M04S02- Distributional Semantics

NLP – Semantic Processing

Introduction

Previous session had the idea of distributional semantics briefly - words that occur in similar contexts have similar meanings. In this session, study **distributional semantics** in detail (also sometimes called vector semantics).

The idea of distributional semantics (implemented through 'word vectors') has been used heavily in semantic processing for a wide variety of applications. In this session, learn about word vectors, word embeddings and the use of word vectors for practical NLP applications.

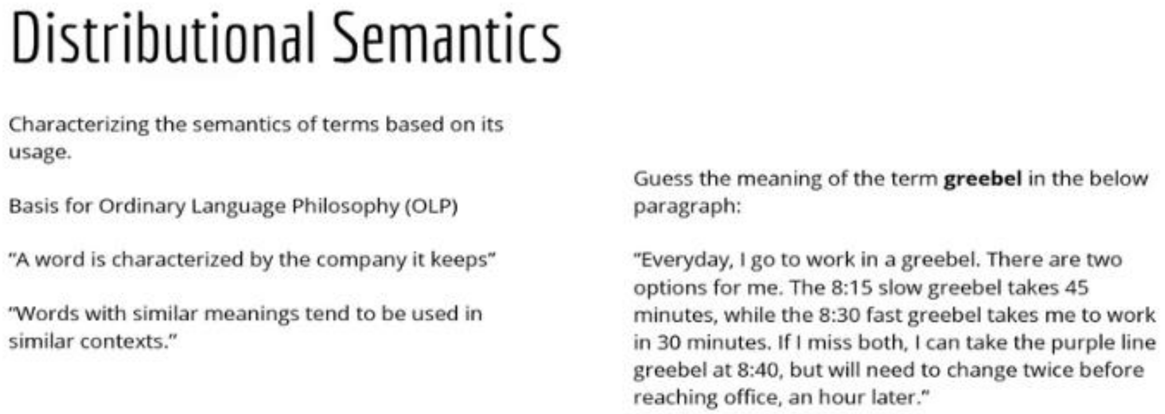
In this session

- Word vectors (occurrence context and co-occurrence matrices)
- Word embeddings (frequency and prediction-based embeddings)
- Frequency-based embeddings: Latent Semantic Analysis (LSA)
- Prediction-based embeddings: Word2Vec
- Using word embeddings in Python for practical applications

Introduction to Distributional Semantics

The English linguist John Firth had said in 1957 -
'You shall know a word by the company it keeps'.

Previous lecture had the concept of **distributional semantics** briefly- words which appear in the same contexts have similar meanings. This simple idea has probably been the most powerful and useful insight in creating semantic processing systems. Now, Learn this idea in detail and learn to use it for various semantic processing applications.



Distributional Semantics

Characterizing the semantics of terms based on its usage.

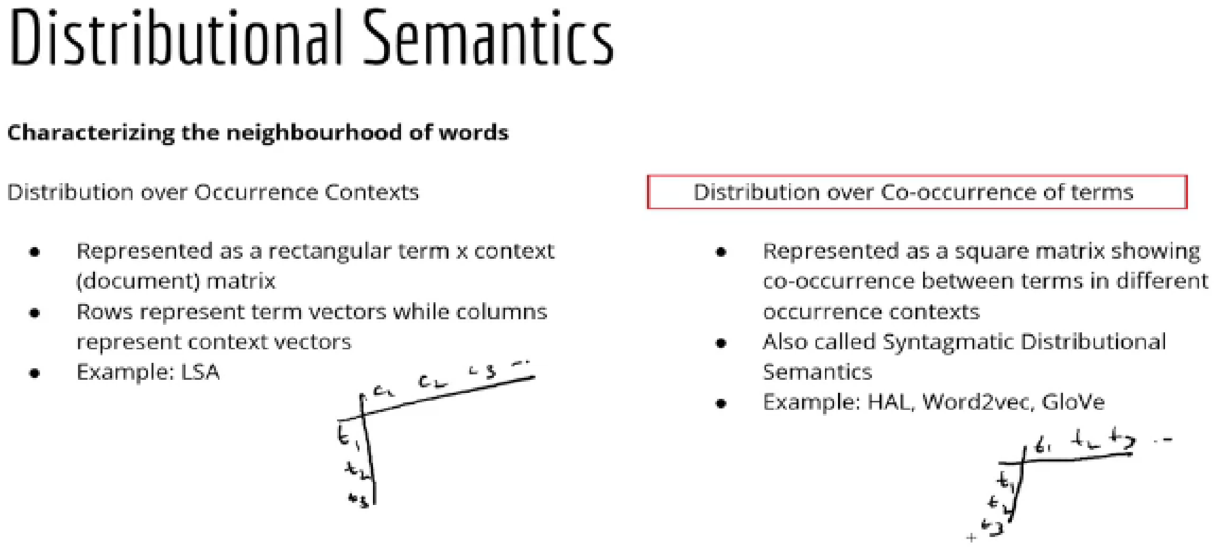
Basis for Ordinary Language Philosophy (OLP)

"A word is characterized by the company it keeps"

"Words with similar meanings tend to be used in similar contexts."

Guess the meaning of the term **greebel** in the below paragraph:

"Everyday, I go to work in a greebel. There are two options for me. The 8:15 slow greebel takes 45 minutes, while the 8:30 fast greebel takes me to work in 30 minutes. If I miss both, I can take the purple line greebel at 8:40, but will need to change twice before reaching office, an hour later."



Distributional Semantics

Characterizing the neighbourhood of words

Distribution over Occurrence Contexts

- Represented as a rectangular term x context (document) matrix
- Rows represent term vectors while columns represent context vectors
- Example: LSA

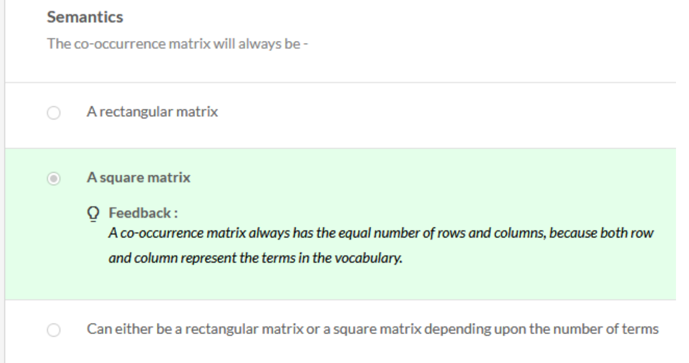
Distribution over Co-occurrence of terms

- Represented as a square matrix showing co-occurrence between terms in different occurrence contexts
- Also called Syntagmatic Distributional Semantics
- Example: HAL, Word2vec, GloVe

The basic idea that we want to use to **quantify the similarity between words** is that words which occur in similar contexts are like each other. To do that, we need to represent words in a format which encapsulates its similarity with other words. For e.g. in such a representation of words, the terms 'greebel' and 'train' will be like each other.

The most commonly used representation of words is using '**word vectors'.** There are two broad techniques to represent words as vectors:

- The term-document **occurrence matrix**, where each row is a term in the vocabulary and each column is a document (such as a webpage, tweet, book etc.)
- The term-term **co-occurrence matrix**, where the ith row and jth column represents the occurrence of the ith word *in the context of* the jth word.



Semantics

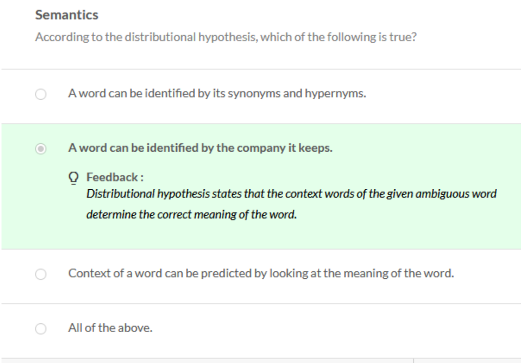
The co-occurrence matrix will always be -

○ A rectangular matrix

● A square matrix

Feedback :
A co-occurrence matrix always has the equal number of rows and columns, because both row and column represent the terms in the vocabulary.

○ Can either be a rectangular matrix or a square matrix depending upon the number of terms



Semantics

According to the distributional hypothesis, which of the following is true?

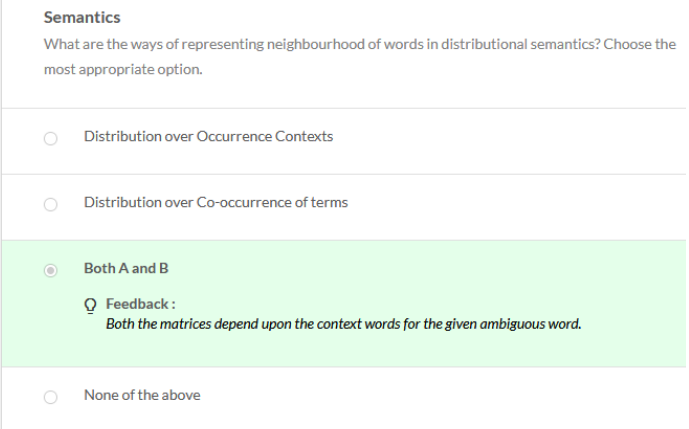
○ A word can be identified by its synonyms and hypernyms.

● A word can be identified by the company it keeps.

Feedback :
Distributional hypothesis states that the context words of the given ambiguous word determine the correct meaning of the word.

○ Context of a word can be predicted by looking at the meaning of the word.

○ All of the above.



Semantics

What are the ways of representing neighbourhood of words in distributional semantics? Choose the most appropriate option.

○ Distribution over Occurrence Contexts

○ Distribution over Co-occurrence of terms

● Both A and B

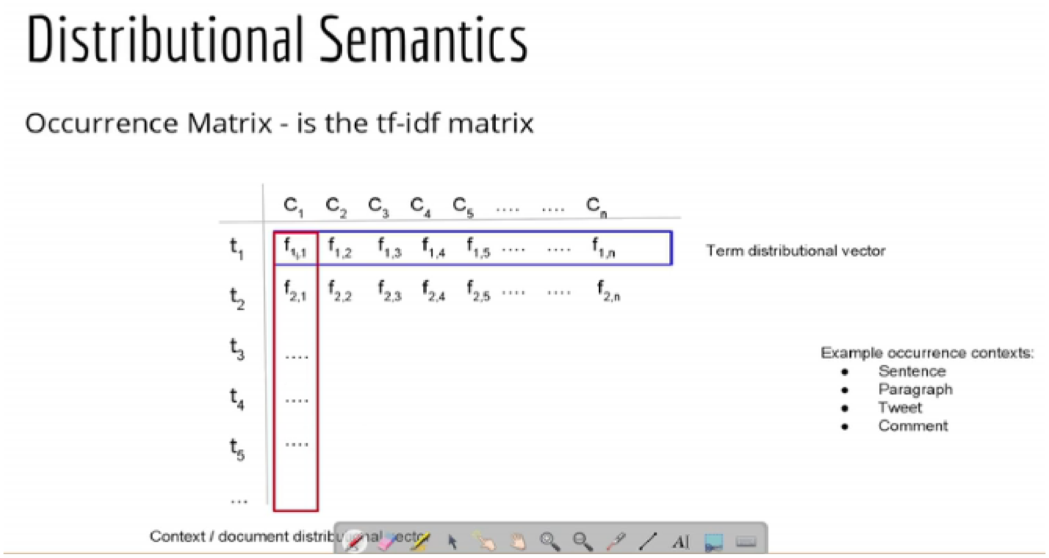
Feedback :
Both the matrices depend upon the context words for the given ambiguous word.

○ None of the above

Occurrence Matrix

There are two broad ways to represent how terms (words) occur in certain contexts

1. The term-occurrence context matrix (or simply the **occurrence matrix**) where each row is a term and each column represent an occurrence context (such as a tweet, a book, a document etc.)
2. the term-term co-occurrence matrix (or the **co-occurrence matrix**) which is a **square matrix** having terms in both rows and columns.



Distributional Semantics

Occurrence Matrix - is the tf-idf matrix

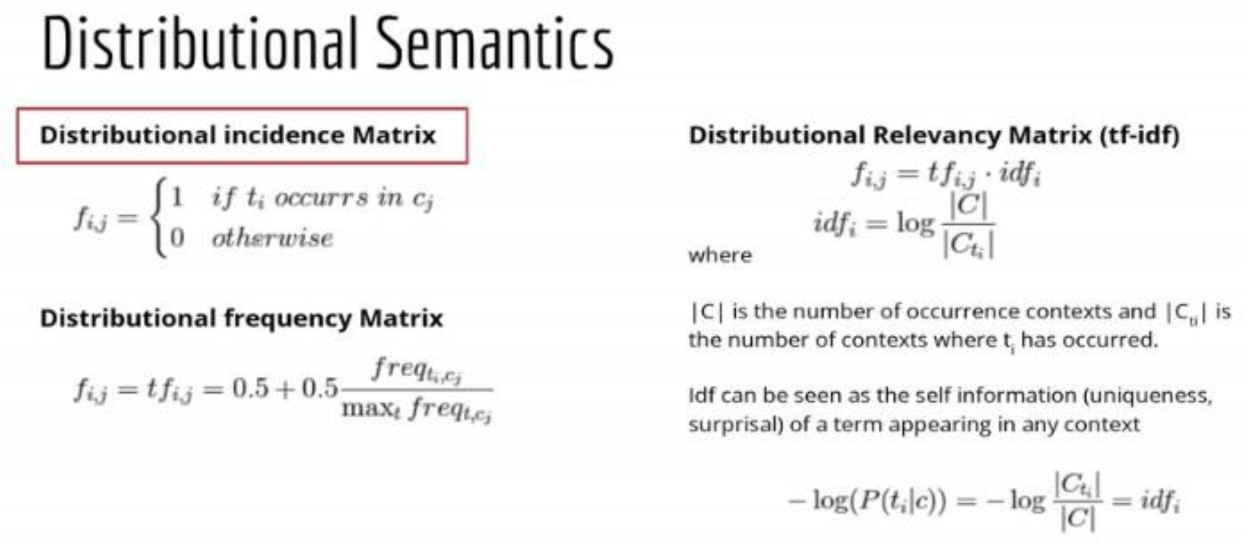
$$\begin{array}{c|cccccccc} & C_1 & C_2 & C_3 & C_4 & C_5 & \cdots & \cdots & C_n \\ \hline t_1 & f_{1,1} & f_{1,2} & f_{1,3} & f_{1,4} & f_{1,5} & \cdots & \cdots & f_{1,n} \\ t_2 & f_{2,1} & f_{2,2} & f_{2,3} & f_{2,4} & f_{2,5} & \cdots & \cdots & f_{2,n} \\ t_3 & \cdots & & & & & & & \\ t_4 & \cdots & & & & & & & \\ t_5 & \cdots & & & & & & & \\ \cdots & & & & & & & & \end{array}$$

Term distributional vector

Example occurrence contexts:

- Sentence
- Paragraph
- Tweet
- Comment

Context / document distribu nal vector



Distributional Semantics

Distributional incidence Matrix

$$f_{i,j} = \begin{cases} 1 & if\ t_i\ occurrs\ in\ c_j \\ 0 & otherwise \end{cases}$$

Distributional frequency Matrix

$$f_{i,j} = tf_{i,j} = 0.5 + 0.5\frac{freq_{t_i,c_j}}{\max_t freq_{t,c_j}}$$

Distributional Relevancy Matrix (tf-idf)

$$f_{i,j} = tf_{i,j} \cdot idf_i$$

$$idf_i = \log\frac{|C|}{|C_{t_i}|}$$

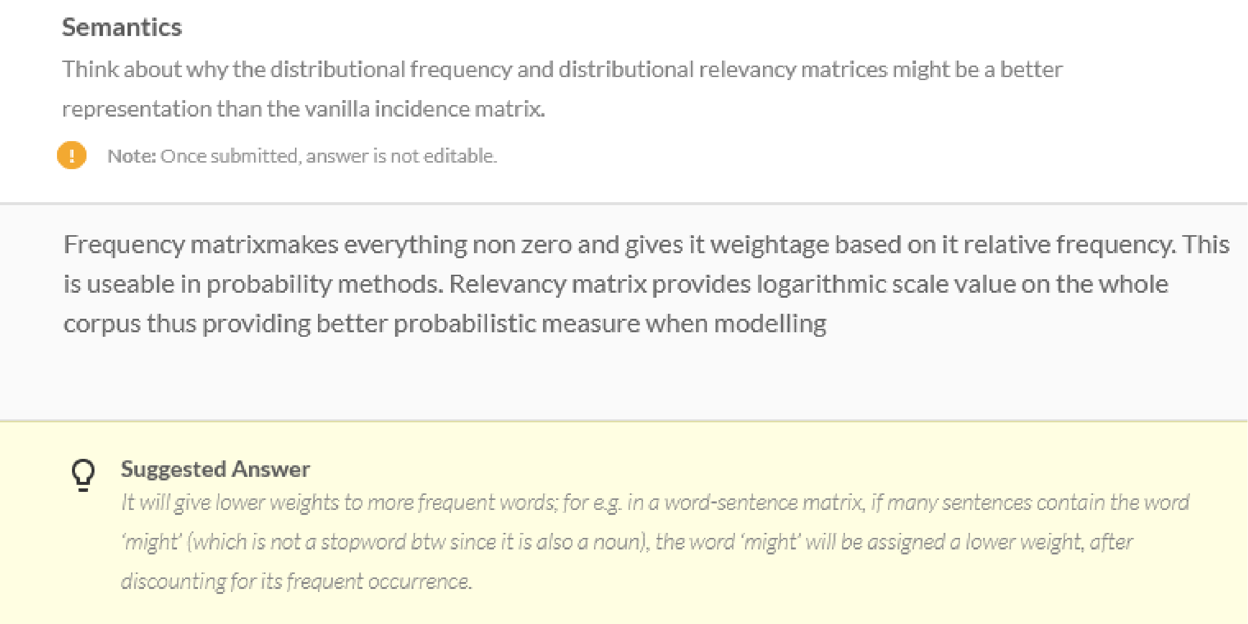
where

|C| is the number of occurrence contexts and $|C_{ti}|$ is the number of contexts where t_i has occurred.

Idf can be seen as the self information (uniqueness, surprisal) of a term appearing in any context

$$-\log(P(t_i|c)) = -\log\frac{|C_{t_i}|}{|C|} = idf_i$$

Each word and document have a corresponding vector representation now - each row is a vector representing a word, while each column is a vector representing a document (or context, such as a tweet, a book etc.). Thus, we can now perform all common vector operations on words and documents.



Semantics

Think about why the distributional frequency and distributional relevancy matrices might be a better representation than the vanilla incidence matrix.

Note: Once submitted, answer is not editable.

Frequency matrixmakes everything non zero and gives it weightage based on it relative frequency. This is useable in probability methods. Relevancy matrix provides logarithmic scale value on the whole corpus thus providing better probabilistic measure when modelling

Suggested Answer

It will give lower weights to more frequent words; for e.g. in a word-sentence matrix, if many sentences contain the word 'might' (which is not a stopword btw since it is also a noun), the word 'might' will be assigned a lower weight, after discounting for its frequent occurrence.

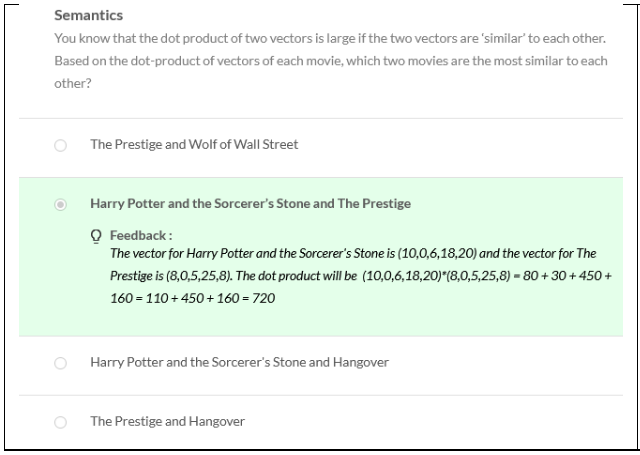
Occurrence matrix is also called a **term-document matrix** since its rows and columns represent terms and documents/occurrence contexts respectively.

Comprehension: The Term-Document Matrix

Consider four documents each of which is a paragraph taken from a movie. Assume that your vocabulary has only the following words: fear, beer, fun, magic, wizard.

The table below summarises the **term-document** matrix, each entry representing the frequency of a term used in a movie:

	Harry Potter and the Sorcerer's Stone	**The Prestige**	**Wolf of Wall Street**	**Hangover**
fear	10	8	2	0
beer	0	0	2	8
fun	6	5	8	8
magic	18	25	0	0
wizard	20	8	0	0



Semantics

You know that the dot product of two vectors is large if the two vectors are 'similar' to each other. Based on the dot-product of vectors of each movie, which two movies are the most similar to each other?

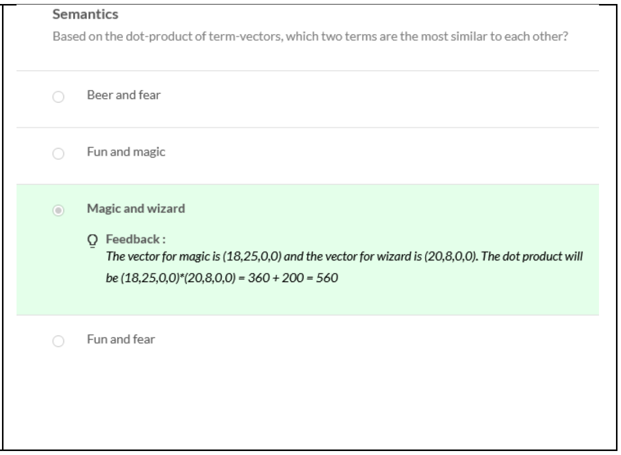
○ The Prestige and Wolf of Wall Street

● **Harry Potter and the Sorcerer's Stone and The Prestige**

Feedback :
The vector for Harry Potter and the Sorcerer's Stone is (10,0,6,18,20) and the vector for The Prestige is (8,0,5,25,8). The dot product will be (10,0,6,18,20)(8,0,5,25,8) = 80 + 30 + 450 + 160 = 110 + 450 + 160 = 720*

○ Harry Potter and the Sorcerer's Stone and Hangover

○ The Prestige and Hangover



Semantics

Based on the dot-product of term-vectors, which two terms are the most similar to each other?

○ Beer and fear

○ Fun and magic

● **Magic and wizard**

Feedback :
The vector for magic is (18,25,0,0) and the vector for wizard is (20,8,0,0). The dot product will be (18,25,0,0)(20,8,0,0) = 360 + 200 = 560*

○ Fun and fear

Term-document matrices (or **occurrence context** matrices) are commonly used in tasks such as **information retrieval**. Two documents having similar words will have similar vectors, where the **similarity between vectors** can be computed using a standard measure such as the **dot product**. Use such representations in tasks where, for example, you want to extract documents like a given document from a large corpus.

A real term-document matrix will be much **larger and sparse**, i.e. it will have as many rows as the size of the vocabulary (typically in tens of thousands) and most cells will have the value 0 (since most words do not occur in most documents).

Using the term-document matrix to compare similarities between terms and documents poses some serious shortcomings such as with **polysemic words**, i.e. words having multiple meanings. For example, the term ‘Java’ is polysemic (coffee, island and programming language), and it will occur in documents on programming, Indonesia and cuisine/beverages.

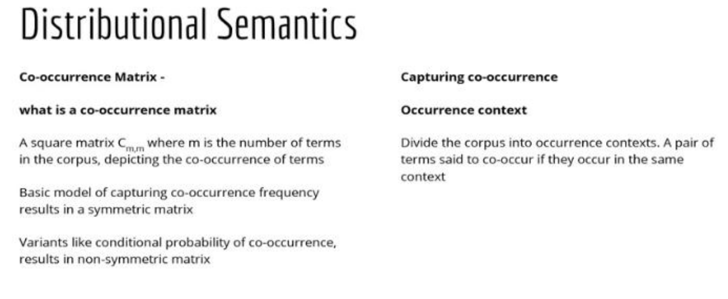
So if you imagine a high dimensional space where each document represents one dimension, the (resultant) vector of the term ‘Java’ will be a vector sum of the term’s occurrence in the dimensions corresponding to all the documents in which 'Java' occurs. Thus, the vector of ‘Java’ will represent some sort of an ‘average meaning’, rather than three distinct meanings (although if the term has a predominant sense, e.g. it occurs much frequently as a programming language than its other senses, this effect is reduced).

Next segment covers an alternate way to generate a distributed representation of words - the term-term **co-occurrence** matrix, where both rows and columns represent a term (word).

Co-occurrence Matrix

Apart from the occurrence context matrix, the other way to create a distributed representation of words is the **term-term co-occurrence** matrix (or simply the **co-occurrence matrix**).

Unlike the occurrence-context matrix, where each column represents a context (such as a document), now the columns also represent a word. Thus, the co-occurrence matrix is also sometimes called the **term-term** matrix.



Distributional Semantics

Co-occurrence Matrix -

what is a co-occurrence matrix

A square matrix $C_{m,m}$ where m is the number of terms in the corpus, depicting the co-occurrence of terms

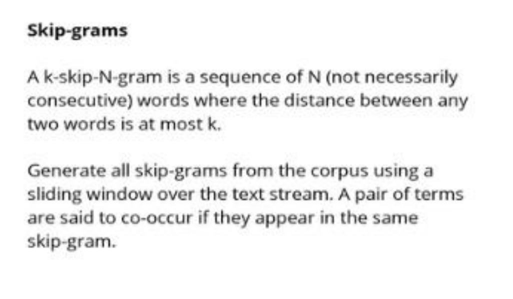
Basic model of capturing co-occurrence frequency results in a symmetric matrix

Variants like conditional probability of co-occurrence, results in non-symmetric matrix

Capturing co-occurrence

Occurrence context

Divide the corpus into occurrence contexts. A pair of terms said to co-occur if they occur in the same context



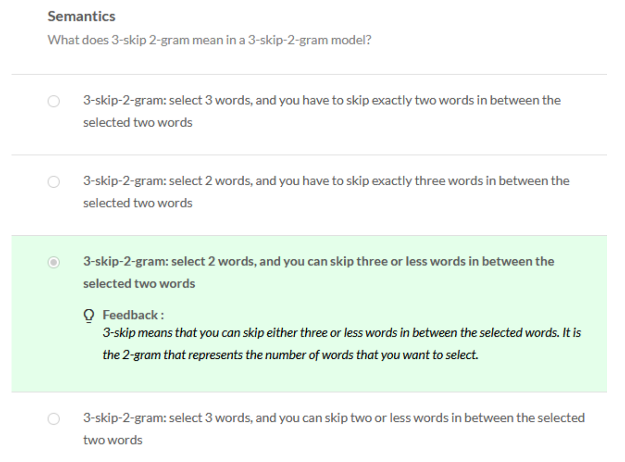
Skip-grams

A k-skip-N-gram is a sequence of N (not necessarily consecutive) words where the distance between any two words is at most k.

Generate all skip-grams from the corpus using a sliding window over the text stream. A pair of terms are said to co-occur if they appear in the same skip-gram.

There are two ways of creating a co-occurrence matrix:

1. **Using the occurrence context (e.g. a sentence):**
   - Each sentence is represented as a context (there can be other definitions as well). If two terms occur in the same context, they are said to have occurred in the same occurrence context.
2. **Skip-grams (x-skip-n-grams):**
   - A sliding window will include the (x+n) words. This window will serve as the context now. Terms that co-occur within this context are said to have co-occurred.



Semantics

What does 3-skip 2-gram mean in a 3-skip-2-gram model?

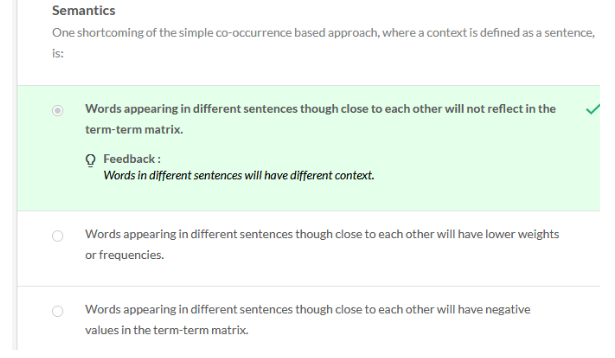
○ 3-skip-2-gram: select 3 words, and you have to skip exactly two words in between the selected two words

○ 3-skip-2-gram: select 2 words, and you have to skip exactly three words in between the selected two words

● **3-skip-2-gram: select 2 words, and you can skip three or less words in between the selected two words**

Feedback :
3-skip means that you can skip either three or less words in between the selected words. It is the 2-gram that represents the number of words that you want to select.

○ 3-skip-2-gram: select 3 words, and you can skip two or less words in between the selected two words



Semantics

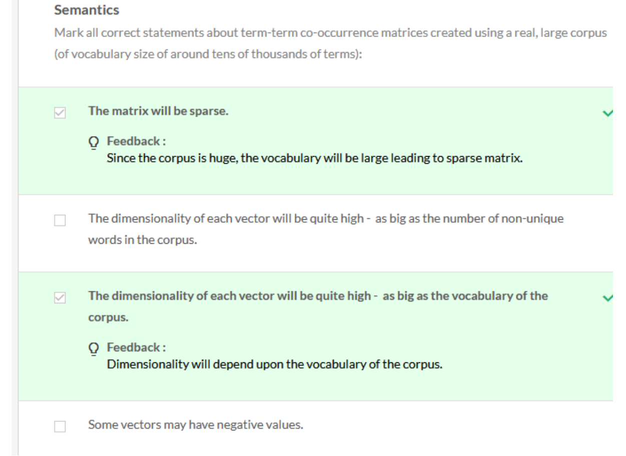
One shortcoming of the simple co-occurrence based approach, where a context is defined as a sentence, is:

● **Words appearing in different sentences though close to each other will not reflect in the term-term matrix.** ✓

Feedback :
Words in different sentences will have different context.

○ Words appearing in different sentences though close to each other will have lower weights or frequencies.

○ Words appearing in different sentences though close to each other will have negative values in the term-term matrix.



Semantics

Mark all correct statements about term-term co-occurrence matrices created using a real, large corpus (of vocabulary size of around tens of thousands of terms):

☑ **The matrix will be sparse.** ✓

Feedback :
Since the corpus is huge, the vocabulary will be large leading to sparse matrix.

□ The dimensionality of each vector will be quite high - as big as the number of non-unique words in the corpus.

☑ **The dimensionality of each vector will be quite high - as big as the vocabulary of the corpus.** ✓

Feedback :
Dimensionality will depend upon the vocabulary of the corpus.

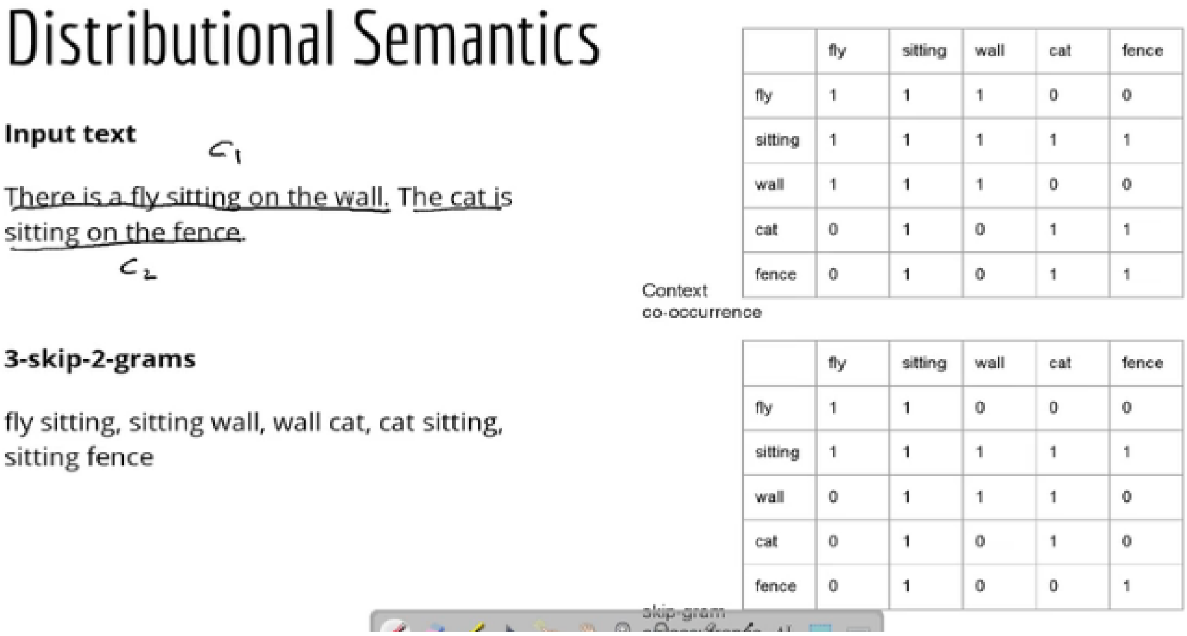
□ Some vectors may have negative values.

Word Vectors

Two approaches to create a co-occurrence matrix:

- Occurrence context matrix
- x-skip-n-grams (more generally, skip-grams)

Now learn each of these approaches in detail. c1 and c2 represent context 1 and context 2 respectively.



Distributional Semantics

Input text

c_1

There is a fly sitting on the wall. The cat is sitting on the fence.

c_2

3-skip-2-grams

fly sitting, sitting wall, wall cat, cat sitting, sitting fence

	fly	sitting	wall	cat	fence
fly	1	1	1	0	0
sitting	1	1	1	1	1
wall	1	1	1	0	0
cat	0	1	0	1	1
fence	0	1	0	1	1

Context co-occurrence

	fly	sitting	wall	cat	fence
fly	1	1	0	0	0
sitting	1	1	1	1	1
wall	0	1	1	1	0
cat	0	1	0	1	0
fence	0	1	0	0	1

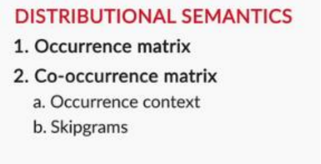
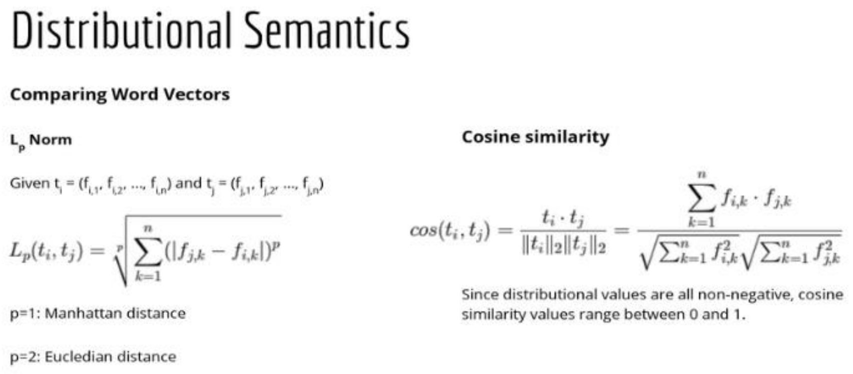
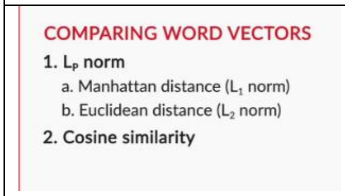
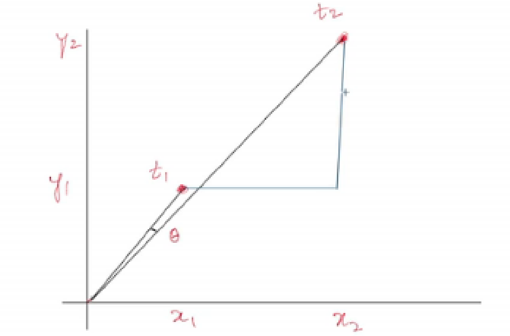
skip-gram

It has been assumed that a word occurs in its own context. So, all the diagonal elements in the matrix are 1.

Two approaches to create the term-term co-occurrence matrix:

1. **Occurrence context**:
   - A context can be defined as, for e.g., an entire sentence. Two words are said to co-occur if they appear in the same sentence.
2. **Skipgrams**:
   - 3-skip means that the two words that are being considered should have at max 3 words in between them, and 2-gram means that we are going to select two words from the window.

For term-document matrices, we can use the dot product of two vectors to compare the similarities between vectors. Let's now look at some other similarity metrics we can use.

 DISTRIBUTIONAL SEMANTICS 1. Occurrence matrix 2. Co-occurrence matrix a. Occurrence context b. Skipgrams	 Distributional Semantics **Comparing Word Vectors** **L_p Norm** Given $t_i = (f_{i,1}, f_{i,2}, \dots, f_{i,n})$ and $t_j = (f_{j,1}, f_{j,2}, \dots, f_{j,n})$ $L_p(t_i, t_j) = \sqrt[p]{\sum_{k=1}^{n} (\lvert f_{j,k} - f_{i,k} \rvert)^p}$ p=1: Manhattan distance p=2: Eucledian distance **Cosine similarity** $cos(t_i, t_j) = \frac{t_i \cdot t_j}{\lVert t_i \rVert_2 \lVert t_j \rVert_2} = \frac{\sum_{k=1}^{n} f_{i,k} \cdot f_{j,k}}{\sqrt{\sum_{k=1}^{n} f_{i,k}^2} \sqrt{\sum_{k=1}^{n} f_{j,k}^2}}$ Since distributional values are all non-negative, cosine similarity values range between 0 and 1.
 COMPARING WORD VECTORS 1. L_p norm a. Manhattan distance (L_1 norm) b. Euclidean distance (L_2 norm) 2. Cosine similarity	 t_2 y_2 y_1 t_1 θ x_1 x_2

Comprehension - Word Vectors

Take following paragraph from the book Harry Potter and the Sorcerer's Stone:

"Sorry, he grunted, as the tiny old man stumbled and almost fell. It was a few seconds before Mr Dursley realized that the man was wearing a violet cloak. He didn't seem at all upset at being almost knocked to the ground."

Let's assume that our vocabulary only contains a few words as listed below. After removing the stop words and punctuations and retaining only the words in our vocabulary, the paragraph becomes:

Man stumbled seconds Dursley man cloak upset knocked ground

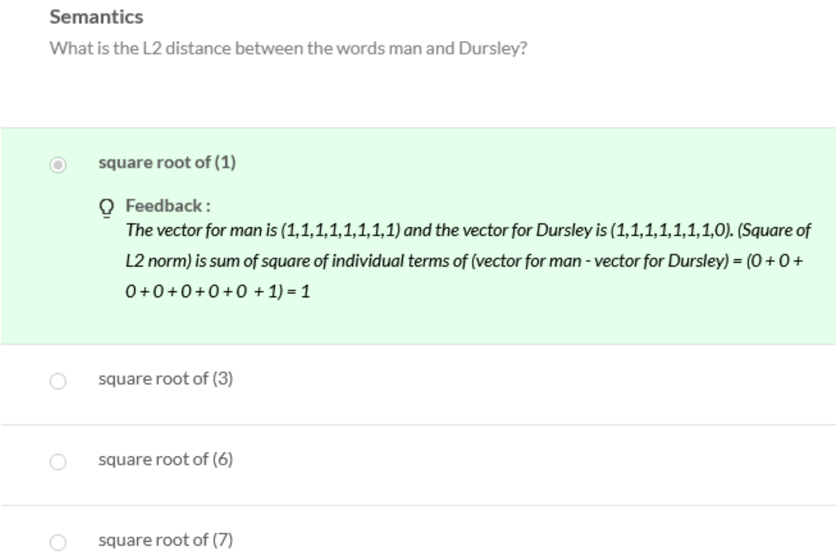
Create a co-occurrence matrix using this paragraph using the 3-skip-2-gram technique and answer the following questions (choose a similarity metric of your choice).

The vocabulary would be:
(man, stumbled, seconds, Dursley, cloak, upset, knocked, ground)

The co-occurrence pairs that you get would be (the positions of left and right words do not matter, they can be switched as well):

	Man	Stumbled	Seconds	Dursley	Cloak	Upset	Knocked	Ground
Man	1	1	1	1	1	1	1	1
Stumbled	1	1	1	1	1	0	0	0
Seconds	1	1	1	1	1	1	0	0
Dursley	1	1	1	1	1	1	1	0
Cloak	1	1	1	1	1	1	1	1

Upset	1	0	1	1	1	1	1	1
Knocked	1	0	0	1	1	1	1	1
Ground	1	0	0	0	1	1	1	1



Semantics

What is the L2 distance between the words man and Dursley?

● **square root of (1)**

Feedback :

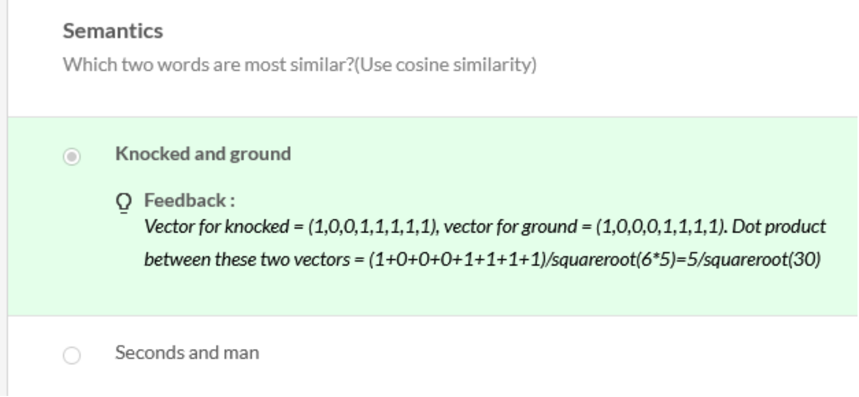
The vector for man is (1,1,1,1,1,1,1,1) and the vector for Dursley is (1,1,1,1,1,1,1,0). (Square of L2 norm) is sum of square of individual terms of (vector for man - vector for Dursley) = (0 + 0 + 0 + 0 + 0 + 0 + 0 + 1) = 1

○ square root of (3)

○ square root of (6)

○ square root of (7)

X`



Semantics

Which two words are most similar?(Use cosine similarity)

● **Knocked and ground**

Feedback :

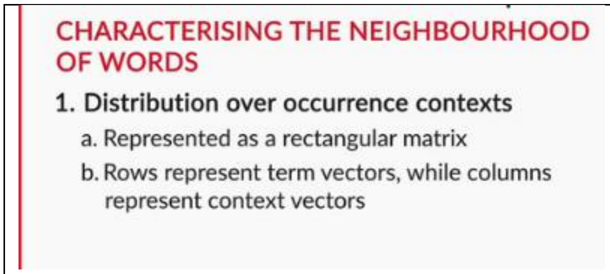
*Vector for knocked = (1,0,0,1,1,1,1,1), vector for ground = (1,0,0,0,1,1,1,1). Dot product between these two vectors = (1+0+0+0+1+1+1+1)/squareroot(6*5)=5/squareroot(30)*

○ Seconds and man

Word Embeddings

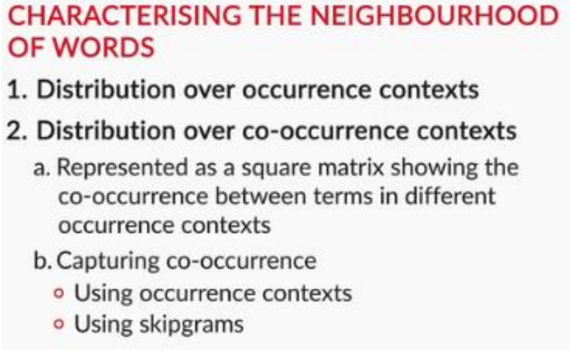
Previous segment created different kinds of word vectors. Occurrence and co-occurrence matrices have really large dimensions (equal to the size of the vocabulary V). This is a problem because working with such huge matrices make them almost impractical to use.

Let’s first summarise all learnt about word vectors.



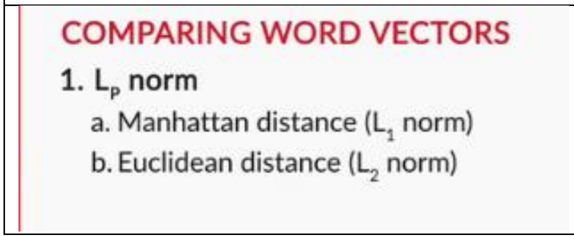
CHARACTERISING THE NEIGHBOURHOOD OF WORDS

1. Distribution over occurrence contexts
   a. Represented as a rectangular matrix
   b. Rows represent term vectors, while columns represent context vectors



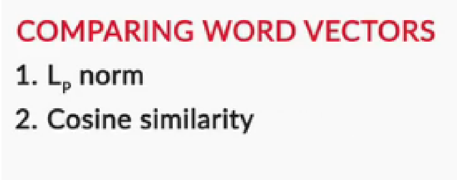
CHARACTERISING THE NEIGHBOURHOOD OF WORDS

1. Distribution over occurrence contexts
2. Distribution over co-occurrence contexts
   a. Represented as a square matrix showing the co-occurrence between terms in different occurrence contexts
   b. Capturing co-occurrence
      - Using occurrence contexts
      - Using skipgrams



COMPARING WORD VECTORS

1. L_p norm
   a. Manhattan distance (L_1 norm)
   b. Euclidean distance (L_2 norm)



COMPARING WORD VECTORS

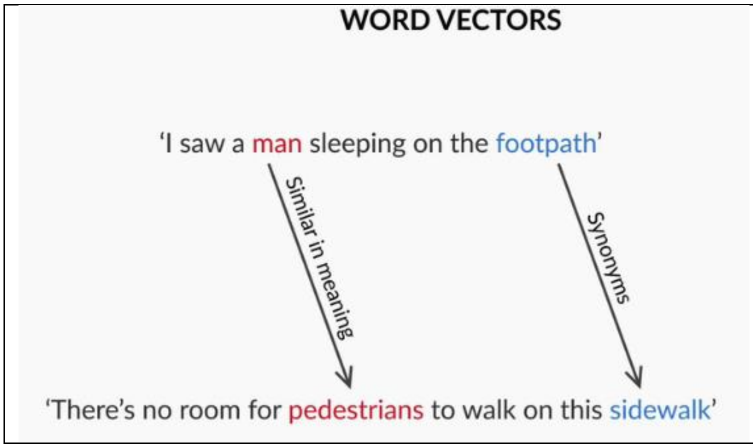
1. L_p norm
2. Cosine similarity

Occurrence and co-occurrence matrices are **sparse** (really sparse!) and **high-dimensional**. Talking about high dimensionality - why not reduce the dimensionality using matrix factorization techniques such as SVD etc.?

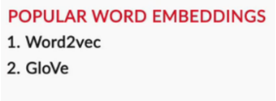
This is what **word embeddings** aim to do. Word embeddings are a compressed, **low dimensional** version of the mammoth-sized occurrence and co-occurrence matrices.

Each row (i.e word) has a much **shorter vector** (of size say 100, rather than tens of thousands) and is **dense**, i.e. most entries are non-zero (and you still get to retain most of the information that a full-size sparse matrix would hold).

Let's see how to create such dense word vectors.

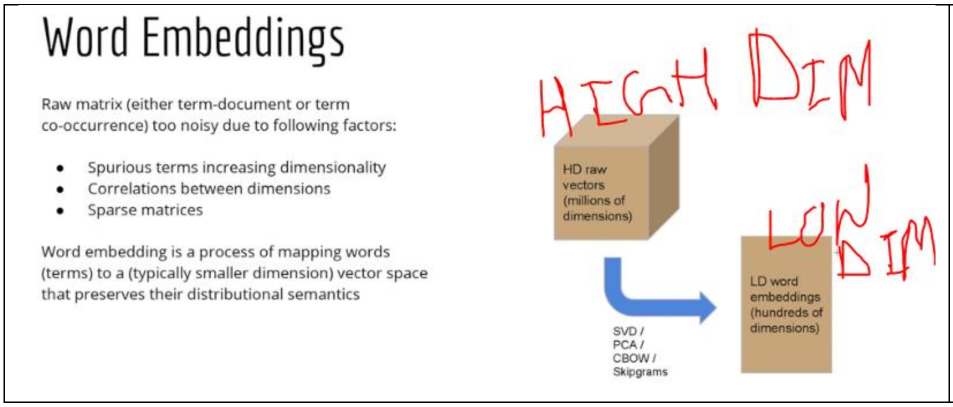


WORD VECTORS
‘I saw a man sleeping on the footpath’
Similar in meaning
Synonyms
‘There’s no room for pedestrians to walk on this sidewalk’



POPULAR WORD EMBEDDINGS

1. Word2vec
2. GloVe



Word Embeddings

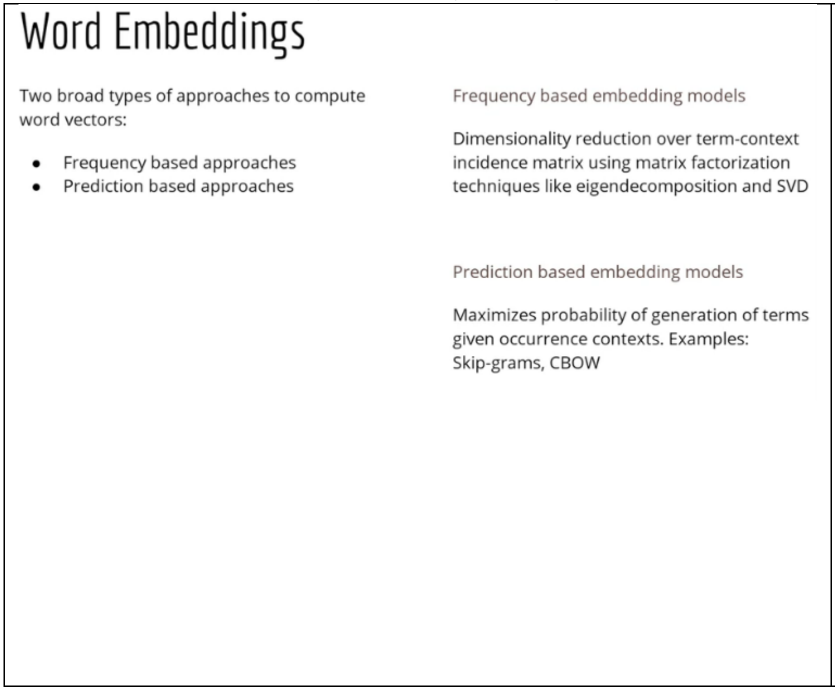
Raw matrix (either term-document or term co-occurrence) too noisy due to following factors:

- Spurious terms increasing dimensionality
- Correlations between dimensions
- Sparse matrices

Word embedding is a process of mapping words (terms) to a (typically smaller dimension) vector space that preserves their distributional semantics

HIGH DIM
HD raw vectors (millions of dimensions)
LOW DIM
LD word embeddings (hundreds of dimensions)
SVD / PCA / CBOW / Skipgrams

What are the different ways in which you can generate word embeddings?



Word Embeddings

Two broad types of approaches to compute word vectors:

- Frequency based approaches
- Prediction based approaches

Frequency based embedding models

Dimensionality reduction over term-context incidence matrix using matrix factorization techniques like eigendecomposition and SVD

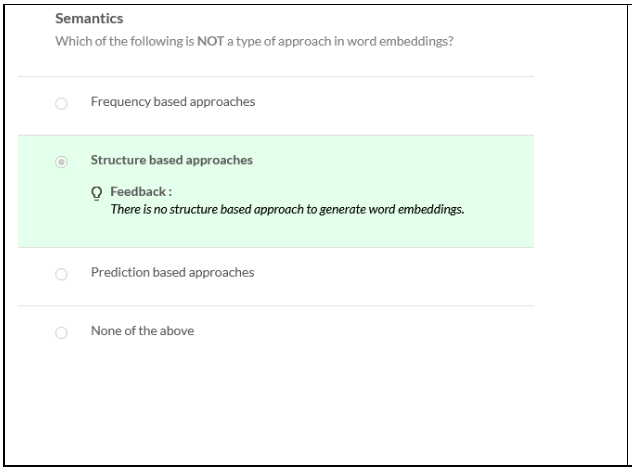
Prediction based embedding models

Maximizes probability of generation of terms given occurrence contexts. Examples: Skip-grams, CBOW

Word embeddings can be generated using two broad approaches:

Frequency-based approach: Reduce term-document matrix (which can as well be a tf-idf, incidence matrix etc.) using a dimensionality reduction technique such as SVD

Prediction based approach: Input is a single word (or a combination of words) and output is a combination of context words (or a single word). A shallow neural network learns the embeddings such that the output words can be predicted using the input words.



Semantics

Which of the following is **NOT** a type of approach in word embeddings?

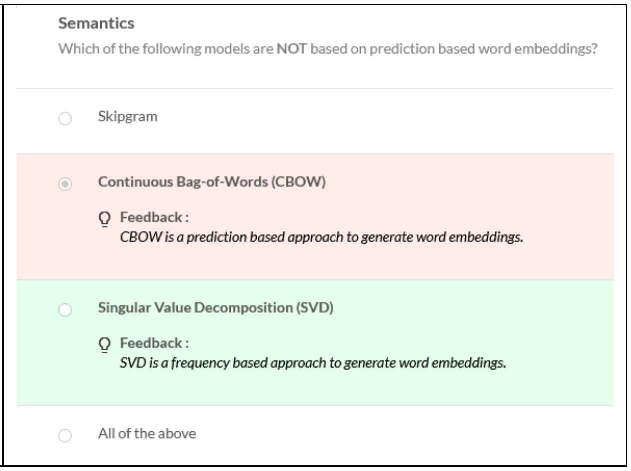
○ Frequency based approaches

● Structure based approaches

Feedback :
There is no structure based approach to generate word embeddings.

○ Prediction based approaches

○ None of the above



Semantics

Which of the following models are **NOT** based on prediction based word embeddings?

○ Skipgram

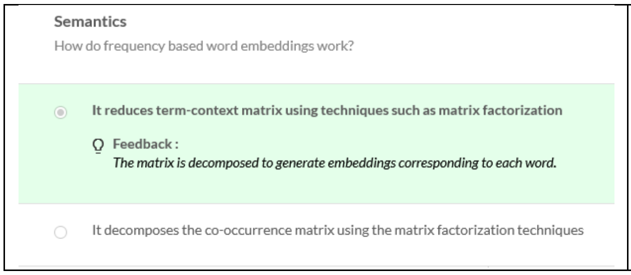
● Continuous Bag-of-Words (CBOW)

Feedback :
CBOW is a prediction based approach to generate word embeddings.

○ Singular Value Decomposition (SVD)

Feedback :
SVD is a frequency based approach to generate word embeddings.

○ All of the above

 **Semantics** How do frequency based word embeddings work? ● **It reduces term-context matrix using techniques such as matrix factorization** **Feedback :** *The matrix is decomposed to generate embeddings corresponding to each word.* ○ It decomposes the co-occurrence matrix using the matrix factorization techniques	

Latent Semantic Analysis (LSA)

A **frequency-based approach** to generate word embeddings**.**

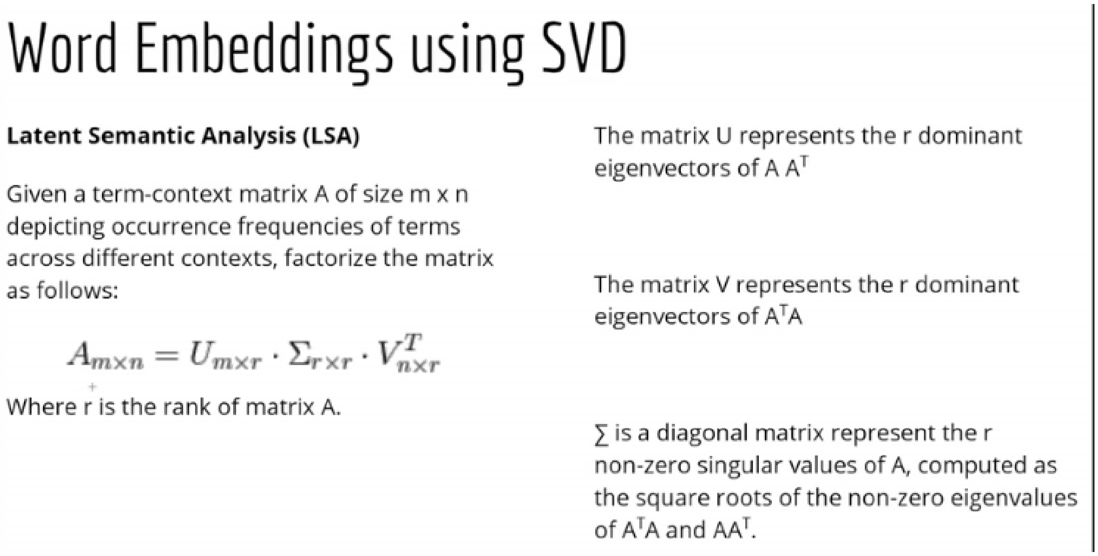
Latent Semantic Analysis (LSA) uses **Singular Value Decomposition (SVD)** to reduce the dimensionality of the matrix.

In LSA, a noisy higher dimensional vector of a word is projected onto a lower dimensional space. The lower dimensional space is a much richer representation of the semantics of the word.

LSA is widely used in processing large sets of documents for various purposes such as **document clustering** and **classification** (in the lower dimensional space), comparing the similarity between documents (e.g. recommending similar books to what a user has liked), finding relations between terms (such as synonymy and polysemy) etc.

Apart from its many advantages, LSA has some **drawbacks** as well. One is that the resulting dimensions are not interpretable (the typical disadvantage of any matrix factorisation-based technique such as PCA). Also, LSA cannot deal with issues such as polysemy. For e.g. we had mentioned earlier that the term ‘Java’ has three senses, and the representation of the term in the lower dimensional space will represent some sort of an ‘average meaning’ of the term rather than three different meanings.

However, the convenience offered by LSA probably outweighs its disadvantages, and thus, it is a commonly used technique in semantic processing.



Word Embeddings using SVD

Latent Semantic Analysis (LSA)

Given a term-context matrix A of size m x n depicting occurrence frequencies of terms across different contexts, factorize the matrix as follows:

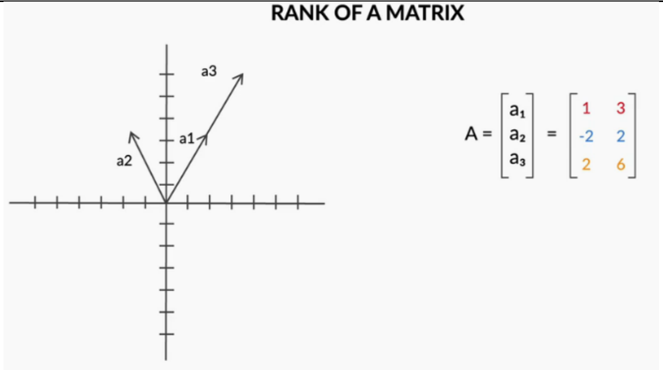
$$A_{m\times n} = U_{m\times r} \cdot \Sigma_{r\times r} \cdot V^{T}_{n\times r}$$

Where r is the rank of matrix A.

The matrix U represents the r dominant eigenvectors of A A^T

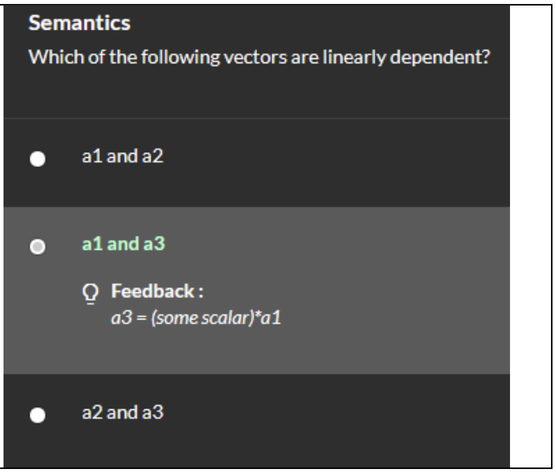
The matrix V represents the r dominant eigenvectors of A^TA

∑ is a diagonal matrix represent the r non-zero singular values of A, computed as the square roots of the non-zero eigenvalues of A^TA and AA^T.



RANK OF A MATRIX

$$A = \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix} = \begin{bmatrix} 1 & 3 \\ -2 & 2 \\ 2 & 6 \end{bmatrix}$$



Semantics

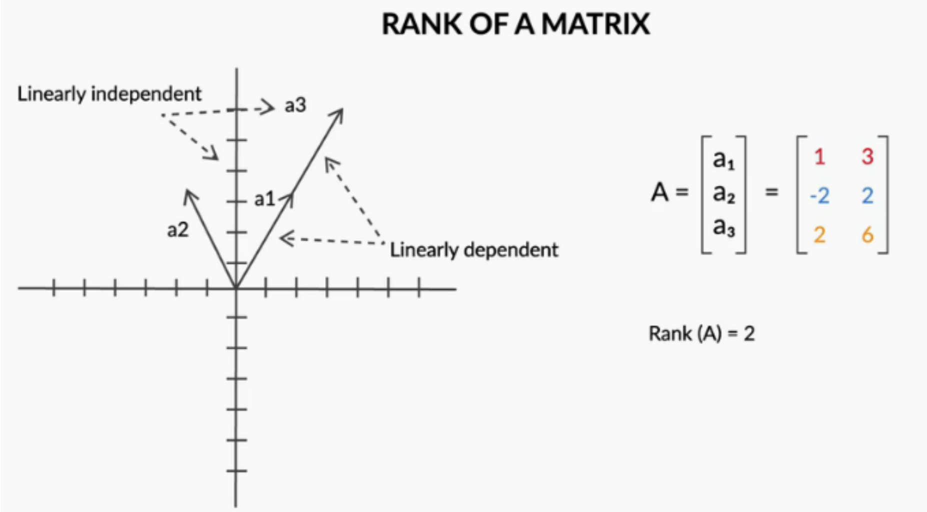
Which of the following vectors are linearly dependent?

○ a1 and a2

● a1 and a3

Feedback :
*a3 = (some scalar)*a1*

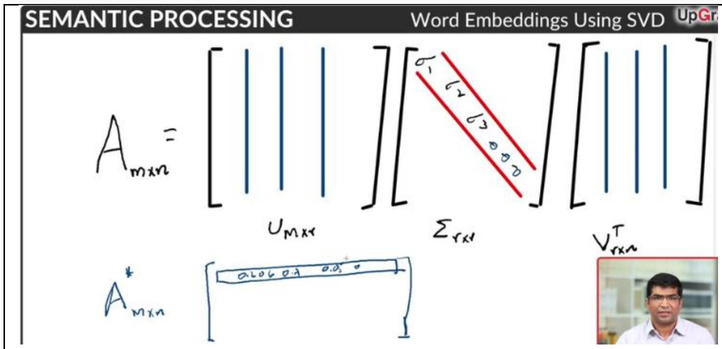
○ a2 and a3



RANK OF A MATRIX

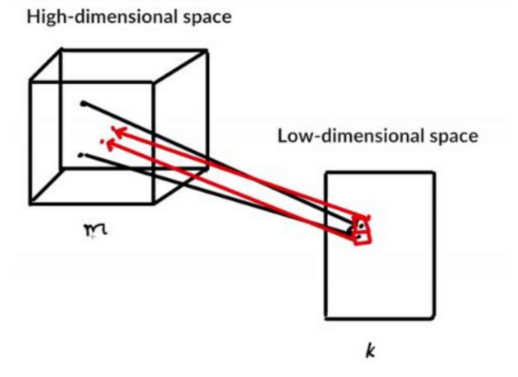
$$A = \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix} = \begin{bmatrix} 1 & 3 \\ -2 & 2 \\ 2 & 6 \end{bmatrix}$$

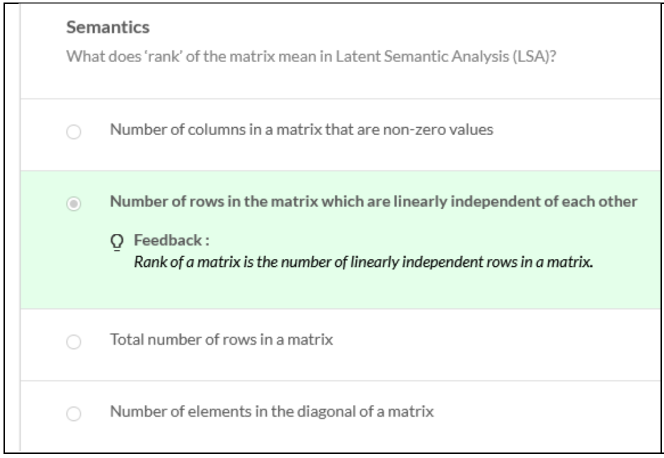
Rank (A) = 2



SEMANTIC PROCESSING

Word Embeddings Using SVD





Semantics

What does 'rank' of the matrix mean in Latent Semantic Analysis (LSA)?

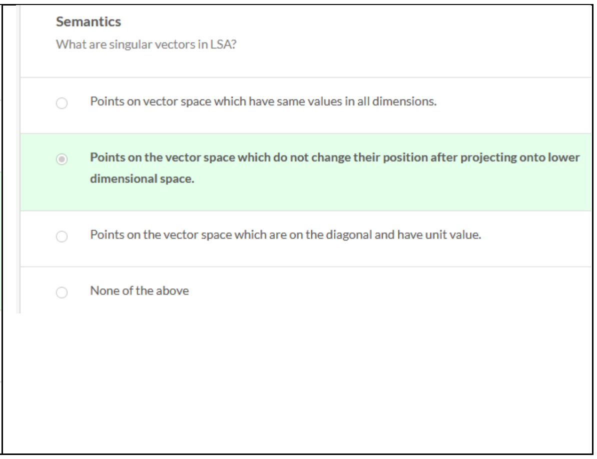
○ Number of columns in a matrix that are non-zero values

● Number of rows in the matrix which are linearly independent of each other

Feedback :
Rank of a matrix is the number of linearly independent rows in a matrix.

○ Total number of rows in a matrix

○ Number of elements in the diagonal of a matrix



Semantics

What are singular vectors in LSA?

○ Points on vector space which have same values in all dimensions.

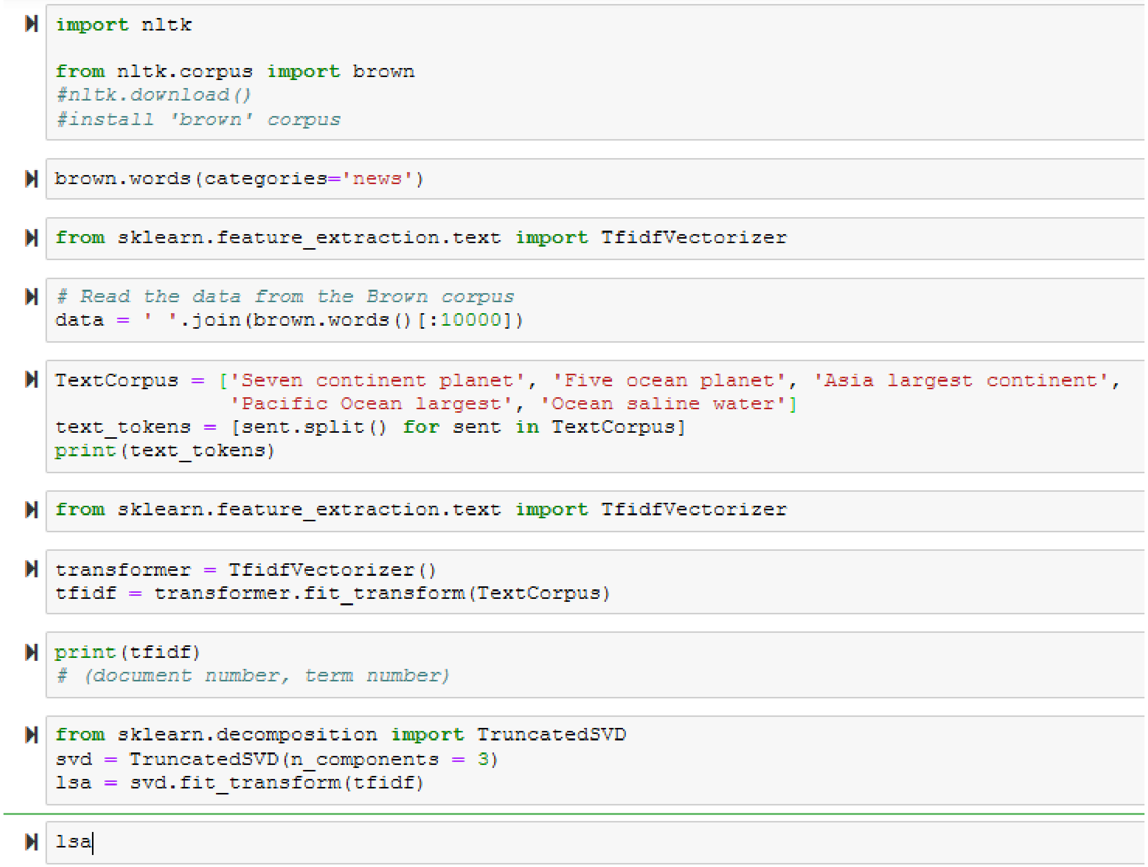
● Points on the vector space which do not change their position after projecting onto lower dimensional space.

○ Points on the vector space which are on the diagonal and have unit value.

○ None of the above

Let's now learn to implement LSA in Python.

The task here is to decrease the number of dimensions of the document-term matrix, i.e. reduce the #documents x #terms matrix to #documents x #LSA_dimensions. So, each document vector will now be represented by a lower number of dimensions.



```
import nltk

from nltk.corpus import brown
#nltk.download()
#install 'brown' corpus
```

```
brown.words(categories='news')
```

```
from sklearn.feature_extraction.text import TfidfVectorizer
```

```
# Read the data from the Brown corpus
data = ' '.join(brown.words()[:10000])
```

```
TextCorpus = ['Seven continent planet', 'Five ocean planet', 'Asia largest continent',
              'Pacific Ocean largest', 'Ocean saline water']
text_tokens = [sent.split() for sent in TextCorpus]
print(text_tokens)
```

```
from sklearn.feature_extraction.text import TfidfVectorizer
```

```
transformer = TfidfVectorizer()
tfidf = transformer.fit_transform(TextCorpus)
```

```
print(tfidf)
# (document number, term number)
```

```
from sklearn.decomposition import TruncatedSVD
svd = TruncatedSVD(n_components = 3)
lsa = svd.fit_transform(tfidf)
```

```
lsa
```

Comprehension - Latent Semantic Analysis

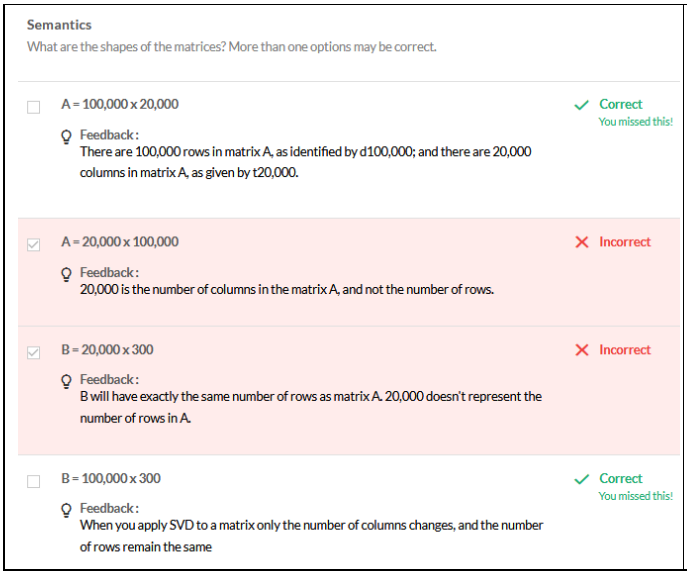
Say a media company has a set of **100,000 news articles** (documents) printed over the past few months. It wants to conduct various types of analyses on news articles, such as recommending relevant articles to a given user, comparing the similarity between articles etc. The size of the **vocabulary** is **20,000 words.**

Create two matrices to capture the data –

- Vanilla **term-document matrix A** where each row represents a document d and each column represents a term w
- Matrix B created by performing LSA with **k=300** dimensions on the matrix A.

Assume that the matrix A is constructed using tf-idf frequencies. Also, assume that each document has a label representing its news category (sports, stock market, politics, startups etc.).

	t1	t2	…	t20,000	label
d1	0.67	1.34	…	2.11	sports
d2	0.87	0.02	…	0.00	finance
…	0.00	3.22	…	0.56	…
d100,000	0.07	0.00	…	0.00	startups



Semantics

What are the shapes of the matrices? More than one options may be correct.

□ A = 100,000 x 20,000 — ✓ Correct — You missed this!

Feedback:
There are 100,000 rows in matrix A, as identified by d100,000; and there are 20,000 columns in matrix A, as given by t20,000.

☑ A = 20,000 x 100,000 — ✗ Incorrect

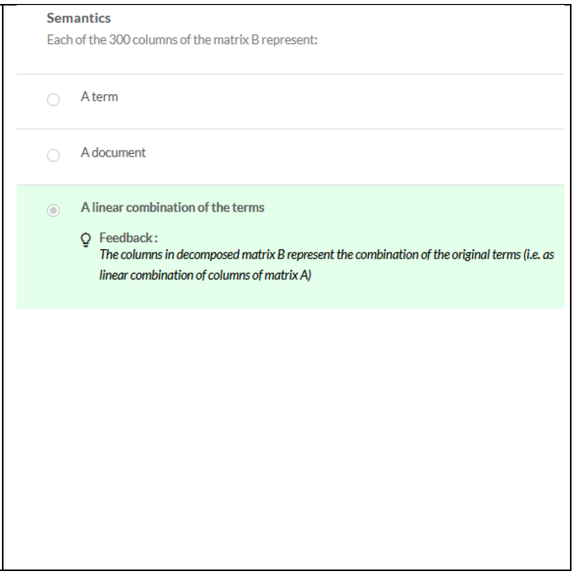
Feedback:
20,000 is the number of columns in the matrix A, and not the number of rows.

☑ B = 20,000 x 300 — ✗ Incorrect

Feedback:
B will have exactly the same number of rows as matrix A. 20,000 doesn't represent the number of rows in A.

□ B = 100,000 x 300 — ✓ Correct — You missed this!

Feedback:
When you apply SVD to a matrix only the number of columns changes, and the number of rows remain the same



Semantics

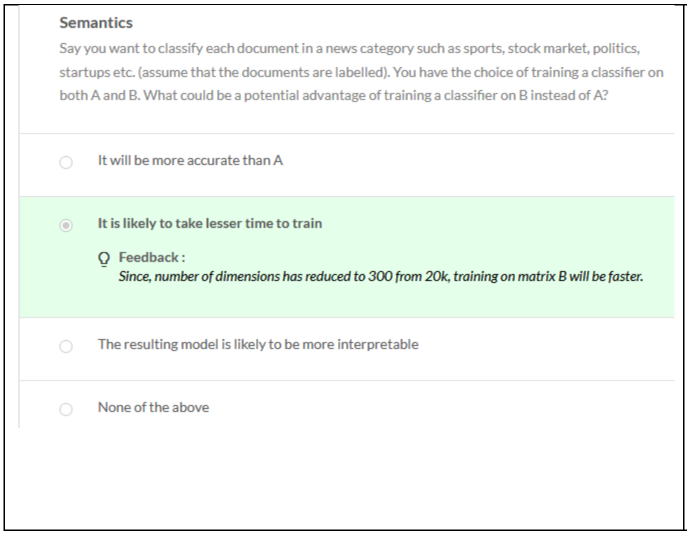
Each of the 300 columns of the matrix B represent:

○ A term

○ A document

● A linear combination of the terms

Feedback:
The columns in decomposed matrix B represent the combination of the original terms (i.e. as linear combination of columns of matrix A)



Semantics

Say you want to classify each document in a news category such as sports, stock market, politics, startups etc. (assume that the documents are labelled). You have the choice of training a classifier on both A and B. What could be a potential advantage of training a classifier on B instead of A?

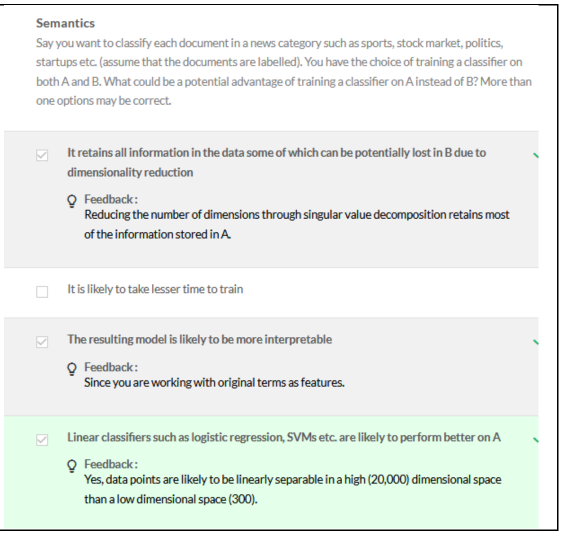
○ It will be more accurate than A

● **It is likely to take lesser time to train**

Feedback :
Since, number of dimensions has reduced to 300 from 20k, training on matrix B will be faster.

○ The resulting model is likely to be more interpretable

○ None of the above



Semantics

Say you want to classify each document in a news category such as sports, stock market, politics, startups etc. (assume that the documents are labelled). You have the choice of training a classifier on both A and B. What could be a potential advantage of training a classifier on A instead of B? More than one options may be correct.

☑ **It retains all information in the data some of which can be potentially lost in B due to dimensionality reduction**

Feedback :
Reducing the number of dimensions through singular value decomposition retains most of the information stored in A.

□ It is likely to take lesser time to train

☑ **The resulting model is likely to be more interpretable**

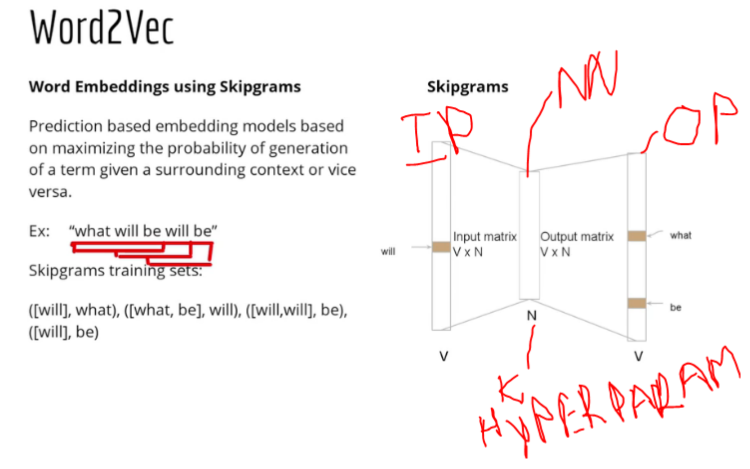
Feedback :
Since you are working with original terms as features.

☑ **Linear classifiers such as logistic regression, SVMs etc. are likely to perform better on A**

Feedback :
Yes, data points are likely to be linearly separable in a high (20,000) dimensional space than a low dimensional space (300).

Skipgram Model

Let's now study **prediction-based approaches** for creating word embeddings.



Word2Vec

Word Embeddings using Skipgrams

Prediction based embedding models based on maximizing the probability of generation of a term given a surrounding context or vice versa.

Ex: "what will be will be"

Skipgrams training sets:

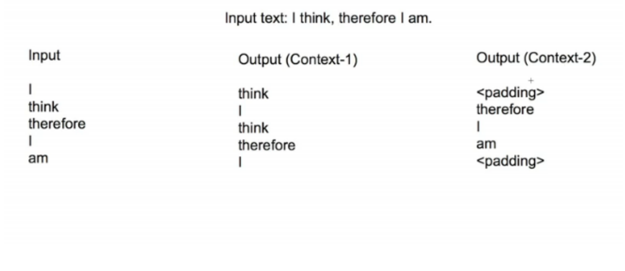
([will], what), ([what, be], will), ([will,will], be), ([will], be)

While considering the training data for the skipgram model, we should also consider ([be,be],will).

V in input is one hot vector and each word in vocabulary is fed t Neural network hidden layer in middle. Output is expected output and its size is same as input but additional 1s in it.

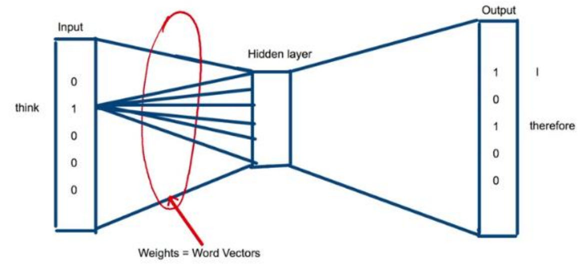
N (or k) is hyperparameter to neural network.

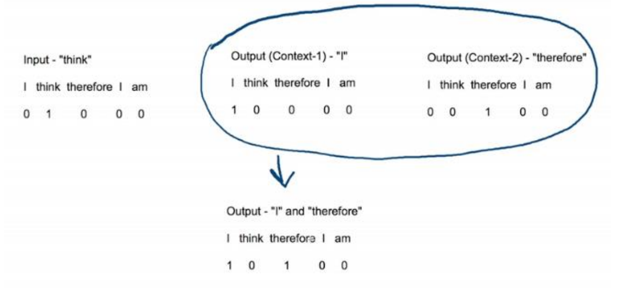
Each input connected through weight to NN hidden layer where summation of inputs generates trigger function. Hidden layer forwards out to output layer of NN where another trigger function is calculated. Each weight is random initially but we calculate error vector based on actual output vector vs expected output vector and then do back propagation to correct weights and keep doing till sufficient training is done.



Input text: I think, therefore I am.

Input	Output (Context-1)	Output (Context-2)
I	think	<padding>
think	I	therefore
therefore	think	I
I	therefore	am
am	I	<padding>





Input - "think"

I	think	therefore	I	am
0	1	0	0	0

Output (Context-1) - "I"

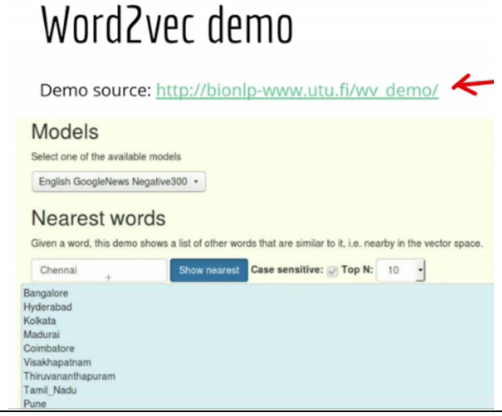
I	think	therefore	I	am
1	0	0	0	0

Output (Context-2) - "therefore"

I	think	therefore	I	am
0	0	1	0	0

Output - "I" and "therefore"

I	think	therefora	I	am
1	0	1	0	0



Word2vec demo

Demo source: http://bionlp-www.utu.fi/wv_demo/

Models

Select one of the available models

English GoogleNews Negative300 ▾

Nearest words

Given a word, this demo shows a list of other words that are similar to it, i.e. nearby in the vector space.

Chennai | Show nearest | Case sensitive: ☑ | Top N: 10

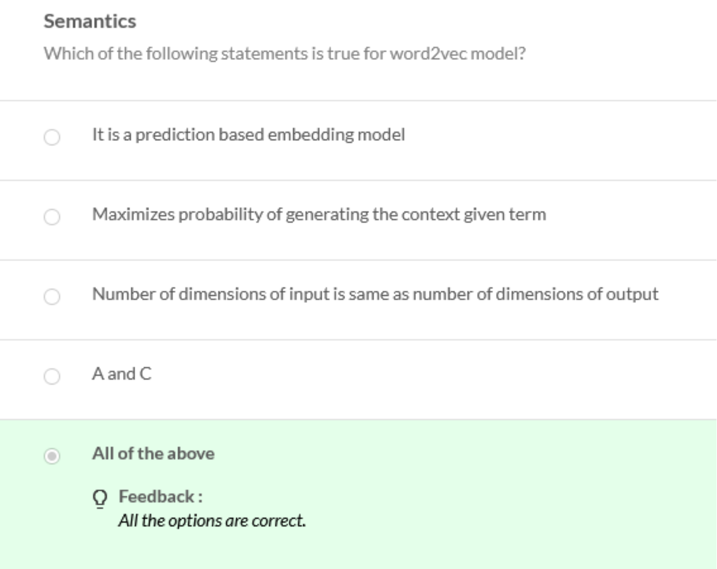
Bangalore
Hyderabad
Kolkata
Madurai
Coimbatore
Visakhapatnam
Thiruvananthapuram
Tamil_Nadu
Pune

Study of neural networks in detail is for later. For now, only need to understand the input and the output fed to the network; ignore how the network is trained, the exact architecture etc.

In the skip-gram approach to generating word vectors, the input is your target word and the task of the neural network is to predict the context words (the output) for that target word. The input word is represented in the form of a 1-hot-encoded vector. Once trained, the weight matrix between the input layer and the hidden layer gives the word embeddings for any target word (in the vocabulary).

Fun exercise!

Check out this fun app - Print words that are highly correlated to the input word. Try entering any random word, it will output all the words that are most related to that word, along with the similarity score!



Semantics

Which of the following statements is true for word2vec model?

○ It is a prediction based embedding model

○ Maximizes probability of generating the context given term

○ Number of dimensions of input is same as number of dimensions of output

○ A and C

● All of the above

Feedback :
All the options are correct.

Comprehension - Word2Vec

Word2vec is a technique used to compute word-embeddings (or word vectors) using some large corpora as the training data.

Say you have a large corpus of vocabulary |V| = 10,000 words. The task is to create a word embedding of say 300 dimensions for each word (i.e. each word should be a vector of size 300 in this 300-dimensional space).

The first step is to create a distributed representation of the corpus using a technique such as skip-gram where each word is used to predict the neighbouring ‘context words’. Let's assume that you have used some k-skip-n-grams.

The model’s (a neural network, shown below) task is to learn to **predict the context words** correctly for each input word. The input to the network is a **one-hot encoded vector** representing one term. For e.g. the figure below shows an input vector for the word ‘ants’.

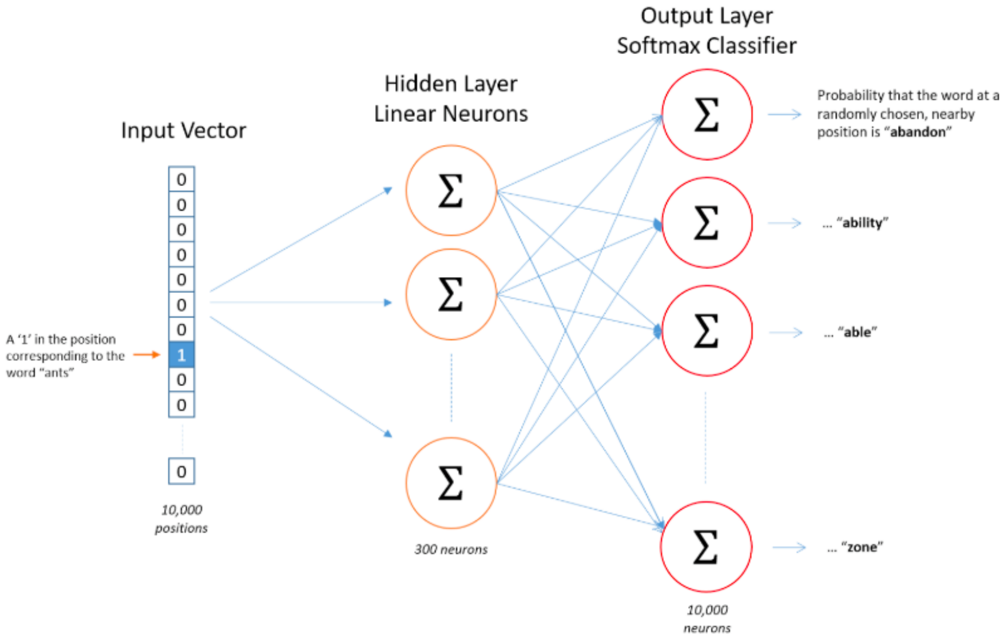


Fig1. Word2Vec

The hidden layer is a layer of *neurons* - in this case, 300 neurons. Each of the 10,000 elements in the input vector is connected to each of the 300 neurons (though only three connections are shown above). Each of these **10,000 x 300 connections** has a **weight** associated to it. This matrix of weights is of size 10,000 x 300, where each row represents a word vector of size 300.

The output of the network is a vector of size 10,000. Each element of this 10,000-vector represents the *probability of an output context word* for the given (one-hot) input word.

For e.g., if the context words for the word ‘ants’ are ‘bite’ and ‘walk’, the elements corresponding to these two words should have much higher probabilities (close to 1) than the other words. This layer is called the

softmax layer since it uses the 'softmax function' to convert discrete classes (words) to probabilities of classes.

The **cost function** of the network is thus the difference between the ideal output (probabilities of ‘bite’ and ‘walk’) and the actual output (whatever the output is with the current set of weights).

The **training task** is to **learn the weights** such that the output of the network is as close to the expected output. Once trained (using some optimisation routine such as gradient descent), the **10,000 x 300 weights** of the network represent the **word embeddings** - each of the 10,000 words having an embedding/vector of size 300.

The neural network mentioned above is informally called 'shallow' because it has only one hidden layer, though one can increase the number of such layers. Such a shallow network architecture was used by Mikolov et al. to train word embeddings for about 1.6 billion words, which become popularly known as Word2Vec.

Other Word Embeddings

After the widespread success of **Word2Vec**, several other (and perhaps more effective) word embedding techniques have been developed by various teams. One of the most popular is **GloVe (Global Vectors for Words)** developed by a Stanford research group.

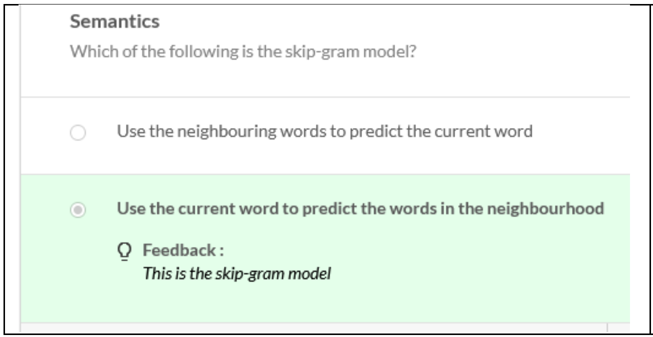
Another recently developed and fast-growing word embedding library is fastText developed by Facebook AI Research (FAIR). Apart from word embeddings of English, it contains pre-trained embeddings for about **157 languages** (including Hindi, Marathi, Goan, Malayalam, Tamil).

These embeddings are trained on billions of words (i.e. unique tokens), and thankfully, are available as pre-trained word vectors ready to use for text applications (for free!). You will learn to use pre-trained word vectors for text processing tasks such as classification and clustering in the upcoming sessions.

Read more about Word2Vec, GloVe and fastText in the additional readings provided below.

Additional Readings

1. Word2Vec: Original paper of word2vec by Mikolov et al.
2. GloVe vectors: Homepage (contains downloadable trained GloVe vectors, training methodology etc.).
3. fastText: Word embeddings developed by FAIR on multiple languages, available for download here



Semantics

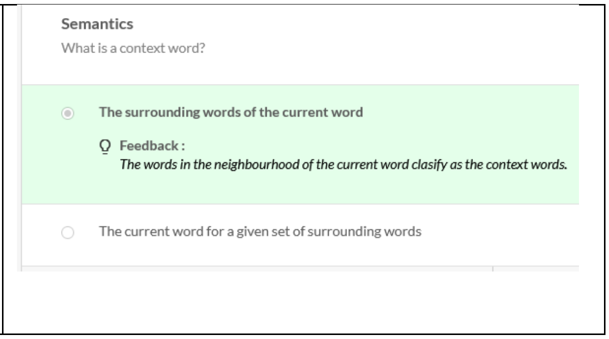
Which of the following is the skip-gram model?

○ Use the neighbouring words to predict the current word

● Use the current word to predict the words in the neighbourhood

Feedback :

This is the skip-gram model



Semantics

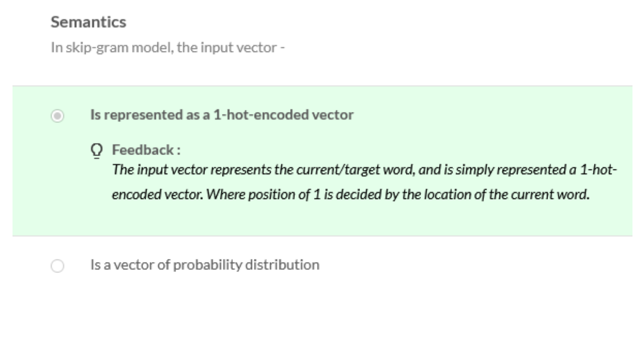
What is a context word?

● The surrounding words of the current word

Feedback :

The words in the neighbourhood of the current word clasify as the context words.

○ The current word for a given set of surrounding words



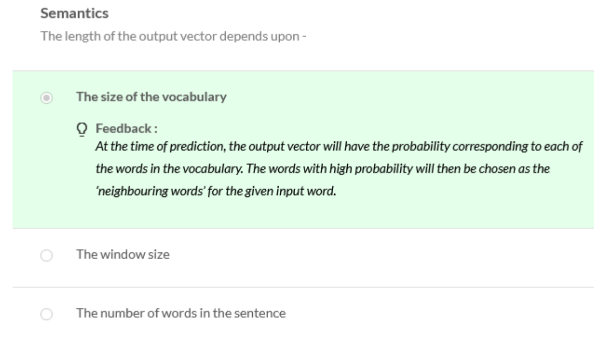
Semantics

In skip-gram model, the input vector -

● **Is represented as a 1-hot-encoded vector**

Feedback :
The input vector represents the current/target word, and is simply represented a 1-hot-encoded vector. Where position of 1 is decided by the location of the current word.

○ Is a vector of probability distribution



Semantics

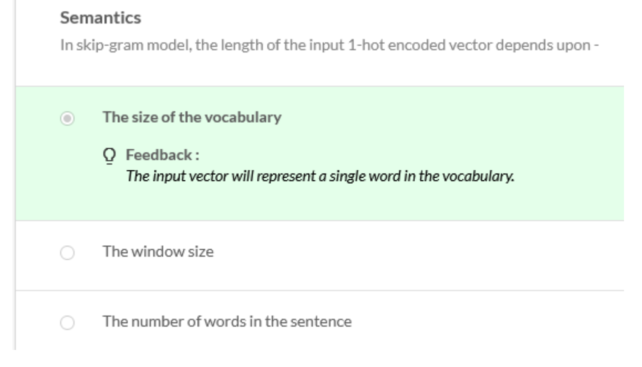
The length of the output vector depends upon -

● **The size of the vocabulary**

Feedback :
At the time of prediction, the output vector will have the probability corresponding to each of the words in the vocabulary. The words with high probability will then be chosen as the 'neighbouring words' for the given input word.

○ The window size

○ The number of words in the sentence



Semantics

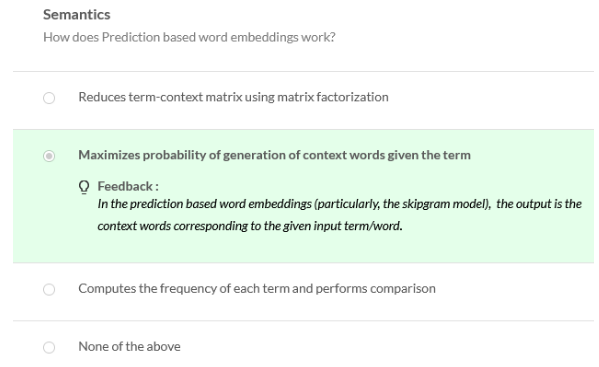
In skip-gram model, the length of the input 1-hot encoded vector depends upon -

● **The size of the vocabulary**

Feedback :
The input vector will represent a single word in the vocabulary.

○ The window size

○ The number of words in the sentence



Semantics

How does Prediction based word embeddings work?

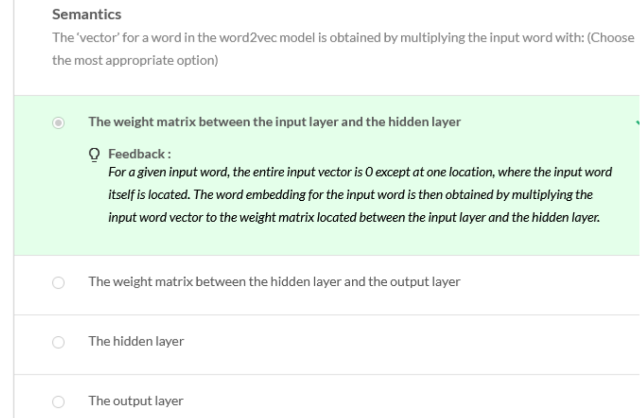
○ Reduces term-context matrix using matrix factorization

● **Maximizes probability of generation of context words given the term**

Feedback :
In the prediction based word embeddings (particularly, the skipgram model), the output is the context words corresponding to the given input term/word.

○ Computes the frequency of each term and performs comparison

○ None of the above



Semantics

The 'vector' for a word in the word2vec model is obtained by multiplying the input word with: (Choose the most appropriate option)

● **The weight matrix between the input layer and the hidden layer**

Feedback :
For a given input word, the entire input vector is 0 except at one location, where the input word itself is located. The word embedding for the input word is then obtained by multiplying the input word vector to the weight matrix located between the input layer and the hidden layer.

○ The weight matrix between the hidden layer and the output layer

○ The hidden layer

○ The output layer

Generate Vectors using LSA

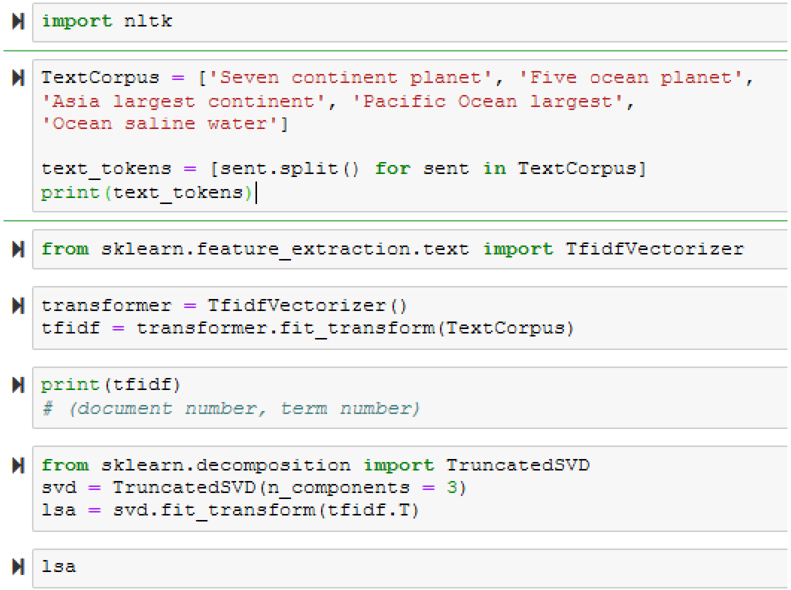
Let's now see how LSA is used to generate vectors for the words in Python.

In the previous lab, the number of documents was kept constant and the column size was reduced, where columns represented the terms.

However, in the following exercise, the number of terms is kept constant while the number of rows (i.e. documents) is reduced.

If you compare this code-lab with the one in LSA, you will find that most of the code is similar except how tf-idf is used. This results in generating vectors for words in this lab, and vectors for documents in the previous lab.

LSA+_+word+vector.ipynb



```
import nltk
```

```
TextCorpus = ['Seven continent planet', 'Five ocean planet',
'Asia largest continent', 'Pacific Ocean largest',
'Ocean saline water']

text_tokens = [sent.split() for sent in TextCorpus]
print(text_tokens)
```

```
from sklearn.feature_extraction.text import TfidfVectorizer
```

```
transformer = TfidfVectorizer()
tfidf = transformer.fit_transform(TextCorpus)
```

```
print(tfidf)
# (document number, term number)
```

```
from sklearn.decomposition import TruncatedSVD
svd = TruncatedSVD(n_components = 3)
lsa = svd.fit_transform(tfidf.T)
```

```
lsa
```

Word2vec in Python - I

In the following few lectures, you will learn to train word vectors using some text corpora and to use pre-trained word vectors for text processing tasks such as classification etc.

01+Exploring+Word+Vectors+v1.ipynb

Word2vec and GloVe in Python - II

In the previous segment, you learnt to train word vectors using a corpus of your choice and did some basic exploratory analysis on them.

In this segment, we will experiment with parameters of word embeddings, such as the length of word vectors, the technique used to generate co-occurrence matrices (skip-gram, continuous bag-of-words or CBOW) etc. and see how the 'quality' of word vectors is affected by them.

Length of Word Embedding

Let's first see how the length (dimensionality) of word vectors affect the output of a word2vec model.

So, you saw that higher word length captures more subtleties. Note that in absence of quantitative metrics to measure the 'goodness' of word vectors, we are measuring 'goodness' purely qualitatively by observing a few instances.

Skip-gram and CBOW

Apart from the skip-gram model, there is one more model that can be used to extract word embeddings for the word. This model is called **Continuous-Bag-of-Words (CBOW)** model.

Recall that **skip-gram** takes the target/given word as the input and predicts the context words (in the window), whereas **CBOW** takes the context terms as the input and predicts the target/given term.

You saw that word embeddings trained using skip-grams are slightly 'better' than those trained using CBOW for less frequent words (such as 'meandering'). By 'better', we simply mean that words similar to 'meandering', an infrequent word, will also be infrequent words.

You also learnt another important (though subtle) concept - the **effect of window size** on what the embeddings learn. When you choose a **small window size** (such as 5), 'similar' words will often be **synonyms**, whereas if you choose a larger window size (such as 10), 'similar' words will be **contextually similar** words.

Word2vec and GloVe in Python - III

In this segment, you will learn to use word embeddings for **classification tasks.** Although while you are going through this exercise, keep in mind that you can do almost any other typical ML task using word embeddings - clustering, PCA + classification etc. that you do with normal numeric features.

Let’s first look at how a normal bag-of-words approach performs on a text classification problem. You will later see whether word embeddings give better results.

In the following lecture, you will use **Bernoulli Naive Bayes** to make the predictions on the **Reuters dataset**. This dataset contains news articles from 8 categories, and the task is to classify the category of an article. Recall that Bernoulli Naive Bayes uses the presence of a term in a document (i.e. incidence) as the feature, i.e. 1 if present, 0 if not.

3 files for exercise are

- 02+Classification+with+word+vectors+v1.ipynb
- r8-test-all-terms.txt
- r8-train-all-terms.txt

You saw how Bernoulli Naive Bayes perform on the dataset. Let's now use **word embeddings** to make predictions using Naive Bayes. We will try using two options:

- Use pre-trained GloVe vectors
- Train our own vectors using the corpus

To download the pre-trained GloVe vectors, please first create an account in Kaggle and then download the following data file GloVe vectors.

While working with word embeddings, you have two options:

1. **Training your own word embeddings**: This is suitable when you have a sufficiently **large dataset** (a million words at least) or when the task is from a **unique domain** (e.g. healthcare). In specific domains, pre-trained word embeddings may not be useful since they might not contain embeddings for certain specific words (such as drug names).
2. **Using pre-trained embeddings**: Generally speaking, you should always start with pre-trained embeddings. They are an important performance benchmark trained on billions of words. You can also use the pre-trained embeddings as the starting point and then use your text data to further train them. This approach is known as 'transfer learning', which you will study in the neural networks course.

Note that although we have used a classification example to demonstrate the use of word embeddings, word embeddings themselves are applicable to a wide variety of ML tasks. For example, say you want to cluster many documents (**document clustering**), you can create a 'document vector' for each document and cluster them using (say) k-means clustering.

You can even use word embeddings in more sophisticated applications such as conversational systems (chatbots), fact-based question answering etc.

Word embeddings can also be used in other creative ways. In the optional session (Social Media Opinion Mining - Semantic Processing Case Study), you will see that word vectors have been modified to contain a 'sentiment score' (the last element of each vector represents 'document sentiment'), which are then used to model opinions in tweets.

With that, you have completed the section on distributional semantics. In the next few segments, you will learn another important area of semantic processing **- topic modelling**.

Basics of Topic Modelling with ESA

In the previous few segments, you have learnt word embeddings and their applications. Let's now move on to another important task in semantic processing - **topic modelling**.

You had briefly studied the concept of 'aboutness' (in the semantic associations segment) in the first session. Recall that the topic modelling task is to infer the 'topics being talked about' in a given set of documents.

Note that in this session, you will study only the basics of topic modelling, though the topic (pun intended :)) has been covered in much more detail in the **optional session named Topic Modelling.**

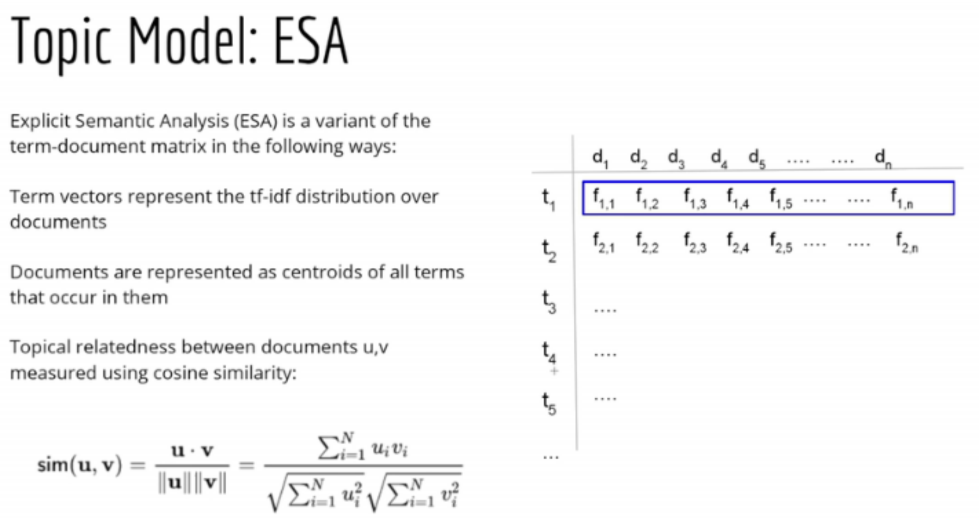
Let's start with the basics of topic modelling.

How do you calculate the centroid of the document? You first calculate the word vector corresponding to each word in the document, and then compute the centroid corresponding to all the word vectors by essentially taking the average of all the word vectors.

There are various ways in which you can extract topics from text:

1. PLSA - Probabilistic Latent Semantic Analysis
2. LDA - Latent Dirichlet Allocation
3. ESA - Explicit Semantic Analysis

Let's now discuss the approach of Explicit Semantic Analysis



Topic Model: ESA

Explicit Semantic Analysis (ESA) is a variant of the term-document matrix in the following ways:

Term vectors represent the tf-idf distribution over documents

Documents are represented as centroids of all terms that occur in them

Topical relatedness between documents u,v measured using cosine similarity:

$$\text{sim}(\mathbf{u},\mathbf{v}) = \frac{\mathbf{u}\cdot\mathbf{v}}{\|\mathbf{u}\|\,\|\mathbf{v}\|} = \frac{\sum_{i=1}^{N} u_i v_i}{\sqrt{\sum_{i=1}^{N} u_i^2}\sqrt{\sum_{i=1}^{N} v_i^2}}$$

	d_1	d_2	d_3	d_4	d_5	....	....	d_n
t_1	$f_{1,1}$	$f_{1,2}$	$f_{1,3}$	$f_{1,4}$	$f_{1,5}$	....	....	$f_{1,n}$
t_2	$f_{2,1}$	$f_{2,2}$	$f_{2,3}$	$f_{2,4}$	$f_{2,5}$	....	....	$f_{2,n}$
t_3	....							
t_4	....							
t_5	....							
...								

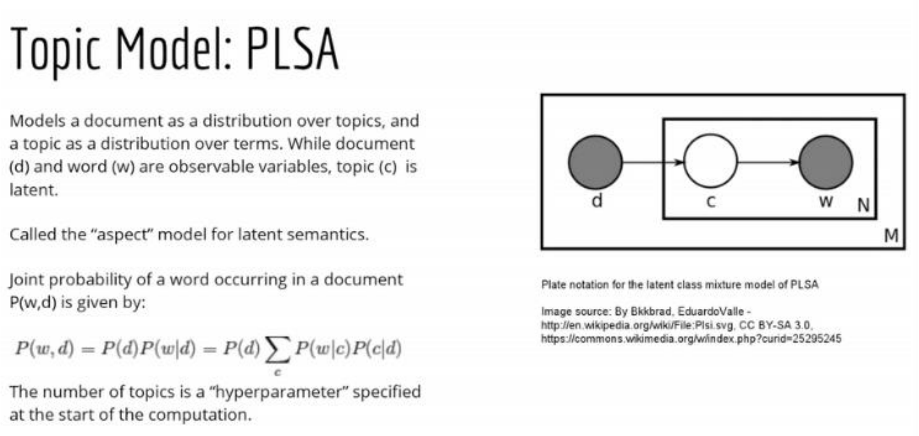
In **ESA**, the 'topics' are represented by a 'representative word' which is closest to the centroid of the document. Say you have N Wikipedia articles as your documents and the total vocabulary is V. You first create a tf-idf matrix of the terms and documents so that each term has a corresponding tf-idf vector.

Now, if a document contains the words sugar, hypertension, glucose, saturated, fat, insulin etc., each of these words will have a vector (the tf-idf vector). The 'centroid of the document' will be computed as the centroid of all these word vectors. The centroid represents 'the average meaning' of the document in some sense. Now, you can compute the distance between the centroid and each word, and let’s say that you find the word vector of ‘hypertension’ is closest to the centroid. Thus, you conclude that the document is most closely about the topic ‘hypertension’.

Note that it is also possible that a word which is not present in the document is closest to its centroid. For example, say you compute the centroid of a document containing the words ‘numpy’, ‘cross-validation’, ‘preprocessing’, ‘overfitting’ and ‘variance’, it is possible that the centroid of this document is closest to the vector of the term ‘machine learning’ which may not be present in the document.

Introduction to Probabilistic Latent Semantics Analysis (PLSA)

Let's now discuss another approach for topic modelling **- PLSA**. It is a more generalized form of LSA. Note that you can study **topic modelling** (PLSA and LDA) in the next (optional) session in detail, though the following lectures will give you a good primer of the next session.



Topic Model: PLSA

Models a document as a distribution over topics, and a topic as a distribution over terms. While document (d) and word (w) are observable variables, topic (c) is latent.

Called the "aspect" model for latent semantics.

Joint probability of a word occurring in a document P(w,d) is given by:

$$P(w,d) = P(d)P(w|d) = P(d)\sum_{c} P(w|c)P(c|d)$$

The number of topics is a "hyperparameter" specified at the start of the computation.

Plate notation for the latent class mixture model of PLSA

Image source: By Bkkbrad, EduardoValle - http://en.wikipedia.org/wiki/File:Plsi.svg, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=25295245

To summarise, the basic idea of PLSA is this -

We are given a list of documents and we want to identify the topics being talked about in each document. For e.g., if the documents are news articles, each article can be a **collection of topics** such as elections, democracy, economy etc. Similarly, technical documents such as research papers can have topics such as hypertension, diabetes, molecular biology etc.

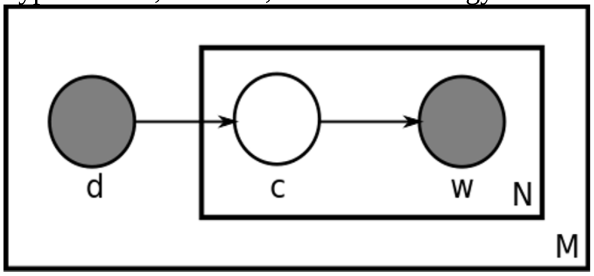


Fig2. PLSA

PLSA is a **probabilistic technique** for topic modelling. First, we fix an arbitrary number of topics which is a hyperparameter (say 20 topics in all documents). The basic model we assume is this - each document is a collection of some topics and each topic is a collection of some terms.

For e.g. a topic t1 can be a collection of terms (hypertension, sugar, insulin, ...) etc. t2 can be (numpy, variance, learning, ...) etc. The topics are, of course, not given to us, we are only given the documents and the terms.

That is, we do not know:

1. How many topics are there in each document (we only know the total number of topics across all documents).
2. What is each 'topic' c, i.e. which terms represent each topic.

The **task** of the PLSA algorithm is to figure out the set of topics c. PLSA is often represented as a graphical model with shaded nodes representing observed random variables (d, w) and unshaded ones unobserved random variables (c). The basic idea for setting up the optimisation routine is to find the set of topics c which maximises the joint probability P(d, w).

Also note that the term '**explicit**' in ESA indicates that the topics are represented by explicit terms such as hypertension, machine learning etc, rather than 'latent' topics such as those used by PLSA.

You can learn PLSA (and other techniques in topic modelling) in the next session on **topic modelling.**

Summary

In this session, you studied the idea of **distributional semantics** and word vectors in detail. You learnt that words can be represented as vectors and that usual vector algebra operations can be performed on these vectors.

Word vectors can be represented as matrices in broadly two ways - using the term-document (**occurrence context** matrices) or the term-term **co-occurrence** matrices. Further, there are various techniques to create the co-occurrence matrices such as **context-based co-occurrence, skip-grams** etc.

You studied that word vectors created using both the above techniques (term-document/occurrence context matrices and the term-term/co-occurrence matrices) are **high-dimensional** and **sparse**.

Word embeddings are a lower-dimensional representation of the word vectors. There are broadly two ways to generate word embeddings - **frequency-based** and **prediction-based:**

- In a **frequency-based** approach, you take the high-dimensional **occurrence-context** or a **co-occurrence** matrix. Word embeddings are then generated by performing the dimensionality reduction of the matrix using matrix factorisation (e.g. LSA).
- **Prediction based** approach involves training a shallow neural network which learns to predict the words in the context of a given input word. The two widely used prediction-based models are the **skip-gram model** and the **Continuous Bag of Words (CBOW)** model. In the skip-gram model, the input is the current/target word and the output are the context words. The embeddings then are represented by the weight matrix between the input layer and the hidden layer. Also, **word2vec** and **GloVe** vectors are two of the most popular pre-trained word embeddings available for use.

You also studied the notion of **aboutness** and the task of topic modelling - text is usually about some (and usually more than one) 'topics'. There are multiple techniques that are used for **topic modelling** such as **ESA, PLSA, LDA etc.** You can study PLSA and LDA in detail in the next session.