Text Processing : Vectorisation & Word Embedding

Content

- Conceptual Terminologies
- Elementary text to numeric conversion techniques
  - Count vectorizer
  - TFIDF
- Introduction to Word Embeddings
  - Word2vec -Skip Gram, CBOW
  - Global vector (GloVe)
  - FastText
  - Pre Trained Word Embeddings
- Keras packages/methods for Text Processing
- Use Case : Sentiment Analysis
- Libraries : Genism, Glove, Keras

Conceptual Text Processing Terminologies

- Document
- Corpus
- Vocabulary and Out-of-Vocabulary
- Word Sense Disambiguation
- Tokenization
- N-grams
- Word Vectorization
- Bag of Words (Bow)

Document

- In NLP, a text body is referred to as a Document. In other words, it is a collection of objects of the text sequence type, known as a “string” in Python.

Example: A document can be a single string or anything such as:

An Email

A News Article

A Movie Review

Corpus

- Collection of all available textual data is known as corpus.
- Such collections may be formed of **a single language of texts, or can span multiple languages.**

Vocabulary & Out-of-Vocabulary

- Vocabulary is the set of distinct words that are used in a language.(Vocabulary of language)
- Vocabulary refers to the distinct words in a selected corpus.(Vocabulary of corpus).
- Any word in a document which is not found in the relevant vocabulary is considered **Out of Vocabulary**.

Example: Let the corpus be:

'Bob ate apples. Fred ate apples. Bob and Fred ate pears.'

Then the vocabulary would be:

'Bob','ate', 'apples', 'Fred', 'and', 'pears'

Word Sense Disambiguation

- The ability to computationally identify the meaning of words in context is known as Word Sense Disambiguation.
- As part of this process, a third-party corpus or knowledge base, such as WordNet or Wikipedia, is frequently used to cross-reference entities.

Example: An algorithm trying to determine whether a text reference to the word "**apple**" refers to the fruit or the brand.

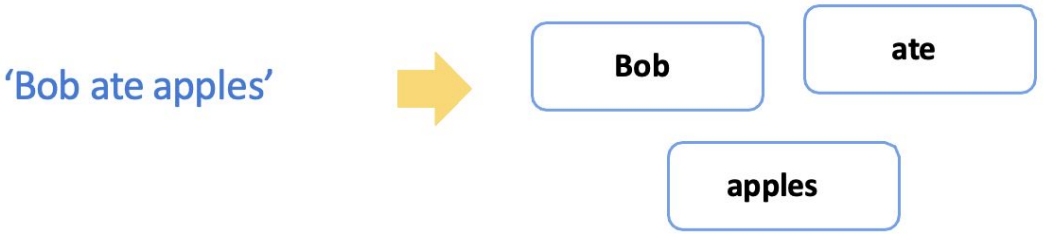
'**Apples** are good for health.'

'**Apple** is the best smartphone brand.'

Tokenization

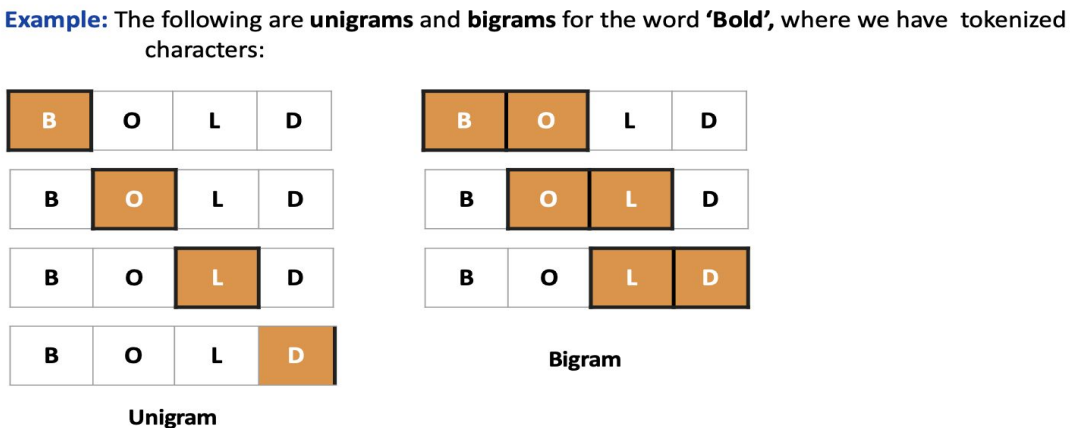
- Tokenization is a step in the NLP process that breaks longer strings of text into smaller pieces or tokens.
- Generally, token are words. Though, tokens can be sentences when we break a body of text into sentences, or tokens can also be individual characters of the language alphabet also can be combination of 2 words.
- What the industry has found in recent years though, is that Sub-word Tokenization, splitting text into sub-words, is in fact the main Text Pre-processing technique that works
- Challenges with tokenization
  - Split up at all non-alphanumeric characters
  - Tackling apostrophes ,two-word entities() e.g. “New Delhi”) ,compound words in other languages (Sanskrit, German etc.,)

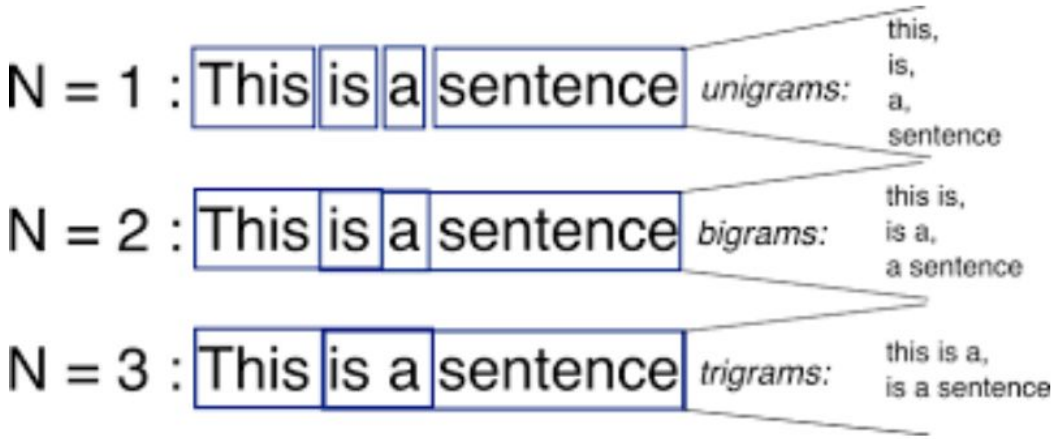
Example: Below is an example of Word Tokenization:



N-grams

- N-grams are a type of tokenization when individual sentences are split into multiple word strings or characters.
- It is a contiguous sequence of n words from a given sample of source text(corpus).
- First image depicts character N-gram and second word N-gram





Word Vectorization

- Machines, cannot really understand text as input.
- In order to perform Machine Learning on text, even after Preprocessing, we need to convert text into a numerical format that machines can understand, to find patterns and make predictions. This is referred to as Word Representation or Word Vectorization, and it is the fundamental idea behind how Natural Language Processing and also Deep Learning / Generative AI methods are able to understand and make predictions on text data.

Bag of Words (Bow)

- Bag of Words, the simplest Text Vectorization technique, is a modified version of One-hot Encoding, in which, rather than storing whether a word in the dictionary is present or absent, we store the count of that word's occurrence in the corpus.
- It Ignores grammar / structure and represents each document by measuring presence of vocabulary words.
- Bag of words does not keep order of words and hence can not be used to understand the meaning of the text.
- Before leveraging BoW, it is advised to standardize the corpus.

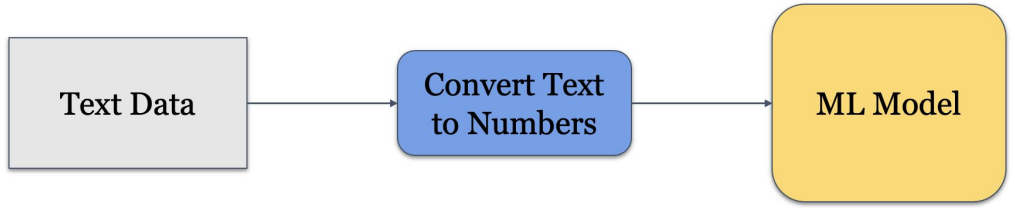
" I like cats, do you like cats? "

Word	I	like	cats	do	you
Count	1	2	2	1	1

Elementary text to numeric conversion techniques

Text to Numeric conversion

- Machine Learning algorithms understand numbers and not words hence, text needs to be converted into numbers before feeding into any modelling algorithm.
- Two basic text to numerical conversion approach are
    1. Counvectorizer
    2. TFIDF



Count Vectorizer

- Text to numeric conversion based on BoW approach.
- Total no of tokens i.e. vocabulary size will equal to matrix column length (no of columns) where each column represents a token.
- A row array represents unique tokens of the record and the number at a cell will indicate count of that token in the record.
- Drawbacks/limitation are :
  - Sparse matrix (large number of zeros)
  - High dimension of matrix
  - Limitation of BOW

Document #1

He is a good boy. She is also good.

Document #2

Radhika is a good person.

Vocabulary

a, also, boy, good, He, is, person, She, Radhika

	a	also	boy	good	He	Is	person	She	Radhika
Index	0	1	2	3	4	5	6	7	8
Document #1	1	1	1	2	1	2	0	1	0
Document #2	1	0	0	1	0	1	1	0	1

Count Vector

Term-Frequency Inverse Document Frequency (TF-IDF)

- Measures importance of a token (called as term) with respect to its record (called as document) in a corpus.
- Every term of a document is assigned a weight after multiplying its term frequency (tf) and inverse document frequency (idf)
- Internally, different libraries use slightly different formula to calculate 'idf' value though underlying idea remains same.
- The TF-IDF score for a term in a document is then obtained by multiplying its TF and IDF values
- The resulting TF-IDF score is used to rank the importance of words in a document or a set of documents.
- TF-IDF attempts to give higher relevance scores to words that occur in fewer documents within the corpus.

$$\text{TF-IDF}(t, d, D) = \text{TF}(t, d) \times \text{IDF}(t, D)$$

Term Frequency (TF):

- It measures the frequency of a term (word) in a document.
- It is calculated by counting the number of times a word appears in a document and then normalizing it by the total number of words in that document. The idea is to emphasize words that appear frequently within a document.

$$\mathrm{TF}(t, d) = \frac{\text{Number of times term } t \text{ appears in document } d}{\text{Total number of terms in document } d}$$

Inverse Document Frequency (IDF)

- This component evaluates the importance of a term across the entire corpus by penalizing words that are common across many documents.
- It is calculated by dividing the total number of documents in the corpus by the number of documents containing the term, and taking the logarithm of the result. This helps to give higher weight to terms that are rare in the corpus.
- The IDF part essentially works as a dampening factor to reduce the importance of terms that are common to a lot of documents.

$$\mathrm{IDF}(t, D) = \log\left(\frac{\text{Total number of documents in the corpus } D}{\text{Number of documents containing term } t}\right)$$

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

Examples of TF-IDF

- Let us assume that the word *dog* appears four times in a document of 1000 words
  - TF = 4/1000 = 4 × 10^{-3} = 0.004
- Let the same word appear 50 times in 1 million documents
  - IDF = log (1000000 / 50) = 4.3
- So, TF-IDF = 0.004 × 4.3 = 0.0172

- Let us assume that the word *is* appears 50 times in a document of 1000 words
  - TF = 50/1000 = 50 × 10^{-3} = 0.05
- Let the same word appear 40,000 times in 1 million documents
  - IDF = log (1000000 / 40000) = 1.398
- So, TF-IDF = 0.05 × 1.398 = 0.0699

Without IDF, *dog* would not be able to compete with *is.*

Document #1

He is a good boy. She is also good.

Document #2

Radhika is a good person.

He	1	Radhika	1
is	2	is	1
a	1	a	1
good	2	good	1
boy	1	person	1
she	1	**Total**	**5**
also	1		
Total	**9**		

$$IDF = log(\frac{Num\ of\ Docs}{Word\ in\ Num\ of\ Docs})$$

IDF(He) = log(2/1) = 0.301

IDF(good) = log(2/2) = 0

TF-IDF(He, doc#1) = 0.11 * 0.301 = 0.03311

TF-IDF(good, doc#1) = 0.22 * 0 = 0

TF-IDF(He, doc#2) = 0 * 0.301 = 0

TF-IDF(good, doc#2) = 0.2 * 0 = 0

Drawbacks of TF-IDF, CountVectorizer, Tf-IDF

- Does not keep order of words
- Can not be used to when understanding **context** is important.

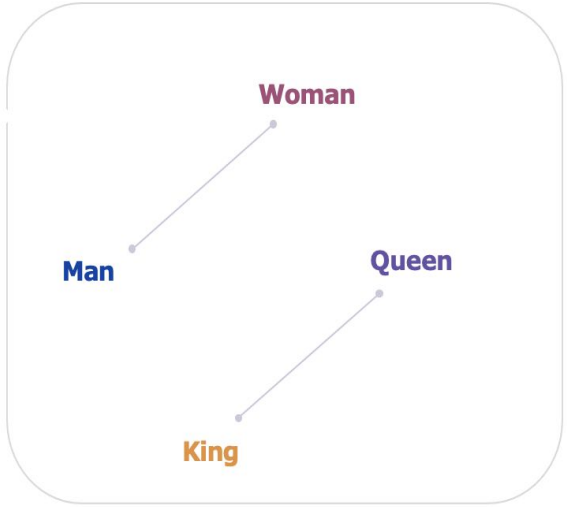
Introduction to Word Embeddings

Word Embeddings

- Word embeddings are **mathematical representations** of words in a continuous vector space, where **semantically similar words are mapped to nearby points in the space.** They are learned as part of a larger deep learning model, typically a neural network, **trained on a large corpus of text data.**

An example of how word embeddings can be understood:

	Living Being	Human	Gender	Royalty
Man	0.8	0.8	0.8	- 0.7
Woman	0.9	0.9	- 0.9	- 0.8
King	0.8	0.7	0.7	0.7
Queen	0.7	0.8	- 0.8	0.8



King - Man +Woman =?

Word Embeddings - Pros and Cons

Pros:

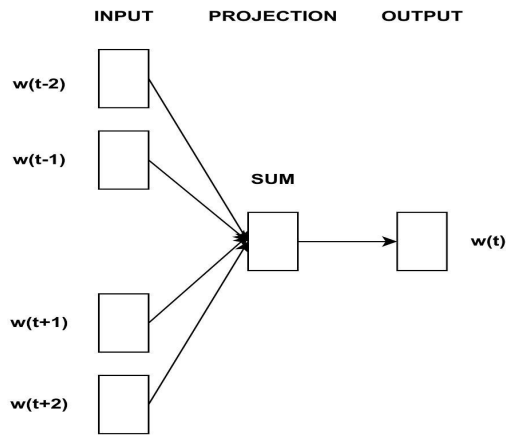
- Represents words in latent space with **semantic meaning.**
- First attempt at true **Natural Language Understanding.**
- It can **handle out-of-vocabulary words** and can generalize to unseen words in a way that one-hot encoding cannot.
- Word embedding can be used as pre - trained features which allows transfer learning and **faster model convergence**

Cons:

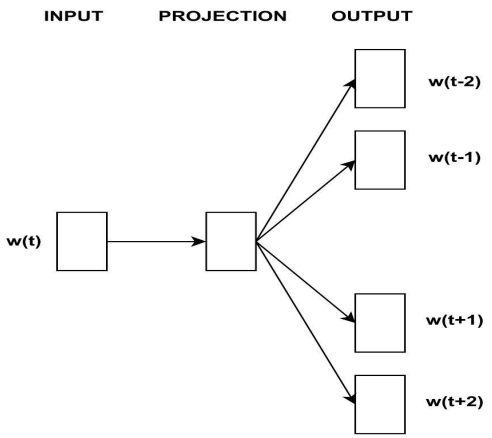
- Doesn't capture **different contexts** in which words occur.
- Does not have scope to capture **sequential** nature of language.
- Training and using word embeddings can be **computationally expensive,** particularly **for large vocabularies.**
- Bias: Word embeddings can **encode societal biases** present in the training data and may **not be suitable for certain sensitive applications.**

Word2Vec

- Word2Vec is group of related models that are used to produce Word Embeddings
- Created & patented by Tomas Mikolov and a group of a research team from Google in 2013.
- Word2Vec relies on local information of language hence the semantics learnt for a given word is only affected by its surrounding words.
- Word2vec has below **two models which are used to train words against their neighbouring words** in the corpus,
    1. Continuous Bag of Words (CBOW) - context to predict the target word
    2. Skip Gram Model –base word to predict a target context



CBOW



Skip-gram

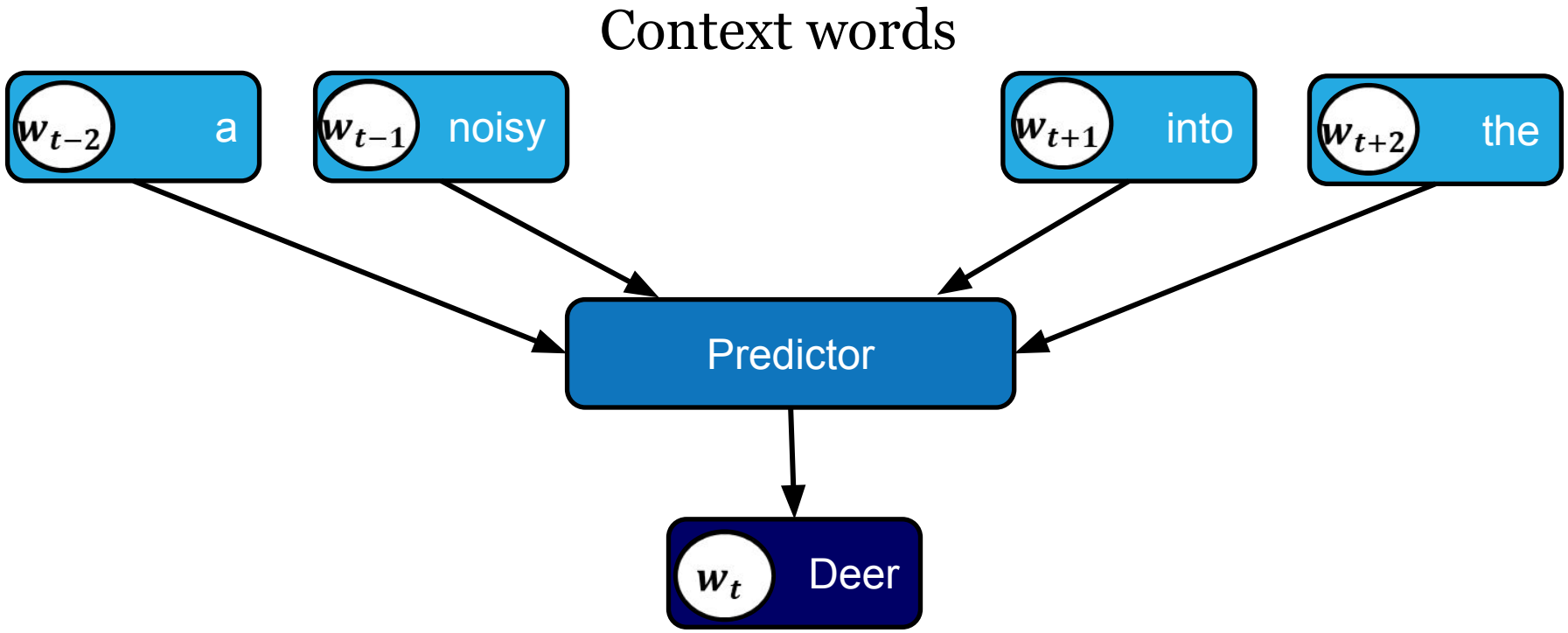
Continuous Bag of Words (CBOW)

- Predict the target word for context
- Uses sliding window concept
- As the window slides against the text, a dataset shall be generated to train a model.
- the input is a list of words, and output is the nearest word.

Content Sequence:

Window size = 1	(in, rises) (in, the)
Window size = 2	(rises, in) (Sun, in) (the, in) (east, in)

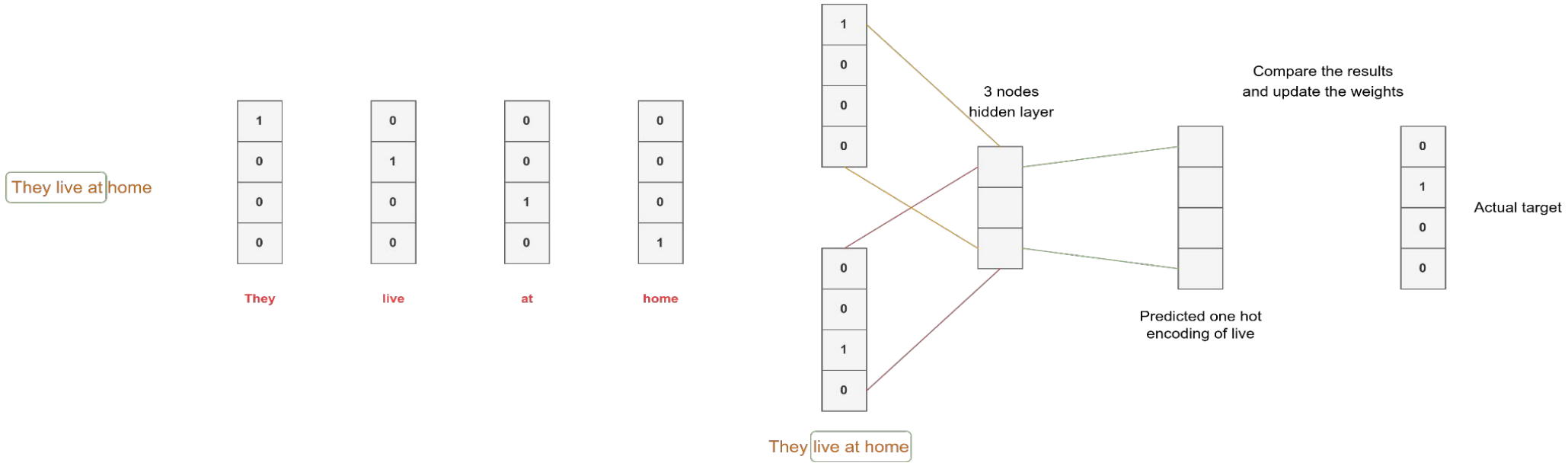
CBOW work-flow



Predict the 'Target' word

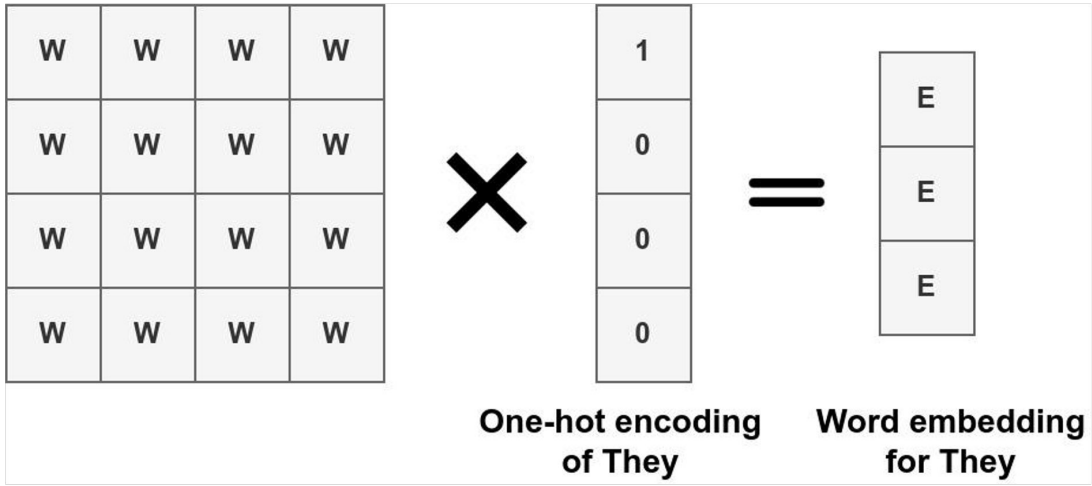
Continuous Bag of Words (CBOW)

- CBOW is trained to predict a single word from a fixed window size of context words.
- In CBOW, to predict the target word, the sum of the background vectors is used.
- The CBOW model calculates the average of the context vectors.



Continuous Bag of Words (CBOW)

- Each word is first transformed into a one-hot encoding form. Also, only particular words that are in a window will be taken into consideration rather than all the words in the sentence.
- For example, for a window size of 3, we only consider 3 words in a sentence. The middle word is to be predicted, and the 2 words surrounding it provide as context for the Neural Network. The window is then slid and the process is repeated again.
- Finally, after repeatedly training the network by sliding the window, we have weights, which we use to compute the embeddings, as shown.

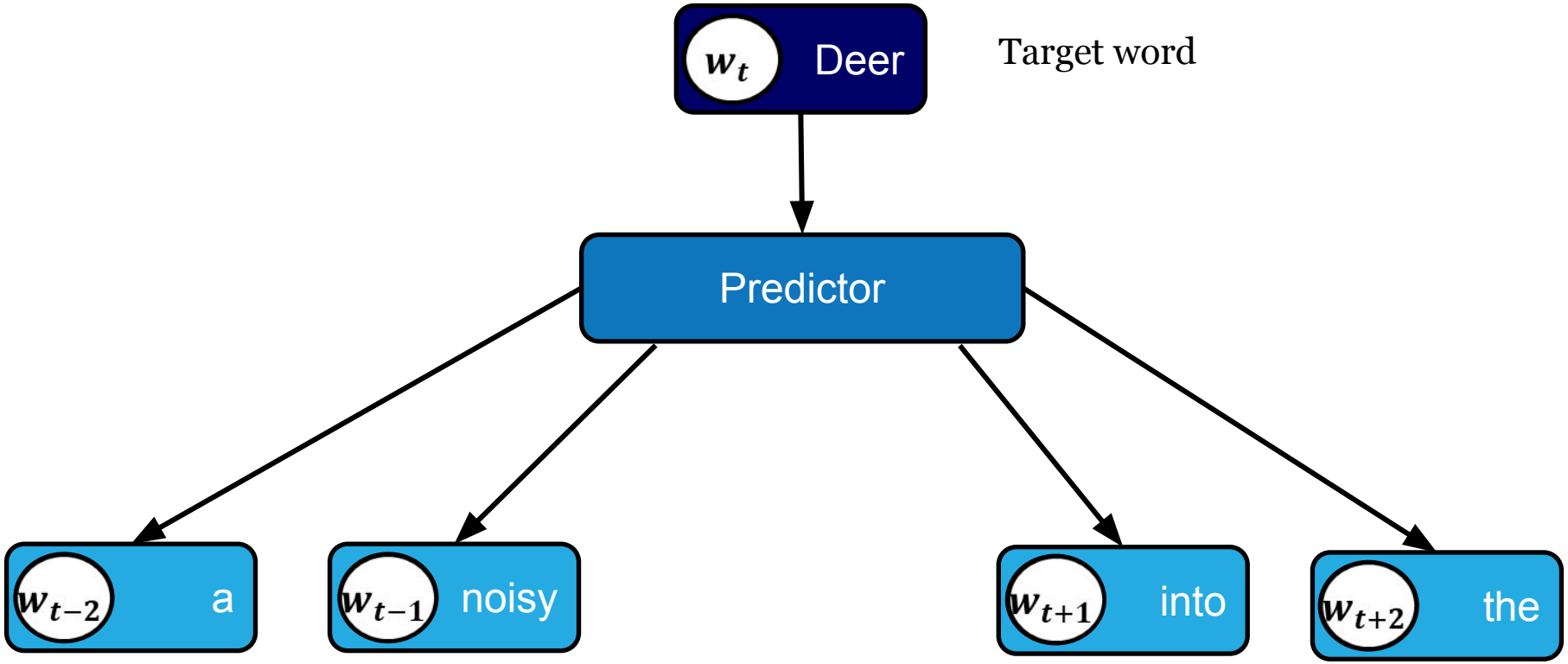


Skip-gram

- Skip-Gram is designed to predict the context from base word i.e. From a given word, Skip-gram model tries to predict its neighbouring words.
- Skip-gram is a [(target, context), relevancy] generator. Skip-gram generator gives us pair of words and their relevance (a float value).
- When we slide on the text, we can generate Context & Target pairs in which the input is a word and the output is the context words.
- The input is a word, and output will be a list of nearest words.

Words highlighted in Green are targets:	**(Context Seq, Target with window size 2)**
The Sun rises **in** **the** east	(The, ([Sun, rises, in, the])
The Sun rises **in** **the** east	(Sun, [the, rises, in, the])
The Sun rises **in** **the** east	(rises, [the, Sun, in, the])
The Sun rises in **the** **east**	(in, [Sun, rises, the, east])
The Sun rises **in** the **east**	(the, [rises, in, east])

Skip Gram work-flow



Predict the
'Context'
words

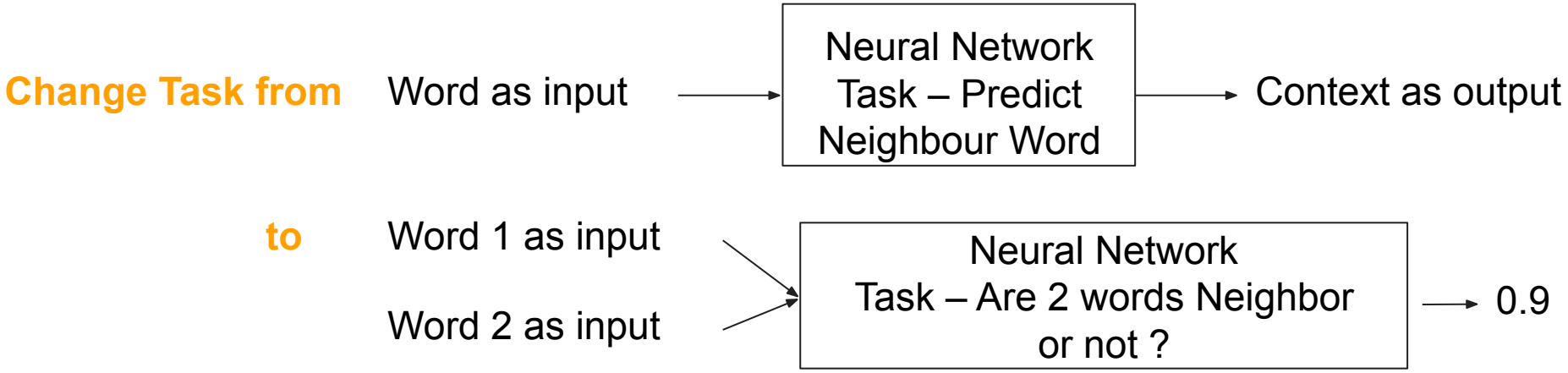
Skip-gram

- Using Skip-gram, The target word is fed at the input, the hidden layer remains unchanged, and the output layer of the neural network is replicated multiple times to match the number of context words chosen.
- Although the skip-gram model's operation is relatively similar to that of the CBOW , there are differences in the design of its neural network and the manner in which the weight matrix is produced
- After obtaining the weight matrix, the steps to get word embeddings is same as CBOW.
- If our vocabulary has a size of 10,000 words, the last output layer will be quite a big vector and we will have to predict many context words. This is computationally very expensive. We also know that generally, a word does not have more than a certain number of context words around it.
- Even rare words or phrases can be accurately represented by skip-gram when only a small portion of the training data is used.
- CBOW is found to train faster than Skip-gram and can better represent more frequent words.

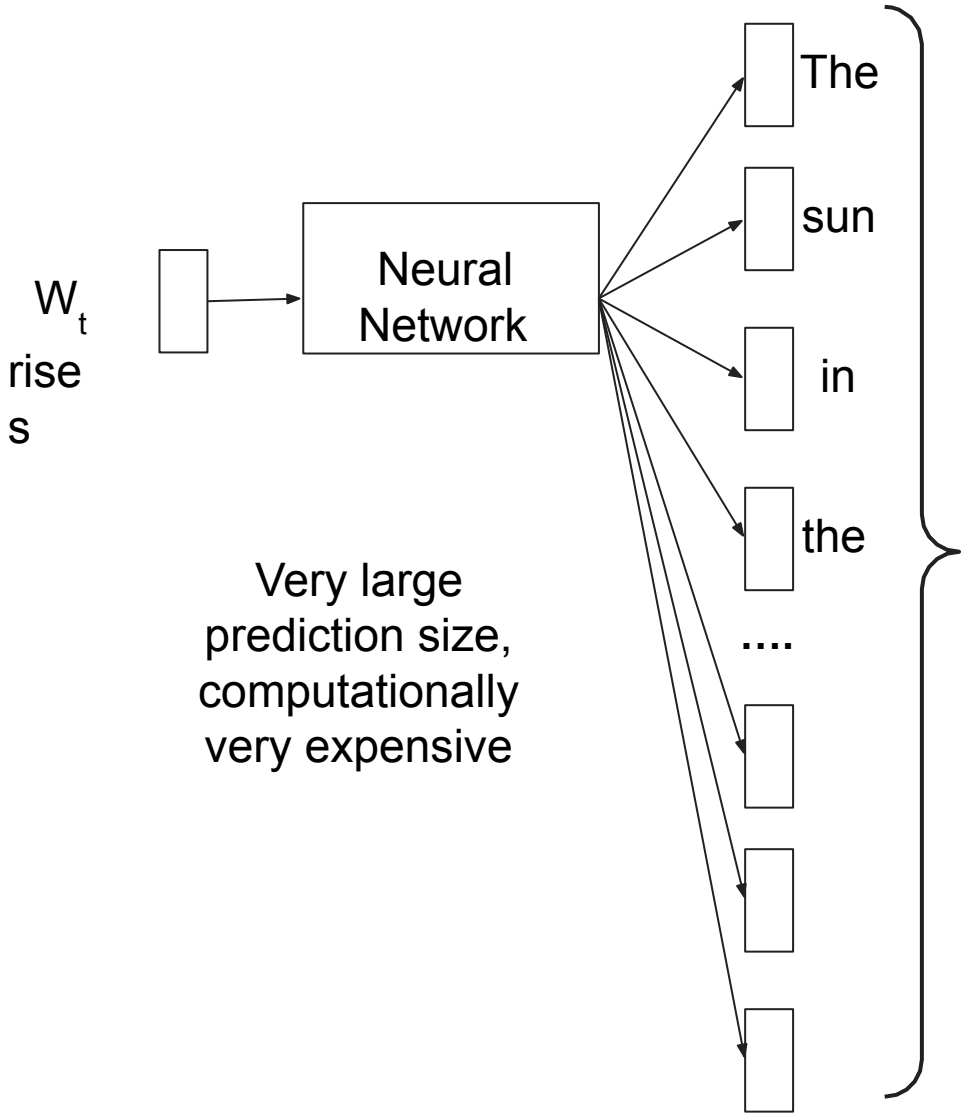
Skip-gram: Negative Sampling

- **Negative sampling** allows us to only modify a small percentage of the weights, rather than all of them for each training sample.

- We do this by slightly modifying our problem. Instead of trying to predict the probability of being a nearby word for all the words in the vocabulary, we try to predict the probability that our training sample words are neighbors or not.



Skip-gram: Negative Sampling



The Sun rise s in the east

(the, [rises, in, east])

input1	**input2** **Target**
the	rises 1
the	in 1
the	east 1

Skip-gram: Negative Sampling

- However if all samples are generated like this, we will only end up having our targets as 1. That is not desirable, as the model will have no learning capabilities

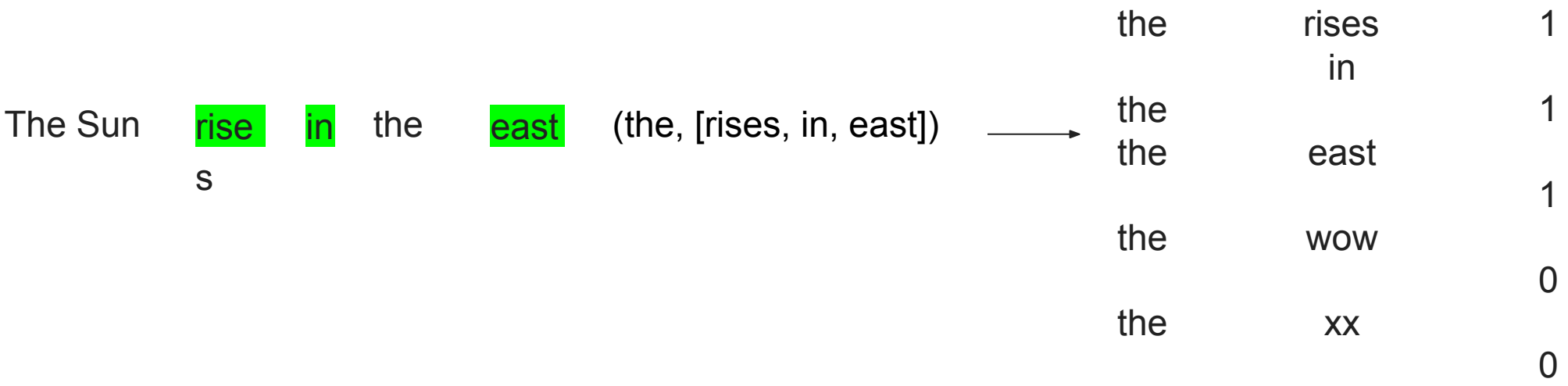
- To address this, we need to introduce negative samples to our dataset – samples of words that are not neighbors. Our model needs to return 0 for those samples.

The Sun rise in the east (the, [rises, in, east]) ▸

the	rises	1
the	in	1
the	east	1

Skip-gram: Negative Sampling

- We randomly find words from vocabulary which are already not marked as 1. We end up having:



- Now, we should be able to train the model faster with a wide understanding of what's near and what's not.

Skip-gram: Negative Sampling

- In terms of training, Skip-gram with Negative Sampling (SGNS) is slower than CBOW. However, it performs effectively with a small amount of the training data and accurately represents even uncommon words or phrases.
- Training involves 2 hyperparameters: window size and the number of negative samples.
- Smaller window sizes (2-15) produce embeddings with high similarity scores, indicating that the words are interchangeable.
- Larger window sizes (15-50, or even more) result in embeddings where similarity is more indicative of the words' relatedness.

How to train

- The objective is to maximize the probability of actual skip-gram, while minimizing the probability of no-existent skip-gram

- We try to find the probability of a presence of a particular word with a particular contextual word with a corpus,

  $\arg\max_{\theta} \pi_{(w,\, c \in D)}\, p(D = 1 \mid w,\ c\ ;\ \theta)$ …. (1)

  - Where , w is a particular word
  - C is a contextual word
  - θ is the model parameter

How to train

- Multiply 1 & 2 and taking log,

$$\arg max_{\Theta} \sum_{w,c \in D} \log \frac{1}{1+e^{-v_w v_c}} + \arg max_{\Theta} \sum_{w',c' \in D'} \log \frac{1}{1+e^{-v_{w'} v_{c'}}}$$

Softmax function — Softmax function

Where, v_w is the vector for the word w
v_c is the vector for the contextual words

- $v_w v_c$ is going to be high for contextual words
- $v_{w'} v_{c'}$ is going to be low for words with similar meaning

Word2Vec Limitations

- One of the biggest problems with Word2vec is the inability to handle unknown or out-of-vocabulary (OOV) words. If your model hasn’t encountered a word before, it will have no idea how to interpret it or how to build a vector for it. You are then forced to use a random vector.
- No shared representations at sub-word levels. If Word2vec encounters a new word which ends with say ‘less’ – for example ‘flawless’, there will be no link between less and flawless – even though we as humans know that it's probably an adjective.
- Word2vec represents every word as an independent vector, even though many words are morphologically similar.
- Scaling to new languages requires new embedding matrices and does not allow for parameter sharing, meaning cross-lingual use of the same model isn’t an option.

Global Vector (GloVe)

- GloVe stands for "Global Vectors". It is a Word Embedding project written in C language and developed by Stanford university researchers in 2014.
- Glove embedding technique is based on firstly, construction of a co-occurrence matrix from a training corpus and then secondly, factorization of co-occurrence matrix in order to yield word vector.
- A co-occurrence matrix is an NxN matrix where each row and each column represents a unique word in a given corpus. Each entry in the co-occurrence matrix represents the number of times the word ‘i’ has occurred with the word ‘j’.

	I	enjoy	deep	nlp	learn	fly	like	.
I	0	1	0	0	0	0	2	0
enjoy	1	0	0	0	0	1	0	0
deep	0	0	0	0	1	0	1	0
nlp	0	0	0	0	0	0	1	1
learn	0	0	1	0	0	0	0	1
fly	0	1	0	0	0	0	0	1
like	2	0	1	1	0	0	0	0
.	0	0	0	1	1	1	0	0

GloVe : Co-occurrence Matrix

- Each entry in the co-occurrence matrix represents the number of times the word ‘i’ has occurred with the word ‘j’.
- The relationship of the words can be studied by observing the ratio of their co-occurrence probabilities.
- Let P(k|ice) be the probability of observing the word ‘k’ with ‘ice’, and P(k|steam) be the probability of observing the word ‘k’ with the word steam.
- In below example, one might assume, ice co-occurs with solids more frequently than with gases, but steam co-occurs with gases more frequently than with solids. Both words usually occur with their shared property water, and both occur infrequently with the unrelated word fashion. Large values (much greater than 1) correlate well enough with ice-specific features, while small values (much less than 1) correlate strongly with steam-specific qualities.
- In this way, the probability ratio encodes some basic meaning connected to the general idea of thermodynamic phase.

	k = solid	k = gas	k = water	k = fashion (random)
P(k|ice)	high	low	high	low
P(k|steam)	low	high	high	low
P(k|ice)/ P(k|steam)	>1	<1	~1	~1

Glove

- GloVe attempts to generate word vectors by using the global co-occurrence relationship. GloVe's training objective is to learn word vectors whose dot product equals the logarithm of the probability of occurrence of the words.

- In the previous slide, we saw how word-to-word co-occurrence probabilities have the potential to encode some form of meaning by looking at an example where we captured certain information about the thermodynamic phase of matter using probability ratios.

- As these word-to-word co-occurrence probabilities have the potential to determine meaning, they are also encoded in the vector representation.

- Therefore, the embeddings created using GloVe capture a significant amount of semantic and syntactic meaning.

GloVe : Global Vectors

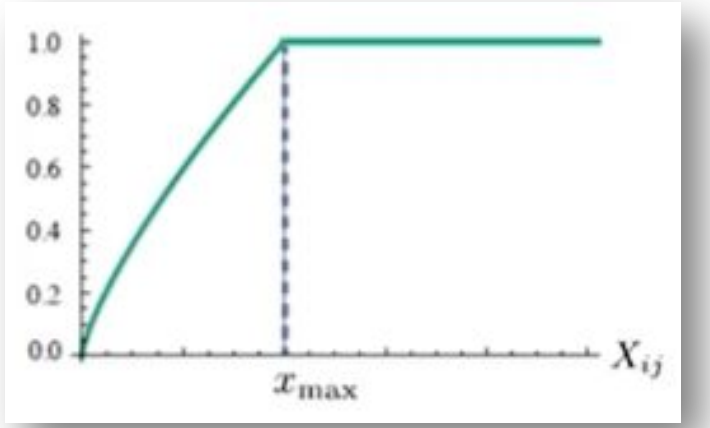
- GloVe captures word-word co-occurances in the entire corpus better
  - GloVe models is F($(w_i - w_j)^T w_k$) = P_{ik} / P_{jk}
  - Where , Let X_{ij} be the co-occurrence probability of words indexed with I and j
  - X_i be $\sum_j X_{ij}$ and
  - P_{ij} = p(j|i) = X_{ij} / X_i

GloVe explanation

- Cost function :

  $J = \sum_{i,j} f(X_{ij})(w_i^T \tilde{w}_j - \log X_{ij})^2$

- For words i , j co-occurrence probability is X_{ij}
- A weighing function f



  - Suppresses rare co-occurences

Word2Vec Vs GloVe

- Both models differ in the way they are trained, and hence they output different word vectors.
- The GloVe model is based on global word to word co-occurrence counts taking the whole corpus into consideration, whereas Word2vec uses co-occurrences of local context (neighbouring words).
- GloVe learns embeddings by constructing the co-occurrence matrix - on the other hand the Word2vec model learns by making predictions by taking context words as inputs and predicting the target words.
- GloVe is thought to capture the semantic as well as the syntactic meaning of the words in its embeddings better than Word2vec.

FastText

- FastText is an open-source, free, lightweight library that allows users to learn text representations and text classifiers. It works on standard, generic hardware. Models can later be reduced in size to even fit on mobile devices.
- Released and opensourced in 2015
- Developed by Meta (Facebook)
- Provides Word embedding Pre-trained on English webcrawl and Wikipedia data
- Refer https://fasttext.cc/docs/en/supervised-tutorial.html
- Fast-Text is an extension of word2vec library.
- FastText assumes a word to be formed by a n-grams of character. For example, sunny is composed of [sun, sunn,sunny],[sunny,unny,nny]... etc, where n could range from 1 to the length of the word The values of the representations are then averaged into one vector at each training step. While this adds a lot of additional computation to training it enables word embeddings to encode sub-word information.
- For previously unseen words, typo errors, and OOV (Out Of Vocabulary) words the model can make an educated guess towards its meaning.

FastText

- Each word is represented as the average of the vector representation of its n-grams along with the word itself.
- Consider the word “equal” and n=3, then the word will be represented by charcter n-gram

<eq, equ, qua, ual,al > and <equal>

So the the word embedding for the word ‘equal’ can be given as the sum of all vector representation of all of its character n-gram and the word itself .

Pre-trained Word Embeddings

- Pretrained word embedding which includes word vectors for a vocabulary of millions words and phrases that they have trained on internet dataset using Word2Vec/GloVe/FastText.
- Through , Transfer learning technique, pretrained word embedding are leveraged in NLP
- Saves compute time and ensure model generality .

Machine Learning NLP Pipeline