Natural Language Processing

Contents

- Introduction to NLP and its Use Cases
- Natural Language Basics
- Text Wrangling
- Feature Engineering

Introduction

- NLP is a subfield of linguistics, computer science, information engineering, and artificial intelligence concerned with the interactions between computers and human (natural) languages, in particular how to program computers to process and analyze large amounts of natural language data.

- In simple terms, a natural language is a language developed and evolved by humans through natural use and communication rather than constructing and creating the language artificially, like a computer programming language.

Some Use Cases of NLP

- Machine Translation
- Question Answering Systems
- Text Summarisation
- Text Categorisation
- Text Analytics

Some Applications

- Spam detection
- News articles categorization
- Social media analysis and monitoring
- Biomedical
- Marketing
- Sentiment analysis
- Chatbots
- Virtual assistants

Natural Languages Background

- Linguistics is the scientific study of language, a special field that deals with various aspects of language.
- The main things to consider in any language are-
  - Syntax
  - Semantics
  - Grammar

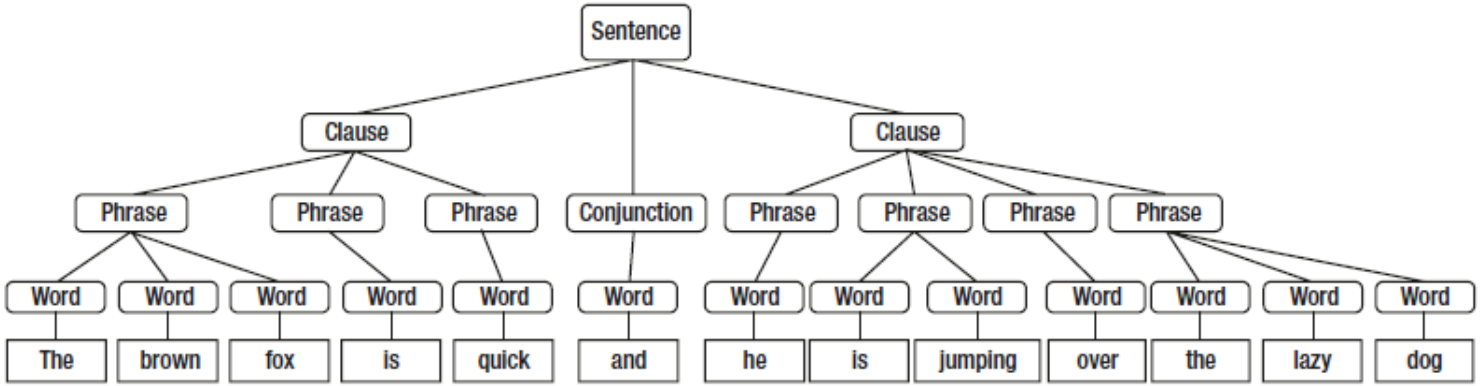
Language Syntax

- Words are combined into phrases. Phrases are combined into clauses and clauses are combined into sentences.
- Shallow parsing (also chunking) is an analysis of a sentence which first identifies constituent parts of sentences (nouns, verbs, adjectives, etc.) and then links them to higher order units that have discrete grammatical meanings (noun groups or phrases, verb groups, etc.).

- Example - The brown fox is quick and he is jumping over the lazy dog.

dog the over he
lazy jumping is the fox
and is quick brown

A collection of words without any relation or structure



Grammar

- It primarily consists of a set of rules that are used to determine how to position words, phrases, and clauses when constructing sentences in a natural language.

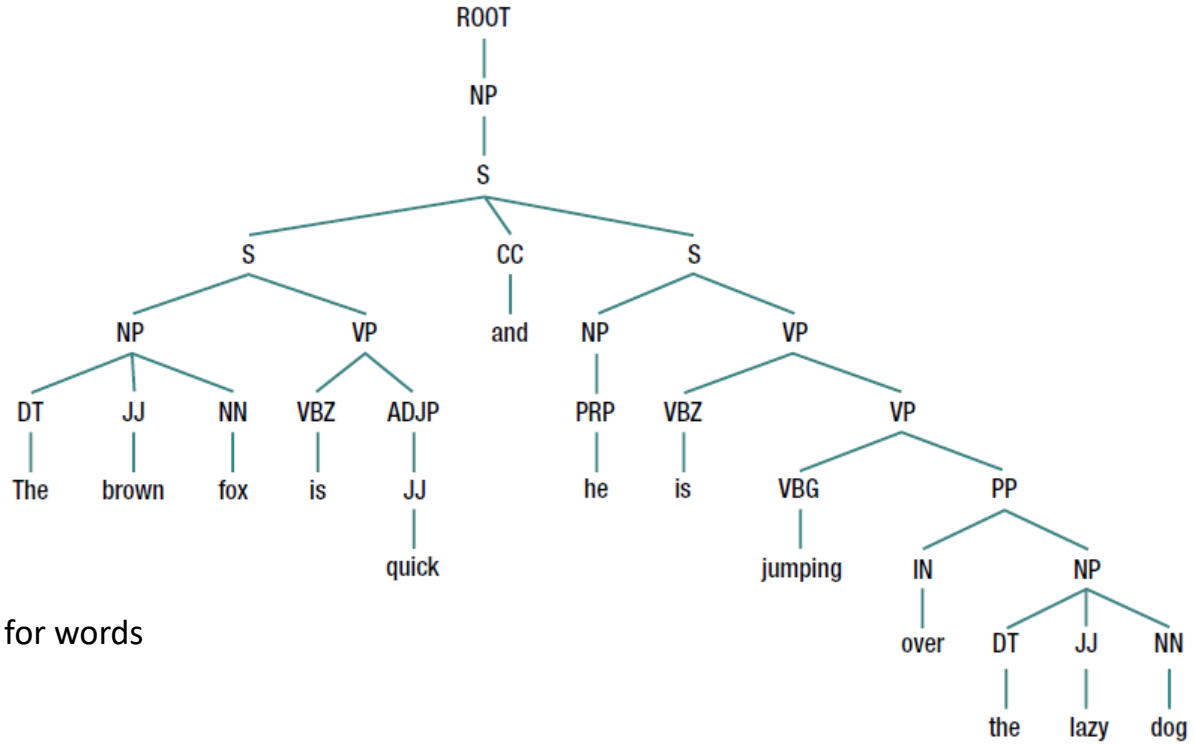
S -> NP VP

NP -> [DET][ADJ]N[PP]

VP -> V|MD[VP][NP][PP][ADJP][ADVP]

PP -> PREP[NP]

Constituency tree

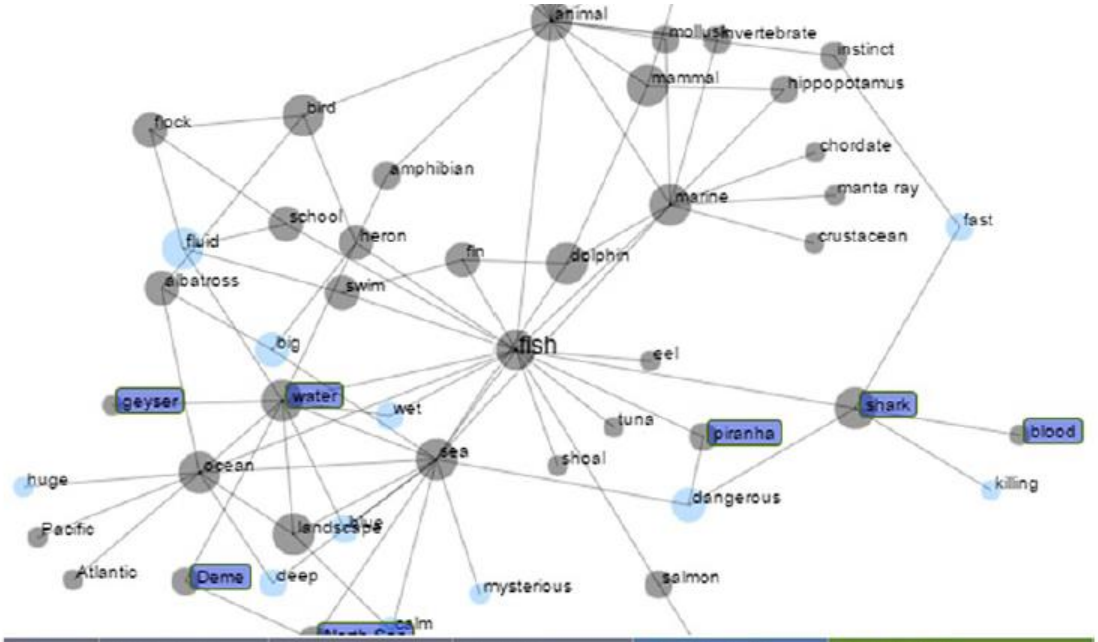


POS tags for words

Semantics

- A Lemma is also known as the canonical or citation form for a set of words.
- The lemma is usually the base form of a set of words, known as a lexeme.

- For example, {eating, ate, eats} is a lexeme and the lemma of these words is the word “eat”.



IS-A	HAS-SPECIFIC	IS-PART-OF	HAS-PART	HAS-PROPERTY	IS-RELATED-TO
animal	eel	ocean	fin	wet	heron
marine	piranha	school			
	salmon	sea			
	shark	water			
	shoal				
	tuna				

Semantic network around the concept of fish

Text Corpora

- A Text corpora is the plural form of “text corpus” and can be defined as large and structured collections of texts or textual data.
- Some of the most commonly used methods and techniques for annotating text corpora are-
- POS tagging: Annotating each word with a POS tag indicating the part of speech associated with it.
- Word stems: A stem for a word is a part of the word to which various affixes can be attached.
- Word lemmas: A lemma is the canonical or base form for a set of words and is also known as the head word.

- Grammar based parsing
- Semantic types and roles: The various constituents of sentences, including words and phrases, are annotated with specific semantic types and roles often obtained from an ontology that indicates what they do. These include things like place, person, time, organization, agent, recipient, theme, etc.

Why is Analysing Text Data Difficult

- Unstructured data.
- Language is inherently very high dimensional and sparse. There are tons of rare words, words can mean the same thing in different contexts.
- There is no way to define a word in an unambiguous way. 2 = 1 + 1, but how do you define e.g. family = ? love = ? There is no universal definition for most of the concepts we use everyday.
- No readily available features like in numerical data, images or audio data.

Text Pre-Processing

- Removing HTML tags – use the library BeautifulSoup or regex
- Tokenization – splitting into tokens using delimiters, sentence/word tokenizers available in nltk and spaCy
- Removing accented characters(converting é to e) – Unicode.normalize
- Removing stopwords(words having littile or no significance, eg - a,an,the… – nltk.corpus.stopwords.words('english')
- Case conversions – string functions
- Removing special characters – regex

- Handling contractions – create a dictionary of contractions and replace them
- Correcting spelling errors – very tricky, libraries like textblob and PyEnchant available
- Stemming – jumping/jumped/jumps = jump (removing prefix/suffix)
- Lemmatization – ate=eat, fancy=fancier

Feature Engineering

- Vector space model:

A document **D** in a document vector space **VS**.

$$VS = \{W1, W2, \ldots , Wn\}$$

where there are **n** distinct words across all documents.

$$D = \{wD1, wD2, \ldots , wDn\}$$

where wDn denotes the weight for word **n** in document **D**. This weight is a numeric value and can be anything ranging from the frequency of that word in the document, the average frequency of occurrence, embedding weights, or the *TF-IDF weight.*

Traditional Feature Engineering Models

- Count based, called Bag Of Words Model

- Occurrence/frequency based
- Bag of N-Grams Model
- TF-IDF Model

Frequency based

	bacon	beans	beautiful	blue	breakfast	brown	dog	eggs	fox	green	ham	jumps	kings	lazy	love	quick	sausages	sky	toast	today
0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
1	0	0	1	1	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	0
2	0	0	0	0	0	1	1	0	1	0	0	1	0	1	0	1	0	0	0	0
3	1	1	0	0	1	0	0	1	0	0	1	0	1	0	0	0	1	0	1	0
4	1	0	0	0	0	0	0	1	0	1	1	0	0	0	1	0	1	0	0	0
5	0	0	0	1	0	1	1	0	1	0	0	0	0	1	0	1	0	0	0	0
6	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0	1
7	0	0	0	0	0	1	1	0	1	0	0	0	0	1	0	1	0	0	0	0

Bag of N-Grams

	bacon eggs	beautiful sky	beautiful today	blue beautiful	blue dog	blue sky	breakfast sausages	brown fox	dog lazy	eggs ham	...	lazy dog	love blue	love green	quick blue	quick brown	sausages bacon	sausages ham	sky beautiful
0	0	0	0	1	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0
1	0	1	0	1	0	0	0	0	0	0	...	0	1	0	0	0	0	0	0
2	0	0	0	0	0	0	0	1	0	0	...	1	0	0	0	1	0	0	0
3	1	0	0	0	0	0	1	0	0	0	...	0	0	0	0	0	0	1	0
4	0	0	0	0	0	0	0	0	0	1	...	0	0	1	0	0	1	0	0
5	0	0	0	0	1	0	0	1	1	0	...	0	0	0	1	0	0	0	0
6	0	0	1	0	0	1	0	0	0	0	...	0	0	0	0	0	0	0	1
7	0	0	0	0	0	0	0	1	1	0	...	0	0	0	0	0	0	0	0

TF-IDF Model

$$tfidf = tf \times idf$$

Term frequency in any document vector is denoted by the raw frequency value of that term in a particular document.

$$tf(w,D) = f_{w_D}$$

$$idf(w,D) = 1 + \log \frac{N}{1 + df(w)}$$

N represents the total number of documents in our corpus, and *df* (*t*) represents the number of documents in which the term ***w*** is present.

TF-IDF Model

$$tfidf = \frac{tfidf}{\| tfidf \|}$$

	bacon	beans	beautiful	blue	breakfast	brown	dog	eggs	fox	green	ham	jumps	kings	lazy	love	quick	sausages	sky	toast	today
0	0.00	0.00	0.60	0.53	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.60	0.00	0.0
1	0.00	0.00	0.49	0.43	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.57	0.00	0.00	0.49	0.00	0.0
2	0.00	0.00	0.00	0.00	0.00	0.38	0.38	0.00	0.38	0.00	0.00	0.53	0.00	0.38	0.00	0.38	0.00	0.00	0.00	0.0
3	0.32	0.38	0.00	0.00	0.38	0.00	0.00	0.32	0.00	0.00	0.32	0.00	0.38	0.00	0.00	0.00	0.32	0.00	0.38	0.0
4	0.39	0.00	0.00	0.00	0.00	0.00	0.00	0.39	0.00	0.47	0.39	0.00	0.00	0.00	0.39	0.00	0.39	0.00	0.00	0.0
5	0.00	0.00	0.00	0.37	0.00	0.42	0.42	0.00	0.42	0.00	0.00	0.00	0.00	0.42	0.00	0.42	0.00	0.00	0.00	0.0
6	0.00	0.00	0.36	0.32	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.72	0.00	0.5
7	0.00	0.00	0.00	0.00	0.00	0.45	0.45	0.00	0.45	0.00	0.00	0.00	0.00	0.45	0.00	0.45	0.00	0.00	0.00	0.0

Limitations of Traditional Models

- Inherent nature of the model being just a bag of unstructured words, we lose additional information like the semantics, structure, sequence, and context around nearby words in each text document.

- Traditional models have some limitations considering sparse representations, leading to feature explosion. This causes the curse of dimensionality.

Advanced Feature Engineering Models

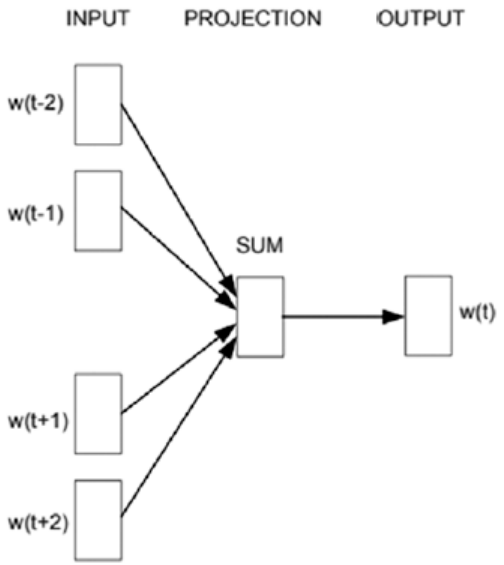
- *Count-based methods* like *Latent Semantic Analysis (LSA)* can be used to calculate statistical measures of how often words occur with their neighbouring words in a corpus and then build dense word vectors for each word from these measures.
- *Predictive methods* like *neural network based language models* try to predict words from their neighbouring words by looking at word sequences in the corpus. In the process, it learns distributed representations giving us dense word embeddings.
- We focus on these predictive methods.

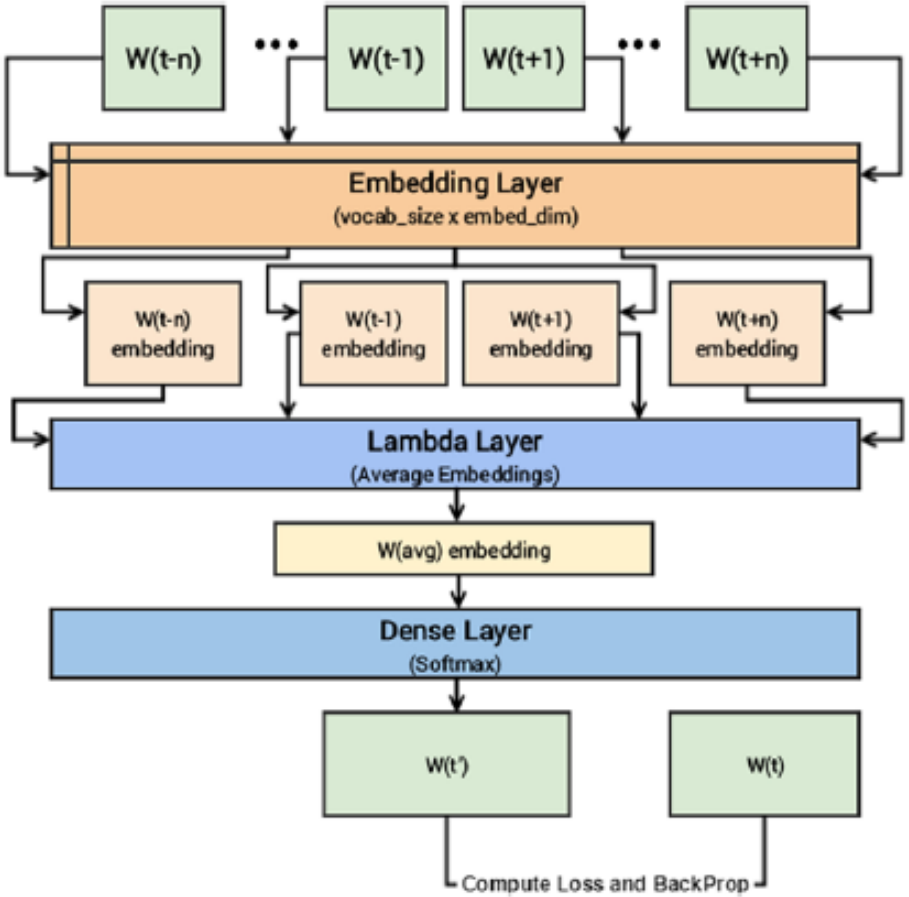
Word2Vec Model

- This model was created by Google in 2013 and is a predictive deep learning based model to compute and generate high quality, distributed, and continuous dense vector representations of words that capture contextual and semantic similarity.
- There are two different model architectures that can be leveraged by Word2Vec to create these word embedding representations:
  - The Continuous Bag of Words (CBOW) model
  - The Skip-Gram model

The Continuous Bag of Words (CBOW) Model

- The CBOW model architecture tries to predict the current target word(the center word) based on the source context words (surrounding words).
- Considering “the quick brown fox jumps over the lazy dog”, this can be pairs of (context _ window, target _ word), where if we consider a context window of size 2, we have examples like ([quick, fox], brown), ([the, brown], quick), ([the, dog], lazy), and so on.




W(t-n)
W(t-1)
W(t+1)
W(t+n)
Embedding Layer
(vocab_size x embed_dim)
W(t-n) embedding
W(t-1) embedding
W(t+1) embedding
W(t+n) embedding
Lambda Layer
(Average Embeddings)
W(avg) embedding
Dense Layer
(Softmax)
W(t')
W(t)
Compute Loss and BackProp

(12424, 100)

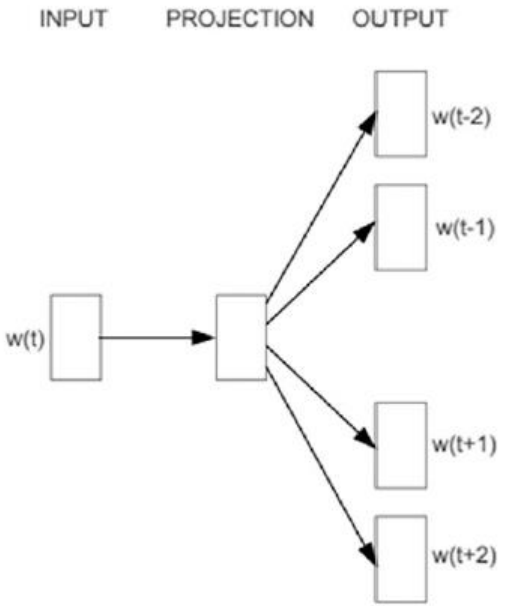
	0	1	2	3	4	5	6	7	8	9	...	90	91	92	93
shall	-1.183386	-2.866214	1.046431	0.943265	-1.021784	-0.047069	2.108584	-0.458692	-1.698881	0.905800	...	0.655786	0.703828	0.821803	-0.093732
unto	-1.725262	-1.765972	1.411971	0.917713	0.793832	0.310631	1.541964	-0.082523	-1.346811	0.095824	...	1.682762	-0.872293	1.908597	0.977152
lord	1.694633	-0.650949	-0.095796	0.950002	0.813837	1.538206	1.125482	-1.655581	-1.352673	0.409504	...	1.553925	-0.819261	1.086127	-1.545129
thou	-1.590623	-0.801968	1.659041	1.314925	-0.455822	1.733872	-0.233771	-0.638922	0.104744	0.490223	...	0.652781	-0.362778	-0.190355	0.040719
thy	0.386488	-0.834605	0.585985	0.801969	-0.165132	0.999917	1.224088	-0.317555	-0.671106	-1.073181	...	1.267184	-0.564660	0.089618	-0.979835

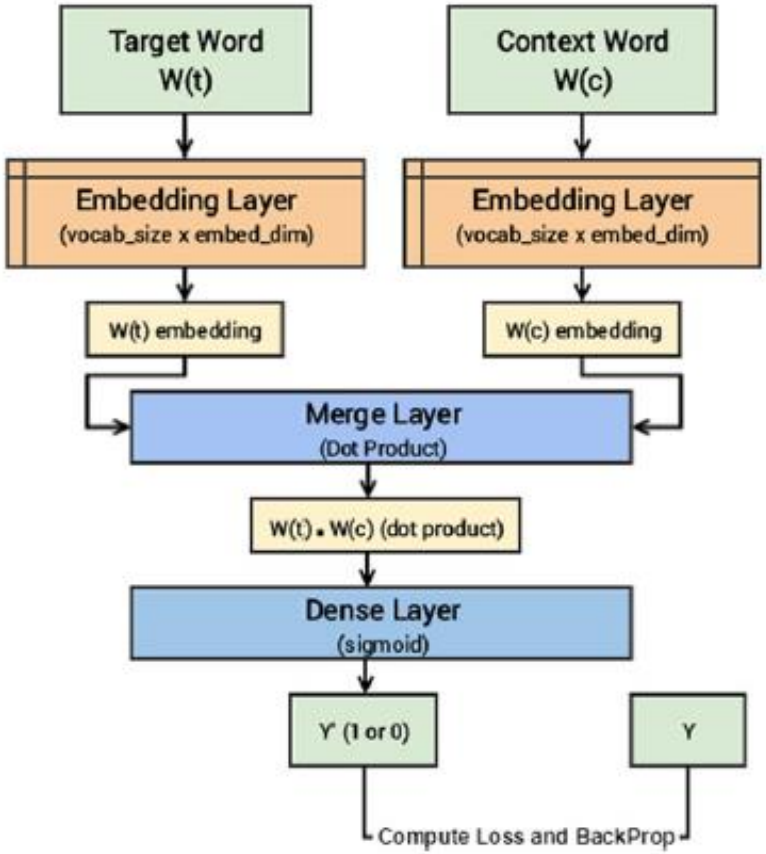
Build a pairwise distance matrix among all the words in our vocabulary based on the dense embedding vectors and then find the n-nearest neighbors of each word of interest based on the shortest (Euclidean) distance.

- {'egypt': ['destroy', 'none', 'whole', 'jacob', 'sea'],
- 'famine': ['wickedness', 'sore', 'countries', 'cease', 'portion'],
- 'god': ['therefore', 'heard', 'may', 'behold', 'heaven'],
- 'gospel': ['church', 'fowls', 'churches', 'preached', 'doctrine'],
- 'jesus': ['law', 'heard', 'world', 'many', 'dead'],
- 'john': ['dream', 'bones', 'held', 'present', 'alive'],
- 'moses': ['pharaoh', 'gate', 'jews', 'departed', 'lifted'],
- 'noah': ['abram', 'plagues', 'hananiah', 'korah', 'sarah']}

The Skip-Gram model

- The Skip-Gram model architecture tries to achieve the reverse of what the CBOW model does. It tries to predict the source context words (surrounding words) given a target word (the center word).
- Consider our simple sentence from earlier, “the quick brown fox jumps over the lazy dog”. If we used the CBOW model, we get pairs of (context_window, target_word), where if we consider a context window of size 2, we have examples such as ([quick, fox], brown), ([the, brown], quick), ([the, dog], lazy) and so on.




Target Word
W(t)
Context Word
W(c)
Embedding Layer
(vocab_size x embed_dim)
Embedding Layer
(vocab_size x embed_dim)
W(t) embedding
W(c) embedding
Merge Layer
(Dot Product)
W(t) . W(c) (dot product)
Dense Layer
(sigmoid)
Y' (1 or 0)
Y
Compute Loss and BackProp

(12424, 100)

	0	1	2	3	4	5	6	7	8	9	...	90	91	92	93
shall	0.043252	0.030233	-0.016057	-0.071856	0.005915	0.053170	0.013578	0.000201	0.037018	-0.151811	...	0.289811	0.014798	-0.022350	0.059966
unto	-0.072916	-0.014941	0.018243	-0.206662	-0.018253	0.071634	0.094720	0.008018	-0.003973	-0.076268	...	0.044276	0.097791	-0.120094	0.057171
lord	-0.024338	0.066582	-0.057416	-0.112375	0.034131	0.103507	-0.000733	0.071466	0.015607	-0.119505	...	0.115495	-0.027881	-0.215636	-0.028494
thou	0.084224	0.048217	0.008529	0.025198	0.019296	-0.005508	0.041746	-0.012590	-0.299545	-0.030134	...	0.079110	-0.037630	-0.016609	0.032280
thy	0.040458	0.054175	-0.033665	-0.031059	0.053622	0.157648	-0.009812	0.032927	-0.229837	0.002110	...	-0.033932	-0.079629	-0.070454	0.051992

- {'egypt': ['taken', 'pharaoh', 'wilderness', 'gods', 'became'],
- 'famine': ['moved', 'awake', 'driven', 'howl', 'snare'],
- 'god': ['strength', 'given', 'blessed', 'wherefore', 'lord'],
- 'gospel': ['preached', 'must', 'preach', 'desire', 'grace'],
- 'jesus': ['disciples', 'christ', 'dead', 'peter', 'jews'],
- 'john': ['peter', 'hold', 'mountain', 'ghost', 'preached'],
- 'moses': ['commanded', 'third', 'congregation', 'tabernacle', 'tribes'],
- 'noah': ['ham', 'terah', 'amon', 'adin', 'zelophehad']}

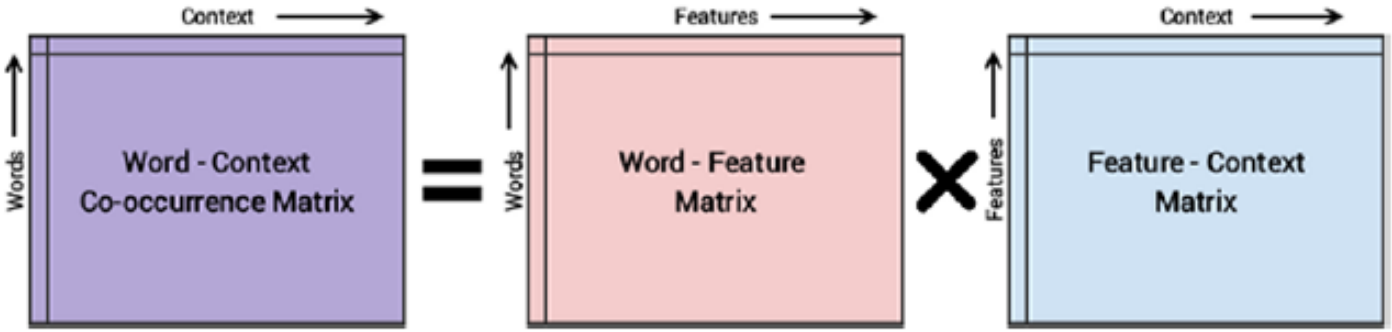
Document Clustering

- Getting Document Level Embeddings : average the word embeddings for each word in a document

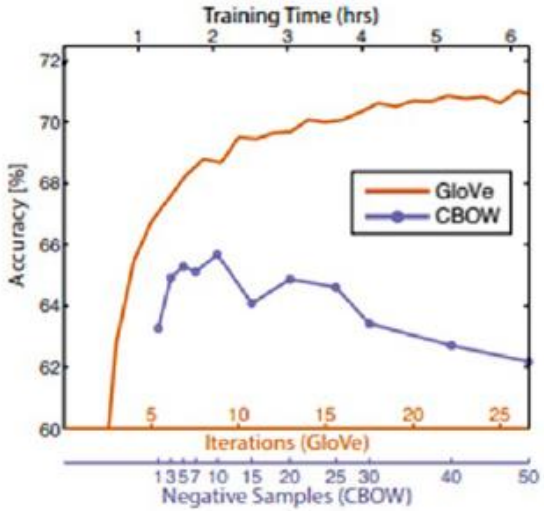
	Document	Category	ClusterLabel
0	The sky is blue and beautiful.	weather	2
1	Love this blue and beautiful sky!	weather	2
2	The quick brown fox jumps over the lazy dog.	animals	1
3	A king's breakfast has sausages, ham, bacon, eggs, toast and beans	food	0
4	I love green eggs, ham, sausages and bacon!	food	0
5	The brown fox is quick and the blue dog is lazy!	animals	1
6	The sky is very blue and the sky is very beautiful today	weather	2
7	The dog is lazy but the brown fox is quick!	animals	1

The GloVe Model

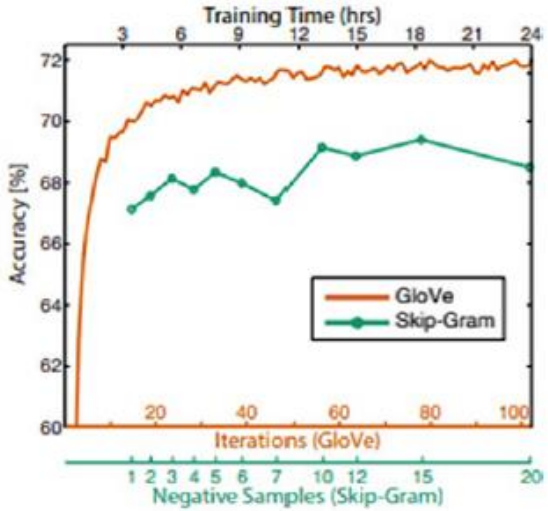
- Initialize *WF* and *FC* with some random weights and attempt to multiply them to get *WC'* (an approximation of WC) and measure how close it is to *WC*. Do this multiple times using *Stochastic Gradient Descent* (SGD) to minimize the error.



Word2Vec starts with local individual examples of word co-occurrence pairs and GloVe starts with global aggregated co-occurrence statistics across all words in the corpus.



(a) GloVe vs CBOW



(b) GloVe vs Skip-Gram