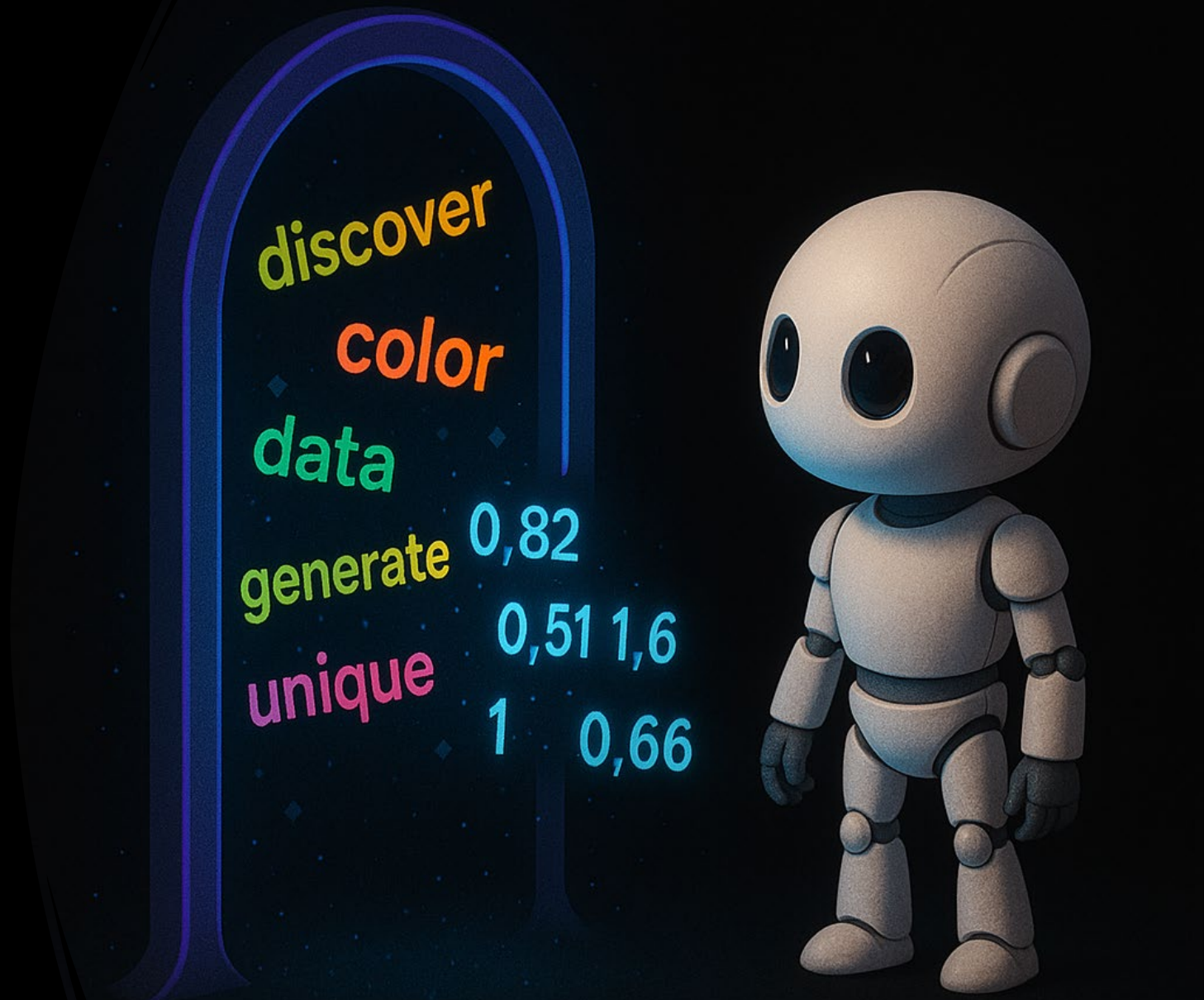
Text Representation From Words to Numbers Part 01

ITAI 2373 Mod 04



By The End of Part 01, You Will Be Able To:

Explain	Explain why text must be converted to numerical representations
Implement and evaluate	Implement and evaluate Bag of Words models
Apply	Apply TF-IDF weighting to improve text representation Understand
Use	N-grams and Phrase detection
Understand	Advanced vectorization methods

Why Convert Text to Numbers

- Computers process numbers, not words
- Machine learning algorithms require numerical inputs
- Text is:
  - Categorical (discrete)
  - Variable-length
  - Hierarchical (letters → words → sentences → documents)
  - Contextual (meaning depends on surrounding words)
- **Our Goal:** Find representations that preserve relevant information while being algorithm-friendly

History of Text Representation

1950s-1960s
Dictionary--based approaches

1970s-1980s
Boolean models

1990s
Vector Space Model, TF-IDF

2000s
Latent Semantic Analysis (LSA)

2010s
Word embeddings revolution
(Word2Vec, GloVe)

2018-Present
Contextual embeddings
(BERT, GPT)

Sparse Representations – One-Hot Encoding

Sparse

aardvark	1	0	0	…	0	0	0
red	0	0	…	1	…	0	0
cat	0	0	…	1	…	0	0
duvet	0	0	…	1	…	0	0
zombie	0	0	0	…	0	0	1

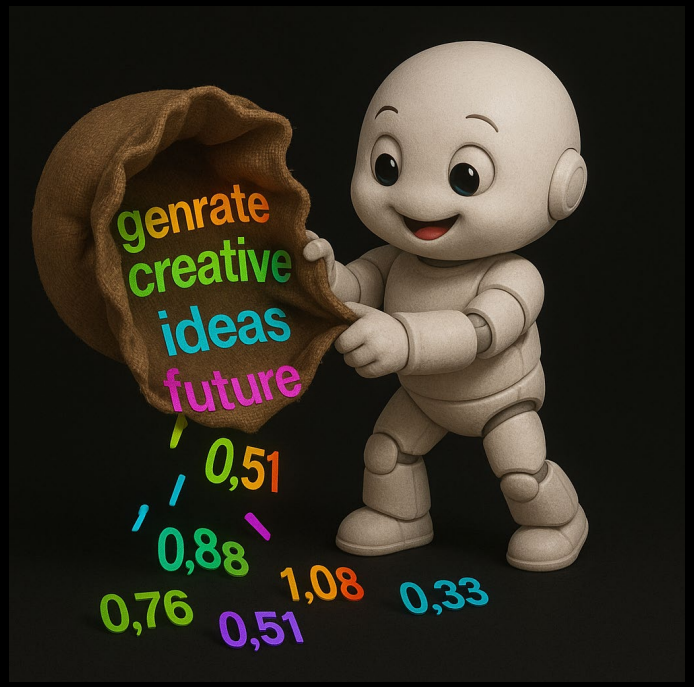
- Most of the elements are zero.
- Words are represented by a binary value based on whether the word appears.

- One-hot encoding:
  - One-hot encoding: Each word gets a unique vector with a single 1.
  - Example: "cat" = [1, 0, 0], "dog" = [0, 1, 0], "bird" = [0, 0, 1].
  - Limitations: No meaning, high memory use.

Bag of Words (BOW) Model

The Simplest Approach

- **Concept:** Document = Unordered collection of word counts
- **Process:**
  - Build vocabulary from all unique words
  - Create fixed-length vector for each document
  - Each position represents a word count
- **Example:**
  - Doc: "The cat sat on the mat"
  - Vocab: {"the": 0, "cat": 1, "sat": 2, "on": 3, "mat": 4}
  - Vector: [2, 1, 1, 1, 1]



Bag-of-words example

Sentence	a	cat	dog	is	it	my	not	old	wolf
It is a dog.	1	0	1	1	1	0	0	0	0
my cat is old.	0	1	0	1	0	1	0	1	0
It is not a dog, it is a wolf.	2	0	1	2	2	0	1	0	1

Result: Each document becomes a vector of word counts

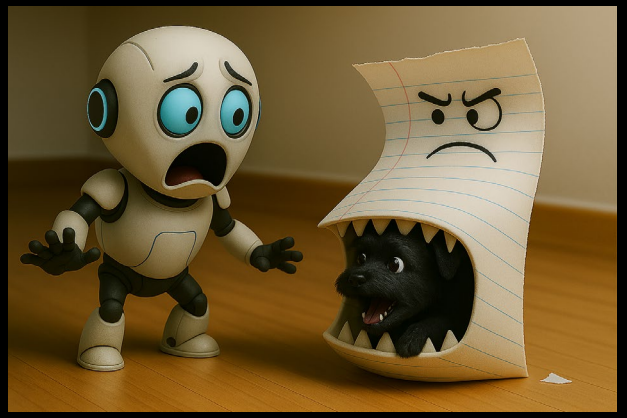
Bag of Word Implementation

- Implementing BOW in Python
- **CountVectorizer:** **C**onverts a collection of text documents to a matrix of token counts
- Challenges:
  - High dimensionality
  - Sparsity
  - Equal weighting of all words

```python
from sklearn.feature_extraction.text import CountVectorizer

docs = ["The cat sat on the mat",
        "The dog lay on the floor"]

vectorizer = CountVectorizer()
X = vectorizer.fit_transform(docs)
print(X.toarray())
# [[2 1 0 0 0 1 1 1]
#  [1 0 1 1 1 0 0 1]]

print(vectorizer.get_feature_names_out())
# ['the', 'cat', 'dog', 'floor', 'lay', 'mat', 'sat', 'on']
```

Result

BOW Limitations



Consider these two sentences:

1."The dog ate my homework."
2."The homework ate my dog."

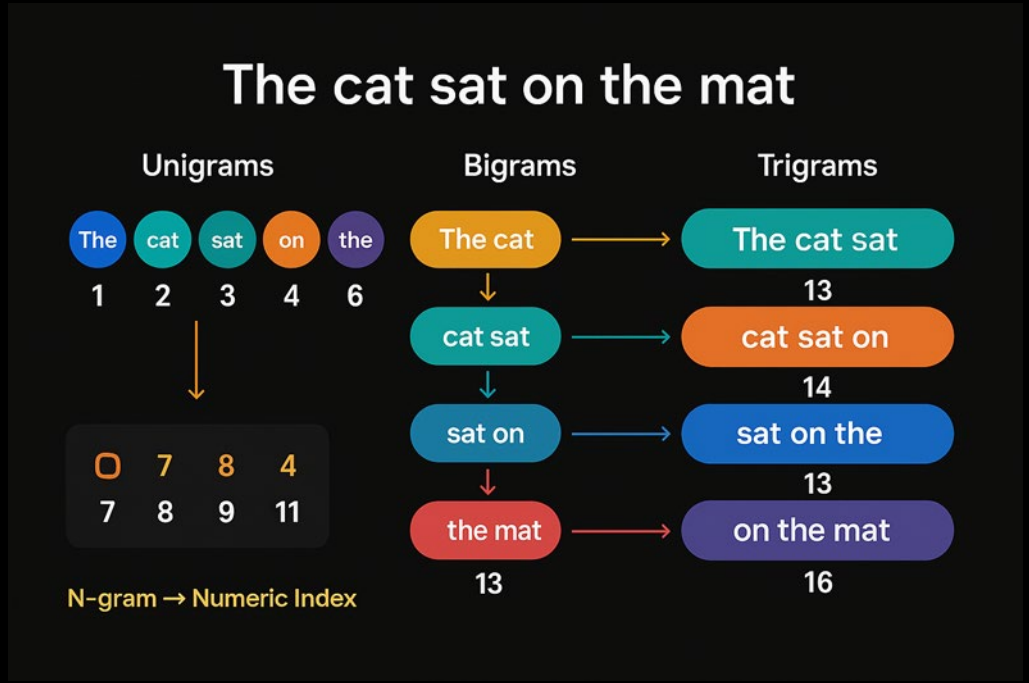
Questions for Discussion:

- What information is lost in BOW representation?
- How might this affect downstream applications?
- Can you think of real-world examples where this would be problematic?
- What modifications might help address these limitations?

N-gram（ part of Tokenization)

Break a given sample of text or speech into a sequence of *n* tokens:

- Only considering single words (unigram)
- Break apart compound words and some proper nouns
- Can include n-grams in term frequencies



Sentence	1-gram (unigram)	2-gram (bigram)	3-gram (trigram)
It is not a dog, it is a wolf	it, is, not, a, dog, it, is, a, wolf	it is, is not, not a, a dog, ….	it is not, is not a, not a dog, ….

Term Frequency (TF) – Weighting Words by Importance

- **Problem:** Not all words are equally important
- **Solution:** Weight words by how distinctive they are
- **Term Frequency (TF):** How often a word appears in a document
  - $tf\,(term, doc) = \frac{\text{number of times the term occurs in the document}}{\text{total number of terms in the document}}$

Sentence	a	cat	dog	is	it	my	not	old	wolf
It is a dog.	0.25	0	0.25	0.25	0.25	0	0	0	0
my cat is old.	0	0.25	0	0.25	0	0.25	0	0.25	0
It is not a dog, it is a wolf.	0.22	0	0.11	0.22	0.22	0	0.11	0	0.11

Inverse document frequency (IDF)

- How unique a word is across documents
- Increases the weights for words that are specific to a document that don't frequently occur in other documents

$$idf(term) = log\left(\frac{n_{\text{documents}}}{n_{\text{documents containing the term}} + 1}\right) + 1$$

Example: $idf(\text{"cat"}) = 1.18$

Term	IDF
a	log(3/3)+1=1
cat	log(3/2)+1=1.18
dog	log(3/3)+1=1
is	log(3/4)+1=0.87
it	log(3/3)+1=1
my	log(3/2)+1=1.18
not	log(3/2)+1=1.18
old	log(3/2)+1=1.18
wolf	log(3/2)+1=1.18

TF-IDF Weighting

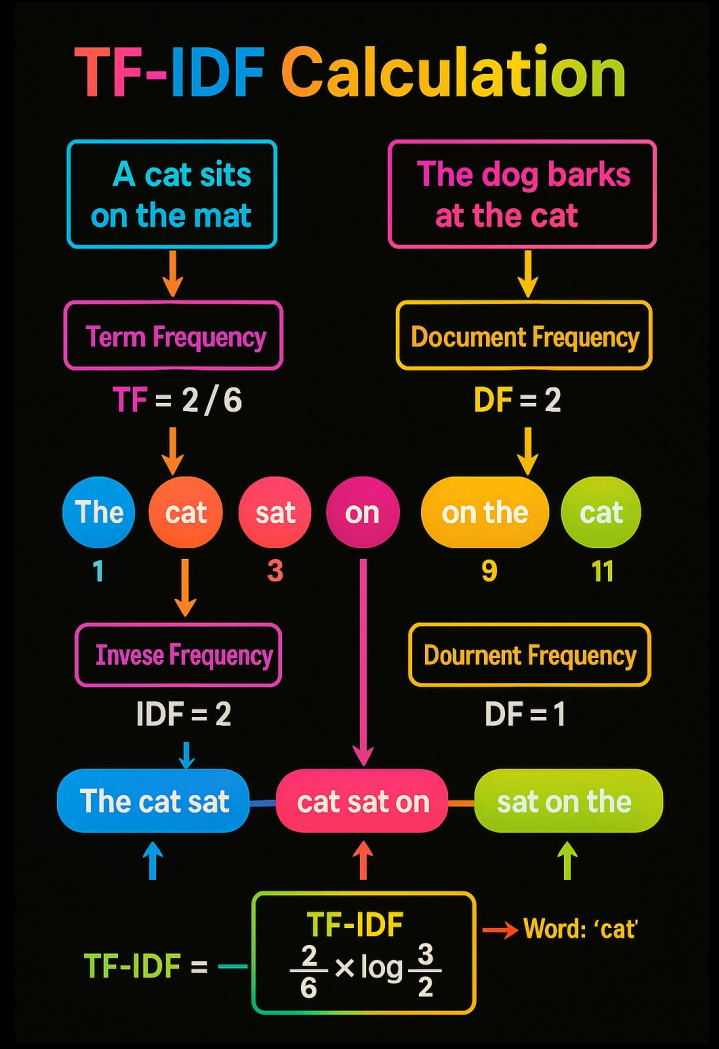
TF-IDF = Term Frequency × Inverse Document Frequency

$$tf_{idf}(term, doc) = tf(term, doc) * idf(term)$$

High TF-IDF scores indicate:

- Words that appear frequently in a document (high TF)
- But rarely in other documents (high IDF)
- These words are likely most distinctive of the document's content

Sentence	a	cat	dog	is	it	my	not	old	wolf
It is a dog.	0.25	0	0.25	0.22	0.25	0	0	0	0
my cat is old.	0	0.3	0	0.22	0	0.3	0	0.3	0
It is not a dog, it is a wolf.	0.22	0	0.11	0.19	0.22	0	0.13	0	0.13



Implementing TF-IDF

Applications:

- Document classification
- Information retrieval
- Document similarity
- Feature extraction

```
from sklearn.feature_extraction.text import TfidfVectorizer

docs = ["The cat sat on the mat",
        "The dog lay on the floor",
        "The cat and dog played on the floor"]

vectorizer = TfidfVectorizer()
X = vectorizer.fit_transform(docs)
print(X.toarray())
# Words like "cat", "dog" get higher weights where distinctive
# Common words like "the", "on" get lower weights
```

Hands-on TF-IDF Example

Let's Calculate TF-IDF Together!

① The cat sat on the mat
② The dog lay on the floor

Step 1 – Term Frequency (TF)

- Doc 1: count(cat) total words = 6 → TF = 1 ÷ 0,17
- Doc 2: count(cat) total words = 6 → TF = 0 ÷ 0,00

Step 2 – Inverse Document Frequency (IDF)

- N = 2 docs, df(cat) = 1 → IDF = ln(2 ÷ 1) = ln 2 = ln 2 = 1 = 0,693

Step 3 – TF-IDF

- Doc 1: TF 0,17 × IDF 0.693 = 0.693
- Doc 2: TF 0.00 × IDF 0.693 = 0,115

Advanced Vectorization: Hashing Trick

Handling Large Vocabularies Efficiently

- **Challenge:** Vocabulary size can be enormous in real applications
- **Solution:** Hash function to map words to fixed vector dimensions
- **Implementation:**

```
from sklearn.feature_extraction.text import HashingVectorizer
vectorizer = HashingVectorizer(n_features=1000)
```

- **Benefits:** Fixed memory usage, no vocabulary storage, streaming-friendly
- **Drawbacks:** Hash Collisions, non-interpretable features, no IDF weighting

Advanced Vectorization: Feature Selection

Reducing Dimensionality Through Selection

- **Chi-squared (χ^2) Test:** Measures dependence between features and classes
- **Mutual Information:** Quantifies information shared between features and classes
- **Implementation:**

```
from sklearn.feature_selection import SelectKBest, chi2
selector = SelectKBest(chi2, k=1000)
X_selected = selector.fit_transform(X, y)
```

- **Benefits:** Reduced dimensionality, improved model performance, faster training
- **Applications:** Text classification, topic modeling

Latent Semantic Analysis (LSA)

Uncovering Hidden Semantic Structure

- **Technique:** Applies SVD (Singular Value Decomposition) to TF-IDF matrix
- **Process:** Decomposes document-term matrix into lower-dimensional space
- **Implementation:**

```
from sklearn.decomposition import TruncatedSVD
from sklearn.pipeline import Pipeline
from sklearn.feature_extraction.text import TfidfVectorizer

pipeline = Pipeline([
    ('tfidf', TfidfVectorizer()),
    ('svd', TruncatedSVD(n_components=100))
])
```

- **Benefits:** Handles synonymy, reduces dimensionality, captures semantic relationships
- **Limitations:** Still relies on bag-of-words, limited contextual understanding

Dense vs. Sparse Representations

- **Word Embeddings: Dense Vector Representations**
- Dense vectors: Compact, meaningful representations.
- Words with similar meanings are close in vector space.
- Example: "king" - "man" + "woman" ≈ "queen."

Dense

	animal	fluffy	dangerous	spooky
aardvark	0.97	0.03	0.15	0.04
red	0.07	0.01	0.20	0.10
cat	0.98	0.98	0.45	0.35
duvet	0.01	0.84	0.12	0.02
zombie	0.74	0.05	0.98	0.93

- Most of the elements are nonzero.
- Similar words are represented by vectors, and dissimilar words are represented by dissimilar vectors.

The Distributional Hypothesis

The Distributional Hypothesis:

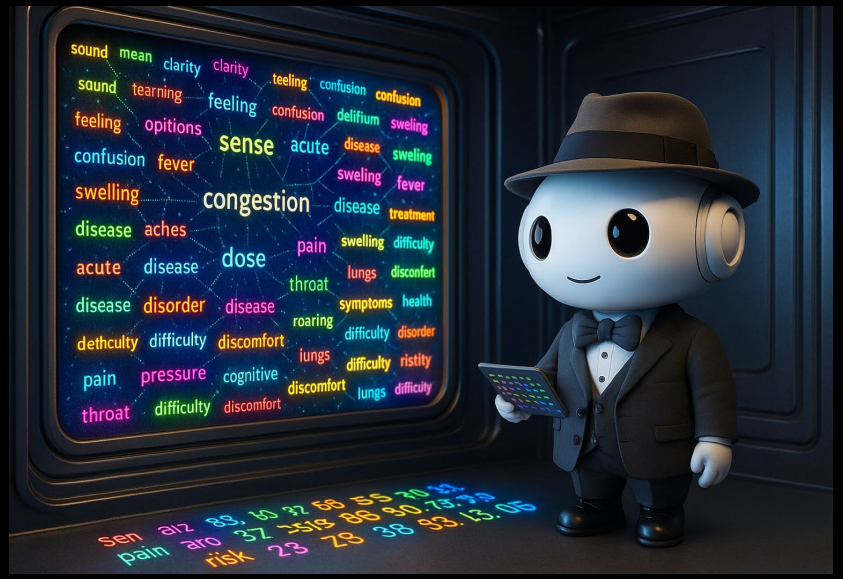
'You shall know a word by the company it keeps’

- Key principle: Words that appear in similar contexts tend to have similar meanings
  - Example: 'dog' and 'cat' often appear near words like 'pet', 'fur', 'veterinarian’
  - 'happy' and 'joyful' often appear near: 'smiled', 'celebration', 'excited'
- This principle underlies most word embedding techniques

Evolution of Word Embeddings

- **Context-Free** Embeddings (Word2Vec, GloVe)
- **Contextualized** Embeddings (ELMo, BERT, GPT)
- **Advanced Generative AI Techniques** (FastText, CLIP, Sentence Transformers)

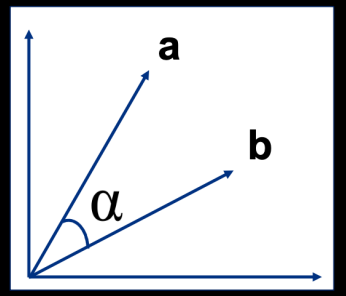
Word Embeddings: The Revolution



AI Def: Word embeddings are like a secret code for computers to understand how words are related by looking at where they live on a special map of meaning.

- **Properties:**
  - Fixed-length vectors (typically 100-300 dimensions)
  - Similar words have similar vectors
  - Semantic relationships preserved in vector space
  - Enable arithmetic operations: king - man + woman ≈ queen
- **Paradigm Shift:** From sparse, discrete representations to dense, continuous ones
- **Applications:** Machine translation, sentiment analysis, document similarity, search engines

Cosine Similarity: Measuring Word Relationships



- Key Idea
  - Direction matters more than size
  - Words = arrows pointing toward meaning
  - Similar words → similar directions
- Simple Analogy
  - Party guests pointing at interests:

👤 → "pets" (quietly) 👤 → "pets" (loudly) 👤 → "technology"

↘ ↘ ↙

Same direction = Similar | Different direction = Different

- Math Explanation
  - Express similarity between different words
  - Angle between two vectors represents similarity
  - For the vectors $a, b \in \mathbb{R}^d$:

$$\text{sim}(a, b) = \frac{a \cdot b}{||a||b||} = \cos(\alpha)$$

  - $\cos(\alpha)$ close to 1 means α close to zero, which means higher similarity
    - Zero between the one-hot vectors of any two different words

Context-Free Word Embeddings

The First Generation of Dense Representations:

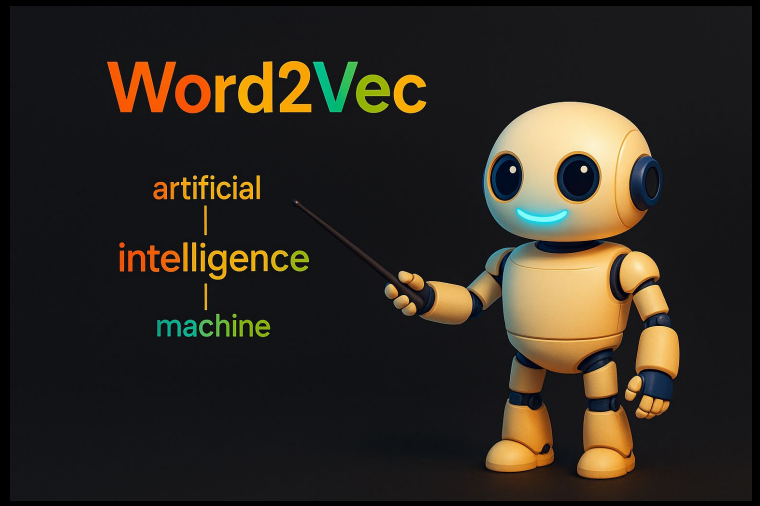
- **Word2Vec (2013)**
  - Uses a **two-layer neural network** to learn word meanings.
  - **Skip-gram:** Predicts **context words** from a center word.
  - **CBOW:** Predicts **center word** from context.

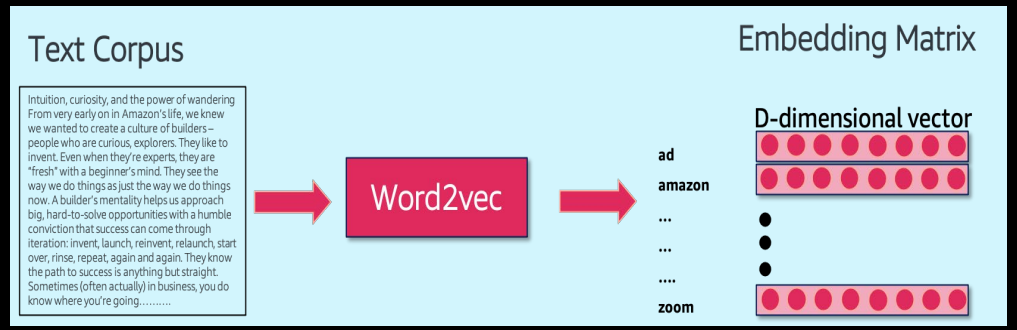
- **GloVe (2014)**
  - Global Vectors for Word Representation
  - Learns from word co-occurrence statistics
  - Combines global and local information

Key Characteristic: Each word gets exactly one vector, regardless of context

What is word2vec?

Core Idea: Learn word representations by predicting context

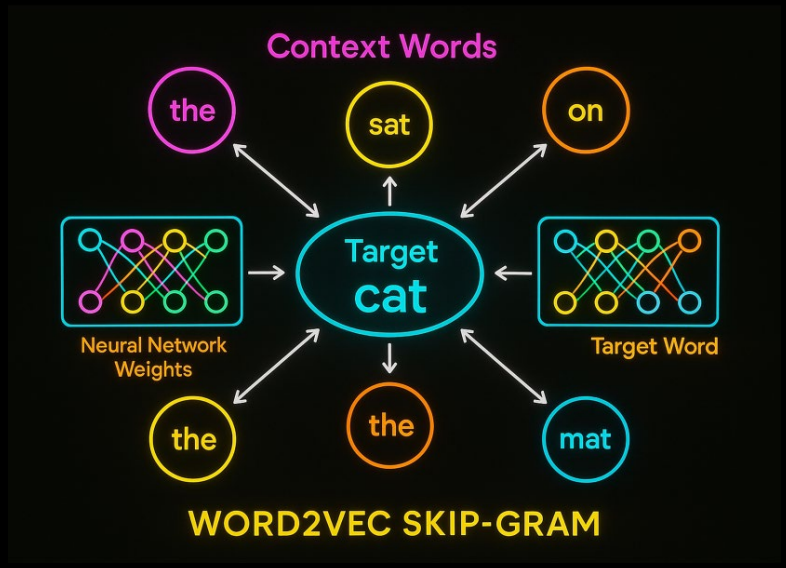


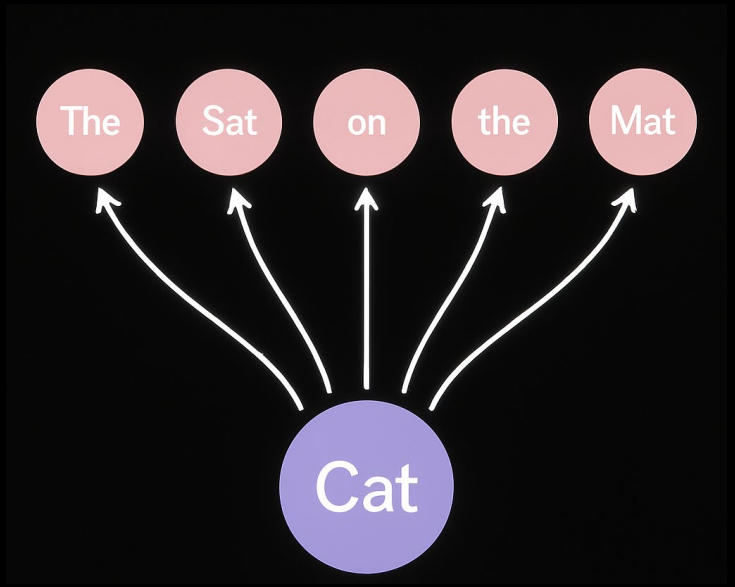


- Turns words into dense vectors (e.g., 300 numbers) that capture meaning.
- Words with similar meanings (like "cat" and "dog") get vectors close together.
- Uses: Finding similar words, analogies (e.g., "king" - "man" + "woman" ≈ "queen").
- Two Models: Skip-gram and CBOW (Continuous Bag of Words).
  - Skip-gram: Predicts context words from center word
  - CBOW: Predicts center word from context words

Word2vec: Skip-gram

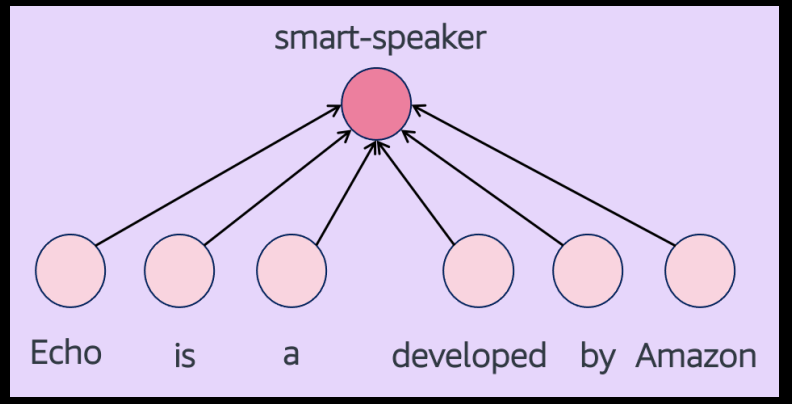
- **Skip-gram:** Predicts context words from a center word.
- **Example:** “The Cat sat on the mat ."
  - Center word: “Cat."
  - Context (window size 2): “the," “sat," “on”.
  - Goal: Predict these from “Cat"
- **How It Learns:**
  - Starts with random vectors.
  - Uses positive examples (words that appear together).
  - Uses negative examples (random words, like "garden," that don’t).
  - Adjusts vectors so dot product is high for positives, low for negatives.
- Strength: Great for rare words and smaller datasets.

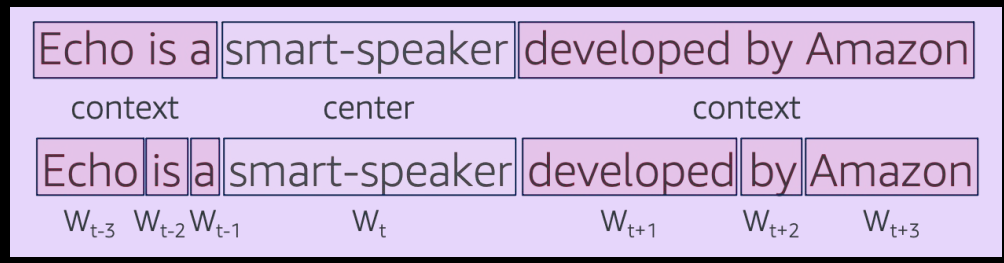




Word2vec: CBOW

- CBOW: Predicts the center word from context words.
- Example: "Echo is a smart-speaker developed by Amazon."
  - Context (window size 2): "is," "a," "developed," "by."
  - Center word: "smart-speaker."
  - Goal: Predict "smart-speaker" from these.
- How It Learns:
  - Random vectors for all words.
  - Trains with positive (co-occurring) and negative (random) pairs.
  - Negative sampling: 5–20 for small data, 2–5 for large.
- Strength: Faster, better for frequent words.
- Output: Dense vectors from hidden layer (e.g., 300-D).





word2vec algorithm: Negative samples

- What: Speeds up training by using a few “wrong” examples.
- Why: Avoids comparing every word to every other word.
- How It Works:
  - Positive: Real pairs (e.g., "cat" and "sat" from "The cat sat").
  - Negative: Random words (e.g., "pizza," "cloud") that don’t fit.
  - Adjusts vectors: "cat" closer to "sat," farther from "pizza."
- Details:
  - 5–20 negatives for small data, 2–5 for large.
  - Picks rare words more often (e.g., "cloud" over "the").
- Result: Fast, effective learning of word relationships.

Negative examples

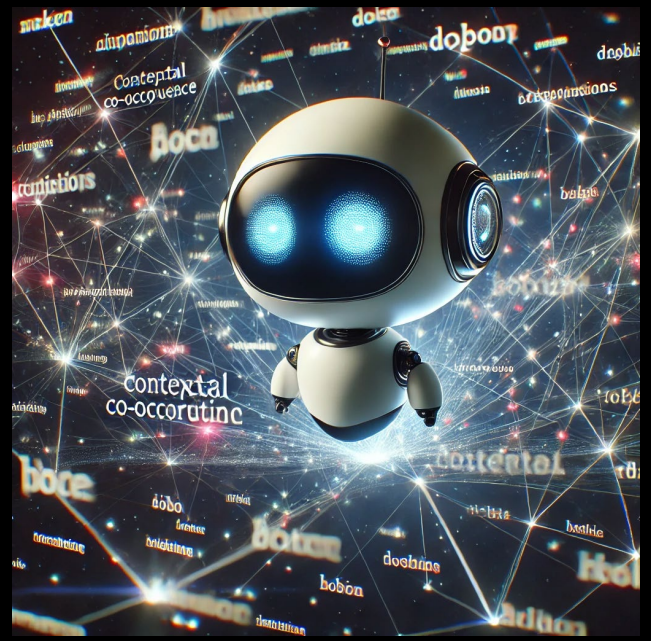
Center Word	Context
smart-speaker	hello
smart-speaker	world
smart-speaker	garden

Comparison of word2vec architectures

Architecture	Predicts	Relative Strength
Skip-gram	Context words from a given target word	• Better for a smaller corpus • Represents rare words well
CBOW	Target word from given context words	• Multiple times faster • Represents frequent words slightly better

GloVe: Global Vectors for word representation

- Introduced in 2014 by Stanford.
- Creates dense word vectors (e.g., 100–300 numbers) using global text patterns.
- Key Idea: Captures how often words appear together across a whole corpus.
- Uses: Word similarity, analogies (e.g., "king" - "man" + "woman" ≈ "queen").
- Differs from Word2Vec: Looks at the big picture, not just nearby words.



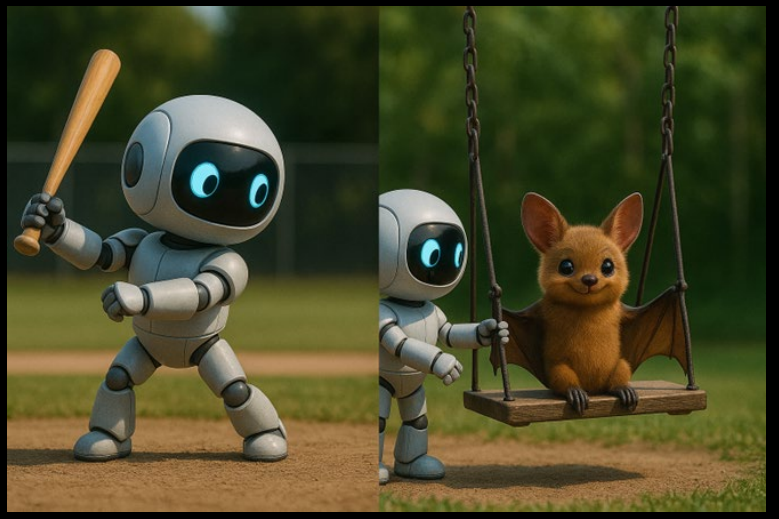
GloVe – The Co-occurrence Matrix

- **Builds a co-occurrence matrix:** Counts word pairs in a corpus.
- Example: "It is not a dog, it is a wolf."
  - Window size 1 (next-door neighbors):
    - "it" and "is": 2 times.
    - "is" and "not": 1 time.
    - "dog" and "a": 1 time, "wolf" and "a": 1 time.
- Goal: Use these counts to learn word relationships.

Word	it	is	not	a	dog	wolf
it	0	2	0	0	1	0
is	2		1	1	0	0
not	0	1	0	1	0	0
a	0	1	1	0	1	1
dog	1	0	0	1	0	0
wolf	0	0	0	1	0	0

Limitations of Word2Vec and GloVe

- Word2Vec Limitations:
  - Treats words as whole units (e.g., "dogs" and "dog" unrelated).
  - Struggles with rare or new (OOV) words.
- GloVe Limitations:
  - Also ignores word structure (e.g., "running" vs. "run").
  - Needs large corpora for best results.



Beyond Basic FastText: Handling Sub-words

- Extends Word2Vec by using **sub word** pieces (n-grams).
- Represents each word as a bag of character n-grams
- Each center word is represented as a set of subwords:
  - “where” using n-gram
  - n=3: <wh, whe, her, ere, re>
- Uses: Handles rare words and out-of-vocabulary (OOV) terms – address **BoW** and **GloVe** limitations
  - E.g. long but infrequent words, such as **pneumonoultramicroscopicsilicovolcanoconiosis**

Sentence Embeddings: *Moving from Words to Documents*

Challenge: How do we represent entire sentences or documents?

Simple Approaches:

- **Average word embeddings:** Mean of all word vectors in document
- **Weighted average:** TF-IDF weighted mean of word vectors
- **Max pooling:** Take maximum value for each dimension

Advanced Approaches:

- **Doc2Vec:** Extension of Word2Vec for documents
- **Sentence-BERT:** Transformer-based sentence embeddings
- **Universal Sentence Encoder:** Google's pre-trained sentence embeddings

- **Applications:** Document similarity, semantic search, clustering

Connecting Text & Audio Representations

Processing Stage	Text (Module 4)	Audio (Module 3)
Raw Input	Characters, Words	Waveforms, Sound waves
Preprocessing	Cleaning, Normalization	Noise reduction, Normalization
Tokenization	Word/Sentence boundaries	VAD, Segmentation
Feature Extraction	BOW, TF-IDF, N-grams	MFCCs, Spectrograms
Dense Representations	Word2Vec, GloVe, BERT	Audio embeddings, wav2vec
Applications	Search, Classification	Speech recognition, TTS

The Bridge: Audio → ASR → Text → Text NLP **Key Insight:** Both modalities transform raw signals into meaningful features for machine learning

Embedding Evaluation Methods

How Do We Know If Our Embeddings Are Good?

Intrinsic Evaluation:

- **Word similarity:** Correlation with human judgments
- **Analogy tasks:** king - man + woman = ?
- **Clustering:** Do similar words cluster together?

Extrinsic Evaluation:

- **Downstream task performance:** How well do they work in actual applications?
- **Classification accuracy:** Using embeddings as features
- **Information retrieval:** Relevance of search results

Evaluation Datasets:

- WordSim-353: Human similarity judgments
- Google Analogy Dataset: Syntactic and semantic analogies
- SimLex-999: Similarity vs relatedness
- **Key Principle:** Good embeddings should capture human intuitions about word relationships

Embedding Models: Advantages and Limitations

Model	Advantages	Limitations	Best For
Word2Vec	Fast, simple, interpretable	OOV problem, no subword info	General purpose, smaller datasets
GloVe	Global statistics, stable training	OOV problem, requires large corpus	Large, stable datasets
FastText	Handles OOV, captures morphology	More complex, larger models	Rare words, multiple languages
BERT	Contextual, state-of-art	Computationally expensive, less interpretable	High-accuracy applications

Ethical considerations in Embeddings

Embeddings Can Perpetuate Societal Biases

Common Bias Examples:

- Gender: "programmer" - "man" + "woman" ≈ "homemaker"
- Race: Ethnic names associated with unpleasant words
- Age: "young" associated with positive traits, "old" with negative

Why This Happens:

- Embeddings learn from human-generated text
- Human text contains societal biases and stereotypes
- Statistical patterns amplify existing prejudices

Mitigation Strategies:

- Bias detection and measurement
- Debiasing techniques (post-processing)
- Diverse training data
- Awareness and ongoing monitoring

- **Ethical Responsibility:** Consider the social impact of your models

Key Takeaways

Dense embeddings capture semantic relationships

Different models excel in different scenarios

Context and task determine the best approach

All representations involve trade-offs (accuracy vs. interpretability vs. efficiency)

Evaluation is crucial – test on your specific task

Bias awareness and mitigation are ethical imperatives

Your Journey: From word counting to semantic understanding

Practical Applications

- **Search Engines:** Understanding query intent beyond exact keyword matching
- **Recommendation Systems:** "Users who liked this also liked..." based on content similarity
- **Machine Translation:** Mapping concepts across languages using shared semantic spaces
- **Sentiment Analysis:** Understanding opinion and emotion in customer feedback
- **Chatbots and Virtual Assistants:** Comprehending user requests and generating responses
- **Content Moderation:** Automatically detecting harmful or inappropriate content
- **Document Organization:** Clustering and categorizing large document collections
- **Personalization:** Tailoring content recommendations to individual users

The Future: Multimodal embeddings combining text, images, audio, and video