384	8.41470768304
it	the	283891	3293296	3347	-1.72037278119
are	of	234458	1761436	1019	-2.09254205335

Source: https://en.wikipedia.org/wiki/Pointwise_mutual_information

PMI matrices for word, context pairs in practice

- Very sparse

Interesting relationships between PMI and SGNS; PMI and Glove

- SGNS is implicitly factorizing PMI shifted by some constant [0]. Specifically, SGNS finds optimal vectors, $\vec{w}$ and $\vec{c}$, such that $\vec{w} \cdot \vec{c} = PMI(w, c) - \log(k)$.
  $W \cdot C^T = M - \log k$

Interesting relationships between PMI and SGNS; PMI and Glove

- SGNS is implicitly factorizing PMI shifted by some constant [0]. Specifically, SGNS finds optimal vectors, $\vec{w}$ and $\vec{c}$, such that $\vec{w} \cdot \vec{c} = PMI(w, c) - \log(k)$. $W \cdot C^T = M - \log k$
- Recall that, in Glove we learn vectors d-dimensional vectors $\vec{w}$ and $\vec{c}$ as well as word and context specific scalars, b_w and b_c such that $\vec{w} \cdot \vec{c} + b_w + b_c = \log(\text{count}(w, c))$ for all word context pairs in data set

Interesting relationships between PMI and SGNS; PMI and Glove

- SGNS is implicitly factorizing PMI shifted by some constant [0]. Specifically, SGNS finds optimal vectors, $\vec{w}$ and $\vec{c}$, such that $\vec{w} \cdot \vec{c} = PMI(w, c) - \log(k)$.
  $W \cdot C^T = M - \log k$
- Recall that, in Glove we learn vectors d-dimensional vectors $\vec{w}$ and $\vec{c}$ as well as word and context specific scalars, b_w and b_c such that
  $\vec{w} \cdot \vec{c} + b_w + b_c = \log(\text{count}(w, c))$ for all word context pairs in data set
- If we fix b_w and b_c such that $b_w = \log(count(w))$ and $b_c = \log(count(c))$, we get a problem nearly equivalent to factorizing PMI matrix shifted by $\log(|D|)$.
  I.e. $W \cdot C^T + b^{\vec{w}} + b^{\vec{c}} = M - \log(|D|)$

Interesting relationships between PMI and SGNS; PMI and Glove

- SGNS is implicitly factorizing PMI shifted by some constant [0]. Specifically, SGNS finds optimal vectors, $\vec{w}$ and $\vec{c}$, such that $\vec{w} \cdot \vec{c} = PMI(w, c) - \log(k)$. $W \cdot C^T = M - \log k$
- Recall that, in Glove we learn vectors d-dimensional vectors $\vec{w}$ and $\vec{c}$ as well as word and context specific scalars, b_w and b_c such that $\vec{w} \cdot \vec{c} + b_w + b_c = \log(\text{count}(w, c))$ for all word context pairs in data set
- If we fix b_w and b_c such that $b_w = \log(count(w))$ and $b_c = \log(count(c))$, we get a problem nearly equivalent to factorizing PMI matrix shifted by $\log(|D|)$. I.e. $W \cdot C^T + b^{\vec{w}} + b^{\vec{c}} = M - \log(|D|)$
- Or in simple terms, SGNS (Word2vec) and Glove aren't too different from PMI

Very briefly: SVD of PMI matrix

- Singular value decomposition of PMI gives us dense vectors

Very briefly: SVD of PMI matrix

- Singular value decomposition of PMI gives us dense vectors
- Factorize PMI matrix, M into product of three matrices i.e. $U \cdot \Sigma \cdot V^T$

Very briefly: SVD of PMI matrix

- Singular value decomposition of PMI gives us dense vectors
- Factorize PMI matrix, M into product of three matrices i.e. $U \cdot \Sigma \cdot V^T$
- Why does that help? $W = U_d \cdot \Sigma_d$ and $C = V_d$

Thesis

- The performance gains of word embeddings are mainly attributed to the optimization of algorithm hyperparameters by algorithm designers rather than the algorithms themselves

Thesis

- The performance gains of word embeddings are mainly attributed to the optimization of algorithm hyperparameters by algorithm designers rather than the algorithms themselves
- PMI and SVD baselines used for comparison in embedding papers were the most “vanilla” versions, hence the apparent superiority of embedding algorithms

Thesis

- The performance gains of word embeddings are mainly attributed to the optimization of algorithm hyperparameters by algorithm designers rather than the algorithms themselves
- PMI and SVD baselines used for comparison in embedding papers were the most “vanilla” versions, hence the apparent superiority of embedding algorithms
- Hyperparameters of Glove and Word2vec can be applied to PMI and SVD, drastically improving their performance

Pre-processing Hyperparameters

- Dynamic Context Window: context word counts are weighted by their distance to the target word. Word2vec does this by setting each context word's weight to $\frac{window\ size - distance + 1}{window\ size}$. Glove uses $\frac{1}{distance}$

Pre-processing Hyperparameters

- <u>Dynamic Context Window</u>: context word counts are weighted by their distance to the target word. Word2vec does this by setting each context word's weight to $\frac{window\ size - distance + 1}{window\ size}$. Glove uses $\frac{1}{distance}$
- <u>Subsampling</u>: remove very frequent words from corpus. Word2vec does this by removing words which are more frequent than some threshold t with probability $1 - \sqrt{\frac{t}{f}}$ where f is the corpus wide frequency of a word.

Pre-processing Hyperparameters

- Dynamic Context Window: context word counts are weighted by their distance to the target word. Word2vec does this by setting each context word's weight to $\frac{window\ size - distance + 1}{window\ size}$. Glove uses $\frac{1}{distance}$
- Subsampling: remove very frequent words from corpus. Word2vec does this by removing words which are more frequent than some threshold t with probability $1 - \sqrt{\frac{t}{f}}$ where f is the corpus wide frequency of a word.
- Subsampling can be "dirty" or "clean". In dirty subsampling, words are removed before word context pairs are formed. In clean subsampling, it is done after.

Pre-processing Hyperparameters

- Dynamic Context Window: context word counts are weighted by their distance to the target word. Word2vec does this by setting each context word's weight to $\frac{window\ size - distance + 1}{window\ size}$. Glove uses $\frac{1}{distance}$
- Subsampling: remove very frequent words from corpus. Word2vec does this by removing words which are more frequent than some threshold t with probability $1 - \sqrt{\frac{t}{f}}$ where f is the corpus wide frequency of a word.
- Subsampling can be "dirty" or "clean". In dirty subsampling, words are removed before word context pairs are formed. In clean subsampling, it is done after.
- Deleting Rare Words: exactly what you would expect. Negligible effect on performance

Association Metric Hyperparameters

- Shifted PMI: As previously discussed, SGNS implicitly factorizes the PMI matrix shifted by $\log(k)$. When working with PMI matrices, we can simply apply this transformation by picking some constant k, meaning each cell of the PMI matrix is $PMI(w, c) - \log(k)$

Association Metric Hyperparameters

- Shifted PMI: As previously discussed, SGNS implicitly factorizes the PMI matrix shifted by $\log(k)$. When working with PMI matrices, we can simply apply this transformation by picking some constant k, meaning each cell of the PMI matrix is $PMI(w,c) - \log(k)$
- Context Distribution Smoothing: used in Word2vec to smooth the context distribution for negative sampling. $p_D(c) = \frac{(count(c))^\alpha}{\sum_{c'}(count(c'))^\alpha}$ where α is some constant. Can be used in PMI in the same sort of way. $PMI_\alpha(w,c) = log \frac{p(w,c)}{p(w)p_\alpha(c)}$ where $p_\alpha(\mathrm{c}) = \frac{(count(c))^\alpha}{\sum_{c'}(count(c'))^\alpha}$

Association Metric Hyperparameters

- Shifted PMI: As previously discussed, SGNS implicitly factorizes the PMI matrix shifted by $\log(k)$. When working with PMI matrices, we can simply apply this transformation by picking some constant k, meaning each cell of the PMI matrix is $PMI(w, c) - \log(k)$
- Context Distribution Smoothing: used in Word2vec to smooth the context distribution for negative sampling. $p_D(c) = \frac{(count(c))^\alpha}{\sum_{c\prime}(count(c\prime))^\alpha}$ where α is some constant. Can be used in PMI in the same sort of way. $PMI_\alpha(w, c) = log\,\frac{p(w,c)}{p(w)p_\alpha(c)}$ where $p_\alpha(\mathrm{c}) = \frac{(count(c))^\alpha}{\sum_{c\prime}(count(c\prime))^\alpha}$
- Context distribution smoothing helps to correct PMI’s bias towards word context pairs where the context is rare

Post-processing Hyperparameters

- Adding Context Vectors: Glove paper [2] suggests that it is useful to return $\vec{w} + \vec{c}$ for word representations rather than just $\vec{w}$

Post-processing Hyperparameters

- Adding Context Vectors: Glove paper [2] suggests that it is useful to return $\vec{w} + \vec{c}$ for word representations rather than just $\vec{w}$
- New $\vec{w}$ holds more information. Previously, two vectors would have high cosine similarity if the two words are replaceable with one another. In the new form, weight is also awarded if the one word tends to appear in the context of the other [0].

Post-processing Hyperparameters

- Adding Context Vectors: Glove paper [2] suggests that it is useful to return $\vec{w} + \vec{c}$ for word representations rather than just $\vec{w}$
- New $\vec{w}$ holds more information. Previously, two vectors would have high cosine similarity if the two words are replaceable with one another. In the new form, weight is also awarded if the one word tends to appear in the context of the other [0].
- Eigenvalue Weighting: In SVD, weight the eigenvalue matrix by raising value to some power, p. Then define $W = U_d \cdot \Sigma_d^p$

Post-processing Hyperparameters

- Adding Context Vectors: Glove paper [2] suggests that it is useful to return $\vec{w} + \vec{c}$ for word representations rather than just $\vec{w}$
- New $\vec{w}$ holds more information. Previously, two vectors would have high cosine similarity if the two words are replaceable with one another. In the new form, weight is also awarded if the one word tends to appear in the context of the other [0].
- Eigenvalue Weighting: In SVD, weight the eigenvalue matrix by raising value to some power, p. Then define $W = U_d \cdot \Sigma_d^p$
- Authors observed that SGNS results in ”symmetric” weight and context matrices. I.e. neither is orthonormal and no bias given to either in training objective [0]. Symmetry obtainable in SVD by letting $p = \frac{1}{2}$

Post-processing Hyperparameters

- Adding Context Vectors: Glove paper [2] suggests that it is useful to return $\vec{w} + \vec{c}$ for word representations rather than just $\vec{w}$
- New $\vec{w}$ holds more information. Previously, two vectors would have high cosine similarity if the two words are replaceable with one another. In the new form, weight is also awarded if the one word tends to appear in the context of the other [0].
- Eigenvalue Weighting: In SVD, weight the eigenvalue matrix by raising value to some power, p. Then define $W = U_d \cdot \Sigma_d^p$
- Authors observed that SGNS results in ”symmetric” weight and context matrices. I.e. neither is orthonormal and no bias given to either in training objective [0]. Symmetry obtainable in SVD by letting $p = \frac{1}{2}$

Post-processing Hyperparameters

- Adding Context Vectors: Glove paper [2] suggests that it is useful to return $\vec{w} + \vec{c}$ for word representations rather than just $\vec{w}$
- New $\vec{w}$ holds more information. Previously, two vectors would have high cosine similarity if the two words are replaceable with one another. In the new form, weight is also awarded if the one word tends to appear in the context of the other [0].
- Eigenvalue Weighting: In SVD, weight the eigenvalue matrix by raising value to some power, p. Then define $W = U_d \cdot \Sigma_d^p$
- Authors observed that SGNS results in ”symmetric” weight and context matrices. I.e. neither is orthonormal and no bias given to either in training objective [0]. Symmetry obtainable in SVD by letting $p = \frac{1}{2}$
- Vector Normalization: general assumption is to normalize word vectors with L_2 normalization, may be worthwhile to experiment with.

Experiments Setup: Hyperparameter Space

Hyper-parameter	Explored Values	Applicable Methods
win	2, 5, 10	All
dyn	none, with	All
sub	none, dirty, clean†	All
del	none, with†	All
neg	1, 5, 15	PPMI, SVD, SGNS
cds	1, 0.75	PPMI, SVD, SGNS
w+c	only w, $w + c$	SVD, SGNS, GloVe
eig	0, 0.5, 1	SVD
nrm	none†, row, col†, both†	All

Experiment Setup: Training

- Train on English Wikipedia dump. 77.5 million sentences, 1.5 billion tokens

Experiment Setup: Training

- Train on English Wikipedia dump. 77.5 million sentences, 1.5 billion tokens
- Use $d = 500$ for SVD, SGNS, Glove

Experiment Setup: Training

- Train on English Wikipedia dump. 77.5 million sentences, 1.5 billion tokens
- Use $d = 500$ for SVD, SGNS, Glove
- Glove trained for 50 iterations

Experiment Setup: Testing

- Similarity task: Models used for word similarity tasks on six datasets. Each dataset human labeled with word pair similarity scores

Experiment Setup: Testing

- <u>Similarity task</u>: Models used for word similarity tasks on six datasets. Each dataset human labeled with word pair similarity scores
- Similarity scores calculated with cosine similarity

Experiment Setup: Testing

- Similarity task: Models used for word similarity tasks on six datasets. Each dataset human labeled with word pair similarity scores
- Similarity scores calculated with cosine similarity
- Analogy task: two analogy datasets used. Given analogies of the form a is to a^* as b is to b^* where b^* is not given. Example: "Paris is to France as Tokyo is to _".

Experiment Setup: Testing

- Similarity task: Models used for word similarity tasks on six datasets. Each dataset human labeled with word pair similarity scores
- Similarity scores calculated with cosine similarity
- Analogy task: two analogy datasets used. Given analogies of the form a is to a^* as b is to b^* where b^* is not given. Example: "Paris is to France as Tokyo is to _".
- Analogies calculated using 3CosAdd and 3CosMul

Experiment Setup: Testing

- Similarity task: Models used for word similarity tasks on six datasets. Each dataset human labeled with word pair similarity scores
- Similarity scores calculated with cosine similarity
- Analogy task: two analogy datasets used. Given analogies of the form a is to a^* as b is to b^* where b^* is not given. Example: "Paris is to France as Tokyo is to _".
- Analogies calculated using 3CosAdd and 3CosMul
- 3CosAdd: $arg\ max_{b^* \in V_w \setminus \{a, a^*, b\}} (\cos(b^*, a^*) - \cos(b^*, a) + \cos(b^*, b))$

Experiment Setup: Testing

- Similarity task: Models used for word similarity tasks on six datasets. Each dataset human labeled with word pair similarity scores
- Similarity scores calculated with cosine similarity
- Analogy task: two analogy datasets used. Given analogies of the form a is to a^* as b is to b^* where b^* is not given. Example: "Paris is to France as Tokyo is to _".
- Analogies calculated using 3CosAdd and 3CosMul
- 3CosAdd: $arg\ max_{b^* \in V_w \setminus \{a, a^*, b\}} (\cos(b^*, a^*) - \cos(b^*, a) + \cos(b^*, b))$
- 3CosMul: $arg\ max_{b^* \in V_w \setminus \{a, a^*, b\}} \frac{\cos(b^*, a^*) \cdot \cos(b^*, b)}{\cos(b^*, a) + \varepsilon}$ where $\varepsilon = .001$

Experiment Results

Method	WordSim Similarity	WordSim Relatedness	Bruni et al. MEN	Radinsky et al. M. Turk	Luong et al. Rare Words	Hill et al. SimLex	Google Add / Mul	MSR Add / Mul
PPMI	.709	.540	.688	.648	.393	.338	.491 / **.650**	.246 / .439
SVD	**.776**	**.658**	**.752**	.557	**.506**	**.422**	.452 / .498	.357 / .412
SGNS	.724	.587	.686	**.678**	.434	.401	.530 / .552	.578 / **.592**
GloVe	.666	.467	.659	.599	.403	.398	.442 / .465	.529 / .576

Table 2: Performance of each method across different tasks in the "vanilla" scenario (all hyperparameters set to default): `win` = 2; `dyn` = none; `sub` = none; `neg` = 1; `cds` = 1; `w+c` = only w; `eig` = 0.0.

Method	WordSim Similarity	WordSim Relatedness	Bruni et al. MEN	Radinsky et al. M. Turk	Luong et al. Rare Words	Hill et al. SimLex	Google Add / Mul	MSR Add / Mul
PPMI	.755	**.688**	.745	**.686**	.423	.354	.553 / **.629**	.289 / .413
SVD	**.784**	.672	**.777**	.625	**.514**	.402	.547 / .587	.402 / .457
SGNS	.773	.623	.723	.676	.431	**.423**	.599 / .625	.514 / .546
GloVe	.667	.506	.685	.599	.372	.389	.539 / .563	.503 / **.559**
CBOW	.766	.613	.757	.663	.480	.412	.547 / .591	.557 / **.598**

Table 3: Performance of each method across different tasks using `word2vec`'s recommended configuration: `win` = 2; `dyn` = with; `sub` = dirty; `neg` = 5; `cds` = 0.75; `w+c` = only w; `eig` = 0.0. CBOW is presented for comparison.

Method	WordSim Similarity	WordSim Relatedness	Bruni et al. MEN	Radinsky et al. M. Turk	Luong et al. Rare Words	Hill et al. SimLex	Google Add / Mul	MSR Add / Mul
PPMI	.755	**.697**	.745	.686	.462	.393	.553 / .679	.306 / .535
SVD	**.793**	.691	**.778**	.666	**.514**	.432	.554 / .591	.408 / .468
SGNS	**.793**	.685	.774	**.693**	.470	**.438**	.676 / **.688**	.618 / **.645**
GloVe	.725	.604	.729	.632	.403	.398	.569 / .596	.533 / .580

Table 4: Performance of each method across different tasks using the best configuration for that method and task combination, assuming `win` = 2.

Experiment Results Cont.

win	Method	WordSim Similarity	WordSim Relatedness	Bruni et al. MEN	Radinsky et al. M. Turk	Luong et al. Rare Words	Hill et al. SimLex	Google Add / Mul	MSR Add / Mul
2	PPMI	.732	**.699**	.744	.654	.457	.382	.552 / .677	.306 / .535
	SVD	.772	.671	**.777**	.647	**.508**	.425	.554 / .591	.408 / .468
	SGNS	**.789**	.675	.773	**.661**	.449	**.433**	.676 / **.689**	.617 / **.644**
	GloVe	.720	.605	.728	.606	.389	.388	.649 / .666	.540 / .591
5	PPMI	.732	**.706**	.738	**.668**	.442	.360	.518 / .649	.277 / .467
	SVD	.764	.679	**.776**	.639	**.499**	**.416**	.532 / .569	.369 / .424
	SGNS	**.772**	.690	.772	.663	.454	.403	.692 / **.714**	.605 / **.645**
	GloVe	.745	.617	.746	.631	.416	.389	.700 / .712	.541 / .599
10	PPMI	.735	**.701**	.741	.663	.235	.336	.532 / .605	.249 / .353
	SVD	.766	.681	.770	.628	**.312**	.419	.526 / .562	.356 / .406
	SGNS	**.794**	.700	**.775**	**.678**	.281	**.422**	.694 / .710	.520 / **.557**
	GloVe	.746	.643	.754	.616	.266	.375	.702 / **.712**	.463 / .519
10	SGNS-LS	.766	.681	**.781**	**.689**	**.451**	.414	.739 / **.758**	.690 / **.729**
	GloVe-LS	.678	.624	.752	.639	.361	.371	.732 / .750	.628 / .685

Table 5: Performance of each method across different tasks using 2-fold cross-validation for hyperparameter tuning. Configurations on large-scale (LS) corpora are also presented for comparison.

Key Takeaways

- Average score of SGNS (Word2vec) is lower than SVD for window sizes 2, 5. SGNS never outperforms SVD by more than 1.7%

Key Takeaways

- Average score of SGNS (Word2vec) is lower than SVD for window sizes 2, 5. SGNS never outperforms SVD by more than 1.7%
- Previous results to the contrary due to using tuned Word2vec vs vanilla SVD

Key Takeaways

- Average score of SGNS (Word2vec) is lower than SVD for window sizes 2, 5. SGNS never outperforms SVD by more than 1.7%
- Previous results to the contrary due to using tuned Word2vec vs vanilla SVD
- Glove only superior to SGNS on analogy tasks using 3CosAdd (generally considered inferior to 3CosMult)

Key Takeaways

- Average score of SGNS (Word2vec) is lower than SVD for window sizes 2, 5. SGNS never outperforms SVD by more than 1.7%
- Previous results to the contrary due to using tuned Word2vec vs vanilla SVD
- Glove only superior to SGNS on analogy tasks using 3CosAdd (generally considered inferior to 3CosMult)
- CBOW seems to only perform well on MSR analogy data set

Key Takeaways

- Average score of SGNS (Word2vec) is lower than SVD for window sizes 2, 5. SGNS never outperforms SVD by more than 1.7%
- Previous results to the contrary due to using tuned Word2vec vs vanilla SVD
- Glove only superior to SGNS on analogy tasks using 3CosAdd (generally considered inferior to 3CosMult)
- CBOW seems to only perform well on MSR analogy data set
- SVD does not benefit from shifting PMI matrix

Key Takeaways

- Average score of SGNS (Word2vec) is lower than SVD for window sizes 2, 5. SGNS never outperforms SVD by more than 1.7%
- Previous results to the contrary due to using tuned Word2vec vs vanilla SVD
- Glove only superior to SGNS on analogy tasks using 3CosAdd (generally considered inferior to 3CosMult)
- CBOW seems to only perform well on MSR analogy data set
- SVD does not benefit from shifting PMI matrix
- Using SVD with an eigenvalue weighting of 1 results in poor performance compared to .5 or 0

Recommendations

- Tune hyperparameters based on task at hand (duh)

Recommendations

- Tune hyperparameters based on task at hand (duh)
- If you’re using PMI, always use context distribution smoothing

Recommendations

- Tune hyperparameters based on task at hand (duh)
- If you're using PMI, always use context distribution smoothing
- If you're using SVD, always use eigenvalue weighting

Recommendations

- Tune hyperparameters based on task at hand (duh)
- If you’re using PMI, always use context distribution smoothing
- If you’re using SVD, always use eigenvalue weighting
- SGNS always performs well and is computationally efficient to train and utilize

Recommendations

- Tune hyperparameters based on task at hand (duh)
- If you're using PMI, always use context distribution smoothing
- If you're using SVD, always use eigenvalue weighting
- SGNS always performs well and is computationally efficient to train and utilize
- With SGNS, use many negative examples, i.e. prefer larger k

Recommendations

- Tune hyperparameters based on task at hand (duh)
- If you're using PMI, always use context distribution smoothing
- If you're using SVD, always use eigenvalue weighting
- SGNS always performs well and is computationally efficient to train and utilize
- With SGNS, use many negative examples, i.e. prefer larger k
- Experiment with $\vec{w} = \vec{w} + \vec{c}$ variation in SGNS and Glove

References

- [0] Levy, Omer, Yoav Goldberg, and Ido Dagan. "Improving distributional similarity with lessons learned from word embeddings." *Transactions of the Association for Computational Linguistics* 3 (2015): 211-225.
- [1] Mikolov, Tomas, et al. "Distributed representations of words and phrases and their compositionality." *Advances in neural information processing systems*. 2013.
- [2] Pennington, Jeffrey, Richard Socher, and Christopher Manning. "Glove: Global vectors for word representation." *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*. 2014.