NLP

Lec2 (word embedding + Word2vec)

- **Word Embeddings** are numeric representations of words in a lower-dimensional space, capturing semantic and syntactic information. They play a vital role in **Natural Language Processing (NLP) tasks**

What is Word Embedding in NLP?

It is an approach for representing words and documents. Word Embedding or Word Vector is a numeric vector input that represents a word in a lower-dimensional space. It allows words with similar meanings to have a similar representation.

- Word Embeddings are a method of extracting features out of text so that we can input those features into a machine learning model to work with text data.

- They try to preserve syntactical and semantic information such as Bag of Words (BOW), CountVectorizer and TF-IDF rely on the word count in a sentence but do not save any syntactical or semantic information.

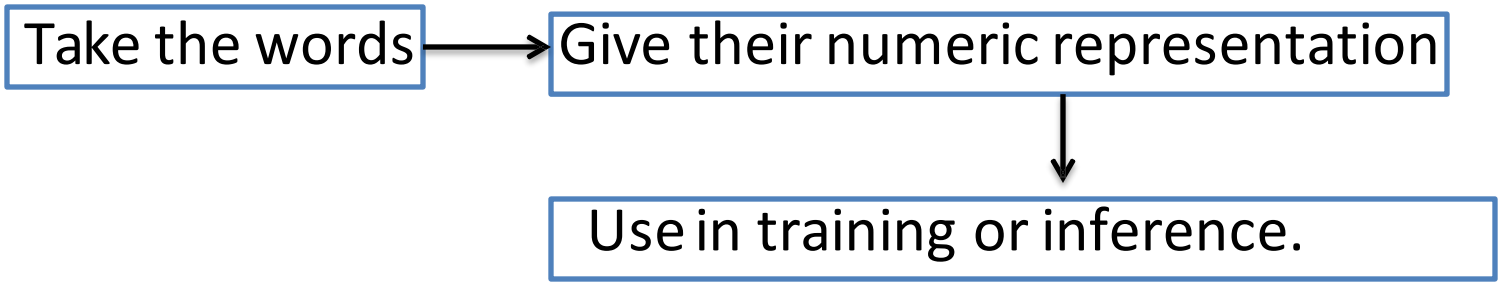
- In these algorithms, the size of the vector is the number of elements in the vocabulary. We can get a sparse matrix if most of the elements are zero. Large input vectors will mean a huge number of weights which will result in high computation required for training.
- Word Embeddings give a solution to these problems.

Need for Word Embedding?

- To reduce dimensionality
- To use a word to predict the words around it.
- Inter-word semantics must be captured.

How are Word Embeddings used?

- They are used as input to machine learning models.



Approaches for Text Representation

- **Traditional Approach**
- **Neural Approach**

Traditional Approach

Traditional Approach

One-Hot Encoding

- It is a simple method for representing words in natural language processing (NLP).
- In this encoding scheme, each word in the vocabulary is represented as a unique vector, where the dimensionality of the vector is equal to the size of the vocabulary.
- The vector has all elements set to 0, except for the element corresponding to the index of the word in the vocabulary, which is set to 1.

One Hot Encoding

Vocabulary:{'mat', 'the', 'bird', 'hat', 'on', 'in', 'cat', 'tree', 'dog'}
Word to Index Mapping:
{'mat': 0, 'the': 1, 'bird': 2, 'hat': 3, 'on': 4, 'in': 5, 'cat': 6, 'tree': 7, 'dog': 8}

One-Hot Encoded Matrix:

cat: [0, 0, 0, 0, 0, 0, 1, 0, 0]
in: [0, 0, 0, 0, 0, 1, 0, 0, 0]
the: [0, 1, 0, 0, 0, 0, 0, 0, 0]
hat: [0, 0, 0, 1, 0, 0, 0, 0, 0]
dog: [0, 0, 0, 0, 0, 0, 0, 0, 1]
on: [0, 0, 0, 0, 1, 0, 0, 0, 0]
the: [0, 1, 0, 0, 0, 0, 0, 0, 0]
mat: [1, 0, 0, 0, 0, 0, 0, 0, 0]
bird: [0, 0, 1, 0, 0, 0, 0, 0, 0]
in: [0, 0, 0, 0, 0, 1, 0, 0, 0]
the: [0, 1, 0, 0, 0, 0, 0, 0, 0]
tree: [0, 0, 0, 0, 0, 0, 0, 1, 0]

Disadvantages of one hot encoding

- The results in high-dimensional vectors, making it computationally expensive and memory-intensive, especially with large vocabularies.
- It does not capture semantic relationships between words; each word is treated as an isolated entity without considering its meaning or context.
- It is restricted to the vocabulary seen during training, making it unsuitable for handling out-of-vocabulary words.

Bag of Word (Bow)

It is a text representation technique that represents a document as an unordered set of words and their respective frequencies. It discards the word order and captures the frequency of each word in the document, creating a vector representation.

Bag-of-Words Matrix:

[[0 1 1 1 0 0 1 0 1]
[0 2 0 1 0 1 1 0 1]
[1 0 0 1 1 0 1 1 1]
[0 1 1 1 0 0 1 0 1]]

Vocabulary (Feature Names): ['and' 'document' 'first' 'is' 'one' 'second' 'the' 'third' 'this']

Disadvantages of Bag of Word

- BoW ignores the order of words in the document, leading to a loss of sequential information and context making it less effective for tasks where word order is crucial, such as in natural language understanding.
- BoW representations are often sparse, with many elements being zero resulting in increased memory requirements and computational inefficiency, especially when dealing with large datasets.

Advantages and Disadvantage of Word Embeddings

Advantages

- It is much faster to train than hand build models like WordNet (which uses ***graph embeddings***).
- Almost all modern NLP applications start with an embedding layer.
- It Stores an approximation of meaning.

Disadvantages

- It can be memory intensive.
- It is corpus dependent. Any underlying bias will have an effect on your model.
- It cannot distinguish between homophones. Eg: brake/break, cell/sell, weather/whether etc.

Neural Approach

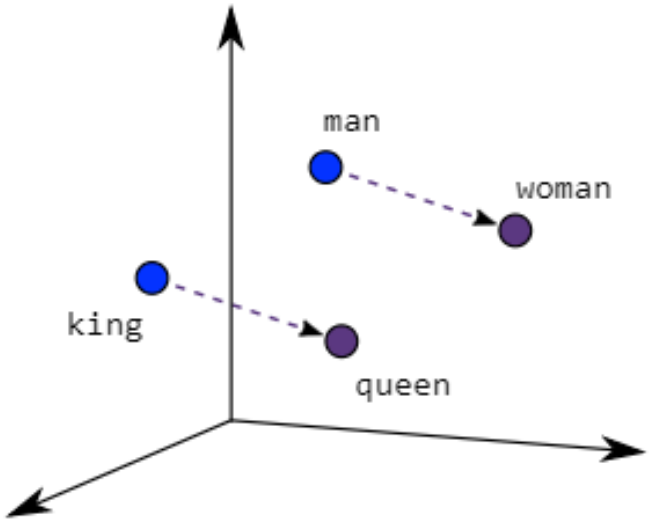
Word2vec

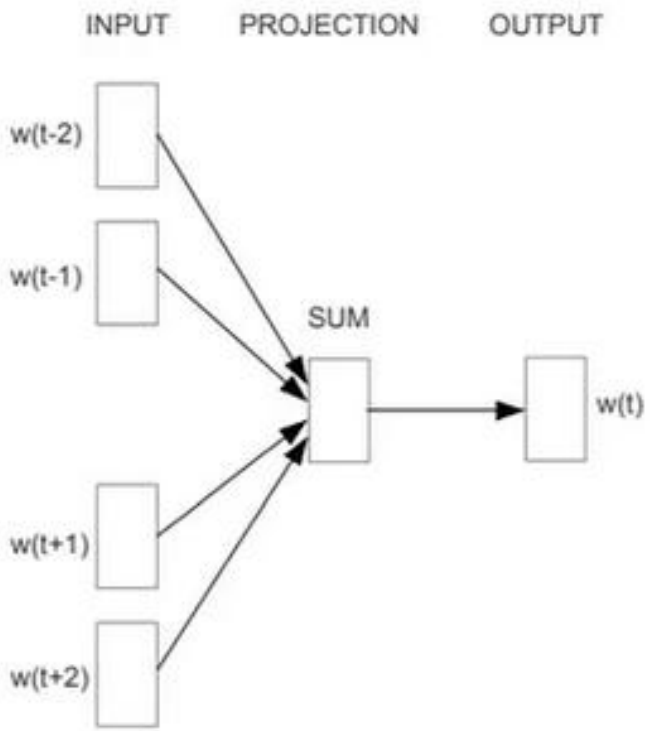
What is Word2Vec?

- It is a widely used method in natural language processing (NLP) that allows words to be represented as vectors in a continuous vector space.
- It is an effort to map words to high-dimensional vectors to capture the semantic relationships between words, developed by researchers at Google. Words with similar meanings should have similar vector representations, according to the main principle of Word2Vec.
- Word2Vec utilizes two architectures:

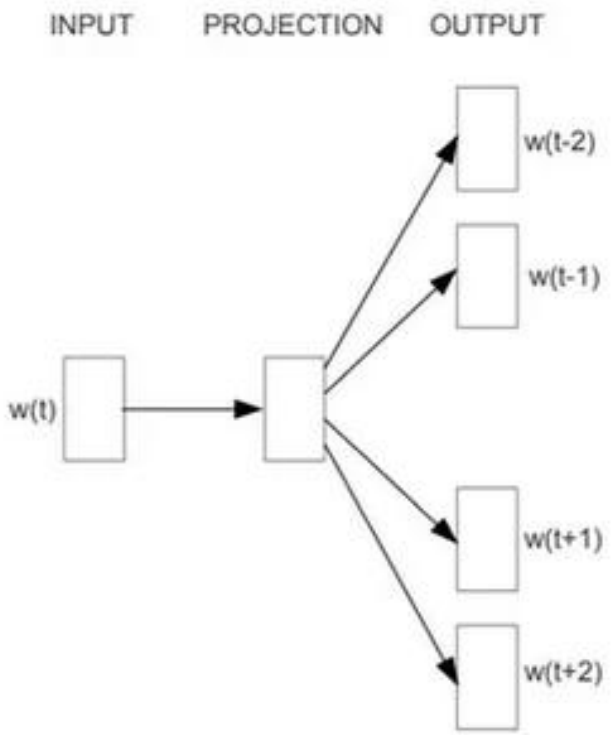
  CBOW (Continuous Bag of Words) and Skip Gram

Word embeddings are plotted, similar meaning words are closer in space, indicating their semantic similarity.





CBOW

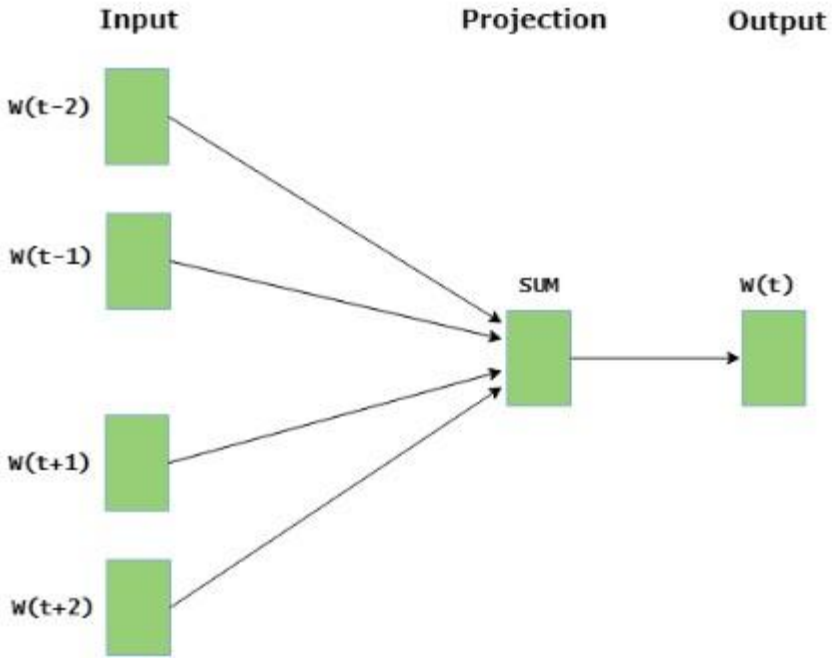


Skip-gram

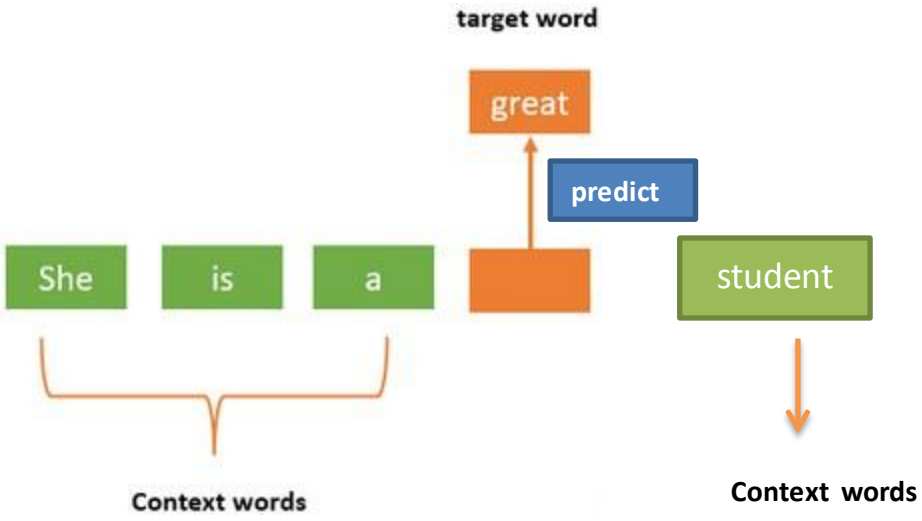
CBO
W

How does CBOW work?

- The CBOW model predicts the current word given context words within a specific window. The input layer contains the context words and the output layer contains the current word. The hidden layer contains the dimensions we want to represent the current word present at the output layer.

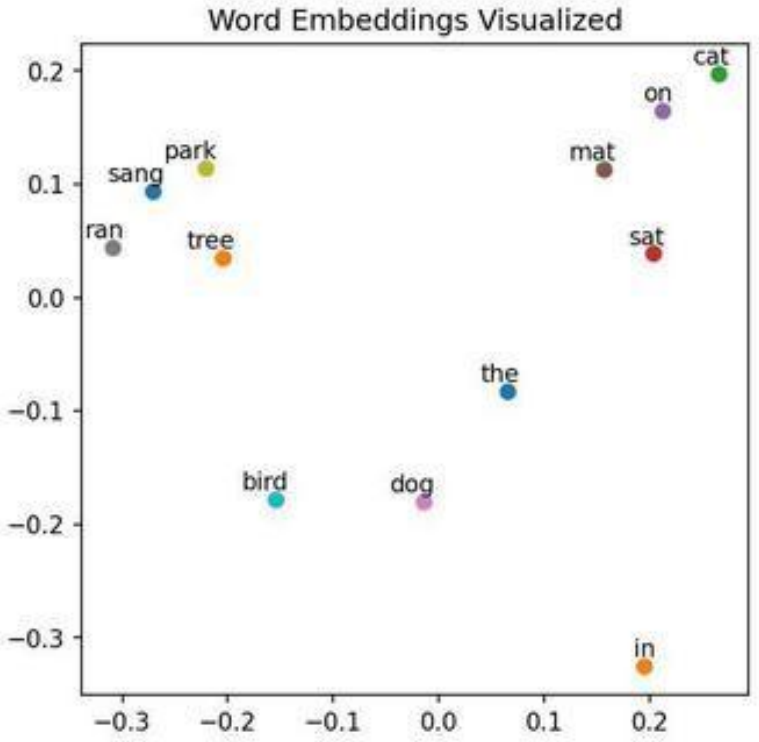


The model considers the context words and tries to predict the target term. The four $1*W$ input vectors will be passed to the input layer if have four words as context words are used to predict one target word. The hidden layer will receive the input vectors and then multiply them by a $W*N$ matrix. The $1*N$ output from the hidden layer finally enters the sum layer, where the vectors are element-wise summed before a final activation is carried out and the output is obtained from the output layer.


target word
great
predict
She
is
a
student
Context words
Context words

- **Input:** Context words
- **Output:** Target word
- **Learning Mechanism:** Predicts the missing word based on surrounding words
- **Advantage:** Fast and works well with frequent words
- **Disadvantage:** Less effective with rare words compared to Skip-gram

This visualization allows us to observe the similarity of the words based on their embeddings. Words that are similar in meaning or context are expected to be close to each other in the plot.



Difference between(BoW) model and (CBOW)

- The Bag-of-Words model and the Continuous Bag-of-Words model are both techniques used in natural language processing to represent text in a computer-readable format, but they differ in how they capture context.

- The **BoW** model represents text as a collection of words and their frequency in a given document or corpus. It does not consider the order or context in which the words appear, and therefore, it may not capture the full meaning of the text. the **CBOW** model can better capture the meaning of a word in a given context.

- The **BoW** model is simple and easy to implement, but it has limitations in capturing the meaning of language.
- In contrast, the **CBOW** model is a neural network-based approach that captures the context of words. It learns to predict the target word based on the words that appear before and after it in a given context window. By considering the surrounding words.

Text Corpus: "The quick brown fox jumps over the red dog"
Window Size: 2 (two words on either side of the target word).

For each target word, the **context words within the window** are used as input, and the **target word** is the output.

Step 1 (Target = "quick")

- **Context Words**: ["The", "brown"]
- **Target Word**: "quick"
- Training Sample: **(["The", "brown"], "quick")**

Step 2 (Target = "brown")

- **Context Words**: ["quick", "fox"]
- **Target Word**: "brown"
- Training Sample: **(["quick", "fox"], "brown")**

Step 3 (Target = "fox")

- **Context Words**: ["brown", "jumps"]
- **Target Word**: "fox"
- Training Sample: **(["brown", "jumps"], "fox")**

Step 4 (Target = "jumps")

- **Context Words**: ["fox", "over"]
- **Target Word**: "jumps"
- Training Sample: **(["fox", "over"], "jumps")**

Context Words	Target Word
["The", "brown"]	"quick"
["quick", "fox"]	"brown"
["brown", "jumps"]	"fox"
["fox", "over"]	"jumps"

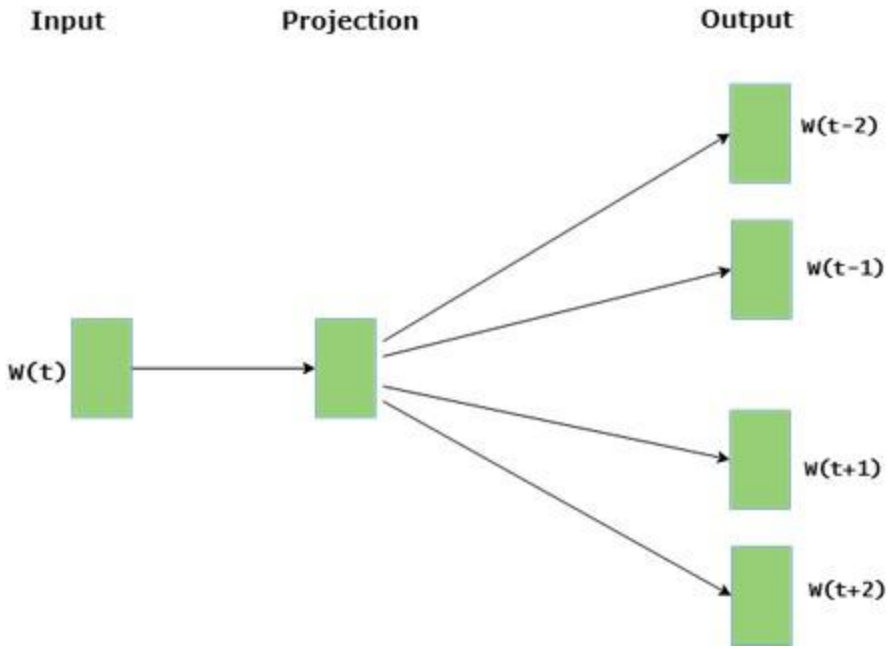
In CBOW, the process involves:

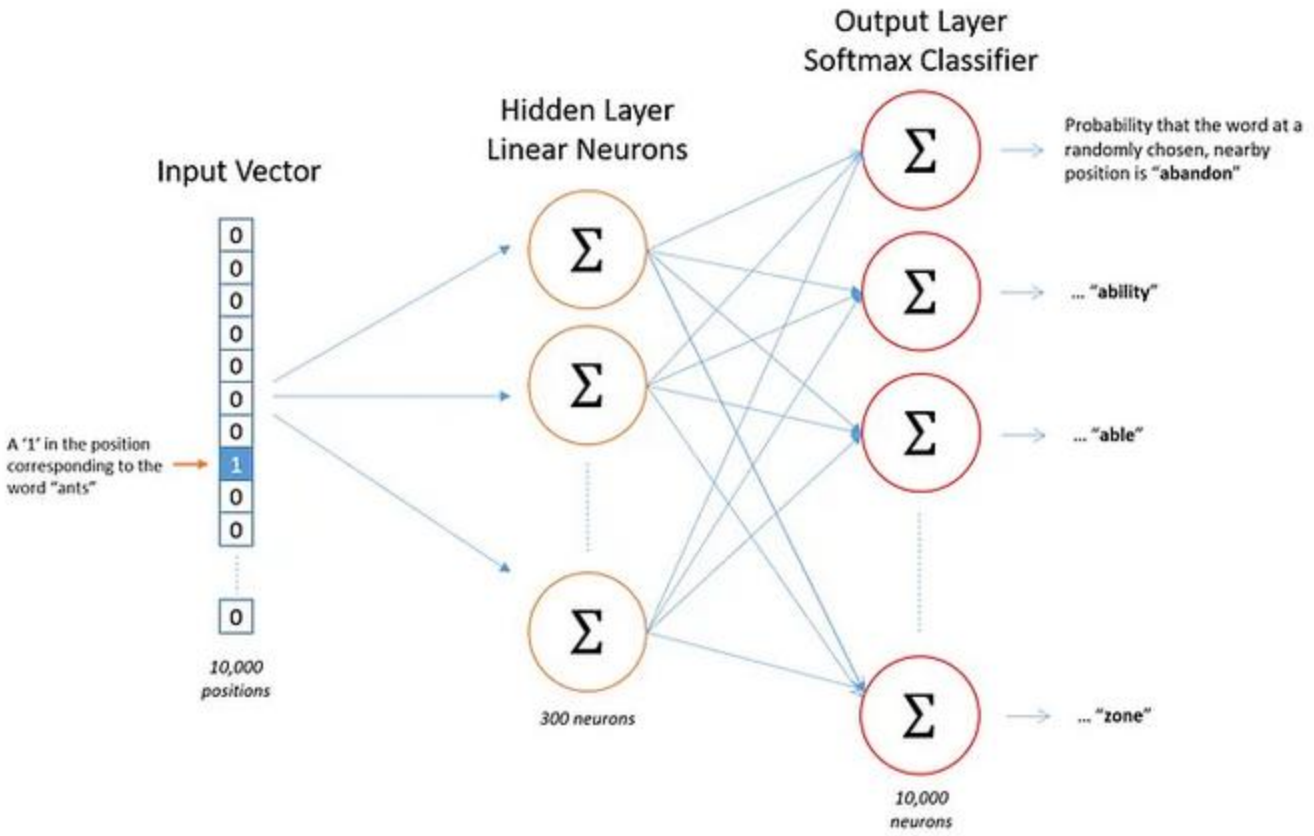
- **Input:** The context words (e.g., ["The", "brown"]).
- **Output:** The target word (e.g., "quick").
- The model learns to predict the target word based on the surrounding context.

Skip-gram

How does Skip-gram works?

Skip gram predicts the surrounding context words within specific window given current word. The input layer contains the current word and the output layer contains the context words. The hidden layer contains the number of dimensions in which we want to represent current word present at the input layer.

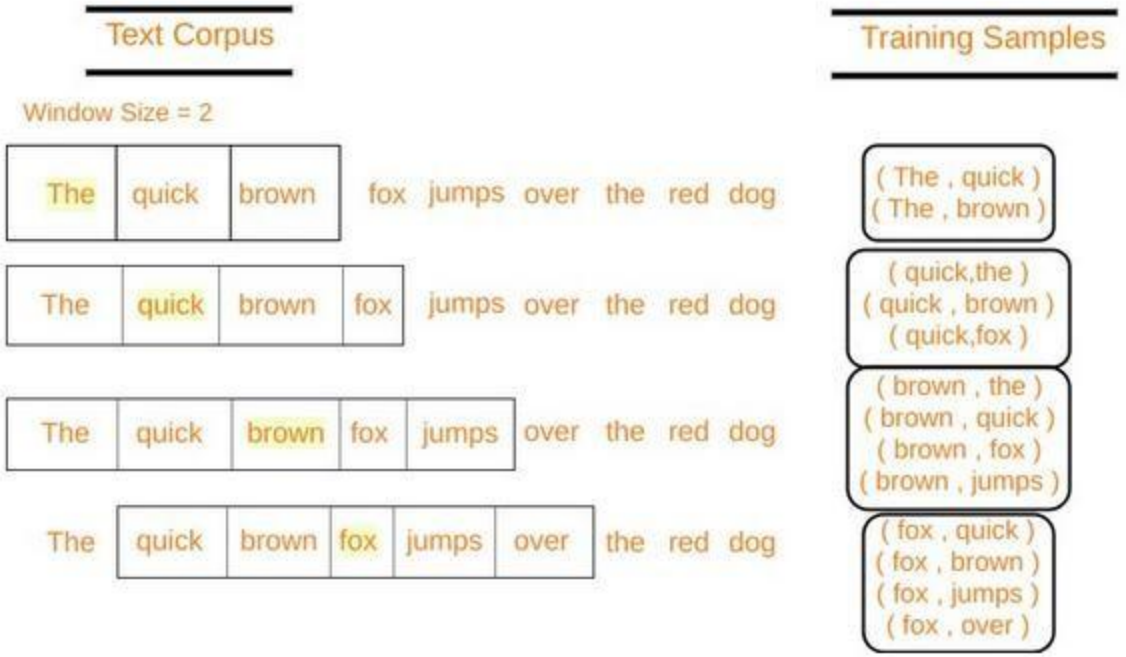



Output Layer
Softmax Classifier
Hidden Layer
Linear Neurons
Input Vector
Probability that the word at a randomly chosen, nearby position is "abandon"
... "ability"
... "able"
... "zone"
A '1' in the position corresponding to the word "ants"
0
0
0
0
0
0
0
1
0
0
0
10,000 positions
300 neurons
10,000 neurons

- Window_size = 2

Source Text	Training Samples generated from source text			
I will have orange juice and eggs for breakfast	(will, I)	(will, have)	(will, orange)	
I will have orange juice and eggs for breakfast	(have, I)	(have, will)	(have, orange)	(have, juice)
I will have orange juice and eggs for breakfast	(orange, will)	(orange, have)	(orange, juice)	(orange, and)
I will have orange juice and eggs for breakfast	(juice, have)	(juice, orange)	(juice, and)	(juice, eggs)
I will have orange juice and eggs for breakfast	(and, orange)	(and, juice)	(and, eggs)	(and, for)
I will have orange juice and eggs for breakfast	(eggs, juice)	(eggs, and)	(eggs, for)	(eggs, breakfast)
I will have orange juice and eggs for breakfast	(for, and)	(for, eggs)	(for, breakfast)	

2.


Text Corpus
Training Samples
Window Size = 2
The quick brown fox jumps over the red dog
(The , quick)
(The , brown)
The quick brown fox jumps over the red dog
(quick,the)
(quick , brown)
(quick,fox)
The quick brown fox jumps over the red dog
(brown , the)
(brown , quick)
(brown , fox)
(brown , jumps)
The quick brown fox jumps over the red dog
(fox , quick)
(fox , brown)
(fox , jumps)
(fox , over)

Step 1 (Target = "The")

- **Context Words:** ["quick", "brown"]
- Training Samples:
  - **("The", "quick")**
  - **("The", "brown")**

Step 2 (Target = "quick")

- **Context Words:** ["The", "brown", "fox"]
- Training Samples:
  - **("quick", "The")**
  - **("quick", "brown")**
  - **("quick", "fox")**

Step 3 (Target = "brown")

- **Context Words:** ["quick", "fox", "jumps"]
- Training Samples:
  - **("brown", "quick")**
  - **("brown", "fox")**
  - **("brown", "jumps")**

Step 4 (Target = "fox")

- **Context Words:** ["brown", "jumps", "over"]
- Training Samples:
  - **("fox", "brown")**
  - **("fox", "jumps")**
  - **("fox", "over")**

Target Word	Context Words	Training Samples Word
"The"	["quick", "brown"]	("The", "quick"), ("The", "brown")
"quick"	["The", "brown", "fox"]	("The", "quick"), ("The", "brown")
"brown"	["quick", "fox", "jumps"]	("brown", "quick"), ("brown", "fox"), ("brown", "jumps")
"fox"	["brown", "jumps", "over"]	("fox", "brown"), ("fox", "jumps"), ("fox", "over")

In Skip-gram:

- The **target word** is input (e.g., "The").
- The model predicts its **context words** within the window size.
- Each word contributes multiple training pairs (target, context), leading to rich training samples.

- In **Skip-gram**, the **target word** is the **input**, and the context words are the **output**.
- In **CBOW**, the **context words** are the **input**, and the target word is the **output**.

Aspect	Skip-gram	CBOW
Objective	Predicts surrounding context words given a target word.	Predicts the target word based on surrounding context words.
Input	A single target word.	Multiple context words within a fixed window.
Output	Context words within the window.	The target word.
Training Samples	Generates multiple (target, context) pairs for each word.	Generates one (context, target) pair per target word.
Performance	Performs better on rare words since each word is individually predicted.	Performs better on frequent words, leveraging aggregated context.
Speed	Slower to train due to more generated samples.	Much faster to train than Skip-gram.
Training Data	Works well with small datasets and represents rare words or phrases effectively.	Performs best with larger datasets, especially for frequent words.
Complexity	More computationally expensive, since it generates more training examples.	More computationally efficient, fewer training examples.
Output Embeddings	Captures fine-grained, detailed semantic relationships between words.	Captures smoother, more general semantic relationships.

Thanks