Natural Language Processing NLP

- Tockenization in NLP

Topics:

1) **Corpus**: Paragraph

2) **Documents**: Sentences

3) **Vocablary**: Unique words in the paragraph(corpus)

4) **Words**: All words in paragraph(corpus)

"My name is Amir and I am currently learning ML, NLP and DL. I am also a web developer.

↓

Corpus

- Tockenization is a process in which we convert text to tokens i.e (corpus to documents)

So After performing tockenization

Tockens (Documents) (sentences):

① My name is Amir and I am currently learning ML, NLP and DL.

② I am also a web developer

↓ **Tockenization (will convert these documents into words)**

e.g

corpus ["I like to drink apple juice. My friend like Mango Juice"

⇓ **Tokenization**

Tockens (Documents) (sentences)

① I like to drink apple juice.

② My friend like Mango Juice.

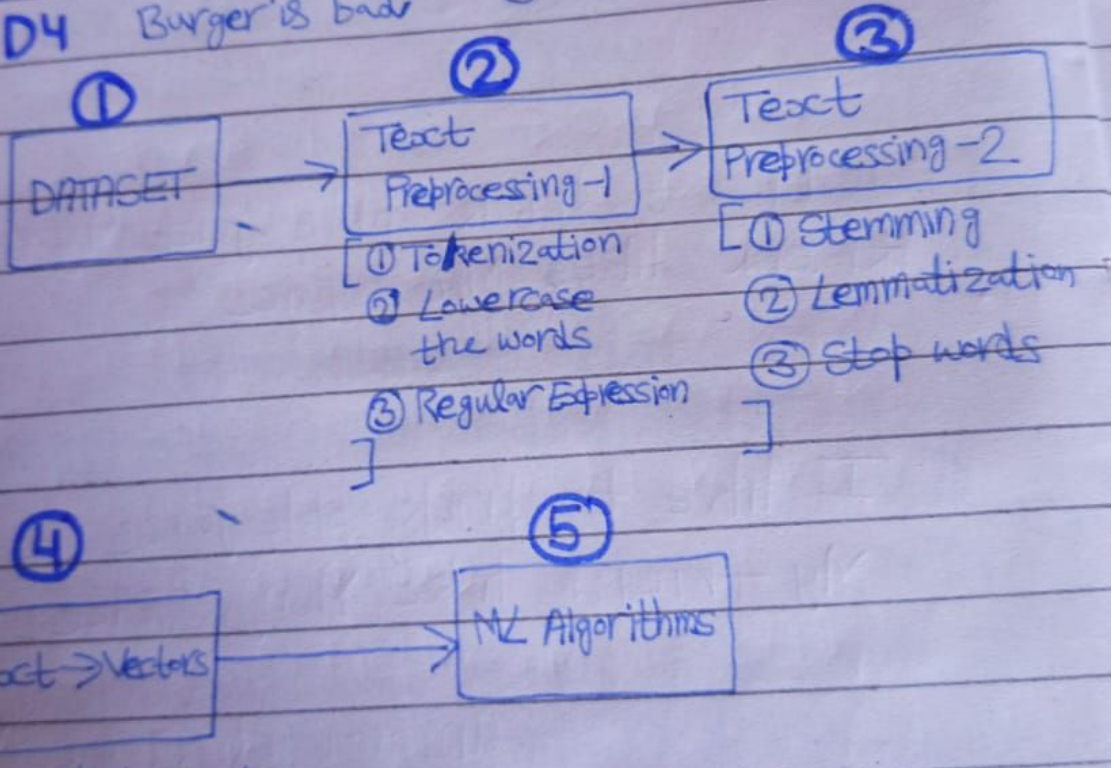
⇓

Vocabulary = Unique words (Total: 9)

Text Preprocessing → What we learnt

⇒ Sentiment Analysis:

	Text	O/P
D1	The food is good	1
D2	The food is bad	0
D3	Pizza is amazing	1
D4	Burger is bad	0



① One hot Encoding
② Bag of Words (BOW)
③ TF-IDF
④ Word2vec
⑤ Avg Word2vec

① One Hot Encoding:

	Text	O/P
D1	The food is good	1
D2	The food is bad	0
D3	Pizza is Amazing	1

Vocabulary (unique words) (size: 7)

The food is good bad Pizza Amazing

(4 words × 7 → Vocabulary size)

D1 [

[1 0 0 0 0 0 0],
[0 1 0 0 0 0 0],
[0 0 1 0 0 0 0],
[0 0 0 1 0 0 0]

]

D2 [

[1 0 0 0 0 0 0],
[0 1 0 0 0 0 0],
[0 0 1 0 0 0 0],
[0 0 0 0 1 0 0],

]

→ Representation of The in one hot Encoding only vocabulary of The is 1 all other 0

Advantages and Disadvantages of One Hot Encoding:

Advantages

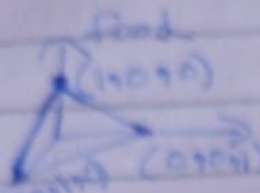
Easy to implement with python [SKlearn onehot Encoder]

Disadvantages:

1. Sparse Matrix → overfitting
2. ML Algorith → Fixed size input (If you encode "Pizza is amazing" the matrix will be 3x7 but in other sentences 4x7 so input is not fixed size which is required by ML Algorithm)
3. No semantic Meaning is getting captured

food	Pizza	burger
[1	0	0]
[0	1	0]
[0	0	1]

These are my vectors



You can see all these words are at same distance means no differentiation or semantic meaning

④ Out of Vocabulary (OOV)

Let's say our new test text is:

"Burger is bad"

but Burger is not in our Vocabulary so we can't encode it.

② Bag of Words (BoW):

Dataset Text	O/P
He is a good boy	1
She is a good girl	1
Boy and girl are good	1

⇓ Lower all words and apply stop words (He, is, a, she will go in stopwords)

S1 → good boy
S2 → good girl
S3 → boy girl good.

Vocabulary	Frequency		[good	boy	girl]	O/P
good	3	S1	[1	1	0]	1
boy	2	S2	[1	0	1]	1
girl	2	S3	[1	1	1]	1

- We sort words in vocabulary in descendin order w.r.t frequency.

Binary BoW vs BoW

{1 and 0}
[if in a sentence a word is presented 2 times still 1]

[count will get updated based on frequency of word in sentence i.e 2 or 3 etc]

Advantages of BoW (Bag of Words):

1. Simple and Intuitive
2. Fixed size Inputs → ML Algorithms (because for every sentence input will be equal to size of vocabulary)

Disadvantages of BoW (Bag of Words):

1. Sparse Matrix/ array → overfitting
2. Ordering of word is getting changed (due to sorting of vocabulary)
3. Out of Vocabulary (OOV) (word not in vocabulary get ignored)
4. Semantic Meaning is still not captured. (As there is only 1 and 0 so the importance of word is not captured)

e.g

The food is good → [1 1 1 1 0 0] → V_1

The Food is not good → [1 1 1 1 1 0] → V_2

These are looking almost similar as only one word change but they are opposite so lag this clarity.

N-grams: e.g bigrams, trigrams

S1 → The food is good

S2 → The food is not good

Let our vocabulary is:

	food	not	good
S1 →	1	0	1
S2 →	1	1	1

Now we can see that these two vectors are almost similar just one change there but actually these sentences are totally opposite

We solve this problem using

N-grams

Bigram (we add new words in vocabulary by combining 2 words)

Vocabulary(Bigram)

	food	not	good	foodgood	foodnot	notgood
S1	1	0	1	1	0	0
S2	1	1	1	0	1	1

Now you can see there is huge difference b/w vectors so model can clearly distinguish as we capture more contextual information.

Sklearn → n-grams ⇒

(1,1) → unigrams

(1,2) → unigram, bigram

(1,3) → unigram, trigram (combine 3 words)

(2,3) → Bigram & trigram

③ TF-IDF (Term Frequency - Inverse Document Frequency)

S1 → good boy
S2 → good girl
S3 → boy girl good

$$\text{Term Frequency (TF)} = \frac{\text{No. of repitition of words in sentence}}{\text{No. of words in sentence}}$$

$$\text{IDF} = \log_e\left(\frac{\text{No. of sentences}}{\text{No. of sentences containing the word}}\right)$$

Term Frequency

	S1	S2	S3
good	1/2	1/2	1/3
boy	1/2	0/2=0	1/3
girl	0/2=0	1/2	1/3

IDF:

Words	IDF
good	$\log_e\left(\frac{3}{3}\right)=0$
boy	$\log_e\left(\frac{3}{2}\right)$
girl	$\log_e\left(\frac{3}{2}\right)$

Final TF-IDF (TF x IDF)

	good	boy	girl
S1	$1/2 \times 0 = 0$	$1/2 \times \log_e\left(\frac{3}{2}\right)$	$0 \times \log_e\left(\frac{3}{2}\right)=0$
S2	$1/2 \times 0 = 0$	$0 \times \log_e\left(\frac{3}{2}\right)=0$	$\frac{1}{2} \times \log_e\left(\frac{3}{2}\right)$
S3	$1/3 \times 0 = 0$	$1/3 \times \log_e\left(\frac{3}{2}\right)$	$1/3 \times \log_e\left(\frac{3}{2}\right)$

• Advantages of TF-IDF:

1. Intuitive
2. Fixed size inputs (depends on vocabulary size)
3. Word Importance is getting captured

e.g good is present in all sentence so TF-IDF is 0 not important but in BoW it will be 1

Disadvantages of TF-IDF:

① Sparse Matrix still exist → overfitting

② out of Vocabulary (OOV)

★ TF-IDF is better than BoW

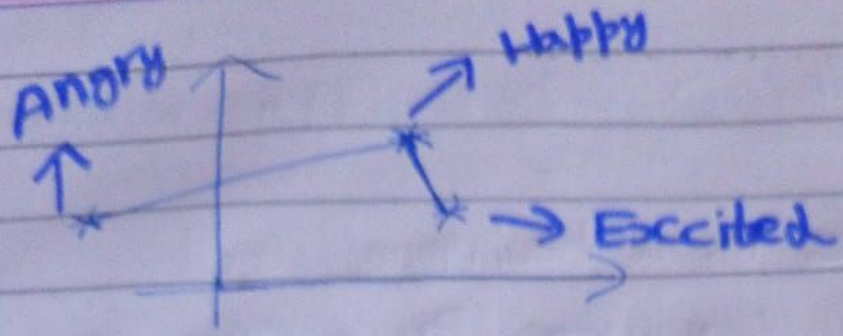
Word Embeddings:

"In Natural Language Processing (NLP), word embedding is a term used for representation of words for text analysis, typically in the form of a real-valued vector that encodes the meaning of word such that the words that are closer in vector space are expected to be similar in meaning."

Angry Happy Excited

[ ] [ ] [ ]

Using PCA we can represent them in two-dimensions.



As Happy and Excited are close they are considered similar in meaning where angry is at distance as it is opposite.

Word Embeddings

Count/Frequency

1. One Hot Encoding (OHE)
2. Bag of Words (BOW)
3. TF-IDF

ANN

Deep learning trained Models

Word2vec

CBOW

[Continuous Bag of words]

Skipgram

Word2Vec ⇒ Feature Representation

Word2Vec is a technique used for Natural Language Processing published in 2013 by Google. The Word2Vec Algorithm uses a neural network model to learn word associations from a large corpus of text. Once trained such a model can detect synonymous words or suggest additional words for a partial sentence. As the name implies, Word2Vec represents each distinct word with a particular list of numbers called a vector.

⇓⇓

It is trained on
300 features
and almost 3 billion
vocabulary words

Vocabulary → unique words in corpus

Vocabulary

Feature representation

	Boy	Girl	KING	QUEEN	Apple	Mango
Gender	-1	1	-0.92	0.93	0.01	0.05
Royal	0.01	0.02	0.95	0.96	-0.02	0.02
Age	0.03	0.02	0.75	0.68	0.95	0.96
Food	—	—	—	—	0.91	0.92
.	—	—	—	—	—	—
.	—	—	—	—	—	—
.	—	—	—	—	—	—
nth	—	—	—	—	—	—

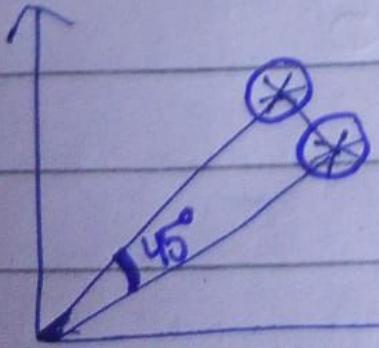
300 dimensions/ features

→ Vector representing word boy

[KING Boy + QUEEN = GIRL]

Relationship calculation using word2vec

Cosine Similarity



Distance = 1 − cosine similarity

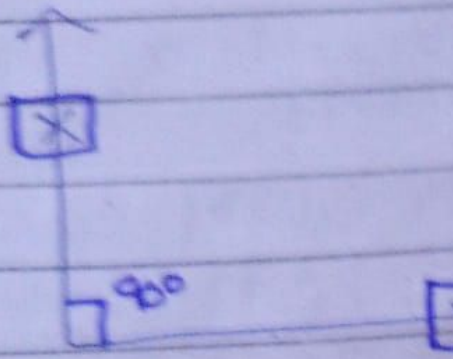
Cosine Similarity = cos 45 = $\frac{1}{\sqrt{2}}$

= 0.7071

Distance = 1 − 0.7071

Distance = 0.29

So these 2 words are almost similar



Distance = 1 − cosine similarity

cosine similarity = cos(90°) = 0

Distance = 1 − 0

Distance = 1

As distance is greater means these 2 words are completely different.

* Word2Vec → CBOW (Continuous Bag of Words)
* Word2Vec → Skipgram

~~Pretrained~~ Model

Train a model from scratch

1) CBOW (Continuous Bag of Words):

Let we have corpus

[iNeuron Company Is Related To DATA SCIENCE] → Vocabulary

let

Window Size = 5

Take odd number

* Actually as window size is 5 we are taking first 5 words marking the center word as output and remaining four as input then sliding window by 1 and repeat.

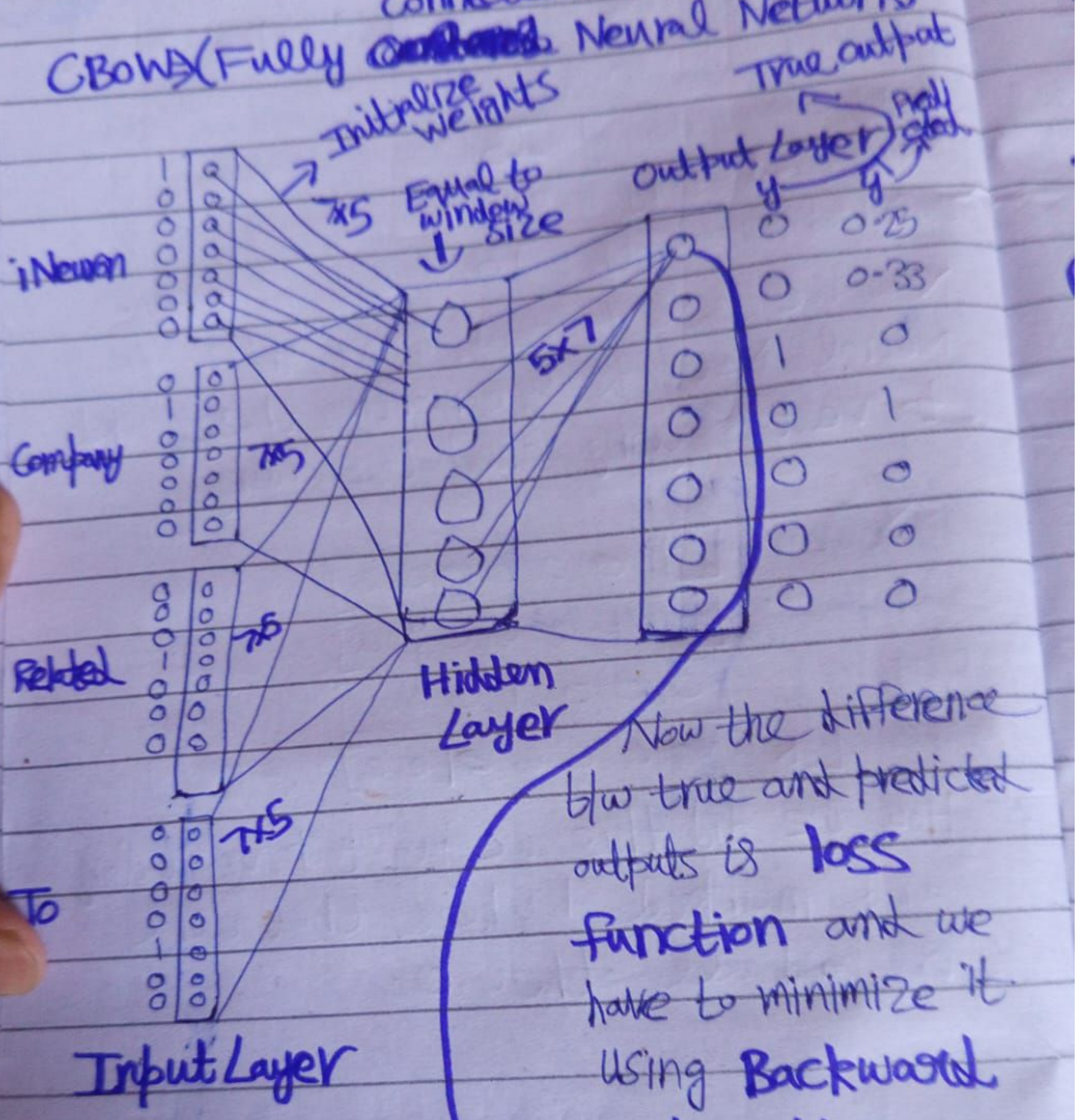
Input	Output
→ [iNeuron, Company, Related, To]	IS
→ [Company, IS, To, Data]	Related
→ [IS, Related, DATA, SCIENCE]	To

Now we have to send these inputs to Neural Network by converting these words to vectors using one hot Encoding:

iNeuron	[1 0 0 0 0 0 0]
Company	[0 1 0 0 0 0 0]
Related	[0 0 0 1 0 0 0]
To	[0 0 0 0 1 0 0]

* How we did this we just took vocabulary and marked 1 where that word is and all other 0.

CBOW (Fully Connected Neural Network)



Now the difference b/w true and predicted outputs is **loss function** and we have to minimize it using **Backward propogation.**

Company [0.92, 0.94, 0.25, 0.36, 0.45]

- The vector representation of word

will be of same size as window size because window size is actually no. of features

- Like this we will get vectors of size 5 for our all 7 vocabulary words.

(2) Skipgram:

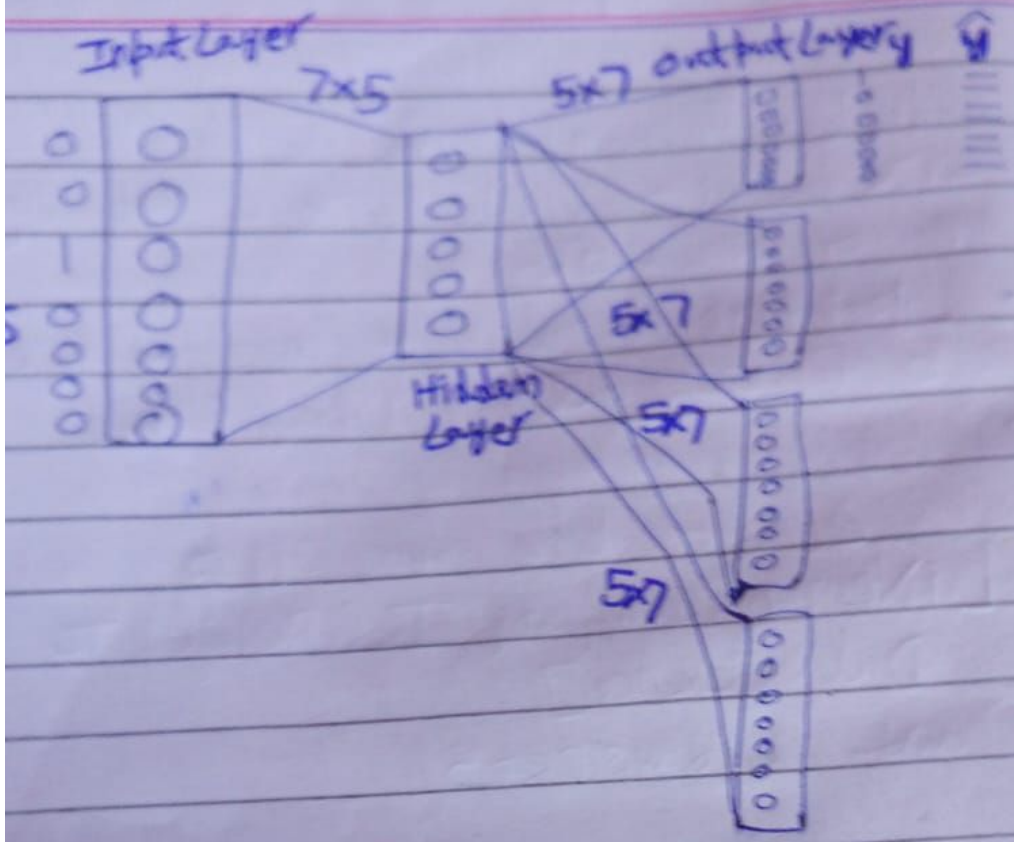
Let we take same corpus

[iNeuron Company Is Related To Data Science]

and **windowsize=5**

Output	Input
→ [iNeuron, Company, Related, To]	IS
→ [Company, Is, To, Data]	Related
→ [Is, Related, DATA, SCIENCE]	TO

* Everything remained same as CBOW we just swapped inputs and outputs.



When should we apply CBOW or Skipgram?

- ~~Small Dataset~~ → ~~CBOW~~
- Huge Dataset → Skipgram

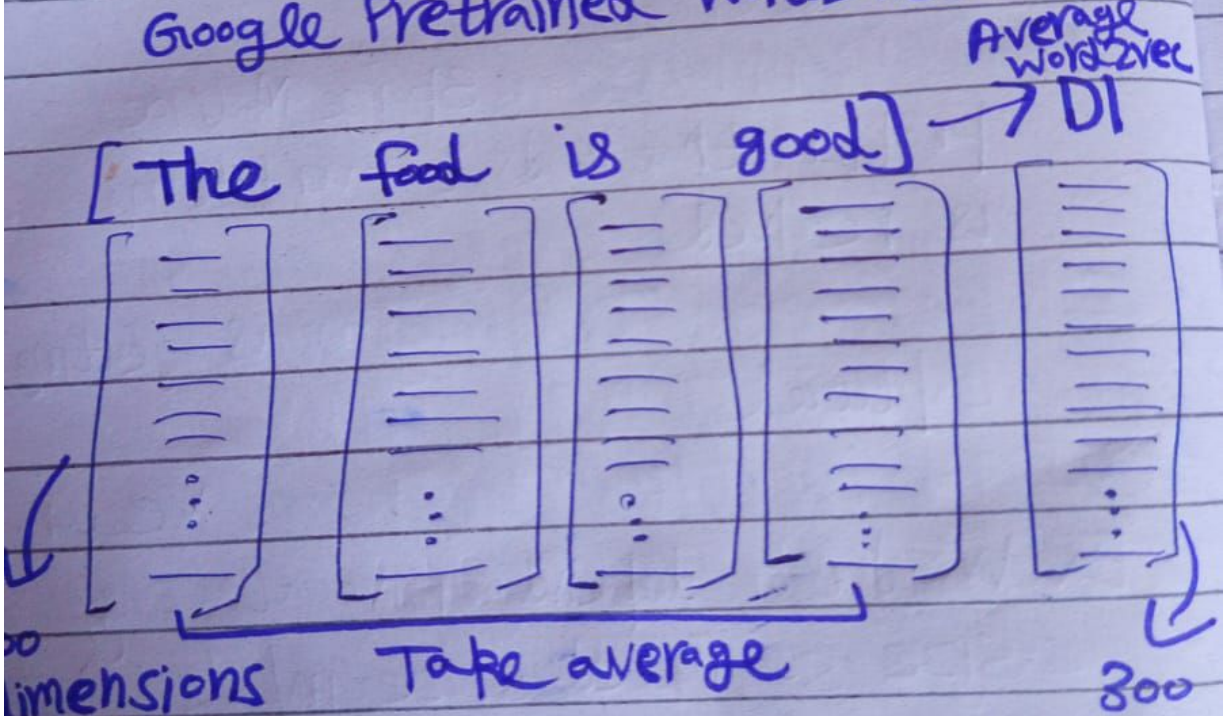
Improve CBOW or Skipgram

1) Increase the training Data
2) Increase the window size (no. of features) which leads to increase of vector dimensions.

⇒ Avg Word2Vec:

	Text	Output	Avg word2vec	output
1	The food is good	1	[. . . . .]	1
2	The food is bad	0	[ ]	—
3	Pizza is Amazing	1	[ ]	—

Let's say we are using
Google Pretrained Word2vec model



* Now the problem is that we have seperate 300 dimension vector for

Google Word2Vec: → Gensim Library

- It is trained on 3 billion words and 300 window size/features
- So it will give feature representation of 300 dimension vectors.

⇒ Advantages of word2vec:

① Dense Matrix (Sparse Matrix problem that leads to overfitting is resolved)

② Semantic Information is getting captured

③ Fixed set of dimensions vector. We don't depend on vocabulary size so fixed size inputs.

④ OOV (out of vocabulary) issue also resolved.

every single word but we want a single 300 dimension vector for whole sentence DI so we take average of the vectors of all words. This is called **AvgWord2Vec.**

Library

Gensim

- Pretrained ~~Google Word2vec~~
- Train Word2vec from scratch