Introduction to GloVe Embeddings

In the previous articles, we have discussed about what word embeddings are and how to train them from scratch or using word2vec models. This article is an intuitive guide to understanding Glove Embeddings which is a powerful word vector learning technique. We focus on why GloVe is better than word2vec in some ways and arrive at cost function of GloVe used for training word vectors.

To recap, word embeddings transform words into a vector space where similar words are placed together and different words wide off. Word2Vec models only consider the local (context, target) words for training word vectors, unlike GloVe which does not rely on local statistics or local context of words but includes global statistics to train word vectors.

Drawbacks of Word2Vec models

To refresh on idea behind Word2Vec models, we considered the local context and target words for training word embeddings and it works quite well so why not just stick with it. Because, as it relies only on local information the semantics learnt for the target word only depend on surrounding context words. For example, consider the following sentence:

An apple a day keeps the doctor away

When trained using word2vec models it doesn't explain whether "the" is special context of words "day, doctor" or is "the" just a stopword.

Global Vectors – GloVe

GloVe model learns vectors or word embeddings from their co-occurrence information. That is how frequently they appear together in large text corpus. Unlike word2vec which is a predictive deep learning model, GloVe is a count-based model.

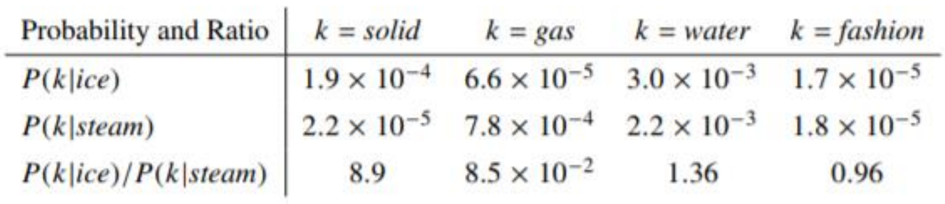
Consider a corpus having ***V*** words, the co-occurrence matrix ***X*** will be of size ***V* x *V***, where X_{ij} denotes the number of times word **j** appears in the context of word **i.** Let's consider these two sentences to form co-occurrence matrix.

I love NLP.

I love to write blogs.

	I	love	NLP	to	write	blogs	.
I	0	2	0	0	0	0	0
love	2	0	1	1	0	0	0
NLP	0	1	0	0	0	0	1
to	0	1	0	0	1	0	0
write	0	0	0	1	0	1	0
blogs	0	0	0	0	1	0	1
.	0	0	1	0	0	1	0

Now, how to obtain a metric that computes semantic similarity between words from the above symmetric co-occurrence matrix. For doing that, we require three words at a time as shown below.



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

The behavior of P_{ik} / P_{jk} for different words [1]

Consider the expression,

$$P_{ik} / P_{jk} \text{ ,where } P_{ik} = X_{ik} / X_i$$

P_{ik} is the probability of seeing words **i** and **k** together computed by dividing no. of times words **i** and **k** appeared together (**X_{ik}**) by total number of times word **i** appeared in the corpus (**X_i**).

Let words **(i, j)** be **(ice, steam),** following inferences can be drawn from the above table:

- When word **k = solid**, the ratio **P_{ik} / P_{jk}** will be very **high** (>1) as the word **solid** is similar to **ice** but irrelevant to **steam.**
- When word **k = gas**, the ratio **P_{ik} / P_{jk}** will be very **low** (<1) as the word **gas** is similar to **steam** but irrelevant to **ice.**
- When word **k is random**, the ratio **P_{ik} / P_{jk}** will be close to 1 as the word **k** is unrelated to **ice** and **steam.**

So, if we can find a way to include the ratio **P_{ik} / P_{jk}** to compute word embeddings we will achieve our goal of using global statistics when training word vectors.

To achieve this, we have mainly 3 problems listed below to overcome:

- The ratio **P_{ik} / P_{jk}** is a scalar whereas the corresponding word vectors are high dimensional and there is a dimensional mismatch.
- As we have three terms (**i, j and k**) involved it is difficult to form loss function with 3 entities, so it should be reduced to only 2 terms.
- We only have an expression but not the equation **$F(i, j, k) = P_{ik}/P_{jk}$**

Transforming independent variables of function F

Going forward, we will solve these three problems and find how it helps us in finding word vector algorithm. Suppose we have a function which gives us the ratio as shown below.

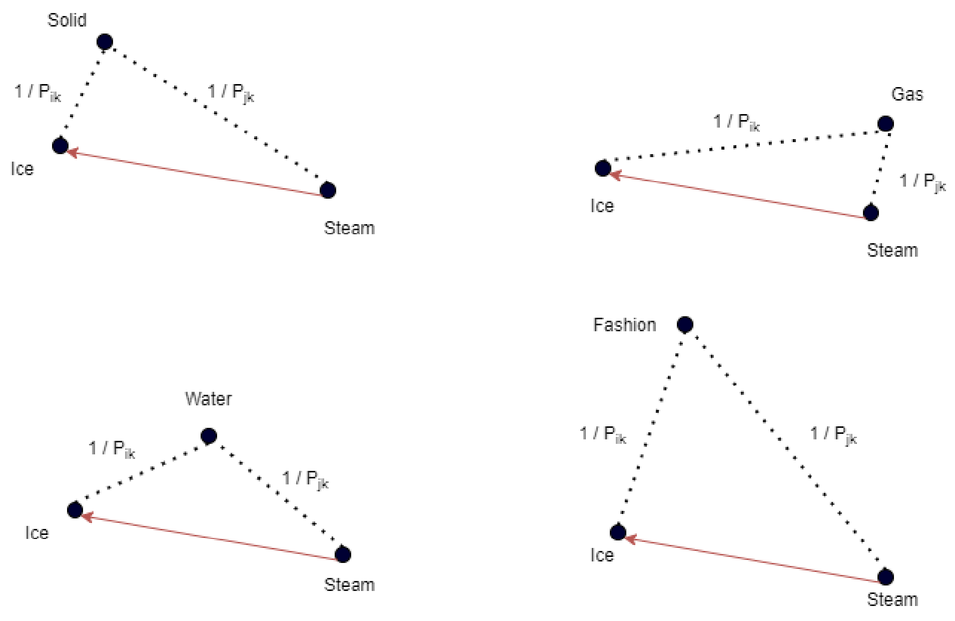
$$\mathbf{F(w_i, w_j, u_k) = P_{ik}/P_{jk}} \qquad \text{--------} \quad \mathbf{(1)}$$

where **w** and **u** are two embedding layers as described in the GloVe paper it helps model to reduce overfitting, they both differ only at random initialization. Since the word vectors $\mathbf{w_i, w_j, u_k}$ are linear systems we can perform arithmetic operations, e.g.

$$\mathbf{w_{king} - w_{male} + w_{female} = w_{queen}}$$

Transforming eq **(1)** to $\mathbf{F(w_i\text{-}w_j, u_k) = P_{ik}/P_{jk}}$ has many added advantages or properties about $\mathbf{P_{ik}/P_{jk}}$ in the embedding space.

Consider the four scenarios shown in below figure, when started with $\mathbf{w_i\text{-}w_j}$ vector and computing distance of target word **k** correlated with reciprocal of $\mathbf{P_{ik}}$ the distance or length of dashed lines clearly changes when different words are considered. So, it is a good idea to start with $\mathbf{w_i\text{-}w_j}$.



Changes of target word **k** with respect to vector $\mathbf{w_i\text{-}w_j}$

Vector – Scalar Problem

We can overcome this by a simple fix by introducing a transpose and dot product between two vectors as the following way.

$$\mathbf{F((w_i\text{-}w_j)^T \cdot u_k) = P_{ik}/P_{jk}}$$

If the word embedding is of size **D x 1** then $\mathbf{(w_i\text{-}w_j)^T}$ will be **1 x D** when multiplied with $\mathbf{u_k}$ gives a scalar as output.

Finding out the function F

Assuming the function **F** has **homomorphism** property between multiplicative and additive groups, e.g. homomorphism ensures that **F(A – B)** can be written as **F(A) / F(B)**, it gives

$$F(w_i * u_k - w_j * u_k) = F(w_i * u_k) / F(w_j * u_k) = P_{ik} / P_{jk}$$

In the paper **[1]**, they have assumed $F(w_i * u_k) / F(w_j * u_k) = P_{ik} / P_{jk}$ to

$$F(w_i * u_k) = cP_{ik} \text{ for some constant } c$$

As you have probably guessed by now, the **exp** function satisfies the homomorphism property. Therefore,

$$\mathrm{Exp}(w_i * u_k) = P_{ik} = X_{ik} / X_i \quad \text{and}$$

$$w_i * u_k = \log(X_{ik}) - \log(X_i) \quad \text{as } X_i \text{ independent of } k$$

$$w_i * u_k + \log(X_i) = \log(X_{ik})$$

Finally expressing **log(X$_i$)** in terms of neural network jargon, we obtain

$$w_i * u_k + bw_i + bu_k - \log(X_{ik}) = 0 \qquad \text{here } bw_i, bu_k \text{ are biases of network}$$

Cost Function

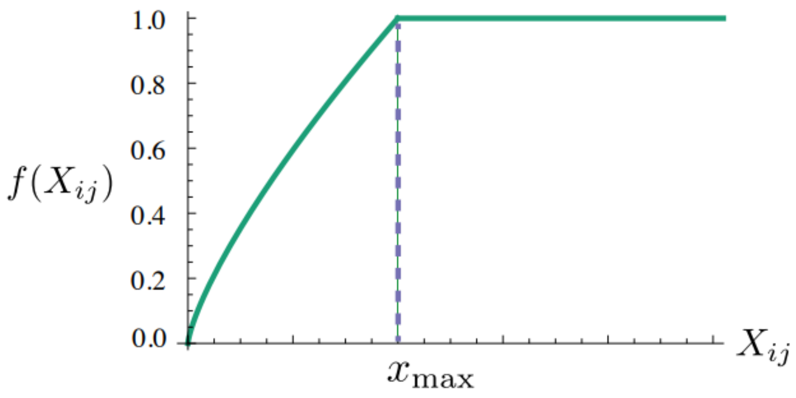
Based on the soft constraint equation above, we get **0** for perfect word embedding vectors, so our goal is to minimize the below objective function **J**

$$J = \sum_{i,j=1}^{V} f\left(X_{ij}\right)\left(w_i^T \tilde{w}_j + b_i + \tilde{b}_j - \log X_{ij}\right)^2$$

V is the size of vocabulary, b_i and b_j are scalar bias terms associated with words **i** and **j**, respectively.

Some words i, j may rarely co-occur or never then the term $\log(X_{ij})$ goes crazy. To avoid this, they have added a weighting function f(x) given below

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



Weighting function f with $\alpha = 3 / 4$ [1]

Advantages

- Fast training as we can incorporate parallel computing during training which is not possible for word2vec models.
- Scalable to very large corpus and also gives good performance with small corpus.

Drawbacks

- Memory intensive process as to faster training process we need to keep co-occurrence matrix in RAM as hash map and perform co-occurrence increments.

Conclusion

So, we have learned GloVe model which utilizes the main benefit of count data of words capturing global statistics which makes GloVe a powerful model for the unsupervised learning of word representations outperforming other models on word analogy, named entity recognition and word similarity tasks.

References

1. **GloVe: Global Vectors for Word Representation (original paper)**