Natural Language Processing (NLP)

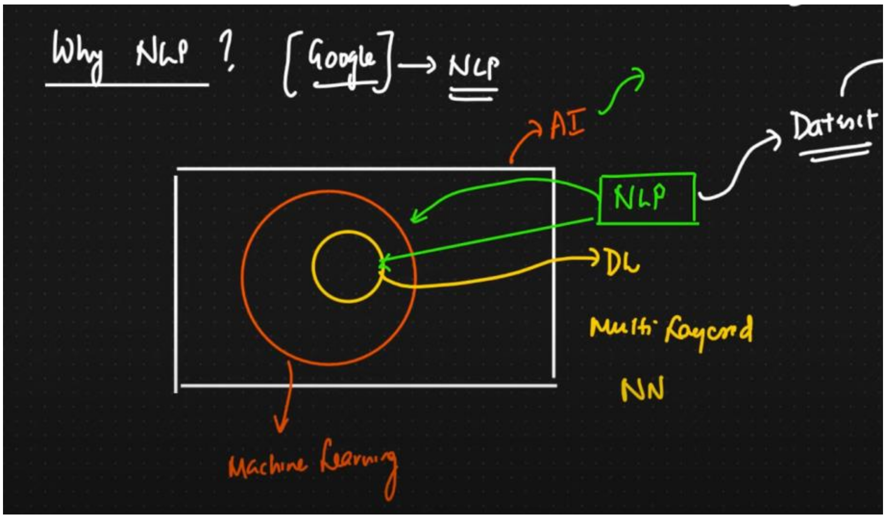
Contents

Why NLP?

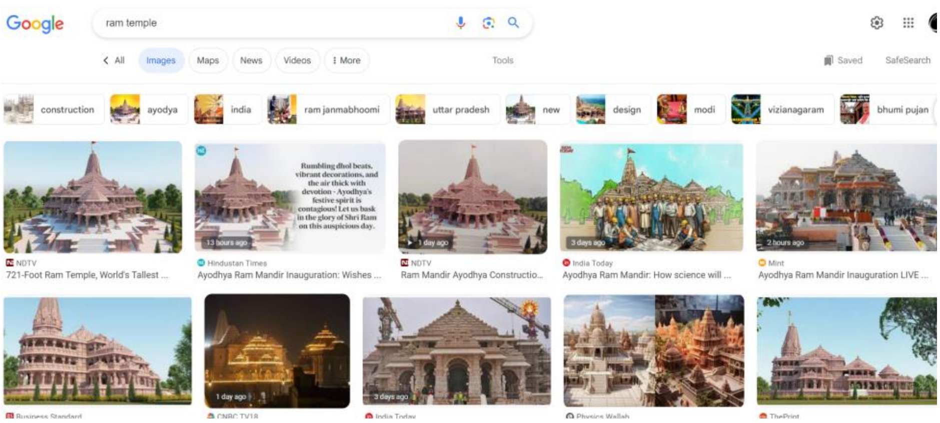
- Content recommendation like news ads based on profile and interest.
- Sentiment analysis
- Prediction of next text for Generative AI
- Text summarization, language translation
- Spam classification
- Chatbot

NLP can be used both machine learning and deep learning. At the end of the day we are creating a AI application.

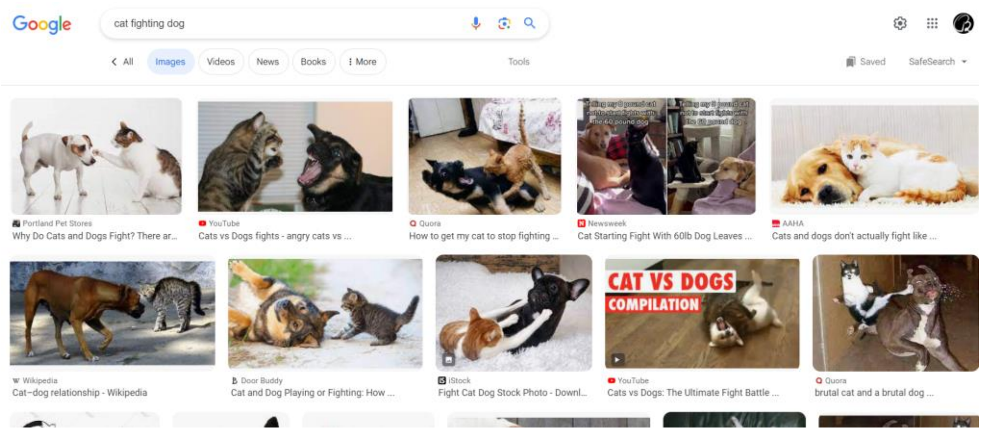
We will see how words can be converted to vector how machine is able to understand text and give output.



Example: Say we are searching in google as ram temple



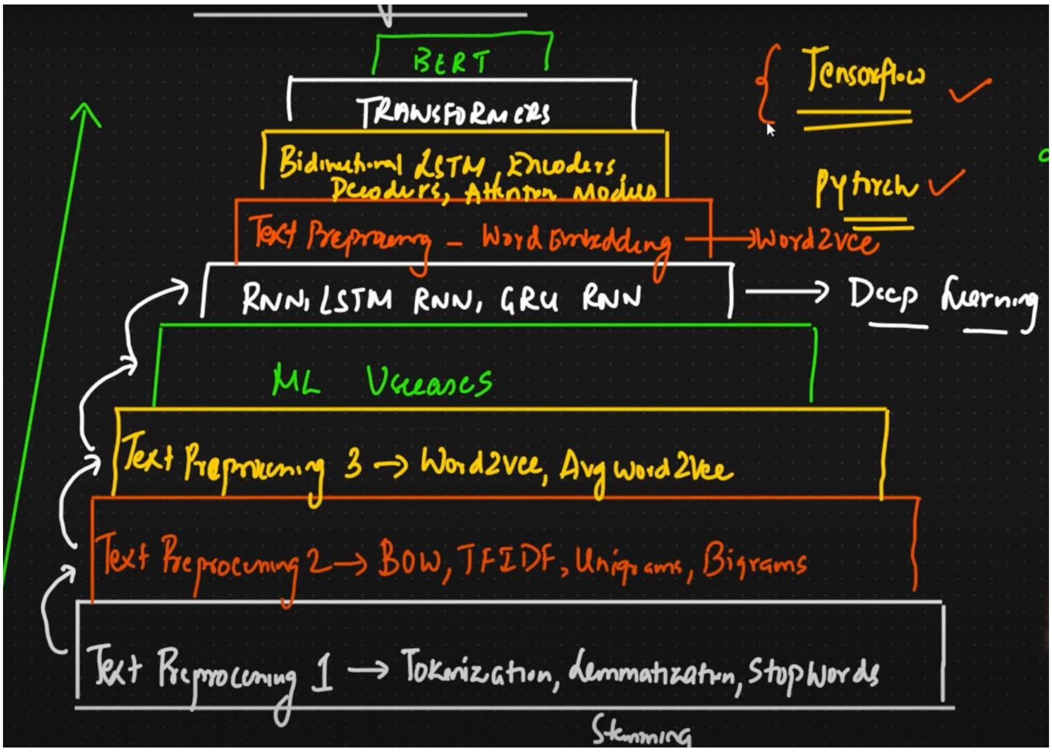
Now how all the images are associated with these images mean in the background text to image conversion is happening that might be using NLP.



Current limitation of NLP

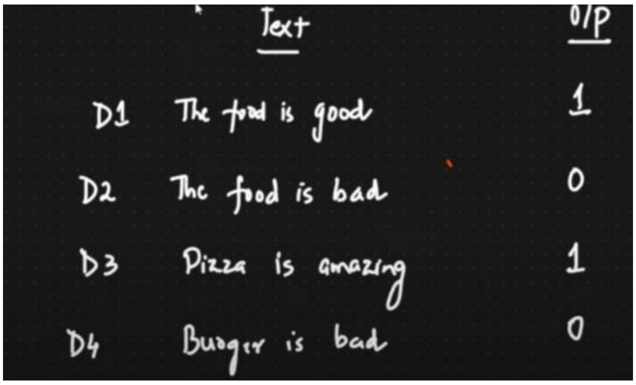
NLP not able to understand sarcasm in text. Machine is not able to understand sarcasm properly, so research is going on.

Roadmap of NLP



Basic Terminology used in NLP

1. CORPUS
2. Documents
3. Vocabulary
4. Words



CORPUS: It's like a **paragraph** like entire data set if we club together. Say my CORPUS is [D1,D2,D3,D4]

Document: Each **sentence** in the paragraph can b called as Document in NLP like D1 D2 etc. Though it is just a sentence but for NLP it is document. e.g. The food is good => D1

Vocabulary: It is the collection of all **unique words** in the CORPUS.

Say in my CORPUS we have 10K unique words in this case my vocabulary will be

The, food, is, good, bad, Pizza, Amazing, Burger

Words: All words in the CORPUS.

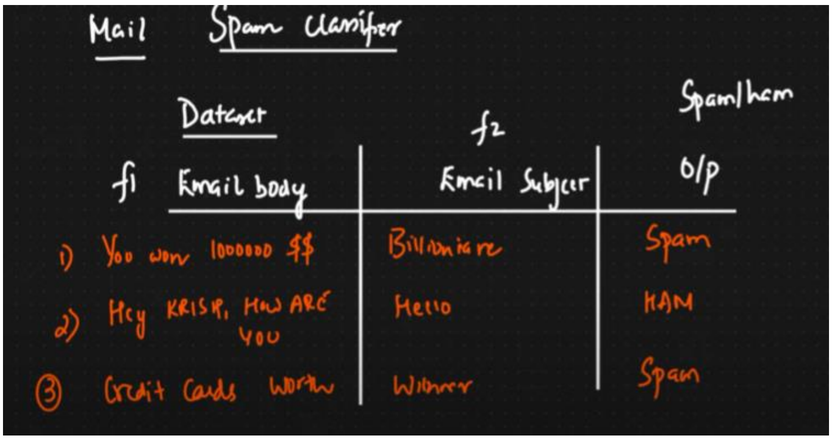
Preprocessing of text data

We need to do preprocessing of raw text before using it for NLP we will be using **NLTK** (Natural Language Toolkit) libraries for doing pre-processing. Here is the way we can do,

Code Link: https://github.com/gourabb8273/ML-Model-HandsOn-Hub/tree/master/NLP

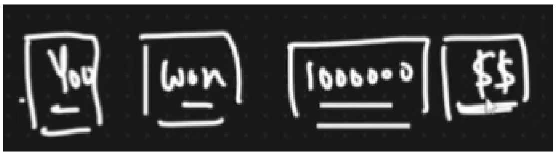
Tokenization

Say we are using spam classifier. Our data set will be like



The first step here would be tokenization. It's about converting sentence into word so that those should be understandable by machine learning algorithm.

Sentence: You won 1000000 $$ after tokenization it will be



Stop words

Next step is stop words. Say we have a sentence like

"Hey buddy I want to go to your house" here we can see some words like **to** are not making any sense in prediction so we can remove those words.

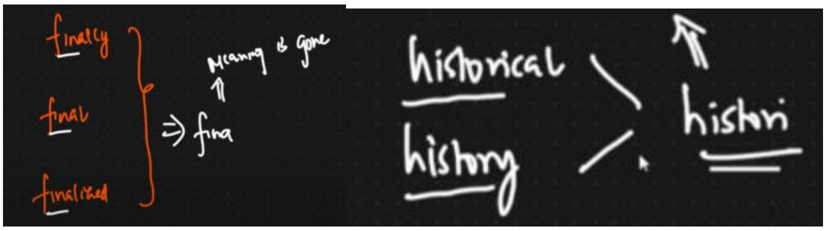
Example of stop words: to, he she, is for of etc.

To remove all these words which are insignificant in output prediction using stop words. We can create our own custom stop words list as well. However we may need this in text summarization but not needed for spam classification or sentiment prediction.

Stemming

Here if we have say many similar types of words in different form we can convert those in base or root word thus we can reduce the number of words. It's a process of reducing words to the base word.

But base word may or may not have any meaning.



Advantage: It's very fast can be applied on large dataset

Disadvantages: It is removing the meaning of the word

Use cases:

- Spam classification
- Review classification

To overcome it we have lemmatization.

Lemmatization

It has entire dictionary of words it will work the same way like stemming but it will give meaningful word.

Advantage: We will be able to get meaningful word.

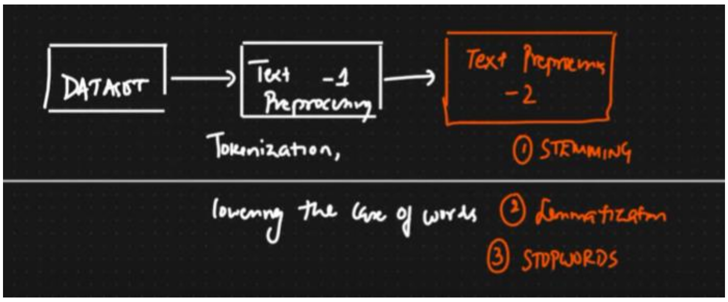
Disadvantages: It is slow as it has to do lots of comparison.

Use cases:

- Text summarization
- Language translation
- Chatbot

Lowering the case of Words

We will be now lowering the case of all word.

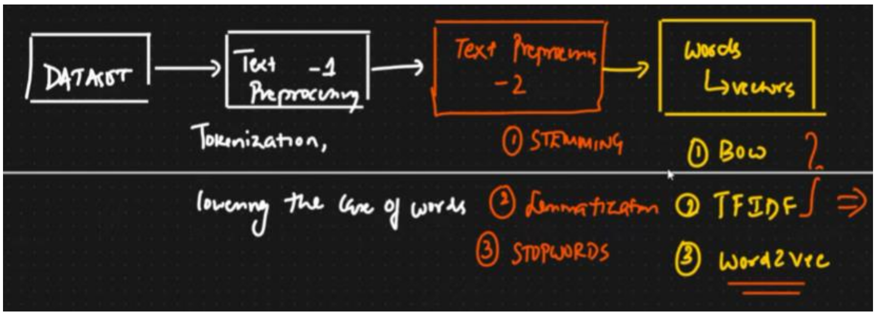


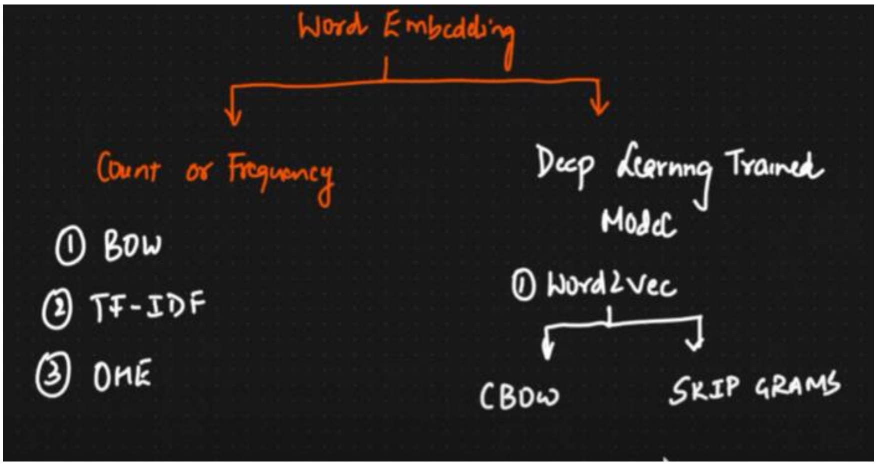
Word Embedding

In the next step we will be converting words to vector using different techniques, Word embedding is a technique to convert **words into vector**.

1. OHE (One Hot Encoding)
2. Bag of words (BOW)
3. TF-IDF (Term Frequency Inverse Document Frequency)
4. WordtoVec

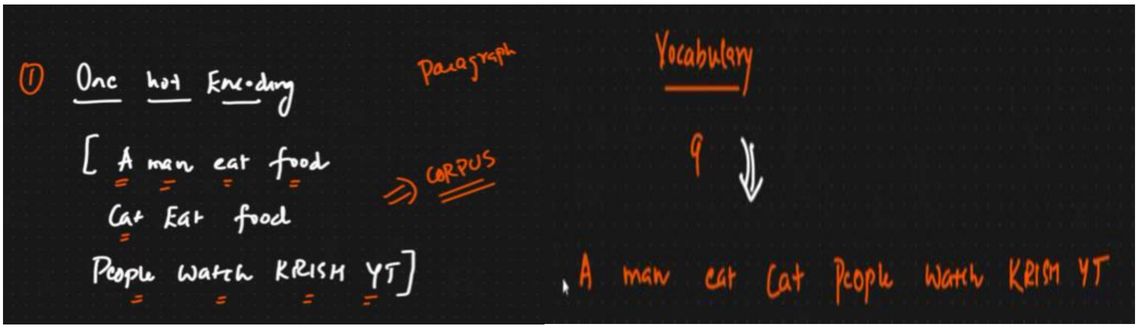
a. CBOW (Continuous Bag of Words)
b. Skip Gram





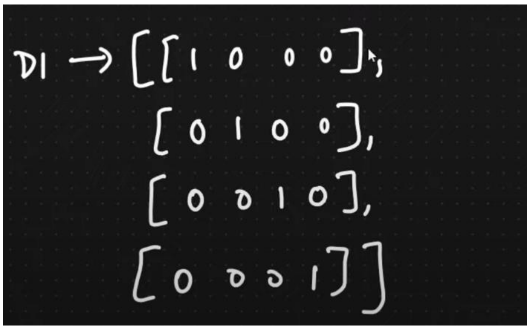
One Hot Encoding

Say my CORPUS is



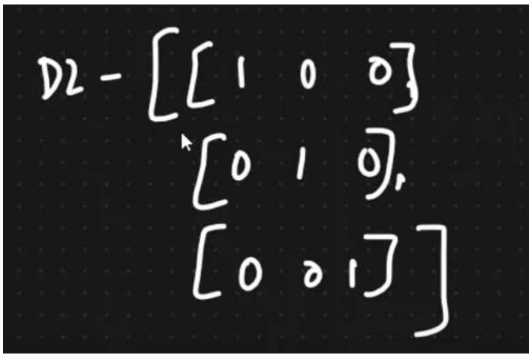
Now each word will be represented as one hot encoding. After doing this we will get as follows

For first document D1 – **A man eat food** we will get.



Here we have 4 words

For the second document D2 – **Cat Eat Food** we will get.



Here we have only 3 words so if vocabulary size is decreases then we can't train the model. Input feature should be fixed constant this called out of vocabulary.

Advantage:

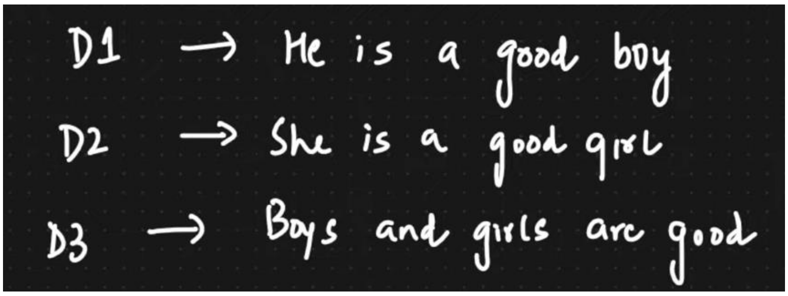
- Simple to implement.
- Intuitive

Disadvantage

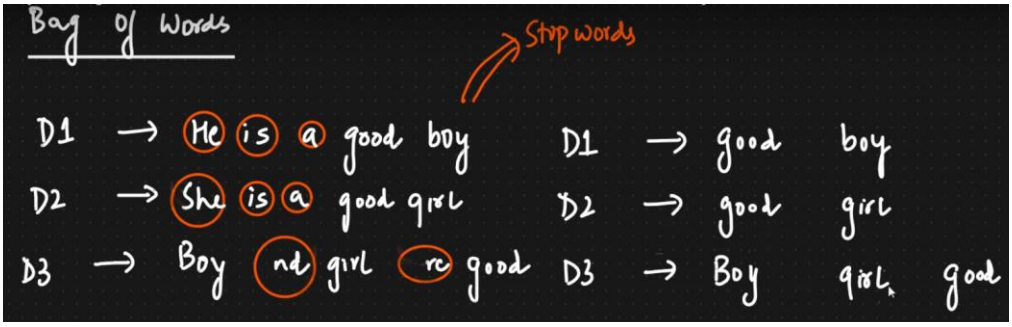
- For large set of vocabular size of sparse (many zero values) matrix will be large difficult to compute
- OOV (Out of vocabulary), my sentence is not fixed size. Any new extra words if we get can't be handled.
- Between the word semantic meaning is not captured like how two words related as each row only one column is 1 rest zero.

Bag of Words (BOW)

It is a technique of converting **text to vector**. Say my CORPUS is as follows,

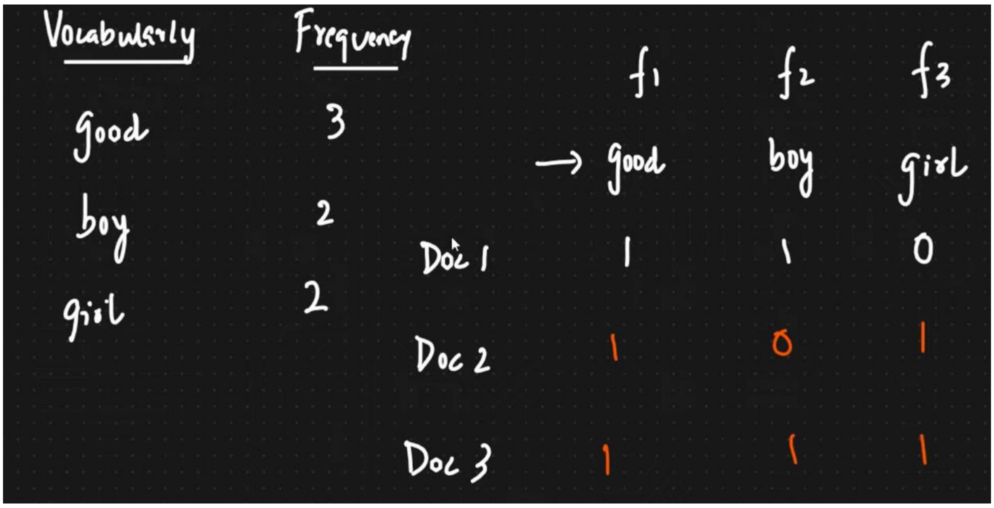


Now we will use stop words and remove unnecessary words also will lower all case.



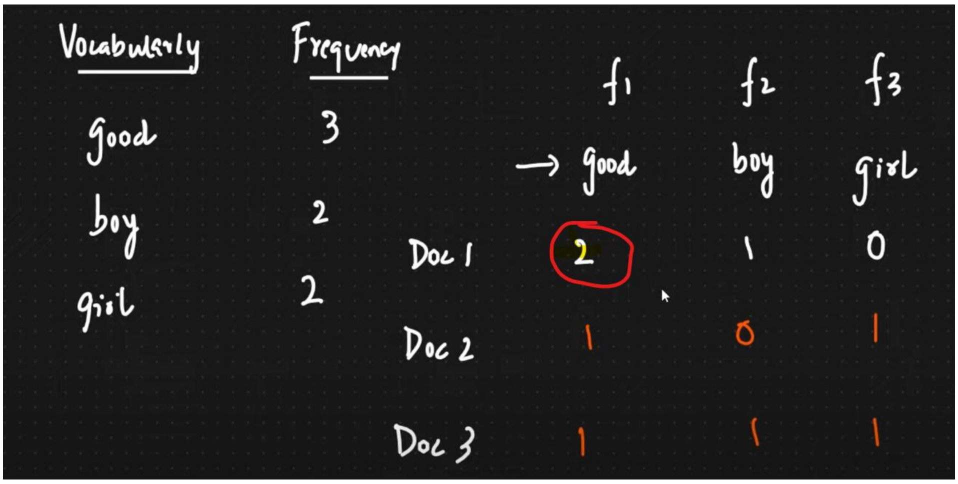
Now my vocabulary unique words will be my feature. Let's see that with frequency like no of time words are repeating,

Order of the feature will be based on frequency e.g. if **good** is repeating more time than other then first feature f1 will be **good**.

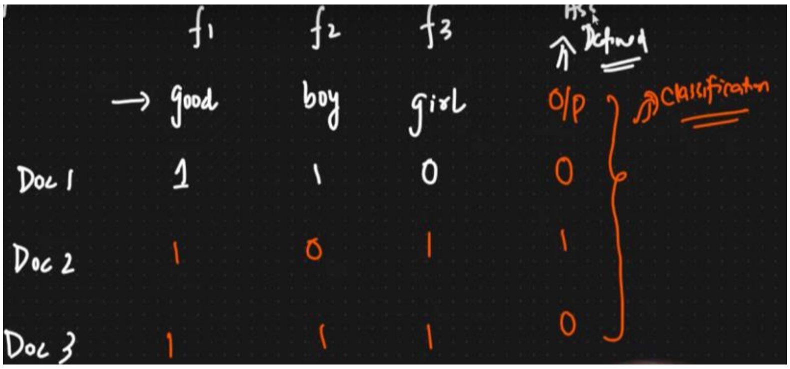


We will just increase the count in the feature based on the document. Like first doc was good boy so f1 and f2 became 1 and f3 is 0.

Say in Document 1 we have one more good like – **good boy good** then count of **f1** will be increased by 2.



Binary Bag of Words: In Bag of Word there is an option called **Binary Bag of Words**. Like if any count more than 1like f1 then also we will make it 1.



Advantage:

- Simple and Intuitive

Disadvantage

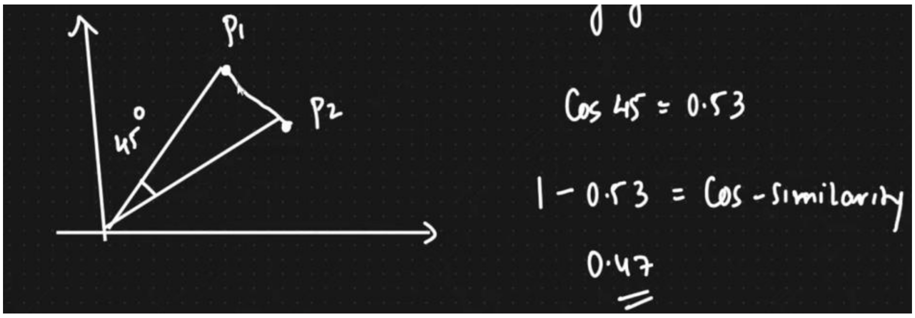
- Sparsity is still there though improved but not fixed.
- OOV (Out of vocabulary), if we get any new word like say **cat** then we won't be able to handle in feature so entire word will get rejected so meaning will get changed.
- Ordering of the word completely changed as we are doing ordering based on max frequency so won't be able to make semantic relationship.
- Semantic meaning not captured well like good and better are similar word so how similar words are.

If we take those features of 3 dimensions and plot it we will be able to calculate distance between them to find the similarity

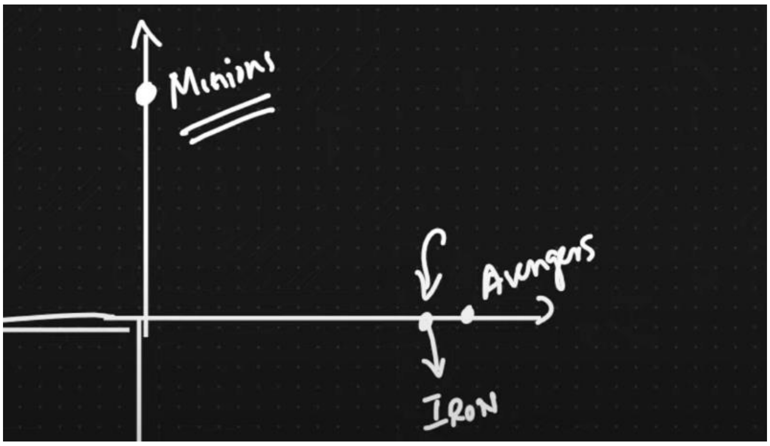
- Euclidian Distance
- Cosine Similarity

Cosine Similarity

Say we have two point and we need to find out the similarity we will find the angle between them and subtract from 1. So P1 and P2 is 47% similar.

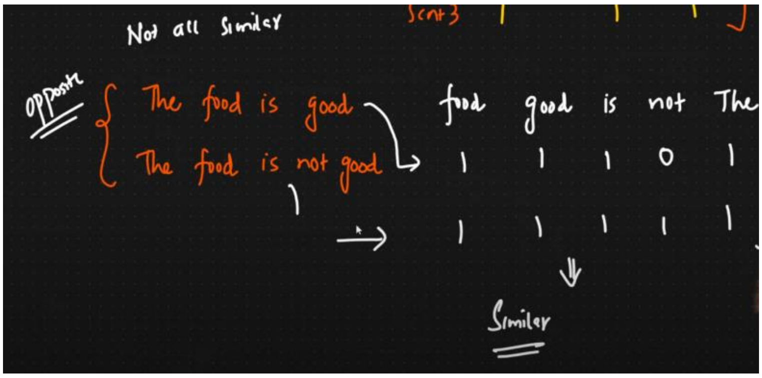


Say another example,



If the cosine similarity is close to 0 value then words are very similar that happens when angle will be less between them

This is heavily used in **recommendation engine** here avenger and Iron man are similar, but Minions is different as they are making 90-degree angle between them.

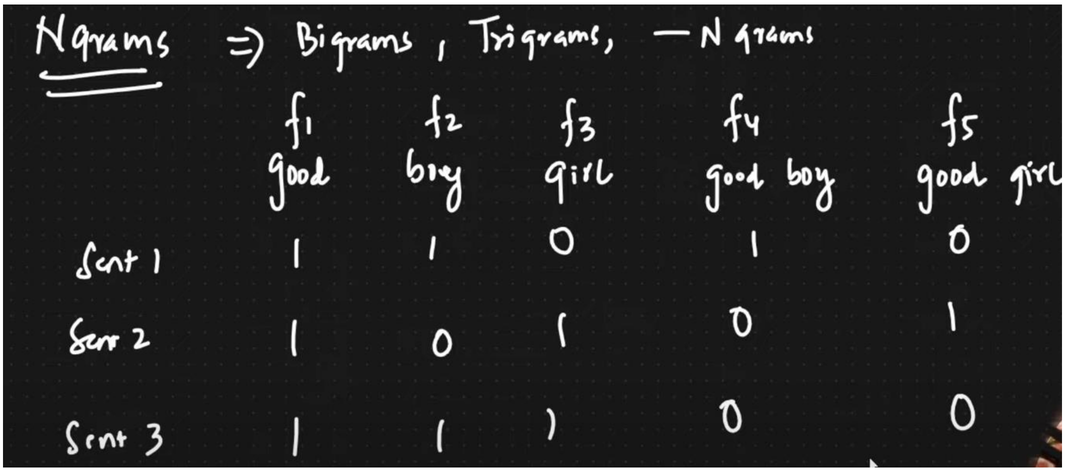


In the above example both the sentence or **documents are different like opposite** but the vector **looks similar** if we find cosine similarity that's the main problem **semantic meaning is missing** like which words should I focus more that will be **fixed using TF-IDF.**

Capturing semantic Information using Ngrams

- Bigrams (combination of two words)
- Trigram (Combination of three words)
- Ngrams (Combination of n words)

Apart from single feature we will be using combinations of features to understand semantic meaning of words.



We are adding two more features f4 and f5 using **Bigrams**.

How many **Bigrams** will be possible for word **I love food**? **I love and love food** only two bigram I food can't be possible.

How many **Trigram** will be possible for word **I am not feeling well**?

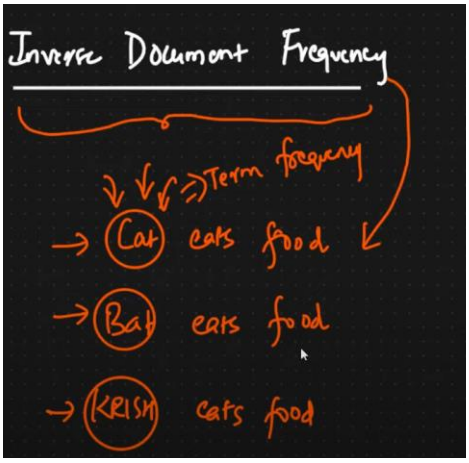
If we say combination of **Ngrams is (1,3)** then we will include single feature (unigram), Bigram and trigram 1 to 3. Say doc is I am feeling well my feature will be.

- f1 – I,
- f2 – am,
- f3- feeling,
- f4- well,
- f5-I am,
- f6-am feeling,
- f7-feeling well,
- f8-I am feeling,
- f9- am feeling well

TF-IDF (Term Frequency - Inverse Document Frequency)

It has two important concept **Term Frequency** and Inverse **Document Frequency**. Here we are trying to solve the issue i.e. semantic meaning of the words to know which word we need to focus more on.

Whichever words are **rarely presen**t in the sentence we will give **more weightage**.



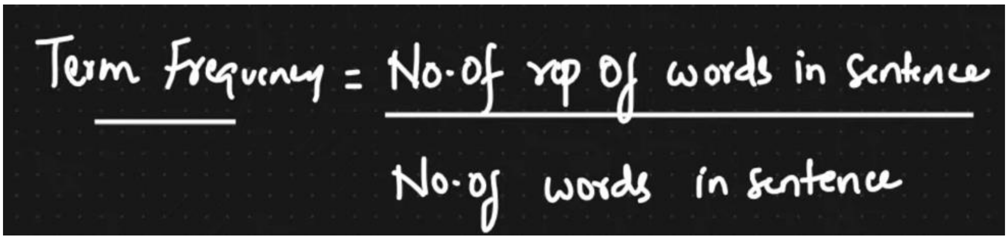
Here rare words are Cat, Bat and Krish rarely present in the sentences so we need to give more weightage.

- These **rare** words will be captured by **Term Frequency** which will have more weightage.
- **Common** words will get captured by **Inverse Document Frequency.**

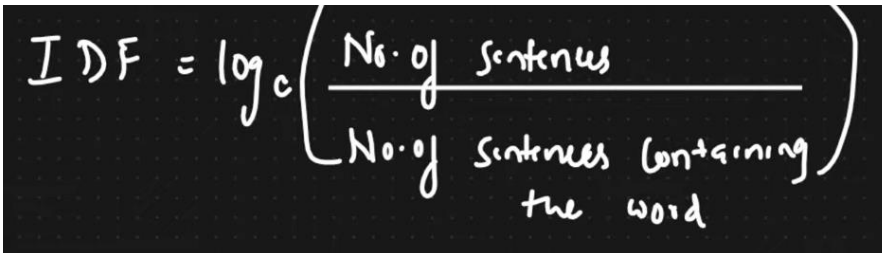
When we combine both semantic meaning will be there.

Term Frequency:

No of repetition of word in sentence / No of words in sentence



Inverse Document Frequency:

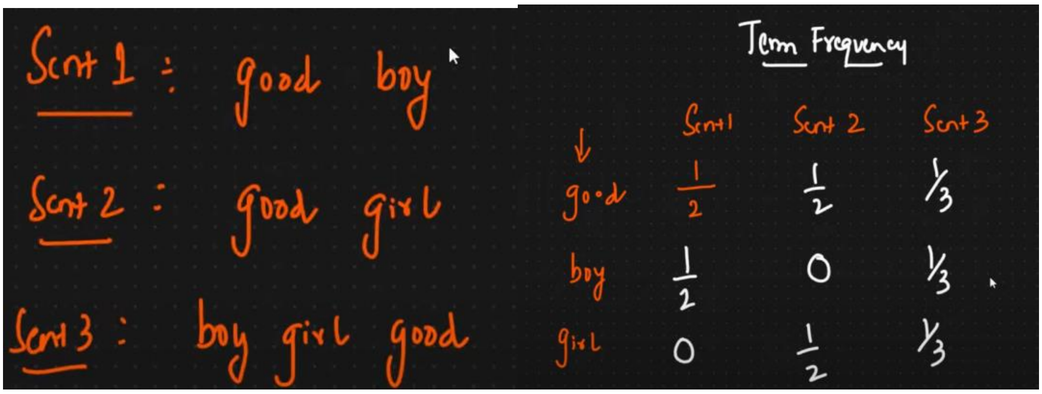


By multiplying both we will get TF-IDF

Example

Find Term Frequency

Here is my sentence.

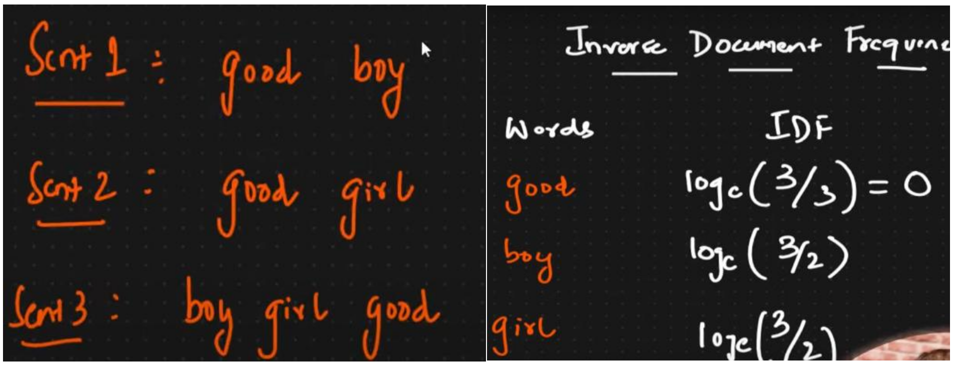


Here for word good in sentence 1 only one time is present and total word in sentence 1 is 2 hence TF is ½.

For boy in sentence-3 only one time is present total word in sentence is 3 so TF is 1/3.

Find Inverse Document Frequency

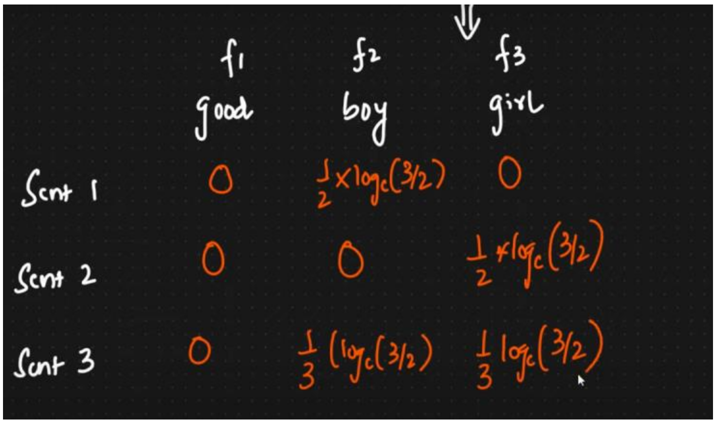
This gets computed for every sentence.



Here no of sentence is fixed that is 3 and good is appearing in all 3 sentences so good is common word and log (3/3) =0 so no weightage.

Similarly for word boy it is appearing only in sentences so log (3/2)

If we multiply both then we will get TF-IDF



Now we will multiply TF and IDF in feature space like for good in sentence 1 we had ½ and from IDF it is 0 so it will be ½*0 =0

- ✓ Thus, we are removing or reducing weightage for common words and increasing weightage for rare word so some semantic meaning is captured.

Advantage

- Intuitive

- Word important is getting captured like semantic meaning.

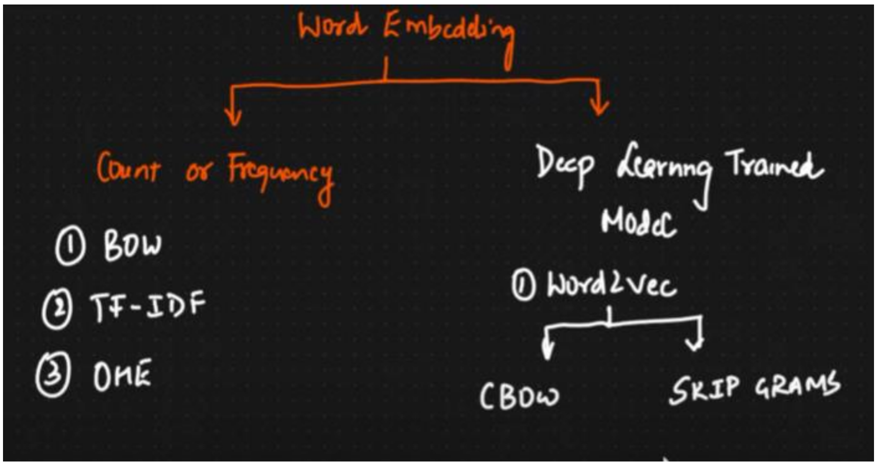
Disadvantage

- Sparsity is still there for large data set though better than BOW.
- Out of Vocabulary is still there.
- Semantic meaning not captured well like good and better are similar word so how similar words are.

Word2Vec

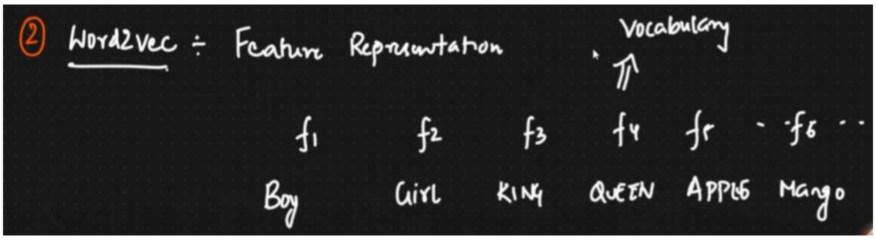
https://github.com/gourabb8273/ML-Model-HandsOn-Hub/blob/master/NLP/Practicle/text_preprocessing_Word2Vec.ipynb

Now we have seen few technique for creating word embedding based on count of frequency now it's time to see another popular technique using Deep Learning Trained Model. Those are CBOW and Skip Gram.



The main two problem that is **sparsity** and **semantic meaning** of words like **how similar two words** will be fixed in word2Vec technique. We will have embedding layer that convert text to vector.

Say we have some features like words in sentences.



- Every word we will create a vector but of **limited dimension** like 100 or 300 dimensions within that many dimensions we will represent the entire word.
- Sparsity will be reduced we won't found zero value all will be non-zero value.

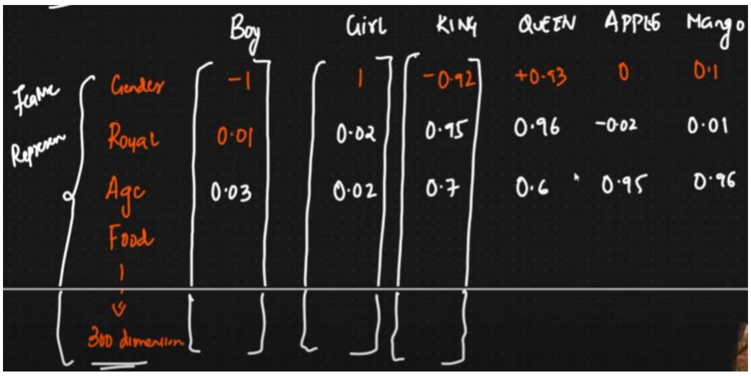
- Semantic meaning will be captured like if two words say good and better are similar then in vector also difference in value will be very less.

Feature Representation

Every word like Boy Girl will be represented as some no of features.

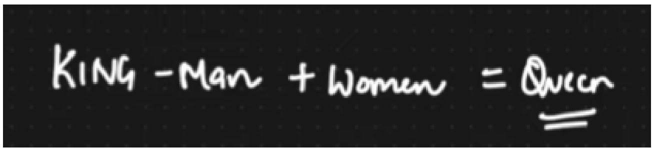
- Say we have 300 dimensions of word or feature like Gender, Royal Age etc and every word of my input document will be represented as weightage of those feature in vector format or relate with those 300 dimensions of feature. Those 300 dimensions of feature will be created by word2vec algorithm.
- Google has done this with more than **3 billion** no of dimension of feature.

- In the row we have feature or vocabulary from our corpus.
- In the column we have 300 dimension of feature those are created by word2vec algo
- We are having weightage or how my vocabulary is related to those features. **Royal is related to King so 0.95 say.** Thus, we are identifying semantic meaning of the words.



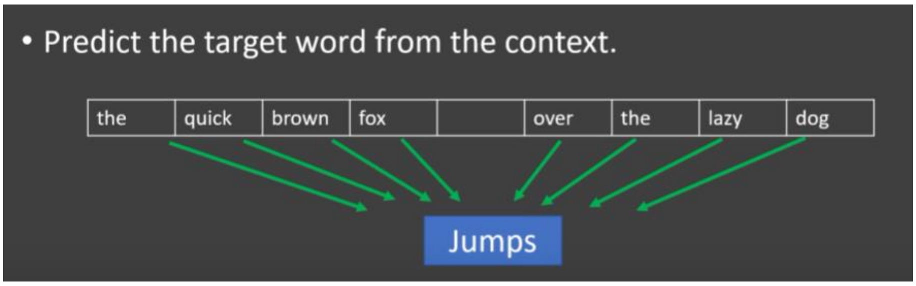
- Here each of my words like **Boy represented as vector of having limited dimensions like here 300** through we are representing based of 300 features. Ideally words will be very high dimensions like billion.

Say assuming we have only 2 feature dimensions and we have words like King, Queen, Man, Women so we will get **King - Man + Women = Queen** using vector addition and subtraction or **cosine similarity** we can also use Euclidian distance or Manhattan distance.



CBOW (Continuous Bag of Words) Architecture

Here we predict the target word from context.



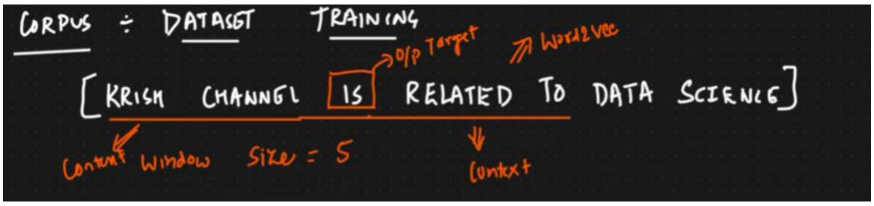
Say we have a Copus collection of sentences.

- We need **window size** for creating our training data.
- With respect to training data we will have independent feature and output feature.

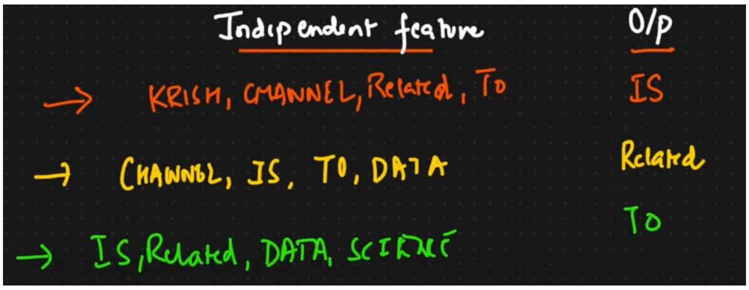
Example:

Assume window size is 5 (choose odd number).

So according to CBOW **centre word** will be the **output** or target and before and after words will be independent feature or context.



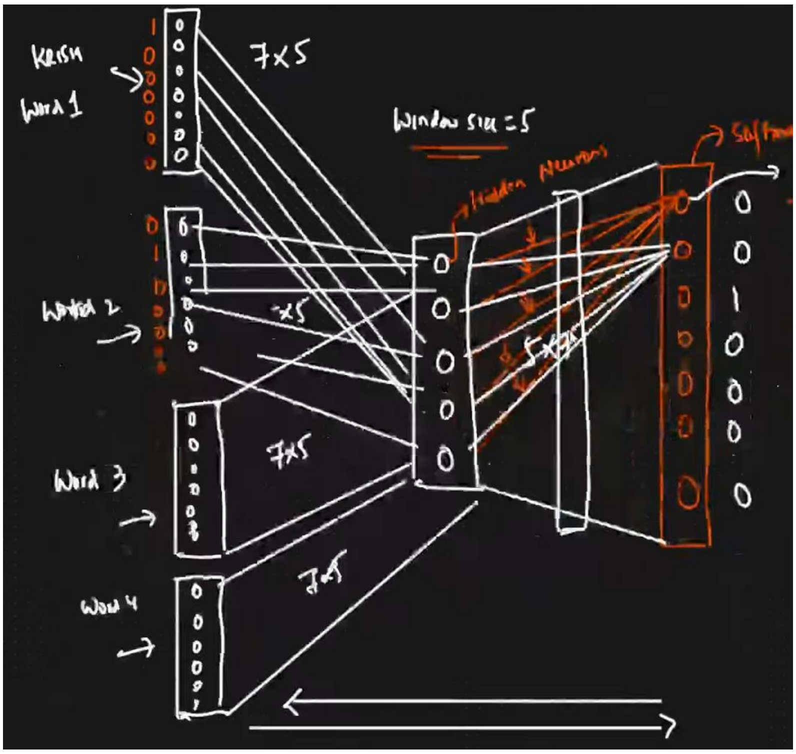
Now I will **slide my window** by one step. So next centre word or target is **Related.** Similarly, we will get the independent feature and target.



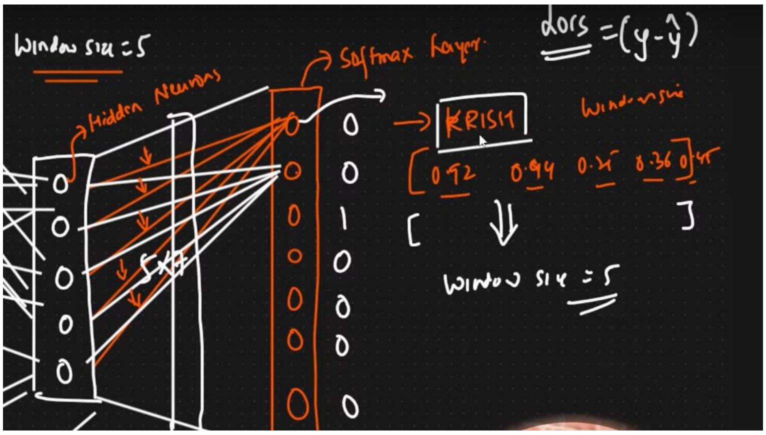
Now I can't slide because I won't get 5 words. In real google take entire dictionary 13 million words to do this with millions of words.

Now we also need to represent the words as One Hot Encoding like OHE. Then both OHE and our feature data will be fed to Deep Learning Neural network i.e. fully connected layer to learn each word representation as vector of feature.

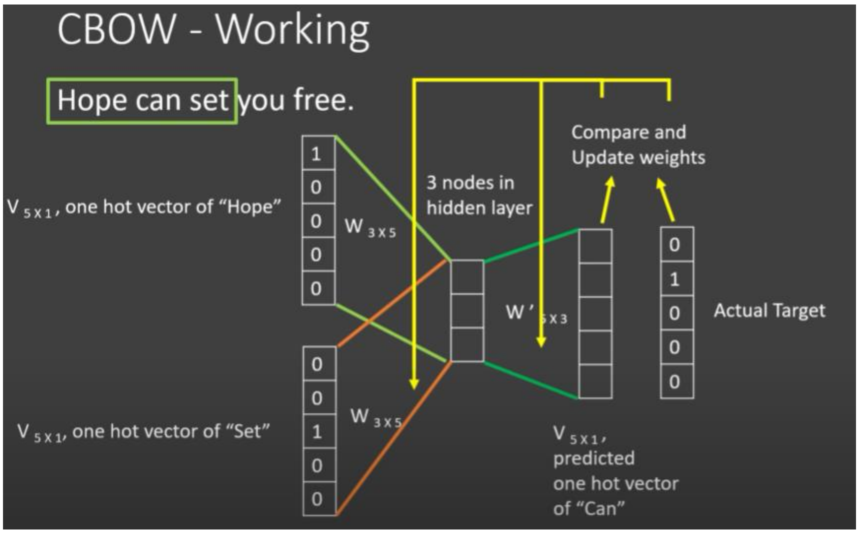
Here input size is 4 words due to 5 window size and each vector is represented by 7 vector using OHE.



- In first sample 4 words like Krish, Channel, Related, To each words will be represented as 7 vector as input feature i.e. 7*5 matrices in ANN.
- **Window size is 5 so hidden layer will be having 5 neurons** with 5*7 dimension.
- Out put layer is having 7 nodes as IS is having 7 vectors in OHE. Here we will use **SoftMax** layer.
- In the out put we have 7 vectors so the weightage of hidden neuron connection will be the actual vector representation of each words of vocabulary. Like first index in output layer is Krish in our vocabulary which is currently 0. Now after training it will have weightage like the how all hidden neurons are connected to first output index.

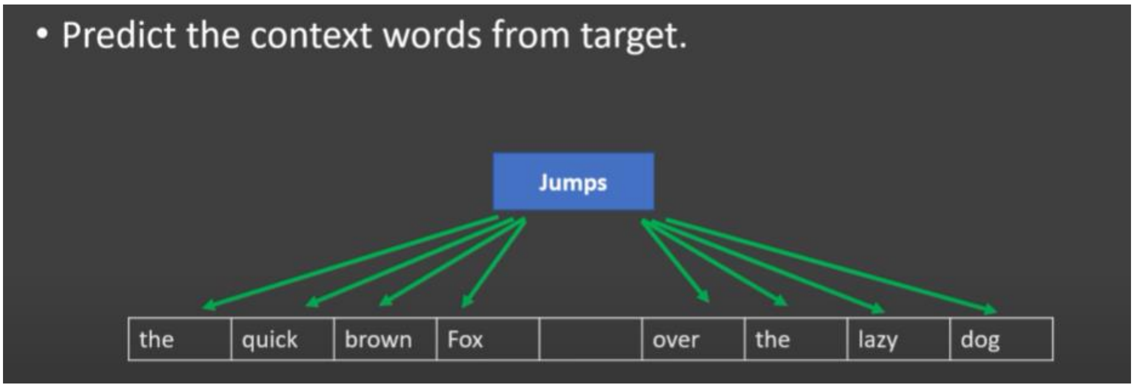


- Thus, we are getting semantic meaning and as those weightages are not zero hence it is not a sparse matrix.
- We can increase the hidden layer thus we can do hyper parameter tuning.

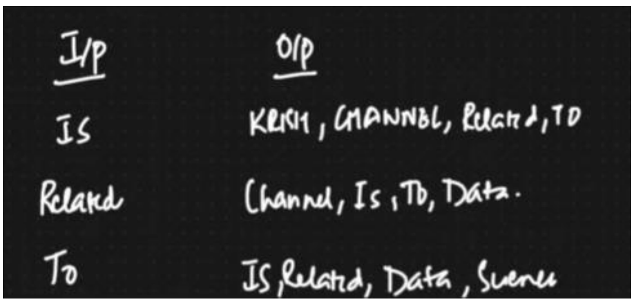


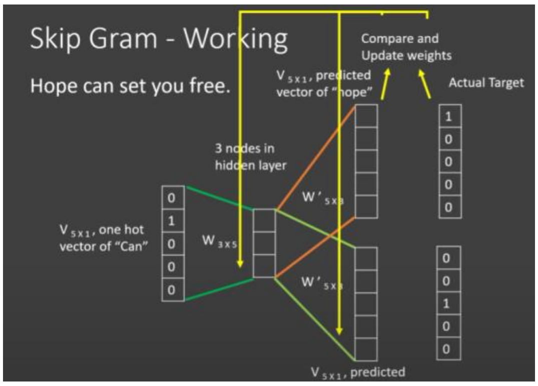
Skip Gram

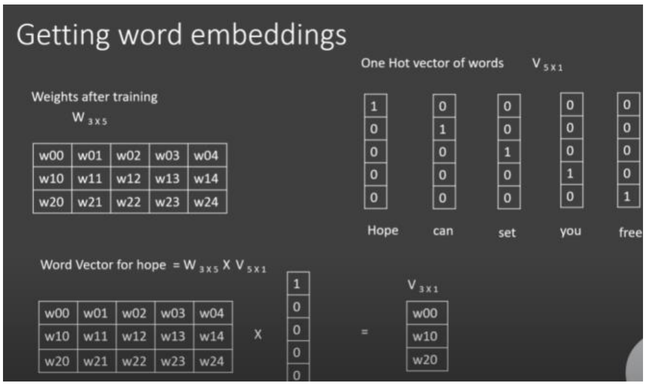
Here we are doing opposite like predicting the context words from target word.



- Consider window size as same as 5.
- Here input will be the centre word and output will be the context words.
- Just reverse the ANN and we will get Skip Gram
- If google is creating a word to vec as 300 dimensions like every word will have 300 values related to other feature word then window size is 300.







- For large data use Skip Gram other wise use CBOW

More window size means very good model more semantic information can be captured.

Gensim is a Python library for natural language processing, offering tools for document similarity, topic modeling, and word embedding. It enables efficient analysis of large text datasets for extracting meaningful patterns and insights.

```python
from gensim.models import Word2Vec, KeyedVectors
```
✓ 0.0s Python

We will be using pre trained model made using googole news information more 3 billion data by Google for word embedding

```python
import gensim.downloader as api
wv = api.load('word2vec-google-news-300') # this eill be of size in 1662MB
```
✓ 6m 3.3s Python

```
[==================================================] 100.0% 1662.8/1662.8MB downloaded
```

```python
wv.most_similar("king")
```
✓ 3.2s

```
[('kings', 0.7138045430183411),
 ('queen', 0.6510956883430481),
 ('monarch', 0.6413194537162781),
 ('crown_prince', 0.6204220056533813),
 ('prince', 0.6159993410110474),
 ('sultan', 0.5864824056625366),
 ('ruler', 0.5797567367553711),
 ('princes', 0.5646552443504333),
 ('Prince_Paras', 0.5432944297790527),
 ('throne', 0.5422105193138123)]
```

```python
wv.most_similar("man")
```
✓ 3m 41.2s

```
[('woman', 0.7664012908935547),
 ('boy', 0.6824871301651001),
 ('teenager', 0.6586930155754089),
 ('teenage_girl', 0.6147903203964233),
 ('girl', 0.5921714305877686),
 ('suspected_purse_snatcher', 0.571636438369751),
 ('robber', 0.5585119128227234),
 ('Robbery_suspect', 0.5584409832954407),
 ('teen_ager', 0.5549196600914001),
 ('men', 0.5489763021469116)]
```

```python
wv.similarity("boy","man")
```
✓ 0.0s

```
0.68248713
```

```python
vector= wv["king"]-wv["man"]+wv["woman"]
```
✓ 0.0s

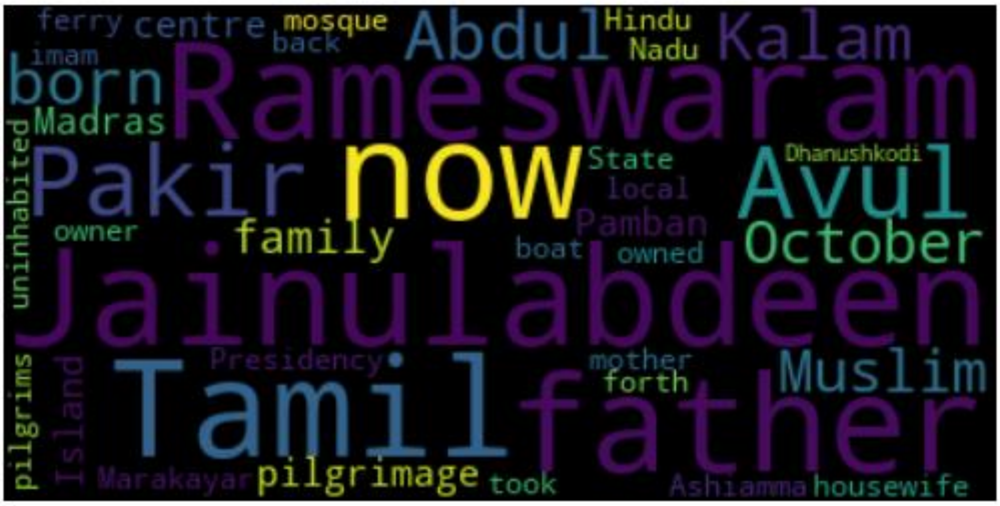
```python
wv.most_similar([vector])
```
✓ 0.3s

```
[('king', 0.8449392318725586),
 ('queen', 0.7300517559051514),
 ('monarch', 0.645466148853302),
 ('princess', 0.6156251430511475),
 ('crown_prince', 0.5818676352500916),
 ('prince', 0.5777117609977722),
 ('kings', 0.5613663792610168),
 ('sultan', 0.5376775860786438),
 ('Queen_Consort', 0.5344247817993164),
 ('queens', 0.5289887189865112)]
```

Word to Cloud

If I convert those 300 dimensions to 2D then it will look as word to Cloud.

wordcloud is a Python library that creates visual representations of word frequency in a text corpus, generating "word clouds" where the size of each word reflects its frequency or importance in the given text.


ferry
centre
mosque
Abdul
Hindu
Kalam
imam
back
Nadu
born
Rameswaram
Madras
Pakir
now
State
Dhanushkodi
Avul
local
Pamban
owner
family
boat
owned
October
uninhabited
Jainulabdeen
Presidency
mother
Muslim
forth
Tamil
father
pilgrims
Island
Marakayar
pilgrimage
took
Ashiamma
housewife