Machine Learning Explained

Deep learning, python, data wrangling and other machine learning related topics explained for practitioners

Paper Dissected: "Glove: Global Vectors for Word Representation" Explained

Pre-trained word embeddings are a staple in deep learning for NLP. The pioneer of word embeddings in mainstream deep learning is the renowned word2vec. GloVe is another commonly used method of obtaining pre-trained embeddings. Unfortunately, there are very few practitioners that seem to actually understand GloVe; many just consider it "another word2vec". In reality, GloVe is a much more principled approach to word embeddings that provides deep insights into word embeddings in general.

GloVe is a must-read paper for any self-respecting NLP engineer that can seem intimidating at first glance. This post aims to dissect and explain the paper for engineers and highlight the differences and similarities between GloVe and word2vec .

TL;DR

- GloVe aims to achieve two goals:
  - (1) Create word vectors that **capture meaning in vector space**
  - (2) Takes advantage of **global count statistics** instead of only local information

- Unlike word2vec – which learns by streaming sentences – GloVe learns based on a **co-occurrence matrix** and trains word vectors so their differences predict **co-occurrence ratios**
- GloVe weights the loss based on word frequency
- Somewhat surprisingly, word2vec and GloVe turn out to be extremely similar, despite starting off from entirely different starting points

Motivation and Brief Review of Word Embeddings

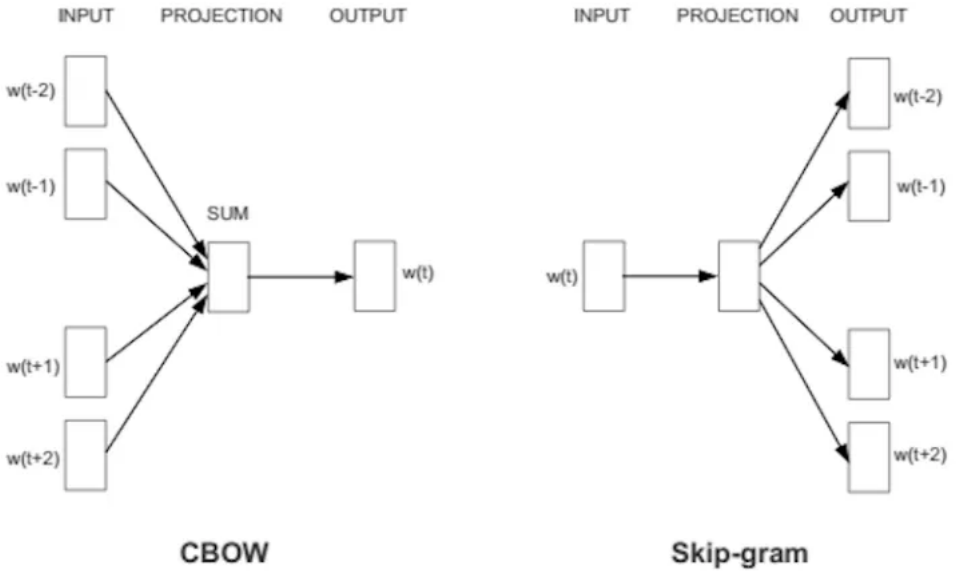
The Problem with Word2vec

GloVe was published after word2vec so the natural question to ask is: **why is word2vec not enough**?

In case you are not familiar, here's a quick explanation of word2vec. Word2vec trains word embeddings by optimizing a loss function with gradient descent, just like any other deep learning model. In word2vec, the loss function is computed by measuring how well a certain word can predict its surroundings words. For instance, take the sentence

"The cat sat on the mat".

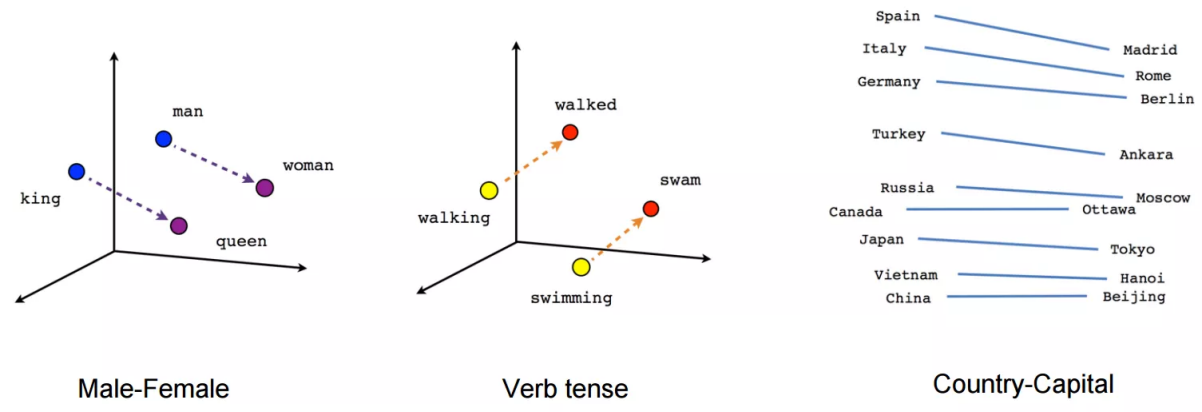
Suppose the context window is size 2, and the word we are focusing on is "sat". The context words are "the", "cat", "on" and "the". Word2vec tries to either predict the word in focus from the context words (this is called the CBOW model) or the context words using the word in focus (this is called the Skip-gram model).



I won't go into the details here since there are plenty of explanations online. The important thing to note is that word2vec **only takes local contexts into account**. It does not take advantage of global count statistics. For example, "the" and "cat" might be used together often, but word2vec does not know if this is because "the" is a common word or if this is because the words "the" and "cat" have a strong linkage (well actually, it indirectly does, but this is a topic I'll cover later on in this post). This is the motivation for using global count statistics.

The problem with global count statistic based methods

Before word2vec, using matrices that contained global count information was one of the major ways of converting words to vectors (yes, the idea of word vectors/embeddings existed far before word2vec). For instance, Latent Semantic Analysis (LSA) computed word embeddings by decomposing term-document matrices using Singular Value Decomposition. Though these methods take advantage of global information, the obtained vectors do not show the same behavior as those obtained by word2vec. For instance, in word2vec, word analogies can be expressed in terms of simple (vector) arithmetic such as in the case of "king − man + woman = queen".



Male-Female Verb tense Country-Capital

This behavior is desirable because it shows that the vectors capture **dimensions of meaning**. Some dimensions of the vectors might capture whether the word is male or female, present tense or past tense, plural or singular, etc.. This means that downstream models can easily extract the meaning from these vectors, which is what we really want to achieve when training word vectors.

GloVe aims to take the best of both worlds: take **global** information into account while learning **dimensions of meaning**. From this initial goal, GloVe builds up a principled method of training word vectors.

Building GloVe, Step by Step

Data Preparation

In word2vec, the vectors were learned so that they could achieve the task of predicting surrounding words in a sentence. During training, word2vec **streams the sentences** and computes the loss for each batch of words.

In GloVe, the authors take a more principled approach. The first step is to build a **co-occurrence matrix**. GloVe also takes local context into account by computing the co-occurrence matrix using a fixed window size (words are deemed to co-occur when they appear together within a fixed window). For instance, the sentence

"The cat sat on the mat"

with a window size of 2 would be converted to the co-occurrence matrix

	the	cat	sat	on	mat
the	2	1	2	1	1
cat	1	1	1	1	0
sat	2	1	1	1	0
on	1	1	1	1	1
mat	1	0	0	1	1

Notice how the matrix is symmetric: this is because when the word "cat" appears in the context of "sat", the opposite (the word "sat" appearing in the context of"cat") also happens.

What Should we Predict?

Now, the question is how to connect the vectors with the statistics computed above. The underlying principle behind GloVe can be stated as follows: the **co-occurrence ratios between two words in a context** are strongly connected to meaning.

This sounds difficult but the idea is really simple. Take the words "ice" and "steam", for instance. Ice and steam differ in their state but are the same in that they are both forms of water. Therefore, we would expect words related to water (like "water" and "wet") to appear equally in the context of "ice" and "steam". In contrast, words like "cold" and "solid" would probably appear near "ice" but would not appear near "steam".

The following table shows the actual statistics that show this intuition well:

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 probabilities shown here are basically just counts of how often the word k appears when the words “ice” and “steam” are in the context, where k refers to the words “solid”, “gas”, “water”, and “fashion”. As you can see, words that are related to the nature of “ice” and “steam” (“solid” and “gas” respectively) occur far more often with their corresponding words that the non-corresponding word. In contrast, words like “water” and “fashion” which are not particularly related to either have a probability ratio near 1. Note that the probability ratios can be computed easily using the co-occurrence matrix.

From here on, we’ll need a bit of mathematical notation to make the explanation easier. We’ll use X to refer to the co-occurrence matrix and X_{ij} to refer to the i, j th element in X which is equal to the number of times word j appears in the context of word i. We’ll also define $X_i = \sum_l X_{il}$ to refer to the total number of words that have appeared in the context of i. Other notation will be defined as we go along.

Deriving the GloVe Equation

(This subsection is not directly necessary to understand how GloVe and word2vec differ, etc. so feel free to skip it if you are not interested.)

Now, it seems clear that we should be aiming to predict the co-occurrence ratios using the word vectors. For now, we’ll express the relation between the ratios with the following equation:

$F(w_i, w_j, \tilde{w_k}) \approx \frac{P_{ij}}{P_{jk}}$

Here, P_{ij} refers to the probability of the word j appearing in the context of i, and can be computed as

$$P_{ij} = \frac{\text{number of times j appeared in context of i}}{\text{number of words that appeared in context of i}} = \frac{X_{ij}}{X_i}.$$

F is some unknown function that takes the embeddings for the words i, k, j as input. Notice that there are two kinds of embeddings: input and output (expressed as w and $\tilde{w}$) for the context and focus words. This is a relatively minor detail but is important to keep in mind.

Now, the question becomes what F should be. If you recall, one of the goals of **GloVe** was to create vectors with meaningful dimensions that expressed meaning using simple arithmetic (addition and subtraction). We should choose F so that the vectors it leads to meet this property.

Since we want simple *arithmetic* between the vectors to have meaning, it's only natural that we make the input to the function F also be the result of *arithmetic* between vectors. The simplest way to do this is by making the input to F the difference between the vectors we're comparing:

$$F(w_i - w_j, \tilde{w_k}) \approx \frac{P_{ij}}{P_{jk}}$$

Now, we want create a **linear relation** between $w_i - w_j$ and $\tilde{w_k}$. This can be accomplished by using the **dot product**:

$$F(\text{dot}(w_i - w_j, \tilde{w_k}) \approx \frac{P_{ij}}{P_{jk}}$$

Now, we'll use two tricks to determine F and simplify this equation:

1. By taking the log of the probability ratios, we can convert the ratio into a subtraction between probabilities
2. By adding a bias term for each word, we can capture the fact that some words just occur more often than others.

These two tricks give us the following equation:

$$\text{dot}(w_i - w_j, \tilde{w_k}) + b_i - b_j = \log(P_{ik}) - \log(Pjk)$$

We can convert this equation into an equation over a single entry in the co-occurrence matrix.

$$\text{dot}(w_i, \tilde{w_k}) + b_i = \log(P_{ik}) = \log(X_{ik}) - \log(X_i)$$

By absorbing the final term on the right-hand side into the bias term, and adding an output bias for symmetry, we get

$$\text{dot}(w_i, \tilde{w_k}) + b_i + \tilde{b}_k = \log(X_{ik})$$

And here we are. This is the core equation behind GloVe.

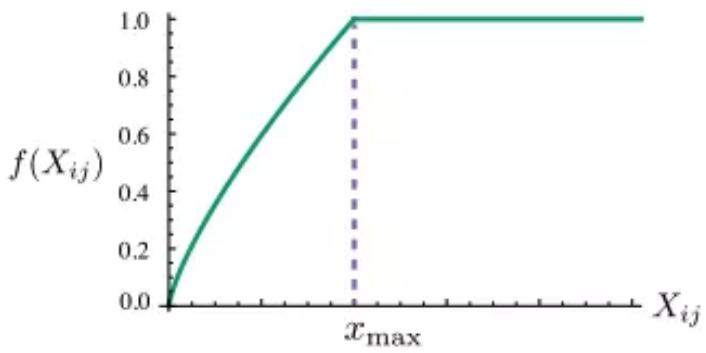
Weighting occurrences

There is one problem with the equation above: it weights all co-occurrences equally. Unfortunately, not all co-occurrences have the same quality of information. Co-occurrences that are **infrequent will tend to be noisy and unreliable**, so we want to weight frequent co-occurrences more heavily. On the other hand, we don't want co-occurrences like "it is" dominating the loss, so we don't want to weight too heavily based on frequency.

Through experimentation, the authors of the paper found the following weighting function to perform relatively well:

$$\text{weight}(x) = \min(1, (x/x_{max})^{\frac{3}{4}})$$

The function becomes easier to imagine when it is plotted:



Essentially, the function gradually increases with x but does not become any larger than 1. Using this function, the loss becomes

$$\sum_{ij} \text{weight}(X_{ij})(\text{dot}(w_i, \tilde{w_j}) + b_i + \tilde{b_k} - \log(X_{ij}))^2)$$

Comparison with Word2Vec

Though GloVe and word2vec use completely different methods for optimization, they are actually surprisingly mathematically similar. This is because, although word2vec does not explicitly decompose a co-occurrence matrix, it implicitly optimizes over one by streaming over the sentences.

Word2vec optimizes the log likelihood of seeing words in the same context windows together, where the probability of seeing word j in the context of i is *estimated* as

$$Q_{ij} = \text{softmax}(\text{dot}(w_i, \tilde{w_j}))$$

The loss function can be expressed as

$$-\sum_{i \in \text{corpus}; j \in \text{context}(i)} \log Q_{ij}$$

The number of times $\log Q_{ij}$ is added to the loss is equivalent to the **number of times the pair i, j appears in the corpus**. Therefore, this loss is equivalent to

$\sum_{i,j} X_{ij} \log Q_{ij}$

Now, if we recall, the *actual* probability of word j appearing in the context of i is

$P_{ij} = X_{ij}/X_i$

Meaning we can group over the context word i and get the following equation:

$\sum_i X_i \sum_j \frac{X_{ij}}{X_i} \log Q_{ij} = \sum_i X_i \sum_j P_{ij} \log Q_{ij}$

Notice how the term $\sum_j P_{ij} \log Q_{ij}$ takes the form of the cross-entropy loss. In other words, the loss of word2vec is actually just a **frequency weighted sum over the cross-entropy** between the predicted word distribution and the actual word distribution in the context of the word i. The weighting factor X_i comes from the fact that we stream across all the data equally, meaning a word that appears n times will contribute to the loss n times. There is no inherent reason to stream across all the data equally though. In fact, the word2vec paper actually argues that filtering the data and reducing updates from frequent context words improves performance. This means that the weighting factor can be an arbitrary function of frequency, leading to the loss function

$\sum_{ij} \text{weight}(X_{ij}) P_{ij} \log Q_{ij}$

Notice how this **only differs from the GloVe equation we derived above in terms of the loss measure** between the actual observed frequencies and the predicted frequencies. Whereas GloVe uses the log mean squared error between the predicted (unnormalized) probabilities and the actual observed (unnormalized) probabilities, word2vec uses the cross-entropy loss over the normalized probabilities.

The GloVe paper argues that log mean squared error is better than cross-entropy because cross entropy tends to put too much weight on the long tails. In reality, this is probably a subject that needs more analysis. The essential thing to know is that word2vec and GloVe are actually virtually mathematically the same, despite having entirely different derivations!

Computational Complexity

If word2vec and GloVe are mathematically similar, the natural thing to ask is if one is faster than the other.

GloVe requires computing the co-occurrence matrix, but since both methods need to build the vocabulary before optimization and the co-occurrence matrix can be built during this process, this is not the defining factor in the computational complexity.

Instead, the computational complexity of GloVe is proportionate to the number of non-zero elements of X. In contrast, the computational complexity of word2vec is proportionate to the size of the corpus $|C|$.

Though I won't go into the details, the frequency of a word pair can be approximated based on its frequency rank r_{ij} (the ranking of the word pair when all words pairs are sorted by frequency):

$X_{ij} \approx \frac{k}{r_{ij}^{\alpha}}$

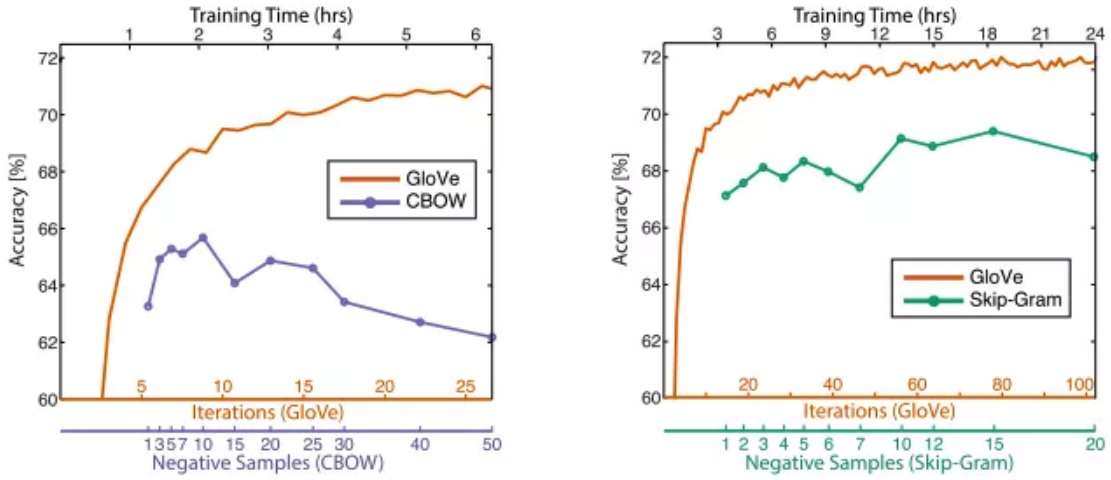
Using this, we can approximate the number of non-zero elements in X as

$\mathcal{O}(|C|)$ if $\alpha < 1$ else $\mathcal{O}(|C|^{\frac{1}{\alpha}})$.

Essentially, if some words occur very frequently it's faster to optimize over the statistics rather than to iterate over all the entire corpus repeatedly.

Results

Though the GloVe paper presents many experimental results, I'll focus the results it presents in terms of comparison with word2vec here.



(a) GloVe vs CBOW (b) GloVe vs Skip-Gram

The authors show that GloVe consistently produces better embeddings faster than word2vec. This makes sense, given how GloVe is much more principled in its approach to word embeddings.

Unfortunately, empirical results published after the GloVe paper seem to favor the conclusion that GloVe and word2vec perform roughly the same for many downstream tasks. Furthermore, most people use pre-trained word embeddings so the training time is not much of an advantage. Therefore, GloVe should be one choice for pre-trained word embeddings, but should not be the only choice.

Conclusion and Further Readings

GloVe is not just "another word2vec"; it is a more systematic approach to word embeddings that sheds light on the behavior of word2vec and embeddings in general. I hope this post has given the reader some more insight into this outstanding and deep paper.

Here are some further readings for those interested:

The original Glove paper

The word2vec paper

Other papers dissected:

Attention is All You Need

Deep Image Prior

Quasi-Recurrent Neural Networks

Share this:

Like this:

Like

Be the first to like this.

Related

An Overview of Sentence Embedding Methods

December 28, 2017
In “Deep Learning”

Paper Dissected: "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding" Explained

January 7, 2019

In “Deep Learning”

Paper Dissected: "Deep Contextualized Word Representations" Explained

June 15, 2018
In “Deep Learning”

keitakurita / April 29, 2018 / Deep Learning, NLP, Paper