ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

DIPARTIMENTO DI
SCIENZE STATISTICHE "PAOLO FORTUNATI"

6 - Word Embedding

Giovanni Della Lunga

giovanni.dellalunga@unibo.it

Halloween Conference in Quantitative Finance

Bologna - October 26-28, 2021

We will talk about...

- Feature Extraction
- Semantic Space
- Bag-of-Words Models
- Vectorization with SkLearn
- Document Similarity
- Word Embedding: word2vec and GloVe
- Using keras

Introduction

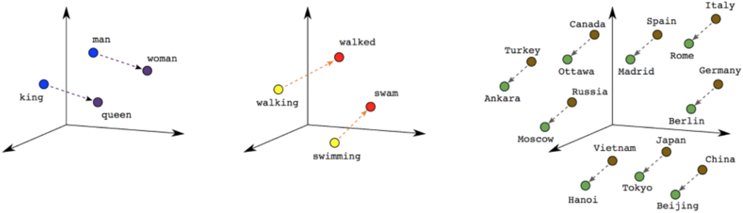
Turning Words into Numbers

Introduction

- Machine learning algorithms operate on a numeric feature space, expecting input as a two-dimensional array where rows are instances and columns are features.
- What are the appropriate features for applying a ML model to a text?
- In order to perform machine learning on text, we need to transform our documents into vector representations such that we can apply numeric machine learning.
- This process is called **feature extraction** or more simply, **vectorization**, and is an essential first step toward language-aware analysis.

Introduction

- For this reason, we must now make a critical shift in how we think about language - from a sequence of words to points that occupy a high-dimensional semantic space



Male-Female Verb Tense Country-Capital

Introduction

- Points in space can be close together or far apart, tightly clustered or evenly distributed.
- Semantic space is therefore mapped in such a way where documents with similar meanings are closer together and those that are different are farther apart.
- By encoding **similarity** as distance, we can begin to derive the primary components of documents and draw decision boundaries in our semantic space.
- The simplest encoding of semantic space is the bag-of-words model, whose primary insight is that meaning and similarity are encoded in **vocabulary**.

Bag-of-Words (BOW)

Bag-of-Words

- To vectorize a corpus with a bag-of-words (BOW) approach, we represent every document from the corpus as a vector whose dimension is equal to the size of the corpus vocabulary.
- The vocabulary is the set of words without repetition present in the whole corpus.

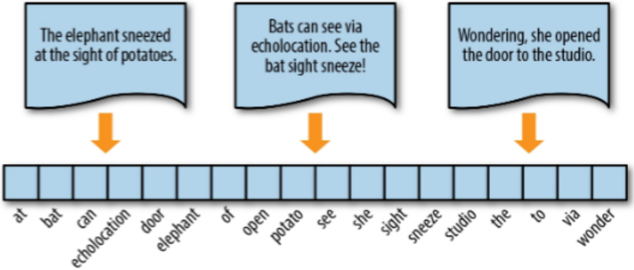

The elephant sneezed at the sight of potatoes.
Bats can see via echolocation. See the bat sight sneeze!
Wondering, she opened the door to the studio.
at
bat
can
echolocation
door
elephant
of
open
potato
see
she
sight
sneeze
studio
the
to
via
wonder

image source: *Bengfort B. et al. Text Analysis with Python*

Bag-of-Words

What should each element in the document vector be?

- We will explore several choices, each of which extends or modifies the base bag-of-words model to describe semantic space.
- We will look at three types of vector encoding - frequency, one-hot, TF-IDF, - and discuss their implementations in Scikit-Learn and NLTK.

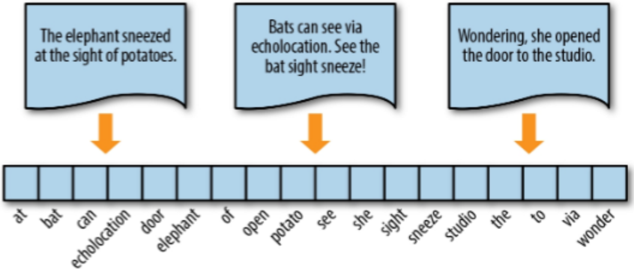

The elephant sneezed at the sight of potatoes.
Bats can see via echolocation. See the bat sight sneeze!
Wondering, she opened the door to the studio.
at
bat
can
echolocation
door
elephant
of
open
potato
see
she
sight
sneeze
studio
the
to
via
wonder

image source: *Bengfort B. et al. Text Analysis with Python*

Bag-of-Words

- **Frequency Vectors**
- The simplest vector encoding model is to simply fill in the vector with **the frequency of each word as it appears in the document**;
- In this encoding scheme each document is represented as the multiset of the tokens that compose it and the value for each word position in the vector is its count;
- This representation can either be a straight count encoding or a normalized encoding where each word is weighted by the total number of words in the document;

Bag-of-Words

Token Frequency as Vector Encoding

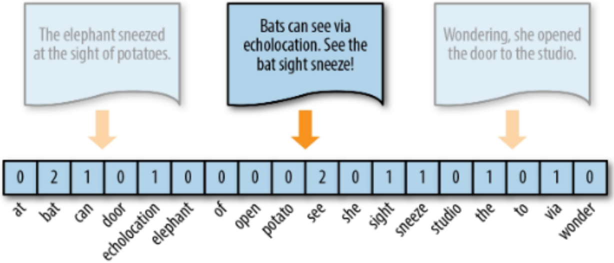
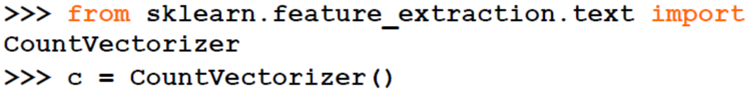

The elephant sneezed at the sight of potatoes.
Bats can see via echolocation. See the bat sight sneeze!
Wondering, she opened the door to the studio.
0 2 1 0 1 0 0 0 0 2 0 1 1 0 1 0 1 0
at
bat
can
door
echolocation
elephant
of
open
potato
see
she
sight
sneeze
studio
the
to
via
wonder

image source: *Bengfort B. et al. Text Analysis with Python*

Text Feature Extraction

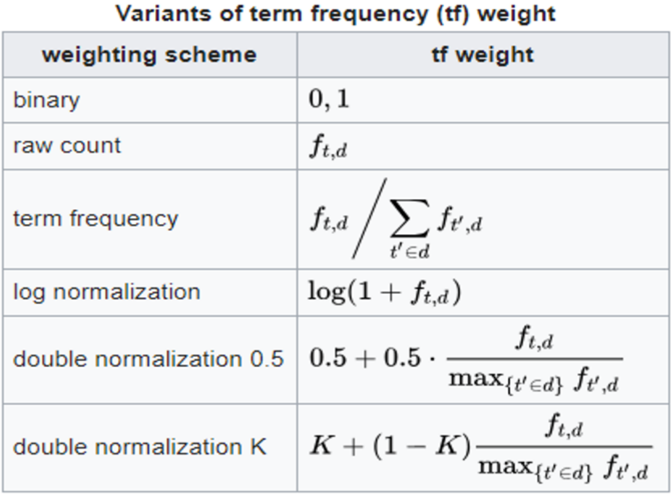
- CountVectorizer: turns a collection of text documents into numerical feature vectors.



```
>>> from sklearn.feature_extraction.text import
CountVectorizer
>>> c = CountVectorizer()
```

- **TF**: Just counting the number of words in each document has 1 issue: it will give **more weightage to longer documents than shorter documents**. To avoid this, we can use frequency (**TF - Term Frequencies**) i.e. #count(word) / #Total words, in each document.

Bag-of-Words

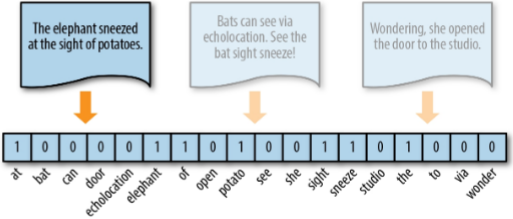


Variants of term frequency (tf) weight

weighting scheme	tf weight
binary	$0, 1$
raw count	$f_{t,d}$
term frequency	$f_{t,d} \Big/ \sum_{t' \in d} f_{t',d}$
log normalization	$\log(1 + f_{t,d})$
double normalization 0.5	$0.5 + 0.5 \cdot \frac{f_{t,d}}{\max_{\{t' \in d\}} f_{t',d}}$
double normalization K	$K + (1 - K)\frac{f_{t,d}}{\max_{\{t' \in d\}} f_{t',d}}$

One-Hot Encoding

- One-Hot encoding is a method that produces a boolean vector.
- A particular vector index is marked as TRUE (1) if the token exist in the document and FALSE (0) if it does not.
- This method can be used also in the limit case of a single word.



The elephant sneezed at the sight of potatoes.

Bats can see via echolocation. See the bat sight sneeze!

Wondering, she opened the door to the studio.

at	bat	can	door	echolocation	elephant	of	open	potato	see	she	sight	sneeze	studio	the	to	via	wonder
1	0	0	0	0	1	1	0	1	0	0	1	1	0	1	0	0	0

image source: *Bengfort B. et al. Text Analysis with Python*

Term Frequency-Inverse Document Frequency

- The bag-of-words representations that we have explored so far only describe a document in a standalone fashion, **not taking into account the context of the corpus**.
- A better approach would be to consider the **relative frequency or rareness of tokens in the document against their frequency in other documents**.
- The central insight is that **meaning is most likely encoded in the more rare terms from a document**.

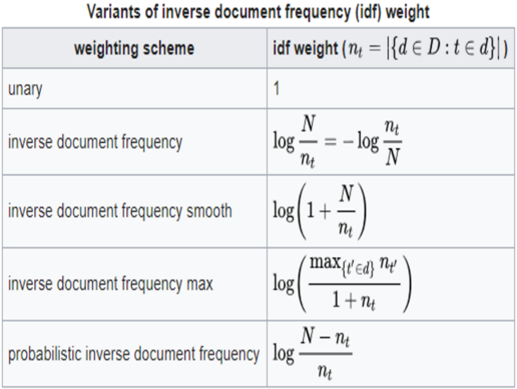
Compute Inverse Document Frequency

- The **inverse document frequency** is a measure of how much information the word provides, that is, whether the term is common or rare across all documents.
- It is the logarithmically scaled inverse fraction of the documents that contain the word, obtained by dividing the total number of documents by the number of documents containing the term, and then taking the logarithm of that quotient:

$$idf(t, D) = \log \frac{N}{|\{d \in D : t \in d\}|} \tag{1}$$

where **the numerator (N) is the total number of documents** in the corpus and **the denominator is the number of documents where the term t appears**.

Compute Inverse Document Frequency

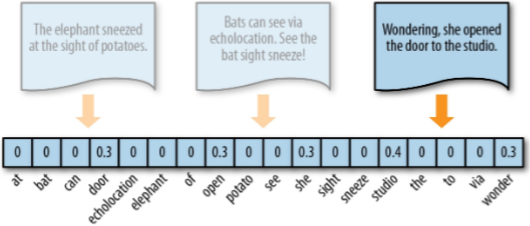


Variants of inverse document frequency (idf) weight

weighting scheme	idf weight ($n_t = \left\|\{d \in D : t \in d\}\right\|$)
unary	1
inverse document frequency	$\log \frac{N}{n_t} = -\log \frac{n_t}{N}$
inverse document frequency smooth	$\log\left(1 + \frac{N}{n_t}\right)$
inverse document frequency max	$\log\left(\frac{\max_{\{t' \in d\}} n_{t'}}{1 + n_t}\right)$
probabilistic inverse document frequency	$\log \frac{N - n_t}{n_t}$

Term Frequency-Inverse Document Frequency

- TF-IDF, term frequency-inverse document frequency, encoding normalizes the frequency of tokens in a document with respect to the rest of the corpus.
- This encoding approach accentuates terms that are very relevant to a specific instance, as shown in Figure, where the token studio has a higher relevance to this document since it only appears there.



The elephant sneezed at the sight of potatoes.

Bats can see via echolocation. See the bat sight sneeze!

Wondering, she opened the door to the studio.

at	bat	can	door	echolocation	elephant	of	open	potato	see	she	sight	sneeze	studio	the	to	via	wonder
0	0	0	0.3	0	0	0	0.3	0	0	0.3	0	0	0.4	0	0	0	0.3

image source: *Bengfort B. et al. Text Analysis with Python*

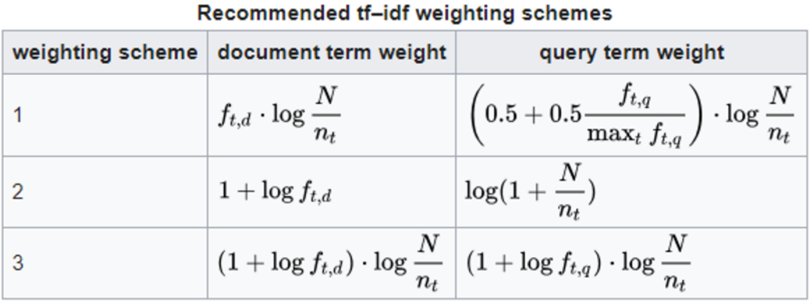
Compute Term Frequency-Inverse Document Frequency

- Then tf-idf is calculated as:

$$tfidf(t, d, D) = tf(t, d) \cdot idf(t, D) \tag{2}$$

- A high weight in tf-idf is reached by a high term frequency (in the given document) and a low document frequency of the term in the whole collection of documents;
- The weights hence tend to filter out common terms.
- Since the ratio inside the idf's log function is always greater than or equal to 1, the value of idf (and tf-idf) is greater than or equal to 0. As a term appears in more documents, the ratio inside the logarithm approaches 1, bringing the idf and tf-idf closer to 0.

Compute Term Frequency-Inverse Document Frequency

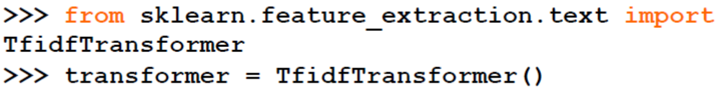


Recommended tf–idf weighting schemes

weighting scheme	document term weight	query term weight
1	$f_{t,d} \cdot \log \frac{N}{n_t}$	$\left(0.5 + 0.5 \frac{f_{t,q}}{\max_t f_{t,q}}\right) \cdot \log \frac{N}{n_t}$
2	$1 + \log f_{t,d}$	$\log(1 + \frac{N}{n_t})$
3	$(1 + \log f_{t,d}) \cdot \log \frac{N}{n_t}$	$(1 + \log f_{t,q}) \cdot \log \frac{N}{n_t}$

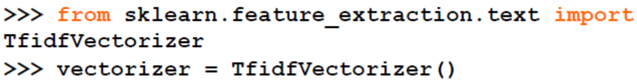
Text Feature Extraction

CountVectorizer: turns a collection of text documents into numerical feature vectors.



```
>>> from sklearn.feature_extraction.text import
TfidfTransformer
>>> transformer = TfidfTransformer()
```

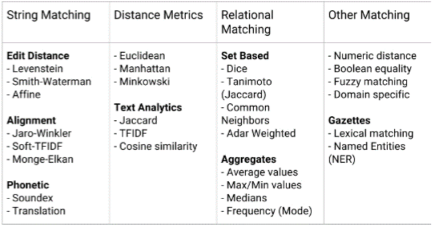
As tf-idf is very often used for text features, there is also another class called TfidfVectorizer that combines all the options of CountVectorizer and TfidfTransformer in a single model



```
>>> from sklearn.feature_extraction.text import
TfidfVectorizer
>>> vectorizer = TfidfVectorizer()
```

Document Similarity

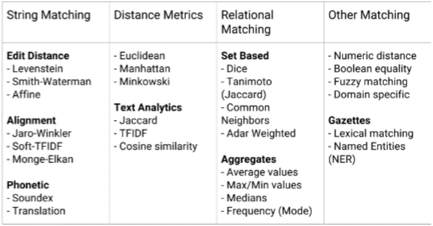
- When you have vectorized your text, we can try to define a distance metric such that documents that are closer together in feature space are more similar.
- There are a number of different measures that can be used to determine document similarity;



String Matching	Distance Metrics	Relational Matching	Other Matching
Edit Distance - Levenstein - Smith-Waterman - Affine **Alignment** - Jaro-Winkler - Soft-TFIDF - Monge-Elkan **Phonetic** - Soundex - Translation	- Euclidean - Manhattan - Minkowski **Text Analytics** - Jaccard - TFIDF - Cosine similarity	**Set Based** - Dice - Tanimoto (Jaccard) - Common Neighbors - Adar Weighted **Aggregates** - Average values - Max/Min values - Medians - Frequency (Mode)	- Numeric distance - Boolean equality - Fuzzy matching - Domain specific **Gazettes** - Lexical matching - Named Entities (NER)

Document Similarity

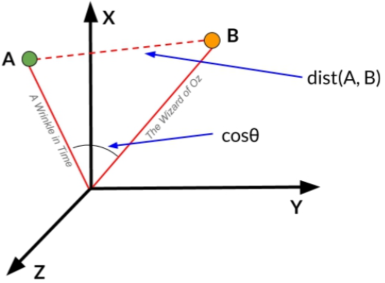
- Fundamentally, each relies on our ability to imagine documents as points in space, where the relative closeness of any two documents is a measure of their similarity.



String Matching	Distance Metrics	Relational Matching	Other Matching
Edit Distance - Levenstein - Smith-Waterman - Affine **Alignment** - Jaro-Winkler - Soft-TFIDF - Monge-Elkan **Phonetic** - Soundex - Translation	- Euclidean - Manhattan - Minkowski **Text Analytics** - Jaccard - TFIDF - Cosine similarity	**Set Based** - Dice - Tanimoto (Jaccard) - Common Neighbors - Adar Weighted **Aggregates** - Average values - Max/Min values - Medians - Frequency (Mode)	- Numeric distance - Boolean equality - Fuzzy matching - Domain specific **Gazettes** - Lexical matching - Named Entities (NER)

Document Similarity

- We can measure vector similarity with cosine distance, using the cosine of the angle between the two vectors to assess the degree to which they share the same orientation.
- In effect, the more parallel any two vectors are, the more similar the documents will be (regardless of their magnitude).



Document Similarity

- Mathematically, Cosine similarity metric measures the cosine of the angle between two n-dimensional vectors projected in a multi-dimensional space. The Cosine similarity of two documents will range from 0 to 1. If the Cosine similarity score is 1, it means two vectors have the same orientation. The value closer to 0 indicates that the two documents have less similarity.
- The mathematical equation of Cosine similarity between two non-zero vectors is:

$$\text{similarity} = \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\| \, \|\mathbf{B}\|} = \frac{\sum\limits_{i=1}^{n} A_i B_i}{\sum\limits_{i=1}^{n} A_i^2 \sum\limits_{i=1}^{n} B_i^2} \tag{3}$$

Word Embedding

What is Word Embedding

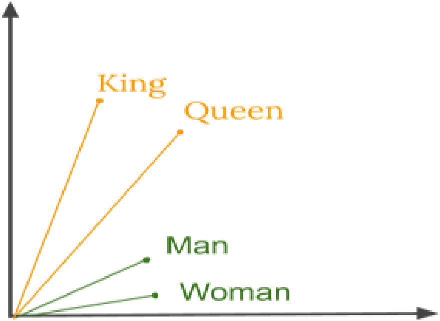
- The different encoding we have discussed so far is arbitrary as **it does not capture any relationship between words**.
- So it's good for document but not for single words!
- Because there is no relationship between the similarity of any two words and the similarity of their encodings, this feature-weight combination is not meaningful.
- Furthermore, **the dimension of the vector space**, being equal to the number of terms in the vocabulary, **tends to increase considerably** as the size of the corpus increases.

What is Word Embedding

- Relationship between words
- For example, we humans understand the words like king and queen, man and woman, tiger and tigress have a certain type of relation between them but how can a computer figure this out?

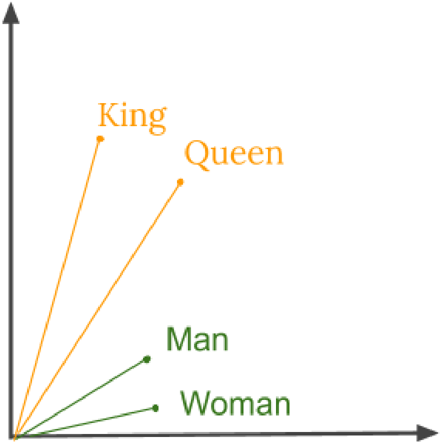
What is Word Embedding

- It should be nice to have representations of text in an n-dimensional space where words that have the same meaning have a similar representation.
- Meaning that two similar words are represented by almost similar vectors that are very closely placed in a vector space.



What is Word Embedding

- Thus when using word embeddings, all individual words are represented as real-valued vectors in a predefined vector space.
- Each word is mapped to one vector and the vector values are learned in a way that resembles a neural network.



The Importance of Context

- The concept of embeddings arises from a branch of Natural Language Processing called - *Distributional Semantics*. It is based on the simple intuition that:
- **Words that occur in similar contexts tend to have similar meanings.**
- In other words, a word meaning is given by the words that it appears frequently with.

Conceptually it involves the mathematical embedding **from space with many dimensions** per word to a **continuous vector space with a much lower dimension**.

Subsection 1

Word2Vec

What is Word2Vec?

- Word2vec is a method to efficiently create word embeddings by using a two-layer neural network.
- It was developed by Tomas Mikolov, et al. at Google in 2013 as a response to make the neural-network-based training of the embedding more efficient and since then has become the de facto standard for developing pre-trained word embedding.
- The input of word2vec is a text corpus and its output is a set of vectors known as feature vectors that represent words in that corpus.

Word2Vec

- Word2vec is not a single algorithm but a combination of two techniques - CBOW(Continuous bag of words) and Skip-gram model.
- Both these are shallow neural networks which map word(s) to the target variable which is also a word(s).
- Both these techniques learn weights which act as word vector representations.

Word2Vec

Continuous Bag of Words (CBOW):

- The CBOW model aims to predict a target word based on the context of surrounding words. It takes a context window of words as input and predicts the target word in the center of the window.
- For example, given the sentence "The quick brown fox jumps over the lazy dog," if the context window size is 2, the input might be ["quick", "brown", "jumps", "over"], and the target word to predict is "fox".

Word2Vec

Skip-gram

- The Skip-gram model works in the reverse way compared to CBOW. It takes a target word as input and tries to predict the context words around it.
- Using the same example sentence, if the target word is "brown," the model might predict ["The", "quick", "fox", "jumps"].

Word2Vec

- The Word2Vec objective function causes the words that have a similar context to have similar embeddings.
- Thus in this vector space, these words are really close. Mathematically, the cosine of the angle (Q) between such vectors should be close to 1, i.e. angle close to 0.

Word2Vec : CBOW approach

- CBOW predicts the probability of a word to occur **given the words surrounding it**. We can consider a single word or a group of words. But for simplicity, we will take a single context word and try to predict a single target word.
- The English language contains almost 1.2 million words, making it impossible to include so many words in our example. So I will consider a small example in which we have only four words i.e. **live**, **home**, **they** and **at**. For simplicity, we will consider that the corpus contains only one sentence, that being, **They live at home**.

Word2Vec : CBOW approach

First, we convert each word into a one-hot encoding form. Also, we will not consider all the words in the sentence but we'll only take certain words that are in a window.

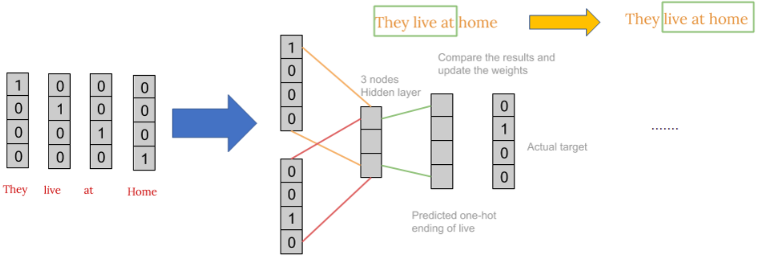


image source: *https://www.mygreatlearning.com/blog/word-embedding/*

Word2Vec : CBOW approach

For example, for a window size equal to three, we only consider three words in a sentence. The middle word is to be predicted and the surrounding two words are fed into the neural network as context. The window is then slide and the process is repeated again.

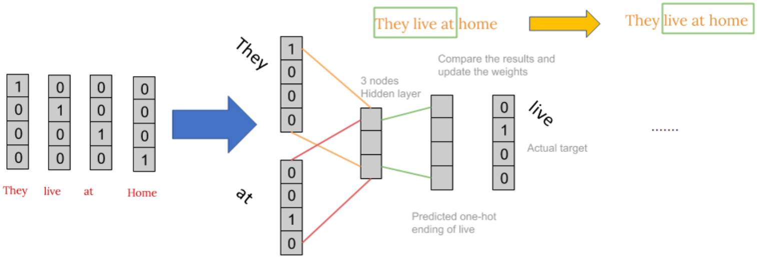


image source: *https://www.mygreatlearning.com/blog/word-embedding/*

Word2Vec : CBOW approach

- Finally, after training the network repeatedly by sliding the window a shown above, we get weights which we use to get the embeddings as shown below.
- Usually, we take a window size of around 8-10 words and have a vector size of 300.

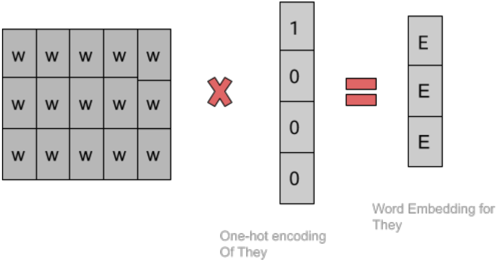


image source: *https://www.mygreatlearning.com/blog/word-embedding/*

Word2Vec : Skip-Gram Model

- The Skip-gram model tries to predict the source context words (surrounding words) given a target word (the centre word)
- The working is conceptually similar to the CBOW, there is just a difference in the architecture of its NN and in the way the weight matrix is generated:

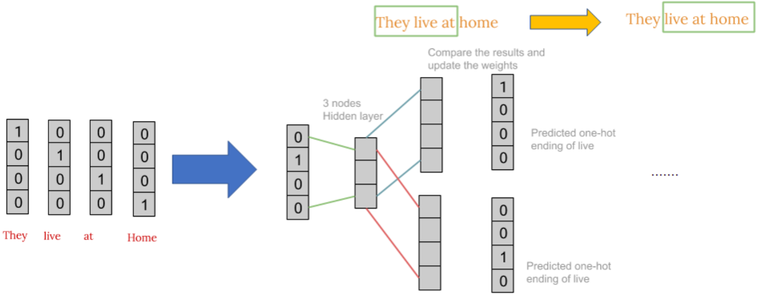


image source: *https://www.mygreatlearning.com/blog/word-embedding/*

Subsection 2

GloVe

Global Vectors for Word Representation

GloVe

- As we have said, word vectors techniques put words to a vector space, where similar words cluster together and different words repel.
- The advantage of GloVe is that, unlike Word2vec, GloVe does not rely just on local statistics (local context information of words), but incorporates global statistics (word co-occurrence) to obtain word vectors.
- In general there is quite a bit of synergy between the GloVe and Word2vec.

Glove : What is a Co-Occurence Matrix

- Generally speaking, a co-occurrence matrix will have specific entities in rows (ER) and columns (EC).
- The purpose of this matrix is to present the number of times each ER appears in the same context as each EC.
- As a consequence, in order to use a co-occurrence matrix, you have to define your entites and the context in which they co-occur.
- In NLP, the most classic approach is to define each entity (ie, lines and columns) as a word present in a text, and the context as a sentence or in the $\pm n$ word window - depends on the application.

GloVe : How to form the Co-occurrence matrix

Let our corpus contain the following three sentences:

- I enjoy flying
- I like NLP
- I like deep learning

Let **window size =1**. This means that context words for each and every word are **1 word to the left and one to the right**.

GloVe : How to form the Co-occurrence matrix

Let our corpus contain the following three sentences:

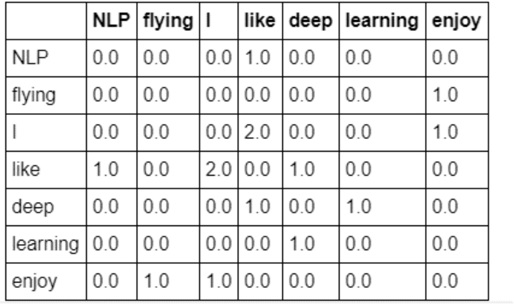
- I enjoy flying
- I like NLP
- I like deep learning

Context words for:

- I → enjoy(1 time), like(2 times)
- enjoy → I (1 time), flying(2 times)
- flying → enjoy(1 time)
- like → I(2 times), NLP(1 time), deep(1 time)
- NLP → like(1 time)
- deep → like(1 time), learning(1 time)
- learning → deep(1 time)

GloVe : How to form the Co-occurrence matrix

Therefore, the resultant co-occurrence matrix A with fixed window size 1 looks like:

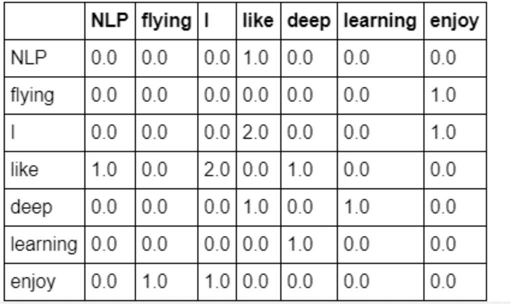


	NLP	flying	I	like	deep	learning	enjoy
NLP	0.0	0.0	0.0	1.0	0.0	0.0	0.0
flying	0.0	0.0	0.0	0.0	0.0	0.0	1.0
I	0.0	0.0	0.0	2.0	0.0	0.0	1.0
like	1.0	0.0	2.0	0.0	1.0	0.0	0.0
deep	0.0	0.0	0.0	1.0	0.0	1.0	0.0
learning	0.0	0.0	0.0	0.0	1.0	0.0	0.0
enjoy	0.0	1.0	1.0	0.0	0.0	0.0	0.0

Co-occurrence matrix A

GloVe : Some Formalism

So, from a formal point of view, given a corpus having V words, the co-occurrence matrix X will be a $V \times V$ matrix, where the i-th row and j-th column of X,X_{ij} denotes how many times word i has co-occurred with word j



	NLP	flying	I	like	deep	learning	enjoy
NLP	0.0	0.0	0.0	1.0	0.0	0.0	0.0
flying	0.0	0.0	0.0	0.0	0.0	0.0	1.0
I	0.0	0.0	0.0	2.0	0.0	0.0	1.0
like	1.0	0.0	2.0	0.0	1.0	0.0	0.0
deep	0.0	0.0	0.0	1.0	0.0	1.0	0.0
learning	0.0	0.0	0.0	0.0	1.0	0.0	0.0
enjoy	0.0	1.0	1.0	0.0	0.0	0.0	0.0

Co-occurrence matrix A

GloVe

- The number of **contexts** is, of course, large, since it is essentially combinatorial in size.
- So then we factorize this matrix to yield a lower-dimensional matrix, where each row now yields a vector representation for each word. In general, this is done by minimizing a **reconstruction loss**.
- This loss tries to find the lower-dimensional representations which can explain most of the variance in the high-dimensional data.

GloVe

From the matrix elements we can define the quantity P_{ik} as the probability of seeing word i and k together, this is computed by dividing the number of times i and k appeared together (X_{ik}) by the total number of times word i appeared in the corpus (X_i):

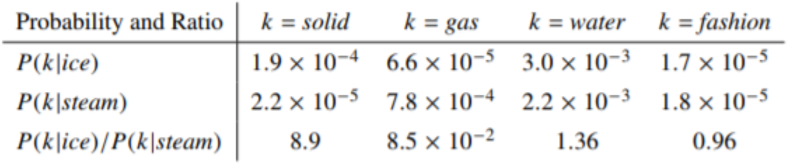
$$P_{ik} = \frac{X_{ik}}{X_i} \tag{4}$$

GloVe

Consider now the ratio

$$\frac{P_{ik}}{P_{jk}} \tag{5}$$

where k is another word in the text



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 various words (Source [1])

Glove

You can see that given two words, i.e. ice and steam, **if the third word** k (also called the *probe word*):

- is very similar to ice but irrelevant to steam (e.g. k=solid), $P_{ik} > P_{jk}$ and P_{ik}/P_{jk} will be very high (> 1),
- is very similar to steam but irrelevant to ice (e.g. k=gas), $P_{ik} < P_{jk}$ and P_{ik}/P_{jk} will be very small (< 1),
- is related or unrelated to either words, then $P_{ik} \sim P_{jk}$ and P_{ik}/P_{jk} will be close to 1

So, if we can find a way to incorporate P_{ik}/P_{jk} in the computation of word vectors we will be achieving the goal of **using global statistics when learning word vectors**.

GloVe : The Model

- Assume that there is a function F which takes in word vectors of i, j and k which outputs the ratio we are interested in:

$$F(w_i, w_j, u_k) = P_{ik}/P_{jk} \tag{6}$$

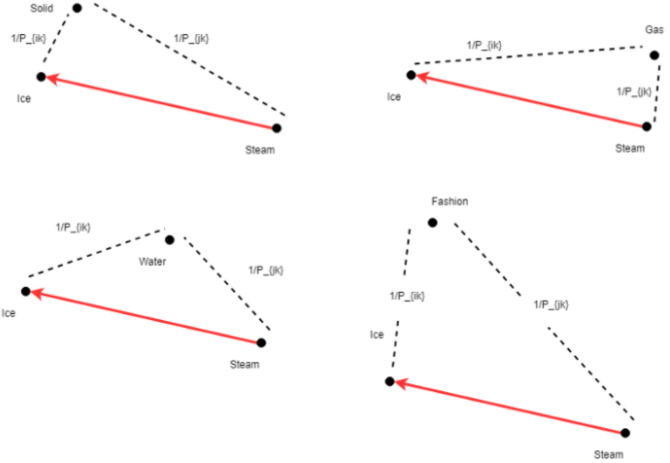
- Word vectors are linear systems. For example, you should perform arithmetic in embedding space, e.g.:

$$\mathbf{w}_{\mathbf{king}} - \mathbf{w}_{\mathbf{male}} + \mathbf{w}_{\mathbf{female}} = \mathbf{w}_{\mathbf{queen}} \tag{7}$$

- Therefore, let us change the above equation to the following:

$$F(w_i - w_j, u_k) = P_{ik}/P_{jk} \tag{8}$$

GloVe : The Model



Behaviour of vector distances to a probe word w.r.t. w_i — w_j

GloVe : The Model

- **Vector to a scalar**
- How do we make LHS a scalar? There is a pretty straight forward answer to this. That is to introduce a transpose and a dot product between the two entities the following way:

$$F((w_i - w_j)^T \cdot u_k) = P_{ik}/P_{jk} \tag{9}$$

- If you assume a word vector as a $D \times 1$ matrix, $(w_i - w_j)^\star$ will be $1 \times D$ shaped which gives a scalar when multiplied with u_k.

GloVe : The Model

if we assume F has a certain property (i.e. homomorphism between additive group and the multiplicative group) which gives,

$$F(w_i^T u_k - w_j^T u_k) = \frac{F(w_i^T u_k)}{F(w_j^T u_k)} = \frac{P_{ik}}{P_{jk}}$$

In other words this particular homomorphism ensures that the subtraction $F(A - B)$ can also be represented as a division $F(A)/F(B)$ and get the same result.

GloVe : The Model

Note that for word-word co-occurrence matrices, the distinction between a word and a context word is arbitrary and that we are free to exchange the two roles. To do so consistently we must not only exchange $w \leftrightarrow u$ but also $X \leftrightarrow X^T$. Our final model is invariant under this relabeling, this is another reason for the particular choice of the homomorphism.

And therefore,

$$\frac{F(w_i^T u_k)}{F(w_j^T u_k)} = \frac{P_{ik}}{P_{jk}} \Rightarrow F(w_i^T u_k) = P_{ik}$$

GloVe : The Model

If we assume F=exp the above homomorphism property is satisfied. Then let us set,

$$\exp(w_i^T u_k) = P_{ik} = \frac{X_{ik}}{X_i}$$

and

$$w_i^T u_k = log(X_{ik}) - log(X_i)$$

Next we note that this equation would exhibit the exchange symmetry if not for the $log(X_i)$ on the right hand side. However this term is independent on k so it can be absorbed into a bias b_i for w_i. Finally adding and addictional bias b_k for u_k restores the symmetry,

$$w_i^T u_k + b_i + b_k = log(X_{ik})$$

GloVe : Defining a Cost Function

- In an ideal setting, where you have perfect word vectors, the above expression will be zero. In other words, that is our goal or objective. So we will be setting the LHS expression as our cost function.

$$J(w_i, w_j) = (w_i^{\star} u_j + b_i^w + b_j^u - log(X_{ij}))^2 \tag{10}$$

- Note that the square makes this a mean square cost function.
- To avoid problem when $X_{ij} = 0$, GloVe use a weighted version of this cost function

$$J(w_i, w_j) = f(X_{ij})(w_i^{\star} u_j + b_i^w + b_j^u - log(X_{ij}))^2 \tag{11}$$

where

$$f(X_{ij}) = (x/x_{max})^{\alpha} \quad \text{if } x < x_{max} \quad \text{else } 0$$

Pre-Trained Embedding Models

- In practice, we use both GloVe and Word2Vec to convert our text into embeddings and both exhibit comparable performances.
- Although in real applications we train our model over Wikipedia text with a window size around 5- 10.
- The number of words in the corpus is around 13 million, hence it takes a huge amount of time and resources to generate these embeddings.
- To avoid this we can use the pre-trained word vectors that are already trained and we can easily use them.
- Here are the links to download pre-trained Word2Vec or GloVe.