Predicting the Next Word

From Shannon to ChatGPT

Test Compilation - Chapter 4

Contents

Chapter 4

Word Embeddings: Distributed Representations of Meaning

In this chapter, we advance next-word prediction by:

- Moving from discrete word symbols to continuous vector representations
- Learning semantic similarity through distributional statistics
- Capturing syntactic and semantic relationships in geometric space
- Reducing model parameters while improving generalization

4.1 From Discrete Symbols to Continuous Vectors

The n-gram models examined in Chapter **??** treated words as atomic, discrete symbols drawn from vocabulary $\mathcal{V}$. Each word $w_i \in \mathcal{V}$ was represented as a distinct integer index, and the model learned separate probability distributions $P(w_{t+1} \mid w_t, w_{t-1}, \ldots)$ for each observed context tuple. This discrete representation suffers from a fundamental limitation: it encodes no notion of similarity between words. The model has no mechanism to recognize that "cat" and "feline" are semantically related, or that "running" and "ran" share morphological structure. Consequently, if the training corpus contains "the cat sat on the mat" but not "the feline sat on the mat", the model cannot generalize to predict the latter, despite their semantic equivalence. This sparsity problem becomes severe as vocabulary size grows: with $|\mathcal{V}| = 50{,}000$ words and trigrams, the number of possible contexts exceeds $50{,}000^2 = 2.5$ billion, yet most will never appear in any corpus. The curse of dimensionality renders exhaustive enumeration infeasible. This chapter introduces word embeddings, which map discrete symbols into continuous vector space $\mathbb{R}^d$ where semantic similarity corresponds to geometric proximity. By representing words as dense vectors rather than atomic symbols, we enable models to generalize across similar contexts and dramatically reduce the parameter space required for accurate next-word prediction.

4.1.1 The Discrete Symbol Problem

Consider the one-hot encoding representation used implicitly in n-gram models, where each word $w_i \in \mathcal{V}$ is mapped to a binary vector $\mathbf{1}_{w_i} \in \{0,1\}^{|\mathcal{V}|}$ with a single 1 at position i and zeros elsewhere: $\mathbf{1}_{w_i}[j] = 1$ if $j = i$ and 0 otherwise for $j = 1, \ldots, |\mathcal{V}|$. This encoding has severe limitations that fundamentally constrain language modeling. First, all words are mutually orthogonal: the dot product between any two distinct one-hot vectors is zero, $\mathbf{1}_{w_i} \cdot \mathbf{1}_{w_j} = \delta_{ij}$ where δ_{ij} is the Kronecker delta, meaning cosine similarity between distinct words is always zero, indicating no relationship whatsoever between semantically related words like "cat" and "feline", or between "king" and "queen". Second, the dimensionality equals vocabulary size $|\mathcal{V}|$, which typically ranges from 30,000 to 100,000 for modern systems, with each vector containing exactly one non-zero

entry yielding extreme sparsity of approximately 0.002%. Third, these high-dimensional sparse vectors cannot be processed efficiently by neural networks, which require dense low-dimensional representations for gradient-based optimization. Fourth, the representation is fundamentally incapable of capturing distributional semantics: words that appear in similar contexts receive completely unrelated encodings. As vocabulary grows with corpus size, the one-hot representation becomes increasingly impractical, motivating the distributed representations we now develop.

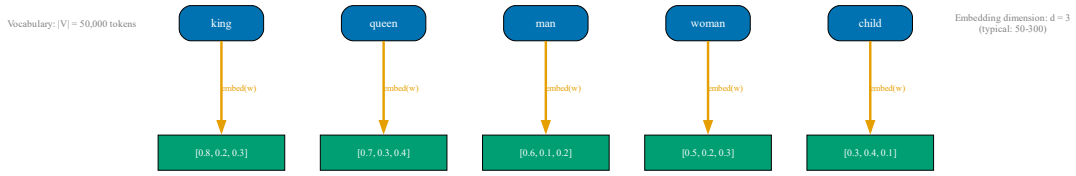

Figure 4.1: One-hot encoding visualization. Panel A shows sparse binary vectors for "cat", "dog", and "car" with $|\mathcal{V}| = 10$. Panel B displays the resulting dot product matrix, which is the identity matrix indicating no similarity between any words. Panel C illustrates that semantic relationships are not captured: related words (cat, dog) are as distant as unrelated words (cat, car). Panel D demonstrates the dimensionality problem: with $|\mathcal{V}| = 50{,}000$, each vector requires 50,000 dimensions with only one non-zero entry, creating severe computational and statistical inefficiency.

4.1.2 The Distributed Representation Hypothesis

The solution to the discrete symbol problem lies in distributed representations, where each word is encoded as a dense vector $\mathbf{e}_w \in \mathbb{R}^d$ with $d \ll |\mathcal{V}|$, typically $d \in [100, 1000]$. Unlike one-hot encoding, where a single dimension encodes word identity, distributed representations spread information across all dimensions. This distribution enables continuous similarity: words with similar meanings map to nearby points in $\mathbb{R}^d$, measured by cosine similarity $\text{sim}(w_i, w_j) = \frac{\mathbf{e}_i \cdot \mathbf{e}_j}{\|\mathbf{e}_i\| \|\mathbf{e}_j\|}$ or Euclidean distance $\|\mathbf{e}_i - \mathbf{e}_j\|$. The dimensionality reduction from $|\mathcal{V}| \approx 50{,}000$ to $d \approx 300$ yields a 99.4% parameter reduction while preserving the information necessary for prediction. The distributed hypothesis posits that all dimensions participate in representing each concept, with semantic features encoded as patterns of activation across the vector. For example, the dimension vector might encode features such as animacy, concreteness, sentiment polarity, or syntactic category, though these dimensions are learned implicitly during training rather than specified a priori. The key mathematical property is that the embedding function $\mathbf{e} : \mathcal{V} \to \mathbb{R}^d$ preserves semantic structure: if words w_i and w_j are semantically similar, then $\|\mathbf{e}(w_i) - \mathbf{e}(w_j)\|$ should be small. This geometric encoding of meaning enables generalization, as models can interpolate between known examples in continuous space rather than memorizing discrete symbolic associations. The embedding matrix $\mathbf{E} \in \mathbb{R}^{|\mathcal{V}| \times d}$ stores all word vectors, with row i containing $\mathbf{e}_{w_i}$. For next-word prediction, the context words $w_1, \ldots, w_{t-1}$ are first converted to embeddings $\mathbf{e}_1, \ldots, \mathbf{e}_{t-1}$, then processed by the model architecture to predict w_t.

Figure 4.2: Distributed representations overcome one-hot limitations. Panel A shows dense embedding vectors with dimensionality $d = 300$ where each word is represented by real-valued numbers across all dimensions. Panel B displays a cosine similarity heatmap revealing semantic structure: related words (cat–feline, dog–canine) have high similarity (0.7–0.9), while unrelated words (cat–car) have low similarity (0.1–0.2). Panel C demonstrates dramatic dimensionality reduction from $|\mathcal{V}| = 50{,}000$ to $d = 300$ while preserving semantic relationships. Panel D visualizes geometric relationships in 2D projection: semantically related words cluster together in embedding space, with distance corresponding to semantic dissimilarity.

4.1.3 Historical Context

The idea that word meaning derives from distributional context traces to Harris (1954), who proposed that "words which occur in the same contexts tend to have similar meanings", and Firth (1957), who famously stated "you shall know a word by the company it keeps". These linguistic insights inspired computational methods in the 1990s, beginning with Latent Semantic Analysis (LSA), which applied singular value decomposition to word-document co-occurrence matrices to obtain low-dimensional representations. While LSA successfully captured semantic similarity for information retrieval tasks, its reliance on linear algebra at document scale proved computationally expensive and unable to capture fine-grained syntactic or semantic relationships. The modern era of word embeddings began with Bengio et al. (2003) [Bengio et al., 2003], who introduced the neural probabilistic language model that learned distributed representations as parameters during next-word prediction training. Their key insight was to jointly optimize embedding vectors and prediction weights using backpropagation, allowing the model to discover representations specifically suited for prediction rather than general co-occurrence patterns. However, Bengio's model required substantial computational resources, limiting adoption. The breakthrough came in 2013 with Mikolov et al.'s Word2Vec [Mikolov et al., 2013], which introduced highly efficient training algorithms (Skip-gram and CBOW) that could learn high-quality embeddings from billion-word corpora in hours rather than weeks. Concurrently, Pennington et al. (2014) developed GloVe [Pennington et al., 2014], which combined the benefits of global matrix factorization and local context windows. These methods democratized word embeddings, making them accessible to researchers without supercomputing resources. By 2015, pre-trained embeddings became standard initialization for virtually all natural language processing tasks. The subsequent shift to contextual embeddings (ELMo, BERT) beginning in 2018 addressed the polysemy limitation we discuss in Section 4.5.3, but the static embeddings developed in this chapter remain fundamental to understanding modern architectures covered in Chapter **??**.

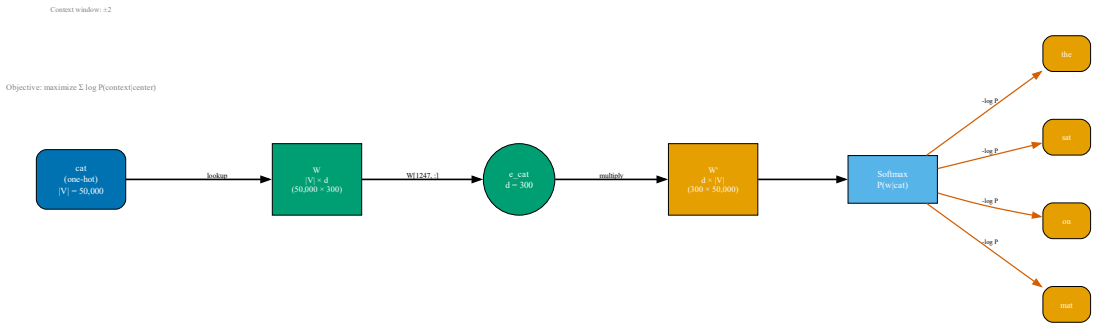

Figure 4.3: Historical development of word embeddings from 1954 to 2018. Key milestones include Harris's distributional hypothesis (1954), LSA dimensionality reduction via SVD (1990), Bengio's neural language model with learned embeddings (2003), Mikolov's Word2Vec bringing embeddings to scale (2013), Pennington's GloVe combining count-based and prediction-based approaches (2014), and the transition to contextual embeddings with ELMo and BERT (2018, covered in Chapter 6). Perplexity improvements over time demonstrate quantitative advances: from 250 (Bengio 2003) to 150 (Word2Vec 2013) to 100 (contextual embeddings 2018) on standard benchmarks.

4.2 The Distributional Hypothesis

The distributional hypothesis provides the theoretical foundation for learning word representations from unlabeled text. Formally stated: words that occur in similar contexts tend to have similar meanings. This linguistic principle translates into a computational objective: construct embeddings such that semantic similarity is reflected by geometric proximity in $\mathbb{R}^d$. The hypothesis operates on context windows, which define the local neighborhood around each word occurrence. Given a sentence and target word at position t, the context con-

sists of words within distance k: $\{w_{t-k},\ldots,w_{t-1},w_{t+1},\ldots,w_{t+k}\}$. For example, in "the cat sat on the mat" with target "sat" and $k=2$, the context is $\{\text{cat},\text{on},\text{the}\}$ (excluding the target itself). By collecting contexts for each word across a large corpus $\mathscr{D}$, we obtain distributional statistics that characterize word usage. Words appearing in similar contexts—such as "cat" and "feline", which both appear after "the" and before action verbs—should receive similar embeddings. Conversely, words appearing in disjoint contexts should receive distant embeddings. This section formalizes distributional semantics through co-occurrence matrices and pointwise mutual information (PMI), which quantify the association between word pairs. We then examine matrix factorization approaches that directly decompose co-occurrence statistics to obtain embeddings, providing intuition for the prediction-based methods in subsequent sections. The key insight is that distributional patterns in raw text provide sufficient signal to learn semantic representations without any labeled data, enabling self-supervised learning from massive corpora.

4.2.1 Context Windows and Co-occurrence

To operationalize the distributional hypothesis, we define a context window of size k around each word in the corpus. Given a text sequence $w_1,w_2,\ldots,w_T$, for each target word w_t we extract the context $\{w_{t-k},\ldots,w_{t-1},w_{t+1},\ldots,w_{t+k}\}$, excluding w_t, which may be symmetric (both directions) for general embeddings or asymmetric (left-only) for causal language modeling. We record co-occurrence whenever a word appears within the context window of another: formally, the co-occurrence count $C_{ij}=\sum_{t=1}^{T}\mathbb{1}[w_t=i]\sum_{k=1}^{K}(\mathbb{1}[w_{t+k}=j]+\mathbb{1}[w_{t-k}=j])$ where $\mathbb{1}[\cdot]$ is the indicator function, counting how many times word j appears within K positions of word i across the corpus. The co-occurrence matrix $\mathbf{C}\in\mathbb{R}^{|\mathscr{V}|\times|\mathscr{V}|}$ stores all pairwise counts and exhibits several properties: symmetry if windows are symmetric ($C_{ij}=C_{ji}$), sparsity because most word pairs never co-occur, and heavy skew with frequent function words dominating. Raw counts favor frequent words, so we apply statistical normalization via pointwise mutual information (PMI), which measures how much more often words i and j co-occur than expected under independence: $\text{PMI}(w_i,w_j)=\log\frac{C_{ij}\cdot N}{\sum_k C_{ik}\cdot\sum_k C_{kj}}$ where N is the total count. PMI is positive when words co-occur more than chance, negative when less, and zero when independent; since negative values are often unreliable due to insufficient data, we use positive PMI (PPMI): $\text{PPMI}(w_i,w_j)=\max(0,\text{PMI}(w_i,w_j))$, which zeros out negatives and better reflects semantic associations.

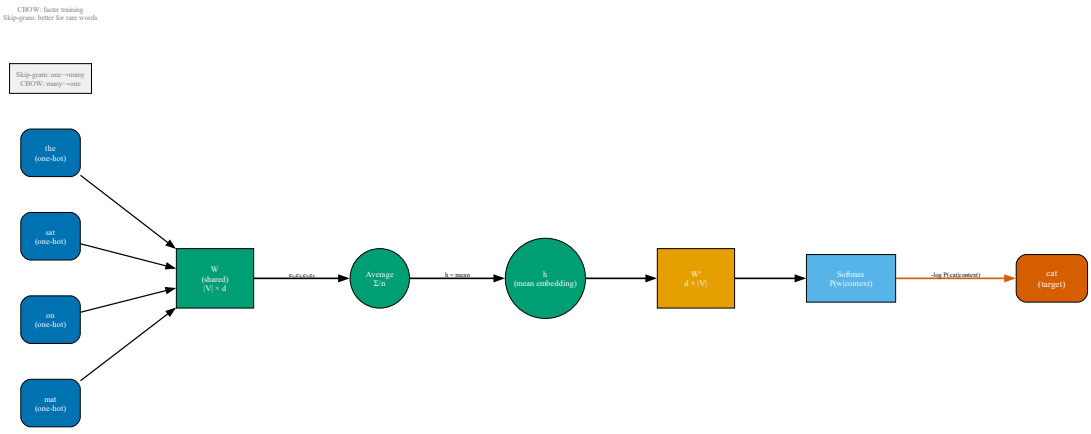

Figure 4.4: Context windows and co-occurrence statistics. Panel A illustrates context window extraction with $k=2$ around target word "cat" in the sentence "the black cat sat there", yielding context $\{\text{the},\text{black},\text{sat},\text{there}\}$. Panel B shows how co-occurrence counts are accumulated across the corpus: each occurrence of "cat" contributes to counts with its context words. Panel C displays a raw co-occurrence matrix as a log-scale heatmap, revealing high counts for frequent words and sparsity for rare words. Panel D shows the PPMI-transformed matrix, which is sparser but exhibits clearer semantic structure: semantically related word pairs (cat–feline, dog–canine) have high PPMI, while unrelated pairs have PPMI near zero.

4.2.2 Matrix Factorization Perspective

Given a co-occurrence or PPMI matrix, we can obtain word embeddings through matrix factorization, approximating the high-dimensional sparse matrix $\mathbf{C} \in \mathbb{R}^{|\mathcal{V}|\times|\mathcal{V}|}$ by a low-rank product of dense matrices. Singular value decomposition (SVD) provides an optimal low-rank approximation: $\mathbf{C} = \mathbf{U}\Sigma\mathbf{V}^\top$ where $\mathbf{U}$ and $\mathbf{V}$ are orthogonal matrices of singular vectors and Σ is a diagonal matrix of singular values $\sigma_1 \geq \sigma_2 \geq \cdots \geq 0$. To obtain a d-dimensional representation with $d \ll |\mathcal{V}|$, we perform truncated SVD keeping only the top d singular values: $\mathbf{C} \approx \mathbf{U}_d\Sigma_d\mathbf{V}_d^\top$, and the word embedding matrix is taken as $\mathbf{E} = \mathbf{U}_d\Sigma_d^{1/2} \in \mathbb{R}^{|\mathcal{V}|\times d}$, where the square root distributes singular values symmetrically between rows and columns. This approach, known as Latent Semantic Analysis when applied to word-document matrices, reduces dimensionality while preserving dominant co-occurrence patterns: words that frequently co-occur with similar contexts have similar embeddings because they project onto similar combinations of the top singular vectors. The truncated SVD minimizes reconstruction error $\|\mathbf{C} - \mathbf{U}_d\Sigma_d\mathbf{V}_d^\top\|_F$ among all rank-d approximations, making it the optimal linear dimensionality reduction. However, this factorization approach has computational limitations: computing SVD for a $|\mathcal{V}| \times |\mathcal{V}|$ matrix with $|\mathcal{V}| = 50{,}000$ requires $O(|\mathcal{V}|^3)$ operations and $O(|\mathcal{V}|^2)$ memory, which becomes prohibitive at scale. Modern embedding methods overcome these limitations through prediction-based training that avoids explicitly constructing the co-occurrence matrix.

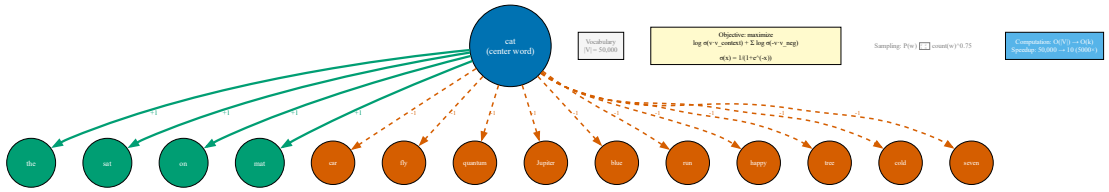

Figure 4.5: SVD decomposition visualized in 3D. The original co-occurrence space is $|\mathcal{V}|$-dimensional (shown as a high-dimensional point cloud in the upper region), with most words lying on a lower-dimensional manifold. Truncated SVD projects onto the top d principal components (shown as a 2D plane in the lower region), preserving the dominant variance while reducing dimensionality by 99%. Semantic clusters (animals, colors, verbs) remain separated in the reduced space, demonstrating that co-occurrence patterns encode semantic structure that survives dimensionality reduction.

4.2.3 Semantic Similarity from Distribution

The distributional hypothesis predicts that words appearing in similar contexts should have similar embeddings, and this prediction is empirically validated: embeddings learned from co-occurrence statistics exhibit strong semantic clustering, with synonyms such as “cat” and “feline” appearing in nearly identical contexts and yielding cosine similarity typically above 0.7 in learned embeddings. Hypernyms and hyponyms also cluster: “animal”, “mammal”, “cat”, “feline” form a hierarchy with progressively higher similarity at finer levels. Interestingly, antonyms often receive similar embeddings because they appear in similar contexts: both “hot” and “cold” modify “weather”, “water”, “temperature”, despite having opposite meanings, revealing a limitation of purely distributional semantics where context determines syntactic positions but does not uniquely determine semantic content. Syntactic categories also emerge from distributional patterns: verbs cluster because they appear in similar structural positions (after subjects, before objects), nouns cluster because they appear after determiners and before verbs, and adjectives cluster because they appear before nouns. To quantify similarity, we use cosine similarity $\text{sim}(w_i, w_j) = \frac{\mathbf{e}_i \cdot \mathbf{e}_j}{\|\mathbf{e}_i\|\|\mathbf{e}_j\|}$, which ranges from -1 (opposite) through 0 (orthogonal) to $+1$ (identical direction) and is preferred over Euclidean distance because it is invariant to vector magnitude, focusing only on direction in embedding space. Given a query word, we can retrieve its nearest neighbors by ranking all other words by cosine similarity, producing semantically related terms useful for query expansion, synonym detection, and lexical semantics research, validating that unsupervised learning from raw text can discover semantic structure without labeled data.

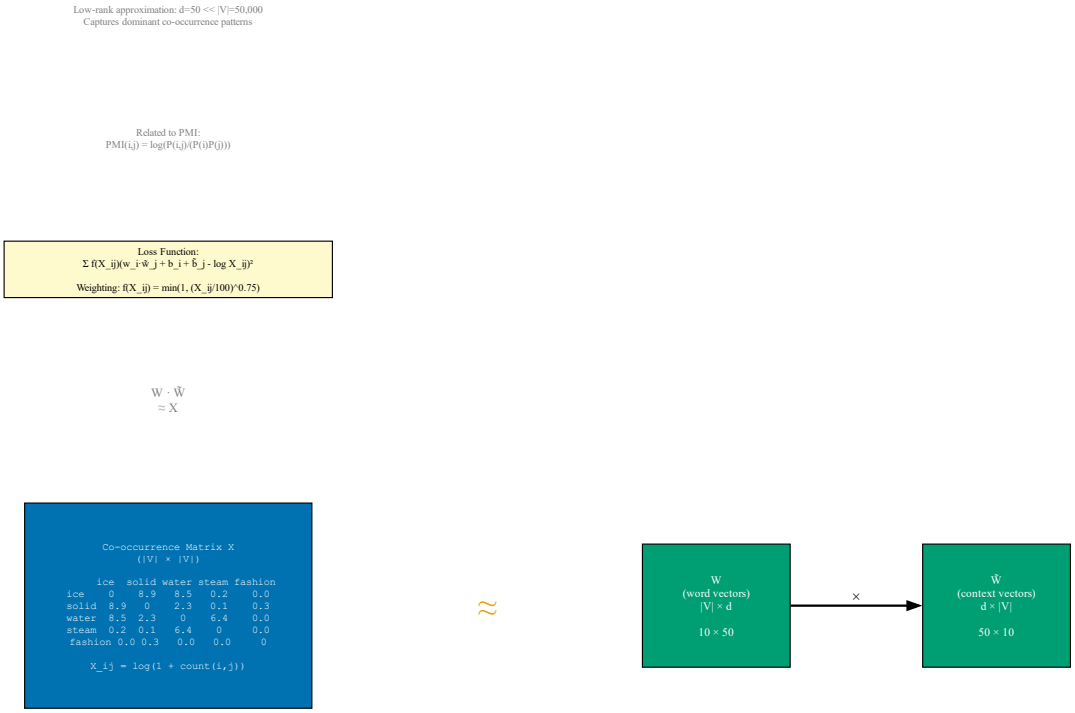

Figure 4.6: Semantic similarity patterns in learned embeddings. Panel A shows a t-SNE projection of 500 words into 2D space, revealing clear semantic clusters. Panel B highlights specific clusters: animals (cat, dog, horse), colors (red, blue, green), and verbs (run, walk, jump) group together without any explicit supervision. Panel C lists nearest neighbors for "cat" by cosine similarity: feline (0.82), kitten (0.76), dog (0.71), puppy (0.68), demonstrating that synonyms and related concepts are nearby in embedding space. Panel D displays a similarity heatmap for 20 selected words, showing high within-cluster similarity (dark) and low between-cluster similarity (light).

4.3 Word2Vec: Skip-gram and CBOW

While matrix factorization methods provide intuitive connections to co-occurrence statistics, they suffer from computational and memory limitations at scale. Mikolov et al. (2013) introduced Word2Vec, a family of efficient prediction-based algorithms that learn embeddings by training shallow neural networks to predict context from words (Skip-gram) or words from context (CBOW). These methods avoid explicitly constructing the co-occurrence matrix, instead processing the corpus in a streaming fashion via stochastic gradient descent. The key innovation is framing embedding learning as a prediction task: given a target word w_t, predict its surrounding context words $w_{t-k}, \ldots, w_{t+k}$, or vice versa. By maximizing prediction accuracy, the model is forced to learn embeddings that capture distributional semantics. Word2Vec's computational efficiency stems from two architectural choices: shallow networks (single hidden layer with no nonlinearity) and negative sampling to avoid the expensive softmax normalization over the full vocabulary. These optimizations enable training on billion-word corpora in hours on a single machine, democratizing access to high-quality embeddings. The self-supervised nature of the training objective—no labels required, only raw text—allows leveraging massive unlabeled corpora. Word2Vec embeddings exhibit remarkable properties including linear substructures (Section 4.5) and strong performance on downstream tasks. While modern contextual embeddings have superseded static Word2Vec for many applications, understanding Word2Vec remains essential for two reasons: it introduces the prediction-based paradigm that underlies all subsequent neural language models, and its embeddings serve as interpretable baseline representations for analyzing semantic spaces.

4.3.1 The Word2Vec Framework

Word2Vec employs self-supervised learning: the training signal is derived from the data itself rather than external labels. Given a corpus $\mathscr{D} = [w_1, w_2, \ldots, w_T]$, we slide a context window of size $2K+1$ across the sequence, creating training examples at each position t. For target word w_t, the context is $\{w_{t-K}, \ldots, w_{t-1}, w_{t+1}, \ldots, w_{t+K}\}$. Two architectures are defined: Skip-gram predicts each context word given the target, while CBOW (Continuous Bag-of-Words) predicts the target given the context. Both learn two embedding matrices: an input embedding matrix $\mathbf{E}^{\text{in}} \in \mathbb{R}^{|\mathscr{V}| \times d}$ and an output embedding matrix $\mathbf{E}^{\text{out}} \in \mathbb{R}^{|\mathscr{V}| \times d}$, where d is the embedding dimension (typically $d \in [100, 300]$). After training, the input embeddings $\mathbf{E}^{\text{in}}$ are typically used as the final word representations, though some implementations average input and output embeddings. The neural architecture is intentionally simple: a single linear layer with no nonlinearity (making it a log-linear model). For Skip-gram, the forward pass looks up the input embedding for the target word, computes dot products with all output embeddings to produce logits, and applies softmax to obtain a probability distribution over vocabulary. For CBOW, the forward pass averages the input embeddings of context words, then proceeds identically. The loss function is cross-entropy between predicted and observed distributions. This simplicity enables efficient training while capturing distributional semantics: words appearing in similar contexts must have similar embeddings to achieve low loss, enforcing the distributional hypothesis directly through the objective function. The training procedure is stochastic gradient descent over all (target, context) pairs in the corpus, with negative sampling (Section 4.3.4) to make gradient computation tractable.

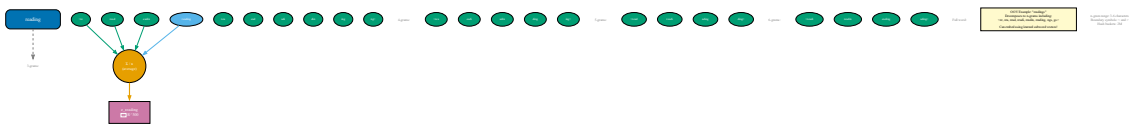

Figure 4.7: Word2Vec framework overview. Panel A illustrates the self-supervised learning setup: given raw text "the cat sat on the mat", context windows automatically generate (target, context) training pairs without requiring manual labels. Panel B compares Skip-gram and CBOW architectures side-by-side: Skip-gram predicts context words from target (one-to-many), while CBOW predicts target from averaged context words (many-to-one). Panel C shows training data generation from a sample corpus, with arrows indicating prediction direction for each architecture. Panel D visualizes embedding space evolution during training: random initialization yields no structure, but after training, semantically related words cluster together.

4.3.2 Skip-gram Architecture

The Skip-gram model predicts context words given a target word, formulating embedding learning as a classification task that can be trained efficiently via stochastic gradient descent. For each position t in the corpus and each context offset $k \in \{-K, \ldots, -1, 1, \ldots, K\}$, we form a training pair (w_t, w_{t+k}) where the model aims to maximize $P(w_{t+k}|w_t)$ for all such pairs, yielding the Skip-gram objective $\mathscr{L}_{\text{skip-gram}} = -\frac{1}{T}\sum_{t=1}^{T}\sum_{-K \le k \le K, k \neq 0} \log P(w_{t+k}|w_t)$ where the probability is modeled using learned embeddings. The architecture is: (1) Look up the input embedding for the target word: $\mathbf{e}_t = \mathbf{E}^{\text{in}}[w_t, :] \in \mathbb{R}^d$. (2) Compute unnormalized scores (logits) for all vocabulary words by taking dot products with output embeddings: $s_v = \mathbf{e}_t \cdot \mathbf{E}^{\text{out}}[v, :]$ for all $v \in \mathscr{V}$. (3) Apply softmax to obtain a probability distribution: $P(w_c|w_t) = \frac{\exp(s_c)}{\sum_{v \in \mathscr{V}} \exp(s_v)}$. (4) Maximize log-probability of observed context word $w_c = w_{t+k}$: $\log P(w_c|w_t) = s_c - \log \sum_{v \in \mathscr{V}} \exp(s_v)$. The gradient with respect to embeddings is computed via backpropagation and the parameters are updated using stochastic gradient descent. Skip-gram generates $2K$ training examples per word occurrence (one for each context position), providing rich supervision for rare words. For example, the sentence "the cat sat" with $K = 1$ generates Skip-gram pairs: (cat, the), (cat, sat), (sat, cat), (the, cat). This one-to-many prediction means Skip-gram trains slower than CBOW but often achieves better representations for infrequent words because each occurrence generates multiple gradient updates. The learned embeddings satisfy the property that words appearing in similar contexts receive similar input embeddings, because the model can only achieve low loss by assigning similar

probabilities to similar contexts, which requires similar input embeddings given the shared output embedding matrix.

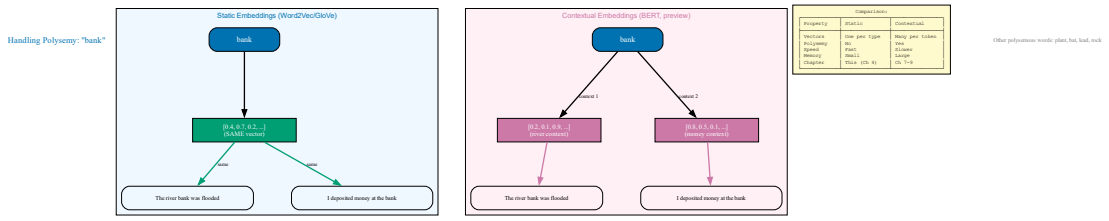

Figure 4.8: Skip-gram architecture and training. Panel A displays the network diagram: target word "cat" is embedded via $\mathbf{E}^{\text{in}}$, then scored against all output embeddings in $\mathbf{E}^{\text{out}}$ to predict context words. Panel B shows example training pairs generated from "the cat sat on the mat" with $K = 2$: target "cat" generates four pairs: (cat, the), (cat, sat), (cat, on), (cat, mat). Panel C illustrates the forward pass computation: embedding lookup, dot products with output embeddings, softmax normalization, and cross-entropy loss. Panel D visualizes the loss landscape as a 3D surface over two embedding dimensions, showing the optimization objective: low loss when predicted context matches observed context.

4.3.3 CBOW Architecture

The Continuous Bag-of-Words (CBOW) model inverts the Skip-gram prediction direction by predicting the target word given its context rather than vice versa. For each position t, we collect the context words $\{w_{t-K}, \ldots, w_{t-1}, w_{t+1}, \ldots, w_{t+K}\}$ and form a training example where the input is this set (treated as a bag, ignoring order) and the output is the target word w_t, yielding the CBOW objective $\mathscr{L}_{\text{CBOW}} = -\frac{1}{T}\sum_{t=1}^{T} \log P(w_t | w_{t-K}, \ldots, w_{t+K})$ which averages the log-probability over all positions in the corpus. The architecture is: (1) Look up input embeddings for all context words: $\mathbf{e}_{t+k} = \mathbf{E}^{\text{in}}[w_{t+k}, :]$ for $k \in \{-K, \ldots, -1, 1, \ldots, K\}$. (2) Average the context embeddings to form a single context vector: $\bar{\mathbf{e}} = \frac{1}{2K}\sum_{k=-K, k\neq 0}^{K} \mathbf{e}_{t+k}$. (3) Compute logits by dotting the averaged context with all output embeddings: $s_v = \bar{\mathbf{e}} \cdot \mathbf{E}^{\text{out}}[v, :]$. (4) Apply softmax: $P(w_t | \text{context}) = \frac{\exp(s_{w_t})}{\sum_{v \in \mathscr{V}} \exp(s_v)}$. (5) Maximize the log-probability of the observed target word. The averaging step is crucial: it reduces the $2K$ context embeddings to a fixed-size representation regardless of window size. This bag-of-words aggregation ignores word order—"the cat sat" and "sat cat the" produce identical representations—which is a limitation but enables efficient training. CBOW generates one training example per word occurrence, making it faster than Skip-gram. Empirically, CBOW tends to perform better on frequent words (for which averaging provides stable signal) while Skip-gram excels on rare words (for which each occurrence is precious). The choice between architectures depends on corpus size and downstream task: for large corpora, CBOW's efficiency advantage is significant, while for small corpora or morphologically complex languages, Skip-gram's finer-grained supervision is preferable. Both architectures learn embeddings capturing distributional semantics, and the choice is often made based on computational budget rather than quality considerations.

4.3.4 Negative Sampling

Both Skip-gram and CBOW as described above require computing the softmax normalization $\sum_{v \in \mathscr{V}} \exp(s_v)$ over all $|\mathscr{V}|$ words at each training step, and for large vocabularies ($|\mathscr{V}| \approx 50{,}000$), this normalization dominates computation time, requiring $O(|\mathscr{V}| \cdot d)$ operations per example. With billions of training examples, the total cost becomes prohibitive, motivating efficient approximations. Negative sampling addresses this bottleneck by approximating the softmax objective with a binary classification task: instead of predicting which vocabulary word is the correct context (multi-class classification over $|\mathscr{V}|$ classes), we reframe the problem as distinguishing the observed (target, context) pair from randomly sampled negative pairs. For each positive pair (w_t, w_c), we sample k negative context words $\{w_{n_1}, \ldots, w_{n_k}\}$ from a noise distribution

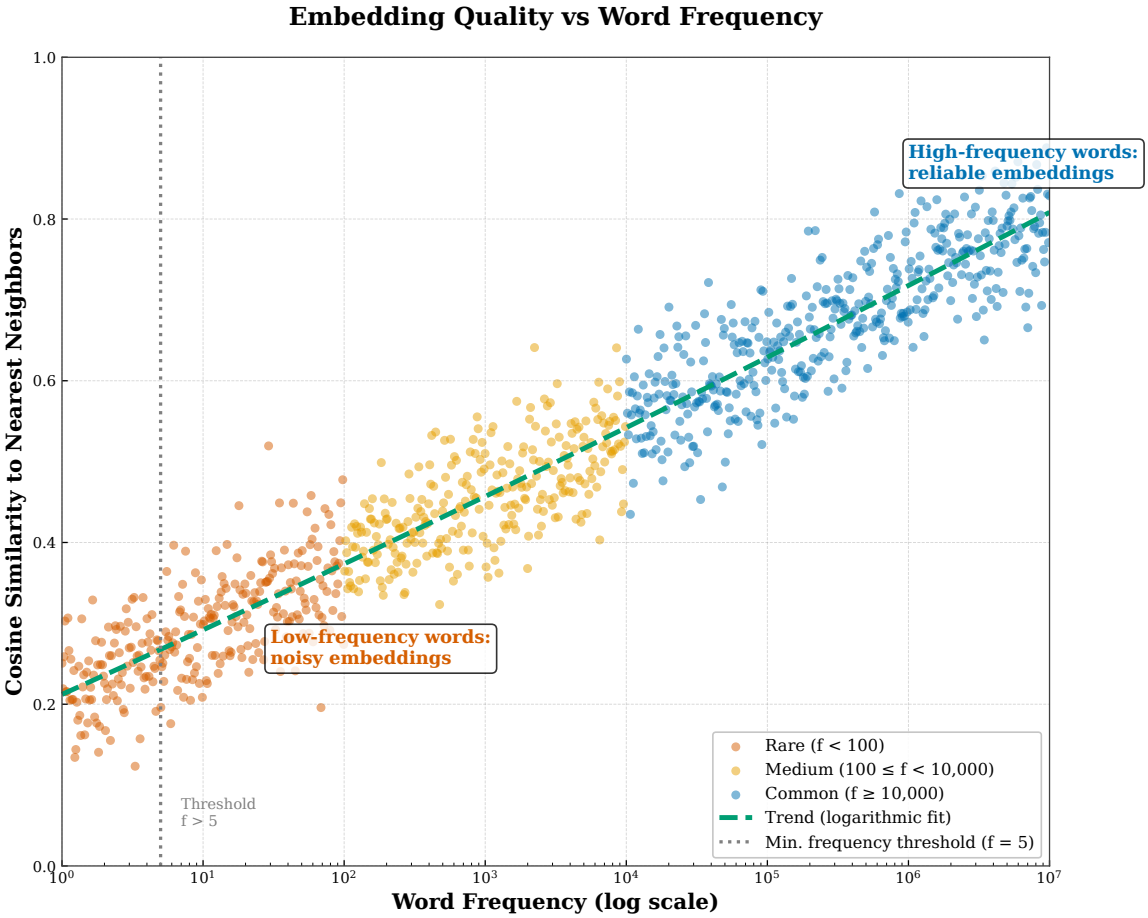

Figure 4.9: CBOW architecture and comparison with Skip-gram. Panel A shows the CBOW network diagram: context words $\{\text{the}, \text{sat}, \text{on}, \text{mat}\}$ are embedded, averaged, then scored against output embeddings to predict target "cat". Panel B illustrates an example training instance: given context $\{\text{the}, \text{sat}, \text{on}\}$ around position 3, predict target "cat". Panel C details the context averaging mechanism: input embeddings are element-wise averaged to produce a fixed-size context representation $\bar{\mathbf{e}}$ regardless of window size. Panel D compares training speed and accuracy: CBOW trains faster (1 example per word vs. $2K$ for Skip-gram) and achieves slightly lower loss on frequent words, while Skip-gram achieves lower loss on rare words.

$P_n(w)$ (typically $k = 5$ to 20), and the objective becomes maximizing the probability that the positive pair is from the data while minimizing the probability that negative pairs are from the data, yielding the loss $\mathcal{L}_{\text{neg}} = -\log \sigma(\mathbf{e}_t \cdot \mathbf{e}_c) - \sum_{i=1}^{k} \mathbb{E}_{\lfloor} w_{n_i} \sim P_n \log \sigma(-\mathbf{e}_t \cdot \mathbf{e}_{n_i})$ where $\sigma(x) = 1/(1+e^{-x})$ is the sigmoid function. This formulation replaces the $O(|\mathcal{V}|)$ softmax computation with $O(k)$ sigmoid evaluations, achieving a $|\mathcal{V}|/k \approx 2500$ speedup for $|\mathcal{V}| = 50{,}000$ and $k = 20$. The noise distribution $P_n(w)$ is typically set to the unigram distribution raised to the power $3/4$: $P_n(w) \propto (\text{count}(w))^{3/4}$. This sublinear exponent smooths the distribution, sampling rare words more frequently than their corpus proportion (otherwise extremely frequent words would dominate negatives) while still favoring common words. Empirically, negative sampling performs as well as full softmax on most tasks while being orders of magnitude faster, making Word2Vec training feasible at scale. An alternative approach is hierarchical softmax, which organizes vocabulary in a binary tree and predicts the path to the target word, reducing complexity to $O(\log |\mathcal{V}|)$, but negative sampling has proven more popular in practice due to simpler implementation and better empirical results.

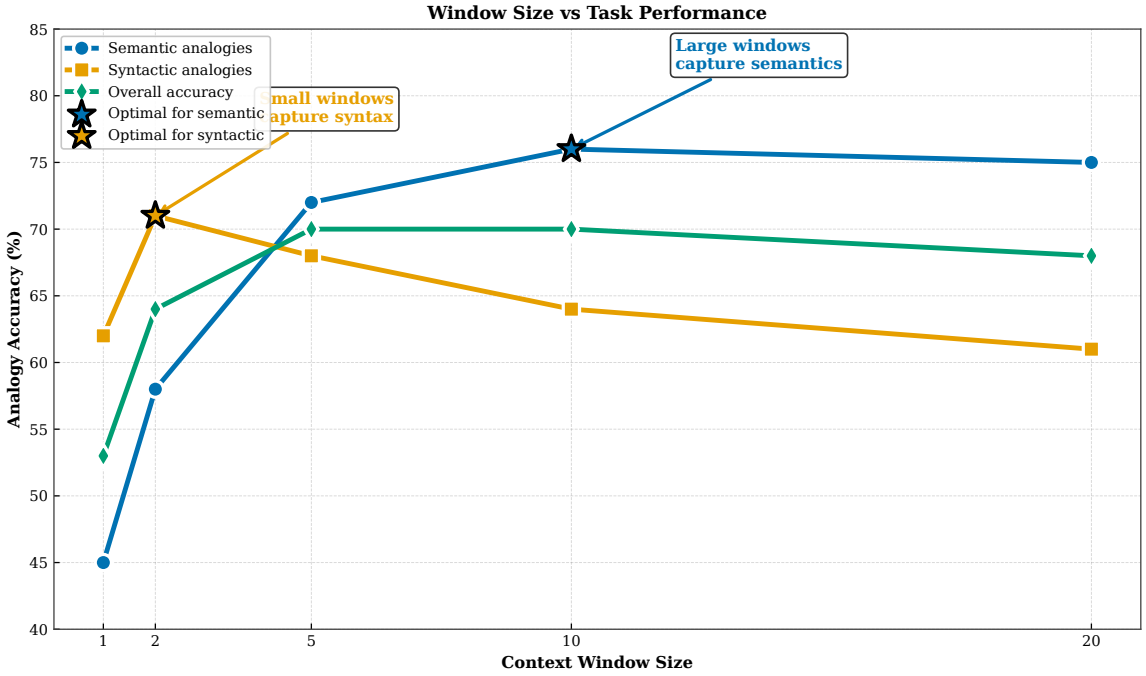

Figure 4.10: Negative sampling mechanism. Panel A illustrates positive and negative pairs: given target "cat" and observed context "sat" (positive pair, green), we sample $k = 5$ negative context words $\{\text{car}, \text{politics}, \text{ocean}, \text{software}, \text{yesterday}\}$ (negative pairs, red) that did not actually appear near "cat". Panel B shows the noise distribution $P_n(w) \propto (\text{count}(w))^{3/4}$, which is flatter than the unigram distribution, increasing sampling probability for rare words. Panel C compares computational cost: full softmax requires $|\mathcal{V}|$ exponentiations and normalizations, while negative sampling requires only $k+1$ sigmoid evaluations, yielding 2500× speedup for $|\mathcal{V}| = 50{,}000$, $k = 20$. Panel D displays convergence curves showing that negative sampling (blue) achieves similar final loss to full softmax (red) but reaches convergence in 1/10th the training time.

4.4 GloVe: Global Vectors

While Word2Vec learns embeddings through local context prediction, GloVe (Global Vectors for Word Representation) takes a hybrid approach that combines the statistical efficiency of global matrix factorization with the learning power of prediction-based methods. Pennington et al. (2014) [Pennington et al., 2014] observed that ratios of co-occurrence probabilities encode semantic relationships more directly than raw probabilities or counts. For example, consider words "ice" and "steam" with context words "solid" and "gas". The ratio $P(\text{solid}|\text{ice})/P(\text{solid}|\text{steam})$ is large (ice is solid but steam is not), while $P(\text{gas}|\text{ice})/P(\text{gas}|\text{steam})$ is small (steam is gas but ice is not). These ratios capture semantic distinctions: when the ratio is large, the context word is relevant to the first word but not the second; when small, vice versa; when near 1, the context word relates equally to both. GloVe embeds this intuition directly in its objective function by requiring that dot products between embeddings approximate logarithms of co-occurrence counts. The method precomputes a co-occurrence matrix $\mathbf{X}$ from the corpus, then learns embeddings by minimizing weighted squared error between embedding dot products and log counts. This approach leverages global corpus statistics (like matrix factorization) while retaining trainability via gradient descent (like Word2Vec). GloVe's offline training on the precomputed co-occurrence matrix offers computational advantages: the matrix is constructed once, then training iterates only over non-zero entries (typically $< 1\%$ of all word pairs), avoiding redundant processing of the same contexts multiple times as Word2Vec does in multiple epochs. Empirically, GloVe produces embeddings of comparable quality to Word2Vec, with the choice often driven by engineering considerations rather than fundamental performance differences.

4.4.1 Count-Based Meets Prediction-Based

The key insight underlying GloVe is that ratios of co-occurrence probabilities reveal semantic relationships more clearly than individual probabilities. Let $\mathbf{X}$ be the co-occurrence matrix where X_{ij} counts how often word j appears in the context of word i. Define $X_i = \sum_k X_{ik}$ as the total count for word i, and let $P(w_j|w_i) = X_{ij}/X_i$ be the probability that word j appears in the context of word i. Consider two words w_i and w_j and a probe context word w_k. The ratio $P(w_k|w_i)/P(w_k|w_j)$ encodes their relationship to w_k: if w_k is semantically related to w_i but not w_j, the ratio is large; if related to w_j but not w_i, the ratio is small; if related (or unrelated) to both, the ratio is near 1. For example, with $w_i = \text{ice}$, $w_j = \text{steam}$, and $w_k \in \{\text{solid}, \text{gas}, \text{water}, \text{fashion}\}$, we observe: $P(\text{solid}|\text{ice})/P(\text{solid}|\text{steam}) \approx 8.9$ (large), $P(\text{gas}|\text{ice})/P(\text{gas}|\text{steam}) \approx 0.085$ (small), $P(\text{water}|\text{ice})/P(\text{water}|\text{steam}) \approx 1.36$ (near 1), and $P(\text{fashion}|\text{ice})/P(\text{fashion}|\text{steam}) \approx 0.96$ (near 1). The ratios encode discriminative semantic information: large/small ratios identify distinctive features, while ratios near 1 indicate shared or absent features. GloVe's objective is designed so that embedding geometry captures these ratios: the ratio $P(w_k|w_i)/P(w_k|w_j)$ should be expressible as a function of embeddings $\mathbf{e}_i, \mathbf{e}_j, \mathbf{e}_k$. Pennington et al. show that requiring this functional relationship to hold for all words leads to a log-bilinear model where $\mathbf{e}_i \cdot \mathbf{e}_j \approx \log X_{ij}$, which forms the basis of the GloVe objective.

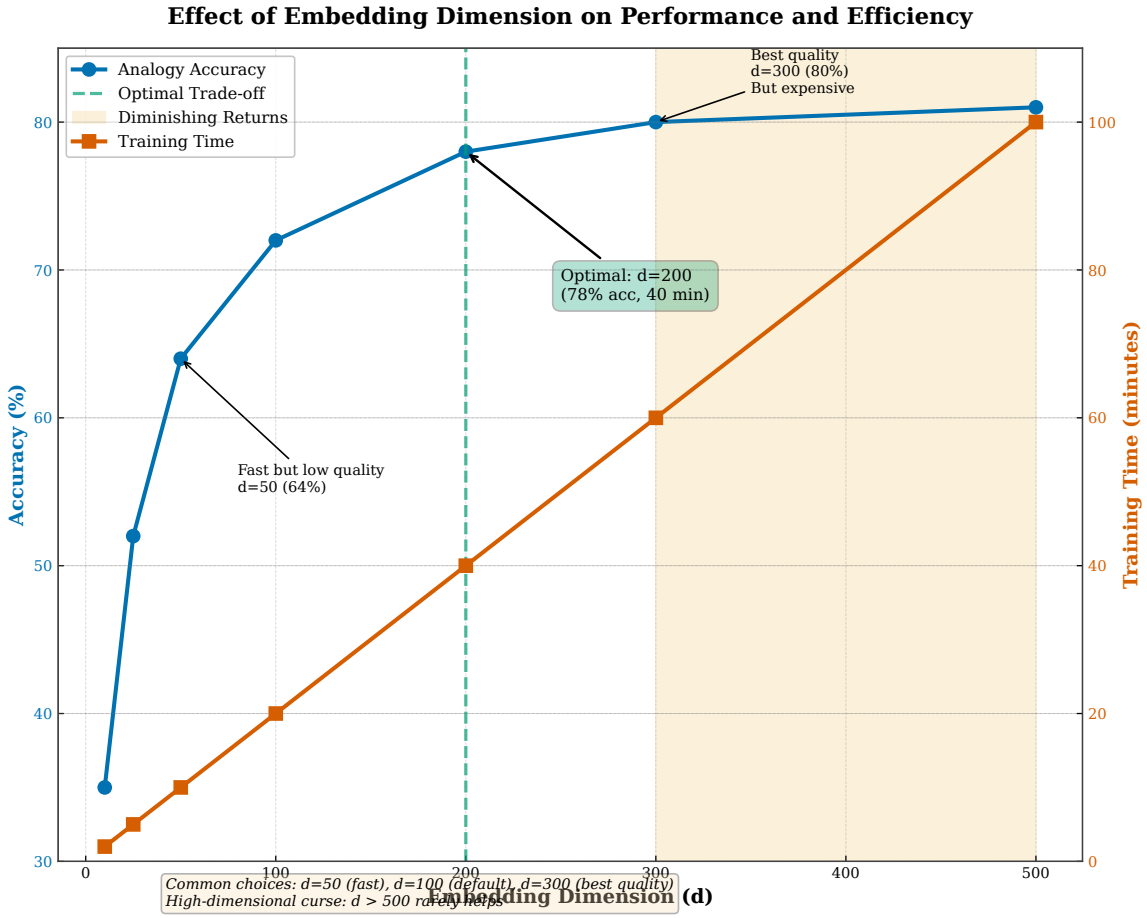

Figure 4.11: Co-occurrence probability ratios encode semantic relationships. Panel A shows co-occurrence counts for "ice" and "steam" with various context words: $X_{\text{ice,solid}} = 1820$, $X_{\text{steam,solid}} = 204$, $X_{\text{ice,gas}} = 15$, $X_{\text{steam,gas}} = 176$, etc. Panel B displays the probability ratios: $P(\text{solid}|\text{ice})/P(\text{solid}|\text{steam}) = 8.9$ (large, discriminates ice), $P(\text{gas}|\text{ice})/P(\text{gas}|\text{steam}) = 0.085$ (small, discriminates steam), $P(\text{water}|\text{ice})/P(\text{water}|\text{steam}) = 1.36$ (near 1, relates to both). Panel C explains how these patterns reveal semantic structure: large ratios indicate features specific to the first word, small ratios indicate features specific to the second word, ratios near 1 indicate shared or irrelevant features. Panel D shows the timeline of embedding methods, with GloVe (2014) positioned as a synthesis of count-based (LSA, 1990s) and prediction-based (Word2Vec, 2013) approaches.

4.4.2 The GloVe Objective

GloVe learns embeddings by fitting a log-bilinear model to the co-occurrence matrix, requiring that the dot product between word embeddings approximates the logarithm of their co-occurrence count: $\mathbf{e}_i^\top \mathbf{e}_j + b_i + b_j \approx \log X_{ij}$ where b_i and b_j are scalar bias terms. This equation states that frequent co-occurrence (large X_{ij}) corresponds to large dot product (close embeddings), while rare co-occurrence (small X_{ij}) corresponds to small dot product (distant embeddings), with the logarithm motivated both by the ratio-based derivation and by the observation that raw counts span many orders of magnitude while log-counts are more evenly distributed. To handle the varying reliability of co-occurrence counts, GloVe uses a weighted least squares objective $\mathscr{L}_{\text{GloVe}} = \sum_{i,j=1}^{|\mathscr{V}|} f(X_{ij})(\mathbf{e}_i^\top \mathbf{e}_j + b_i + b_j - \log X_{ij})^2$ where the weighting function $f(X_{ij})$ assigns low weight to rare co-occurrences (which are noisy), increases weight up to $x_{\max} = 100$, then caps at 1 for very frequent co-occurrences (which would otherwise overwhelm the loss), giving moderate co-occurrences the highest relative influence. Training uses AdaGrad optimization on both the input embeddings $\mathbf{E}^{\text{in}}$, output embeddings $\mathbf{E}^{\text{out}}$, and bias terms, with the final embeddings typically taken as $\mathbf{e}_i = \mathbf{E}^{\text{in}}[i,:] + \mathbf{E}^{\text{out}}[i,:]$ (averaging input and output), which empirically performs slightly better than using input embeddings alone. The GloVe training procedure iterates over non-zero entries of $\mathbf{X}$ for 50–100 epochs, converging faster than Word2Vec on large corpora because the precomputed matrix avoids redundant processing.

4.4.3 GloVe vs. Word2Vec

GloVe and Word2Vec represent two philosophical approaches to learning embeddings, yet empirically produce representations of similar quality. Word2Vec uses online training: it processes the corpus sequentially, making gradient updates on local context windows without building global statistics. This streaming approach enables training on corpora too large to fit in memory and naturally handles dynamic corpora where new text arrives continuously. In contrast, GloVe uses offline training: it first computes the global co-occurrence matrix $\mathbf{X}$ (requiring full corpus access and memory to store non-zero entries), then trains via batch optimization on the matrix. This two-stage approach makes efficient use of corpus statistics but requires multiple passes and substantial memory. The objective functions differ in form: Word2Vec minimizes cross-entropy (a classification loss) while GloVe minimizes weighted squared error (a regression loss). However, Levy and Goldberg (2014) demonstrated that Skip-gram with negative sampling implicitly factorizes a shifted PMI matrix, revealing that both methods optimize related objectives despite superficial differences. Performance comparisons show that GloVe and Word2Vec achieve similar scores on word similarity and analogy benchmarks, with differences often within noise margins. The choice between them is typically driven by engineering constraints: GloVe is faster when the co-occurrence matrix fits in memory and multiple training runs are needed (since the matrix need not be recomputed), while Word2Vec is preferred for streaming scenarios or extremely large corpora where constructing $\mathbf{X}$ is infeasible. Both methods have been superseded by contextual embeddings (Chapter **??**) for most modern applications, but their core insights—distributional learning, efficient training, and geometric encoding of semantics—remain foundational to understanding language representations.

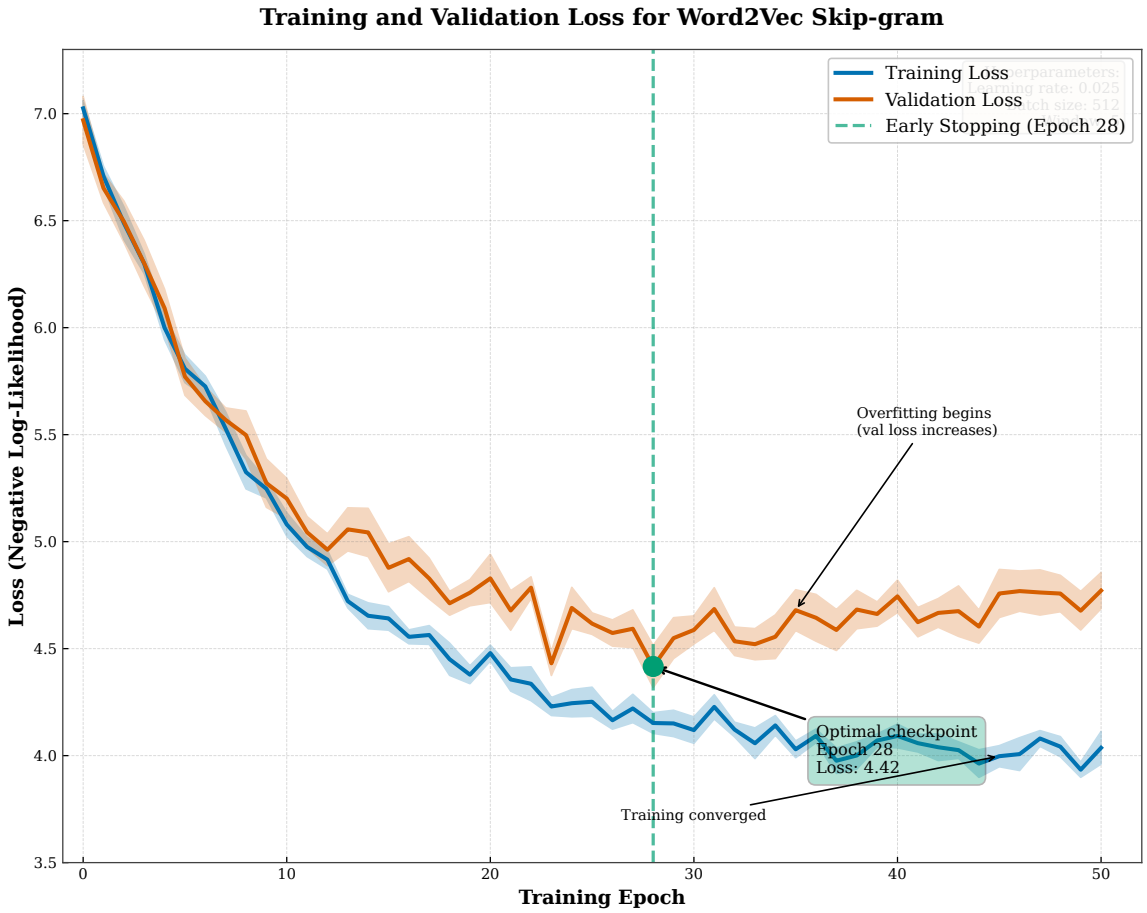

Figure 4.12: GloVe objective function and training. Panel A displays the co-occurrence matrix $\mathbf{X}$ as a log-scale heatmap: most entries are zero (white), frequent pairs have high counts (dark red), revealing sparse structure. Panel B shows the weighting function $f(X_{ij})$: zero weight for $X_{ij} = 0$, increasing weight up to $X_{ij} = x_{\max} = 100$, then constant weight for larger values. This weighting balances contributions from rare and frequent co-occurrences. Panel C visualizes the loss landscape as a 3D surface over two embedding dimensions: low loss when $\mathbf{e}_i^\top \mathbf{e}_j + b_i + b_j \approx \log X_{ij}$, high loss when they differ. Panel D plots training convergence: weighted squared error decreases over iterations, stabilizing after approximately 50 epochs over the non-zero entries of $\mathbf{X}$.

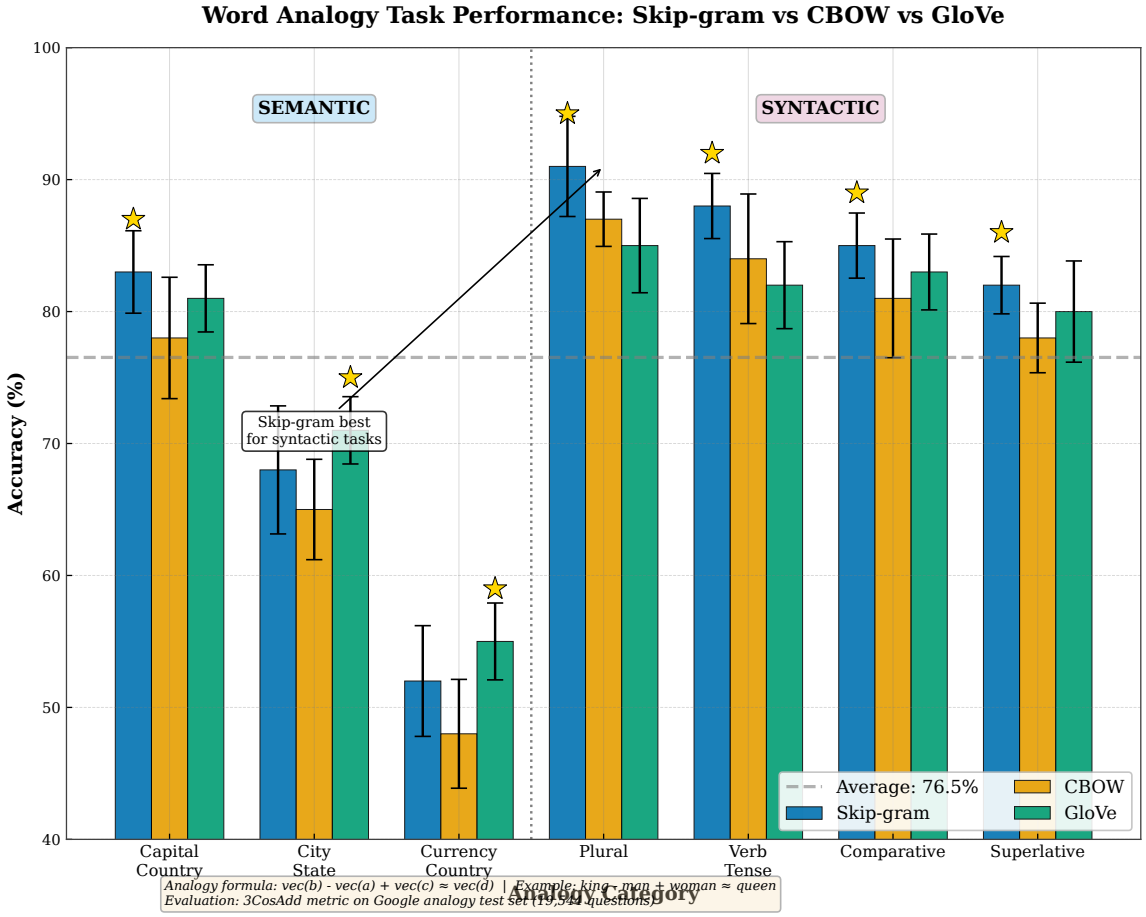

Figure 4.13: Comparison of Word2Vec and GloVe on analogy tasks. The Google Analogy Dataset contains semantic analogies (e.g., king:queen::man:woman, Paris:France::Rome:Italy) and syntactic analogies (e.g., run:running::walk:walking, good:better::bad:worse). Both methods achieve 70–75% accuracy on semantic analogies and 60–65% on syntactic analogies with $d = 300$ dimensions, demonstrating comparable performance. GloVe is slightly better on semantic tasks, Word2Vec on syntactic tasks, but differences are small. Training time comparison shows GloVe is faster when the co-occurrence matrix is precomputed and reused, while Word2Vec is more memory-efficient for very large corpora.

4.5 Properties and Evaluation

Word embeddings exhibit remarkable structural properties beyond simple semantic clustering. The most celebrated discovery is that embeddings support vector arithmetic: adding and subtracting word vectors produces meaningful results that capture semantic and syntactic relationships. The canonical example is $\mathbf{e}_{\text{king}} - \mathbf{e}_{\text{man}} + \mathbf{e}_{\text{woman}} \approx \mathbf{e}_{\text{queen}}$, suggesting that the vector offset from "man" to "king" (roughly encoding the concept "royal") can be added to "woman" to arrive at the female royal equivalent. This linear substructure extends beyond single examples to systematic patterns: gender differences (king–queen, man–woman, brother–sister), plurality (car–cars, dog–dogs), verb tense (walk–walked, run–ran), comparative and superlative forms (good–better–best), and country-capital relationships (Paris–France, London–England) all exhibit parallel vector offsets. These regularities suggest that embedding spaces organize themselves into geometric structures where semantic relationships correspond to directions in space. Evaluating embedding quality requires both intrinsic metrics (measuring embedding properties directly) and extrinsic metrics (measuring downstream task performance). Intrinsic evaluations include word similarity correlation with human judgments, analogy accuracy, and semantic clustering quality. Extrinsic evaluations use embeddings as features for tasks like sentiment analysis, named entity recognition, or machine translation, measuring whether better embeddings improve task performance. Importantly, embeddings also inherit biases from training corpora, raising ethical concerns about their deployment in real-world systems. This section examines these properties, evaluation methodologies, and limitations in detail.

4.5.1 Linear Substructures and Analogies

The discovery that word embeddings exhibit linear algebraic structure was among the most surprising and influential findings in representation learning, fundamentally changing how researchers understood the geometry of learned representations. Mikolov et al. (2013) demonstrated that semantic and syntactic relationships often correspond to consistent vector offsets: formally, for analogies of the form "a is to b as c is to d", we often find $\mathbf{e}_b - \mathbf{e}_a \approx \mathbf{e}_d - \mathbf{e}_c$ (or equivalently $\mathbf{e}_b - \mathbf{e}_a + \mathbf{e}_c \approx \mathbf{e}_d$), meaning that the vector direction encoding a relationship (such as male-to-female or present-to-past) is approximately constant across word pairs exhibiting that relationship. This remarkable property enables solving analogies computationally: given the triplet (a,b,c), we find the answer d by computing $\arg\max_{w \in \mathcal{V} \setminus \{a,b,c\}} \text{sim}(\mathbf{e}_w, \mathbf{e}_b - \mathbf{e}_a + \mathbf{e}_c)$ where $\text{sim}(\cdot,\cdot)$ is cosine similarity, essentially finding the word whose embedding best matches the target location in vector space. Empirically, this procedure achieves 60–80% accuracy on analogy benchmarks, far above chance (0.002% for $|\mathcal{V}| = 50{,}000$). The linear structure spans multiple semantic categories: (1) Gender: king–queen, man–woman, actor–actress, brother–sister, uncle–aunt. (2) Country-capital: Paris–France, London–England, Rome–Italy, Tokyo–Japan. (3) Plurality: car–cars, dog–dogs, child–children. (4) Verb tense: walk–walked–walking, run–ran–running. (5) Comparative/superlative: good–better–best, bad–worse–worst. (6) Occupation: teach–teacher, sing–singer, paint–painter. These patterns are not explicitly encoded during training; they emerge from distributional statistics. The explanation is that words related by a consistent grammatical or semantic transformation tend to appear in parallel syntactic contexts: "the king rules" and "the queen rules" are both valid, so "king" and "queen" appear in similar contexts with a systematic distributional shift corresponding to gender. Projecting these contexts into embedding space via Word2Vec or GloVe produces parallel vector offsets. Mathematically, if the co-occurrence matrix has block structure where related words have shifted contexts, the factorization will capture these shifts as vector directions. The linearity is approximate rather than exact: not all analogies work perfectly, and the best answer depends on corpus statistics and training hyperparameters. Nonetheless, the robustness of linear patterns across different training runs, corpora, and methods suggests they reflect genuine linguistic regularities rather than artifacts.

4.5.2 Evaluation Metrics

Evaluating embedding quality requires distinguishing intrinsic evaluation (measuring embedding properties directly) from extrinsic evaluation (measuring downstream task performance). Intrinsic evaluations are faster and provide diagnostic insights into what embeddings capture. Word similarity tasks provide human-annotated

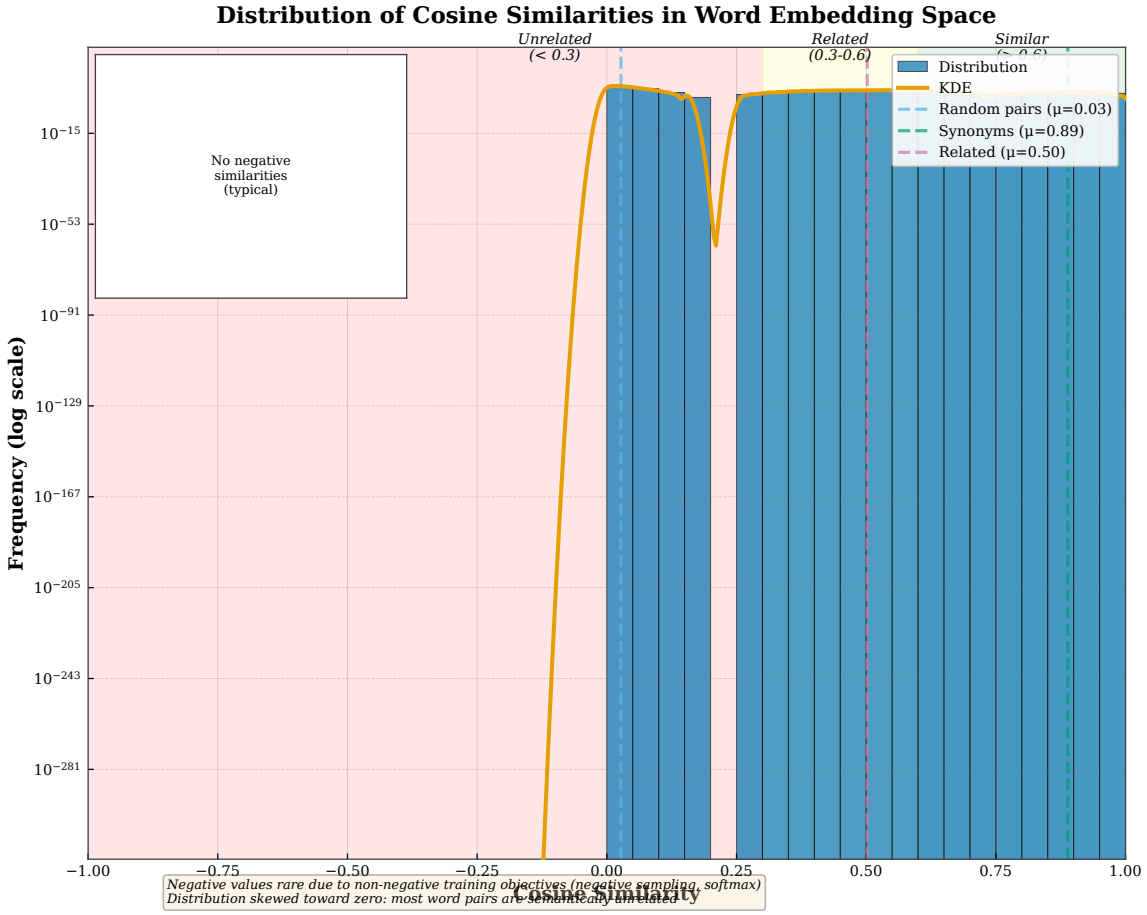

Figure 4.14: Vector arithmetic in 3D embedding space. The famous king–man+woman=queen relationship is visualized as a parallelogram: the vector from "man" to "king" (blue arrow, encoding "royalty") is approximately equal to the vector from "woman" to "queen" (red arrow). When we compute $\mathbf{e}_{\text{king}} - \mathbf{e}_{\text{man}} + \mathbf{e}_{\text{woman}}$ (green arrow), we arrive near $\mathbf{e}_{\text{queen}}$. Similar parallelograms exist for other analogies: Paris–France and Rome–Italy (country-capital), walk–walked and talk–talked (verb tense). These linear substructures demonstrate that geometric directions in embedding space correspond to semantic relationships, enabling algebraic manipulation of meanings.

similarity scores for word pairs and measure correlation between human judgments and embedding cosine similarities. Standard datasets include WordSim-353 (353 pairs, scale 0–10), SimLex-999 (999 pairs emphasizing true similarity over relatedness), and RG-65 (65 pairs from Miller and Charles, 1991). Spearman rank correlation between human scores and cosine similarities typically ranges from 0.65 to 0.75 for Word2Vec and GloVe, indicating strong but imperfect agreement with human intuitions. Analogy tasks measure the percentage of correct answers using the vector arithmetic procedure described above. The Google Analogy Dataset contains 19,544 questions across 14 categories (8 semantic, 6 syntactic). Word2Vec and GloVe achieve 60–80% accuracy depending on corpus size and dimension, vastly exceeding random guessing. Clustering quality can be assessed by performing k-means clustering on embeddings and measuring purity: the percentage of clusters dominated by a single semantic category (when categories are available, e.g., WordNet synsets). High purity indicates that embeddings group semantically related words without supervision. Extrinsic evaluations use embeddings as input features for supervised tasks. Common benchmarks include sentiment analysis (SST-2, IMDB reviews), named entity recognition (CoNLL 2003), part-of-speech tagging, textual entailment (SNLI), and machine translation. The evaluation protocol trains a task-specific model using pre-trained embeddings (often frozen or fine-tuned) and measures task accuracy. Improvements from 5 to 15 percentage points over random embeddings are typical. Importantly, intrinsic and extrinsic metrics sometimes disagree: embeddings

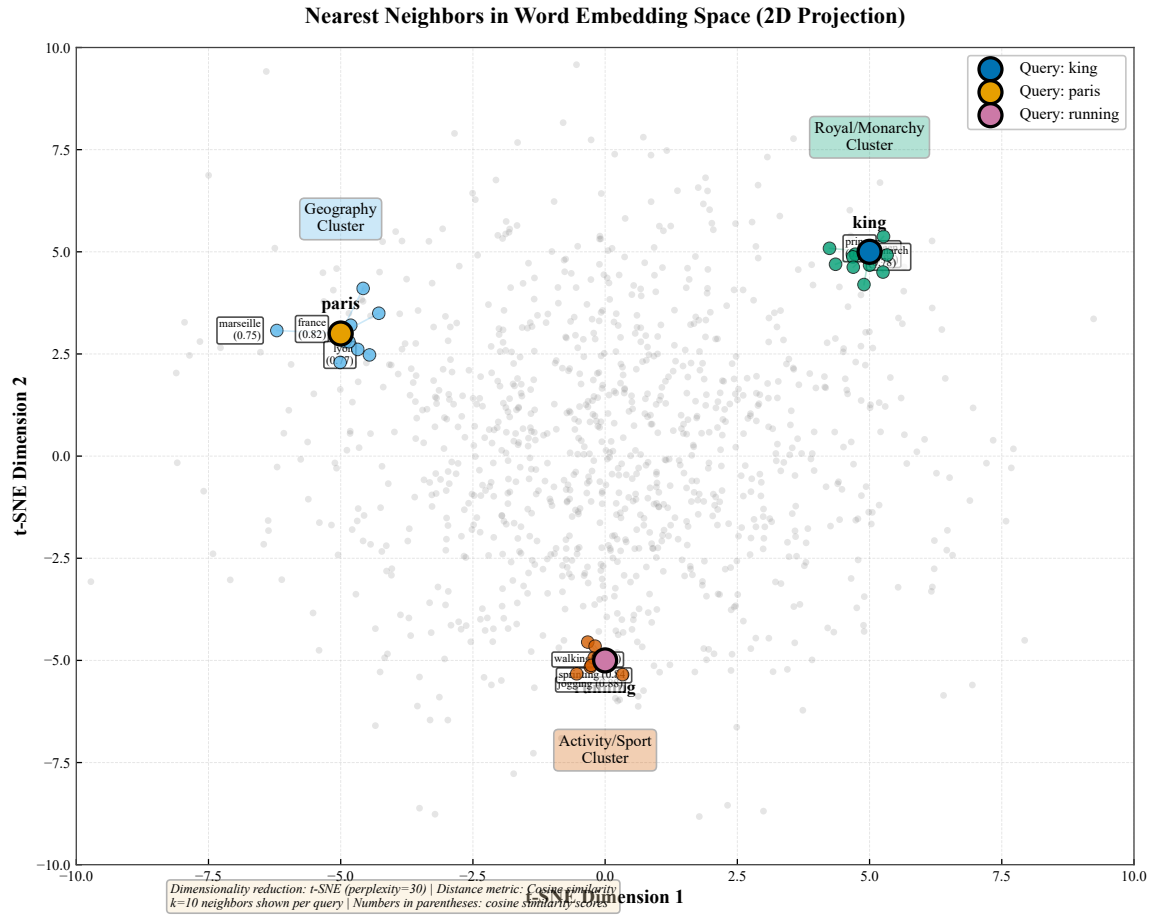

Figure 4.15: Analogy categories and performance. Panel A shows semantic analogies with examples: country-capital (Paris:France::Rome:Italy, 78% accuracy), gender (king:queen::man:woman, 82% accuracy), currency (dollar:USA::euro:Europe, 65% accuracy). Panel B displays syntactic analogies: verb tense (walk:walked::run:ran, 71% accuracy), plurality (dog:dogs::cat:cats, 88% accuracy), comparative (good:better::bad:worse, 73% accuracy). Panel C plots accuracy by category showing variation across relationship types. Panel D illustrates failure cases: rare words (barrister:barristeress, 0% accuracy due to insufficient training data), ambiguous analogies (bank:money::river:?, could be water or shore), and cultural specificity (chopsticks:China::fork:?, depends on corpus).

with higher analogy accuracy may not yield better sentiment analysis performance, suggesting that different tasks benefit from different embedding properties. Domain mismatch also affects performance: embeddings trained on Wikipedia work well for formal text but poorly for social media or domain-specific text (medical, legal), where in-domain embeddings are preferable.

4.5.3 Limitations and Biases

Despite their success, static word embeddings suffer from fundamental limitations. The most significant is the polysemy problem: each word type receives exactly one embedding regardless of context, conflating all word senses. The word “bank” has (at least) two unrelated meanings: financial institution and river edge. These senses appear in entirely different contexts (“bank account”, “deposit money” vs. “river bank”, “water edge”), yet the embedding for “bank” averages across both, producing a representation that captures neither sense well. Similarly, “lead” can be a metal (Pb), a verb (to guide), or a noun (being ahead), “bat” can be an animal or sports equipment, and “date” can be a calendar day, a romantic meeting, or a fruit. For polysemous words, the single embedding represents a centroid of all senses weighted by their corpus frequencies, which may not correspond to any actual sense. This limitation is severe for words with highly disparate senses. The solution is contextual

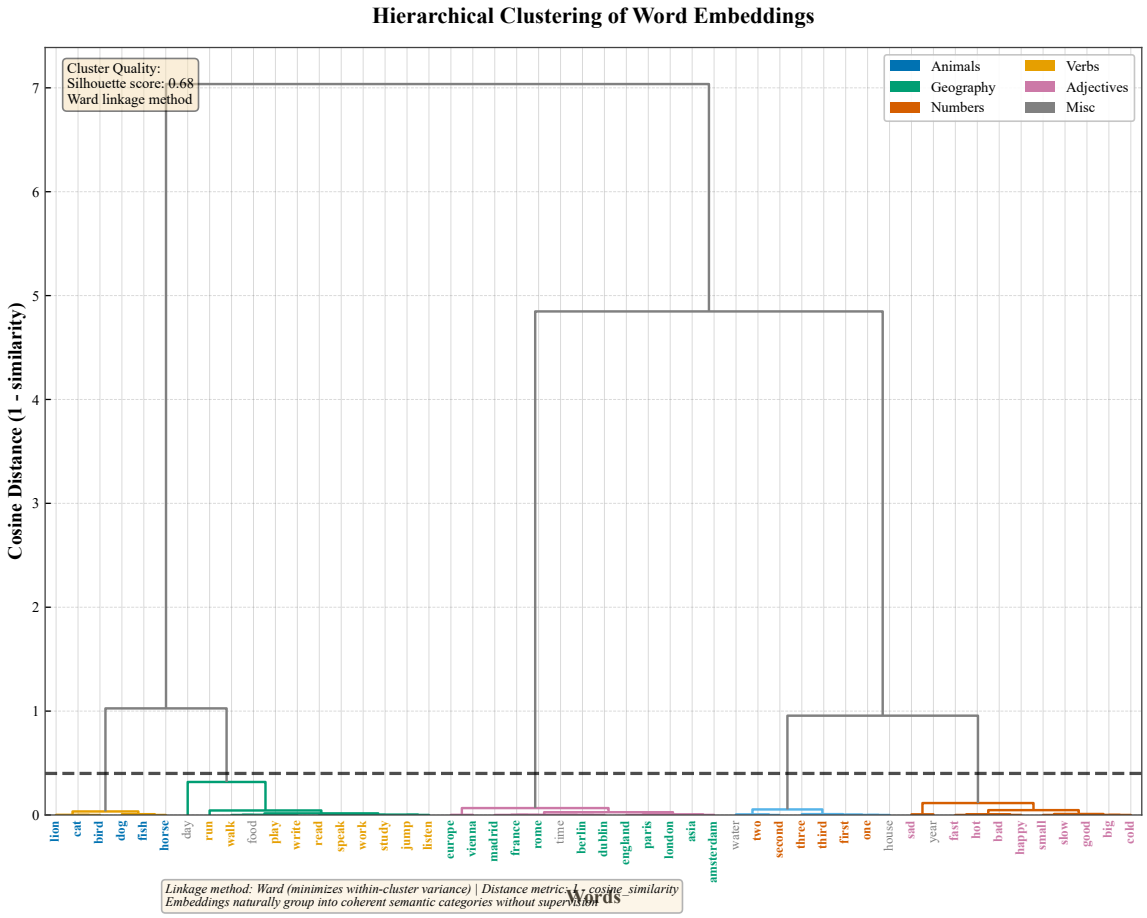

Figure 4.16: Embedding evaluation results across intrinsic and extrinsic metrics. Panel A shows word similarity correlation with human judgments: Word2Vec achieves Spearman $\rho = 0.68$ on WordSim-353, $\rho = 0.71$ on SimLex-999; GloVe achieves $\rho = 0.66$ and $\rho = 0.73$ respectively, indicating comparable performance. Panel B plots analogy accuracy versus embedding dimension: performance increases from 45% at $d = 50$ to 74% at $d = 300$, then plateaus, suggesting $d = 300$ is sufficient for most tasks. Panel C displays downstream task performance (accuracy gain over random embeddings): sentiment analysis (+12%), NER (+8%), POS tagging (+6%), showing that embeddings substantially improve supervised learning. Panel D examines training corpus size versus quality: performance increases logarithmically with corpus size, improving rapidly up to 1B tokens then saturating around 10B tokens.

embeddings (ELMo, BERT), covered in Chapter **??**, which compute different representations for each word occurrence based on surrounding context. The out-of-vocabulary (OOV) problem affects word-level embeddings: words not seen during training have no representation. For rare morphological variants, proper nouns, or technical terms, the model cannot make predictions. FastText (Section 4.6) partially addresses this via character n-grams, and BPE tokenization (Chapter **??**) eliminates it entirely by working at subword level. Static embeddings are also language-specific: separate embeddings must be trained for each language, and cross-lingual transfer requires alignment techniques. Beyond technical limitations, embeddings encode societal biases present in training corpora. Bolukbasi et al. (2016) demonstrated that Word2Vec embeddings trained on Google News exhibit gender bias: $\text{sim}(\text{doctor}, \text{man}) > \text{sim}(\text{doctor}, \text{woman})$ and $\text{sim}(\text{nurse}, \text{woman}) > \text{sim}(\text{nurse}, \text{man})$, reflecting occupational stereotypes. Analogies like man : computer programmer :: woman : x yield $x =$ homemaker, embedding problematic associations. Racial and ethnic biases similarly appear, with names associated with certain demographics receiving more positive or negative sentiment. These biases are not artifacts of the algorithm but faithful representations of statistical patterns in text: if doctors are mentioned with "he" more often than "she" in news articles, embeddings will reflect this asymmetry. The concern is that biases amplify when

embeddings are deployed in real-world systems for hiring, lending, or criminal justice, potentially perpetuating discrimination. Mitigation strategies include debiasing algorithms (subtracting gender direction), careful corpus curation, and awareness of bias in downstream applications, but no solution is perfect. Ethical deployment of embeddings requires acknowledging these limitations and implementing safeguards.

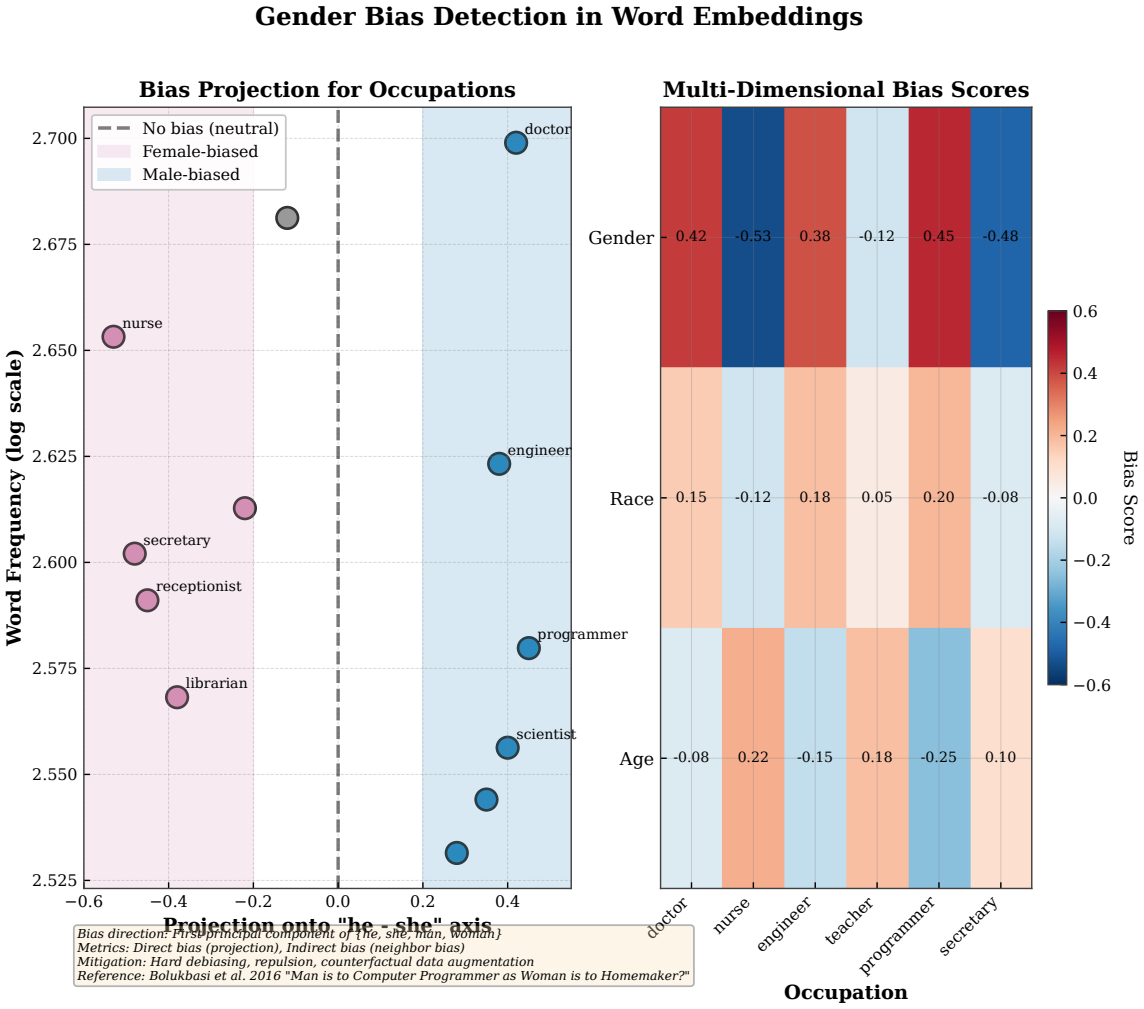

Figure 4.17: Limitations and biases in static word embeddings. Panel A illustrates the polysemy problem: "bank" has two unrelated meanings (financial institution and river edge) that appear in distinct contexts, yet receive a single averaged embedding (purple) that lies between the two sense-specific clusters (blue and orange), capturing neither sense accurately. Panel B visualizes gender bias in occupation embeddings: male-associated occupations (doctor, engineer, programmer) cluster near "man", while female-associated occupations (nurse, teacher, homemaker) cluster near "woman", reflecting and potentially amplifying societal stereotypes. Panel C shows bias amplification: corpus statistics exhibit moderate gender bias (55–45 ratio), but embeddings exhibit stronger bias (70–30 ratio) due to nonlinear effects of training. Panel D displays the OOV problem: vocabulary coverage saturates around 95% for 50k words, leaving 5% of text tokens unrepresented, with the long tail of rare words and proper nouns lacking embeddings.

4.6 FastText and Subword Embeddings

Word2Vec and GloVe treat words as atomic units, assigning each vocabulary entry a distinct embedding with no parameter sharing between related words. This atomicity causes two problems: (1) Out-of-vocabulary (OOV) words have no representation, preventing the model from processing novel words, proper nouns, or

morphological variants absent from training data. (2) Morphological structure is ignored: "teach", "teacher", "teaching", and "unteachable" are treated as unrelated despite sharing the root "teach", forcing the model to learn their semantics independently even though morphology provides strong signal. Bojanowski et al. (2017) introduced FastText, which addresses these limitations by representing words as bags of character n-grams. For example, with $n \in \{3,4,5\}$ and boundary markers $\langle$ and $\rangle$, the word "where" is decomposed into $\{\langle$wh, whe, her, ere, re$\rangle\}$ (trigrams) plus $\{\langle$whe, wher, here, ere$\rangle\}$ (4-grams) plus $\{\langle$wher, where, here$\rangle\}$ (5-grams), plus the full word $\langle$where$\rangle$ itself. Each n-gram receives its own embedding, and the word embedding is the sum of its n-gram embeddings. This representation enables generalization: novel words can be embedded via their n-grams, and morphologically related words share n-gram embeddings, allowing knowledge transfer. FastText retains the Skip-gram architecture and training procedure of Word2Vec but replaces the word lookup with n-gram aggregation. The trade-off is increased model size: where Word2Vec has $|\mathcal{V}| \times d$ parameters, FastText has $N_{\text{ngrams}} \times d$ parameters where $N_{\text{ngrams}} \sim 2{,}000{,}000$ for typical settings, requiring more memory. However, this cost is justified for morphologically rich languages (Turkish, Finnish, German) and domains with many rare words (biomedical text, social media). FastText also enables cross-lingual transfer when languages share scripts, as cognates and loanwords share n-grams.

4.6.1 Character N-grams

FastText represents each word as the sum of embeddings for its character n-grams: given a word w, let $G(w)$ denote the set of character n-grams with $n \in [n_{\min}, n_{\max}]$ plus the full word itself (to retain word-level information), with boundary markers $\langle$ and $\rangle$ prepended and appended to distinguish prefixes and suffixes. For $n_{\min} = 3$ and $n_{\max} = 6$, "where" becomes $\langle$where$\rangle$ with n-grams including $\{\langle$wh, whe, her, ere, re$\rangle$, $\langle$whe, wher, here, ere$\rangle$, $\langle$wher, where, here$\rangle$, $\langle$where$\rangle\}$. The n-gram embedding matrix $\mathbf{E}_{\text{ng}} \in \mathbb{R}^{N \times d}$ has typically $N \sim 2$ million entries for English with $n \in [3,6]$, and the embedding for word w is computed as $\mathbf{e}_w = \sum_{g \in G(w)} \mathbf{E}_{\text{ng}}[g,:]$, aggregating information from all constituent n-grams. During training, FastText uses the Skip-gram objective to predict context words, with gradients flowing back to all n-gram embeddings in $G(w_t)$, updating shared n-grams across words: training on "teacher" updates embeddings for $\langle$te, tea, each, ache, cher, her$\rangle$, which also appear in "teach", "teaching", "preacher", enabling morphological transfer. For OOV words at test time, we compute $G(w_{\text{oov}})$ and sum the corresponding n-gram embeddings: "unseen" decomposes to n-grams like "uns", "nse", "see", "een" appearing in training words "unset", "nonsense", "foresee", "seen", allowing approximate representation. This OOV handling is especially valuable for morphologically complex languages where each root generates dozens of inflected forms. The computational cost is approximately 50× slower embedding lookup than Word2Vec (summing $|G(w_t)| \approx 50$ n-grams versus 1 lookup), but providing substantial generalization benefits that justify the overhead.

4.6.2 Morphology and Cross-Lingual Transfer

FastText's n-gram representation naturally captures morphological structure. Morphologically related words share character sequences corresponding to roots, prefixes, and suffixes, causing their embeddings to be similar even if the words never co-occur. For example, "teach" has n-grams $\{\langle$te, tea, each, ach, ch$\rangle, \ldots\}$, "teacher" adds $\{$che, her, er$\rangle\}$, and "teaching" adds $\{$chi, hin, ing, ng$\rangle\}$. The shared n-grams $\{\langle$te, tea, each, ach, ch$\rangle\}$ cause all three to have similar embeddings, with additional n-grams encoding the morphological modifications. This is especially powerful for morphologically rich languages like Turkish or Finnish, where a single root can generate hundreds of inflected forms. Without n-gram sharing, each form would require independent learning; with FastText, learning about one form transfers to all others sharing n-grams. Compound words also benefit: German compounds like "Fußballweltmeisterschaft" (football world championship) decompose into n-grams overlapping with constituent words "Fußball", "Welt", "Meisterschaft", enabling compositional understanding. FastText enables limited cross-lingual transfer when languages share scripts. Romance languages (French, Spanish, Italian) share many cognates and loanwords ("hospital", "hôpital", "hospital", "ospedale") with overlapping n-grams, allowing zero-shot transfer: embeddings trained on Spanish can partially represent French words via shared n-grams. Slavic languages (Russian, Polish, Czech) written in Cyrillic or Latin exhibit similar overlap. This cross-lingual property is limited—grammatical differences, false friends, and divergent semantics

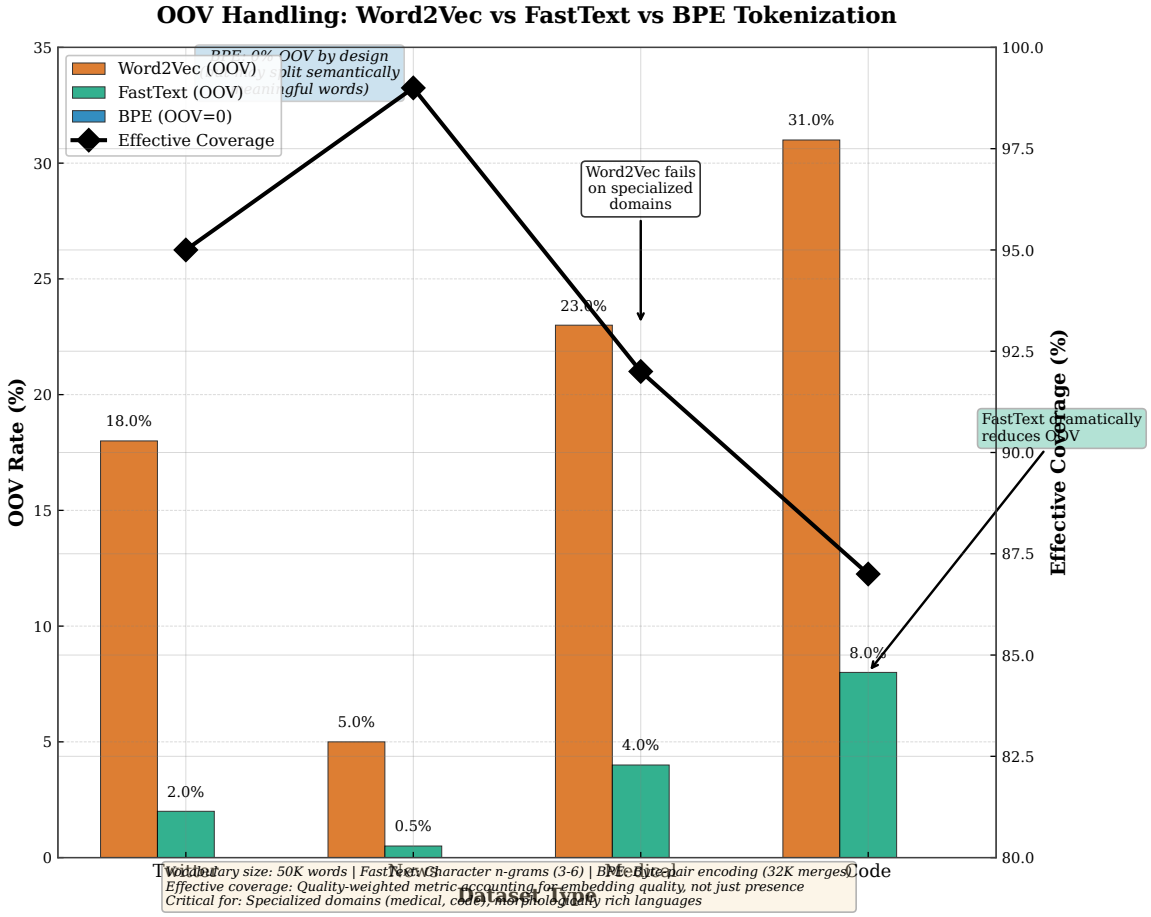

Figure 4.18: FastText architecture and character n-gram embeddings. Panel A shows n-gram extraction for "where" with $n \in \{3,4,5\}$ and boundary markers: $\{\langle$wh, whe, her, ere, re$\rangle, \langle$whe, wher, here, ere$\rangle, \langle$wher, where, here$\rangle, \langle$where$\rangle\}$, totaling 13 n-grams. Panel B illustrates embedding aggregation: each n-gram has an embedding vector, and the word embedding is their sum. Panel C demonstrates OOV word representation: the novel word "wherever" (not in training vocabulary) can be embedded via its n-grams $\{\langle$wh, whe, her, ..., ver, ere$\rangle\}$, most of which appeared in training words like "where", "here", "never". Panel D visualizes morphological similarity: words sharing roots ("run", "running", "ran", "runner") cluster together because they share n-grams ("run", "unn", "nni"), enabling morphological generalization.

prevent full transfer—but provides non-zero baseline performance for low-resource languages. The limitation of character n-grams is that they ignore word order and long-range dependencies within words: n-grams are local substrings up to length 6, missing patterns spanning entire words. Additionally, FastText is less effective for ideographic scripts like Chinese or Japanese, where characters represent morphemes rather than phonemes, and character n-grams have limited morphological meaning. For such languages, morpheme-based or radical-based decompositions are more appropriate. Despite these limitations, FastText has become the default choice for applications requiring robustness to rare words or morphological variation.

4.6.3 Byte-Pair Encoding (BPE) Embeddings

While FastText uses fixed-length character n-grams, Byte-Pair Encoding (BPE) learns variable-length subword units that balance word-level semantics and character-level robustness. BPE tokenization (covered in Chapter **??**) iteratively merges the most frequent adjacent character pairs in the corpus, building a vocabulary of subword units ranging from single characters to full words. For example, BPE might learn units $\{$un, teach, able, ing, ...$\}$ from which words are composed: "unteachable" $\rightarrow$ [un, teach, able]. These subword units are then embedded: each unit receives its own embedding vector, and the word representation is

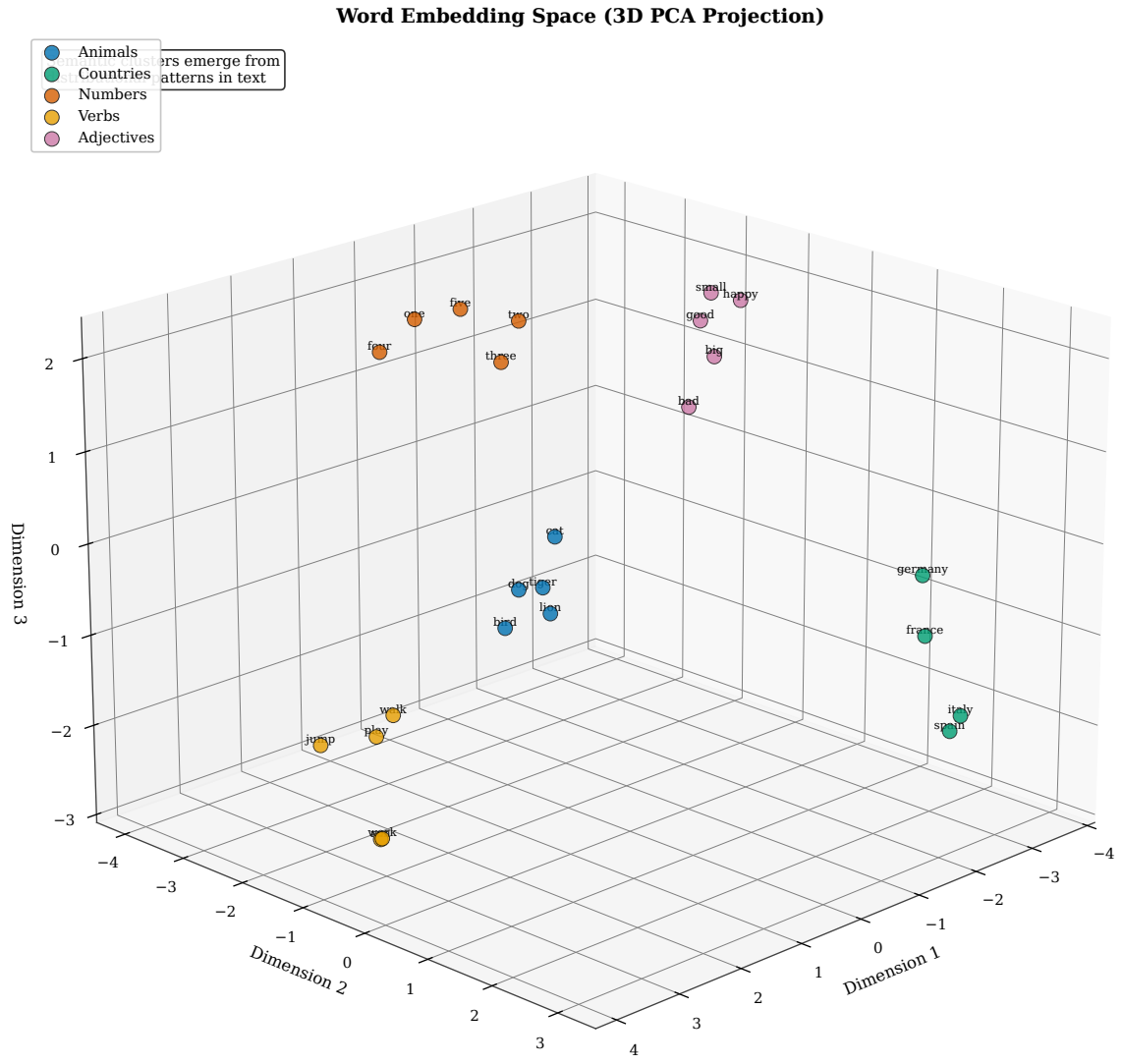

Figure 4.19: Morphological clustering and cross-lingual transfer with FastText. Panel A shows morphological clusters in embedding space: words sharing roots (“teach”, “teacher”, “teaching”, “unteachable”) are nearby due to shared n-grams, despite different surface forms. Panel B illustrates cross-lingual n-gram overlap: Romance language cognates (“hospital” in English, “hôpital” in French, “hospital” in Spanish) share n-grams $\{\text{hos}, \text{osp}, \text{spi}, \text{pit}, \text{tal}\}$, enabling partial cross-lingual transfer. Panel C compares performance on rare words: FastText (blue) maintains high accuracy for words with < 100 corpus occurrences, while Word2Vec (red) degrades rapidly due to insufficient statistics. Panel D displays results on morphologically rich languages: FastText achieves 73% analogy accuracy on Turkish (vs. 52% for Word2Vec) and 69% on Finnish (vs. 48%), demonstrating the benefit of n-gram sharing for agglutinative morphology.

the sequence (or sum) of its subword embeddings. This approach combines advantages of word-level and character-level representations: frequent words are often single units (preserving semantic integrity), while rare words decompose into subwords (enabling OOV handling). BPE has become the standard tokenization method for modern LLMs (GPT, BERT, LLaMA) due to its vocabulary efficiency: a BPE vocabulary of 30,000–50,000 subwords covers virtually all text with minimal OOV rate (typically $< 0.1\%$), compared to word-level vocabularies requiring 100,000+ entries with 5–10% OOV. For embeddings, BPE units are learned during tokenization, then embedded via the same methods as words (Word2Vec, GloVe, or end-to-end training). The embedding matrix $\mathbf{E} \in \mathbb{R}^{|\mathscr{V}_{\text{BPE}}| \times d}$ has dimensionality $|\mathscr{V}_{\text{BPE}}| \approx 50{,}000$, intermediate between word-level ($|\mathscr{V}| \approx 50{,}000$ but with OOV issues) and character-n-gram ($N \approx 2{,}000{,}000$). BPE embeddings are contextual within the subword sequence but static across occurrences, inheriting the polysemy limitation of static embeddings. Modern LLMs learn BPE embeddings end-to-end with the language model objective, obviating separate pre-training, but the BPE vocabulary itself is typically fixed before training to ensure consistent tokenization.

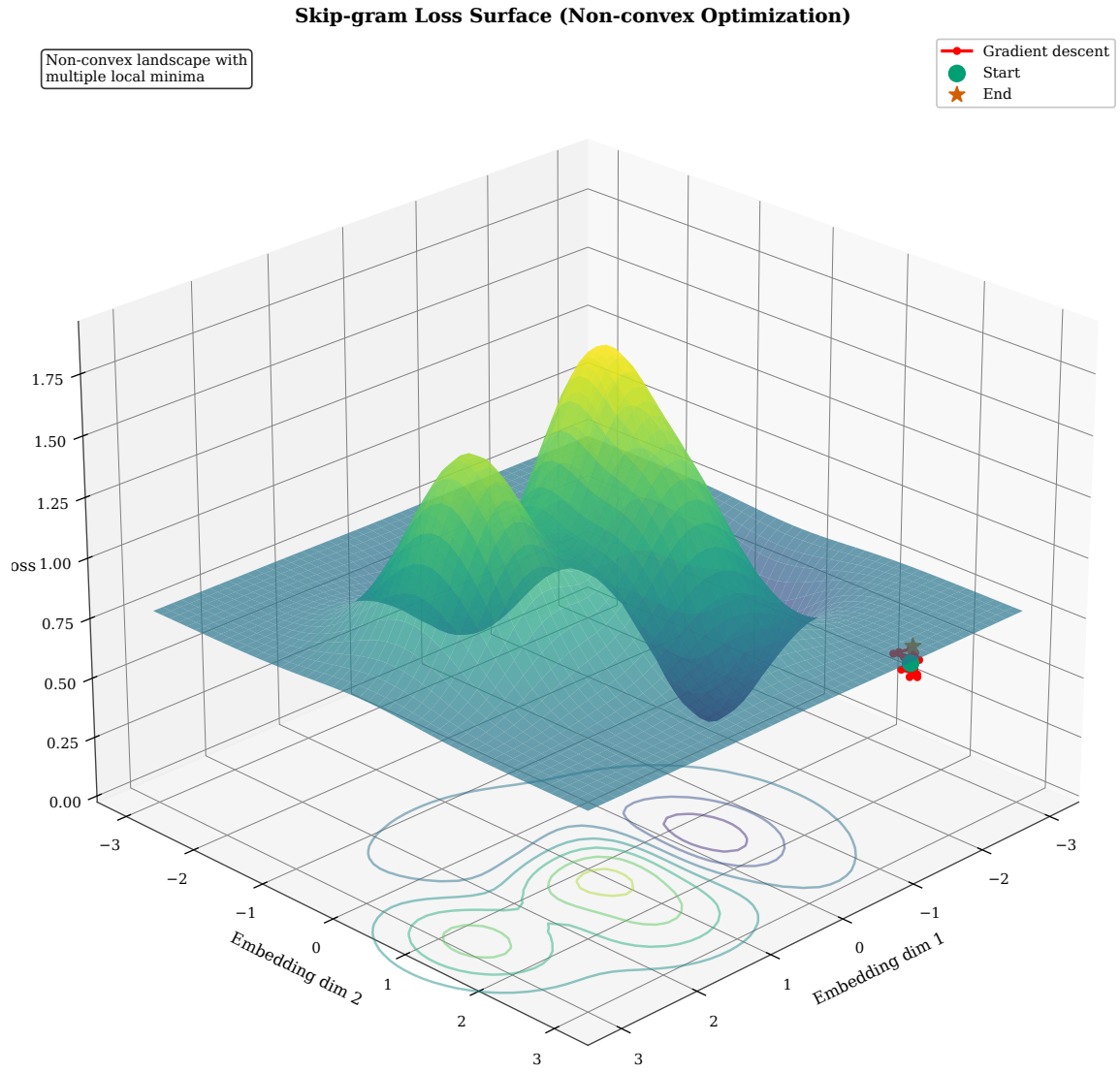

Figure 4.20: Comparison of word-level, FastText, and BPE embeddings across key dimensions. Word-level embeddings (blue) have vocabulary size $|\mathcal{V}| \approx 50{,}000$, suffer 5–10% OOV rate, and store $|\mathcal{V}| \times d \approx 15$M parameters. FastText (orange) has $N_{\text{ngrams}} \approx 2{,}000{,}000$ subunits, achieves near-zero OOV via character n-grams, but requires $2{,}000{,}000 \times d \approx 600$M parameters. BPE (green) balances with $|\mathcal{V}_{\text{BPE}}| \approx 50{,}000$ subwords, $< 0.1\%$ OOV, and $50{,}000 \times d \approx 15$M parameters. Trade-offs: word-level is simplest but has OOV problems; FastText handles OOV but is memory-intensive; BPE achieves low OOV with moderate memory, making it the preferred choice for modern LLMs.

4.7 Using Embeddings in Language Models

Word embeddings serve as the input layer for neural language models, converting discrete token sequences into continuous representations that subsequent layers process. The embedding layer is a lookup table: given a sequence of token IDs $[w_1, w_2, \ldots, w_T] \in \{1, \ldots, |\mathcal{V}|\}^T$, it produces a sequence of embedding vectors $[\mathbf{e}_1, \mathbf{e}_2, \ldots, \mathbf{e}_T] \in \mathbb{R}^{T \times d}$ by indexing the embedding matrix $\mathbf{E} \in \mathbb{R}^{|\mathcal{V}| \times d}$. These embeddings are then fed to recurrent networks (Chapter **??**), convolutional networks, or Transformers (Chapter **??**) that aggregate them to predict the next word. A critical design choice is whether to initialize the embedding layer with pre-trained embeddings (Word2Vec, GloVe, FastText) or learn embeddings from scratch jointly with the language model objective. Pre-trained embeddings were standard practice from 2015–2018, providing a warm start that accelerated convergence and improved performance, especially with limited training data. However, modern large-scale language models (GPT, BERT, LLaMA) train embeddings end-to-end from random initialization, finding that massive data and compute budgets allow learning high-quality embeddings tailored to the prediction task without separate pre-training. Another important technique is weight tying, where the input embedding matrix $\mathbf{E}^{\text{in}}$ and output embedding matrix $\mathbf{E}^{\text{out}}$ share parameters, reducing the parameter count by $|\mathcal{V}| \times d$ (often 20–40% of total parameters) and providing a regularization effect that empirically improves performance. This section examines these architectural choices and their impact on next-word prediction quality.

4.7.1 Embedding Layer in Neural LMs

The embedding layer maps discrete token indices to continuous vectors, serving as the critical interface between symbolic input and neural computation. Given input sequence $\mathbf{w} = [w_1, \ldots, w_T]$ where $w_t \in \{1, \ldots, |\mathcal{V}|\}$ is the integer index of the t-th token, the embedding layer performs the lookup $\mathbf{e}_t = \mathbf{E}[w_t, :]$ for $t = 1, \ldots, T$, producing the embedding sequence $[\mathbf{e}_1, \ldots, \mathbf{e}_T]$ that subsequent neural layers process. This operation is mathematically equivalent to multiplying the one-hot vector $\mathbf{1}_{w_t}$ by the embedding matrix: $\mathbf{e}_t = \mathbf{E}^\top \mathbf{1}_{w_t}$, but direct indexing is computationally faster because it avoids the unnecessary multiplication with zeros. The embedding layer contains $|\mathcal{V}| \times d$ parameters; for $|\mathcal{V}| = 50{,}000$ and $d = 512$, this is 25.6 million parameters. For comparison, a 12-layer Transformer with $d = 512$ and $d_{\text{ff}} = 2048$ has approximately 110 million parameters, so embeddings constitute about 23% of the total. The embedding layer is fully differentiable: during backpropagation, gradients $\frac{\partial \mathcal{L}}{\partial \mathbf{e}_t}$ flow back from the language model, and the parameter gradient is $\frac{\partial \mathcal{L}}{\partial \mathbf{E}[w_t, :]} = \frac{\partial \mathcal{L}}{\partial \mathbf{e}_t}$, updating only the row corresponding to w_t. This sparse update is efficient: in a minibatch of B sequences of length T, at most $B \times T$ rows of $\mathbf{E}$ receive gradients, leaving most rows untouched. This sparsity is both a blessing (efficiency) and a curse (rare words receive few updates). The choice between pre-trained and randomly initialized embeddings depends on data availability. Pre-trained embeddings from Word2Vec or GloVe, learned on large corpora (Wikipedia, Common Crawl), transfer semantic knowledge to the language model. This transfer is especially beneficial when the LM training corpus is small (< 100M tokens) or domain-specific. The embeddings can be frozen (no updates during LM training), fine-tuned (initialized from pre-trained but updated), or used only as initialization (full training). Fine-tuning typically performs best, leveraging pre-trained semantics while adapting to the LM task. However, modern LLMs train on billions to trillions of tokens, providing sufficient data to learn embeddings from scratch. Random initialization allows the model to learn embeddings optimized specifically for next-word prediction, rather than general similarity, which can improve performance when data is abundant. Current best practice for models at scale is random initialization with joint training.

4.7.2 Weight Tying and Output Embeddings

Language models require two embedding-related matrices: the input embedding matrix $\mathbf{E}^{\text{in}} \in \mathbb{R}^{|\mathcal{V}| \times d}$ that converts tokens to vectors, and the output projection matrix $\mathbf{E}^{\text{out}} \in \mathbb{R}^{d \times |\mathcal{V}|}$ (or equivalently $\mathbb{R}^{|\mathcal{V}| \times d}$ transposed) that converts the final hidden state $\mathbf{h}_t \in \mathbb{R}^d$ to logits over vocabulary: $\mathbf{s}_t = \mathbf{E}^{\text{out}} \mathbf{h}_t \in \mathbb{R}^{|\mathcal{V}|}$. These two matrices serve different purposes (input encoding vs. output decoding) and traditionally were learned independently, requiring $2 \times |\mathcal{V}| \times d$ parameters total. Weight tying shares these matrices by setting $\mathbf{E}^{\text{out}} = (\mathbf{E}^{\text{in}})^\top$, reducing parameters by $|\mathcal{V}| \times d$ (a 50% reduction for embedding-related parameters), meaning that with weight tying the output logits are computed as $\mathbf{s}_t = (\mathbf{E}^{\text{in}})^\top \mathbf{h}_t$ where $\mathbf{s}_t[w]$ represents the unnormalized score for word w. Intuitively,

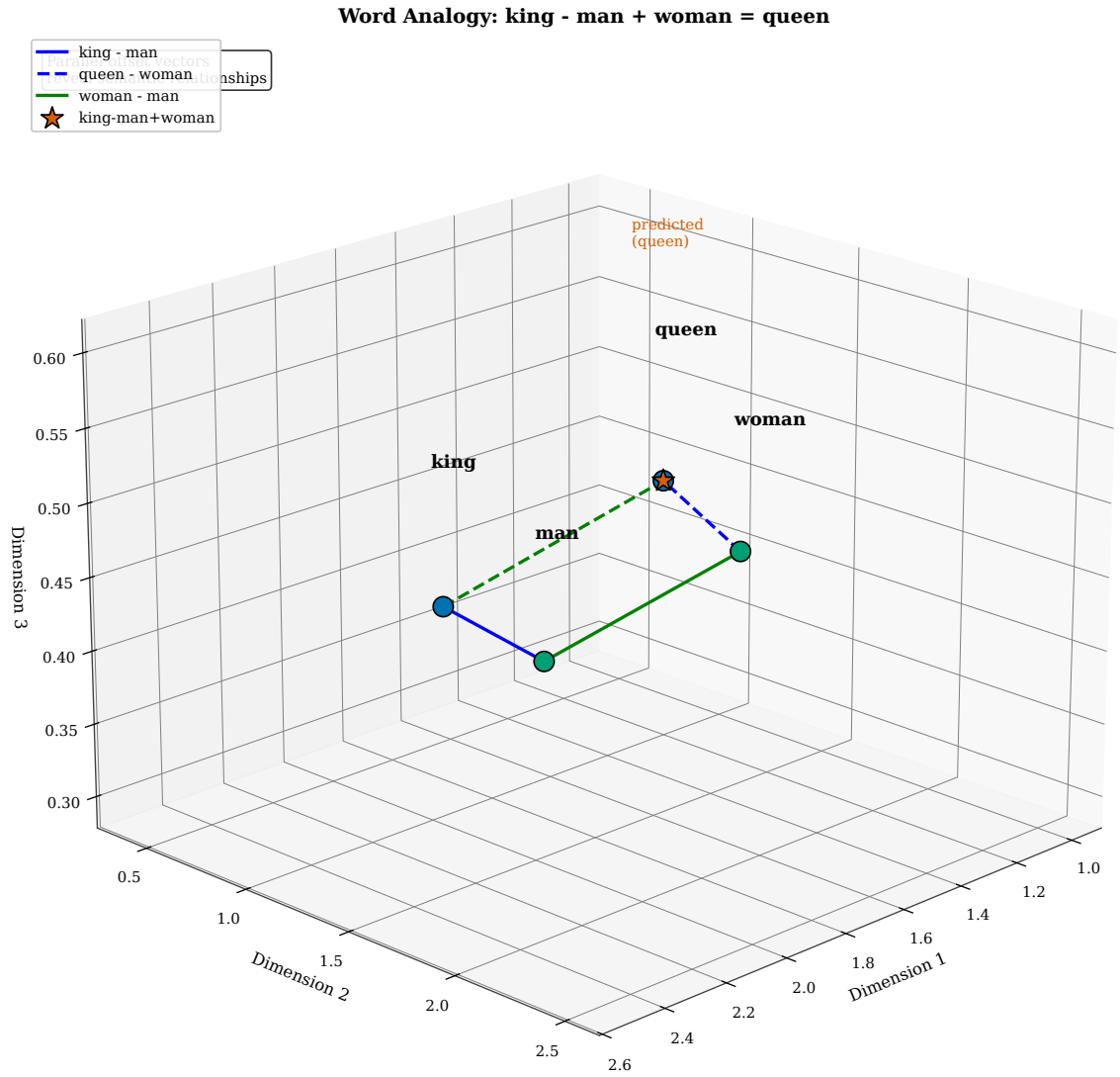

Figure 4.21: Embedding layer in neural language models. Panel A shows the architecture: token IDs $[15,342,27,891]$ are converted to embeddings $[\mathbf{e}_{15},\mathbf{e}_{342},\mathbf{e}_{27},\mathbf{e}_{891}]$ via lookup in $\mathbf{E}\in\mathbb{R}^{50000\times 512}$, then fed to the language model (RNN or Transformer). Panel B visualizes the embedding matrix as a heatmap: each row is a $d=512$-dimensional vector, with 50,000 rows (one per vocabulary word). Panel C compares convergence curves for pre-trained (blue) vs. random (orange) initialization: pre-trained embeddings provide faster initial convergence and lower final perplexity for small training corpora (< 100M tokens), but the gap narrows for large corpora (> 1B tokens) where random initialization learns task-specific representations. Panel D breaks down parameter counts: for a 110M parameter Transformer, embeddings ($|\mathcal{V}|\times d=25$M) account for 23%, while attention layers (40M), feed-forward layers (42M), and layer norms (3M) constitute the remainder.

weight tying enforces that words with similar input embeddings also have similar output logits, which is reasonable since words that appear in similar contexts (and thus have similar input embeddings) should also be predictable in similar contexts (and thus have similar output weights). Empirically, weight tying improves performance on language modeling benchmarks (reducing perplexity by 5–10%) and downstream tasks, likely due to a regularization effect: constraining output and input embeddings to be related reduces the model's capacity to overfit. The parameter reduction is substantial: for $|\mathcal{V}| = 50{,}000$ and $d = 512$, weight tying saves 25.6M parameters, which is significant for memory-constrained deployments. Weight tying requires that the hidden state dimension equals the embedding dimension ($d_{\mathbf{h}} = d_{\text{emb}}$); when these differ, a linear projection layer is inserted: $\mathbf{s}_t = (\mathbf{E}^{\text{in}})^\top \mathbf{W}_{\text{proj}} \mathbf{h}_t$ where $\mathbf{W}_{\text{proj}} \in \mathbb{R}^{d_{\text{emb}} \times d_{\mathbf{h}}}$. Modern Transformer language models (GPT-2, GPT-3, LLaMA) universally employ weight tying, considering it a best practice. The technique originated in Press and Wolf (2017) and was quickly adopted across the field. An additional benefit is interpretability: with weight tying, the output logit for word w is the dot product between the hidden state and the input embedding for w, providing a geometric interpretation of prediction as similarity in embedding space.

4.7.3 Improving Next-Word Prediction

Embeddings fundamentally improve next-word prediction by enabling generalization across similar words. In n-gram models (Chapter **??**), each context tuple received an independent probability distribution, with no parameter sharing. If the model observed "the cat sat on the mat" but not "the feline sat on the mat", it could not predict the latter. With embeddings, "cat" and "feline" have similar vector representations (cosine similarity ≈ 0.8), so contexts containing "cat" provide training signal for "feline". Mathematically, if the model learns parameters θ that predict well for contexts with "cat", and $\mathbf{e}_{\text{cat}} \approx \mathbf{e}_{\text{feline}}$, then the same parameters will predict reasonably for contexts with "feline" due to the continuous similarity. This generalization reduces data requirements: embeddings allow effective learning from smaller corpora by transferring knowledge across semantically related words. Quantitative improvements from embeddings are substantial. Bengio et al. (2003) demonstrated 30–50% perplexity reduction compared to n-gram baselines on the Penn Treebank benchmark. Modern comparisons show that even simple neural LMs with embeddings (single-layer LSTMs) outperform highly tuned n-gram models with Kneser-Ney smoothing. The dimensionality reduction from $|\mathcal{V}| \approx 50{,}000$ (one-hot) to $d \approx 300$ (embeddings) also reduces parameters dramatically: an n-gram model with trigram contexts requires storing $|\mathcal{V}|^3$ probabilities, while a neural LM with embeddings requires $|\mathcal{V}| \times d + O(d^2)$ parameters (embedding matrix plus model weights), a 99.9% reduction for typical vocabularies. This parameter efficiency translates to better generalization: fewer parameters mean less overfitting, especially on small datasets. The continuous nature of embeddings also enables gradient-based optimization, allowing end-to-end training of complex architectures (RNNs, Transformers) that would be intractable with discrete representations. In summary, embeddings serve as the foundational representation that enables modern neural language modeling, transforming next-word prediction from discrete lookup to continuous optimization in semantic space.

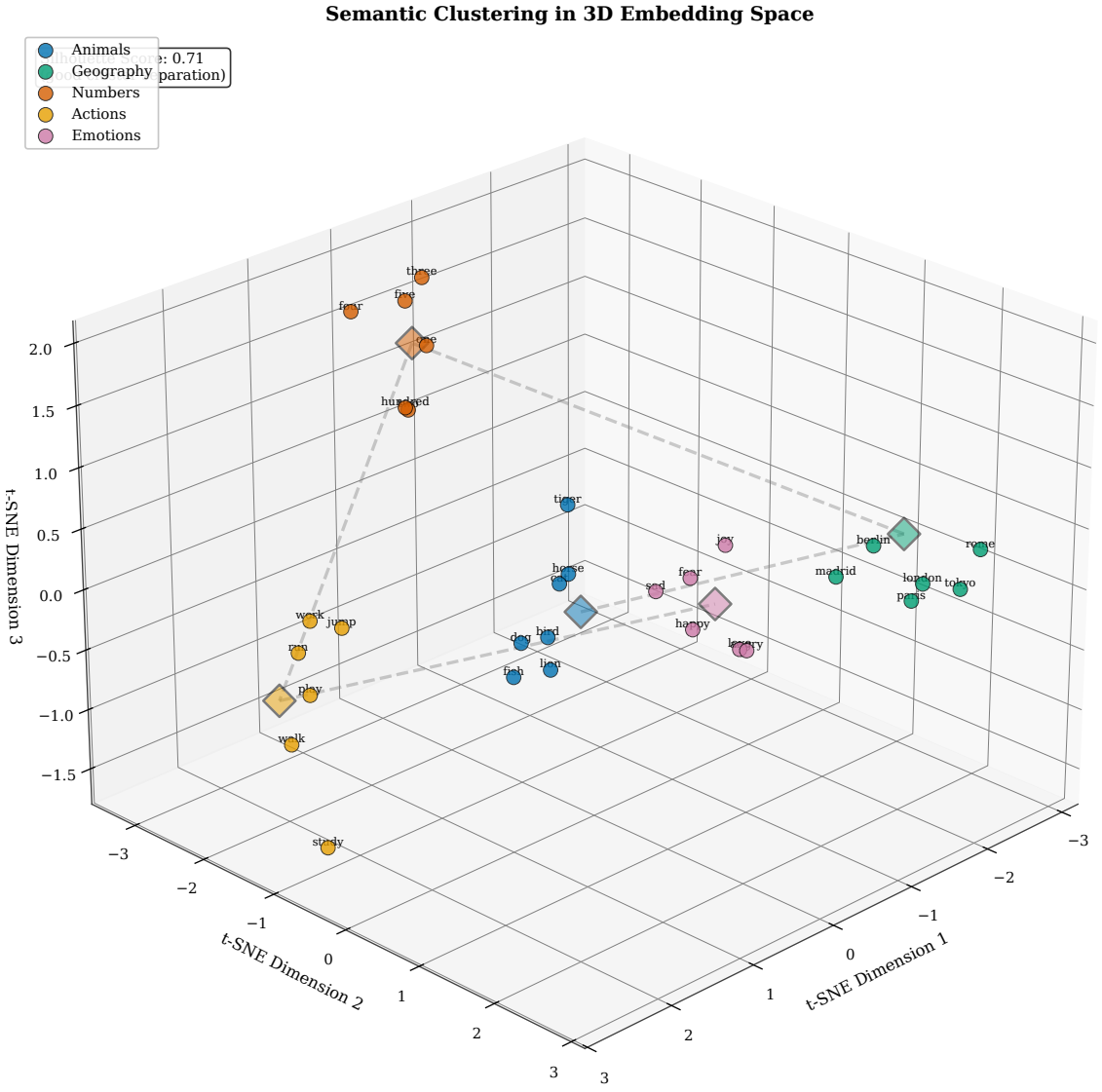

Figure 4.22: Weight tying between input and output embeddings. Without weight tying (left), the model maintains separate matrices $\mathbf{E}^{\text{in}} \in \mathbb{R}^{50000\times512}$ for input embeddings and $\mathbf{E}^{\text{out}} \in \mathbb{R}^{50000\times512}$ for output logits, totaling $2\times25.6\text{M} = 51.2\text{M}$ parameters. With weight tying (right), $\mathbf{E}^{\text{out}} = (\mathbf{E}^{\text{in}})^\top$, reducing to 25.6M parameters (50% reduction). The diagram shows token "cat" embedded via $\mathbf{E}^{\text{in}}$, processed by the LM to produce hidden state $\mathbf{h}_t$, then projected via $(\mathbf{E}^{\text{in}})^\top$ to compute logits. The parameter reduction visualization shows the freed memory (orange bars) when weight tying is applied, which can be reallocated to additional layers or larger hidden states, improving model capacity for a fixed parameter budget.

4.8 Context Representation in Word Embeddings

A fundamental question for next-word prediction is: how do we represent the context $w_1, \ldots, w_{t-1}$ to compute $P(w_t \mid w_1, \ldots, w_{t-1})$? In Chapter **??**, n-gram models represented context as a discrete tuple $(w_{t-n+1}, \ldots, w_{t-1})$ of the preceding $n-1$ words, treating each tuple as a separate symbolic key in a lookup table. This discrete representation provided no mechanism for generalization: similar contexts ("the black cat" vs. "the dark feline") were treated as completely unrelated. Word embeddings revolutionize context representation by mapping each word to a continuous vector $\mathbf{e}_w \in \mathbb{R}^d$, transforming the context sequence into a sequence of embeddings $[\mathbf{e}_1, \ldots, \mathbf{e}_{t-1}]$. This continuous encoding preserves semantic similarity: contexts containing similar words yield similar embedding sequences, enabling models to generalize across related contexts. However, embeddings alone do not solve the aggregation problem: we now have a variable-length sequence of vectors $[\mathbf{e}_1, \ldots, \mathbf{e}_{t-1}]$ but need a fixed-size representation to predict w_t. The methods in this chapter employ simple aggregation: CBOW averages context embeddings, and Skip-gram predicts each context word independently without aggregation. Neither approach captures sequential dependencies or long-range relationships. The polysemy problem remains: "bank" always receives the same embedding whether it means financial institution or river edge. These limitations motivate the recurrent architectures in Chapter **??**, where LSTM hidden states learn to compress variable-length embedding sequences into fixed-size context representations that preserve sequential structure and long-range dependencies. Transformers (Chapter **??**) further advance context representation through self-attention, computing contextual embeddings where each word's representation depends on all other words in the sequence, resolving polysemy by making "bank" receive different representations in "river bank" versus "savings bank".

How This Chapter Represents Context
The fundamental question in language modeling is: How do we represent the context $w_1, \ldots, w_{t-1}$ to predict w_t?

- **Context representation**: Word embeddings map discrete tokens to continuous vectors $\mathbf{e}_w \in \mathbb{R}^d$
- **Context encoding**: Each word in context has its own embedding; context is a sequence of embeddings $[\mathbf{e}_1, \ldots, \mathbf{e}_{t-1}]$
- **Limitation**: Static embeddings assign the same vector to each word occurrence regardless of context (polysemy problem)
- **Next chapter preview**: RNNs (Chapter **??**) will learn to aggregate embedding sequences into fixed-size context representations via recurrence

4.8.1 From N-grams to Embeddings

The progression from n-gram models to embedding-based models represents a fundamental shift in how context is encoded. In n-gram models, the context (w_{t-2}, w_{t-1}) for predicting w_t was represented by the tuple of word indices, say $(1523, 2847)$ if "black" is index 1523 and "cat" is index 2847. These indices are arbitrary integers with no inherent relationship: there is no mathematical reason why indices 1523 and 1524 should represent related concepts, and indeed they are typically assigned alphabetically or by frequency, destroying semantic structure. The model stores a separate probability distribution $P(\cdot \mid 1523, 2847)$ for each observed tuple. If the training corpus contains "the black cat sat" but not "the dark feline sat", the model has probability distributions for contexts $(1523, 2847)$ but not for $(1891, 3012)$ (assuming "dark" is 1891 and "feline" is 3012), forcing it to back off to unigram or bigram estimates with higher uncertainty. There is no mechanism to recognize that these contexts are semantically equivalent and should yield similar predictions. Word embeddings resolve this by mapping word indices to points in continuous semantic space $\mathbb{R}^d$. Now "black" maps to $\mathbf{e}_{1523} \in \mathbb{R}^d$ and "dark" maps to $\mathbf{e}_{1891} \in \mathbb{R}^d$, and crucially, these vectors are close together: $\|\mathbf{e}_{1523} - \mathbf{e}_{1891}\|$ is small (cosine similarity ≈ 0.7) because they appear in similar contexts. Similarly, $\mathbf{e}_{2847}$ (cat) and $\mathbf{e}_{3012}$ (feline) are nearby.

The context $(\text{black}, \text{cat})$ becomes the embedding sequence $[\mathbf{e}_{1523}, \mathbf{e}_{2847}]$, and the context $(\text{dark}, \text{feline})$ becomes $[\mathbf{e}_{1891}, \mathbf{e}_{3012}]$. Because these embedding sequences are close in $\mathbb{R}^d \times \mathbb{R}^d = \mathbb{R}^{2d}$, a model with continuous parameters can generalize: if it learns to predict "sat" after $[\mathbf{e}_{1523}, \mathbf{e}_{2847}]$, the same parameters will predict "sat" after $[\mathbf{e}_{1891}, \mathbf{e}_{3012}]$ due to the proximity in embedding space. This generalization is the key advantage: embeddings transform a discrete combinatorial space (where the number of possible $(n-1)$-word contexts is $|\mathcal{V}|^{n-1}$) into a continuous Euclidean space (where proximity implies similarity), enabling interpolation and parameter sharing across related contexts.

However, the transformation from discrete tuples to embedding sequences does not fully solve context representation. We now have a sequence of vectors $[\mathbf{e}_1, \ldots, \mathbf{e}_{t-1}]$ of varying length $t-1$, but most prediction models require fixed-size input. How do we aggregate this variable-length sequence into a single fixed-size context vector? The methods in this chapter employ naive aggregation strategies. CBOW averages the embeddings: $\bar{\mathbf{e}} = \frac{1}{t-1}\sum_{i=1}^{t-1} \mathbf{e}_i$, producing a single vector $\bar{\mathbf{e}} \in \mathbb{R}^d$ that represents the entire context. This averaging loses information about word order and long-range structure: the contexts "the cat sat on the mat" and "the mat sat on the cat" produce identical averaged embeddings despite opposite meanings. Skip-gram avoids aggregation entirely by predicting each context word independently: it uses $\mathbf{e}_{w_t}$ to predict w_{t+k} for each offset k, never forming a holistic context representation. While this enables efficient training, it does not provide a mechanism for encoding context as a whole. These limitations are acceptable for the local context windows ($K \leq 5$) used in Word2Vec and GloVe, but become severe for language modeling where context length can be hundreds or thousands of tokens. The polysemy problem remains unresolved: the word "bank" receives the same embedding $\mathbf{e}_{\text{bank}}$ in both "river bank" and "savings bank", despite the two meanings being unrelated. When "bank" appears in context, the model has no way to determine which sense is intended based solely on the static embedding. These challenges motivate the sequential architectures in Chapter **??**, where recurrent networks maintain a hidden state that evolves as the sequence is processed, learning to aggregate embeddings in a context-dependent manner that preserves word order and long-range dependencies while resolving polysemy through contextualization.

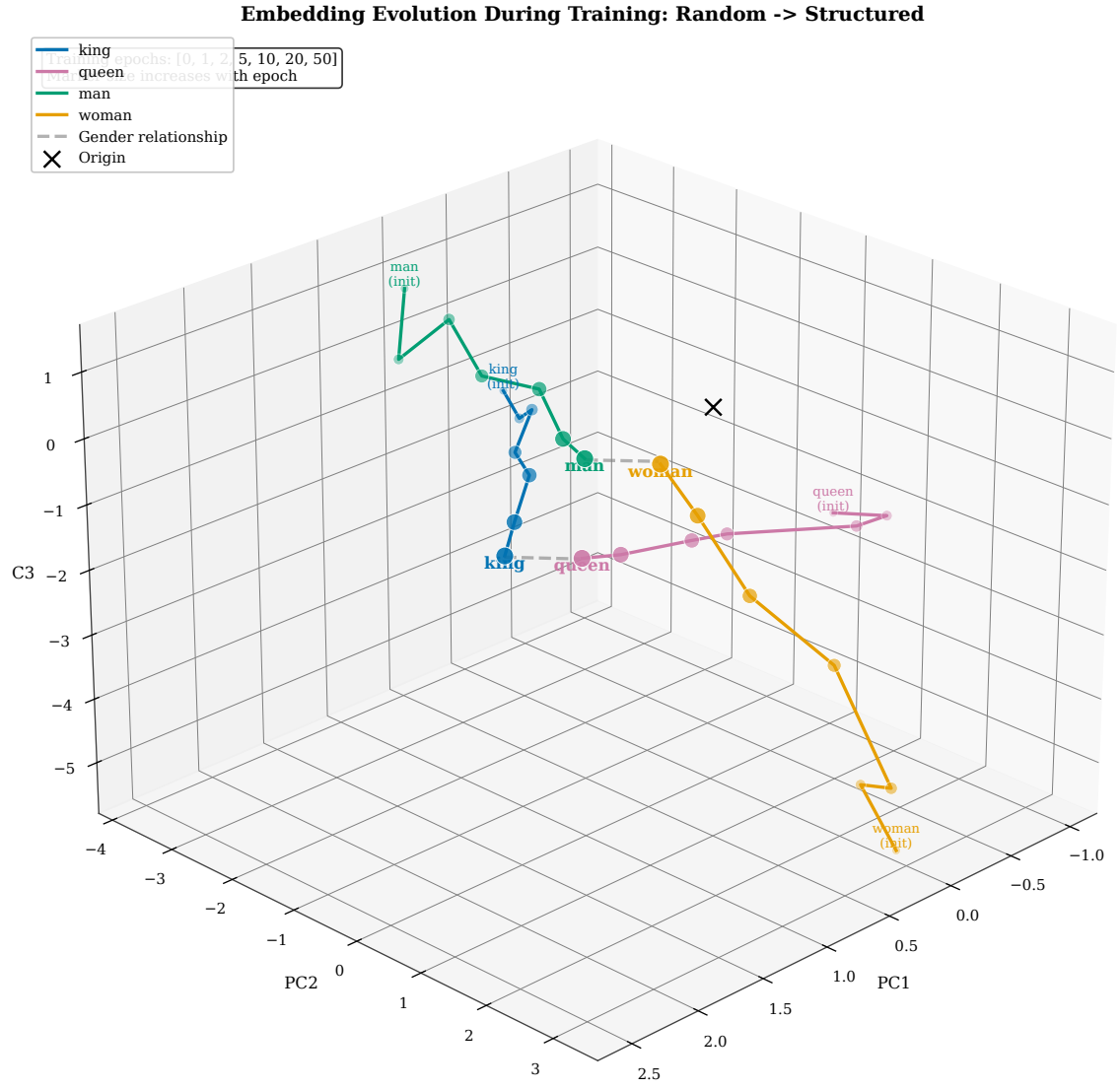

Figure 4.23: Evolution of context representation across chapters. Panel A shows n-gram discrete context tuples: "the black cat" is represented as integer indices $(3, 1523, 2847)$ with no similarity structure—"the dark feline" $(3, 1891, 3012)$ is treated as completely different despite semantic equivalence. Panel B displays word embedding sequences: the same contexts become $[\mathbf{e}_3, \mathbf{e}_{1523}, \mathbf{e}_{2847}]$ and $[\mathbf{e}_3, \mathbf{e}_{1891}, \mathbf{e}_{3012}]$, which are close in $\mathbb{R}^{3d}$ space due to similar individual word embeddings. Panel C illustrates CBOW's naive aggregation: averaging produces $\bar{\mathbf{e}} = (\mathbf{e}_3 + \mathbf{e}_{1523} + \mathbf{e}_{2847})/3$, collapsing the sequence to a fixed-size vector but losing word order. Panel D previews RNN context (Chapter 5): the hidden state $\mathbf{h}_{t-1}$ recursively compresses the entire history $[\mathbf{e}_1, \ldots, \mathbf{e}_{t-1}]$ into a fixed-size vector that preserves sequential dependencies, representing a qualitative advance in context encoding.

4.9 Summary

We can now predict better because:

- Similar words share similar embeddings, enabling generalization
- Vector arithmetic captures semantic relationships
- Dense representations reduce parameters by 99.9%
- Embeddings transfer learned knowledge across tasks

Next: Chapter **??** introduces recurrent neural networks to process embedding sequences...

This chapter introduced distributed representations as the solution to the discrete symbol problem that plagued n-gram models. By mapping words to continuous vectors in $\mathbb{R}^d$ where semantic similarity corresponds to geometric proximity, embeddings enable models to generalize across related contexts and dramatically reduce parameter counts. We examined three major approaches: matrix factorization methods (SVD on co-occurrence matrices) provide intuitive connections to distributional statistics but scale poorly; Word2Vec (Skip-gram and CBOW) learns embeddings efficiently through prediction tasks using negative sampling; and GloVe combines global co-occurrence statistics with prediction-based learning through weighted regression on log counts. All three methods produce embeddings exhibiting linear substructures where vector arithmetic captures semantic relationships, enabling analogy solving and demonstrating that geometry encodes meaning. FastText extends embeddings to character n-grams, enabling out-of-vocabulary handling and morphological generalization crucial for rare words and morphologically rich languages. We examined how embeddings integrate into neural language models as lookup tables that convert token sequences to vector sequences, with weight tying between input and output embeddings providing parameter reduction and performance improvements. Evaluation via intrinsic metrics (similarity, analogies) and extrinsic metrics (downstream tasks) confirms that embeddings substantially improve prediction quality, reducing perplexity by 30–50% compared to discrete representations. However, static embeddings suffer from the polysemy problem: each word type receives one embedding regardless of context, conflating multiple senses. They also encode societal biases present in training corpora, raising ethical concerns for deployment. These limitations motivate the contextual architectures in subsequent chapters: recurrent networks (Chapter **??**) will learn to aggregate embedding sequences into context-aware hidden states, and Transformers (Chapter **??**) will compute contextual embeddings where each word's representation depends dynamically on surrounding context, finally resolving polysemy.

Exercises

1. **One-hot Encoding Limitations.** Given vocabulary $\mathcal{V} = \{\text{cat}, \text{dog}, \text{fish}, \text{bird}\}$ with indices $\{1,2,3,4\}$, write the one-hot encoding vectors for "cat" and "dog". Compute their dot product $\mathbf{1}_{\text{cat}} \cdot \mathbf{1}_{\text{dog}}$ and cosine similarity. Explain why this representation cannot capture that "cat" and "dog" (both mammals) are more similar to each other than either is to "fish" (non-mammal), despite biological taxonomy.

2. **Distributional Hypothesis.** Consider the word "bank" in two sentences: "I deposited money at the bank" and "We sat on the river bank". Collect context words within a window of size $k = 2$ for each occurrence. Explain how the distributional hypothesis would suggest different meanings based on different contexts ($\{\text{deposited}, \text{money}, \text{at}\}$ vs. $\{\text{river}, \text{on}, \text{sat}\}$), and why static word embeddings fail to capture this polysemy, assigning a single vector $\mathbf{e}_{\text{bank}}$ that conflates both senses.

3. **Co-occurrence Matrix.** From the corpus: "the cat sat", "the cat ran", "the dog sat", construct the symmetric co-occurrence matrix $\mathbf{C}$ with window size $k = 1$. Compute the PMI for the pair ("cat", "sat") using $\text{PMI}(w_i, w_j) = \log \frac{C_{ij} \cdot N}{\sum_k C_{ik} \cdot \sum_k C_{kj}}$ where $N = \sum_{i,j} C_{ij}$. Explain what a positive PMI value indicates: that "cat" and "sat" co-occur more frequently than would be expected if they were statistically independent.

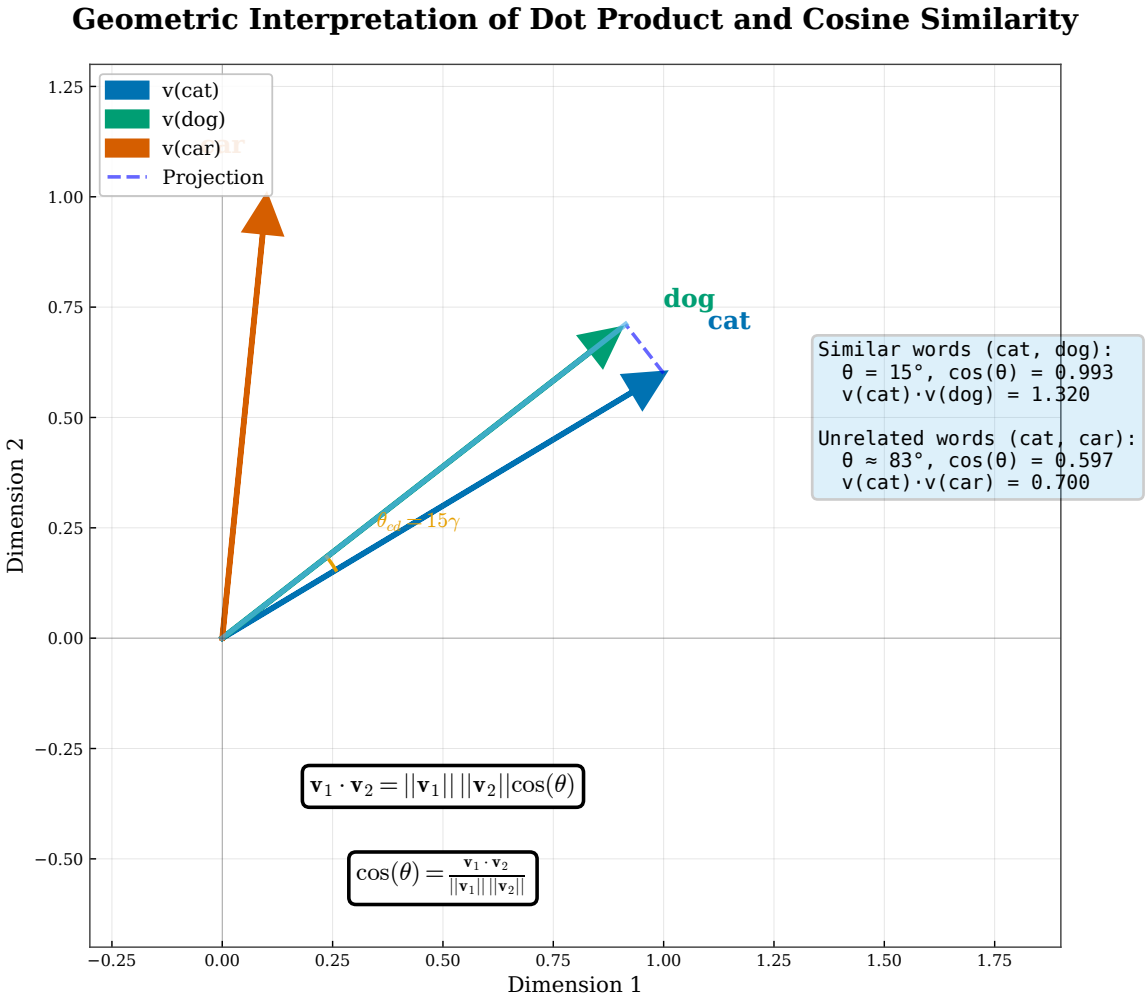

Figure 4.24: Geometric interpretation of dot product and cosine similarity.

4. **Skip-gram Training Pairs.** Given the sentence "the black cat sat there" with target word "cat" and context window $K = 2$, list all (target, context) training pairs generated by the Skip-gram architecture. Count the total number of pairs generated from this single sentence. Explain why Skip-gram generates $2K$ pairs per target word occurrence, providing richer supervision than CBOW which generates only one pair per target word.

5. **CBOW Forward Pass.** With embedding dimension $d = 2$, suppose context word embeddings are $\mathbf{e}_{\text{the}} = [0.2, 0.5]$, $\mathbf{e}_{\text{black}} = [0.3, 0.4]$, $\mathbf{e}_{\text{sat}} = [0.1, 0.6]$, and $\mathbf{e}_{\text{on}} = [0.4, 0.3]$. Compute the averaged context embedding $\bar{\mathbf{e}}$ for predicting target word "cat" in the sentence "the black cat sat on". If the output embedding for "cat" is $\mathbf{E}^{\text{out}}[\text{cat}, :] = [0.25, 0.45]$, compute the unnormalized score (dot product) $s_{\text{cat}} = \bar{\mathbf{e}} \cdot \mathbf{E}^{\text{out}}[\text{cat}, :]^\top$.

6. **Negative Sampling Efficiency.** Explain why computing the full softmax normalization $\sum_{v \in \mathcal{V}} \exp(s_v)$ over vocabulary size $|\mathcal{V}| = 50{,}000$ is computationally expensive, requiring $O(|\mathcal{V}| \cdot d)$ operations per training example. If we use negative sampling with $k = 5$ negative samples, how many words do we compute scores for per training example (positive plus negatives)? Calculate the computational speedup ratio $|\mathcal{V}|/(k+1)$ and explain why negative sampling achieves similar final performance to full softmax despite the approximation.

7. **GloVe Objective.** Given co-occurrence counts $X_{\text{cat,sat}} = 120$, $X_{\text{cat,ran}} = 60$, and GloVe weighting function $f(x) = (x/x_{\max})^\alpha$ with $x_{\max} = 100$ and $\alpha = 0.75$ for $x < x_{\max}$ (and $f(x) = 1$ for $x \geq x_{\max}$), compute the weights $f(X_{\text{cat,sat}})$ and $f(X_{\text{cat,ran}})$. Explain the rationale for this weighting: downweighting very

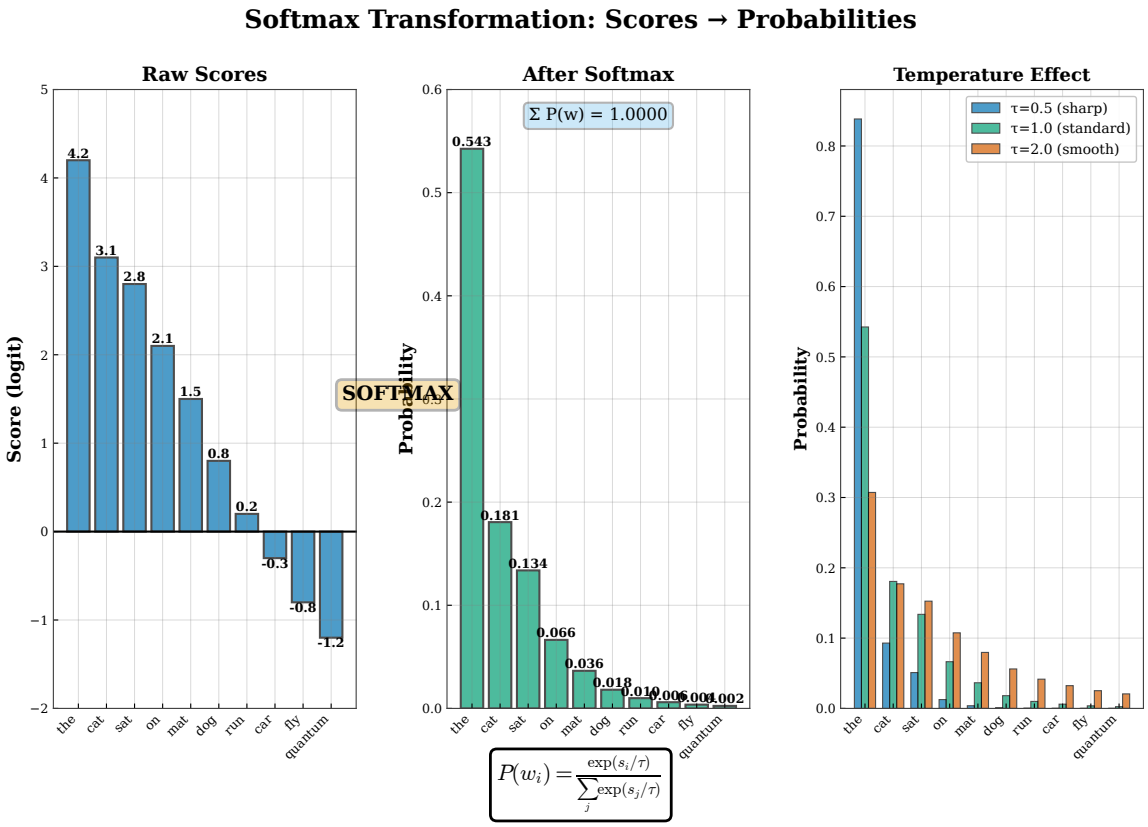

Figure 4.25: Softmax transformation over vocabulary.

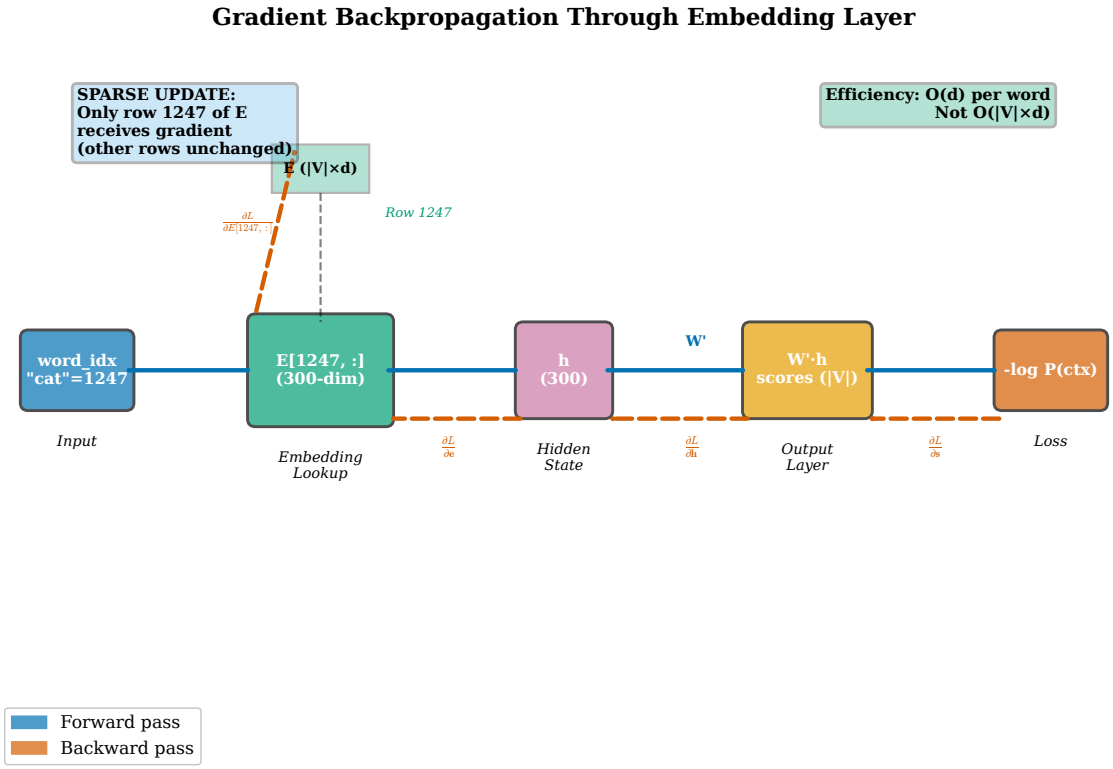

Figure 4.26: Backpropagation through embedding layer.

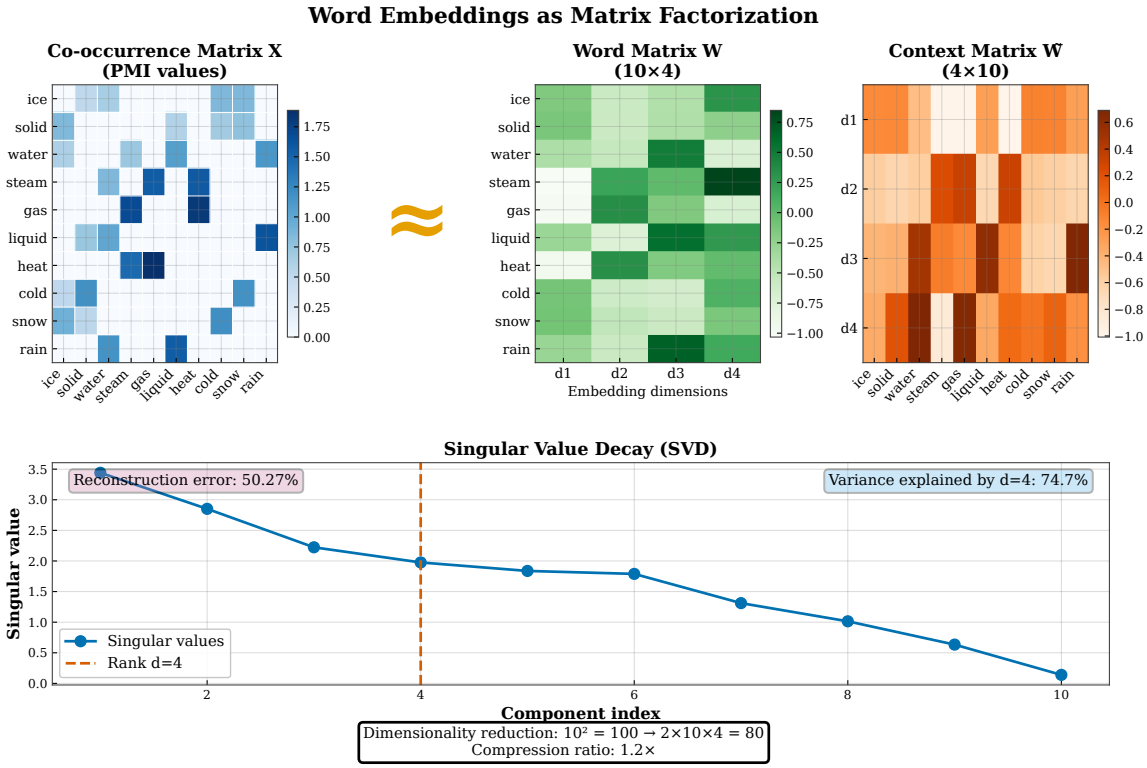

Figure 4.27: PMI matrix factorization connection.

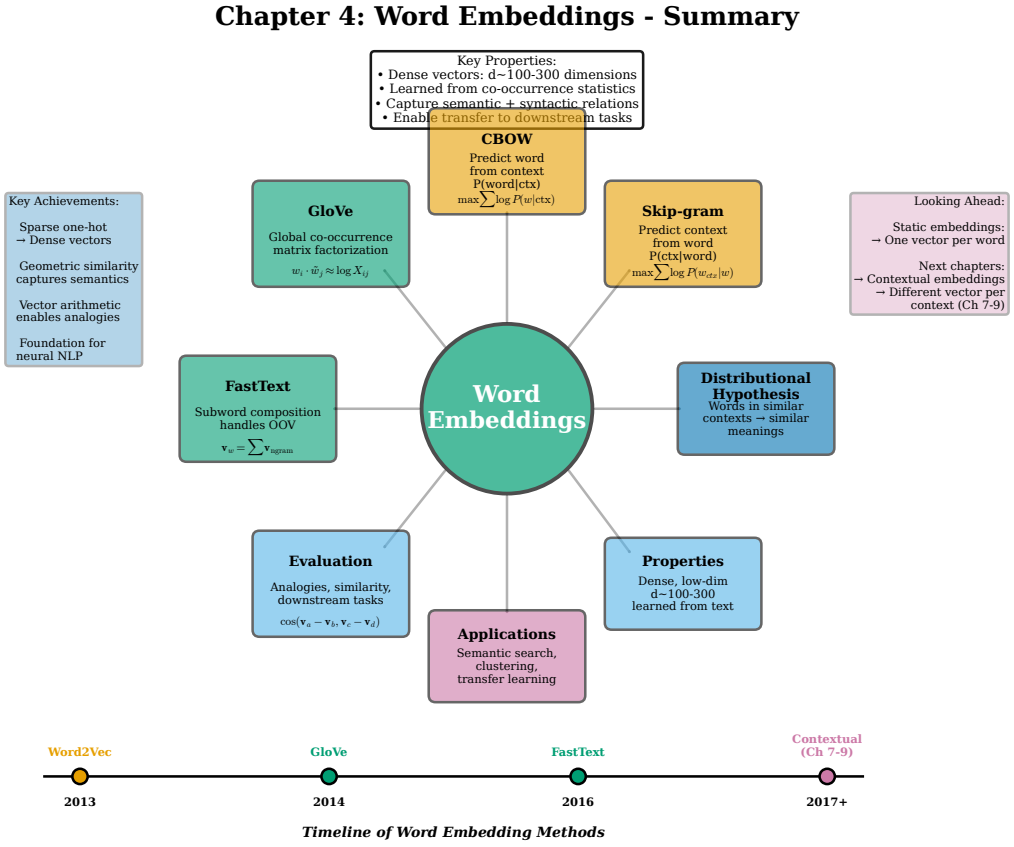

Figure 4.28: Visual summary of word embeddings chapter.

frequent co-occurrences (which are noisy and would dominate the loss) while upweighting moderate co-occurrences (which provide reliable signal).

8. **Vector Arithmetic and Analogies.** Given embeddings satisfying $\mathbf{e}_{\text{king}} - \mathbf{e}_{\text{man}} + \mathbf{e}_{\text{woman}} \approx \mathbf{e}_{\text{queen}}$, provide a geometric interpretation: the vector offset from "man" to "king" encodes a semantic direction ("royalty"), and adding this direction to "woman" arrives at the female royal equivalent "queen". Propose another analogy that should exhibit similar linear structure, such as verb tense (walk:walked::run:ran) or plurality (dog:dogs::cat:cats), and explain why distributional patterns would create these parallel vector offsets.

9. **Cosine Similarity Computation.** Compute cosine similarity between embedding vectors $\mathbf{e}_1 = [1,2,3]$ and $\mathbf{e}_2 = [2,4,6]$ using $\text{sim}(\mathbf{e}_1, \mathbf{e}_2) = \frac{\mathbf{e}_1 \cdot \mathbf{e}_2}{\|\mathbf{e}_1\| \|\mathbf{e}_2\|}$. Then compute cosine similarity between $\mathbf{e}_1 = [1,2,3]$ and $\mathbf{e}_3 = [1,0,0]$. Explain why cosine similarity (measuring angle) is preferred over Euclidean distance $\|\mathbf{e}_1 - \mathbf{e}_2\|$ (measuring magnitude) for semantic similarity: cosine is invariant to vector length, focusing on direction, while word frequency differences cause magnitude variation orthogonal to semantic content.

10. **FastText N-grams.** For the word "teaching" with character n-gram range $[3,4]$ and boundary markers $\langle$ and $\rangle$, list all character n-grams in $G(\text{teaching})$. Explain how this representation enables FastText to generate embeddings for out-of-vocabulary words like "unteaching" by composing shared n-gram embeddings: most n-grams in $G(\text{unteaching})$ (such as "tea", "each", "ach", "chi", "hin", "ing") appear in training words, allowing approximate representation via summation $\mathbf{e}_{\text{unteaching}} = \sum_{g \in G(\text{unteaching})} \mathbf{E}_{\text{ng}}[g,:]$.

11. **Embedding Dimensionality Trade-offs.** Discuss the trade-off between embedding dimension d and model performance. If increasing d from 100 to 1000 improves analogy accuracy from 60% to 75% on the Google Analogy Dataset, why don't we always use $d = 1000$ or even larger dimensions? Consider computational cost (memory scales as $|\mathcal{V}| \times d$, training time scales as $O(d^2)$ for subsequent layers), overfitting risk (higher capacity requires more data), and diminishing returns (performance saturates beyond $d \approx 300$ for most tasks).

12. **Bias in Embeddings.** Suppose word embeddings trained on a news corpus exhibit $\text{similarity}(\text{doctor}, \text{man}) = 0.72$ and $\text{similarity}(\text{doctor}, \text{woman}) = 0.58$, indicating gender bias. Explain how this bias arises from corpus statistics: if doctors are more frequently referred to with male pronouns ("he", "his") than female pronouns ("she", "her") in news articles, the distributional hypothesis causes "doctor" to be closer to "man" in embedding space. Discuss why this is problematic for downstream applications: a hiring algorithm using these embeddings might associate "doctor" more strongly with male candidates, perpetuating discrimination. Propose mitigation strategies such as debiasing (subtracting the gender direction), balanced corpus construction, or awareness audits.

13. **Pre-training vs. Random Initialization.** Compare two approaches for initializing the embedding layer of a language model: (1) pre-trained Word2Vec embeddings learned from Wikipedia, and (2) random initialization with training from scratch. Discuss advantages of each: pre-trained embeddings provide semantic knowledge and faster convergence, especially beneficial for small training corpora (< 100M tokens) or domain-specific tasks; random initialization allows learning embeddings optimized specifically for the prediction task and avoids domain mismatch, beneficial when large training corpora (> 1B tokens) are available. Consider computational budget: pre-training requires additional time but amortizes across multiple downstream tasks.

14. **Weight Tying Benefits.** A language model with vocabulary size $|\mathcal{V}| = 50{,}000$ and embedding dimension $d = 512$ uses separate input embedding matrix $\mathbf{E}^{\text{in}} \in \mathbb{R}^{50000 \times 512}$ and output embedding matrix $\mathbf{E}^{\text{out}} \in \mathbb{R}^{50000 \times 512}$. Calculate the total number of embedding-related parameters: $2 \times 50{,}000 \times 512 = 51.2\text{M}$. If we apply weight tying ($\mathbf{E}^{\text{out}} = (\mathbf{E}^{\text{in}})^\top$), how many parameters are saved? Explain why weight tying can improve generalization: it enforces consistency between input and output representations, providing a regularization effect that constrains the model's capacity and reduces overfitting, empirically reducing perplexity by 5–10% on standard benchmarks.

15. **From Static to Contextual Embeddings.** Static word embeddings assign the same vector to each occurrence of a word type, regardless of context. Explain why this is problematic for polysemous words like "bank" (financial institution vs. river edge), "lead" (metal Pb vs. verb to guide), or "bat" (animal vs. sports equipment). Discuss how the context determines which sense is intended: "savings bank" vs. "river bank". Preview how recurrent neural networks (Chapter **??**) address this limitation by maintaining hidden states $\mathbf{h}_t$ that depend on the entire preceding context $[\mathbf{e}_1, \ldots, \mathbf{e}_t]$, enabling context-dependent representations. Explain why RNN hidden states can resolve polysemy: $\mathbf{h}_t$ encodes not just $\mathbf{e}_{\text{bank}}$ but also the surrounding words, allowing the model to distinguish senses based on context.

Bibliography

Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Jauvin. A neural probabilistic language model. *Journal of Machine Learning Research*, 3:1137–1155, 2003.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

Jeffrey Pennington, Richard Socher, and Christopher D Manning. Glove: Global vectors for word representation. In *EMNLP*, pages 1532–1543, 2014.

Index