NLP Basics :-

Count Based Methods → One hot Encoding vectors
→ Countvectorizer

Corpus :- (1) The food is awesome → ['food', 'awesome']

(2) Ambience is good, food is good → ['ambience', 'good', 'food']

(3) Ambience is good, but food is bad → ['ambience', 'good', 'food', 'bad']

Vocabulary = V = {'food', 'awesome', 'ambience', 'good', 'bad',}

One hot Encoding

	'food'	'awesome'	'ambience'	'good'	'bad'
(1)	1	1	0	0	0
(2)	1	0	1	1	0
(3)	1	0	1	1	1

Count Vectorizer

Vector are also of the size of vocabulary.

$$\vec{d} = \left[C_1^{(d)}, C_2^{(d)}, C_3^{(d)}, \ldots\ldots, C_n^{(d)} \right]$$

$$C_i^{(d)} = \text{Count}(w_i, d) \qquad w_i \in V$$

With bravian example the count vector will looke like :-

	food	awesome	ambience	good	bad
(1)	1	1	0	0	0
(2)	1	0	1	(2)	0
(3)	1	0	1	1	1

tf-idf vectorizer:-

tf → term frequency , idf → inverse document frequency

document from:- w_i → i^{th} term

n_i → number of documents it has occured.

$df_i = \frac{n_i}{n}$ n → total no. of documents.

$$idf_i = \log\left(\frac{n}{n_i}\right)$$

$$tf\text{-}idf(w_i, d_j) = tf(w_i, d_j) \times \log\left(\frac{n}{n_i}\right)$$

term freq:- $tf(w_i, d_j) = Count(w_i, d_j)$

Word2Vec

In word2vec the words are represented by dense vectors of similar dimension ($d = 300$)

- CBOW → Continuous Bag of words
- Skip-gram method.

CBOW

(1) Obtain one-hot encoded vector of each word in the vocabulary.

$V = \{$ 'food', 'good', 'ambience', 'awesome', 'bad', 'wine' $\}$

$$OHE(w_i) = [\underset{1}{0}, \underset{2}{0}, \underset{3}{0}, \ldots, \underset{i}{1}, 0, 0 \ldots\ldots 0]^T_{|V| \times 1}$$

$OHE(food) = [1,0,0,0,0,0]^T$, $OHE(ambience) = [0,0,1,0,0,0]^T$

$OHE(bad) = [0,0,0,0,1,0]$

(2) Obtain Context Matrix with a specified <u>context window</u> (k)

$C \rightarrow$ context matrix, $C_{|V| \times |V|}$, $C_{ij} \in C \rightarrow$ How many times word W_i has appeared in the context of W_j

k=2

Corpus = ["I like NLP, but I hate python"
"I like python, but I love CV"]

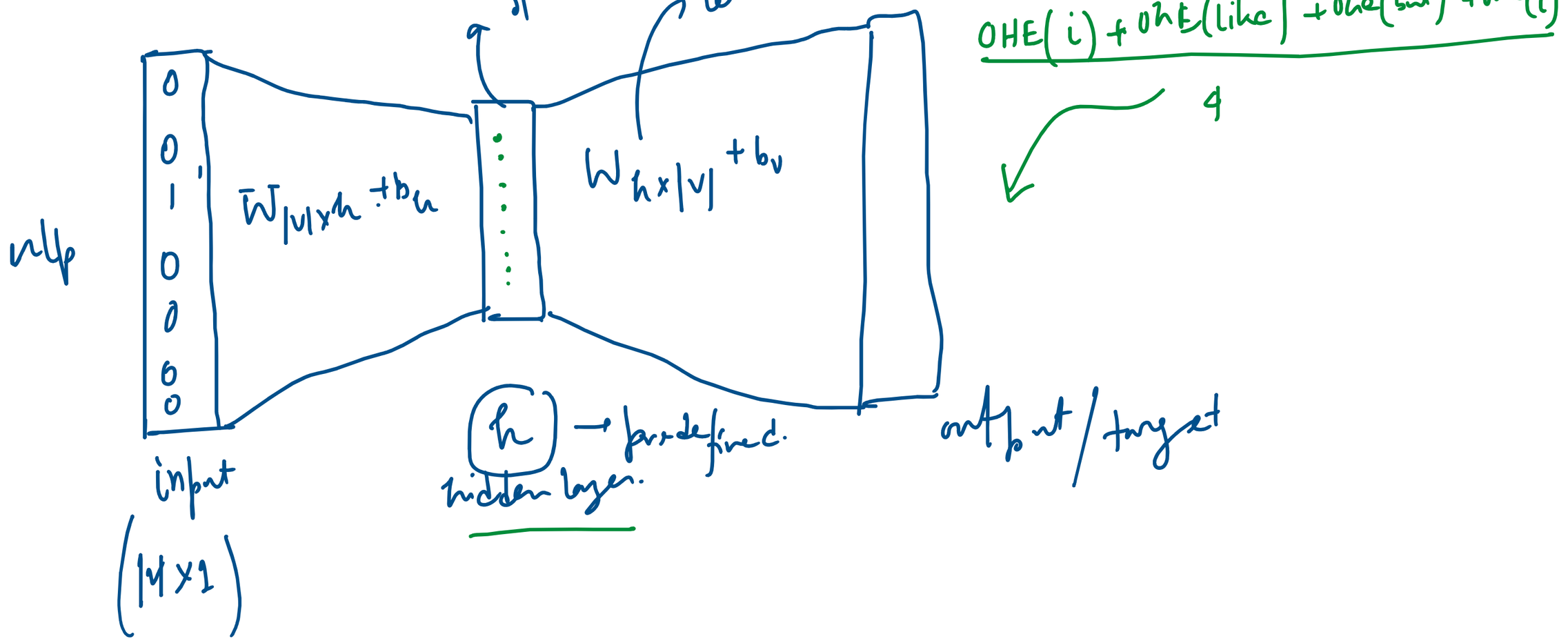
$V = \{$ 'I', 'like', 'nlp', 'hate', 'pytnon', 'love', 'CV' $\}$

k=2

	'i'	'like'	'nlp'	'hate'	'pytnon'	'love'	'CV'
i	0	2	1	1	2	1	1
like	2	0	1	0	1	0	0
nlp							
hate							
pytnn	2	1	0	1	0	0	0
love							
CV							

(3) Train a shallow neural net

input is the OHE(w_i) & output/target average of the context of w_i



Advantages of dense representation of word vectors over sparse representation

1) It takes less space to store dense representation.
2) It preserves semantic meaning.

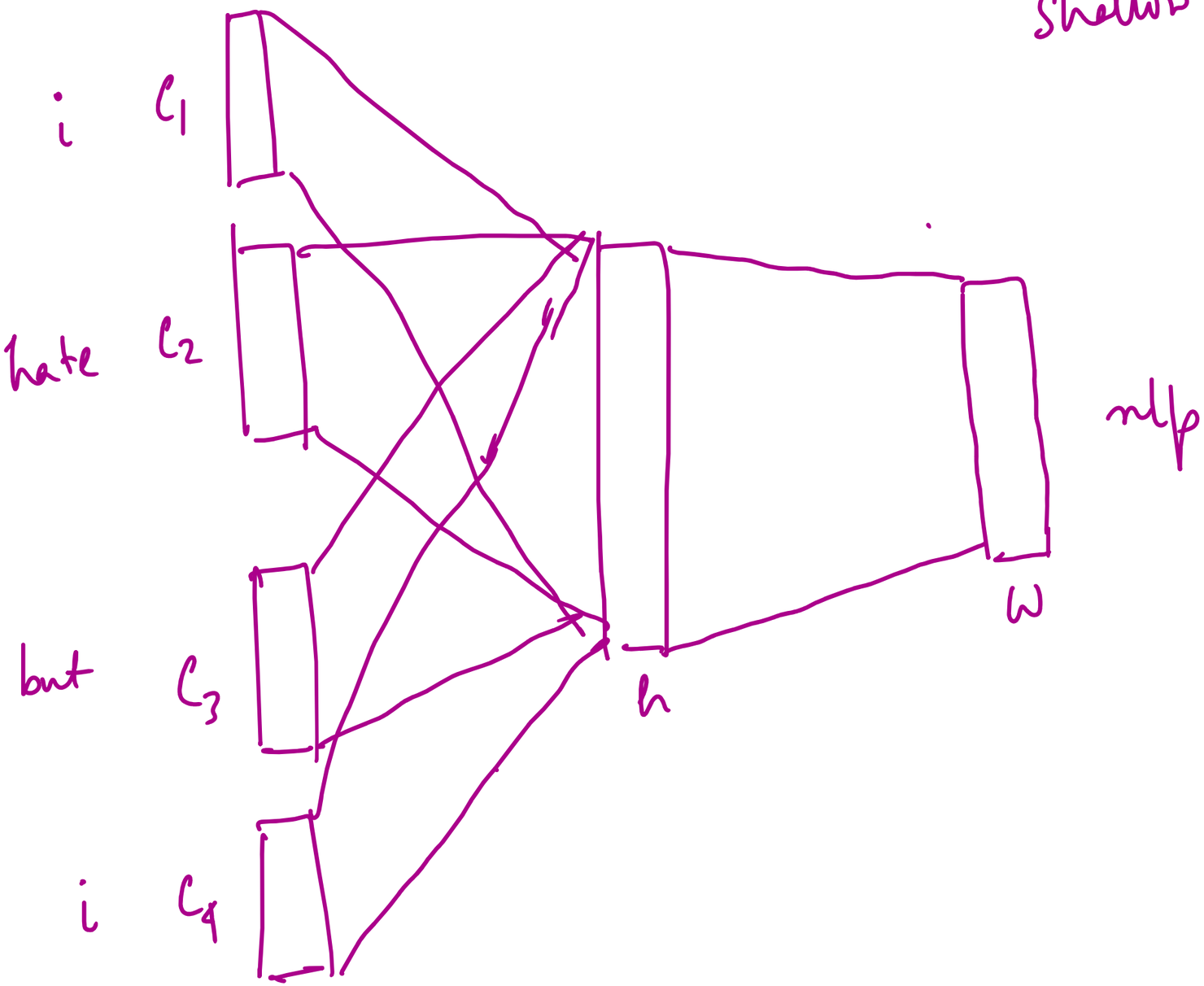
USA ⟶ Washington DC

India ⟶ ? (Delhi)

$$V(\text{USA}) - V(\text{Washington DC}) \approx V(\text{India}) - V(\text{delhi})$$

Skip-gram model :- k=2

Shallow dense neural network.



Pretrained Word2Vec

→ Google has trained Word2Vec models with wikipedia data.

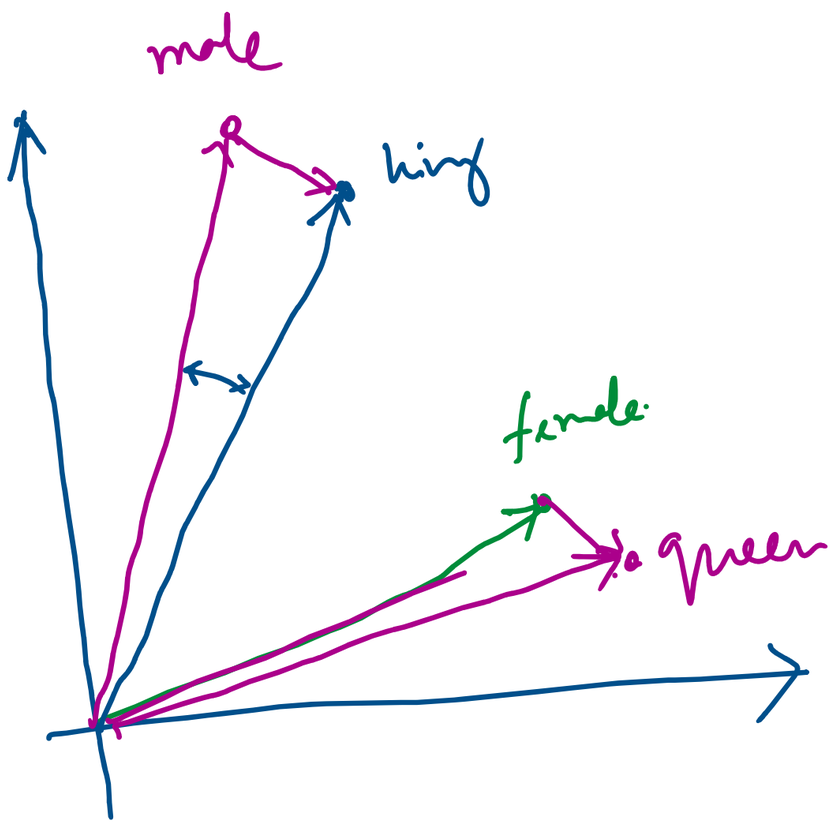
→ Using that model it has generated word-vectors for all words in english.

→ It has also word-vector repository for other languages.

→ These pretrained word vectors can be used directly for other NLP tasks such as: Chatbot, Sentiment analysis etc.

→ These pretrained models are available to download for free.

We can also create our own <u>domain aware</u> word2vec using python library 'gensim'.



king, queen.

cosine sim (king, male) is higher

compared to cosinesim(male, female)

$\overrightarrow{male} - \overrightarrow{king} \approx \overrightarrow{female} - \overrightarrow{queen}$

male → female

king → queen.

deer, bison, chair