**Bag of Words** – sentences ko literal bag of words ki tarah samjhta tha ( Yani they don’t have any meanings )

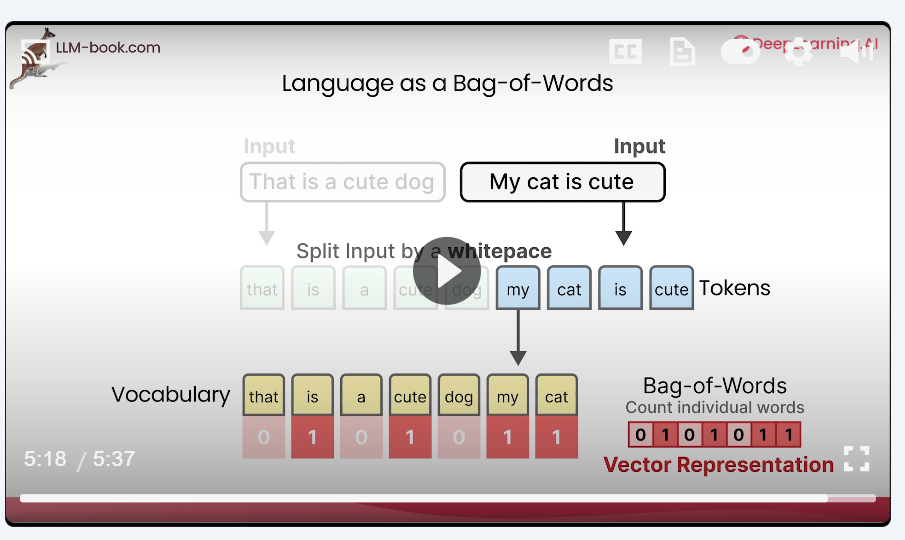
**Word2Vec** could capture meaning of words in vector embeddings through neural networks

**Bag of Words**

**Tokenization** is the process of converting a sentence into smaller chunks called **tokens**

**Vocabulary**  is kind of a list containing unique elements from an input ( ex attached )

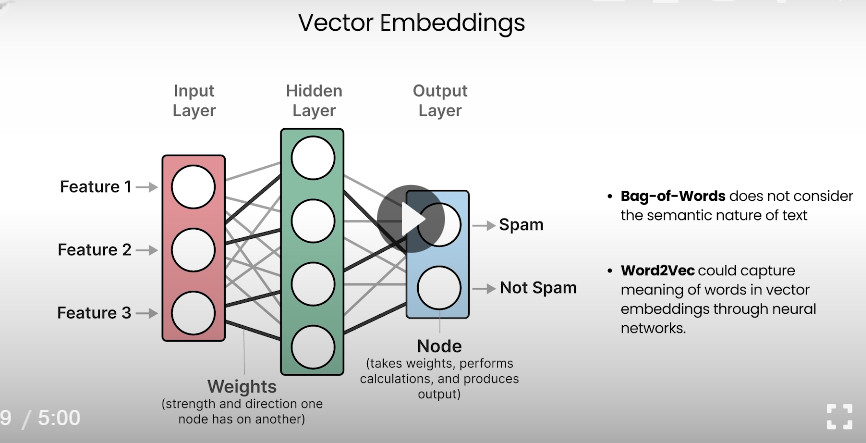
Vocabulary size is the size of this vocabulary



Ab is vector representation ko normalize kardete hain ( to get the vector embeddings ) so that the range stay in between 0-1 ( reason is case mein to 0,1 lekin ye 0,1 hi rahe ye zaruri to ni isliye we normalize)

So basically Bag of words count each word in a sentence and create the above vector

**Word2Vec**



Process

* Imagine you're teaching a computer what words mean by showing it lots of sentences. Word2Vec learns by looking at **which words appear near each other**.
* There are two main ways it does this:

**1. CBOW (Continuous Bag of Words)**

* It looks at the **context words** (words around a missing word) and tries to **predict the missing word**.

Example:  
Sentence: “The cat sat on the \_\_\_.”  
Context: “The cat sat on the” → Predict: “mat”

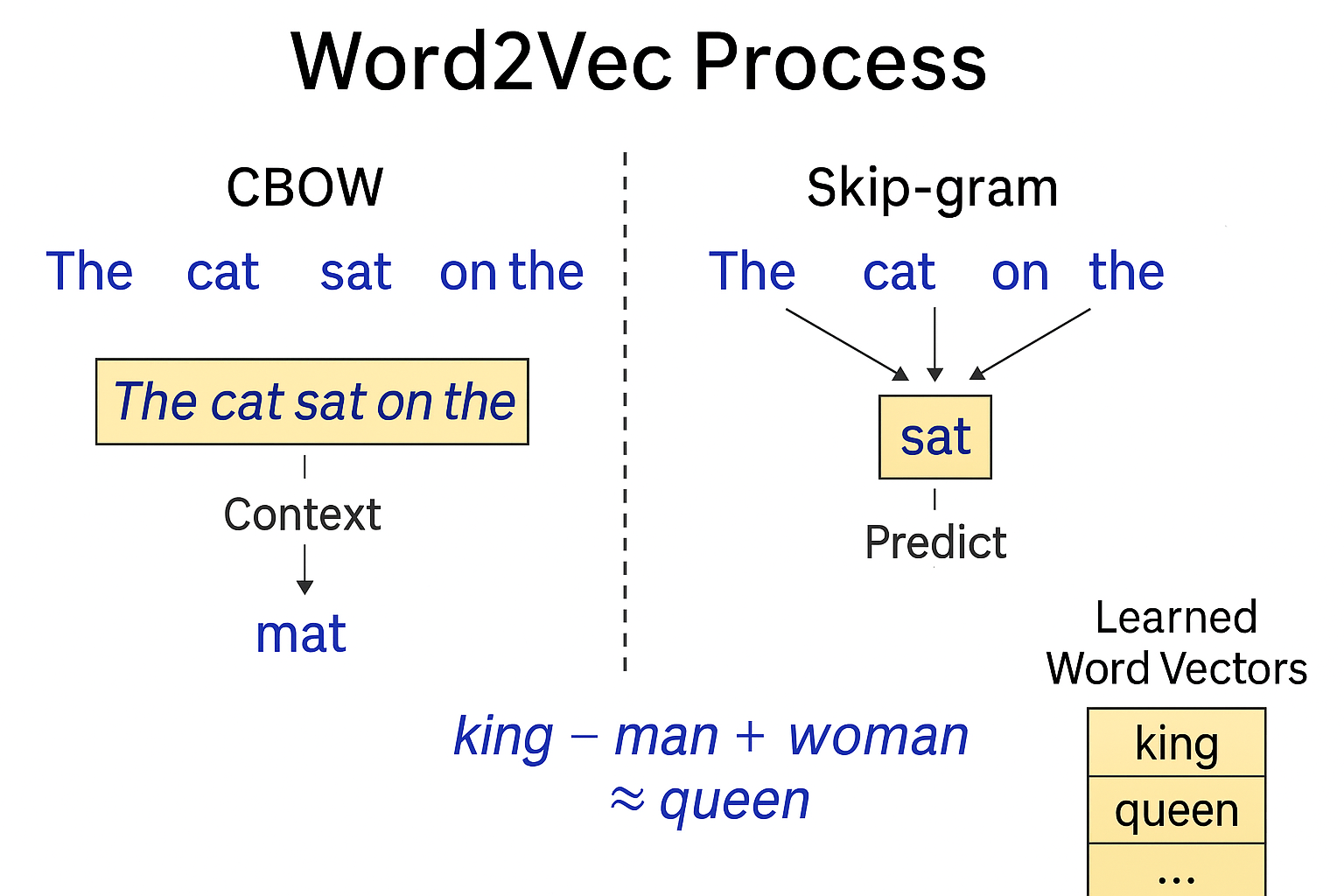
**2. Skip-Gram**

It does the opposite: it looks at a **word** and tries to **predict the surrounding words**.

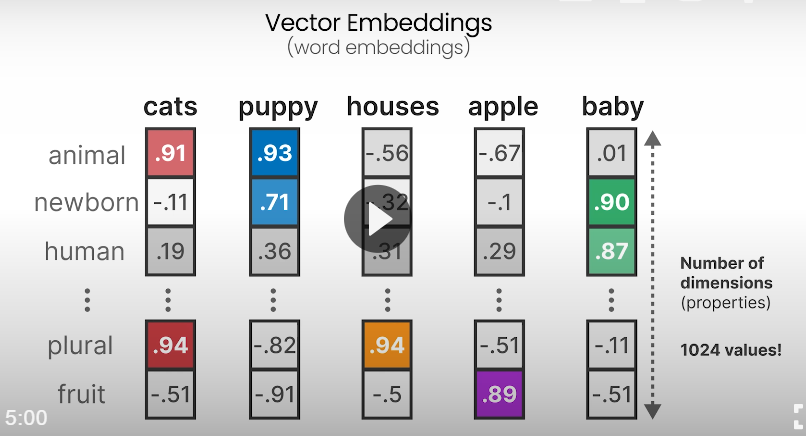
* Example:  
  Word: “sat” → Predict: “The”, “cat”, “on”, “the”

**🔢 What Happens Behind the Scenes?**

* Each word starts as a random vector (just a list of numbers).
* As the model trains, it adjusts these numbers so that **similar words end up with similar vectors**.
* For example, “king” and “queen” will have vectors that are close together.

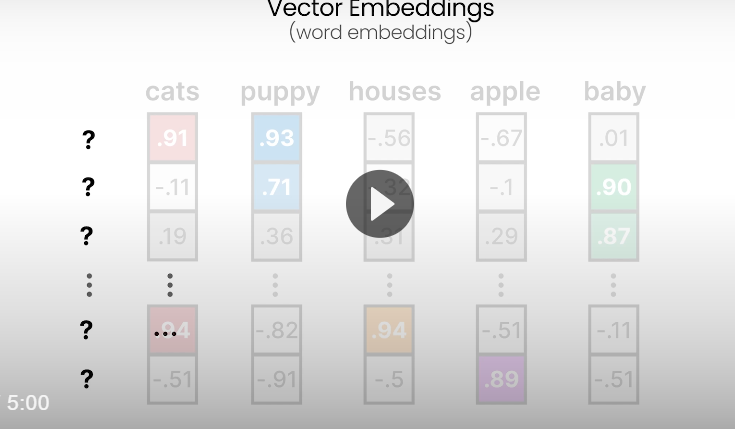


It generates vector embeddings like this

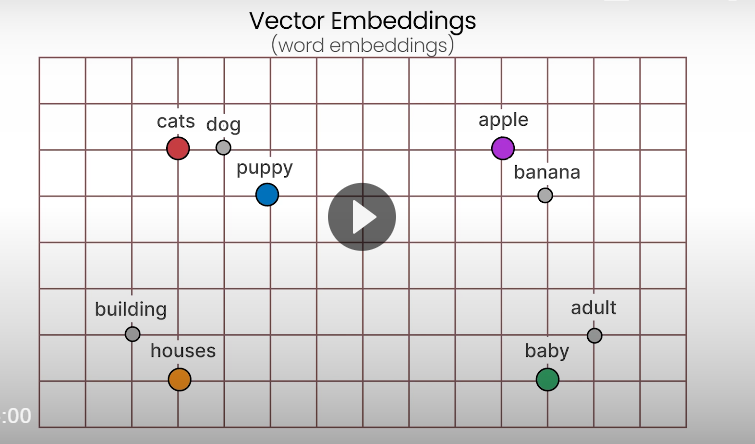


Like cats will score more in animal, and plural and less on the others and similary for the others

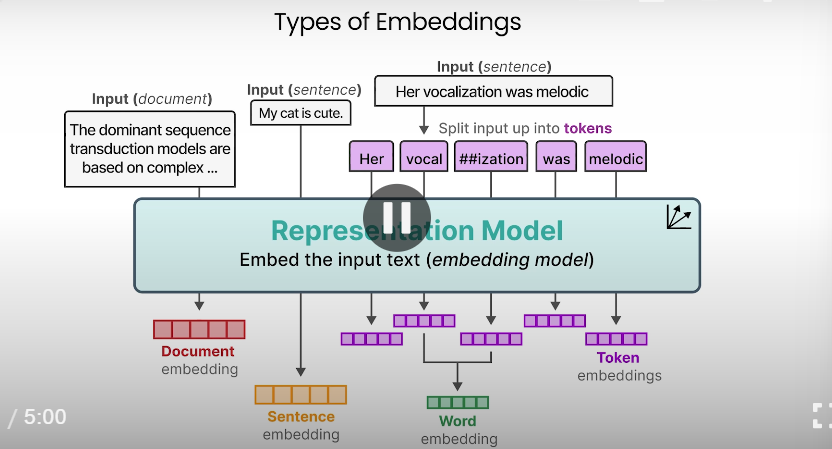
But in reality we don’t know which value corresponds to which , the model learns it using complex mathematical calculations



So by this vector representation it will group similar items together



**Types of Embedding**

****

**Encoding and Decoding Context with Attention**

**🤖 Transformers Made Simple**

**💡 What is it?**

A **Transformer** is a smart model that reads and understands text **all at once**, not word-by-word like older models.

**🧠 How does it work?**

Imagine you're reading a sentence like:

“The cat sat on the mat.”

Instead of reading word-by-word, the Transformer looks at **all the words together** and figures out which ones are important for each other.

**🔍 Key Idea: Attention**

It’s like asking:

“Which words should I pay attention to when understanding this word?”

For example:

* To understand “sat,” it might look at “cat” and “mat.”

**🧱 What’s inside?**

Just remember these 3 parts:

1. **Input**: Words turned into numbers.
2. **Attention**: Figures out which words matter most.
3. **Output**: Gives meaning or generates new text.

**🧙‍♂️ Why is it magical?**

* It understands **context** really well.
* It can **translate**, **summarize**, **chat**, and even **write poems**!
* It powers models like **ChatGPT**, **BERT**, and **Google Translate**.

**A bit detailed overview**

**Tokenize → Embed → Add Position → Pay Attention → Refine → Output**

**Step 1: Tokenization**

* Break the sentence into smaller pieces (words or subwords).
* Example:  
  “The cat sat” → ["The", "cat", "sat"]

**Step 2: Embedding**

* Turn each token into a list of numbers (a vector) that represents its meaning.
* These numbers help the model understand the word.

**Step 3: Positional Encoding**

* Since Transformers look at all words at once, they need a way to know the **order** of words.
* Positional encoding adds info about where each word is in the sentence.

**Step 4: Transformer Block**

Each block contains:

Multiple blocks are stacked to make the model deeper and smarter.

**a: Self-Attention**

* Each word looks at **all other words** in the sentence to decide which ones are important.
* Example: “sat” might pay more attention to “cat” than “The”.

**b: Multi-Head Attention**

* The model does self-attention multiple times in parallel.
* Each “head” learns different relationships between words.

**c: Feed-Forward Network**

* After attention, each word’s info is passed through a small neural network to refine it.

**Step 5: Output**

* Depending on the task, the model might:
  + Predict the next word (like GPT)
  + Understand the sentence (like BERT)
  + Translate it (like T5)

**Step 1–4: Encoder Side (Understanding the Input)**

1. **Tokenization**  
   → Break input sentence into tokens.
2. **Embedding**  
   → Convert tokens into vectors.
3. **Positional Encoding**  
   → Add position info to each word.
4. **Transformer Block (Encoder)**  
   → Uses **self-attention** to understand relationships between words in the input.

📌 **Used in tasks like**:

* Text classification
* Sentiment analysis
* Question answering
* BERT

**✅ Step 5–7: Decoder Side (Generating Output)**

1. **Decoder Input**  
   → Starts with a special token (like <start>), then uses previously generated words.
2. **Transformer Block (Decoder)**  
   → Uses:
   * **Self-attention** (on decoder input)
   * **Cross-attention** (to look at encoder output)
   * **Feed-forward network**
3. **Output**  
   → Predicts the next word or generates full sentences.

📌 **Used in tasks like**:

* Text generation
* Translation
* Summarization
* GPT, T5