# Practice 5

## GloVe

Stanford researchers also have their own word embedding algorithm like word2vec called Global Vectors for Word Representation, or GloVe for short.

Similar to Word2Vec, the intuition behind GloVe is also creating contextual word embeddings. But given the great performance of Word2Vec, why was there a need for something like GloVe?

How does GloVe improve over Word2Vec?

Word2Vec is a window-based method, in which the model relies on local information for generating word embeddings, which in turn is limited to the adjudged window size that we choose. This means that the semantics learned for a target word is only affected by its surrounding words in the original sentence, which is a somewhat inefficient use of statistics, as there's a lot more information we can work with.
GloVe on the other hand captures both global and local statistics in order to come up with the word embeddings.
We saw local statistics used in Word2Vec, but what are global statistics now?

GloVe derives semantical meaning by training on a co-occurrence matrix. It's built on the idea that word-word co-occurrences are a vital piece of information and using them is an efficient use of statistics for generating word embeddings. This is how GloVe manages to incorporate "global statistics" into the end result.

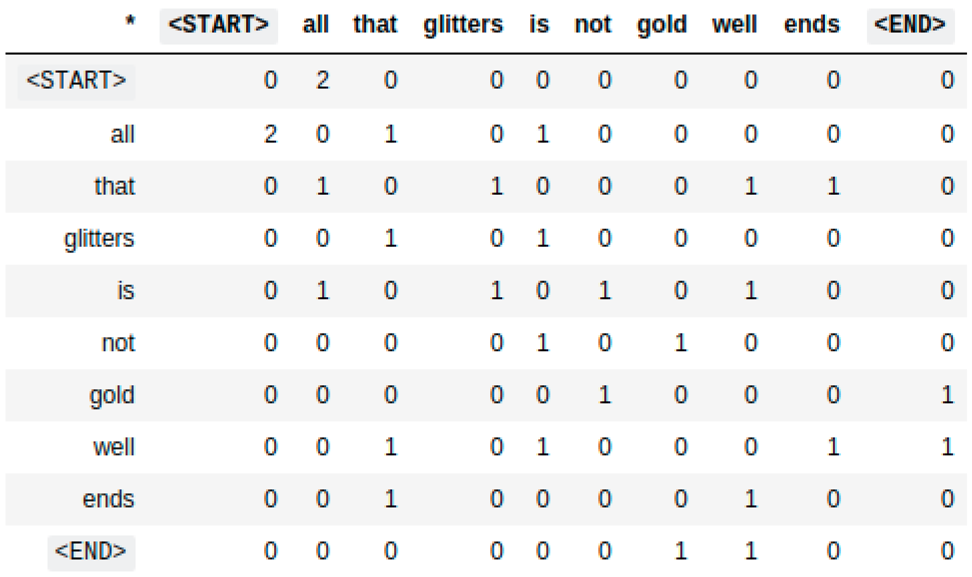
For those of you who aren't aware of the co-occurrence matrix, here's an example:

Let's say we have two documents or sentences.

Document 1: All that glitters is not gold.

Document 2: All is well that ends well.

Then, with a fixed window size of n = 1, our co-occurrence matrix would look like this:



| * | <START> | all | that | glitters | is | not | gold | well | ends | <END> |
|---|---|---|---|---|---|---|---|---|---|---|
| <START> | 0 | 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| all | 2 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| that | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 0 |
| glitters | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| is | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0 |
| not | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 |
| gold | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 |
| well | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 1 |
| ends | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| <END> | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 |

If you take a moment to look at it, you realize the rows and columns are made up of our vocabulary, i.e. the set of unique tokenized words obtained from both documents.
Here, <START> and <END> are used to denote the beginning and end of sentences.
The window of size 1 extends in both directions of the word, since 'that' & 'is' occur only once in the window vicinity of 'glitters', that is why the value of (that, glitters) and (is, glitters) = 1, you get the idea now how to go about this table.

Finally, let ***P(j|i)=P$_{i,j}$ = Xij/Xi*** be the probability that word *j* appear in the context of *i*. Where ***Xi*** – sum *Xij* in row *i*. Here, in below table if we want to calculate P(is|glitters) = 1/2.

A little about its training, GloVe model is a weighted least squares model and thus its cost function looks something like this [https://nlp.stanford.edu/pubs/glove.pdf]:

$$J = \frac{1}{2}\sum_{i,j=1}^{W} f(P_{i,j})(u_i^T v_j - \log (P_{i,j})^2,$$

where $W$ is the size of the vocabulary, $f\left(P_{i,j}\right)$ – is the weighting function.

For every pair of words ($i$, $j$) that might co-occur, we try to minimize the difference between the product of their word embeddings ($u_i^T v_j$) and the log of the co-occurrence count of $P(i,j)$. The term $f(P_{i,j})$ makes it a weighted summation and allows us to give lower weights to very frequent word co-occurrences, capping the importance of such pairs.

The weighting function should obey the following properties:
1. $f(0) = 0$. If $f$ is viewed as a continuous function, it should vanish as $x \rightarrow 0$ fast enough that the $lim_{x\rightarrow 0} f(x) \, log^2 x$ is finite.
2. $f(x)$ should be non-decreasing so that rare co-occurrences are not overweighted.
3. $f(x)$ should be relatively small for large values of $x$, so that frequent co-occurrences are not overweighted.

Of course a large number of functions satisfy these properties, but one class of functions that we found to work well can be parameterized as,

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

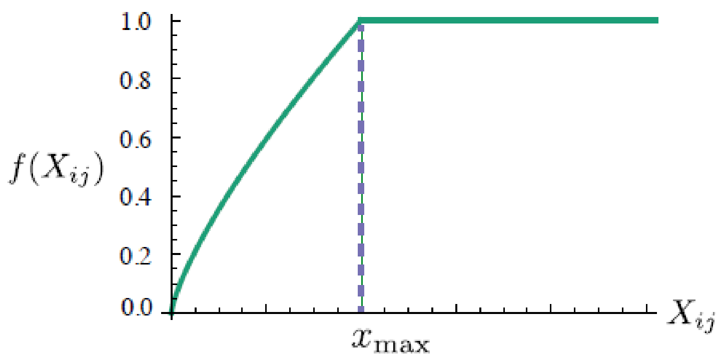


Figure 1: Weighting function $f$ with $\alpha = 3/4$.

When to use GloVe?

GloVe has been found to outperform other models on word analogy, word similarity, and Named Entity Recognition tasks, so if the nature of the problem you're trying to solve is similar to any of these, GloVe would be a smart choice.
Since it incorporates global statistics, it can capture the semantics of rare words and performs well even on a small corpus.

Task 1: Download GloVe embeddings from here: https://nlp.stanford.edu/data/glove.6B.zip. There are several files: `'glove.6B.50d.txt'`, `'glove.6B.100d.txt'`, `'glove.6B.200d.txt'`, `'glove.6B.300d.txt'` with GloVe embeddings of corresponding vector dimensions. For addition information visit: https://nlp.stanford.edu/projects/glove/.

Load dictionary of dimension 100 from file `'glove.6B.100d.txt'`:

```
import numpy as np
from scipy import spatial

emmbed_dict = {}
with open('glove.6B/glove.6B.100d.txt','r', encoding="utf8")
as f:
  for line in f:
    values = line.split()
    word = values[0]
    vector = np.asarray(values[1:],'float32')
    emmbed_dict[word]=vector
```

`Output a GloVe vector for some word, ex. india:`

```
emmbed_dict['india']
```

Task 2: Define a function `similar_word(vector)`, that finds 10 nearest words with an Euclidian metrics. Output 10 nearest words to word `'holiday'`.
Task 3: pass to the defined function a result of vector sum: `(emmbed_dict['king'] + emmbed_dict['queen'] + emmbed_dict['prince'])`

To visualize the vectors, we are using a method called distributed stochastic gradient neighbor embeddings in short known as TSNE, which is used to reduce data dimensions. Here we are dealing with 100-dimensional data TSNE will break it down into components as we want here; we will break it into two dimensions.

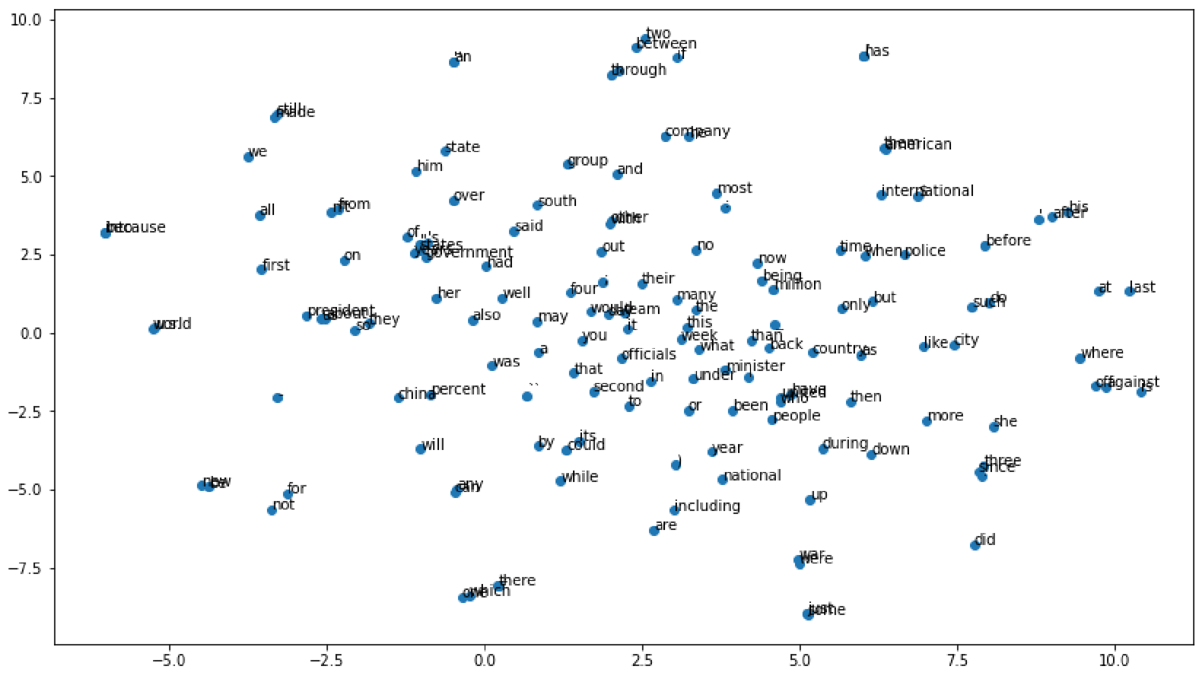
After training and fitting the TSNE model, it is all about plotting the vector; we are using a scatter plot as the vector is distributed over the space. The matplotlib library can do this. Annotating each point will give a more insightful vector representation.

```
import matplotlib.pyplot as plt
from sklearn.manifold import TSNE

distri = TSNE(n_components=2)
words = list(emmbed_dict.keys())
vectors = [emmbed_dict[word] for word in words]
y = distri.fit_transform(vectors[700:850])
plt.figure(figsize=(14,8))
plt.scatter(y[:, 0],y[:,1])

for label,x,y in zip(words,y[:, 0],y[:,1]):
  plt.annotate(label,xy=(x,y),xytext=(0,0),textcoords='offset points')
plt.show()
```

Task 4: add detailed comments to the code above.



## Convert GloVe Embedding to word2Vec

We can also convert the initial GloVe embeddings file to word2vec format:

```
from gensim.scripts.glove2word2vec import glove2word2vec
glove_input_file = 'glove.6B.100d.txt'
word2vec_output_file = 'glove.6B.100d.txt.word2vec'
glove2word2vec(glove_input_file, word2vec_output_file)
```

Task 5: do task 2 and 3 for word2vec embeddings after converting