Big Data Computing

Master's of Science in Computer Science

Sapienza University of Rome

2026/2027

Gabriele Tolomei

tolomei@di.uniroma1.it

Module 2: High-Dimensional Data Representations

Lecture 2.3: Vector Representations of Text & Embeddings

Philosophy of Distributional Meaning

> *"You shall know a word by the company it keeps."*
> *— J. R. Firth (1957)*

Lectures 2.1–2.2 treated data as mathematical vectors. This lecture asks a more basic question — how does raw, symbolic text become a vector at all, and what happens to Lecture 2.1's geometry when it does?

Lecture Roadmap

This lecture examines how *unstructured text* is transformed into vector representations suitable for large-scale computation, and the scalability trade-offs this transformation entails. Specifically, we will study:

1. **Text as High-Dimensional Data** — framing the problem
2. **Sparse Representations** — one-hot encoding and TF-IDF
3. **Scalability Limits of Sparse TF-IDF Spaces** — where the curse of dimensionality bites
4. **The Distributional Hypothesis** — meaning via context
5. **From Sparse to Dense: Word Embeddings** — a fixed, compact alternative
6. **The Word2Vec Formulation** — Skip-gram, softmax, and negative sampling
7. **Training Word2Vec at Scale** — making web-scale training tractable
8. **Tokenization at Scale** — the prerequisite step, revisited
9. **Synthesis** — assembling the full argument

> *Every representation in this lecture is a different answer to one question: how much structure must we impose on text before a machine can compute with it?*

Why This Matters: The First Unstructured Modality

Lectures 2.1–2.2 established, respectively, the geometry of high-dimensional spaces and a tool (JL) for reducing dimensionality while preserving structure.

- Text is the first genuinely **unstructured** modality in this course — there is no natural schema to a sentence.
- Yet once text is mapped into vectors, everything from Lectures 2.1–2.2 applies directly: distance concentration, dimensionality reduction, and the trade-off between exact and approximate representations.

This lecture is where Module 2's mathematical machinery first meets real, messy, web-scale data.

§ 1 — Text as High-Dimensional Data

Why Text Must Become Vectors

- Machine learning models, similarity search, and clustering algorithms all operate on **numerical vectors** — not raw symbolic strings.
- To apply any tool from this course to text, we need a systematic, well-defined map from documents or words into $\mathbb{R}^d$.
- This lecture's central question: what should that map look like, and how does it scale to web-sized corpora?

The Vocabulary as a Coordinate System

Fix a **vocabulary** $V = \{w_1, w_2, \ldots, w_{|V|}\}$: the set of all distinct terms appearing in a corpus.

- Each dimension of the resulting vector space corresponds to exactly **one** vocabulary term.
- A document (or a word) is represented as a point/vector in $\mathbb{R}^{|V|}$ — the vocabulary itself *defines* the ambient dimensionality $d = |V|$.

The Bag-of-Words (BoW) Model

Represent a document purely by the **counts** of the words it contains, discarding order entirely:

$$\mathbf{x}_d \in \mathbb{R}^{|V|}, \qquad x_{d,i} = \text{count of } w_i \text{ in document } d$$

*"The dog bit the man" and "The man bit the dog" produce **identical** BoW vectors — word order carries no information in this model.*

Diagram: Document as a Vector Over Vocabulary

```
Vocabulary:  [the, dog, bit, man, cat, ran, ...]  (dimensions)

Document: "the dog bit the man"

Vector:      [ 2 ,  1 ,  1 ,  1 ,  0 ,  0 , ... ]
              the  dog  bit  man  cat  ran
```

Every distinct vocabulary term is a coordinate axis; the document is a single point whose coordinates are its word counts.

One-Hot Encoding: The Simplest Word Vector

A single word w_i can be represented as the standard basis vector $\mathbf{e}_i \in \{0,1\}^{|V|}$: a 1 in position i, 0 elsewhere.

$$\mathbf{e}_i \cdot \mathbf{e}_j = 0 \quad \text{for all } i \neq j, \qquad \|\mathbf{e}_i\|_2 = 1 \text{ for all } i$$

Every pair of distinct one-hot word vectors is exactly orthogonal — this representation encodes ***no notion of similarity whatsoever*** *between any two distinct words.*

Vocabulary Size in Practice: Web-Scale Corpora

- English Wikipedia alone contains on the order of a few **million** unique tokens (including names, numbers, and misspellings).
- Web-scale crawls (e.g., Common Crawl) push this into the tens of millions of distinct tokens.
- The ambient dimension $d = |V|$ for realistic text corpora is not a modest constant — it is routinely large enough for Lecture 2.1's asymptotic results to become directly relevant.

Connecting Back to Lecture 2.1: Text as a Concrete Instance of High d

Recall Lecture 2.1's **Theorem (Distance Concentration):** as $d \to \infty$, pairwise Euclidean distances concentrate, and the relative discriminative power of distance metrics vanishes.

- One-hot and bag-of-words text vectors, with $d = |V|$ in the hundreds of thousands to millions, sit squarely in this asymptotic regime.
- *Open question for this lecture:* does a naive counting-based text representation actually suffer this curse in practice — and if so, what alternative avoids it?

The Bag-of-Words Assumption: What Is Lost

- **Word order** is discarded entirely — "not good" and "good not" yield identical vectors.
- **Polysemy** (multiple meanings per word form, e.g., "bank" as riverbank vs. financial institution) is not disambiguated — one vector per word form, regardless of context.
- Despite these limitations, BoW and TF-IDF (Section 2) remain foundational, computationally cheap baselines still used widely in production search and retrieval systems.

✅ Checkpoint 1 — Text as High-Dimensional Data

Let's test our basic understanding:

1. Why must text be converted into vectors before most algorithms in this course can operate on it?
2. What does it mean for the vocabulary to define the "coordinate system" of the text vector space?
3. Why are all one-hot word vectors mutually orthogonal, and what does this imply about the similarity information they encode?
4. Connect the typical vocabulary size of web-scale corpora to Lecture 2.1's regime of interest for the curse of dimensionality.

Write down your thoughts before moving to Section 2.

§ 2 — Sparse Representations

From Counts to Weighted Counts: Motivating TF-IDF

Raw bag-of-words counts overweight frequent but largely uninformative words ("the," "is," "of").

- We need a weighting scheme that favors terms that are frequent **within** a specific document, but rare **across** the corpus as a whole.
- **Term Frequency–Inverse Document Frequency (TF-IDF)** is the classical solution to exactly this problem.

Term Frequency (TF)

For a term w and document d, define the normalized **term frequency**:

$$\mathrm{TF}(w, d) = \frac{\mathrm{count}(w, d)}{\sum_{w' \in d} \mathrm{count}(w', d)}$$

Normalizing by document length prevents longer documents from trivially dominating shorter ones purely due to having more words.

Why Raw Counts Are Misleading

- Function words ("the," "a," "of") appear in nearly every document with high frequency, yet carry almost no discriminative meaning about a document's specific topic.
- A useful weighting scheme must **penalize** terms that are common across the *entire* corpus, not merely reward terms that are frequent within one document.

Document Frequency and Inverse Document Frequency (IDF)

Let N be the total number of documents in the corpus, and $\mathrm{df}(w)$ the number of documents containing w at least once. Define:

$$\mathrm{IDF}(w) = \log\left(\frac{N}{\mathrm{df}(w)}\right)$$

- $\mathrm{IDF}(w)$ is **large** for rare terms (small $\mathrm{df}(w)$), and approaches **zero** for terms appearing in nearly every document.

The Intuition Behind IDF

- If $\mathrm{df}(w) = N$ (the word appears in every document): $\mathrm{IDF}(w) = \log(1) = 0$ — it contributes nothing to any TF-IDF vector.
- If $\mathrm{df}(w) = 1$ (the word appears in exactly one document): $\mathrm{IDF}(w) = \log(N)$ — its maximal possible weight, since this term is a strong discriminative signal for that single document.

Combining TF and IDF: The Full Formula

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

The final document vector is $\mathbf{x}_d \in \mathbb{R}^{|V|}$, with $x_{d,i} = \text{TF-IDF}(w_i, d)$ — a weighted refinement of the raw bag-of-words vector from Section 1.

Worked Numeric Example

Consider a corpus of $N = 1{,}000{,}000$ documents:

- Word **"the"**: $\mathrm{df}(\mathrm{the}) = 950{,}000$, so $\mathrm{IDF}(\mathrm{the}) = \log(1{,}000{,}000 / 950{,}000) \approx \log(1.053) \approx 0.051$.
- Word **"quasar"**: $\mathrm{df}(\mathrm{quasar}) = 12$, so $\mathrm{IDF}(\mathrm{quasar}) = \log(1{,}000{,}000 / 12) \approx \log(83{,}333) \approx 11.33$.

A roughly 220-fold difference in IDF weight — "quasar" carries vastly more discriminative signal than "the," exactly as intended.

TF-IDF Variants: Smoothing and Sublinear Scaling

Practical implementations commonly use refined variants for numerical robustness:

- **Smoothed IDF:** $\mathrm{IDF}(w) = \log\left(\frac{N}{1+\mathrm{df}(w)}\right) + 1$, avoiding division by zero for previously unseen terms.
- **Sublinear TF scaling:** $\mathrm{TF}(w, d) = 1 + \log(\mathrm{count}(w, d))$, dampening the effect of extremely high in-document repetition.

These variants refine the formula for stability; the underlying conceptual framework — rare terms matter more — is unchanged.

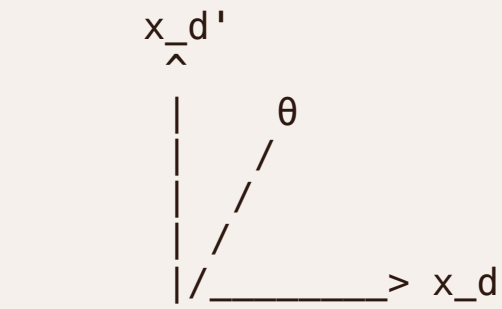
TF-IDF Vector Space & Cosine Similarity

Since TF-IDF vectors vary in magnitude with document length, similarity is typically measured via **cosine similarity** rather than raw Euclidean distance:

$$\cos(\theta) = \frac{\mathbf{x}_d \cdot \mathbf{x}_{d'}}{\|\mathbf{x}_d\|_2 \, \|\mathbf{x}_{d'}\|_2}$$

Cosine similarity normalizes out document length, focusing purely on the directional (compositional) similarity between two documents.

Diagram: Cosine Similarity Between Two Sparse Vectors



cos(θ) close to 1 → vectors point in nearly the same direction (similar)
cos(θ) close to 0 → vectors nearly orthogonal (dissimilar / no shared terms)

Sparsity of TF-IDF Vectors

- A typical document contains a few hundred to a few thousand **distinct** terms — while $|V|$ may be in the millions.
- The fraction of nonzero entries (nnz) relative to $|V|$ is typically well under 1%:

$$\text{Sparsity}(\mathbf{x}_d) = 1 - \frac{\text{nnz}(\mathbf{x}_d)}{|V|} \gtrsim 0.999$$

TF-IDF vectors are, in practice, extremely sparse — almost every coordinate is exactly zero.

Efficient Sparse Storage

- Storing a million-dimensional vector densely would be enormously wasteful — nearly all entries are zero.
- Real systems store only **(index, value) pairs** for nonzero entries: *store only what carries information.*
- This sparse representation ensures memory consumption scales with the number of non-zero entries rather than the full vocabulary dimension $|V|$.

✅ Checkpoint 2 — Sparse Representations

Let's test our understanding:

1. Derive the TF-IDF formula, and explain the intuition behind each of its two factors.
2. Compute the IDF for a word appearing in 500 out of 2,000,000 documents.
3. Why is cosine similarity generally preferred over raw Euclidean distance for comparing TF-IDF vectors?
4. Why does sparse (index, value) storage matter for TF-IDF vectors at web scale?

Discuss with your colleague or write down your answers.

§ 3 — Scalability Limits of Sparse TF-IDF Spaces

Web-Scale Vocabularies: How Large Does d Get?

- English Wikipedia: on the order of a few million unique tokens.
- Web-scale crawls (e.g., Common Crawl): tens of millions of distinct tokens, across many languages.
- Recall Lecture 2.1's regime of interest — "d large enough for asymptotic effects to dominate" — TF-IDF spaces routinely reach $d = 10^6$–10^7, comfortably inside this regime.

The Curse of Dimensionality Revisited for Text

Recall Lecture 2.1's **Theorem (Distance Concentration):** pairwise distances in $\mathbb{R}^d$ concentrate as $d \to \infty$; relative discriminative power vanishes.

- Sparse TF-IDF vectors formally live in $\mathbb{R}^{|V|}$ — even though almost all coordinates of any *one* vector are zero, pairwise comparisons still nominally occur in this enormous ambient space.
- *The question this section resolves: does sparsity change the nature of the curse, or merely relocate it?*

Cosine Similarity Degradation in High-Dimensional Sparse Spaces

Because documents are sparse, two documents typically share very few nonzero dimensions. The expected overlap between two random sparse documents scales as:

$$\mathbb{E}[\text{overlap}] \propto \frac{\text{nnz}(\mathbf{x}_d) \cdot \text{nnz}(\mathbf{x}_{d'})}{|V|}$$

As $|V|$ grows, expected overlap shrinks toward zero — even topically related documents can appear artificially dissimilar under raw term overlap. This is a distinct, sparsity-driven pathology, related to but not identical to Lecture 2.1's dense-vector concentration.

Sparsity Does Not Mean "Cheap": Storage Cost at Scale

- A single sparse vector may be cheap to store — but a corpus of 10^9 documents, each with a few hundred nonzero entries out of $|V| = 10^7$, still requires storing on the order of 10^{11}–10^{12} total (index, value) pairs.
- The **vocabulary mapping itself** (assigning consistent dimension indices to terms) must be maintained coherently across a distributed system — a nontrivial coordination problem in its own right.

Computational Cost of Similarity Search Over Sparse Vectors

- Computing all pairwise cosine similarities across n documents costs $O(n^2)$ vector comparisons — intractable for large n, regardless of individual vector sparsity.
- Even a single nearest-document query against a large corpus requires comparing against sparse representations of every candidate — motivating either specialized index structures (inverted indices, Lecture 3.1) or reduced dense representations (this lecture, Section 5 onward).

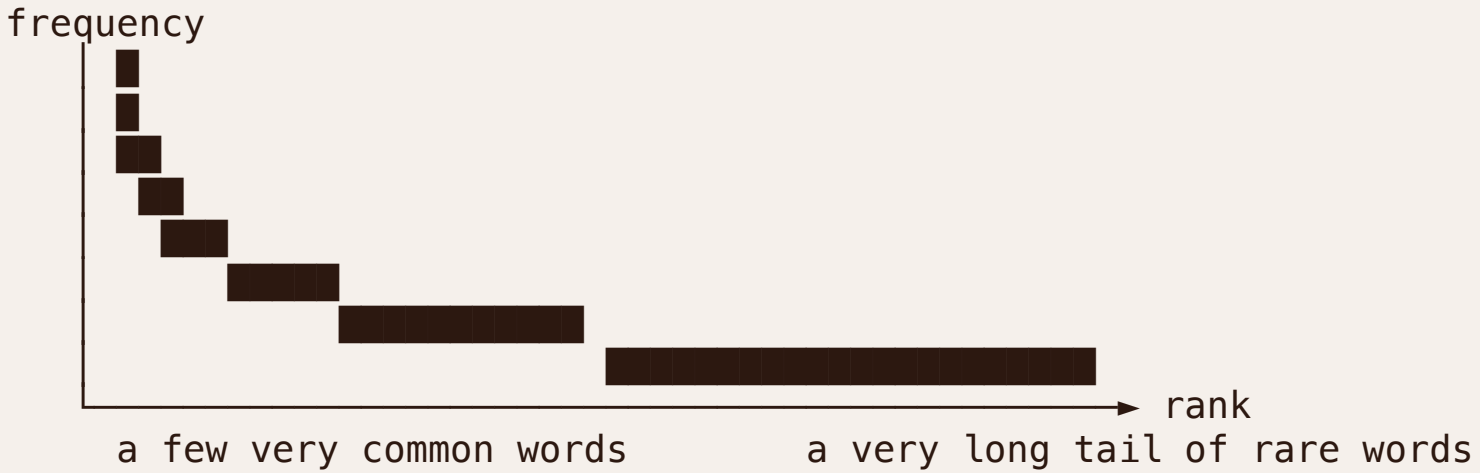
The Vocabulary Growth Problem: Zipf's Law

Empirically, word frequencies in natural language follow a striking regularity:

$$\text{frequency}(w) \propto \frac{1}{\text{rank}(w)}$$

- A small number of words ("the," "of," "and") account for a disproportionate share of all word occurrences.
- A very long **tail** of rare words each occur only once or twice in the entire corpus (so-called *hapax legomena*).

Diagram: Zipf's Law



Why Adding More Data Makes Vocabulary Bigger, Not Smaller

Heaps' Law (empirical): vocabulary size grows roughly as $|V| \approx Kn^{\beta}$ for a corpus of n tokens, with $\beta \in (0, 1)$ typically around 0.4–0.6.

- Unlike many statistical quantities that stabilize with more data, vocabulary size **keeps growing** as more text is observed — new rare words, names, and typos are continually introduced.
- This means $d = |V|$ is not merely large — it is **unbounded and growing** with corpus size, a genuine, ongoing scalability concern for sparse representations.

Case Study: Web Search Engines and Vocabulary Explosion

- Web-scale search engines index billions of documents; a naive TF-IDF vocabulary must accommodate every distinct token appearing anywhere on the crawled web — including misspellings, product codes, and every represented language.
- This is a concrete, practical instance of this section's scalability concerns, and is precisely why Module 3 (inverted indices, Lecture 3.1) revisits web-scale document representation from a systems and indexing perspective.

The Case for Dimensionality Reduction on Text

Sparse TF-IDF representations face two compounding scalability problems:

1. An enormous, **ever-growing** ambient dimension $|V|$ (Heaps' Law).
2. Expensive full-corpus similarity computation, further degraded by sparsity-induced near-orthogonality.

*This motivates seeking a **fixed, modest-dimensional** representation ($d \sim$ hundreds) that still captures meaningful semantic similarity — the subject of the remainder of this lecture.*

✅ Checkpoint 3 — Scalability Limits of Sparse TF-IDF Spaces

Let's test our understanding:

1. Why does the expected overlap between two random sparse document vectors shrink as $|V|$ grows?
2. Explain Zipf's Law and Heaps' Law, and why together they imply an ever-growing vocabulary.
3. Why can sparse vectors be individually cheap yet collectively very expensive to store and search at corpus scale?
4. What two compounding scalability problems motivate moving away from sparse TF-IDF representations?

Review your answers before moving on.

§ 4 — The Distributional Hypothesis

Firth's Dictum

> *"You shall know a word by the company it keeps."*
> *— J. R. Firth (1957)*

- This is the **distributional hypothesis**: words that occur in similar linguistic contexts tend to have similar meanings.
- It is the conceptual foundation for essentially all modern word-embedding methods, including the Word2Vec formulation developed in Section 6.

Formalizing the Idea: Co-occurrence as a Proxy for Meaning

Rather than defining a word's meaning directly, define it **operationally**: via the distribution of words that tend to surround it in text.

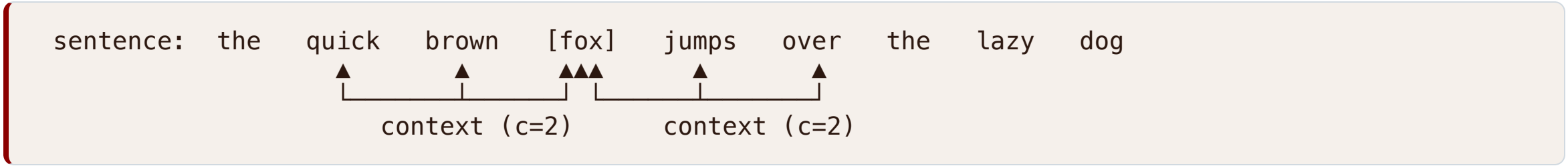
- Two words with similar surrounding-word distributions — e.g., "dog" and "cat," both appearing near "pet," "leash/litter," "bark/meow" — are hypothesized to be semantically related.
- This reframes "meaning" as a measurable, corpus-derived statistical property, rather than an unobservable mental concept.

The Context Window, Defined

Fix a window size c (e.g., $c = 2$). For a target word w_t at position t in a sequence, its **context** is the set of words within c positions on either side:

$$\text{context}(w_t) = \{w_{t-c}, \ldots, w_{t-1},\ w_{t+1}, \ldots, w_{t+c}\}$$

Diagram: Context Window Around a Target Word



For $c = 2$, the target word "fox" has context $\{\text{quick}, \text{brown}, \text{jumps}, \text{over}\}$.

Worked Example: Extracting Context Pairs from a Sentence

Sentence: "*the quick brown fox jumps over the lazy dog*", with $c = 2$ and target word "fox":

$$\text{context}(\text{fox}) = \{\text{quick}, \text{brown}, \text{jumps}, \text{over}\}$$

This yields four (target, context) training pairs: (fox, quick), (fox, brown), (fox, jumps), (fox, over) — exactly the kind of pairs Section 6's Skip-gram model is trained to predict.

Why Co-occurrence Captures Semantic and Syntactic Similarity

- **Semantically** similar words (synonyms, related concepts) tend to appear in overlapping contexts, since they are used to describe similar situations.
- **Syntactically** similar words (e.g., different verb forms, plural nouns) also share distributional patterns, since grammar constrains which words can plausibly appear near which others.

From Counting Co-occurrences to Learning Embeddings

- Classical distributional semantics methods explicitly build a co-occurrence **count matrix** and then apply dimensionality reduction (e.g., SVD — recall Lecture 2.2's comparative note on PCA/SVD) to obtain compact vectors.
- Word2Vec instead **learns** dense vectors via a predictive objective — predicting context words from target words — without ever materializing the full co-occurrence matrix explicitly. This is the subject of the next two sections.

✅ Checkpoint 4 — The Distributional Hypothesis

Let's test our understanding:

1. State the distributional hypothesis in your own words, and explain how it operationalizes "meaning."
2. Define the context window formally, and extract the context of a word of your choosing from a sentence with $c = 3$.
3. Give one example each of semantic and syntactic similarity captured by co-occurrence patterns.
4. How does Word2Vec's approach to using co-occurrence differ from classical count-matrix-plus-SVD methods?

Consolidate your answers before proceeding.

§ 5 — From Sparse to Dense: Word Embeddings

The Dense Embedding Idea

Instead of one dimension per vocabulary word, represent each word as a **dense** vector $\mathbf{v}_w \in \mathbb{R}^d$, with d typically in the range of 50–300 — orders of magnitude smaller than $|V|$.

- Every dimension of $\mathbf{v}_w$ is, in general, nonzero — hence "dense," in direct contrast to Section 2's sparse TF-IDF vectors.

Dimensionality Contrast: Sparse vs. Dense

	Sparse (TF-IDF)	Dense (Word2Vec)
Dimension d	$\lvert V\rvert \sim 10^6$–10^7	Fixed, ~ 50–300
Nonzero entries	A handful per vector	Essentially all
Interpretability	Each dimension = one word	No single dimension is individually meaningful
Similarity signal	Requires shared vocabulary terms	Learned semantic proximity

Diagram: Sparse One-Hot Vector vs. Dense Embedding Vector

```
One-hot ("cat"), |V| = 1,000,000:
[0,0,...,0,1,0,...,0]           ← exactly one nonzero, dimension = 10^6

Dense embedding ("cat"), d = 100:
[0.12,-0.87,0.33,...,0.05,-0.61] ← all 100 dimensions populated
```

What Dense Dimensions "Mean": Distributed Representation

- Unlike one-hot vectors, no single dimension of a dense embedding corresponds to an interpretable, named feature.
- Meaning is encoded in the **overall pattern** across all d dimensions jointly — a **distributed representation**, in exactly the sense invoked by the title of Mikolov et al. (2013): *Distributed Representations of Words and Phrases.*

Semantic Geometry: Similarity as Proximity

Words with related meanings are mapped to **nearby** points in $\mathbb{R}^d$, typically measured via cosine similarity — exactly as in Section 2, but now in a compact, dense space rather than a sparse, enormous one:

$$\cos(\mathbf{v}_{w_1}, \mathbf{v}_{w_2}) = \frac{\mathbf{v}_{w_1} \cdot \mathbf{v}_{w_2}}{\|\mathbf{v}_{w_1}\|_2 \, \|\mathbf{v}_{w_2}\|_2}$$

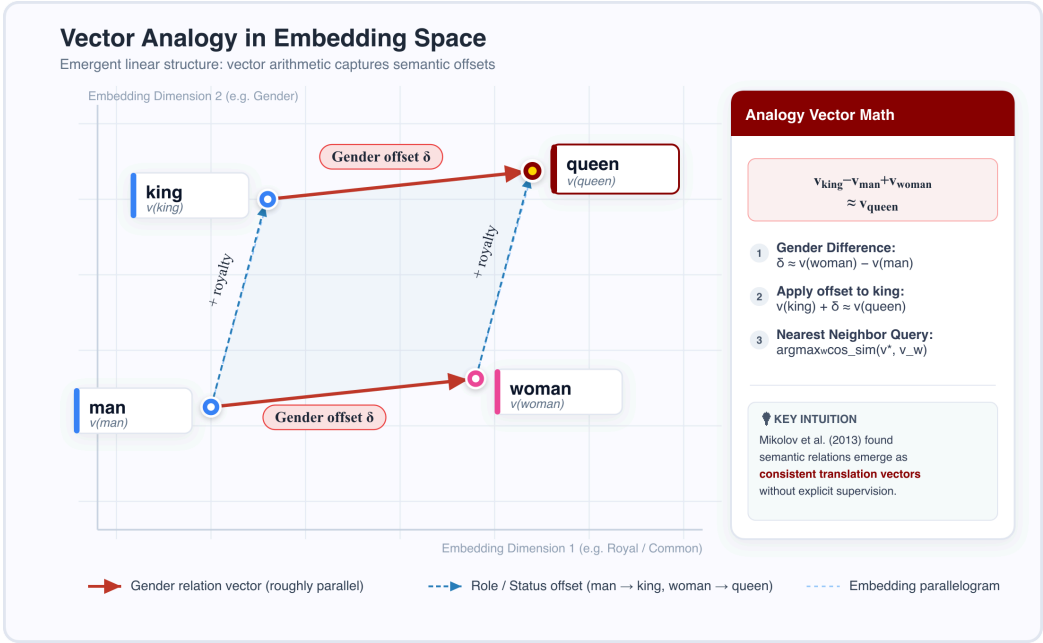
The Famous Analogy Arithmetic Example

A striking empirical property of learned word embeddings, prominently highlighted in Mikolov et al. (2013):

$$\mathbf{v}_{\text{king}} - \mathbf{v}_{\text{man}} + \mathbf{v}_{\text{woman}} \approx \mathbf{v}_{\text{queen}}$$

Linear vector arithmetic in embedding space appears to capture certain semantic and syntactic relationships — an emergent property of training, not an explicitly imposed design goal.

Diagram: Embedding Space Analogy Illustration



The "king → queen" and "man → woman" offsets are approximately parallel vectors in embedding space — illustrating that the direction encoding "gender" is roughly consistent across related word pairs.

More Analogy Examples

Relationship	Vector arithmetic
Gender	$\mathbf{v}_{\text{king}} - \mathbf{v}_{\text{man}} + \mathbf{v}_{\text{woman}} \approx \mathbf{v}_{\text{queen}}$
Capital city	$\mathbf{v}_{\text{Paris}} - \mathbf{v}_{\text{France}} + \mathbf{v}_{\text{Italy}} \approx \mathbf{v}_{\text{Rome}}$
Verb tense	$\mathbf{v}_{\text{walking}} - \mathbf{v}_{\text{walk}} + \mathbf{v}_{\text{swim}} \approx \mathbf{v}_{\text{swimming}}$

These emergent relationships were among the most widely cited empirical findings of Mikolov et al. (2013), sparking substantial follow-up research into embedding geometry.

✅ Checkpoint 5 — From Sparse to Dense: Word Embeddings

Let's test our understanding:

1. Contrast sparse and dense word representations along dimensionality, sparsity, and interpretability.
2. What does it mean for a word embedding to be a "distributed representation"?
3. State the king/man/woman/queen analogy, and explain what property of the embedding space it demonstrates.
4. Why is cosine similarity still the natural similarity measure in the dense setting, just as in the sparse one?

Consolidate your answers before proceeding.

§ 6 — The Word2Vec Formulation

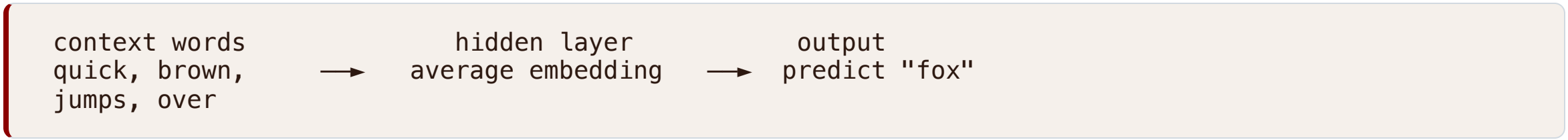
Two Architectures: CBOW and Skip-gram

Mikolov et al. (2013) propose two closely related architectures, both learning dense word vectors as a byproduct of a simple predictive task:

- **CBOW (Continuous Bag-of-Words):** predict the target word from its surrounding context words.
- **Skip-gram:** predict the surrounding context words from the target word.

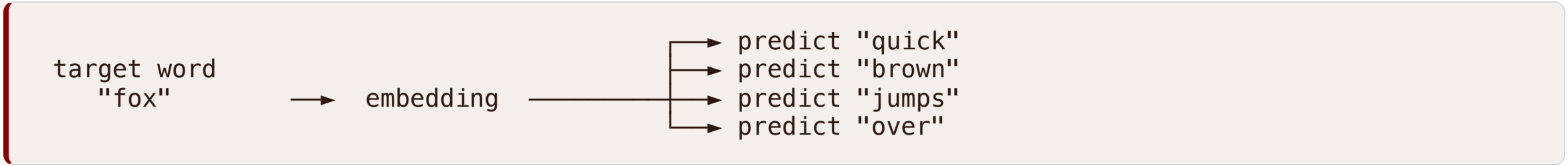
Neither architecture ever builds an explicit co-occurrence matrix — both learn vectors directly via a prediction objective.

Diagram: CBOW Architecture



CBOW averages the embeddings of all context words to predict the single missing target word.

Diagram: Skip-gram Architecture



Skip-gram uses the single target word's embedding to predict each context word independently.

Why This Lecture Focuses on Skip-gram

- Skip-gram tends to perform better on rare words and larger datasets, per Mikolov et al.'s empirical findings.
- CBOW and Skip-gram share the same underlying vector-learning mechanics; understanding one transfers directly to the other.

Formal Setup

- The corpus is a sequence of words $w_1, w_2, \ldots, w_T$.
- Fix a context window size c. For each position t, the context words are w_{t+j} for $j \in \{-c, \ldots, -1, 1, \ldots, c\}$.
- **Goal:** learn word vectors that make observed (target, context) pairs likely under the model.

The Skip-gram Objective Function

The Skip-gram model seeks parameters θ (word vectors) that maximize the average log probability of observed context words given target words:

$$J(\theta) = \frac{1}{T} \sum_{t=1}^{T} \sum_{\substack{-c \leq j \leq c \\ j \neq 0}} \log p(w_{t+j} \mid w_t;\, \theta)$$

Modeling $p(w_{t+j} \mid w_t)$ via Softmax

Associate with each word w two vectors: an **input** vector $\mathbf{v}_w$ (used when w is the target) and an **output** vector $\mathbf{v}'_w$ (used when w is a context word). Then:

$$p(w_O \mid w_I) = \frac{\exp\left(\mathbf{v}'^{\top}_{w_O} \mathbf{v}_{w_I}\right)}{\sum_{w=1}^{|V|} \exp\left(\mathbf{v}'^{\top}_{w} \mathbf{v}_{w_I}\right)}$$

The softmax converts raw dot-product "compatibility" scores into a valid probability distribution over the entire vocabulary.

Word Vectors as Model Parameters

- The dense embeddings we ultimately want — the input vectors $\mathbf{v}_w$ — are literally the learned parameters of this simple prediction task.
- After training, the output vectors $\mathbf{v}'_w$ are typically discarded, and $\mathbf{v}_w$ is used as "the" word embedding in downstream applications.

Continuous vs. Discrete: Why "Distributed Representations"?

Recall the paper's full title: *Distributed Representations of Words and Phrases and Their Compositionality.*

- "Distributed" reflects that meaning is spread across all d continuous dimensions **jointly** — not localized to any single discrete symbol or coordinate, as it was in Section 2's one-hot vectors.
- This is the direct conceptual contrast between the purely discrete representation of Section 1–2 and the continuous representation developed here.

The Softmax Cost Bottleneck

Every single training update requires computing the full normalization sum over the **entire vocabulary**:

$$\text{Cost per update} = O(|V|)$$

With $|V|$ in the millions (Section 3), and training corpora of billions of words, this per-update cost compounds to an infeasible total computational burden.

Why Full Softmax Is Intractable at Web Scale

- Corpora used in Mikolov et al. (2013) contain up to billions of training tokens; naive full-softmax training would require $O(|V|)$ work for **every single** (target, context) pair observed.
- This is precisely the kind of situation — "exact computation becomes intractable at scale" — motivating this entire course's approximate-methods philosophy, stated in the syllabus.

Hierarchical Softmax: A Brief Prior Remedy

One earlier proposed solution: organize the vocabulary of output words as a binary tree (e.g., a Huffman tree), and represent $p(w_O \mid w_I)$ (predicting context word w_O given target w_I) as a product of binary decisions traversing the tree from root to leaf:

$$\text{Cost per update} = O(\log |V|)$$

*Mikolov et al. (2013) note that **negative sampling**, developed next, is simpler and often performs better in practice — hence its emphasis as this paper's core algorithmic contribution.*

Negative Sampling: The Core Idea

Rather than normalizing over the **entire** vocabulary, reformulate the problem as a **binary classification task**: distinguish real (target, context) pairs from randomly sampled "negative" (target, random word) pairs.

*Intuition: instead of asking "what is the full probability distribution over all possible context words?", ask a much cheaper question repeatedly: "is **this specific** word a plausible context word, or not?"*

Deriving the Negative Sampling Objective

Let $\sigma(x) = \dfrac{1}{1 + e^{-x}}$ be the logistic sigmoid. For a true observed pair (w_I, w_O), the negative sampling objective for one training example is:

$$\log \sigma\left(\mathbf{v}'^{\top}_{w_O} \mathbf{v}_{w_I}\right) \; + \; \sum_{i=1}^{k} \mathbb{E}_{w_i \sim P_n(w)} \left[\log \sigma\left(-\mathbf{v}'^{\top}_{w_i} \mathbf{v}_{w_I}\right) \right]$$

- The first term pushes the true context word's vector **toward** alignment with the target.
- The second term, summing over k randomly sampled "negative" words, pushes their vectors **away** from the target.

Intuition: Why $\log \sigma(-\mathbf{v}'^{\top}_{w_{\text{neg}}} \mathbf{v}_w)$ Unaligns Negative Vectors

Recall that the sigmoid function satisfies the symmetry identity:

$$\sigma(-s) = 1 - \sigma(s)$$

Let $s = \mathbf{v}'^{\top}_{w_{\text{neg}}} \mathbf{v}_w = \|\mathbf{v}'_{w_{\text{neg}}}\| \|\mathbf{v}_w\| \cos(\theta)$ be the dot-product similarity (cosine compatibility).

- To **maximize** the objective, the model must **maximize** $\log \sigma(-s)$, which means maximizing $\sigma(-s) \to 1$.
- Because $\sigma(-s) \to 1 \iff -s \to +\infty \iff s \to -\infty$, the dot product $\mathbf{v}'^{\top}_{w_{\text{neg}}} \mathbf{v}_w$ is driven toward **negative values**.
- Geometrically, $\cos(\theta) < 0$ forces the angle $\theta > 90^\circ$: the vectors are actively pushed to become **orthogonal or opposite (unaligned)**.
- *Without this negative term, all embeddings would trivially collapse into a single point to maximize the positive term!*

Softmax vs. Negative Sampling: A Direct Comparison

Dimension	Full Softmax Formulation	Negative Sampling (SGNS)
Formulation	Multiclass classification over $\|V\|$ classes	$k+1$ independent binary logistic regressions
Probability	$p(w_O \mid w_I) = \frac{\exp(\mathbf{v}'^{\top}_{w_O}\mathbf{v}_{w_I})}{\sum_{w\in V}\exp(\mathbf{v}'^{\top}_{w}\mathbf{v}_{w_I})}$	$P(\text{real} \mid w_I, w_O) = \sigma(\mathbf{v}'^{\top}_{w_O}\mathbf{v}_{w_I})$
Normalization	Global partition function $Z = \sum_{w\in V} e^{\mathbf{v}'^{\top}_{w}\mathbf{v}_{w_I}}$	**None** (no partition function needed!)
Cost / Update	$O(\|V\| \cdot d)$ — touches every vocabulary word	$O(k \cdot d)$ — touches only target + k noise words
Theoretical Link	Multinomial likelihood maximization	Implicitly factorizes shifted Pointwise Mutual Information matrix (Levy & Goldberg)

Negative sampling transforms an intractable global competition into $k+1$ cheap local coin-flip decisions.

The Final Skip-gram with Negative Sampling (SGNS) Objective

$$J_{\text{SGNS}}(\theta) = \frac{1}{T}\sum_{t=1}^{T}\sum_{\substack{-c\le j\le c\\ j\neq 0}}\left[\log\sigma\left(\mathbf{v}'^{\top}_{w_{t+j}}\mathbf{v}_{w_t}\right) + \sum_{i=1}^{k}\mathbb{E}_{w_i\sim P_n(w)}\log\sigma\left(-\mathbf{v}'^{\top}_{w_i}\mathbf{v}_{w_t}\right)\right]$$

- Cost per update is now $O(k)$, **independent of** $|V|$ — with k typically 5–20.
- $P_n(w)$, the noise distribution, is typically the unigram distribution raised to the power $3/4$, empirically found by Mikolov et al. to outperform the plain unigram distribution.

How Word2Vec Is Trained: Stochastic Gradient Descent (SGD)

Word2Vec is trained end-to-end using **Stochastic Gradient Descent (SGD)** over streaming word windows:

1. **Streaming window passes:** Slide a window of radius c across the corpus token stream $w_1, w_2, \ldots, w_T$.
2. **For each target-context pair** (w_t, w_{t+j}) (target $w_I = w_t$, true context $w_O = w_{t+j}$):
   - Sample k negative words $w_{1,\text{neg}}, \ldots, w_{k,\text{neg}} \sim P_n(w)$ from $P(w)^{3/4}$.
   - The gradient of the loss for this pair with respect to target vector $\mathbf{v}_{w_t}$ is:

$$\nabla_{\mathbf{v}_{w_t}} = \Big(1 - \sigma(\mathbf{v}'^{\top}_{w_{t+j}} \mathbf{v}_{w_t})\Big)\mathbf{v}'_{w_{t+j}} - \sum_{i=1}^{k} \Big(1 - \sigma(-\mathbf{v}'^{\top}_{w_{i,\text{neg}}} \mathbf{v}_{w_t})\Big)\mathbf{v}'_{w_{i,\text{neg}}}$$

3. **Sparse in-place update:** Apply gradient ascent $\theta \leftarrow \theta + \eta \nabla_\theta$ **only** to the active vectors:

$$\{\mathbf{v}_{w_t}, \mathbf{v}'_{w_{t+j}}, \mathbf{v}'_{w_{1,\text{neg}}}, \ldots, \mathbf{v}'_{w_{k,\text{neg}}}\}$$

   - All other $|V| - k - 2$ word vectors in the vocabulary remain completely untouched!
   - Learning rate η decays linearly across training tokens.

✅ Checkpoint 6 — The Word2Vec Formulation

Let's test our understanding:

1. Contrast the CBOW and Skip-gram architectures in terms of what each predicts from what.
2. Write the full-softmax formula for $p(w_O \mid w_I)$, and explain why its cost is $O(|V|)$ per update.
3. Derive, in words, why the negative sampling objective reduces this to $O(k)$ per update.
4. What role does the noise distribution $P_n(w)$ play, and why is the $3/4$-power unigram distribution used in practice?

Consolidate your answers before proceeding.

§ 7 — Training Word2Vec at Scale

Cost Comparison: Full Softmax vs. Negative Sampling

Method	Cost per update	Dependence on $\lvert V \rvert$
Full softmax	$O(\lvert V \rvert)$	Direct, linear
Hierarchical softmax	$O(\log \lvert V \rvert)$	Logarithmic
Negative sampling	$O(k)$, $k \approx 5$–20	**None**

$$\text{Speedup factor} \approx \frac{|V|}{k}$$

For $|V| = 10^6$ and $k = 10$: roughly a 10^5-fold reduction in per-update computation.

Billion-Word Training Corpora in Practice

- Mikolov et al. (2013) train on corpora containing hundreds of millions to billions of words (e.g., large news-text collections).
- At this scale, even the $O(k)$-per-update cost of negative sampling, multiplied across billions of training pairs, requires substantial distributed compute — negative sampling makes each update cheap, not the training run trivial.

Subsampling of Frequent Words

A further practical technique from the paper: probabilistically discard very frequent words (e.g., "the," "a") during training, with discard probability:

$$P(\text{discard } w) = 1 - \sqrt{\frac{t}{f(w)}}$$

where $f(w)$ is w's corpus frequency and t is a chosen threshold.

This reduces the number of largely uninformative training pairs involving extremely common words, speeding up training and improving vector quality for rarer, more informative words.

Data-Parallel Training: Sharding the Corpus Across Workers

- The training corpus — a sequence of billions of words — is naturally splittable into disjoint chunks, distributed across worker machines, directly analogous to MapReduce input splits (Lecture 1.2).
- Each worker computes gradient updates for its local chunk of (target, context, negative-sample) triples, using the SGNS objective from Section 6.

Asynchronous Parameter Updates: Forward Callback

- The shared word-vector parameters ($\mathbf{v}_w$, $\mathbf{v}'_w$ for every $w \in V$) must be updated consistently across many workers operating in parallel.
- This is precisely the setting formalized later in **Module 4**: **Hogwild!** (asynchronous, lock-free shared-memory SGD, Lecture 4.3) and the **Parameter Server** architecture (Lecture 4.4) — both directly applicable to distributed Word2Vec training.

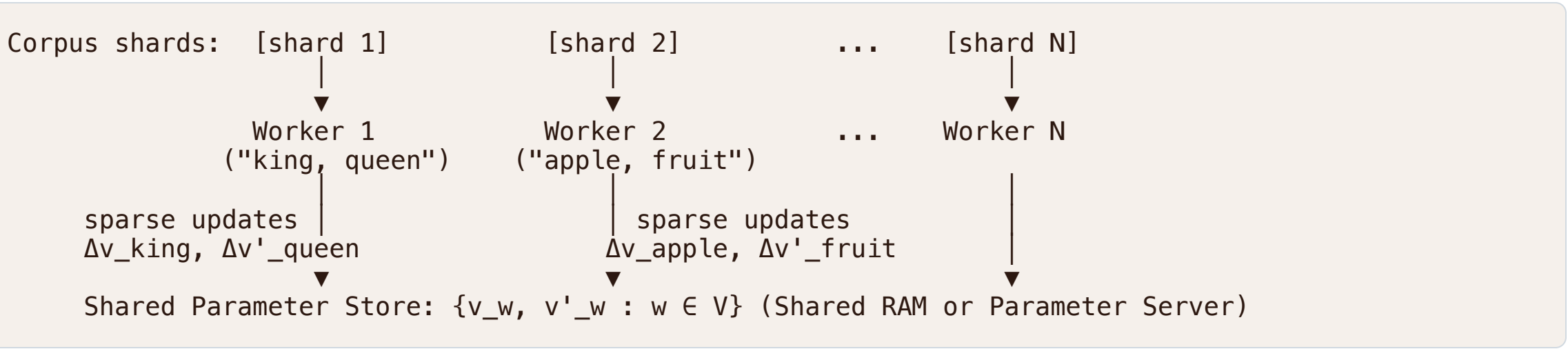
Why Negative Sampling Is What Makes Distributed Training Feasible

- Full softmax's $O(|V|)$ per-update cost would make each worker's individual gradient computation itself a severe bottleneck, compounding with cross-worker communication cost.
- Negative sampling's $O(k)$ cost keeps each update cheap, so the **dominant** remaining cost becomes coordinating shared parameter updates across workers — exactly the problem Module 4's Parameter Server architecture is designed to solve.

Empirical Scale vs. Quality Trade-off

- Mikolov et al. (2013) report that increasing both training data size and vector dimensionality generally improves embedding quality (as measured on analogy tasks), though with diminishing returns and rising computational cost.
- This exemplifies a recurring theme of this course: more data and larger models generally help, but the computation needed to exploit them at scale must itself be engineered — precisely the role of negative sampling and distributed training covered in this section.

Diagram: Distributed Word2Vec Training Pipeline



- **Global parameter store:** A single shared set of word vectors is maintained centrally (in shared memory or across a parameter server cluster).
- **Asynchronous SGD:** Workers stream through their local corpus shards in parallel, reading current vectors, computing SGNS gradients, and immediately applying updates $\mathbf{v}_w \leftarrow \mathbf{v}_w - \eta \nabla \mathcal{L}$ without locks.
- **Sparse updates prevent contention:** Because negative sampling touches only $\approx k + 2$ words per pair, workers rarely write to the same word vector at the exact same time (preview of *Hogwild!*, Module 4).

✅ Checkpoint 7 — Training Word2Vec at Scale

Let's test our understanding:

1. Quantify the speedup factor negative sampling provides over full softmax, for $|V| = 5 \times 10^6$ and $k = 10$.
2. Explain the intuition behind subsampling of frequent words, and why it improves training.
3. How does sharding the training corpus across workers parallel the MapReduce input-split model from Module 1?
4. Why does negative sampling specifically enable the distributed training architectures previewed for Module 4?

Make sure you understand these mechanisms before wrapping up this section.

§ 8 — Tokenization at Scale

From Raw Text to Tokens: The First Step of Any Pipeline

- Every representation in this lecture (TF-IDF, Word2Vec) assumes text has already been split into discrete units called **tokens** — this step, **tokenization**, is a prerequisite, not an afterthought.
- Tokenization choices directly determine the vocabulary size $|V|$ (Sections 2–3) and what counts as "a word" for embedding training (Sections 5–7).

Word-Level Tokenization and Its Limits

- The simplest approach: split text on whitespace and punctuation, treating each resulting string as one token.
- **Limitation:** any word not seen during training — a novel word, a misspelling, a rare proper noun — has **no vector at all** available at inference time.

The Out-of-Vocabulary (OOV) Problem

- Formally: given a fixed vocabulary V built from a training corpus, any token $w \notin V$ encountered later cannot be mapped to any embedding.
- This is a genuine scalability concern: language is productive — new words, names, and terms are continuously coined — so no finite fixed vocabulary can ever be truly complete.

Subword Tokenization: The Conceptual Fix

- Rather than treating whole words as the atomic unit, break words into smaller, reusable **subword units** — e.g., "unbelievable" → "un" + "believ" + "able".
- Any novel word can then be represented as a sequence of *known* subword pieces, dramatically reducing (though not entirely eliminating) the out-of-vocabulary problem.

Byte-Pair Encoding (BPE): The Iterative Algorithm

Adapted from data compression (Sennrich et al., 2016; used in GPT-2/4, LLaMA):

1. **Initialize vocabulary V:** Split training corpus into individual characters + end-of-word marker `_`.
   $V \leftarrow \text{all unique characters}.$
2. **Set target budget:** Choose K (desired number of merge operations).
3. **Repeat for $k = 1, \ldots, K$:**
   - Count frequencies of all adjacent symbol pairs across words in the corpus.
   - Find the most frequent pair $(A, B) = \arg\max \text{count}(A, B)$.
   - Create merged symbol AB; add to vocabulary: $V \leftarrow V \cup \{AB\}$.
   - In all corpus words, replace occurrences of adjacent `A B` with `AB`.
4. **Result:** Fixed vocabulary $|V| = |V_{\text{initial}}| + K$ and an **ordered list of K merge rules**.

BPE by Example: Step-by-Step Merge Walkthrough

Mini-corpus: `"low_": 5`, `"lower_": 2`, `"newest_": 6`, `"widest_": 3`
Initial tokens: `l o w _`, `l o w e r _`, `n e w e s t _`, `w i d e s t _` ($|V_0| = 11$)

Step	Pair (A, B)	Corpus Count	Action & Vocabulary Update
$k = 1$	`('e', 's')`	$6 + 3 = 9$	Add **`es`** $\rightarrow$ words have `... n e w es t _`, `... w i d es t _`
$k = 2$	`('es', 't')`	$6 + 3 = 9$	Add **`est`** $\rightarrow$ words have `... n e w est _`, `... w i d est _`
$k = 3$	`('est', '_')`	$6 + 3 = 9$	Add **`est_`** $\rightarrow$ words have `... n e w est_`, `... w i d est_`
$k = 4$	`('l', 'o')`	$5 + 2 = 7$	Add **`lo`** $\rightarrow$ words have `lo w _`, `lo w e r _`

Inference on unseen text: Replay the K merge rules in exact order:

- Unseen word `"lowest_"` $\rightarrow$ `l o w e s t _` $\xrightarrow{\text{rule } 1,2,3}$ `l o w est_` $\xrightarrow{\text{rule } 4}$ `['lo', 'w', 'est_']`
- No Out-of-Vocabulary (OOV) error: unmerged letters fall back to atomic characters!

WordPiece: Likelihood-Driven Merging & Inference

Used in **BERT** (Schuster & Nakajima 2012; Wu et al. 2016):

- **BPE limitation:** If characters `e` and `s` are ubiquitous individually, `('e', 's')` has high raw count *by pure chance*.
- **WordPiece merge criterion:** Maximizes the language model likelihood of the corpus:

$$\text{Score}(A, B) = \frac{\text{count}(AB)}{\text{count}(A) \times \text{count}(B)} \propto \frac{P(AB)}{P(A) \cdot P(B)}$$

  Pointwise Mutual Information (PMI): penalizes common characters in the denominator unless they appear together disproportionately.

- **Subword continuation syntax:** Suffix pieces receive `##` (e.g. `['un', '##believ', '##able']`).
- **Inference without merge rules:** Rather than replaying history, WordPiece uses greedy **longest-match-first** from left to right directly against the final vocabulary.

Tokenization Trade-offs Summary

Granularity	Handles OOV?	Vocabulary size	Sequence length
Word-level	No	Very large, unbounded	Short
Subword (BPE/WordPiece)	Mostly	Fixed, moderate	Medium
Character-level	Yes (fully)	Small, fixed	Very long

Subword tokenization is the practical compromise adopted by most modern large-scale text pipelines.

Tokenization as a Distributed Preprocessing Pipeline

- At web scale, tokenizing a multi-billion-word corpus is itself a data-parallel task: split the corpus into shards (Lecture 1.2's MapReduce splits), tokenize each shard independently, and aggregate token/vocabulary statistics.
- This preprocessing step, typically run via Spark or a MapReduce job (Lectures 1.2–1.3), precedes both TF-IDF construction (Section 2) and Word2Vec training (Sections 6–7) in any real production pipeline.

✅ Checkpoint 8 — Tokenization at Scale

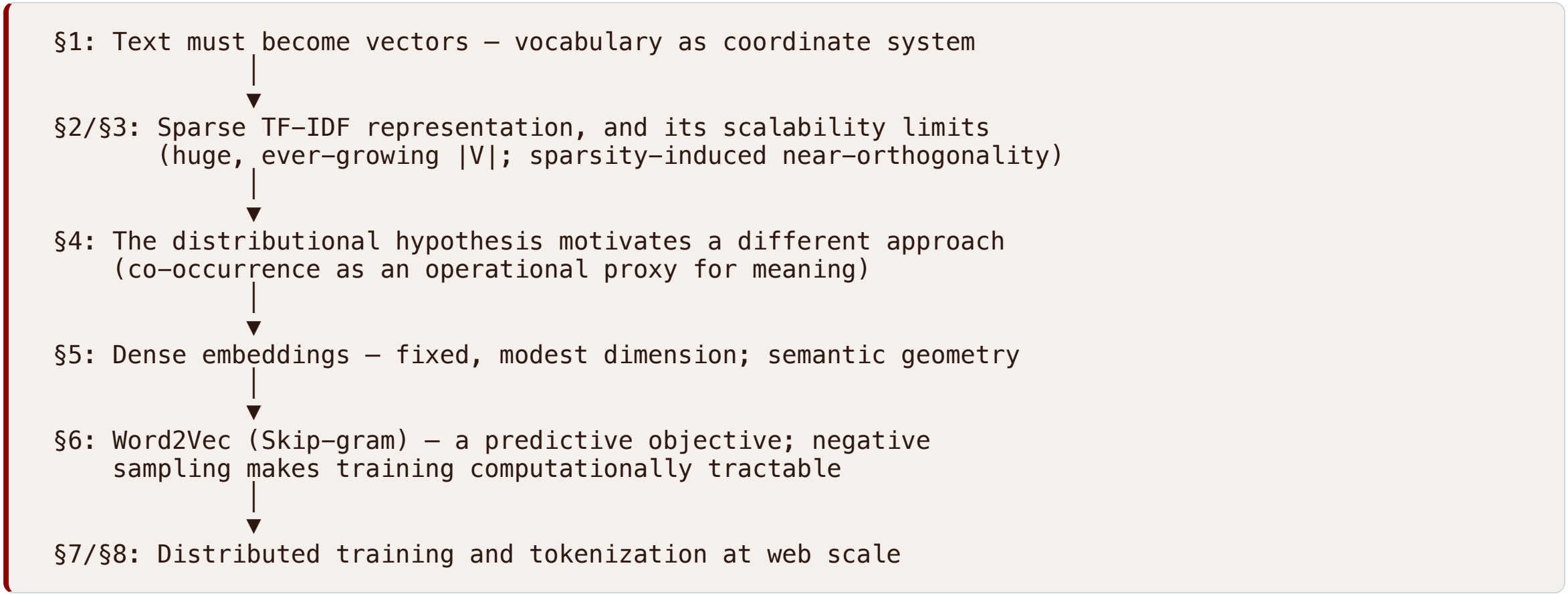
Let's test our understanding:

1. Why does tokenization granularity directly determine the vocabulary size $|V|$?
2. Explain the out-of-vocabulary problem, and why it specifically afflicts word-level tokenization.
3. Sketch the first two merge steps of Byte-Pair Encoding on a small toy corpus of your choosing.
4. Why is tokenization naturally implemented as a distributed MapReduce/Spark preprocessing job at scale?

Consolidate your answers before the final synthesis.

§ 9 — Synthesis & Wrap-up

The Chain of Results Recap Diagram



Summary of Key Results

This lecture established, in sequence:

1. Text can be represented as vectors over a fixed vocabulary — but naive sparse counting-based representations (TF-IDF) face a growing, extremely high-dimensional space.
2. The distributional hypothesis reframes word meaning operationally, via co-occurrence with context.
3. Word2Vec learns dense, low-dimensional embeddings via a simple predictive task, avoiding an explicit co-occurrence matrix.
4. Negative sampling reduces per-update training cost from $O(|V|)$ to $O(k)$, making web-scale training computationally feasible.
5. Tokenization granularity (word-level vs. subword) determines vocabulary size and the severity of the out-of-vocabulary problem.

Key Terms Glossary

Term	Meaning
Distributional hypothesis	Words with similar contexts have similar meanings
TF-IDF	Sparse term weighting balancing in-document and corpus-wide frequency
Distributed representation	Meaning encoded jointly across all dense embedding dimensions
Skip-gram	Predicting context words from a target word
Negative sampling	Reformulating softmax as cheap binary classification against sampled negatives
Byte-Pair Encoding (BPE)	Merge-based subword tokenization algorithm

From Sparse to Dense: A Side-by-Side Recap

	TF-IDF (Sections 2–3)	Word2Vec (Sections 5–7)
Dimension	$\lvert V \rvert$, large and growing	Fixed, ~ 50–300
Sparsity	Extremely sparse	Dense
How obtained	Direct counting formula	Learned via prediction objective
Main scalability lever	Sparse storage	Negative sampling

Lecture 2.3 Complete

Recommended Reading

- Mikolov, T., Sutskever, I., Chen, K., Corrado, G., & Dean, J. (2013). *Distributed representations of words and phrases and their compositionality*. In Proc. of NeurIPS.
- Course reference materials on TF-IDF, tokenization, and distributed word embeddings

Contact & Q&A

- **Course:** Big Data Computing
- **Office Hours:** Drop me a message, and we'll arrange a time to meet, either in person or online! 🙂

Thank you for your attention!