MTH 227

Applied Natural Language Processing

Batuhan Bardak

Lecture 2: Text to Vector

Date: 09.10.2025

Our Roadmap For Today

Part 1: Frequency-Based (Sparse) Vectors

- One-hot Encoding: The Basic Idea
- Bag-of-Words (BoW): Counting Occurrences
- TF-IDF: Smart Counting

Part 2: Prediction-Based Vectors - Word Embeddings

- Word2Vec (CBOW & Skip-Gram)
- Glove: Combining Counting & Prediction
- FastText: Learning from Sub-words
- Bonus: Doc2Vec (Document Embeddings)

Part 3: Practical Field Guide

- Choosing the Right Vectorizer (A Decision Guide)
- Common Pitfalls and How to Avoid Them

Why Do We Need Vectors?

Why can’t model just “Read”?

- Machine Learning models are mathematical functions.
- They require numerical input, not raw text.
- **The goal**: To represent in a way that captures their meaning, relationships, and context.
- **The Journey**: From simple counting to capturing semantics

Preprocessing & Vectorization

How Last Week's Topic Drives This Week's Choices

- **Sparse Models (BoW, TF-IDF)**: LOVE Aggressive Preprocessing
  a. Why? Their goal is to reduce the feature space. A smaller vocabulary is better.
  b. Recommended: Stop Word Removal, Stemming/Lemmatization
- **Dense Models (Word2Vec, etc.) Prefer Context.**
  a. Why? They learn from context. Over-cleaning may/can remove valuable signs.
  b. **Word2Vec/Glove:** Moderate preprocessing. Lowercasing and light stop word removal is common. Avoid aggressive stemming.
  c. **FastText:** Minimal preprocessing. It learns from sub-words, so morphology is its strength! Never use stemming.

One-Hot Encoding (A Simple Start)

Every word is an Island

- **How it works**: Building a vocabulary and assigning a unique binary vector to each word.
- Visual example: [cat, dog, man] -> cat: [1, 0, 0], dog: [0, 1, 0].
- **Major Drawbacks:**
  a. Huge vector dimensions (curse of dimensionality)
  b. Extremely sparse (mostly zeros)
  c. No notion of similarity: The vectors for “cat” and “dog” are mathematically unrelated.

One-Hot Encoding (A Simple Start)

id	color
1	red
2	blue
3	green
4	blue

One Hot Encoding

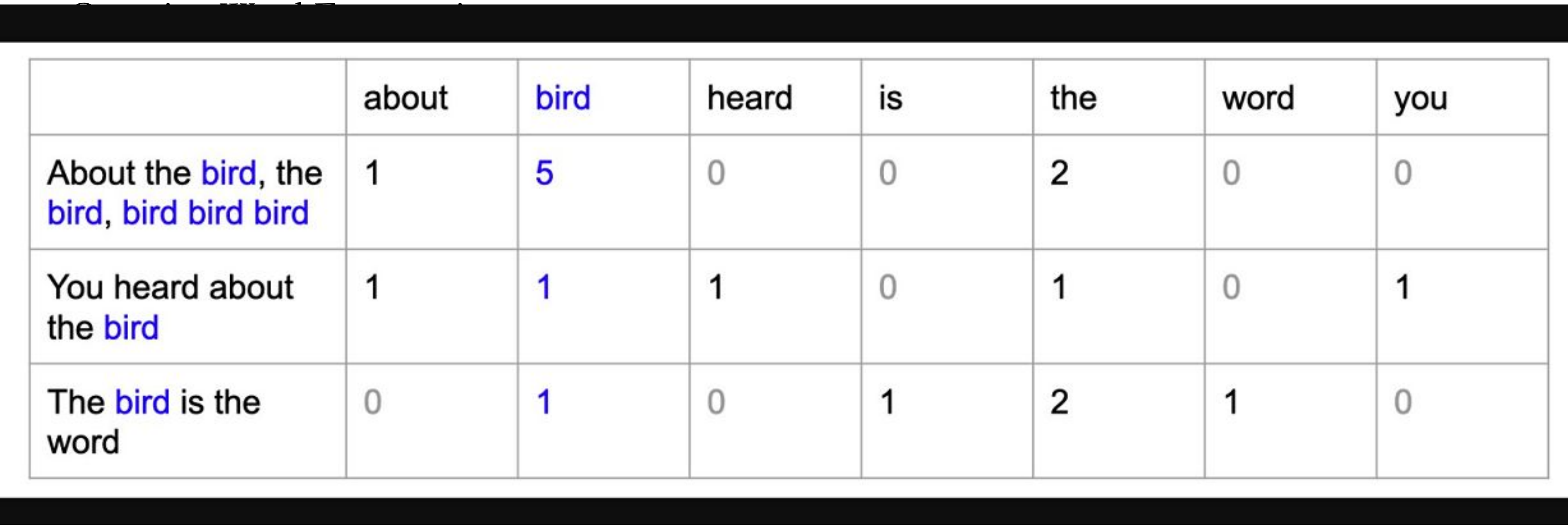
id	color_red	color_blue	color_green
1	1	0	0
2	0	1	0
3	0	0	1
4	0	1	0

Bag-of-Words (BoW)

Counting Word Frequencies

- **The "Bag" metaphor**: Ignoring grammar and word order, only counting frequency.
- **How it works**: Creating a vector representing the word counts in a document.
- Example: "the cat sat on the mat" -> {"the": 2, "cat": 1, "sat": 1, ...}.
- **Improvements & Lingering Problems:**
  - Improvement over One-Hot: Captures word importance (frequency).
  - Problems: Still sparse, loses word order, common words (like "the") can dominate.
  - “Man bites dog” and “Dog bites man” produce the exact same vector. We lose crucial context.

Bag-of-Words (BoW)



	about	bird	heard	is	the	word	you
About the bird, the bird, bird bird bird	1	5	0	0	2	0	0
You heard about the bird	1	1	1	0	1	0	1
The bird is the word	0	1	0	1	2	1	0

TF-ID (Smarter Counting)

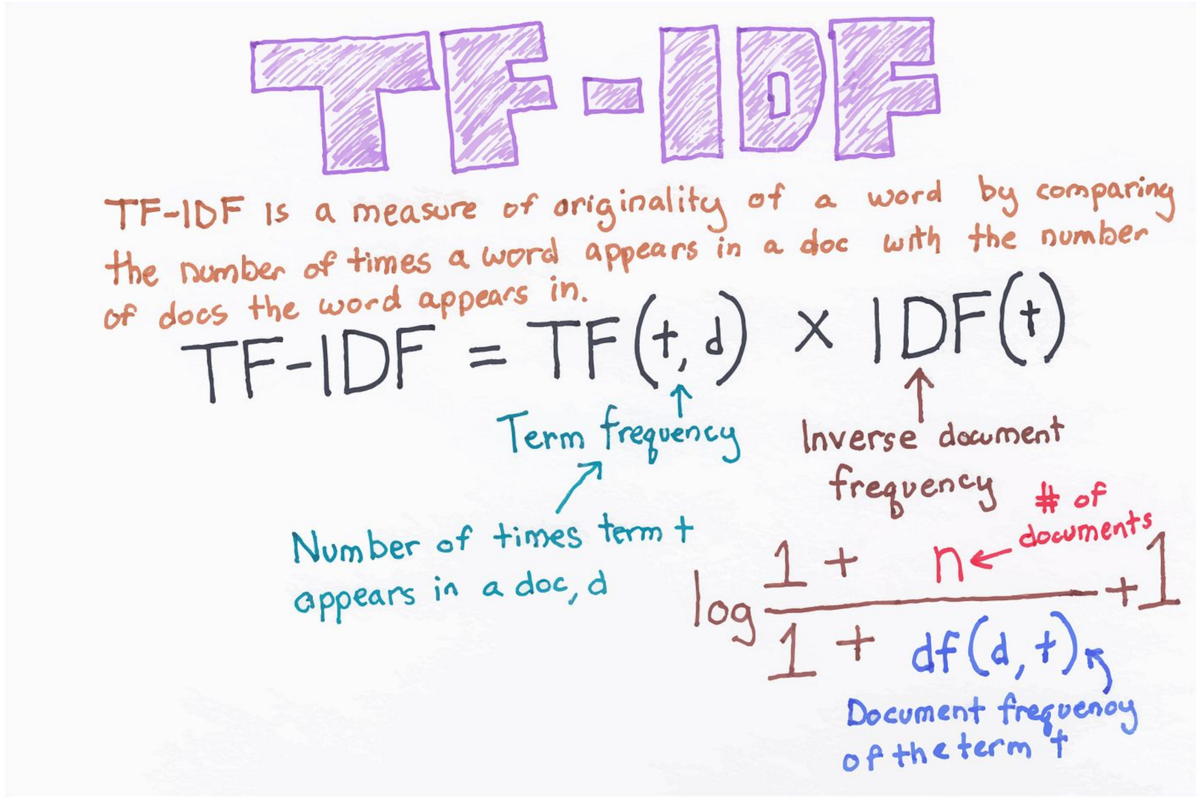
Weighing Words by Importance

- **The Intuition:** A word is more important if it appears frequently in one document but rarely in all other documents.
- Breaking it down:
  a. **TF (Term Frequency)**: How often does a word appear in a document? (Same of BoW)
     i. ***(# of times term t appear in document d / total # of terms in document d)***
  b. IDF (Inverse Document Frequency): Penalizes words that are common across the entire corpus.
     i. log (# of documents in corpus D / # of documents containing term t)
- Formula overview: **TF - IDF (t,d) = TF (t,d) * IDF (t)**
- **The result**: A score that highlights signature words of a document

TF-ID (Smarter Counting)

- **High Score:** The word is frequent in this document but rare overall. (e.g., "spiderman" in a movie review). **This word is important!**
- **Low Score:** The word is either very common everywhere (like "the") or very rare and appears only once.

TF-ID (Smarter Counting)



TF-IDF

TF-IDF is a measure of originality of a word by comparing the number of times a word appears in a doc with the number of docs the word appears in.

$$\text{TF-IDF} = \text{TF}(t, d) \times \text{IDF}(t)$$

Term frequency

Number of times term t appears in a doc, d

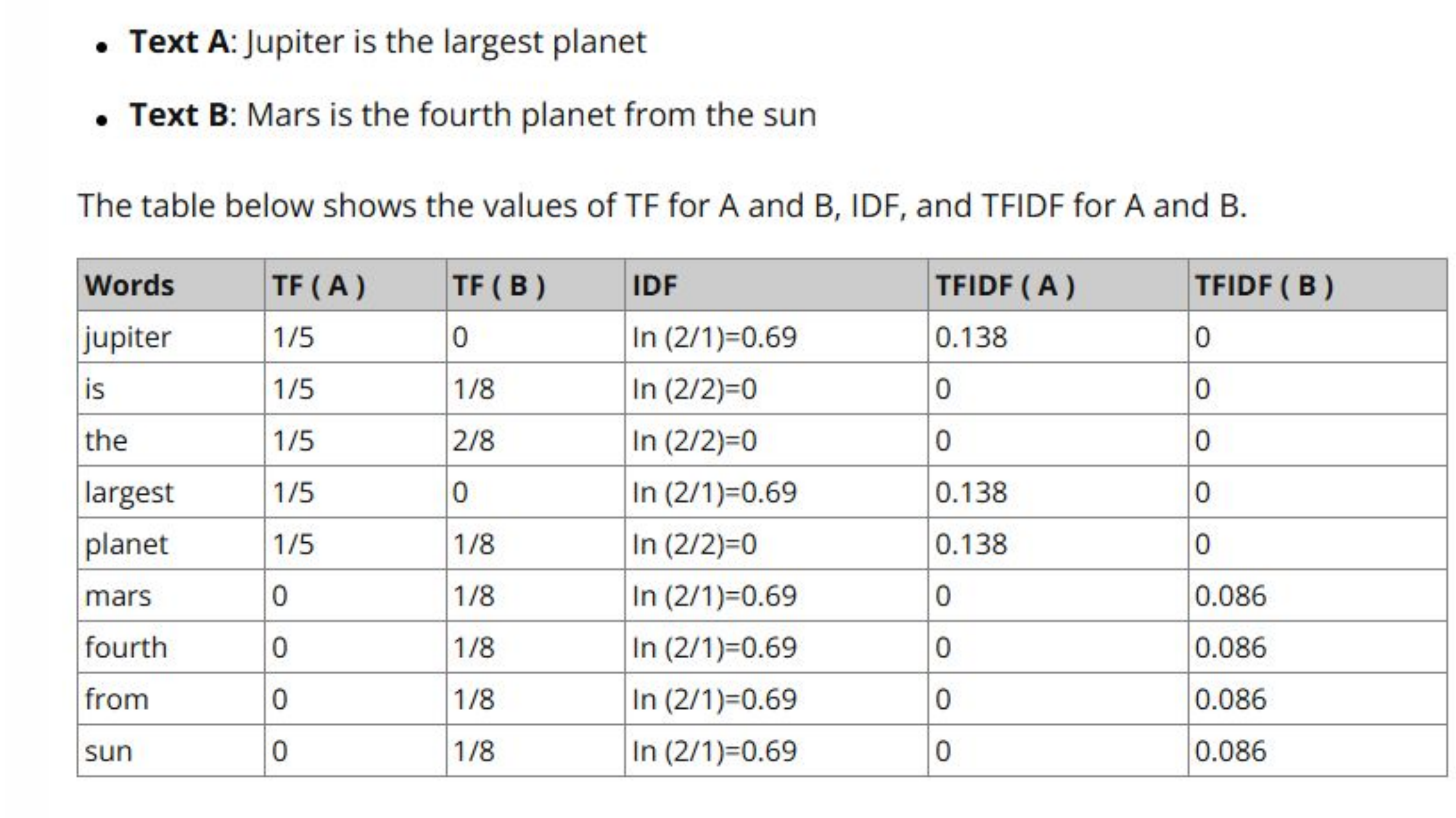
Inverse document frequency

$$\log \frac{1 + n}{1 + df(d, t)} + 1$$

# of documents

Document frequency of the term t

TF-ID (Smarter Counting)



- **Text A**: Jupiter is the largest planet
- **Text B**: Mars is the fourth planet from the sun

The table below shows the values of TF for A and B, IDF, and TFIDF for A and B.

Words	**TF (A)**	**TF (B)**	**IDF**	**TFIDF (A)**	**TFIDF (B)**
jupiter	1/5	0	ln (2/1)=0.69	0.138	0
is	1/5	1/8	ln (2/2)=0	0	0
the	1/5	2/8	ln (2/2)=0	0	0
largest	1/5	0	ln (2/1)=0.69	0.138	0
planet	1/5	1/8	ln (2/2)=0	0.138	0
mars	0	1/8	ln (2/1)=0.69	0	0.086
fourth	0	1/8	ln (2/1)=0.69	0	0.086
from	0	1/8	ln (2/1)=0.69	0	0.086
sun	0	1/8	ln (2/1)=0.69	0	0.086

TF-ID (Smarter Counting)

Pros

- Simple and fast.
- Vastly superior to BoW by giving weights to words.
- Very effective for many tasks like search engines and text classification.

Cons

- Still a "bag-of-words" approach: **no context or word order.**
- **Doesn't understand meaning.** The vectors for "car," "auto," and "vehicle" are completely different and unrelated.

Limitations of Frequency-Based Methods

The Wall We Hit with Counting

- **Sparsity:** Vectors are still very large and mostly zero.
- **No Semantic Understanding:** "Car", "auto", and "vehicle" are treated as completely different, unrelated words.
- **Context is Lost:** The meaning of "bank" (river vs. financial) cannot be captured.
- The need for a better approach: We need dense, meaning-rich vectors.

Word Embeddings

Core Intuition

- "You shall know a word by the company it keeps." - J.R. Firth
- Words that appear in similar contexts probably have similar meanings.
- **Goal:** Represent words as dense vectors in a multi-dimensional space, where similar words are placed close together.

The Vector Space of Meaning

Embeddings place words in a geometric space. The distance and direction between vectors capture semantic relationships.

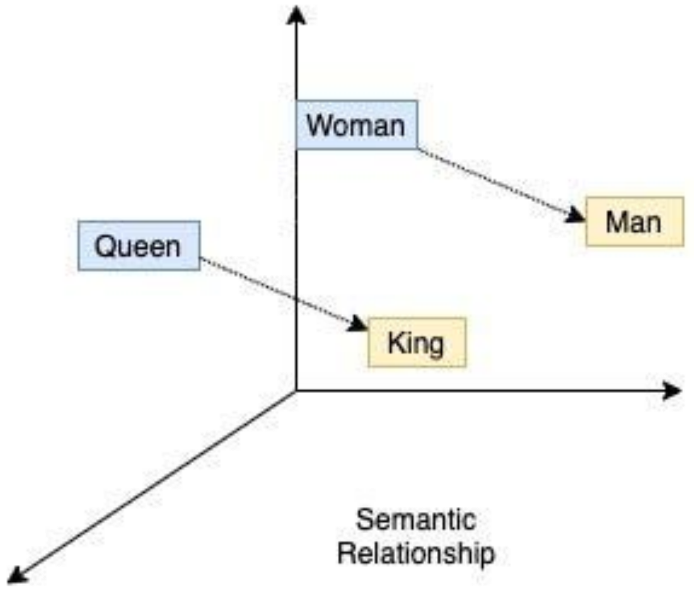
This space allows us to see that King is to Queen as Man is to Woman.

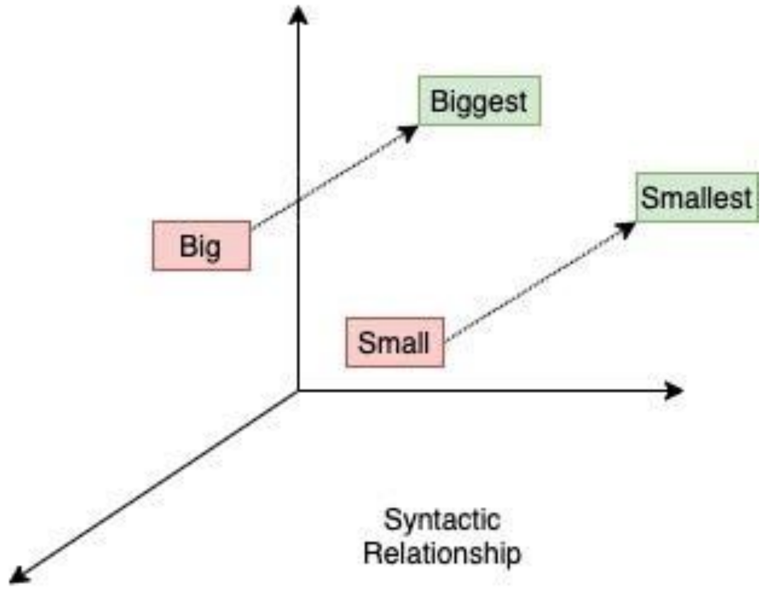
Why Dense Vectors?

- Short vectors are easier to use as features in ML systems

- Dense vectors may generalize better than storing explicit counts

The Vector Space of Meaning





Neural Networks 101

- **Check the other slides for NN**

Word2Vec

- **Input**: a large text corpara, V, d
  a. V: a pre-defined vocabulary
  b. D: dimension of word vectors (e.g. 30
  c. Text corpora:
     i. Wikipedia + Gigaword 5: 6B
     ii. Twitter: 27B
     iii. Common Crawl: 840B

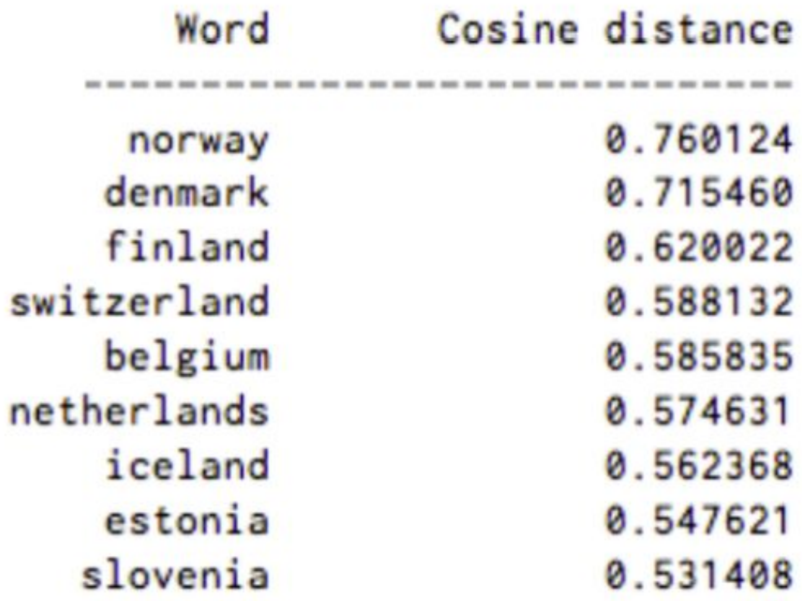
$$v_{\text{cat}} = \begin{pmatrix} -0.224 \\ 0.130 \\ -0.290 \\ 0.276 \end{pmatrix} \quad v_{\text{dog}} = \begin{pmatrix} -0.124 \\ 0.430 \\ -0.200 \\ 0.329 \end{pmatrix}$$

$$v_{\text{the}} = \begin{pmatrix} 0.234 \\ 0.266 \\ 0.239 \\ -0.199 \end{pmatrix} \quad v_{\text{language}} = \begin{pmatrix} 0.290 \\ -0.441 \\ 0.762 \\ 0.982 \end{pmatrix}$$

- **Output:** $f : V \to \mathbb{R}^d$

Word2Vec

word = “sweden”



```
        Word       Cosine distance
----------------------------------
      norway              0.760124
     denmark              0.715460
     finland              0.620022
 switzerland              0.588132
     belgium              0.585835
 netherlands              0.574631
     iceland              0.562368
     estonia              0.547621
    slovenia              0.531408
```

Word2Vec Architecture

Word2Vec isn't one model, but two main architectures that use a "fake" prediction task to learn embeddings.

1. **CBOW (Continuous Bag-of-Words)**
   - **Goal:** Predict a target word based on its context words.
   - **Analogy:** Filling in the blank. [The, cat, ___, on, the, mat] -> Predict sat.
   - **Fast and efficient** for frequent words.
2. **Skip-gram**
   - **Goal:** Predict context words based on a single input word.
   - **Analogy:** Given a word, guess its neighbors. sat -> Predict [The, cat, on, the].
   - **Slower but better** for infrequent words and larger datasets.

-

Word2Vec: The "Fake" Task & Training

The key insight: **We don't actually care about the prediction task!**

We care about the weights learned in the **hidden layer**. After the model is trained, we discard the output layer and use the hidden layer's weights as our word embeddings. This learned weight matrix is our lookup table for word vectors.

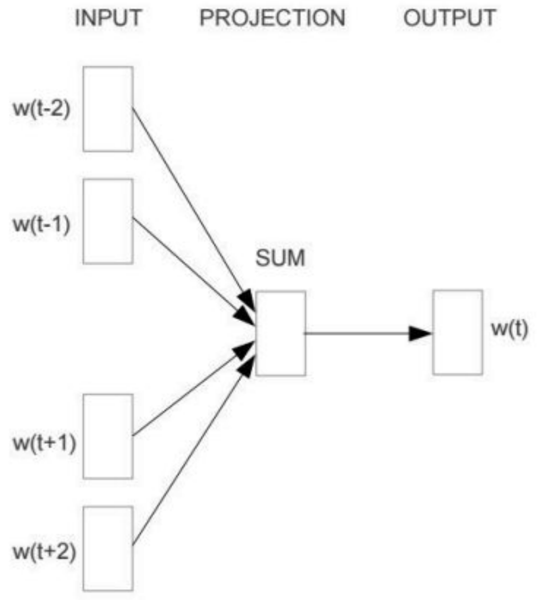
Training Optimization: Negative Sampling

Updating millions of weights for every training sample is too slow. **Negative sampling** solves this: for each training instance, instead of updating all weights, we only update:

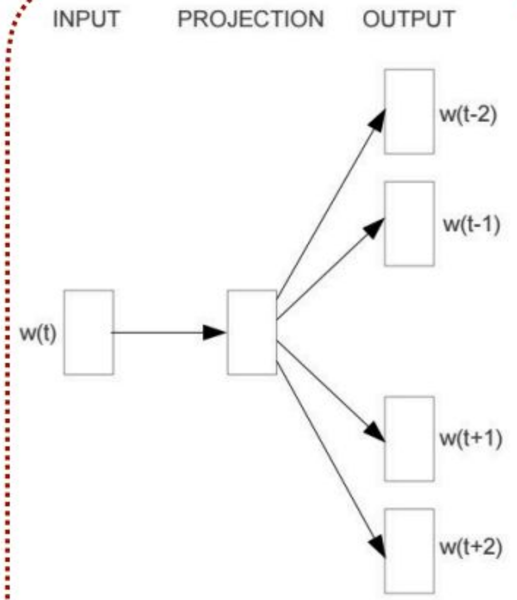
- The **correct** output word (the "positive" sample).
- A few randomly chosen **incorrect** words (the "negative" samples).

This makes training dramatically more efficient.

Word2Vec



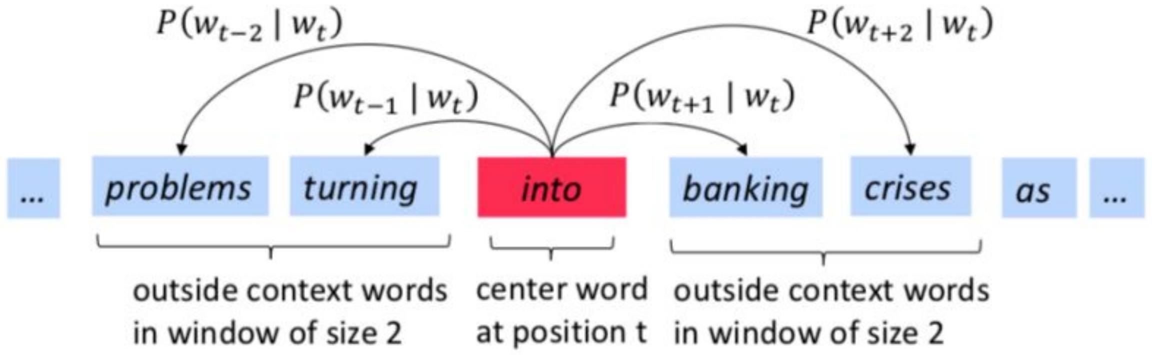
Continuous Bag of Words (CBOW)



Skip-grams

Skip-gram

- **The idea:** We want to use words to **predict** their context words
- **Context**: a fixed window of size 2m



Ref: Washington CS-Sep517

Skip-gram: Objective Function

- For each position $t = 1,2,\ldots T$, predict context words within context size m, given center word w_j:

all the parameters to be optimized

$$\mathcal{L}(\theta) = \prod_{t=1}^{T} \prod_{-m \le j \le m, j \ne 0} P(w_{t+j} \mid w_t; \theta)$$

- The objective function $J(\theta)$ is the (average) negative log likelihood:

$$J(\theta) = -\frac{1}{T} \log \mathcal{L}(\theta) = -\frac{1}{T} \sum_{t=1}^{T} \sum_{-m \le j \le m, j \ne 0} \log P(w_{t+j} \mid w_t; \theta)$$

How to define $P(w_{t+j} \mid w_t; \theta)$?

- We have two sets of vectors for each word in the vocabulary

$\mathbf{u}_i \in \mathbb{R}^d$: embedding for target word *i*

$\mathbf{v}_{i'} \in \mathbb{R}^d$: embedding for context word *i'*

- Use inner product $\mathbf{u}_i \cdot \mathbf{v}_{i'}$ to measure how likely word *i* appears with context word *i'*, the larger the better

"softmax" we learned last time!

$$P(w_{t+j} \mid w_t) = \frac{\exp(\mathbf{u}_{w_t} \cdot \mathbf{v}_{w_{t+j}})}{\sum_{k \in V} \exp(\mathbf{u}_{w_t} \cdot \mathbf{v}_k)}$$

$\theta = \{\{\mathbf{u}_k\}, \{\mathbf{v}_k\}\}$ are all the parameters in this model!

Q: Why two sets of vectors?

Any issues?

How to train the model

Calculating all the gradients together!

$$\theta = \{\{\mathbf{u}_k\}, \{\mathbf{v}_k\}\}$$

$$J(\theta) = -\frac{1}{T}\sum_{t=1}^{T}\sum_{-m\le j\le m, j\ne 0}\log P(w_{t+j} \mid w_t; \theta) \qquad \nabla_\theta J(\theta) = ?$$

Q: How many parameters are in total?

We can apply stochastic gradient descent (SGD)!

$$\theta^{(t+1)} = \theta^{(t)} - \eta\nabla_\theta J(\theta)$$

Skip-gram with negative sampling

Idea: recast problem as binary classification!

- Target word is positive example
- All words not in context are negative

$$P(D=1 \mid t,c) = \sigma(\mathbf{u}_t \cdot \mathbf{v}_c)$$

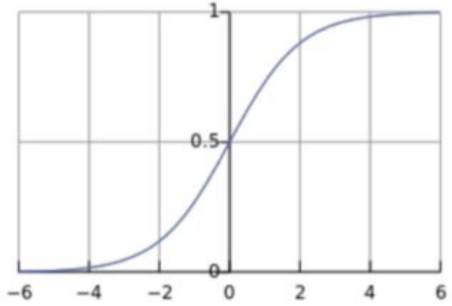
$$\sigma(x) = \frac{1}{1+\exp(-x)}$$

positive examples +

t	c
apricot	tablespoon
apricot	of
apricot	jam
apricot	a

negative examples -

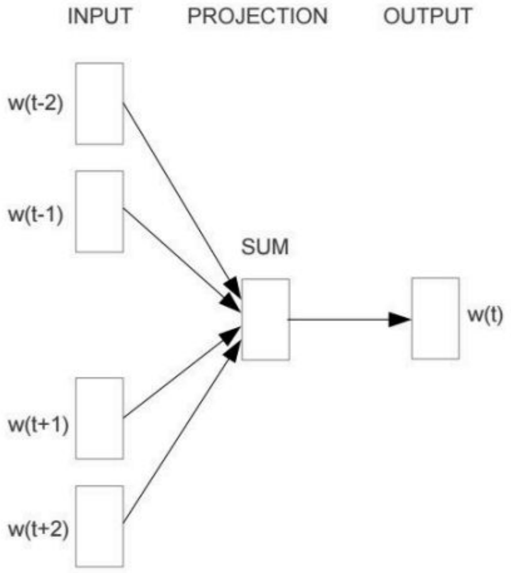
t	c	t	c
apricot	aardvark	apricot	seven
apricot	my	apricot	forever
apricot	where	apricot	dear
apricot	coaxial	apricot	if



To compute loss, pick K random words as negative examples:

$$J(\theta) = -P(D=1 \mid t,c) - \frac{1}{K}\sum_{i=1}^{K} P(D=0 \mid t_i, c)$$

Continuous Bag of Words (CBOW)



$$L(\theta) = \prod_{t=1}^{T} P\left(w_t \mid \{w_{t+j}\}, -m \leq j \leq m, j \neq 0\right)$$

$$\bar{\mathbf{v}}_t = \frac{1}{2m} \sum_{-m \leq j \leq m, j \neq 0} \mathbf{v}_{t+j}$$

$$P(w_t \mid \{w_{t+j}\}) = \frac{\exp(\mathbf{u}_{w_t} \cdot \bar{\mathbf{v}}_t)}{\sum_{k \in V} \exp(\mathbf{u}_k \cdot \bar{\mathbf{v}}_t)}$$

Word2Vec: Key Parameters

When training a Word2Vec model, you'll often encounter these parameters:

- vector_size: The dimensionality of your word vectors (e.g., 100, 300). Larger sizes can capture more information but require more data.
- window: The maximum distance between the current and predicted word within a sentence. A window of 5 means looking at 5 words before and 5 words after.
- min_count: Ignores all words with a total frequency lower than this. Useful for filtering out noise.
- sg or cbow: sg=1 selects the Skip-gram algorithm; cbow=0 selects CBOW.

Skip-Gram vs CBOW

Feature	**CBOW**	**Skip-Gram**
Objective	Context to Word	Word to Context
Speed	Fast	Slower
Rare Words	Poor	Excellent
Large datasets	Ideal	Slower
Semantic Richness	Moderate	High

Skip-Gram vs CBOW (When to use Which)

Situation	**Recommended Model**
Large dataset, emphasis on speed	CBOW
Small dataset, or rare-word focus	Skip-Gram
Fine semantic similarity tasks	Skip-Gram
General embedding for downstream tasks	CBOW (faster)

Pros & Cons of Word2Vec

- **Pros**
  - Captures complex semantic relationships (like the King - Queen example)
  - High-quality embeddings from relatively simple neural network architecture
  - Pioneered the field of modern word embeddings.
- **Cons**
  - Only uses local context. It ignores global word co-occurrence statistics.
  - Cannot handle out-of-vocabulary (OOV) words. If a word was not in the training data, it has no vector.

Glove: Global Vectors

A Different Philosophy

While Word2Vec is a **predictive** model that looks at local context windows, GloVe is a **count-based** model that leverages global statistics. It combines the strength of traditional count based methods (like LSA) and predictive methods (like Word2Vec)

Core Idea

GloVe learns by looking at a **global word-word co-occurrence matrix**. This matrix tells us how frequently word *i* appears in the context of word *j* across the entire corpus.

The model is trained to learn vectors whose dot product equals the logarithm of their co-occurrence probability. This focus on co-occurrence ratios is what makes GloVe so good vord analogy tasks.

Glove: Global Vectors

How it works:

- First, it builds a large **word co-occurrence matrix** from the entire corpus. Each element X_{ij} in the matrix represents how often word *i* appears in the context of word *j*.
- Then, it uses matrix factorization to reduce the dimensionality of this matrix, learning a vector for each word.
- The training objective is to learn vectors such that their dot product equals the logarithm of their co-occurrence probability.

Pros & Cons of Glove

- **Pros**
  - Very fast to train as it capitalizes on global statistics.
  - Often perform better that Word2Vec on word analogy tasks and named entity recognition.
  - Makes explicit use of the vast statistical information available in the corpus.
- Cons
  - Requires a large amount of memory to store the co-occurrence matrix, especially very large vocabulary (there are some solutions to reduce this memory for sparse matrix and with the objective function trick).
  - Like Word2Vec, **it cannot handle OOV words**.

FastText: Sub-Word Embeddings

Core Idea: A word is not a single atomic unit but is composed of smaller parts (subwords or character n-grams).

How it Works:

- It's an extension of the Word2Vec (Skip-gram) model.
- Instead of learning a vector for each word, FastText learns vectors for **character n-grams**.
- For a word like "apple" and n=3 (trigrams), it would break it down into:
    - <ap, app, ppl, ple, le> (plus the special sequence for the whole word <apple>).
- The final vector for the word "apple" is the sum of the vectors of its constituent n-grams.

Pros & Cons of FastText

- **Pros:**
  - **Excellent handling of OOV words.** It can generate a vector for a word it has never seen before by summing the vectors of its character n-grams. This is its biggest advantage.
  - Works very well for morphologically rich languages (e.g., Turkish, German, Finnish) where many words are formed by adding prefixes and suffixes.
  - Often outperforms Word2Vec and GloVe on syntax-based tasks.
- **Cons:**
  - Training is much slower and more memory-intensive than Word2Vec or GloVe because it needs to store all the n-gram vectors.
  - The generated embeddings for rare or OOV words might be of lower quality, but it's better than having no vector at all.

Embedding Models: Head-to-Head Comparison

Feature	**Word2Vec**	**Glove**	**FastText**
Training Method	Predictive (Local Context Window)	Count-based (Global Co-occurrence Matrix)	Predictive (Local Context + Subwords)
Core Unit	Word	Word	Character n-grams
OOV Handling	No	No	Yes - Can generate vectors for any word.
Training Speed	Fast	Very Fast	Slow
Memory Usage	Low	High (for co-occurrence matrix)	Very High (n-gram dictionary)
Best For	General semantic tasks, clean corpora.	Word analogies, named entities, large corpora	Morphologically rich languages, noisy text

When Should I Use Which Embedding?

Start Here: Use a **pre-trained GloVe or Word2Vec** model. They are powerful, general-purpose baselines that work well for a huge range of standard English NLP tasks.

Is your text full of typos, slang, or rare words? Or are you working with a morphologically rich language (e.g., German, Turkish)?

Yes: Use **fastText**. Its ability to handle OOV words is a game-changer.

Do you need to train embeddings from scratch on your own domain-specific data?

- **Word2Vec (CBOW)** is often the fastest to train.
- **GloVe** can be very effective if your corpus is large enough to build meaningful co-occurrence statistics.

Practical Tips & Common Pitfalls

Practical Tips

- **Always use pre-trained models first!** Don't train your own unless you have a very large, domain-specific corpus (e.g., medical texts, legal documents).
- **Fine-tuning:** For best results, you can take a pre-trained model and continue training it (fine-tune) on your own data.
- **Vector size matters.** 300 is a common and effective dimension, but for smaller, specific datasets, 50-100 might be sufficient.

Practical Tips & Common Pitfalls

Common Pitfalls

- **Preprocessing Mismatch:** Pre-trained models were trained on text that was preprocessed a certain way (e.g., all lowercase). You should apply the *same* preprocessing to your text.
- **Averaging is not enough:** Taking the average of word vectors to get a sentence vector is a good start, but it loses all word order. This is a major limitation for complex tasks, which leads to more advanced models like LSTMs and Transformers.

How to embed whole sentences?

Averaging Word Embeddings

This is the simplest baseline.

1. For each word in the sentence, get its pre-trained word vector.
2. Compute the element-wise average of all these vectors.

Pros:

- Incredibly simple, fast, and often a surprisingly good baseline.

Cons:

- **Ignores word order!** It also treats every word as equally important, meaning common stop words can dilute the overall meaning.

Weighted Averaging Word Embeddings

We can improve this by giving more weight to more "important" words.

1. Calculate a weight for each word, commonly using its **TF-IDF score.**
2. Compute a weighted average of the word vectors.

Pros:

- Better than simple averaging as it highlights important terms.

Cons:

- Still ignores word order and syntax.

The Fatal Flaw of Averaging

Let's consider why ignoring word order is so problematic. Take these two sentences:

- **Sentence A:** "The dog bit the man."
- **Sentence B:** "The man bit the dog."

If we use simple averaging, we sum and average the exact same set of word vectors: {"the", "dog", "bit", "man"}.

- The resulting sentence vectors for A and B would be **identical**, even though their meanings are completely different. This proves we need a more sophisticated approach—one that understands structure and context.

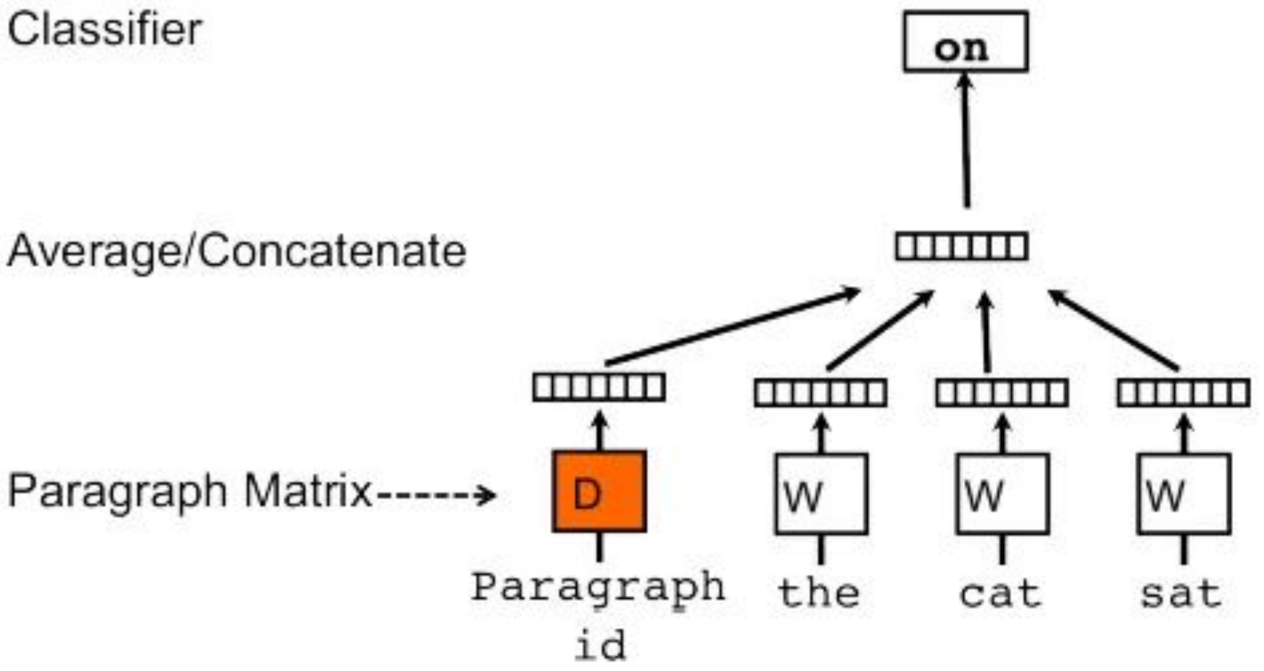
Introduction to Doc2Vec (Paragraph Vectors)

Doc2Vec, also known as Paragraph Vectors, was introduced by Quoc Le and Tomas Mikolov at Google in 2014. It's a direct and clever extension of the Word2Vec model.

The Core Idea: Instead of just learning vectors for words, Doc2Vec learns a vector representation for a piece of text (a sentence, paragraph, or document) **directly** during training.

Each document is represented by a unique vector, which we can call the **Document Vector** or **Paragraph ID**. This vector acts as a form of memory, capturing the overall topic or semantic context of the document that is missing from a small window of context words.

Introduction to Doc2Vec (PV-DM)



Doc2Vec Architectures

Just like Word2Vec has two main training algorithms (CBOW and Skip-gram), Doc2Vec also has two parallel architectures.

1. **Distributed Memory (PV-DM):** This is analogous to the CBOW model in Word2Vec. It uses context words *plus* the document vector to predict a target word.
2. **Distributed Bag of Words (PV-DBOW):** This is analogous to the Skip-gram model. It is simpler and ignores word order. It uses the document vector alone to predict random words from the document.

Architecture 1: Distributed Memory (PV-DM)

The PV-DM model is built on a simple but powerful idea: the document's topic influences the words that appear.

- **Goal:** Predict a target word using its surrounding context words **AND** the unique document vector.

How it works:

1. Every document is mapped to a unique vector, D. Every word is also mapped to a vector, W.
2. To predict a target word, the model takes the document vector D and the vectors of the context words (e.g., the 4 words before and 4 words after).
3. These vectors are averaged or concatenated to form a single input vector.
4. This input vector is then fed through a neural network to predict the target word using a softmax classifier.

Architecture 2: Distributed Bag of Words (PV-DBOW)

The PV-DBOW model is much simpler and often faster to train.

- **Goal:** Predict a random sample of words from the document, given only the document vector.

How it works:

1. The model forces the document vector D to predict words chosen randomly from that document.
2. In essence, the input is just the document vector, and the network is trained to output a probability distribution of words. We then train it to increase the probability of words that are actually in the document.

This model is called "Distributed Bag of Words" because it completely **ignores word order**. It's essentially a Skip-gram model where the input is the document ID instead of a target word.

Doc2Vec in Practice & Applications

Key Advantages:

- It captures semantic meaning more robustly than simple averaging.
- The PV-DM model accounts for word order and context.
- It produces a fixed-length, numerical vector for any piece of text, regardless of its length. This is perfect for machine learning!

Common Applications:

- **Text Classification:** Use the document vectors as input features for a classifier (e.g., for sentiment analysis or topic categorization).
- **Semantic Similarity:** Find similar documents by calculating the cosine similarity between their document vectors. This is great for recommendation engines or plagiarism detection.
- **Information Retrieval:** Given a search query (as a document), you can find the most relevant documents in a large database by finding the vectors closest to the query vector.

Doc2Vec in Practice & Applications

Training Phase

- We provide a large corpus of documents.
- The model learns the word vectors (W) and the document vectors (D) for **all documents in the training corpus** simultaneously.
- The result is a trained model and a vector for each document in the training set.

Inference Phase (For a NEW, Unseen Document)

How do we get a vector for a document the model has never seen? We can't just look it up.

1. **Freeze Model Weights:** We take our pre-trained model and **freeze** the word vectors (W) and the neural network's internal weights. They will not be updated.
2. **Assign New Vector:** We assign a new, randomly initialized document vector (Dnew) to the new document.
3. **Infer the Vector:** We "train" the model again, but **only the new document vector (Dnew) is updated**. Through a few iterations of gradient descent, this vector shifts to a position in the vector space that minimizes the prediction error for the words in the new document.
4. **Result:** The converged vector Dnew is the embedding for the new document.

Doc2Vec in Practice & Applications

Modern Context: Today, for many state-of-the-art results, Doc2Vec has been surpassed by large, pre-trained **Transformer models** like BERT. However, Doc2Vec is still a fantastic tool: it's computationally much cheaper, faster to train, and remains a very strong baseline. Understanding it is fundamental to understanding the evolution of NLP.

Key Takeaways-I

Core Problem: Machine learning models require numerical input, not raw text. Our goal is to convert text into vectors that capture meaning.

Two Philosophies:

- **Frequency-Based (Sparse):** Methods like **BoW** and **TF-IDF** represent documents based on word counts. They are fast and effective baselines but fail to capture semantic meaning or word order.
- **Prediction-Based (Dense):** Methods like **Word2Vec**, **GloVe**, and **FastText** learn compact, meaning-rich vectors by predicting words from their context. These *embeddings* place semantically similar words close together in vector space.

Key Takeaways-II

A quick guide to the models we discussed:

- **Word2Vec:** The pioneer. Learns from local context windows. Skip-gram is great for rare words; CBOW is faster. A fantastic general-purpose choice.
- **GloVe:** Combines counting and prediction. Learns from a global word-word co-occurrence matrix. It's very fast to train and excels at word analogy tasks.
- **FastText:** Learns vectors for sub-words (character n-grams) instead of whole words. Its key advantage is the ability to create vectors for **out-of-vocabulary (OOV) words** and its strength in morphologically rich languages.

Rule of Thumb: Always start with a high-quality pre-trained model (like GloVe or FastText) before training your own.

Key Takeaways-III

The Challenge: How do you represent a whole sentence or document?

- **Simple Baselines:**
  - **Averaging word vectors** is a fast and simple start.
  - Its fatal flaw is that it **completely ignores word order**, giving "man bites dog" and "dog bites man" the same vector.
- **A Sophisticated Solution: Doc2Vec**
  - Doc2Vec extends Word2Vec to learn document vectors directly.
  - It treats the **entire document as a special context word** that acts as a "memory" for the document's topic.
  - It successfully captures word order (in the PV-DM model) and produces superior, fixed-length vectors for variable-length texts.

Next Class:

NLP Applications