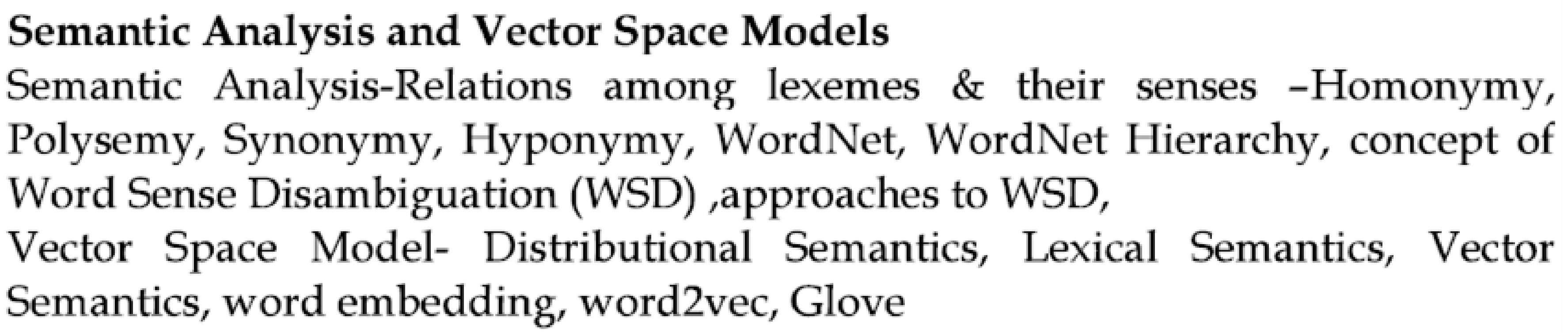
Unit 5
Semantic Analysis and Word Space Models

Madhura Vyawahare

Syllabus

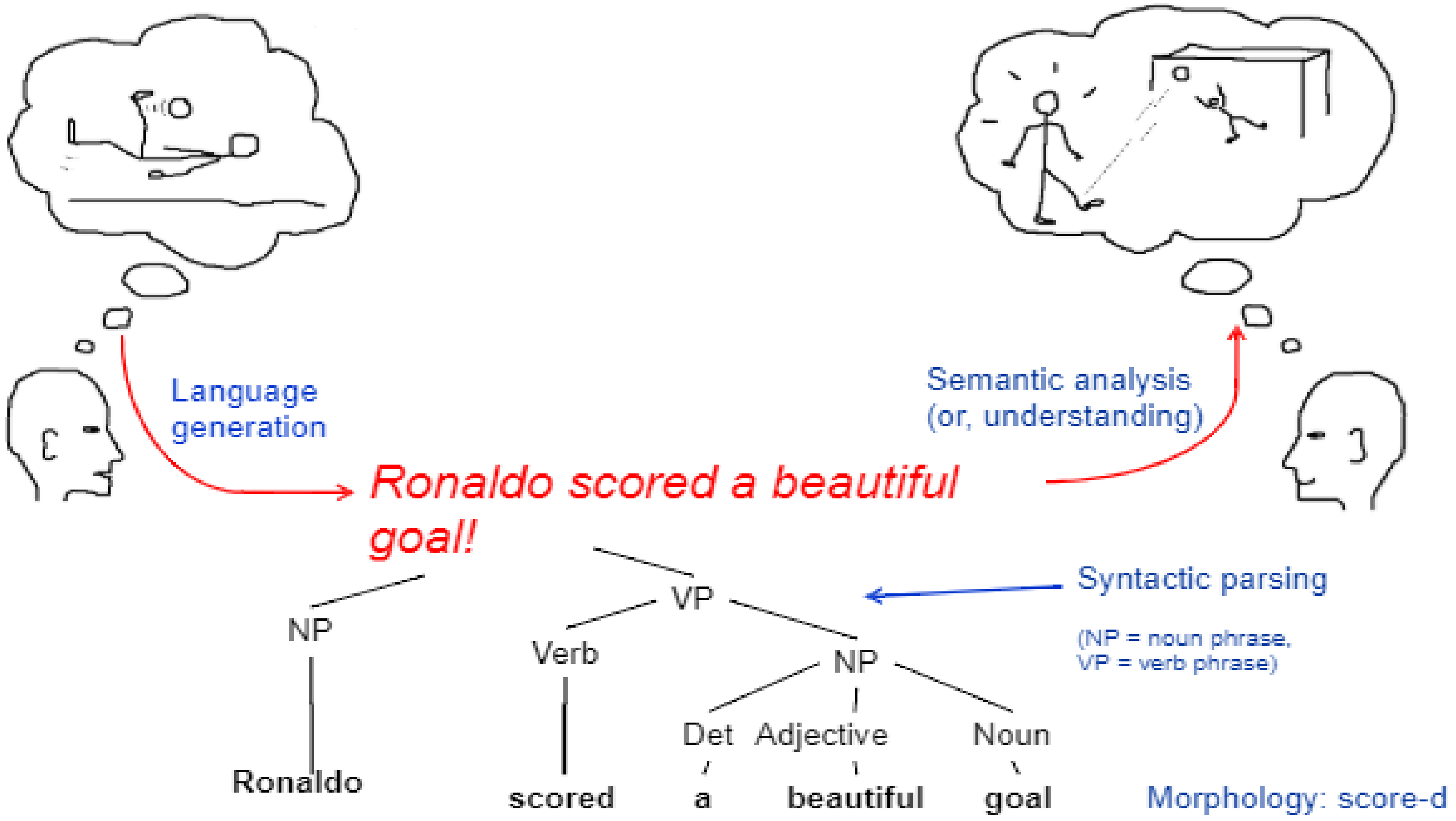


Semantic Analysis and Vector Space Models

Semantic Analysis-Relations among lexemes & their senses –Homonymy, Polysemy, Synonymy, Hyponymy, WordNet, WordNet Hierarchy, concept of Word Sense Disambiguation (WSD) ,approaches to WSD,

Vector Space Model- Distributional Semantics, Lexical Semantics, Vector Semantics, word embedding, word2vec, Glove

Semantic analysis



Semantic Analysis

- **Semantic analysis** is a subfield of natural language processing (NLP) that focuses on **understanding the meaning of text.** It involves extracting the underlying **meaning of words, phrases, and sentences, going beyond the surface-level syntactic structure.**

- **Lexical Semantics** is a subfield of linguistics that focuses on the **meaning of words and how they relate to each other.**
- It explores
  - the **internal structure of words,**
  - their r**elationships with other words,** and
  - their **meanings in different contexts.**

Relations among lexemes & their senses

Lexemes are the basic units of meaning in a language, while **senses are the different meanings a lexeme can have**.

Understanding the relationships between lexemes and their senses is crucial for natural language processing tasks.

key **relationships:**

- **Synonymy and Antonymy**
- **Hyponymy and Hypernymy**
- **Polysemy and Homonymy**
- **Collocation**

Homonymy

Homonyms are words that **share the same spelling or pronunciation** but have **different meanings**.

Types:

- **Homographs**: Same spelling, different meanings (may have different pronunciations).
  Example: lead (to guide) vs. lead (a metal).
- **Homophones**: Same pronunciation, different meanings (may have different spellings).
  - **Example**: *Bat* (an animal) vs. *Bat* (a sports equipment).

Relations among lexemes & their senses

- **Synonyms:** Words with similar meanings.

  Example: "big" and "large" are synonyms.

- **Antonyms:** Words with opposite meanings.

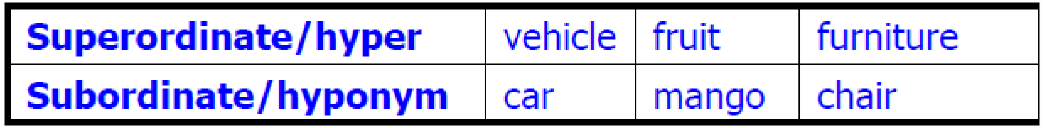
  Example: "hot" and "cold" are antonyms.

Hyponymy and Hypernymy

- **Hyponymy:** A word is a hyponym of another if it is a **more specific instance** of it.

  Example: "dog" is a hyponym of "animal."

- **Hypernymy:** A word is a hypernym of another if it is a more general term.
  - Example: "animal" is a hypernym of "dog."



Superordinate/hyper	vehicle	fruit	furniture
Subordinate/hyponym	car	mango	chair

Relations among lexemes & their senses

Polysemy and Homonymy

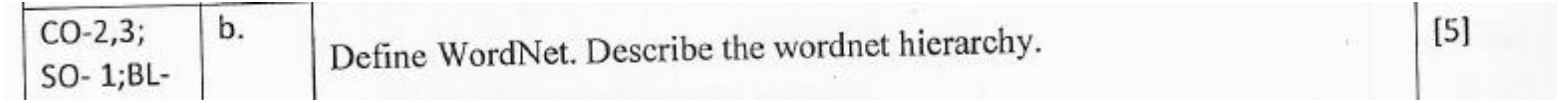
- **Polysemy:** A word with multiple related meanings.
- **Homonymy:** Words that are spelled and pronounced the same but have unrelated meanings.
  - **Homonyms**:
    - "**bear**" (the animal) and "**bear**" (to endure).
    - **"Bat"**: Can refer to a flying mammal or a piece of sports equipment used in baseball.
    - "**Rose**": A flower (homonym) and the past tense of "**rise**".
  - **Polysemy**:
    - "**Head**": The top of a person's body (polysemy) and the **leader** of a company.
    - **"Bank"**: Can mean a **financial institution** or the **side of a river**.

Collocation

- **Collocations:** Words that frequently occur together.
  - Example: "strong tea" is a common collocation.

Wordnet

- **WordNet** is a large, **lexical database of English** that groups words into sets of synonyms called **synsets**, and provides **definitions**, **examples**, and **semantic relationships** between them.
- It's widely used in **Natural Language Processing (NLP)**, **computational linguistics**, and **artificial intelligence** for tasks involving **word meaning**, **semantic similarity**, and **word sense disambiguation**.



CO-2,3; SO- 1;BL-	b.	Define WordNet. Describe the wordnet hierarchy.	[5]

Wordnet

- **WordNet** is a large lexical database of the English language that groups words into sets of synonyms (called **synsets**) and provides a rich network of semantic relationships between these words.
- It is widely used in natural language processing (NLP) tasks such as word sense disambiguation, information retrieval, and machine translation.
- WordNet is structured like a thesaurus but with more complex interrelations between words.
- Nouns, verbs, adjectives, and adverbs are all handled separately, each having their own synsets.
- WordNet does not include functional words like prepositions or conjunctions, as its focus is primarily on content words (nouns, verbs, adjectives, adverbs).

Key Features of WordNet

a) Synsets (Synonym Sets):

Words that have **similar meanings** are grouped into synsets. Each synset expresses a **unique concept**.

Example: The words *car*, *automobile*, *auto*, and *machine* all belong to the same synset because they all refer to the same concept of a motor vehicle.

b) Semantic Relations: WordNet organizes words based on various semantic relationships, including:

- **Synonymy**: Words in the same synset share synonymous meanings.
  - Example: *fast* and *quick*.
- **Antonymy**: Opposite meanings.
  - Example: *hot* vs. *cold*.
- **Hyponymy/Hypernymy**: Represents a hierarchy of more specific (hyponyms) and more general terms (hypernyms).
  - Example: *Dog* is a hyponym of *Animal*.
- **Meronymy/Holonymy**: Part-whole relationships.
  - Example: *Wheel* is a meronym of *Car* (a part of the whole car).
- **Troponymy**: Specific types of actions.
  - Example: *Run* is a troponym of *Move*

c) Word Senses: A word can have multiple meanings, and each meaning
corresponds to a different synset.
This is where **polysemy** is handled.
Example: *Bank* has one synset for a financial institution and another for the
side of a river.

d) **Lexical Relations**: WordNet also defines relationships like

Derived Forms: Words that are derived from others, such as play (verb) and

player (noun).

Python Code

- import nltk
- from nltk.corpus import wordnet as wn

- # Get synsets (synonyms sets) of the word "car"
- synsets = wn.synsets("car")
- print(synsets)

- # Get the definition of the first synset
- print(synsets[0].definition())

- # Get the hypernym (more general term) of "car"
- hypernym = synsets[0].hypernyms()
- print(hypernym[0].lemma_names())

WordNet Hierarchy

- **WordNet Hierarchy** refers to the way concepts (lexical items) are structured in WordNet based on their **semantic relationships**, particularly **hyponymy** (the "is-a" relationship) and **hypernymy** (the "is-a-kind-of" relationship).

- This hierarchical structure enables WordNet to represent concepts in a way that reflects their levels of generality or specificity.

1) Hypernymy and Hyponymy:

- **Hypernyms**: More general terms, also known as **superordinate** terms. These words describe broader categories.
  - Example: *Animal* is a hypernym of *Dog*.

- **Hyponyms**: More specific terms, known as **subordinate** terms. These words describe more specific instances or examples of a category.
  - Example: *Dog* is a hyponym of *Animal*.

- **2) Top-Level Categories**: WordNet organizes nouns, verbs, adjectives, and adverbs differently. **Nouns** and **verbs** have the most well-defined hierarchical structure.
- **Nouns**: The hierarchy starts with the most general concept, the **entity** (i.e., anything that exists).
  - From this, it branches into subcategories like **object**, **living thing**, **organism**, **animal**, and so on.
  - Example Hierarchy for *Dog*:
    - Entity → Physical Object → Living Thing → Organism → Animal → Mammal → Dog

3) Root Synset:

- In WordNet, the **root synset** of nouns is **"entity"**, which represents the most general concept. All nouns in WordNet are ultimately connected to this root.
- Verbs have a root concept like **"action"** or **"event"**, though they are less hierarchical than nouns.

4) Multiple Inheritance:

- Some words may have more than one hypernym, meaning they belong to more than one category. This is known as **multiple inheritance**.
- Example: *Whale* can be categorized under both *mammal* and *marine animal*, so it inherits properties from both categories.

5) Meronymy and Holonymy:

- Besides the hierarchical relations, WordNet also includes **part-whole** relationships:
  - **Meronymy** (part of): A word that describes a part of something else.
    - Example: *Wheel* is a meronym of *Car*.
  - **Holonymy** (whole of): A word that represents a whole that includes a given part.
    - Example: *Car* is a holonym of *Wheel*.

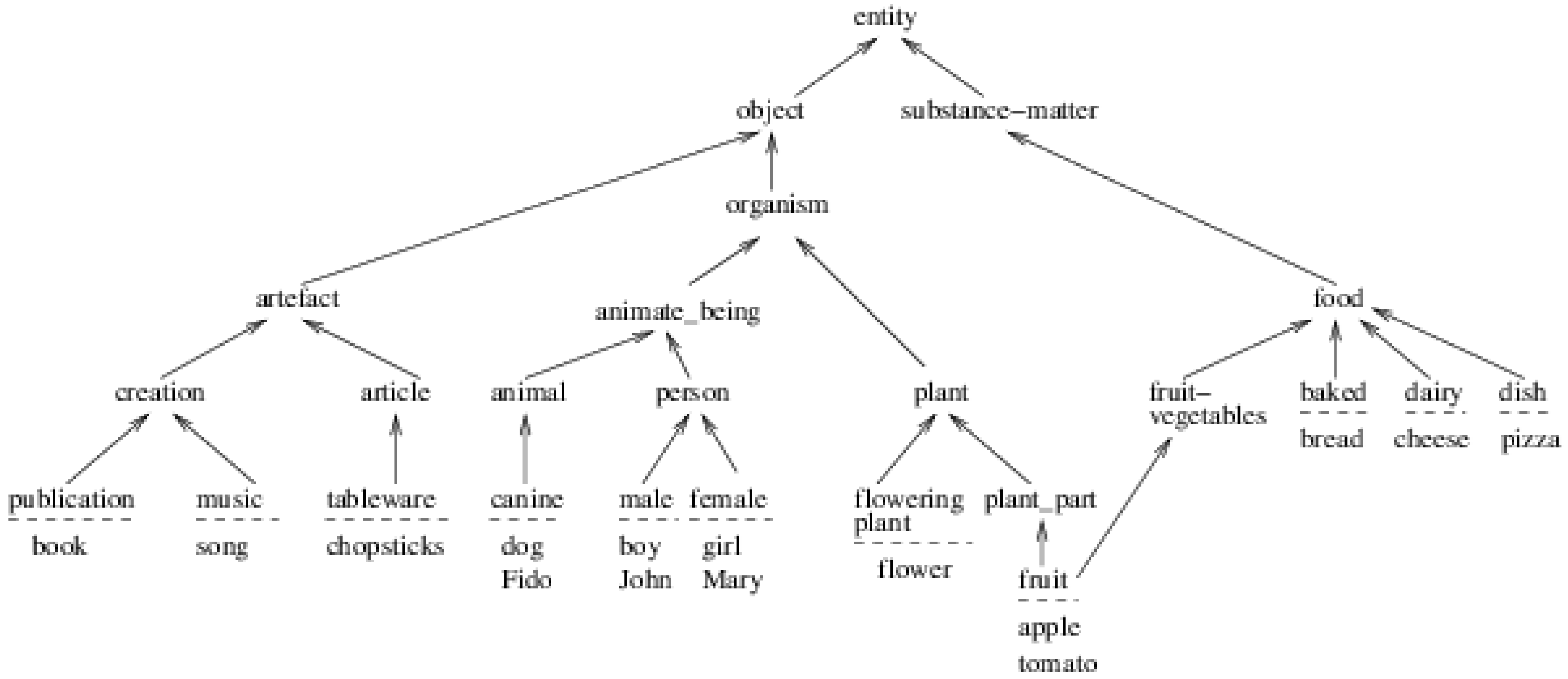
Example of Noun Hierarchy in WordNet:

Consider the word *dog* and how it fits into the WordNet hierarchy:

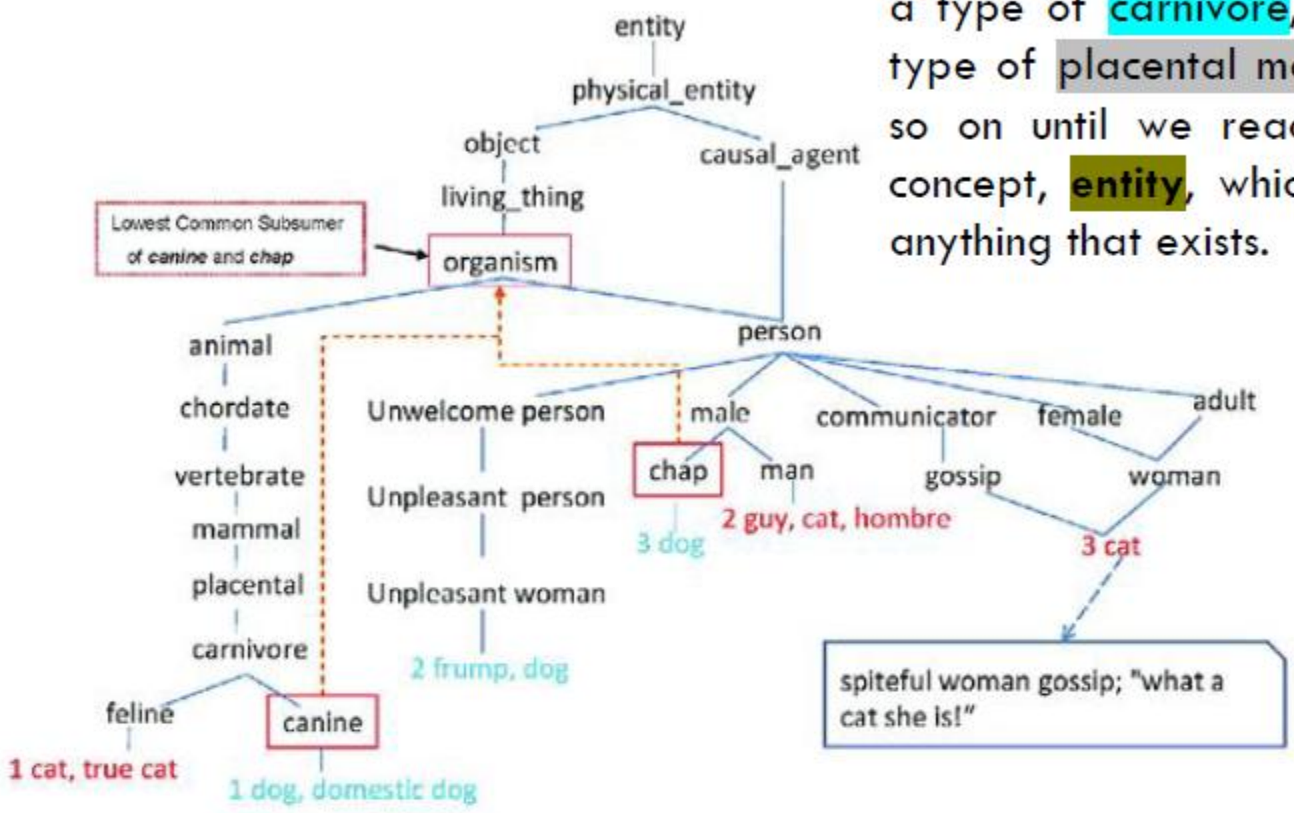
- **Dog**:
  - **Hypernyms** (more general terms):
    - Mammal
    - Animal
    - Living thing
  - **Hyponyms** (more specific terms):
    - Terrier
    - Dalmatian
  - **Meronyms** (parts of a dog):
    - Paw
    - Tail
  - **Holonym** (the whole to which a dog belongs):
    - Pet

```
import nltk
from nltk.corpus import wordnet as wn

# Get the synset for 'dog'
dog = wn.synset('dog.n.01')

# Get the hypernyms (general categories) of 'dog'
print("Hypernyms of dog:", dog.hypernyms())

# Get the hyponyms (specific types) of 'dog'
print("Hyponyms of dog:", dog.hyponyms())

# Get the meronyms (parts) of 'dog'
print("Meronyms of dog:", dog.part_meronyms())
```

Wordnet Hierarchy



WordNet Hierarchy



Dog is a type of canine, which is a type of carnivore, which is a type of placental mammal, and so on until we reach the root concept, entity, which refers to anything that exists.

How to Use WordNet in NLTK

Import and Download WordNet

```
import nltk
from nltk.corpus import wordnet as wn
nltk.download('wordnet')
```

Examples:

1. Get Synsets for a Word

```
print(wn.synsets('bank'))
[Synset('bank.n.01'), Synset('bank.n.02'), ..., Synset('bank.v.09')]
```

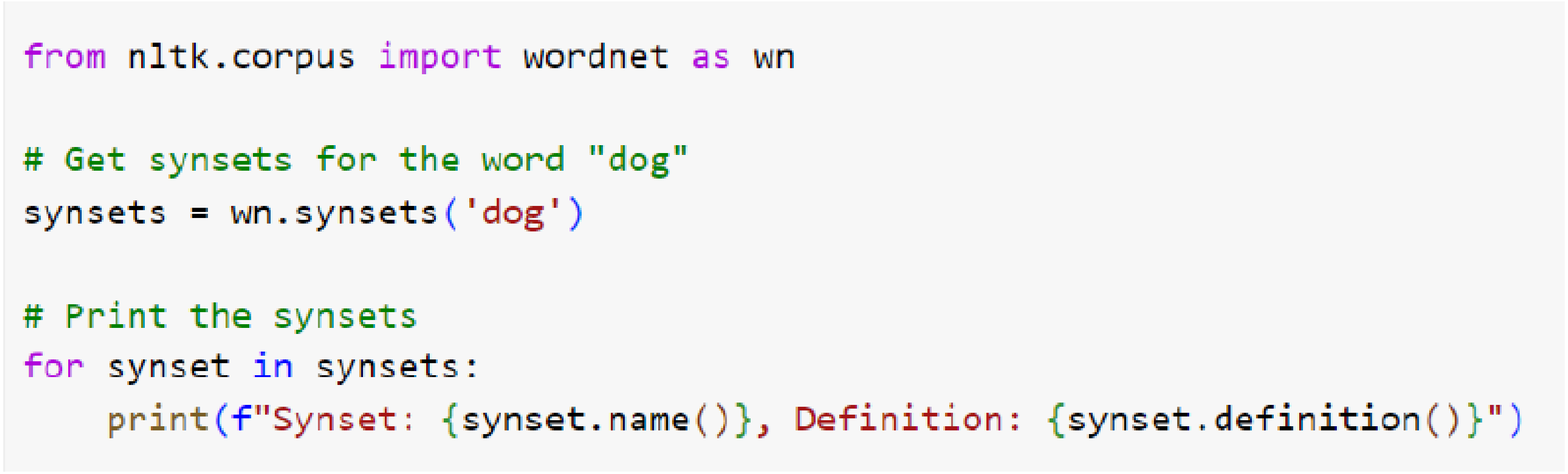
2. Get Definition and Examples of a Synset

```
syn = wn.synsets('bank')[0]
print("Definition:", syn.definition())
print("Examples:", syn.examples())
                    Definition: sloping land (especially the slope beside a body of water)
          Examples: ['they pulled the canoe up on the bank']
```

3. Lemmas (Synonyms in a Synset)

```
print(syn.lemmas())
print([lemma.name() for lemma in syn.lemmas()])
                    [Lemma('bank.n.01.bank')]
          ['bank']
```

WordNet Implementation – Synset Example



```python
from nltk.corpus import wordnet as wn

# Get synsets for the word "dog"
synsets = wn.synsets('dog')

# Print the synsets
for synset in synsets:
    print(f"Synset: {synset.name()}, Definition: {synset.definition()}")
```

WordNet

- WordNet is a semantically oriented dictionary of English, similar to a traditional thesaurus but with a richer structure.
- NLTK includes the English WordNet, with 155,287 words and 117,659 synonym sets.

Senses and Synonyms

a. Benz is credited with the invention of the motorcar.

b. Benz is credited with the invention of the automobile.

Synonyms

```
>>> from nltk.corpus import wordnet as wn
>>> wn.synsets('motorcar')
[Synset('car.n.01')]
```

- motorcar has just one possible meaning and it is identified as car.n.01, the first noun sense of car.
- The entity car.n.01 is called a synset, or “synonym set,” a collection of synonymous words (or “lemmas”):
- wn.synset('car.n.01').lemma_names
- ['car', 'auto', 'automobile', 'machine', 'motorcar']

- Synsets also come with a prose definition and some example sentences:

```
>>> wn.synset('car.n.01').definition
'a motor vehicle with four wheels; usually propelled by an internal combustion engine'
>>> wn.synset('car.n.01').examples
['he needs a car to get to work']
```

- With the help of wordnet we can get the senses and synonyms for individual word in corpus which is useful for removing the lexical ambiguities.

Word Sense Disambiguation (WSD)

- **Word Sense Disambiguation (WSD)** is the task of identifying the correct sense of a word in a given context.
- challenging when a **word has multiple meanings**, and the correct sense can only be determined by considering the surrounding words and the overall context of the sentence.
- WSD is a natural classification problem: Given a word and its possible senses, as defined by a dictionary, classify an occurrence of the word in context into one or more of its sense classes.

For example:

- The word **"bank"** can refer to a financial institution, the edge of a river, or a bench.
- To disambiguate the word in the sentence "I went to the bank to deposit my check," we need to consider the context and determine that "**bank" refers to a financial institution.**

Consider the two examples of the distinct sense that exist for the word “bass”

I can hear bass sound.

He likes to eat grilled bass.

- The occurrence of the word bass clearly denotes the distinct meaning. In first sentence, **it means frequency and in second, it means fish**.

Applications of WSD

- Machine Translation
- Information Retrieval
- Text Mining
- Lexicography
- Text to Speech
- Find more applications

- Task
  - Given a word and its context
  - Find its sense
- Task
  - Translate an English play into Hindi or Spanish
- Task
  - Spelling correction
  - aid/aide

Approaches and Methods to do WSD

- Dictionary Based
  - Overlap Based: Lesk Algorithm
  - Thesaurus-based Word Sense Disambiguation
  - Random Walk/ Walker’s Algorithm
- Supervised
- Unsupervised

Word Sense Disambiguation Approaches

Knowledge-based Approaches

- **Lesk Algorithm**: Uses dictionary definitions (glosses) and measures overlap with the context.
- **Thesaurus-based Approaches**: Utilize lexical resources like thesauri to find related words and determine the best sense.
- **Graph-based Methods (Random Walks)**: Represent senses and words as nodes in a graph and use algorithms like PageRank to find the most relevant sense.

Lesk Algorithm for Word Sense Disambiguation (WSD)

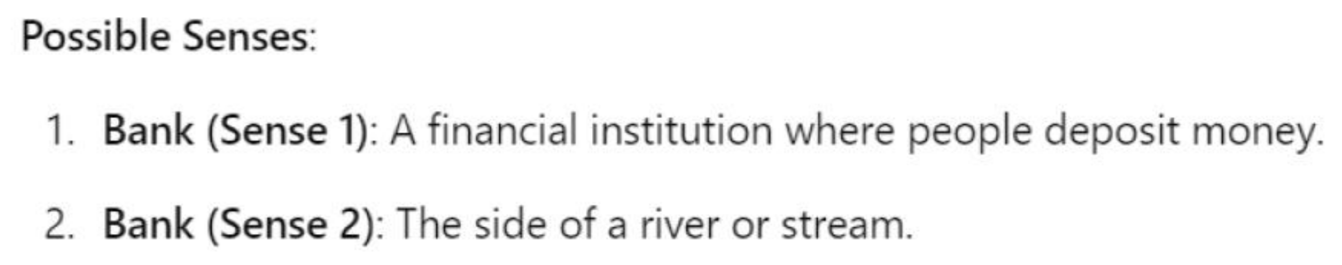
The **Lesk Algorithm** is a popular dictionary-based method for WSD that compares the o**verlap between the context of a word and the definitions of its senses in a dictionary or thesaurus**.

It assumes that the sense of a word that shares the most context with its surrounding words is the most likely sense.

Example:

Consider the sentence: I went to deposit money in the bank

The word "bank" has multiple meanings:


Possible Senses:

1. **Bank (Sense 1)**: A financial institution where people deposit money.
2. **Bank (Sense 2)**: The side of a river or stream.

Lesk Algorithm Steps:

1. **Context Extraction:** Extract the context of the ambiguous word "bank," which might include the surrounding words "I," "went," "to," "deposit," and "money."
2. **Dictionary Lookup:** Look up the definitions of each sense of "bank" in a dictionary or thesaurus.
3. **Context Overlap:** Calculate the overlap between the context and the definitions of each sense. This can be done by counting the number of words that appear in both the context and the definition.
4. **Sense Selection:** Choose the sense with the highest degree of overlap as the most likely sense.

Eg: I went to deposit money in the bank

Step 1 – Collect glosses of candidate senses of "bank"

- Bank_1 gloss: {institution, accepts, deposits, lends, money}
- Bank_2 gloss: {sloping, land, beside, body, water}

Step 2 – Collect glosses of Context (surrounding) words

- *deposited* gloss: {put, money, bank, safe}

  money gloss: {currency, coins, cash, deposits}

Step 3 – Compute overlaps

- For Bank_1 (financial):
  - Overlap with gloss of *deposited*: {money, bank, deposits} → 3 overlaps
  - Overlap with gloss of *money*: {money, deposits} → 2 overlaps
  - Total overlap = 5
- For Bank_2 (river):
  - Overlap with gloss of *deposited*: {} → 0
  - Overlap with gloss of *money*: {} → 0
  - Total overlap = 0

Result → Bank_1 (financial institution) is chosen.

- from nltk.wsd import lesk
- from nltk.tokenize import word_tokenize

- # Example sentence
- sentence = "He went to the bank to deposit some money."

- # Tokenize the sentence
- tokens = word_tokenize(sentence)

- # Use Lesk algorithm to disambiguate the word "bank"
- sense = lesk(tokens, 'bank')

- # Print the selected sense and its definition
- print(f"Selected sense: {sense}")
- print(f"Definition: {sense.definition()}")

Simplified Lesk Algorithm

Compares the **gloss of each candidate sense of the target word** directly with the **actual context words in the sentence**

Step 1 – Collect glosses of candidate senses of "bank"

- Bank_1 gloss: {institution, accepts, deposits, lends, money}
- Bank_2 gloss: {sloping, land, beside, body, water}

Step 2 – Compare each gloss directly with context words

Context words = {he, deposited, money, in, the}

- For Bank_1 (financial): overlap = {deposits, money} → 2 overlaps
- For Bank_2 (river): overlap = {} → 0

Result → Bank_1 (financial institution) is chosen.

Difference between LESK and Simplified LESK

- **LESK**
  Disambiguate a word by comparing the overlap between the **dictionary definition (gloss)** of each sense of the word and the glosses of *all* the words in the context (not just raw sentence words).
- **Limitation:**
  Needs access to **glosses for all words in the sentence** → computationally heavy, especially with large dictionaries like WordNet.
- **Simplified LESK**
  Instead of comparing glosses of all context words, compare the gloss of each sense of the target word with the actual context sentence (words around the target).

Knowledge Based WSD Approach – Lesk Algorithm

Sentence: *He sat on the bank of the river.*

Possible Senses:

1. **Bank (Sense 1)**: A financial institution where people deposit money.
2. **Bank (Sense 2)**: The side of a river or stream.

Lesk Algorithm Steps:

- **Context Words**: sat, river.
- **Gloss Overlap**:
    - **Sense 1**: Definition - "a financial institution where people deposit money." No overlap with context words.
    - **Sense 2**: Definition - "the side of a river or stream." Overlap with "river."

Selected Sense: Sense 2 (the side of a river).

Knowledge Based WSD Approach – Lesk Algorithm

Sentence: *The plant needs water to grow.*

Possible Senses:

1. **Plant (Sense 1)**: A living organism that typically grows in soil, has leaves, and produces oxygen.
2. **Plant (Sense 2)**: An industrial facility or factory.

Lesk Algorithm Steps:

- **Context Words**: needs, water, grow.
- **Gloss Overlap**:
  - **Sense 1**: Definition - "a living organism that grows in soil, has leaves, and produces oxygen." Overlap with "grow."
  - **Sense 2**: Definition - "an industrial facility for manufacturing." No overlap.

Selected Sense: Sense 1 (a living organism).

Knowledge Based WSD Approach – Lesk Algorithm

Sentence: *The bat flew across the night sky.*

Possible Senses:

1. **Bat (Sense 1)**: A nocturnal flying mammal.
2. **Bat (Sense 2)**: A wooden stick used in sports like baseball.

Lesk Algorithm Steps:

- **Context Words**: flew, night, sky.
- **Gloss Overlap**:
  - **Sense 1**: Definition - "a nocturnal flying mammal." Overlap with "night" and "flew."
  - **Sense 2**: Definition - "a wooden stick used to hit a ball in sports." No overlap.

Selected Sense: Sense 1 (a nocturnal flying mammal).

Show how you would disambiguate the following sentence using the Simple Lesk approach. Describe the algorithm and show how it would apply in this instance. **[5 Marks]**

I have tea and a slice of toast with a tablespoon of jam for breakfast.

Sense jam-1
Gloss: a crowded mass that impedes or blocks <a traffic jam>

Example: Trucks sat in a jam for ten hours waiting to cross the bridge.

Sense jam-2
Gloss: an often impromptu performance by a group especially of jazz musicians that is characterized by improvisation

Example: The saxophone players took part in a free-form jazz jam

Sense jam-3
Gloss: a food made by boiling fruit and sugar to a thick consistency

Example: He spread home-made jam on his toast.

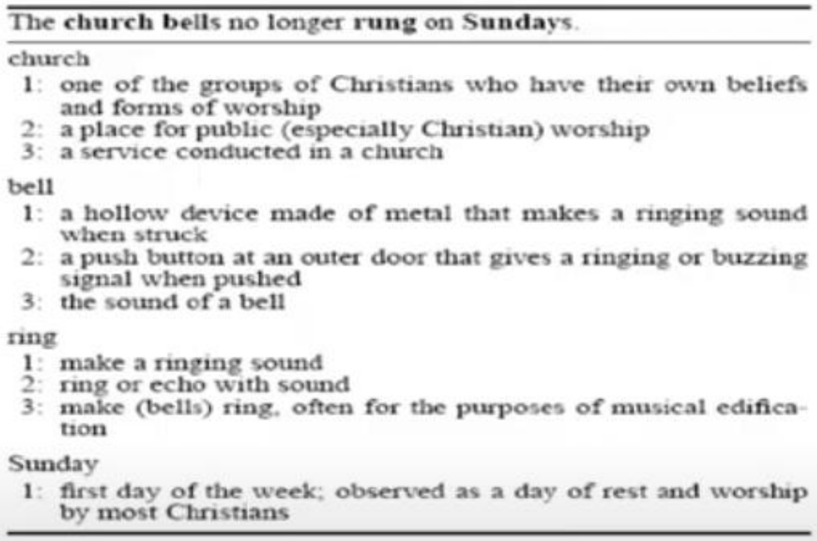
Walker's Algorithm

- **Used to disambiguate multiple words together from same sentence**
- **Thought all the words are having different senses, as they are in same sentence they are connected to each other**

1. Words in a sentence (or text) and possible senses of each word (from WordNet or any lexicon) are represented as nodes.
2. Edges are formed based on **semantic similarity** or **overlap** between senses.
3. The algorithm performs a **walk** (like random walk/PageRank style) to spread activation across the graph.
4. The sense with the **highest activation score** is selected as the most appropriate sense for the word in context.

This is conceptually similar to ***PageRank*** *used in Google search ranking.*

WSD Using Random Walk Algorithm



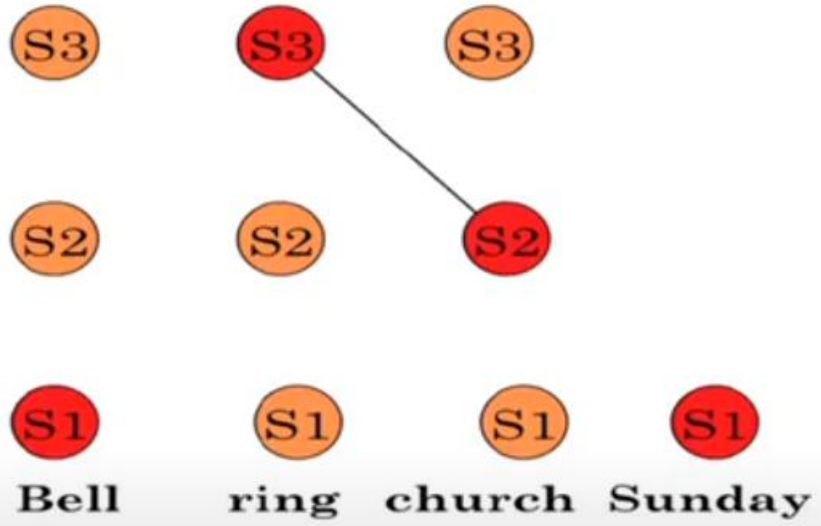
The **church bells** no longer **rung** on **Sundays**.

church
1: one of the groups of Christians who have their own beliefs and forms of worship
2: a place for public (especially Christian) worship
3: a service conducted in a church

bell
1: a hollow device made of metal that makes a ringing sound when struck
2: a push button at an outer door that gives a ringing or buzzing signal when pushed
3: the sound of a bell

ring
1: make a ringing sound
2: ring or echo with sound
3: make (bells) ring, often for the purposes of musical edification

Sunday
1: first day of the week; observed as a day of rest and worship by most Christians



Step 1: Add a vertex for each possible sense of each word in the text.

https://www.youtube.com/watch?v=O0s0UAEYQn0

WSD Using Random Walk Algorithm

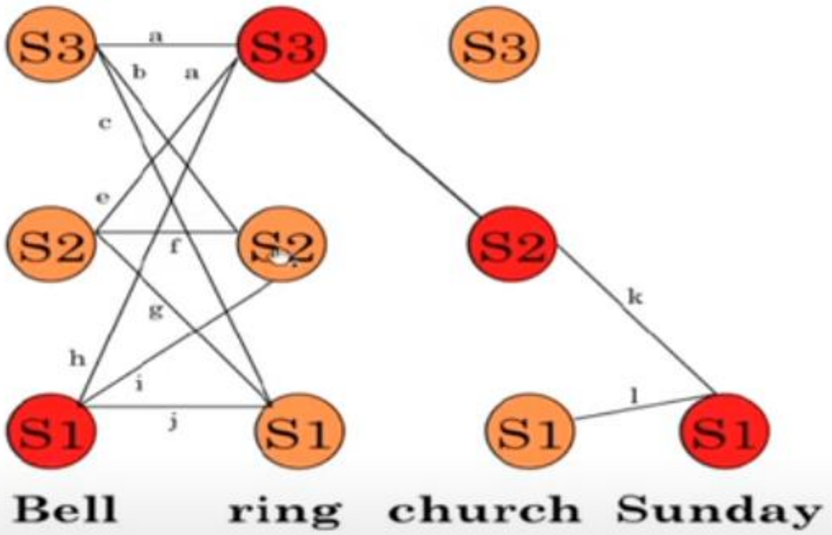
The **church bells** no longer **rung** on **Sundays**.

church
1: one of the groups of Christians who have their own beliefs and forms of worship
2: a place for public (especially Christian) worship
3: a service conducted in a church

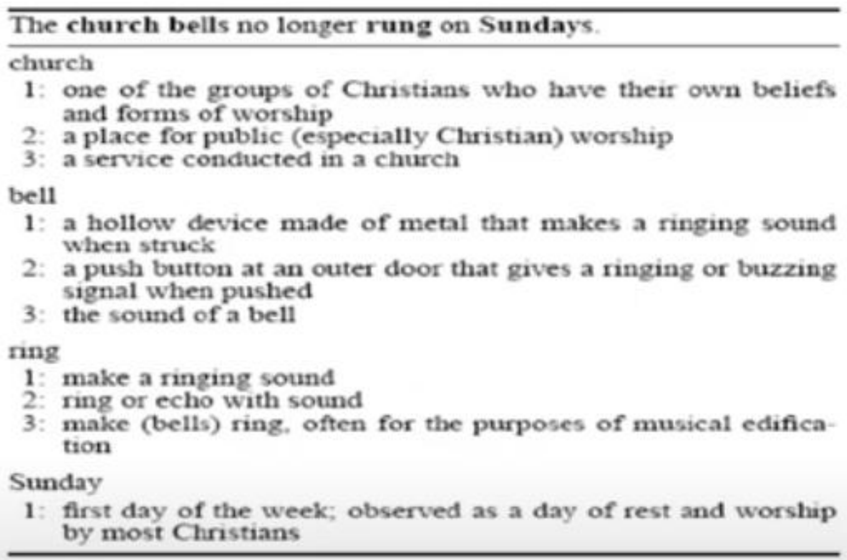
bell
1: a hollow device made of metal that makes a ringing sound when struck
2: a push button at an outer door that gives a ringing or buzzing signal when pushed
3: the sound of a bell

ring
1: make a ringing sound
2: ring or echo with sound
3: make (bells) ring, often for the purposes of musical edification

Sunday
1: first day of the week; observed as a day of rest and worship by most Christians



Step 2: Add weighted edges using definition based semantic similarity (Lesk's method).



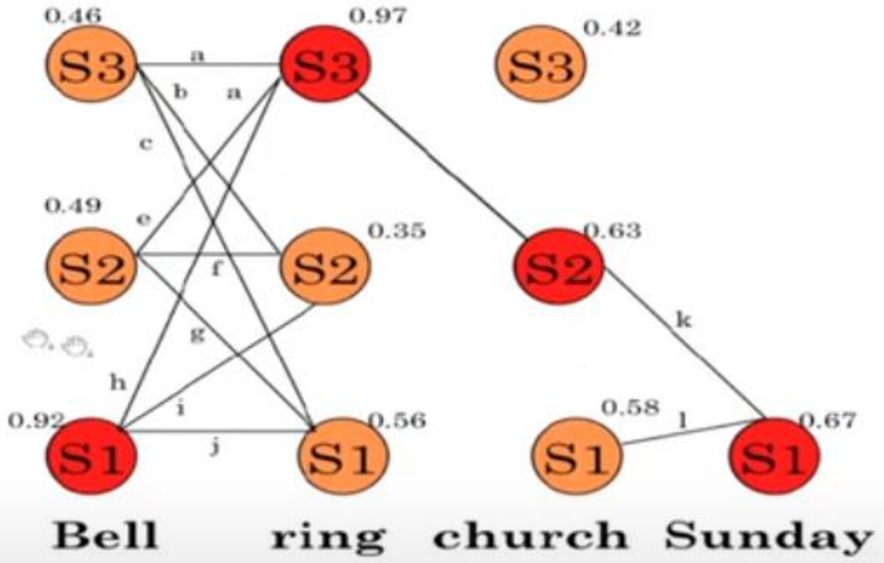
The **church bells** no longer **rung** on **Sundays**.

church
- 1: one of the groups of Christians who have their own beliefs and forms of worship
- 2: a place for public (especially Christian) worship
- 3: a service conducted in a church

bell
- 1: a hollow device made of metal that makes a ringing sound when struck
- 2: a push button at an outer door that gives a ringing or buzzing signal when pushed
- 3: the sound of a bell

ring
- 1: make a ringing sound
- 2: ring or echo with sound
- 3: make (bells) ring, often for the purposes of musical edification

Sunday
- 1: first day of the week; observed as a day of rest and worship by most Christians



Step 3: Apply graph based ranking algorithm to find score of each vertex (i.e. for each word sense).

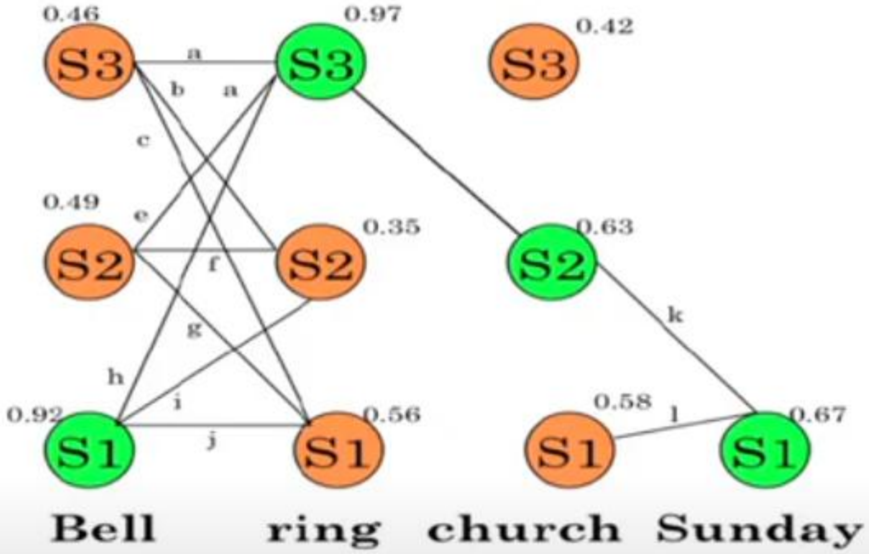
The **church bells** no longer **rung** on **Sundays**.

church
1: one of the groups of Christians who have their own beliefs and forms of worship
2: a place for public (especially Christian) worship
3: a service conducted in a church

bell
1: a hollow device made of metal that makes a ringing sound when struck
2: a push button at an outer door that gives a ringing or buzzing signal when pushed
3: the sound of a bell

ring
1: make a ringing sound
2: ring or echo with sound
3: make (bells) ring, often for the purposes of musical edification

Sunday
1: first day of the week; observed as a day of rest and worship by most Christians



Step 4: Select the vertex (sense) which has the highest score.

Supervised Approaches

- Supervised methods treat WSD as a **classification problem**, where each sense of a word is treated as a distinct class.
- These approaches require a **labeled dataset** where each word instance is tagged with its correct sense.
- The goal is to train a machine learning model using these labeled examples and predict the sense for new instances.

a) Naive Bayes Classifier:

- The Naive Bayes classifier uses the **frequencies of context words** to calculate the probability of a sense given the context. It assumes that the features (context words) are conditionally independent.

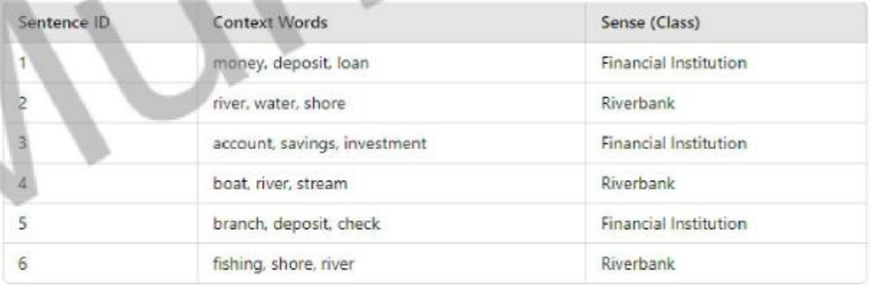
b) Decision Trees, SVMs, and Neural Networks:

- Other supervised classifiers such as **Support Vector Machines (SVMs)**, **Decision Trees**, and **Neural Networks** can also be used for WSD. These classifiers can be trained on labeled data to learn which sense of a word is more likely in a particular context.

Example on Naïve Bayes approach for WSD

- Question
  - For the given sentence, classify the word "bank" in its appropriate sense
  - As the **stream** flowed gently past the **shore**, we set up camp by the **bank** of the river, discussing the possibility of borrowing a **loan** to buy a boat for future trips
  - Here the highlighted words apart from bank are the context words
  - Consider the following two senses of the word bank
    - **Sense 1**: Bank as a financial institution
    - **Sense 2**: Bank as a riverbank.

- Consider the following training data



Sentence ID	Context Words	Sense (Class)
1	money, deposit, loan	Financial Institution
2	river, water, shore	Riverbank
3	account, savings, investment	Financial Institution
4	boat, river, stream	Riverbank
5	branch, deposit, check	Financial Institution
6	fishing, shore, river	Riverbank

Example on Naïve Bayes approach for WSD

- Solution
- $P(s_k|c) = \frac{P(c|s_k)P(s_k)}{P(c)}$
- Here $c = context\ word, s_k = sense\ class$
- Vocabulary of context word is
- $\{money, deposit, loan, river, water, shore, account, savings, investment, boat, stream, fishing, branch, check\}$

Vocabulary = 14 words.

$P(S_1) = P(\text{Financial Institute}) = \frac{3}{6} = \boxed{0.5}$

$P(S_2) = P(\text{River Bank}) = \frac{3}{6} = \boxed{0.5}$

$P(S_1 | c) = P(\text{financial institute} \mid \text{context word})$

$= \frac{\text{Context word in F.I.} + 1}{\text{Total financial institutional words} + 14}$ with smoothing

Context words :- Stream, loan, shore.

$P(\text{loan} | S_1) = \frac{1+1}{9+14} = \frac{2}{23}$

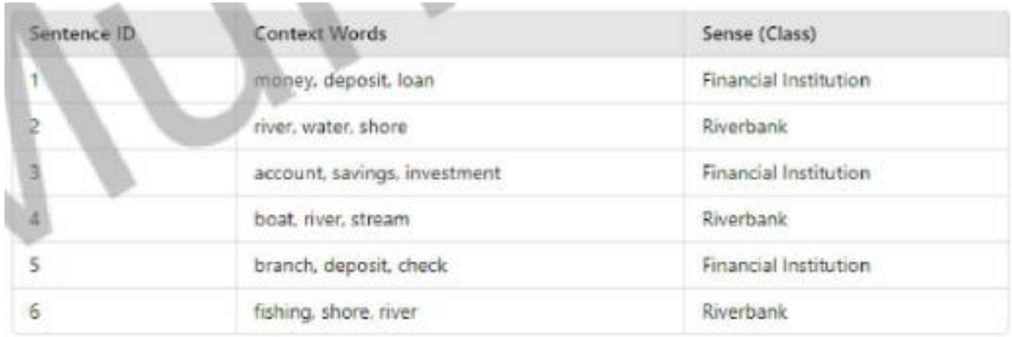
$P(\text{Stream} | S_1) = \frac{0+1}{9+14} = \frac{1}{23}$

$P(\text{Shore} | S_1) = \frac{0+1}{9+14} = \frac{1}{23}$

$P(\text{loan} | S_2) = \frac{0+1}{9+14} = \frac{1}{23}$

$P(\text{stream} | S_2) = \frac{1+1}{9+14} = \frac{2}{23}$

$P(\text{shore} | S_2) = \frac{1+1}{9+14} = \frac{2}{23}$



Sentence ID	Context Words	Sense (Class)
1	money, deposit, loan	Financial Institution
2	river, water, shore	Riverbank
3	account, savings, investment	Financial Institution
4	boat, river, stream	Riverbank
5	branch, deposit, check	Financial Institution
6	fishing, shore, river	Riverbank

$P(S_1 \mid \text{stream, shore, loan})$

$= P(S_1) * P(\text{stream} \mid S_1) * P(\text{shore} \mid S_1)$
$\quad * P(\text{loan} \mid S_1)$

$= \frac{1}{2} * \frac{1}{23} * \frac{1}{23} * \frac{2}{23} = \left(\frac{1}{23}\right)^3 = \frac{1}{(23)^3}$

$P(S_2 \mid \text{stream, shore, loan}) =$

$= P(S_2) * \{P(\text{stream} \mid S_2) * P(\text{shore} \mid S_2)$
$\quad * P(\text{loan} \mid S_2)$

$= \frac{1}{2} * \frac{1}{23} * \frac{2}{23} * \frac{2}{23} = \frac{2}{(23)^3}$

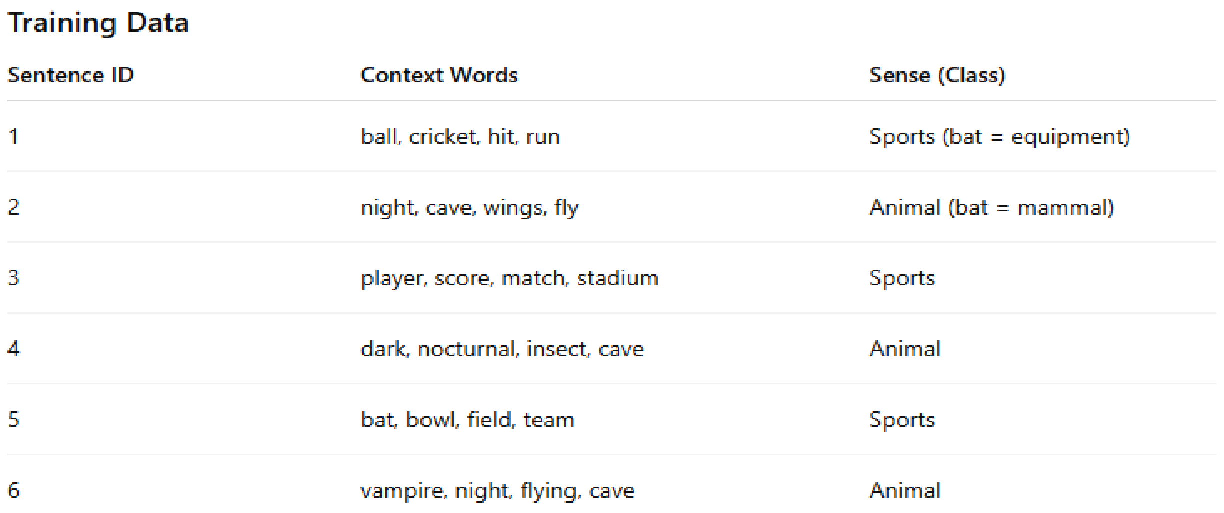
$\therefore P(S_1 \mid \text{stream, shore, loan}) <$
$\qquad P(S_2 \mid \text{stream, shore, loan}).$

$\therefore$ Sense 2 is selected.

$\therefore$ Bank here is River bank.

Example 2

- The bat flew out of the cave at night.
- Two possible senses of **bat**:
- **Sense 1**: Bat (Sports equipment – used in cricket/baseball)
- **Sense 2**: Bat (Nocturnal flying mammal)



Training Data

Sentence ID	Context Words	Sense (Class)
1	ball, cricket, hit, run	Sports (bat = equipment)
2	night, cave, wings, fly	Animal (bat = mammal)
3	player, score, match, stadium	Sports
4	dark, nocturnal, insect, cave	Animal
5	bat, bowl, field, team	Sports
6	vampire, night, flying, cave	Animal

Solution:

- Vocabulary = {ball, cricket, hit, run, night, cave, wings, fly/flying, player, score, match, stadium, dark, nocturnal, insect, team, bat, bowl, field, vampire} for smoothing

  - Sports total tokens = 12
  - Animal total tokens = 12
  - Vocabulary = 20

Training Data

Sentence ID	Context Words	Sense (Class)
1	ball, cricket, hit, run	Sports (bat = equipment)
2	night, cave, wings, fly	Animal (bat = mammal)
3	player, score, match, stadium	Sports
4	dark, nocturnal, insect, cave	Animal
5	bat, bowl, field, team	Sports
6	vampire, night, flying, cave	Animal

Considering the root word for flew = fly while comparing the context

For Sports Sense

- flew: count = 0 → $\frac{0+1}{12+20} = \frac{1}{32} \approx 0.031$
- cave: count = 0 → $\frac{1}{32} \approx 0.031$
- night: count = 0 → $\frac{1}{32} \approx 0.031$

$$P(context|Sports) \approx (0.031)^3 = 0.000030$$

For Animal Sense

- flew: synonym "fly/flying" appears → count = 2 → $\frac{2+1}{32} = \frac{3}{32} \approx 0.094$
- cave: count = 3 → $\frac{3+1}{32} = \frac{4}{32} = 0.125$
- night: count = 2 → $\frac{2+1}{32} = \frac{3}{32} \approx 0.094$

$$P(context|Animal) = 0.094 \times 0.125 \times 0.094 \approx 0.0011045$$

Step 4: Posterior Comparison

- Sports: $0.5 \times 0.000030 = 0.000015$
- Animal: $0.5 \times$ 0.0011045 = 0.00055225

$$P(Animal|context) > P(Sports|context)$$

Advantages of Naive Bayes for WSD:

1. **Simplicity**: Naive Bayes is easy to implement and train, and it scales well to large datasets.
2. **Efficiency**: It is computationally efficient, making it suitable for real-time applications.
3. **Probabilistic Interpretation**: The classifier provides probabilistic outputs, which can be useful for decision-making in uncertain situations.

3) Unsupervised Approaches:

Unsupervised methods attempt to solve WSD **without relying on labeled data**. These methods use **clustering or similarity-based techniques** to group word usages into distinct senses.

a) Clustering-Based Methods:
Words are represented as vectors based on their surrounding context. These vectors are then clustered, and each cluster represents a different sense of the word.

b) Contextual Embeddings: More modern approaches use word embeddings (such as Word2Vec, GloVe, or contextualized embeddings like BERT) to represent words and their contexts. The idea is that similar meanings will have similar vector representations.

c) Word Embeddings:
Using unsupervised learning techniques like Word2Vec, we can generate embeddings for words based on their contexts. This allows us to capture the multiple senses of a word in vector space, where each word's context determines its proximity to one sense or another.

d) Sense Embeddings: More advanced techniques create embeddings not just for words, but for individual senses of a word. This can be done by clustering different uses of the word and generating separate embeddings for each sense.

4) Semi-Supervised Approaches

Semi-supervised approaches leverage a small amount of labeled data along with a larger set of unlabeled data. The labeled examples are used to bootstrap the learning process, after which the model is refined using the unlabeled examples.

a) Bootstrapping:

This method begins with a small number of labeled instances for each sense. The system then iteratively labels new instances based on confidence thresholds and updates the model.

b) Co-Training:

Two different classifiers are trained on separate views of the data (e.g., one classifier might use the words before the target word, while the other uses the words after the target word). These classifiers teach each other by labeling data for the other, thus growing the training set.

Applications of WSD

Information Retrieval (IR): Improves search engine results by distinguishing between different meanings of ambiguous terms.

Machine Translation: Ensures that words are translated into the correct sense in the target language.

Question Answering: Disambiguates the meaning of words in questions to provide more accurate answers.

Text Summarization: Helps ensure the correct sense of words is used in summaries.

Vector Semantics in NLP

- Vector semantics is an approach in natural language processing (NLP) that represents words, phrases, or even sentences as mathematical objects, specifically **vectors** in a multi-dimensional space.
- Word vectors are simply vectors of numbers that represent the meaning of a word.
- These **vectors encode semantic information** based on the distribution of words in a corpus, allowing for the quantification of meaning and similarity between linguistic elements.

Applications of Vector Semantics in NLP

Semantic Similarity: Word vectors can be used to find words or documents that are semantically similar. For example, finding synonyms or detecting duplicate content.

Machine Translation: Vector semantics helps align words and phrases across languages, facilitating automatic translation.

Information Retrieval: Search engines and recommendation systems can leverage vector semantics to match queries with relevant documents based on semantic similarity.

Sentiment Analysis: Word and sentence embeddings are used to identify the sentiment behind a text by capturing its context and tone.

Named Entity Recognition (NER): By using embeddings, NER systems can identify named entities in text more accurately, even when those entities are rare or context-specific.

Question Answering and Chatbots: Contextual embeddings allow systems like BERT to handle complex question answering, where understanding the context and meaning of a query is crucial.

Basic Vectorization Approaches

Assigning Unique Numbers

One Hot Encoding

Simple Bag of Word

TF-IDF

Issue with Unique numbers

Numbers are random

They don't capture relationships between words

Issue with One Hot encoding

- As data increases more number of 0s as compared to 1s (Sparse Matrix)
- Also it is not good while representing lexical similarity.
  - Eg. Good and great are somewhat similar but they will be treated as complete separate entity when dealing with one hot encoding
  - Eg. Apple and Mango have same characteristics as fruits.
- Does Not capture relationship between words
- Computationally inefficient

How can we capture similarities between two words?

Word Embedding

Frequency based: BOW, TF-IDF, Glove

Prediction based: Word2Vec

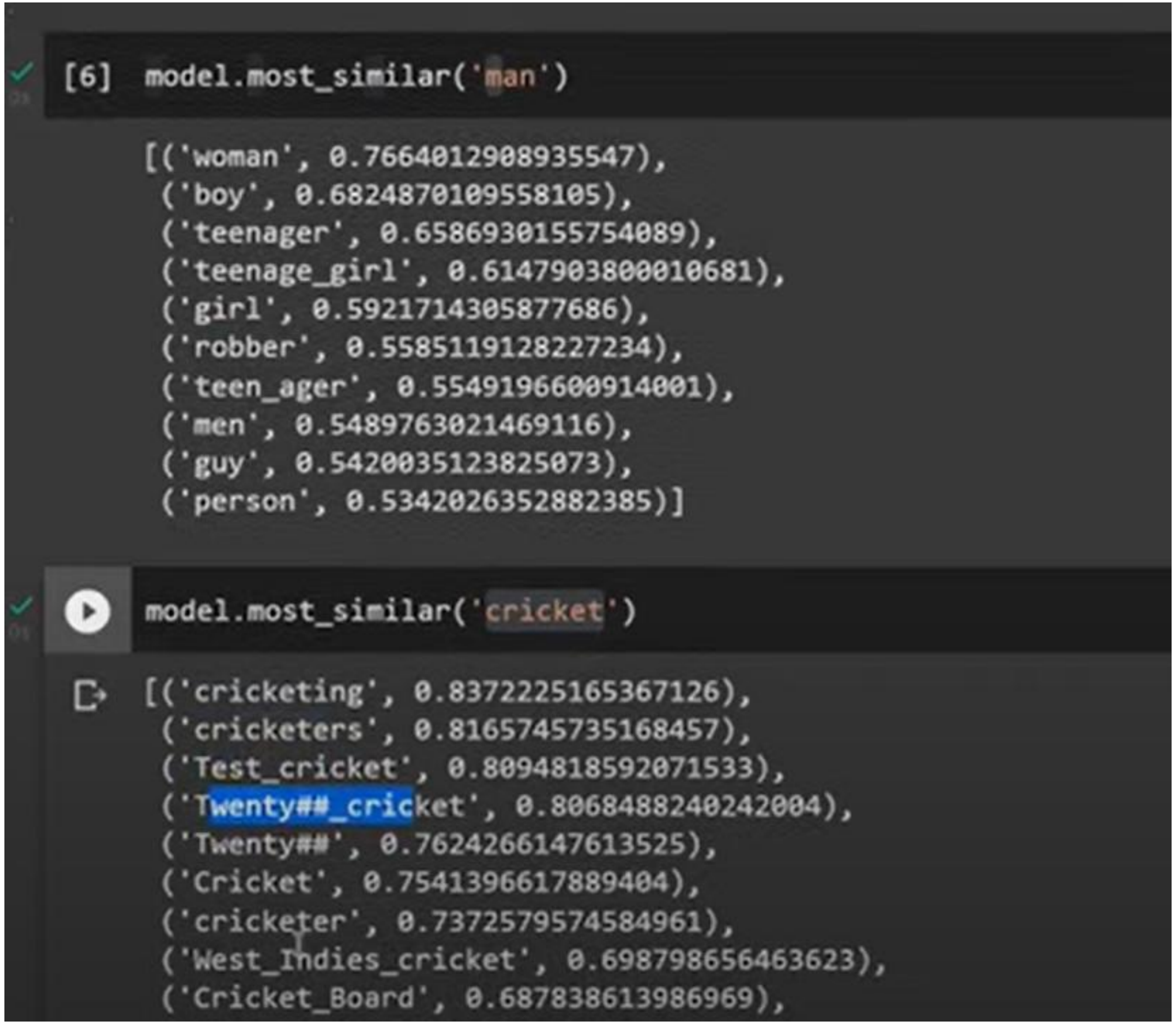
Word2Vec

The **Word2Vec** model is a popular word embedding technique in **natural language processing (NLP).**

It is a **pre-trained Neural Network model. Trained on Google news corpus containing 3 billion words.**

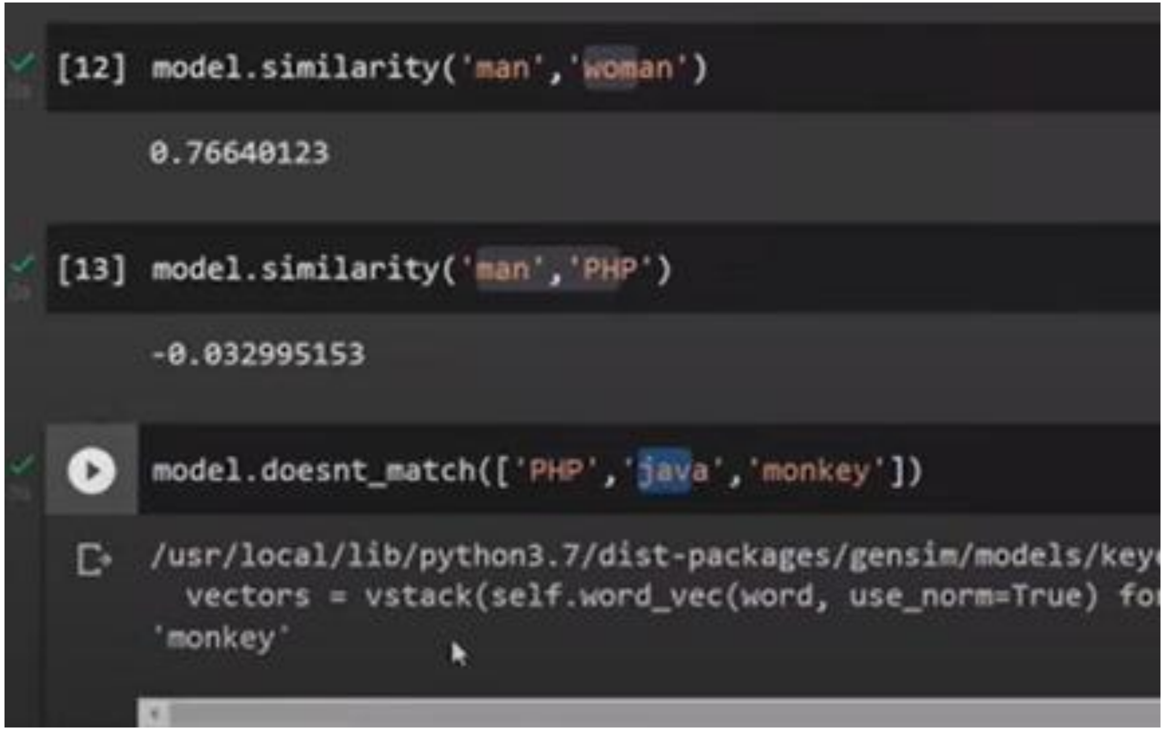
Model consists of 300 dimensional vector for 3 billion words and phrases

- It represents words as vectors in a continuous vector space where semantically similar words are placed close to each other.
- Word2Vec is based on the **distributional hypothesis** — the idea that words appearing in similar contexts tend to have similar meanings.
- Apple and Mango will be represented as vectors whose cosine similarity will be nearly similar.



```
[6] model.most_similar('man')

[('woman', 0.7664012908935547),
 ('boy', 0.6824870109558105),
 ('teenager', 0.6586930155754089),
 ('teenage_girl', 0.6147903800010681),
 ('girl', 0.5921714305877686),
 ('robber', 0.5585119128227234),
 ('teen_ager', 0.5549196600914001),
 ('men', 0.5489763021469116),
 ('guy', 0.5420035123825073),
 ('person', 0.5342026352882385)]
```

```
model.most_similar('cricket')

[('cricketing', 0.8372225165367126),
 ('cricketers', 0.8165745735168457),
 ('Test_cricket', 0.8094818592071533),
 ('Twenty##_cricket', 0.8068488240242004),
 ('Twenty##', 0.7624266147613525),
 ('Cricket', 0.7541396617889404),
 ('cricketer', 0.7372579574584961),
 ('West_Indies_cricket', 0.698798656463623),
 ('Cricket_Board', 0.687838613986969),
```



```
[12] model.similarity('man','woman')

     0.76640123

[13] model.similarity('man','PHP')

     -0.032995153

     model.doesnt_match(['PHP','java','monkey'])

     /usr/local/lib/python3.7/dist-packages/gensim/models/key
       vectors = vstack(self.word_vec(word, use_norm=True) fo
     'monkey'
```

Two Main Architectures

- **CBOW(Continuous Bag of Words)**:
  - Predicts a word given its surrounding context.
  - This model averages the context word vectors and uses this to predict the target word.
  - Example: Given the sentence "The cat sat on the ___",
  - CBOW will predict the word "mat" based on the context of "The cat sat on the”.

- **Skip-gram**:
  - Predicts surrounding words given a target word.
  - This model works better with smaller datasets and rare words.
  - Example: Given the word "cat", Skip-gram will predict the surrounding words like "The", "sat", and "on".

CBOW

The CBOW model tries to predict a target word based on its surrounding context.

1. It takes a window of words before and after the target word
2. averages their embeddings
3. and uses this information to predict the target word.

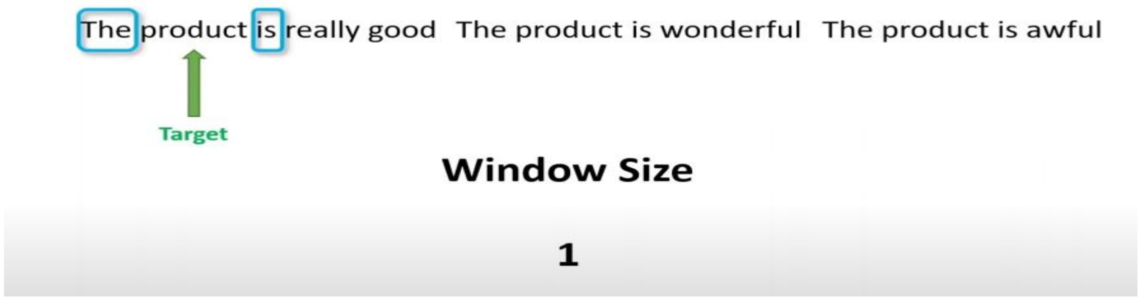
Example:

For the sentence: "The cat sat on the mat."

- If the target word is **"sat"**, and we are using a **window size of 2**, the context words will be:
- Window size = 2 means we take **2 words on the left** and **2 words on the right** of the target word.
- Context words: ["The", "cat", "on", "the"]
- In **CBOW**, you would use [The, cat, on, the] to predict **"sat"**.
- In **Skip-gram**, you would use **"sat"** to predict each of [The, cat, on, the]

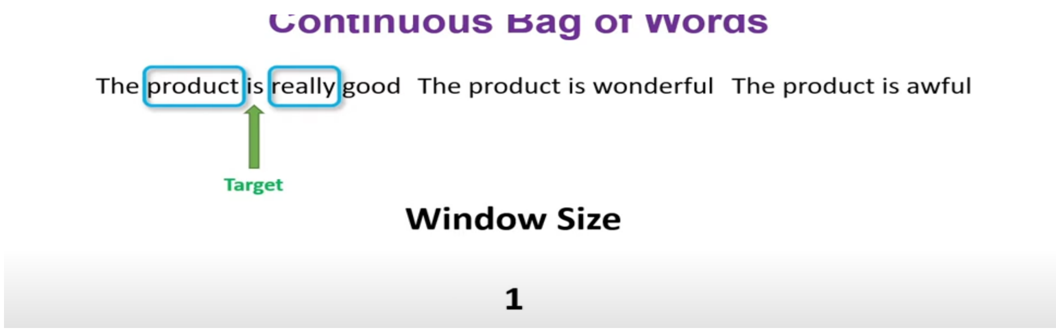
Continuous Bag of Words

The product is really good   The product is wonderful   The product is awful


The product is really good   The product is wonderful   The product is awful

Window Size

1


Continuous Bag of Words

The product is really good   The product is wonderful   The product is awful

Window Size

1

Steps in CBOW Model

1. **Context Window:** For each target word, select a window of words around it. The size of the window can be controlled.

1. **Vector Averaging**: For each context word in the window, retrieve its word embedding (initially random vectors that get updated during training). The model takes the average of these embeddings to form a combined context vector.

1. **Prediction**: The averaged context vector is then passed through a neural network (typically a simple feed-forward network) to predict the target word.

1. **Training**: The model is trained by minimizing the prediction error over the entire dataset. Specifically, it maximizes the probability of predicting the correct target word from the context words.

Example

- Consider the sentence: **"The cat sits on the mat.“**
- Let’s assume we want to predict the word **“sits”** based on its context, and we use a **window size of 2** (which means we will consider 2 words before and 2 words after the target word).

Step 1: Context Window

Context and Target Word:

- In the sentence "The cat sits on the mat", with a window size of 2, we consider the following context words around the target word **“sits”**:
- Context words: ["The", "cat", "on", "the"]
- Target word: "sits"

Step 2: Vector Averaging

Input to CBOW

So, vocabulary size = 5.

Each word is represented as a **one-hot vector** (for training):

- the → [1, 0, 0, 0, 0]
- cat → [0, 1, 0, 0, 0]
- sat → [0, 0, 1, 0, 0]
- on → [0, 0, 0, 1, 0]
- mat → [0, 0, 0, 0, 1]

Average them (CBOW uses average of context vectors):

([1,0,0,0,0] + [0,1,0,0,0] + [0,0,0,1,0] + [1,0,0,0,0]) / 4

= [2,1,0,1,0] / 4

= [0.5, 0.25, 0, 0.25, 0]

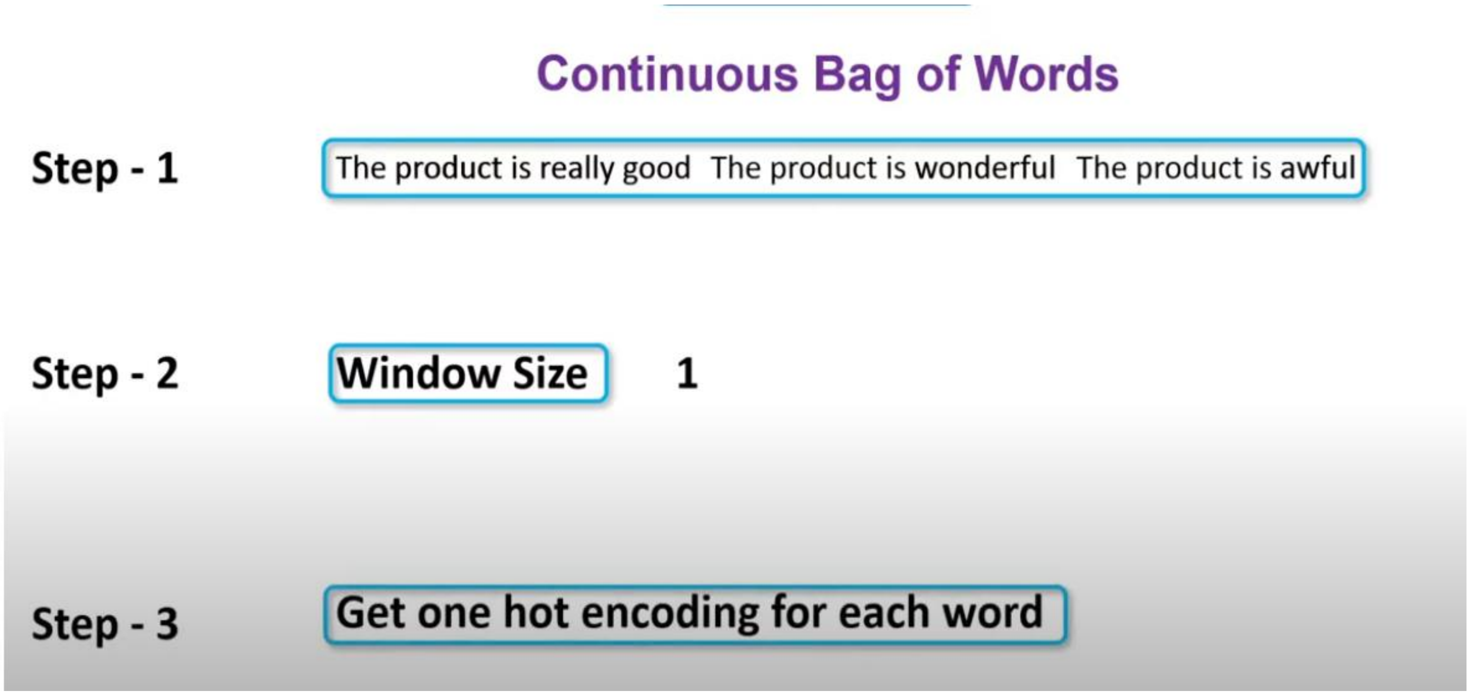
Step 3: Prediction

- CBOW has a hidden layer (the **embedding layer**) of size N (say N=2).
- Input vector is multiplied by weight matrix **W (5×2)**.
- This produces a 2D embedding representation.
- Then another weight matrix maps it back to vocabulary size (2×5), and softmax gives probabilities for all words.
- The model tries to **predict the target word = "sat"**.
- So it adjusts weights using **backpropagation** until the probability of “sat” is highest.
- Context words → Average → Predict target word

So in this example:

- Input = average of context one-hots for [the, cat, on, the]
- Target = one-hot of "sat"
- Training updates weights so that embeddings capture word meaning.

Example



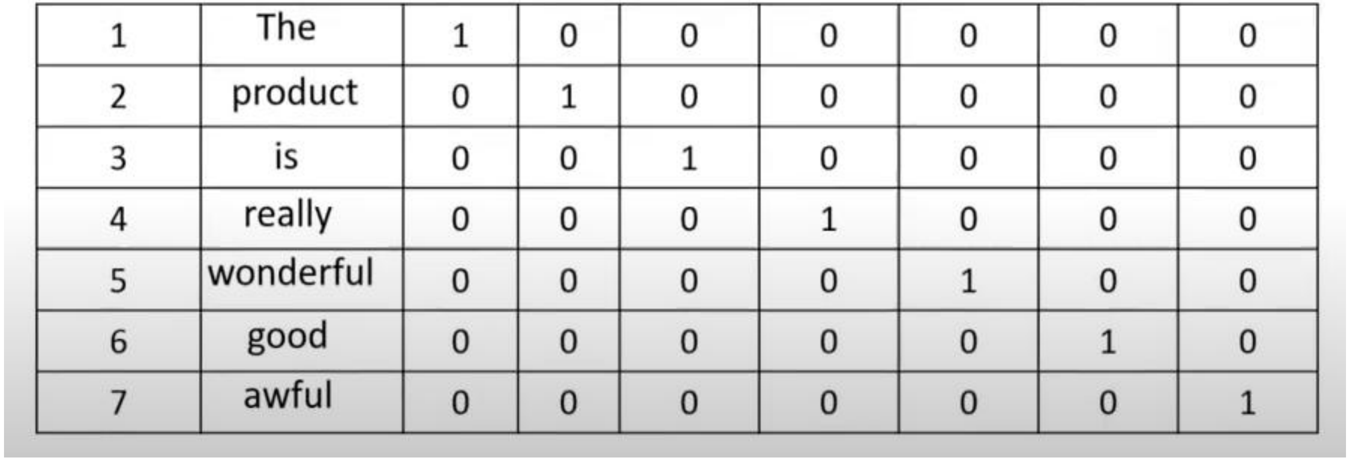
Continuous Bag of Words

Step - 1 The product is really good The product is wonderful The product is awful

Step - 2 Window Size 1

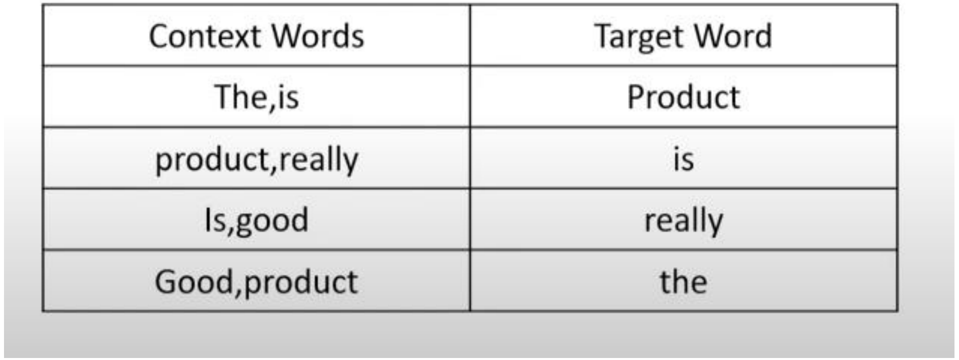
Step - 3 Get one hot encoding for each word

- **One Hot Encoding of Corpus**

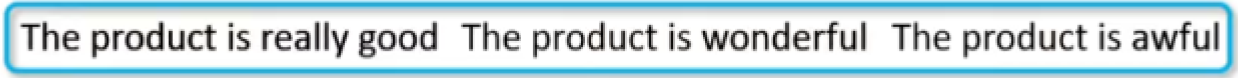


1	The	1	0	0	0	0	0	0
2	product	0	1	0	0	0	0	0
3	is	0	0	1	0	0	0	0
4	really	0	0	0	1	0	0	0
5	wonderful	0	0	0	0	1	0	0
6	good	0	0	0	0	0	1	0
7	awful	0	0	0	0	0	0	1

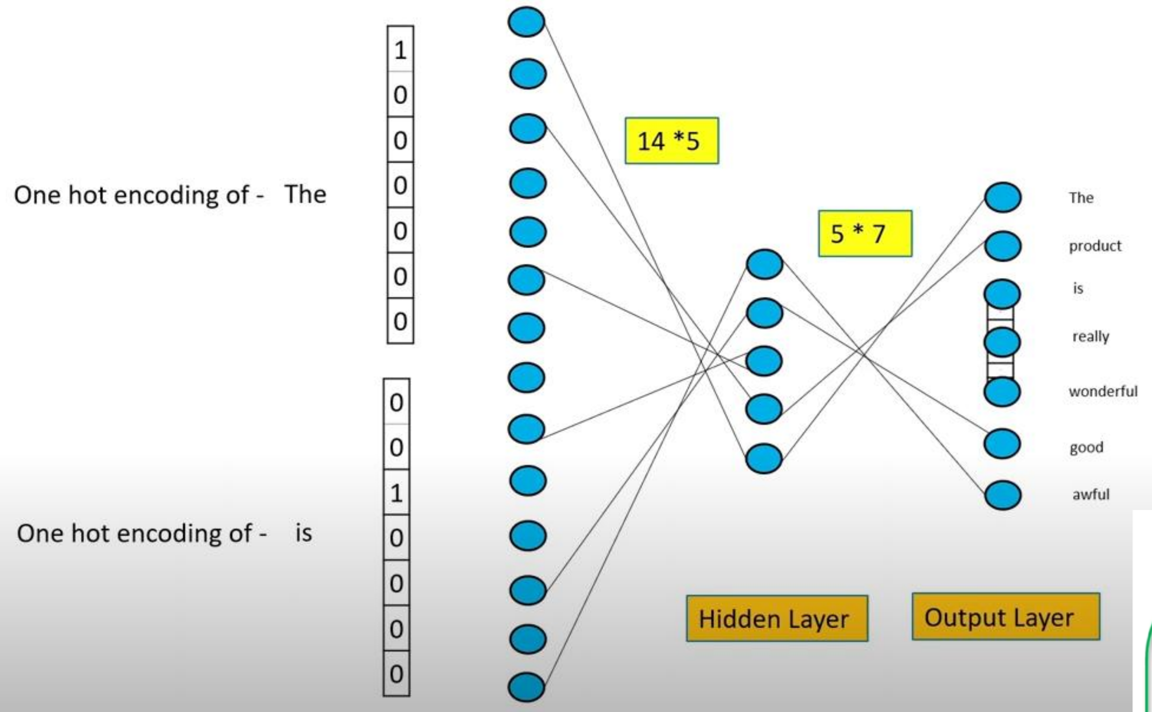
Training Data

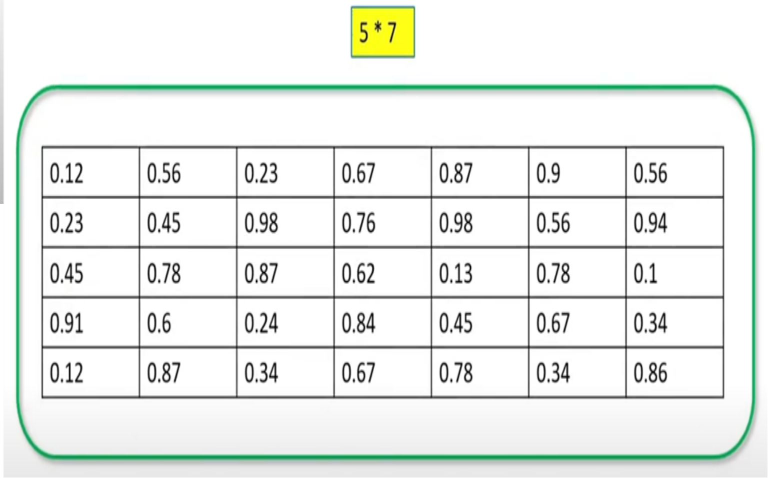


Context Words	Target Word
The,is	Product
product,really	is
Is,good	really
Good,product	the

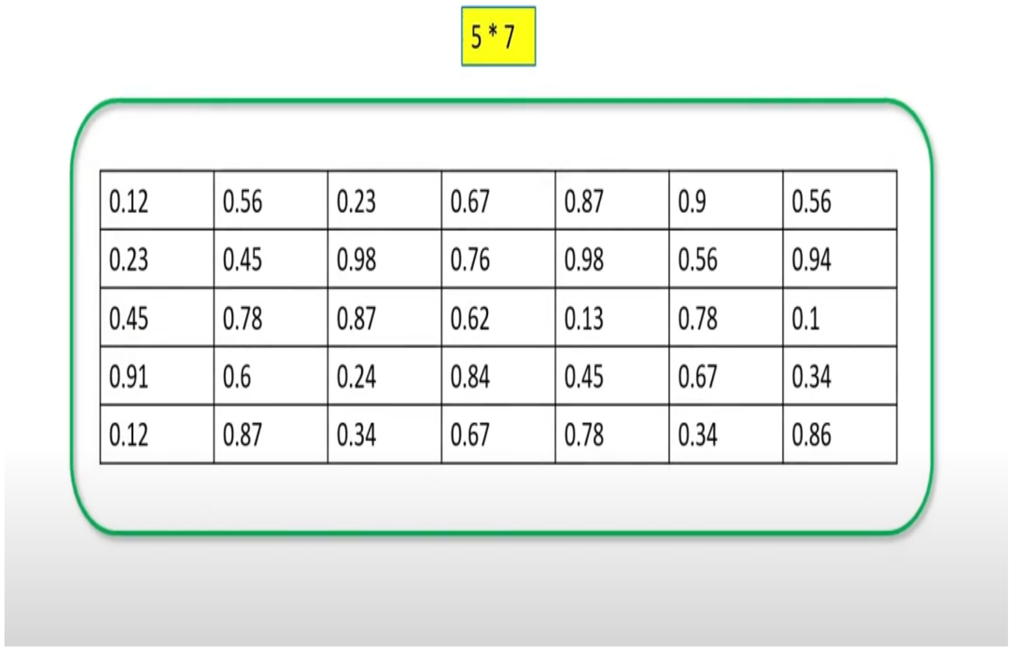


The product is really good The product is wonderful The product is awful

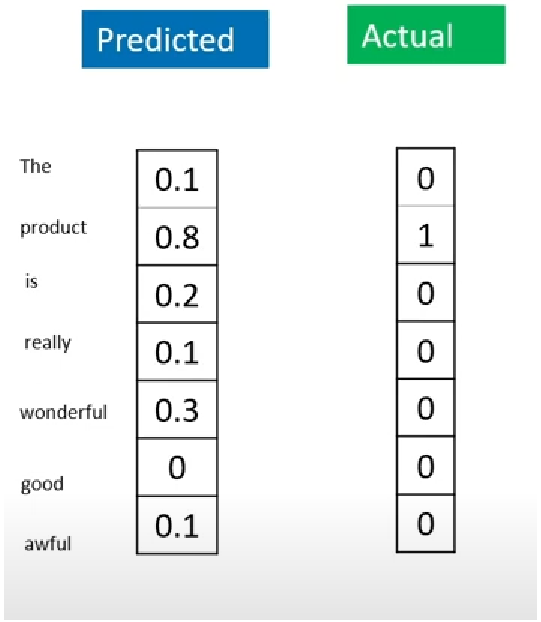




0.12	0.56	0.23	0.67	0.87	0.9	0.56
0.23	0.45	0.98	0.76	0.98	0.56	0.94
0.45	0.78	0.87	0.62	0.13	0.78	0.1
0.91	0.6	0.24	0.84	0.45	0.67	0.34
0.12	0.87	0.34	0.67	0.78	0.34	0.86


5 * 7

0.12	0.56	0.23	0.67	0.87	0.9	0.56
0.23	0.45	0.98	0.76	0.98	0.56	0.94
0.45	0.78	0.87	0.62	0.13	0.78	0.1
0.91	0.6	0.24	0.84	0.45	0.67	0.34
0.12	0.87	0.34	0.67	0.78	0.34	0.86



	Predicted	Actual
The	0.1	0
product	0.8	1
is	0.2	0
really	0.1	0
wonderful	0.3	0
good	0	0
awful	0.1	0

- Context words → Average → Predict target word

So in this example:

- Input = average of context one-hots for
- Target = one-hot of target
- Training updates weights so that embeddings capture word meaning

- **Advantages of CBOW:**

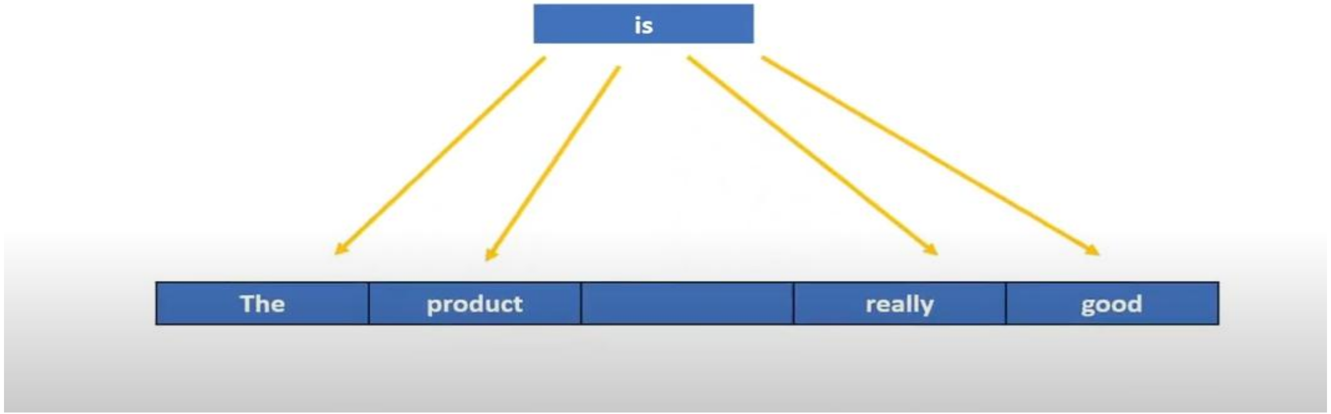
1. **Efficient**: CBOW tends to be faster and more efficient to train than the Skip-gram model, especially on larger datasets.
2. **Performance with Frequent Words**: CBOW is particularly effective when working with frequent words since it averages the surrounding context.

- **Limitations:**

1. **Word Sense Ambiguity**: Since CBOW averages context word vectors, it might struggle with words that have multiple meanings (polysemy) because it generates one fixed embedding per word.
2. **Information Loss**: Averaging context words may result in a loss of fine-grained information, especially when dealing with smaller datasets or less frequent words.

Skip-gram

- While **CBOW (Continuous Bag of Words)** predicts a target word from surrounding context words, the **Skip-gram** model does the opposite: it predicts the context words based on a given target word.



Example

- For the sentence:
- "The cat sat on the mat."
- If the target word is **"sat"** and we use a window size of 2, the goal is to predict the surrounding words:
- Target word: "sat"
- Context words: ["The", "cat", "on", "the"]
- The Skip-gram model takes **"sat"** as the input and tries to predict these context words.

- **Context and Target Word Pairs:**
- For the target word "sat", the Skip-gram model will try to predict the surrounding words within the given window.
- The context and target word pairs are created as follows:
- Target word: **"sat"**
- Context words: **["The", "cat", "on", "the"]**
- So, the model would learn from these pairs:
- ("sat", "The")
- ("sat", "cat")
- ("sat", "on")
- ("sat", "the")
- These word pairs are what the model will use for training.

- **Word Representation**
- Each word in the vocabulary is represented as a vector (an embedding) in a high-dimensional space.
- Initially, these vectors are random, but they will be updated during training.
- For simplicity, let's assume our vocabulary contains five unique words: ["The", "cat", "sat", "on", "mat"]

One-hot encoded vectors:

- the → [1, 0, 0, 0, 0]
- cat → [0, 1, 0, 0, 0]
- sat → [0, 0, 1, 0, 0]
- on → [0, 0, 0, 1, 0]
- mat → [0, 0, 0, 0, 1]

- Input = **one-hot for "sat"** = [0, 0, 1, 0, 0]
- Hidden layer = embedding vector (learned)
- Output = probability distribution over vocabulary (softmax)
- Loss = high when predicted word ≠ context word → backprop adjusts weights

So model learns embeddings such that "sat" is close to words that usually appear near it (e.g., “cat”, “on”).

- **Skip-gram Model Architecture**

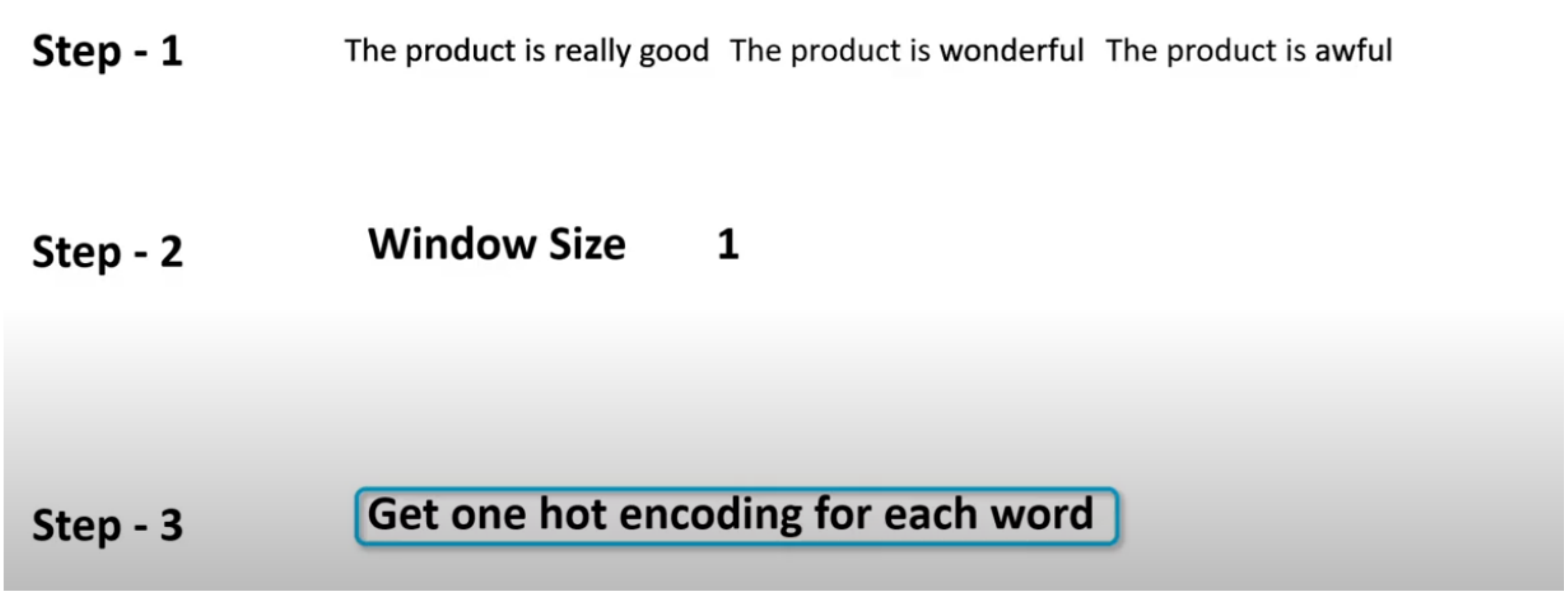
The Skip-gram model has a very simple architecture:

1. **Input Layer**: The target word (e.g., "sat").
2. **Embedding Layer**: The input word is transformed into its vector embedding (e.g., "sat" → [0, 0, 1, 0, 0]).
3. **Prediction Layer**: The model predicts which words (context words) are likely to be near the target word by computing the similarity between the target word's embedding and the context words' embeddings.

- During training, for each input word **"sat"**, the model learns to predict the words in its context (i.e., "The", "cat", "on", "the").

- **Training**
- The model is trained using the pairs **("sat", "The")**, **("sat", "cat")**, **("sat", "on")**, and **("sat", "the")**. The idea is to maximize the probability that the model will correctly predict these context words given the target word "sat".

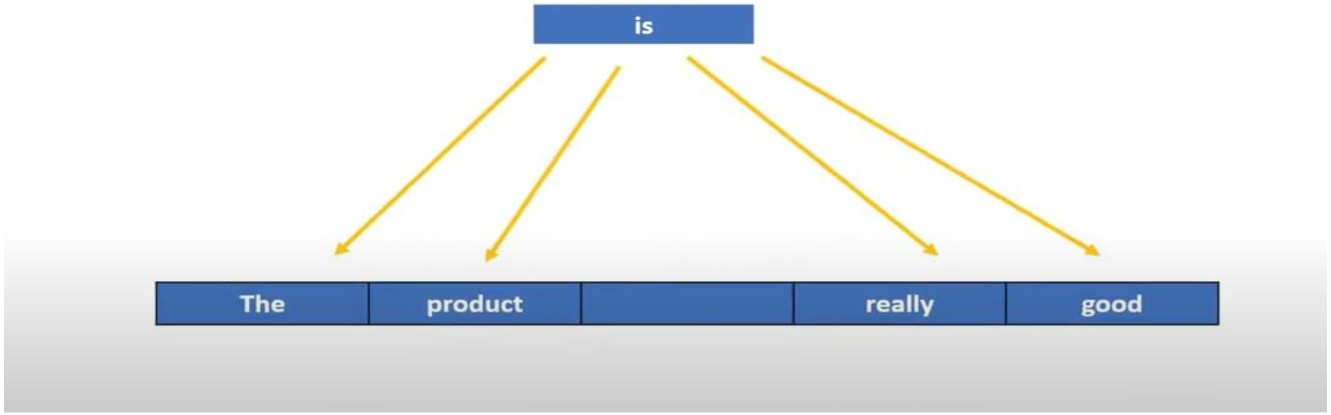
Prerequisites



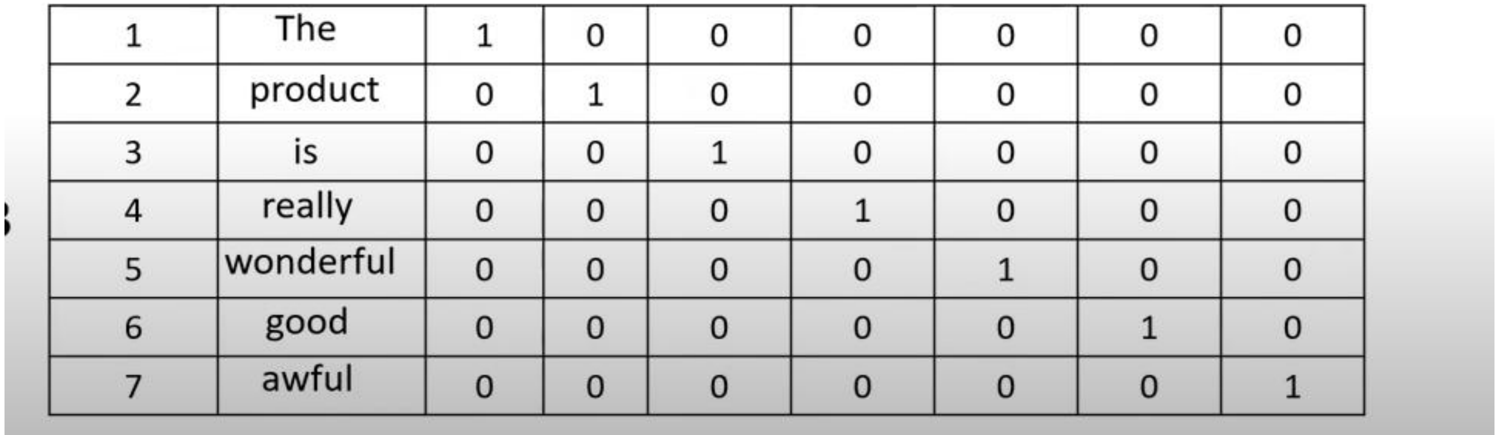
Step - 1 The product is really good The product is wonderful The product is awful

Step - 2 **Window Size** **1**

Step - 3 **Get one hot encoding for each word**

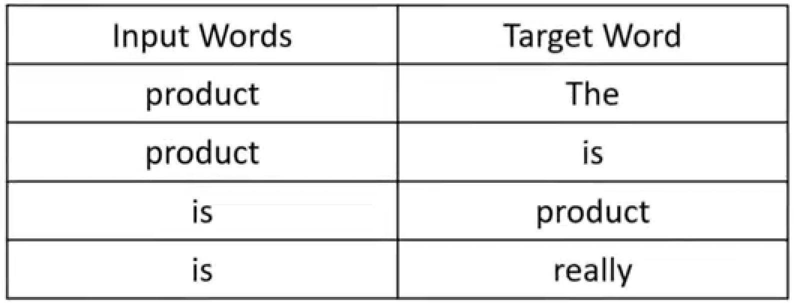


One Hot Encoding



1	The	1	0	0	0	0	0	0
2	product	0	1	0	0	0	0	0
3	is	0	0	1	0	0	0	0
4	really	0	0	0	1	0	0	0
5	wonderful	0	0	0	0	1	0	0
6	good	0	0	0	0	0	1	0
7	awful	0	0	0	0	0	0	1

- Training Words



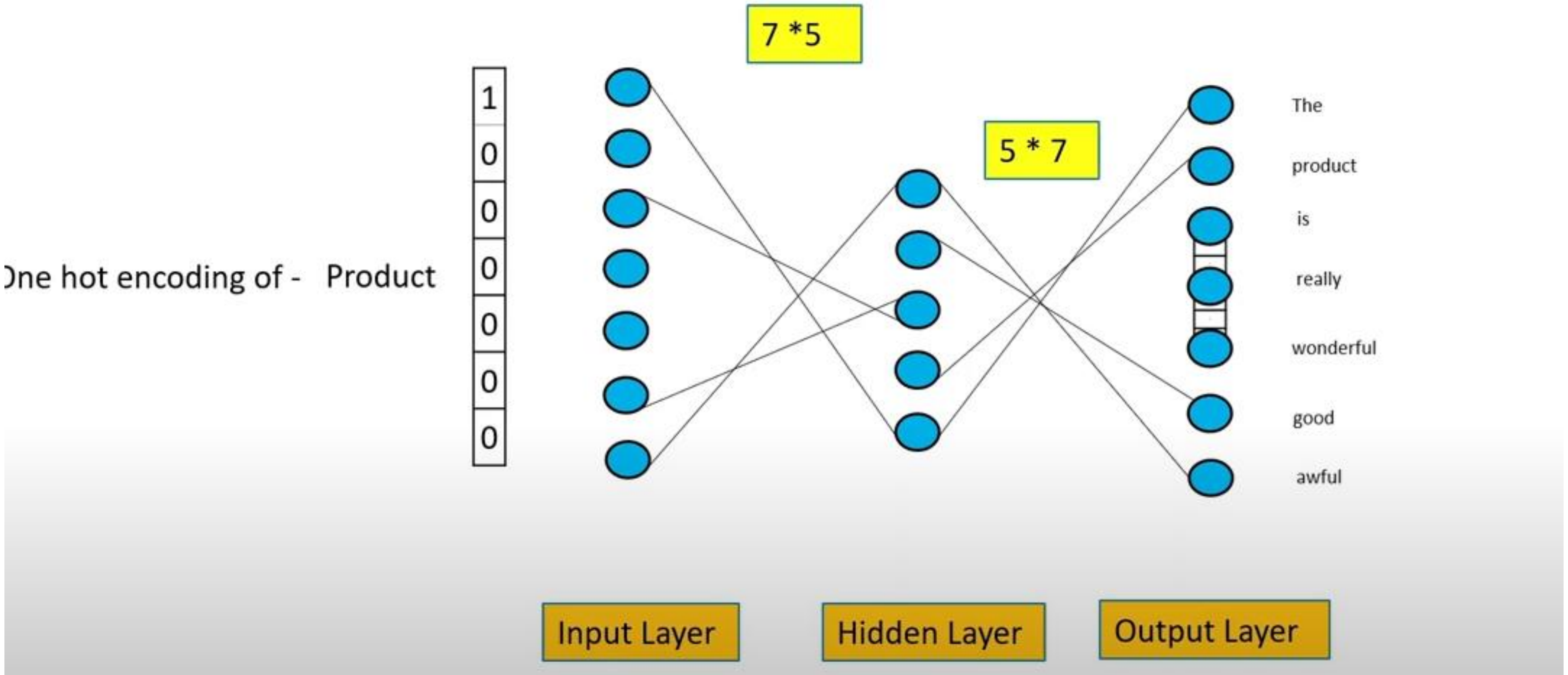
Input Words	Target Word
product	The
product	is
is	product
is	really

One Hot Encoding

1	The	1	0	0	0	0	0	0
2	product	0	1	0	0	0	0	0
3	is	0	0	1	0	0	0	0
4	really	0	0	0	1	0	0	0
5	wonderful	0	0	0	0	1	0	0
6	good	0	0	0	0	0	1	0
7	awful	0	0	0	0	0	0	1

Training Data

Input Words	Target Word
product	The
product	is
is	product
is	really


7 *5
5 * 7
ne hot encoding of - Product
1
0
0
0
0
0
0
The
product
is
really
wonderful
good
awful
Input Layer
Hidden Layer
Output Layer

	Predicted	Actual
The	0.9	1
product	0.2	0
is	0.2	0
really	0.1	0
wonderful	0.3	0
good	0	0
awful	0.1	0

5 * 7

0.12	0.56	0.23	0.67	0.87	0.9	0.56
0.23	0.45	0.98	0.76	0.98	0.56	0.94
0.45	0.78	0.87	0.62	0.13	0.78	0.1
0.91	0.6	0.24	0.84	0.45	0.67	0.34
0.12	0.87	0.34	0.67	0.78	0.34	0.86