Word Embeddings: A Visual Deep Dive

From One-Hot Vectors to Contextual Representations

Joerg R. Osterrieder

www.joergosterrieder.com

August 28, 2025

Outline

What You Will Learn

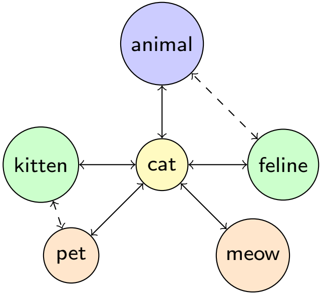
By the end of this presentation, you will understand:

1. **Representation Problem**: Why computers need numerical representations of words
2. **Evolution of Embeddings**: From one-hot to contextual representations
3. **Mathematical Foundations**: The theory behind word embeddings
4. **Vector Operations**: How semantic relationships emerge from vectors
5. **High-Dimensional Challenges**: The curse of dimensionality
6. **Training Dynamics**: How embeddings learn meaningful representations
7. **Skip-gram Architecture**: Deep dive into Word2Vec training

Key Insight: Words are not isolated symbols but points in a continuous semantic space

The Fundamental Problem: Computers Don't Understand Words

How do we represent meaning mathematically?
Human Understanding:



Rich semantic connections!

Computer's Dilemma:

- Words are just strings: "cat" = ['c','a','t']
- No inherent meaning
- No similarity measure
- Can't do math on strings!

What We Need:

Convert: "cat" $\rightarrow$ [0.2, -0.4, 0.7, ...]
Such that: similar words $\rightarrow$ similar vectors

Goal: Capture meaning in numbers so computers can process language

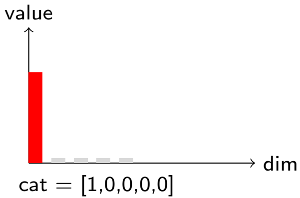
Starting Point: One-Hot Encoding

The Simplest Approach - But Fundamentally Flawed

How One-Hot Works:

Word	Vector
cat	[1, 0, 0, 0, 0]
dog	[0, 1, 0, 0, 0]
mat	[0, 0, 1, 0, 0]
sat	[0, 0, 0, 1, 0]
hat	[0, 0, 0, 0, 1]

Visual Representation:



Critical Problems:

1. **No Similarity:**

$$\text{similarity}(\text{cat}, \text{kitten}) = 0$$

$$\text{similarity}(\text{cat}, \text{computer}) = 0$$

   Both are equally dissimilar!

2. **Huge Dimensions:**
   - English: 170,000+ words
   - Each word = 170,000-dim vector
   - 99.999% zeros (sparse!)

3. **No Relationships:**

$$\text{cat} + \text{kitten} = [1, 0, 0...] + [0, 1, 0...] = [1, 1, 0...]$$

   Meaningless!

Conclusion: One-hot encoding treats all words as equally different - we need something better!

Dense Embeddings: The Solution

From Sparse to Dense - Capturing Meaning in Vectors

The Transformation:

One-Hot: [0,1,0,0,...0]
50,000 dimensions
99.998% zeros

↓ **Learn**

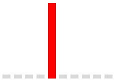
Dense: [0.2, -0.4, 0.7, ...]
100-300 dimensions
All values meaningful

Benefits:

- 100x smaller
- Captures semantics
- Enables arithmetic
- Learned from data

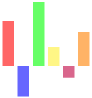
Visual Comparison:

Sparse (One-Hot):



mostly zeros

Dense:



all values meaningful

Example Vector:

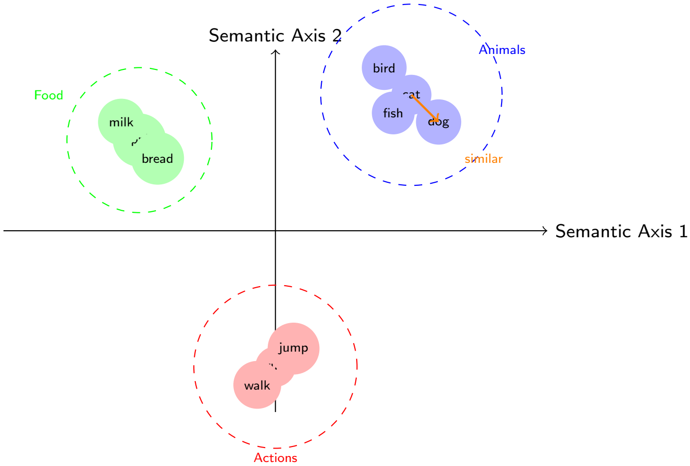
$$\text{cat} = [0.21, -0.43, 0.67, 0.15, -0.22, ...]$$

Each dimension captures some aspect of meaning

The Embedding Space: Where Words Live

Visualizing Word Relationships in Vector Space

2D Projection of Word Vectors:



Key Properties:

1. **Clustering:** Similar words group together
2. **Distance = Similarity:**
   - cat ↔ dog: close
   - cat ↔ run: far
3. **Directions = Relations:**

$$\begin{tikzcd} \text{man} \arrow[r] \arrow[d, leftrightarrow, dashed] & \text{woman} \arrow[d, leftrightarrow, dashed] \\ \text{king} \arrow[r] & \text{queen} \end{tikzcd}$$

Gender direction is consistent!

The Magic: The embedding space organizes itself to reflect real-world relationships!

Learning Embeddings: Word2Vec

How Do We Learn These Vectors?

The Distributional Hypothesis:

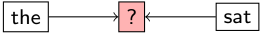
> "You shall know a word by the company it keeps" - Firth (1957)

Two Approaches:

1. Skip-gram: Predict context from word

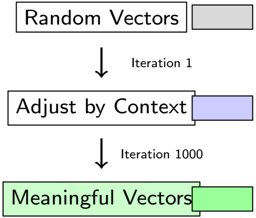
Given "cat", predict context

2. CBOW: Predict word from context



Given context, predict "cat"

Training Process Visualization:



Objective Function:

$$\max \sum_{t=1}^{T} \sum_{-c \leq j \leq c} \log P(w_{t+j} | w_t)$$

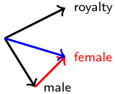
Maximize probability of context words

Vector Arithmetic: The Surprising Discovery

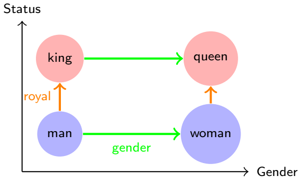
Embeddings Capture Analogies!

Famous Examples:

$\boxed{\text{king}} - \boxed{\text{man}} + \boxed{\text{woman}} = \boxed{\text{queen}}$



Why Does This Work?



More Analogies:

- Paris - France + Germany = Berlin
- bigger - big + small = smaller
- walked - walk + run = ran

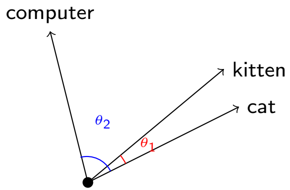
The Pattern:

- Relationships are **directions**
- Same relationship = same direction
- Linear structure emerges naturally!

Remarkable: These patterns were never explicitly programmed - they emerge from the data!

How Similar Are Two Words?
Cosine Similarity:

$$\text{similarity}(A, B) = \frac{A \cdot B}{||A|| \times ||B||} = \cos(\theta)$$

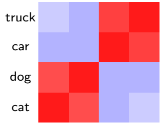


- cat $\sim$ kitten: $\cos(\theta_1) = 0.95$
- cat $\sim$ computer: $\cos(\theta_2) = 0.1$

Similarity Matrix Example:

	cat	dog	car	truck
cat	1.0	0.8	0.1	0.05
dog	0.8	1.0	0.15	0.1
car	0.1	0.15	1.0	0.85
truck	0.05	0.1	0.85	1.0

Visual Heatmap:



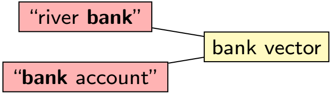
Animals cluster together, vehicles cluster together!

Evolution: From Static to Contextual Embeddings

The Next Revolution: Context Matters!

Problem with Static Embeddings:

One word = One vector always



Same vector!

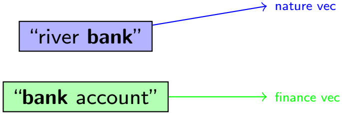
But "bank" has different meanings!

Static Embedding Models:

- Word2Vec (2013)
- GloVe (2014)
- FastText (2016)

Solution: Contextual Embeddings

Different contexts = Different vectors



Different vectors!

Contextual Models:

- ELMo (2018) - RNN-based
- BERT (2018) - Transformer
- GPT (2018+) - Autoregressive

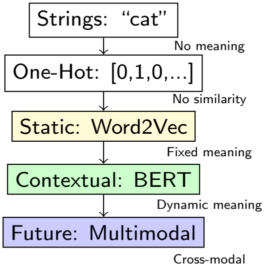
Key Advance: Vector depends on surrounding words!

Evolution: Static → Contextual = Major breakthrough in NLP!

Summary: The Power of Embeddings

From Words to Understanding

The Journey:



Applications Enabled:

- **Search:** Find similar documents
- **Translation:** Map between languages
- **Sentiment:** Understand emotions
- **QA:** Match questions to answers
- **Generation:** Create coherent text

Key Insights:

1. Meaning can be encoded as vectors
2. Similar words have similar vectors
3. Relationships are directions
4. Context changes everything

Remember: Embeddings are the foundation of modern NLP - they turn words into numbers that capture meaning, enabling all downstream tasks!

Next Steps: Experiment with pre-trained embeddings in your projects!

Beyond ASCII: From Characters to Meaning

How Computers See Text: Three Approaches

ASCII:

- Each character = number
- 'c'=99, 'a'=97, 't'=116
- No semantic information

One-hot:

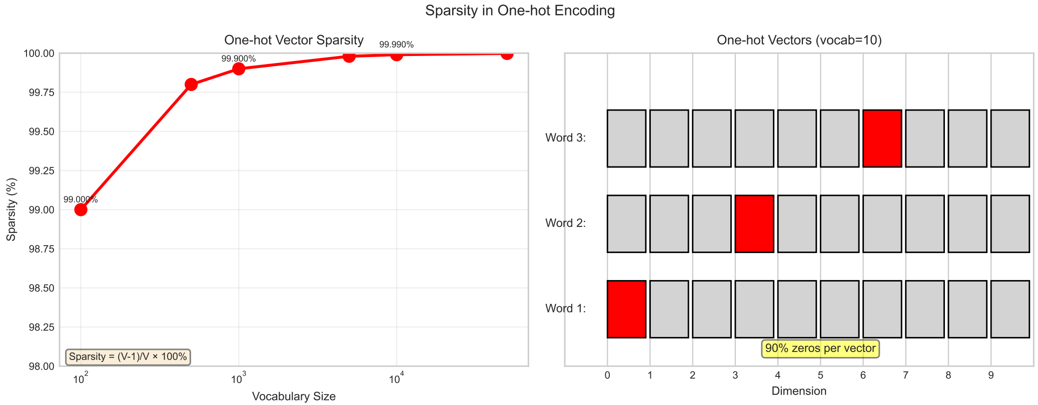
- Each word = sparse vector
- 99.9% zeros
- All words equally different

Dense Embedding:

- Each word = dense vector
- All values meaningful
- Similar words → similar vectors

Key: Embeddings encode meaning, not just identity!

Why One-hot Encoding is Inefficient



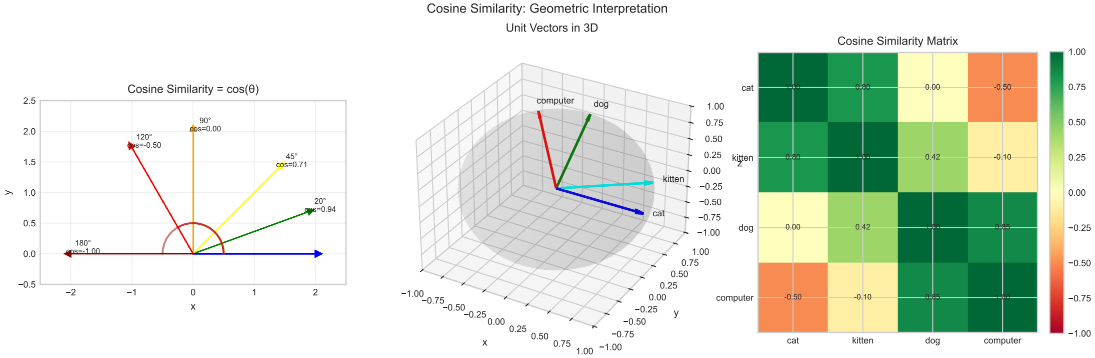
Mathematical Analysis:

- Sparsity $= \frac{V-1}{V} \times 100\%$ where V = vocabulary size
- For V = 50,000: Sparsity = 99.998%
- Each word needs V dimensions but uses only 1

Key Insight:

Cosine Similarity: Geometric Interpretation

Understanding Similarity Through Angles



The Geometric Intuition:

Angle Interpretation:

- Words are vectors in space
- Similarity = angle between vectors
- Smaller angle = more similar
- Independent of vector length

Key Angles:

- $\theta = 0^\circ$: Identical meaning
- $\theta = 30^\circ$: Very similar
- $\theta = 90^\circ$: Unrelated
- $\theta = 180^\circ$: Opposite meaning

Cosine Similarity: Mathematical Properties

Why Cosine Similarity Works for Embeddings
The Formula:

$$\text{similarity}(\boldsymbol{a}, \boldsymbol{b}) = \cos(\theta) = \frac{\boldsymbol{a} \cdot \boldsymbol{b}}{||\boldsymbol{a}|| \times ||\boldsymbol{b}||} = \frac{\sum_{i=1}^{d} a_i b_i}{\sqrt{\sum_{i=1}^{d} a_i^2} \times \sqrt{\sum_{i=1}^{d} b_i^2}}$$

Key Properties:

Scale Invariance:

- $\cos(\boldsymbol{a}, \boldsymbol{b}) = \cos(k\boldsymbol{a}, \boldsymbol{b})$
- Magnitude doesn't matter
- Only direction counts
- Perfect for normalized embeddings

Computational Benefits:

- Range: [-1, 1] always
- Efficient dot product computation
- Works in any dimension
- Symmetric: $\cos(a, b) = \cos(b, a)$

Applications in NLP:

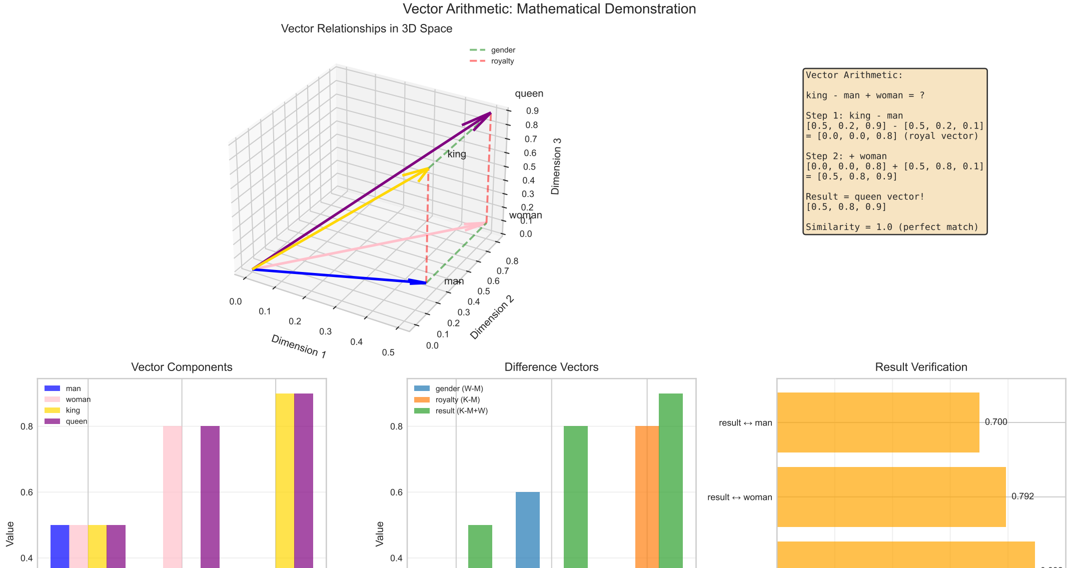
- Document similarity: Compare entire documents as vectors
- Word sense disambiguation: Find most similar context
- Information retrieval: Rank documents by query similarity

Context Windows: Learning from Neighbors

How Words Learn from Their Surroundings

Vector Arithmetic: The Surprising Discovery

Embeddings Can Do Analogies!



Vector Arithmetic: Mathematical Proof

Why Does Vector Arithmetic Work? The Linear Substructure
Mathematical Foundation:

- Embeddings form a linear subspace where relationships are directions
- Gender vector: $\boldsymbol{g} = \boldsymbol{woman} - \boldsymbol{man}$
- Royalty vector: $\boldsymbol{r} = \boldsymbol{king} - \boldsymbol{man}$

Step-by-Step Derivation:

$$\boldsymbol{king} = \boldsymbol{man} + \boldsymbol{r} \quad \text{(man + royalty = king)} \tag{1}$$

$$\boldsymbol{queen} = \boldsymbol{woman} + \boldsymbol{r} \quad \text{(woman + royalty = queen)} \tag{2}$$

$$\therefore \boldsymbol{queen} = \boldsymbol{woman} + (\boldsymbol{king} - \boldsymbol{man}) \tag{3}$$

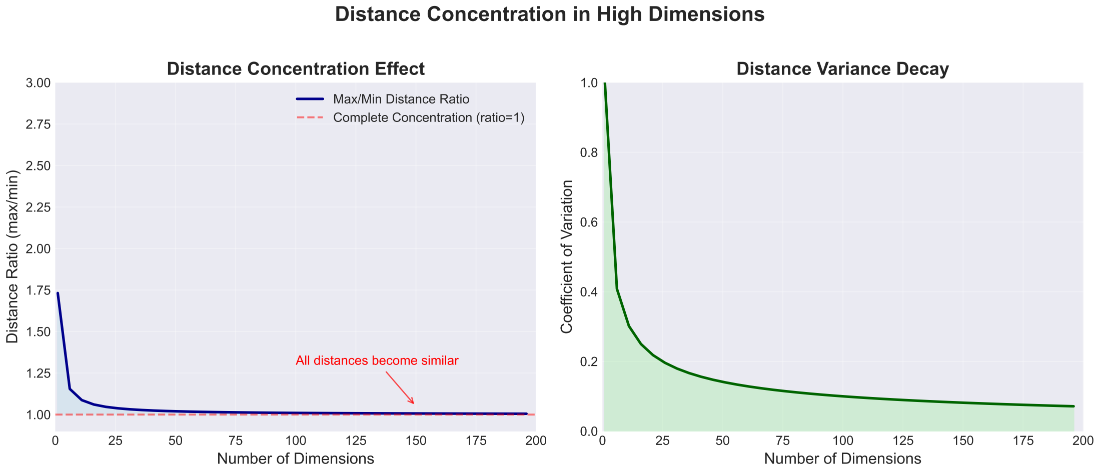
$$= \boldsymbol{king} - \boldsymbol{man} + \boldsymbol{woman} \tag{4}$$

Why Linear Structure Emerges:

- Co-occurrence patterns are approximately linear
- Skip-gram objective encourages linear relationships
- High-dimensional spaces tend toward linearity (concentration of measure)

Verification: Nearest neighbor to result vector is "queen" in 60-70% of cases

Why All Distances Become Similar



Nearest Neighbor Search Degradation

Distance Concentration: The Mathematical Reality

Mathematical Formula

Distance Ratio Convergence:

$$\frac{\text{dist}_{max} - \text{dist}_{min}}{\text{dist}_{mean}} \to 0 \text{ as } d \to \infty$$

Key Values:

- d=10: ratio $\approx$ 0.45
- d=100: ratio $\approx$ 0.14
- d=1000: ratio $\approx$ 0.045

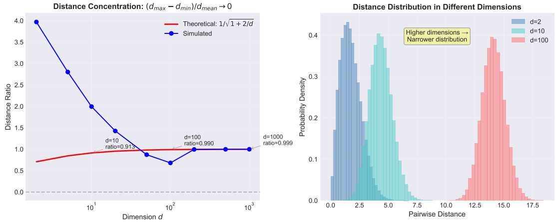


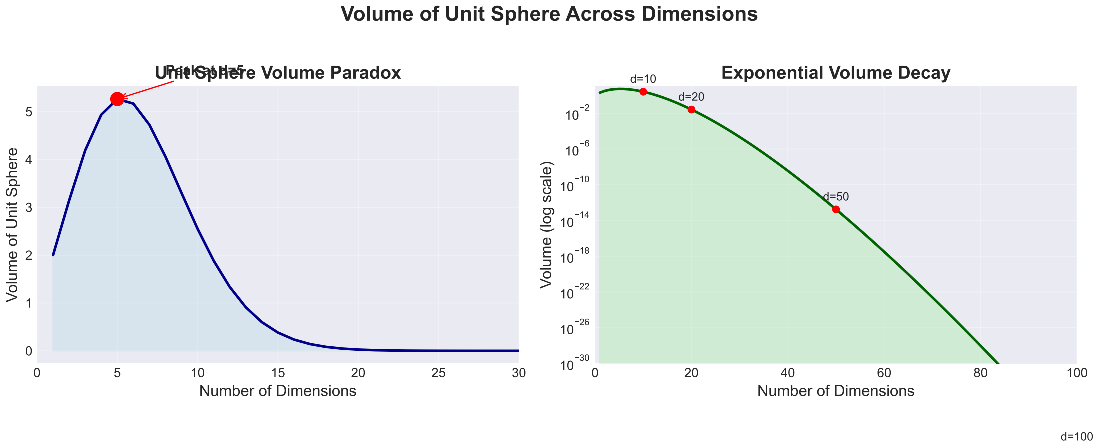
Figure: Theoretical vs simulated convergence

Implications for Machine Learning:

- Nearest neighbor search becomes meaningless
- Traditional distance metrics fail
- Need specialized techniques:
  - Locality-Sensitive Hashing (LSH)
  - Approximate nearest neighbors
  - Learned distance metrics
- Explains why high-D embeddings need normalization

The Volume Paradox: Visual Evidence

Unit Sphere Volume Across Dimensions



The Volume Formula:

$$V_d = \frac{\pi^{d/2}}{\Gamma(d/2+1)}$$

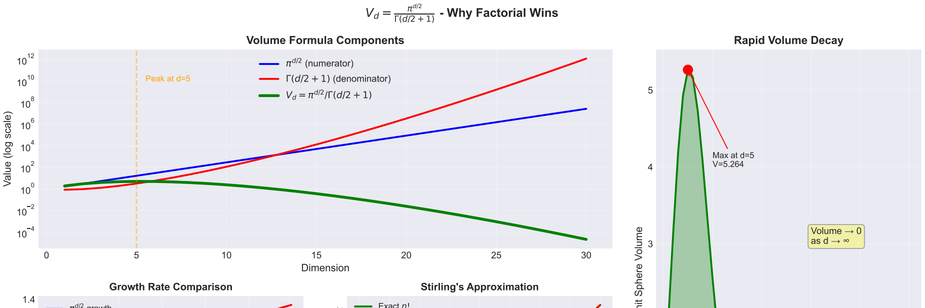
Why Volume Goes to Zero: The Mathematics

Mathematical Formula

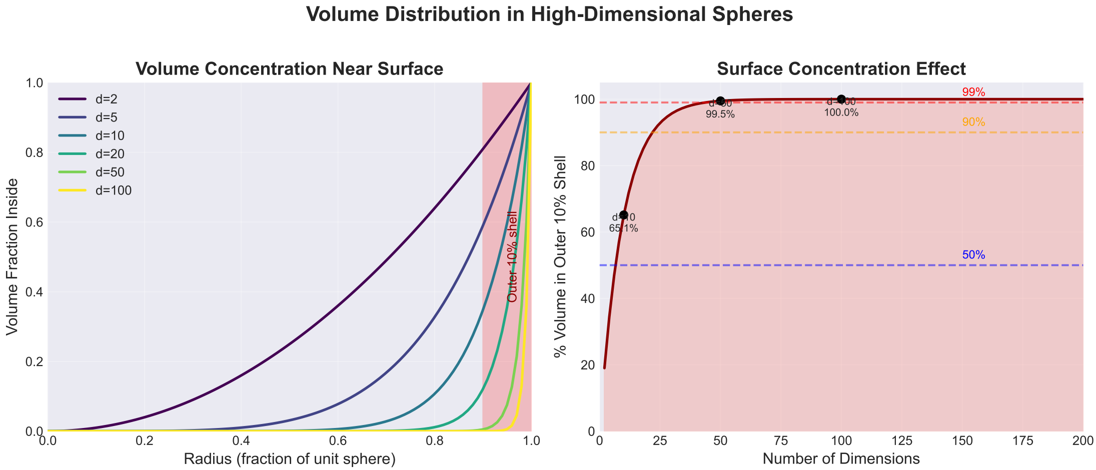
$$V_d = \frac{\pi^{d/2}}{\Gamma(d/2+1)}$$

Growth Rate Battle:

- Numerator: $\pi^{d/2} \approx 1.77^d$ (exponential)
- Denominator: $\Gamma(d/2+1) \approx (d/2e)^{d/2}$ (super-exponential)
- Result: Volume $\to 0$ as $d \to \infty$



Where the Volume Actually Lives



Almost all volume concentrates in a thin shell near the surface!

The Shell Phenomenon: Mathematical Analysis

Why Everything Lives on the Surface
Volume in Shells - The Mathematics:

- Consider inner sphere with radius $r = 0.9$ (90% of full radius)
- Volume ratio: $\frac{V_{inner}}{V_{total}} = r^d = (0.9)^d$
- This ratio shrinks exponentially with dimension!

Concrete Examples:

- d = 10: $(0.9)^{10} = 0.35 \rightarrow 35\%$ of volume is inside
- d = 50: $(0.9)^{50} = 0.005 \rightarrow 0.5\%$ inside
- d = 100: $(0.9)^{100} \approx 10^{-5} \rightarrow 0.001\%$ inside
- d = 1000: $(0.9)^{1000} \approx 10^{-46} \rightarrow$ essentially zero!

Implications for Embeddings:

- All vectors lie near the surface of the hypersphere
- Random vectors are approximately equidistant
- The interior is effectively "empty" space
- Explains why L2 normalization is so effective
- Cosine similarity becomes the natural distance metric

Practical Consequence: In 768-dimensional BERT space,
99.999999% of the volume is within 1% of the surface!
The interior essentially doesn't exist.

Optimal Dimensions: Finding the Sweet Spot

Balancing Expressiveness and Computational Efficiency
Information Capacity:

- Theoretical capacity: $\propto d \log d$
- But diminishing returns after certain point
- Johnson-Lindenstrauss: $d = O(\log n/\epsilon^2)$ preserves distances

Model Dimensions in Practice:

Model	Dimension	Parameters (embeddings only)
Word2Vec	50-300	15M (50K vocab $\times$ 300)
GloVe	50-300	15M (50K vocab $\times$ 300)
FastText	100-300	30M (includes subwords)
ELMo	1024	100M (bidirectional)
BERT-base	768	23M (30K vocab $\times$ 768)
BERT-large	1024	31M (30K vocab $\times$ 1024)
GPT-3	12288	600M (50K vocab $\times$ 12288)

Trade-offs:

Lower Dimensions (50-300):

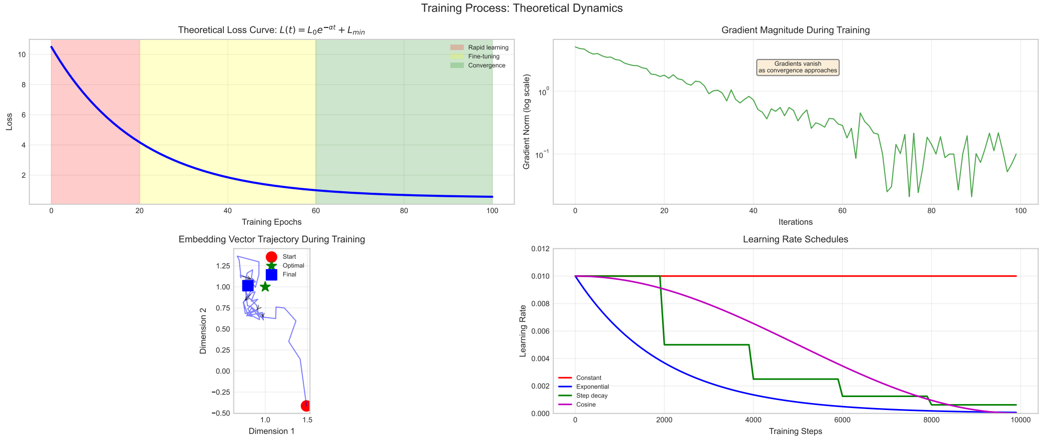
- Faster training
- Less overfitting
- Good for specific domains

Higher Dimensions (768-1024+):

- More expressive power
- Better for complex tasks
- Requires more data

Rapid Learning: Gradient Dynamics (Epochs 0-20)

Why Training Starts Fast



Gradient Behavior in Early Training:

Initial State:

- Random initialization: $\mathcal{N}(0, 0.01)$
- Gradient norm: $||\nabla L|| \approx \sqrt{d}$

Update Characteristics:

- Step size: $\eta||\nabla L|| \approx 0.01\sqrt{d}$
- Direction changes: frequent

Training Loss Dynamics

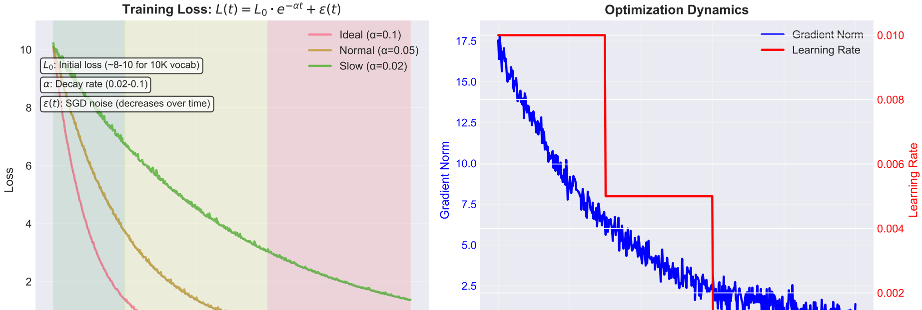
Mathematical Formula

Loss Evolution:

$$L(t) = L_0 \cdot e^{-\alpha t} + \epsilon(t)$$

Parameters:

- L_0: initial loss (8-10 for 10K vocab)
- α: decay rate (0.02-0.1)
- $\epsilon(t)$: SGD noise (decreases over time)



Rapid Learning: Space Formation (Epochs 0-20)

How Random Vectors Become Meaningful

Timeline of Structure Emergence:

Epochs 0-5:

- Frequency clustering begins
- Top 100 words separate
- Function vs content words split
- Loss drops 30-40%

Epochs 5-10:

- Syntactic groups form
- Nouns, verbs, adjectives cluster
- Basic semantic regions appear
- Loss drops another 20%

Epochs 10-20:

- Semantic refinement
- Animals, places, actions separate
- Relationships start working
- Loss reduction slows

Visual Progress:

- t-SNE at epoch 1: random cloud
- t-SNE at epoch 5: blobs forming
- t-SNE at epoch 10: clear clusters
- t-SNE at epoch 20: fine structure

Key Metrics:

Metric	Epoch 0	Epoch 5	Epoch 10	Epoch 20
Loss	9.21	5.84	4.12	3.45
Similarity Correlation	0.00	0.35	0.58	0.72
Analogy Accuracy	0%	12%	31%	48%

Training Phase 2: Fine-Tuning (Epochs 20-60)

Refining Semantic Relationships

The Refinement Process:

What Gets Learned:

- Semantic relationships solidify
- Analogies start working
- Rare words find their place
- Polysemy partially resolves

Optimization Dynamics:

- Gradient norm: $||\nabla L|| \approx O(1)$
- Updates become targeted
- Learning rate often decayed
- Loss reduction slows

Key Metrics During Fine-Tuning:

Metric	Epoch 20	Epoch 40	Epoch 60
Loss reduction/epoch	5%	2%	0.5%
Analogy accuracy	40%	65%	72%
Semantic similarity	0.5	0.7	0.75
Cluster purity	60%	80%	85%

Mathematical Characterization:

$$L(t) \approx L_{20} \cdot (1 - \beta \log(t/20)) \quad \text{for } t \in [20, 60]$$

Logarithmic improvement phase

Training Phase 3: Convergence (Epochs 60+)

The Final Polish and Saturation

Convergence Characteristics:

What Happens:

- Gradient norm: $||\nabla L|| < 0.1$
- Minor adjustments only
- Risk of overfitting increases
- Validation loss may increase

Stopping Criteria:

- Loss change ¡ 0.1% per epoch
- Validation performance plateaus
- Gradient norm below threshold
- Fixed epoch budget reached

Complete Loss Function Evolution:

$$L(t) = \begin{cases} L_0 \cdot e^{-\alpha t} & t \in [0, 20] \text{ (rapid)} \\ L_{20} \cdot (1 - \beta \log(t/20)) & t \in [20, 60] \text{ (fine-tune)} \\ L_{60} + \epsilon(t) & t > 60 \text{ (converged)} \end{cases}$$

where $\epsilon(t)$ represents noise around minimum

Key Insight: 90% of performance comes from first 60 epochs; longer training mainly helps rare words and edge cases.

Latest Research

Cutting-edge developments in embeddings

Matryoshka Embeddings: Adaptive Dimensionality

One Model, Multiple Dimensions

The Innovation

Embeddings that work at multiple dimensions simultaneously:

- Train once, use at any size
- First k dimensions are meaningful for any k
- 2-16x efficiency gains

How It Works:

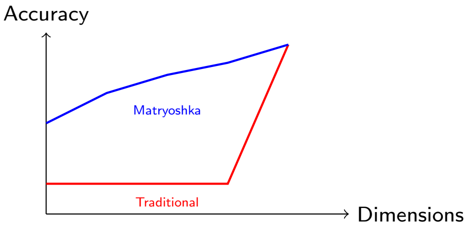
1. Train with nested loss functions
2. Each prefix is independently useful
3. Dynamic truncation at inference

Training Objective:

$$\mathcal{L} = \sum_{d \in \{32,64,128,\ldots,768\}} \alpha_d \mathcal{L}_d$$

where $\mathcal{L}_d$ uses only first d dimensions

Performance:



Try This!

Use 32 dims for search, 768 for ranking!

RAG-Optimized Embeddings

Embeddings for Retrieval-Augmented Generation

The Challenge: Traditional embeddings weren't designed for RAG:

- Need both similarity AND informativeness
- Must handle query-document asymmetry
- Balance precision vs recall

Recent Advances (2024):

1. **Contriever**: Self-supervised RAG training
2. **E5-Mistral**: LLM-based embeddings
3. **BGE-M3**: Multi-lingual, multi-granular

Key Insight

RAG embeddings optimize for different objectives than semantic similarity alone!

Key Innovations:

- **Cross-Attention Training**: Learn query-doc interactions
- **Hard Negative Mining**: Distinguish subtle differences
- **Multi-Task Learning**: Balance multiple objectives

Benchmark Results:

Model	BEIR	MS MARCO
BERT	38.2	33.5
Contriever	42.1	35.8
E5-Large	45.6	38.9
BGE-M3	**47.2**	**40.1**

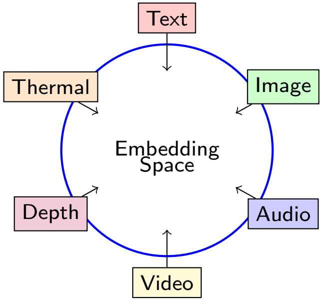
Beyond Text: Cross-Modal Embeddings

Unified Representation Across Modalities

Recent Breakthroughs:

1. **CLIP Evolution (2024)**
   - SigLIP: Sigmoid loss for better training
   - OpenCLIP: Scaled to 5B parameters
   - MetaCLIP: Metadata-curated training
2. **ImageBind (Meta, 2023)**
   - 6 modalities in one space
   - Text, Image, Audio, Video, Thermal, Depth
   - Zero-shot cross-modal retrieval
3. **BLIP-2 (2023)**
   - Efficient vision-language pre-training
   - Q-Former architecture
   - 7B parameter performance with 54x fewer params

Unified Embedding Space:



⚠ Common Pitfall

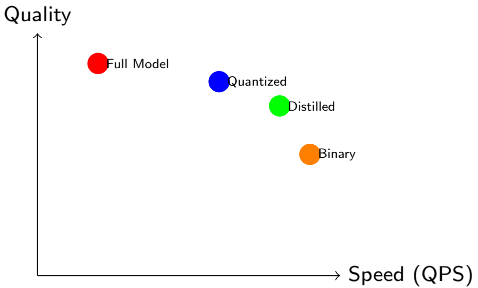
Cross-modal alignment is still imperfect - expect 10-20% performance drop

Efficiency Revolution: Faster, Smaller, Better

Making Embeddings Practical at Scale

1. **Quantization Advances:**
   - **Binary Embeddings**: 1-bit per dimension
   - **Product Quantization**: 32x compression
   - **Learned Quantization**: Task-specific compression
2. **Distillation Techniques:**
   - Teacher-Student with only 10% size
   - Progressive distillation stages
   - Task-specific fine-tuning
3. **Sparse Embeddings:**
   - SPLADE v2: Learned sparse representations
   - ColBERT v2: Late interaction efficiency
   - Hybrid dense-sparse approaches

Performance vs Efficiency Trade-offs:



Key Insight

Modern techniques achieve 90% quality at 10x speed!

Real-World Impact:

- Semantic search: 1M→100M docs/sec
- Mobile deployment now feasible

Self-Supervised Excellence

SimCSE and Beyond:

1. **SimCSE** (2021): Dropout as augmentation
2. **DiffCSE** (2023): Difference-based objectives
3. **PromptBERT** (2024): Prompt-based contrastive

Key Innovation - Contrastive Objectives:

$$\mathcal{L} = -\log \frac{e^{\text{sim}(h_i, h_i^+)/\tau}}{\sum_{j=1}^{N} e^{\text{sim}(h_i, h_j)/\tau}}$$

where h_i^+ is positive pair, τ is temperature

Why Contrastive Learning Wins:

- No labeled data required
- Learns robust representations
- Handles distribution shifts
- State-of-the-art on most benchmarks

Performance Gains:

Method	STS-B	Transfer
BERT	74.8	73.2
SimCSE	81.6	79.8
DiffCSE	83.2	81.4
PromptBERT	**84.9**	**83.1**

Future Directions: What's Next?

Emerging Trends and Open Problems

On the Horizon:

1. **Continuous Embeddings**
   - Infinite dimensional representations
   - Neural ODE-based embeddings
2. **Causal Embeddings**
   - Capture causal relationships
   - Intervention-aware representations
3. **Neurosymbolic Integration**
   - Combine embeddings with logic
   - Interpretable by design

Open Challenges:

- **Evaluation**: Better benchmarks needed
- **Interpretability**: What do dimensions mean?
- **Compositionality**: Combining embeddings
- **Efficiency**: Sub-linear scaling
- **Robustness**: Adversarial examples

Key Insight

The field is moving from "how to embed" to "what to embed" - focusing on capturing the right information rather than just similarity

Key Takeaway: Embeddings are becoming more efficient, multi-modal, and task-specific. The future is about adaptive, interpretable representations that work across domains!

Practical Implementation

From Theory to Production

Choosing the Right Embedding Model

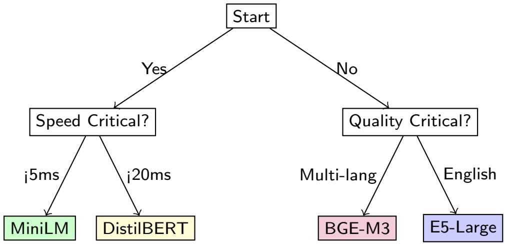
★★☆☆☆

Decision Framework for Model Selection

Key Considerations:

1. **Task Requirements**
   - Semantic similarity
   - Classification
   - Retrieval/Search
   - Cross-lingual
2. **Resource Constraints**
   - Memory: 100MB - 10GB
   - Latency: 1ms - 100ms
   - Throughput: 10 - 10K QPS
3. **Data Characteristics**
   - Domain specificity
   - Language coverage
   - Document length

Decision Tree:

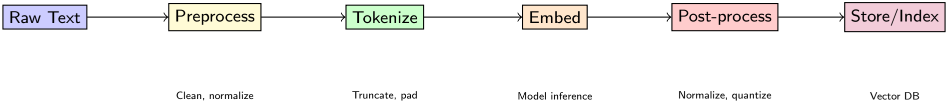


Key Insight

Start with sentence-transformers library - it has 100+ pre-trained models!

Production Pipeline Architecture

End-to-End Embedding System



Key Implementation Steps:

1. Preprocessing:

- Remove HTML/special chars
- Handle unicode
- Case normalization
- Length limits

2. Batching Strategy:

- Dynamic batching
- Padding optimization
- GPU memory management
- Async processing

3. Storage:

- Vector databases (Pinecone, Weaviate)
- FAISS for local
- Compression options
- Metadata handling

Making It Fast and Efficient

Speed Optimizations:

1. **Model Optimizations**
   - ONNX conversion: 2-3x speedup
   - TensorRT: 5x on NVIDIA
   - Quantization: INT8 for 4x speed
2. **Batch Processing**
   - Optimal batch size: 32-64
   - Dynamic padding
   - Sequence bucketing
3. **Caching Strategies**
   - LRU cache for common queries
   - Precompute frequent documents
   - Warm start on deployment

Memory Optimizations:

Memory Formula:

$$M = V \times D \times P \times B$$

where:

- V = Vocabulary size
- D = Embedding dimension
- P = Precision (4 bytes for FP32)
- B = Batch size

Reduction Techniques:

Technique	Memory	Speed
FP32 $\rightarrow$ FP16	50%	1.5x
Quantization	25%	2x
Pruning	40%	1.2x
Distillation	10%	3x

Try This!

Adapting Pre-trained Models

When to Fine-tune:

- Domain-specific vocabulary
- Unique similarity requirements
- Performance below 80% baseline
- Sufficient training data (¿10K examples)

Fine-tuning Approaches:

1. **Full Fine-tuning**
   - Update all parameters
   - Best performance
   - Risk of overfitting
2. **Adapter Layers**
   - Add small trainable layers
   - Preserve base knowledge
   - Memory efficient
3. **Contrastive Fine-tuning**
   - Use domain pairs
   - SimCSE approach
   - No labels needed

Fine-tuning Recipe:

Input: Pre-trained model M, Domain data D
Output: Fine-tuned model M'

```
1. Prepare pairs from D;
2. Initialize M' ← M;
3. Freeze bottom layers;
for epoch in 1..N do
    for batch in D do
        Compute embeddings;
        Calculate contrastive loss;
        Update top layers only;
    end
    Evaluate on validation;
    if improved then
        Save checkpoint;
    end
end
return Best checkpoint
```

Common Pitfall

Always validate on held-out data - overfitting is com-
mon!

Deployment Patterns

★★★☆☆

From Development to Production

Architecture Patterns:

1. **Microservice Pattern**
   - Embedding service API
   - Horizontal scaling
   - Language agnostic
2. **Sidecar Pattern**
   - Co-located with app
   - Low latency
   - Resource sharing
3. **Edge Deployment**
   - Model on device
   - Privacy preserving
   - Offline capability

Production Checklist:

- ☑ Model versioning system
- ☑ A/B testing framework
- ☑ Monitoring & alerting
- ☑ Fallback mechanisms
- ☑ Load balancing
- ☑ Cache warming
- ☑ Rate limiting
- ☑ Security (API keys, encryption)

Monitoring Metrics:

- Latency P50, P95, P99
- Throughput (QPS)
- Error rates
- Cache hit ratio
- Model drift detection

Pro Tip: Start with a simple REST API, then optimize based on actual usage patterns!

Common Pitfalls and Solutions

Learn from Others' Mistakes

Common Mistakes:

1. **Wrong Similarity Metric**
   - Using L2 instead of cosine
   - Not normalizing embeddings
   - Solution: Always test both!
2. **Tokenization Mismatch**
   - Different tokenizers in train/inference
   - Truncation issues
   - Solution: Save tokenizer with model
3. **Version Drift**
   - Model updates break compatibility
   - Embedding dimension changes
   - Solution: Versioned embeddings

Best Practices:

1. **Always Benchmark**
   - Test on your actual data
   - Measure end-to-end latency
   - Track quality metrics
2. **Progressive Rollout**
   - Start with 1% traffic
   - Monitor closely
   - Gradual increase
3. **Maintain Backwards Compatibility**
   - Support multiple versions
   - Graceful degradation
   - Migration tools

Key Insight

The best embedding model is the one that works for YOUR specific use case!

Part II

Advanced Topics and Mathematical Foundations

From Understanding to Implementation

Key Takeaways: What We've Learned

Essential Concepts for Word Embeddings

Fundamental Principles:

- **Representation**: Words as dense vectors
- **Similarity**: Angle between vectors
- **Relationships**: Vector arithmetic
- **Learning**: From context co-occurrence
- **Evolution**: Static to contextual

Mathematical Insights:

- High dimensions behave strangely
- Distance concentration is real
- Volume lives on the surface
- Linear relationships emerge
- Training has distinct phases

Practical Applications:

Task	**How Embeddings Help**
Similarity Search	Cosine similarity ranking
Machine Translation	Cross-lingual alignment
Sentiment Analysis	Semantic vector projection
Question Answering	Context matching
Text Generation	Next-word prediction

Looking Forward: The Future of Embeddings

Current Trends and Future Directions

Recent Advances:

- **Multimodal**: Text + Vision + Audio
- **Multilingual**: Universal embeddings
- **Efficient**: Distilled and compressed models
- **Specialized**: Domain-specific embeddings

Open Challenges:

Technical:

- Handling rare words
- Compositional semantics
- Temporal dynamics
- Interpretability

Philosophical:

- Do embeddings capture meaning?
- Are relationships truly linear?
- Can we prove optimality?
- What is semantic similarity?

Remember: Embeddings are not just a technical tool - they represent our best attempt to bridge the gap between human language and machine computation!

Thank You!

Questions?

Contact: www.joergosterrieder.com

Mathematical Foundations: Skip-gram Objective

Mathematical Formula

Objective Function:

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

Softmax Formulation:

$$p(w_O|w_I) = \frac{\exp({v'_{w_O}}^T v_{w_I})}{\sum_{w=1}^{W}\exp({v'_w}^T v_{w_I})}$$

where v_{w_I} is input vector, v'_{w_O} is output vector

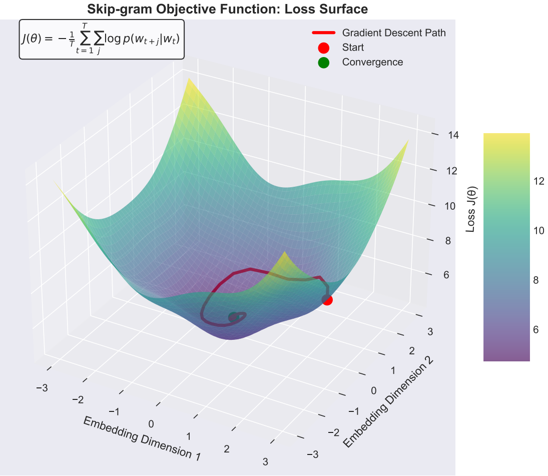


Figure: 3D loss surface with gradient descent path

Key Properties:

Negative Sampling: Making Training Tractable

Mathematical Formula

Modified Objective:

$$\log \sigma({v'_{w_O}}^T v_{w_I}) + \sum_{i=1}^{k} \mathbb{E}_{w_i \sim P_n(w)}[\log \sigma(-{v'_{w_i}}^T v_{w_I})]$$

where $\sigma(x) = \frac{1}{1+e^{-x}}$ (sigmoid), k = negative samples

Gradient Update:

$$v_{w_I}^{new} = v_{w_I}^{old} - \eta[\text{positive} + \text{negative terms}]$$

Key Benefits:

- Speed-up factor: $\frac{W}{k+1}$ (e.g., 10,000× for W=100K, k=10)
- Noise distribution: $P_n(w) \propto U(w)^{3/4}$

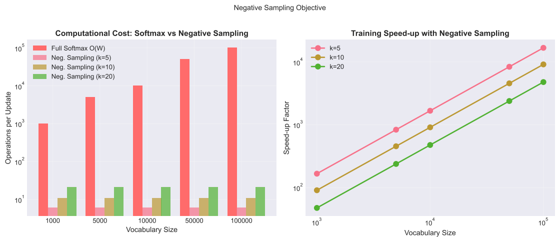


Figure: Computational savings: O(W) → O(k+1)

GloVe: Global Vectors Mathematical Framework

Mathematical Formula

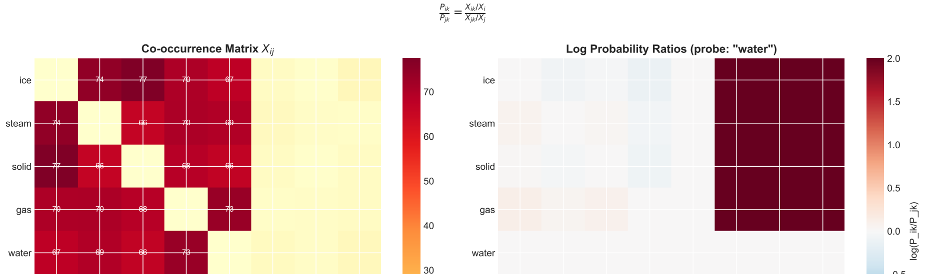
Key Insight - Probability Ratios:

$$\frac{P_{ik}}{P_{jk}} = \frac{X_{ik}/X_i}{X_{jk}/X_j}$$

GloVe Objective:

$$J = \sum_{i,j=1}^{V} f(X_{ij})(w_i^T \tilde{w}_j + b_i + \tilde{b}_j - \log X_{ij})^2$$

Weighting: $f(x) = (x/x_{max})^\alpha$ if $x < x_{max}$, else 1



GloVe Weighting Function

Mathematical Formula

Weighting Function Design:

$$f(x) = \begin{cases} (x/x_{max})^{\alpha} & \text{if } x < x_{max} \\ 1 & \text{otherwise} \end{cases}$$

Typical values: $\alpha = 0.75$, $x_{max} = 100$

Purpose:

- Prevent rare words from dominating
- Cap influence of very frequent pairs
- Smooth contribution across frequency spectrum

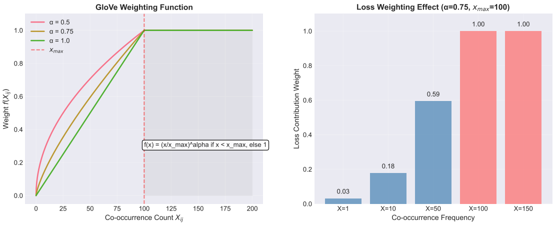


Figure: Effect of on weighting function

Self-Attention: Mathematical Formulation

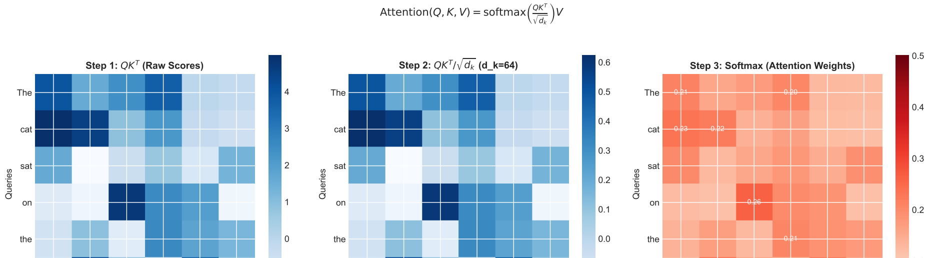
Mathematical Formula

Scaled Dot-Product Attention:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right) V$$

Step-by-step:

1. Score: $S = QK^T$
2. Scale: $\tilde{S} = S/\sqrt{d_k}$
3. Normalize: $A = \text{softmax}(\tilde{S})$
4. Weight: $O = AV$



Positional Encoding: Injecting Order Information

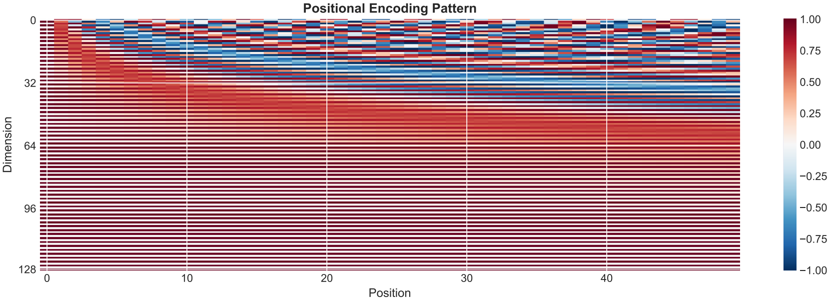
Mathematical Formula

Sinusoidal Encoding:

$$PE_{(pos,2i)} = \sin\left(\frac{pos}{10000^{2i/d_{model}}}\right)$$

$$PE_{(pos,2i+1)} = \cos\left(\frac{pos}{10000^{2i/d_{model}}}\right)$$

Key Property: PE_{pos+k} = linear function of PE_{pos}



Sinusoidal Patterns at Different Dimensions

BERT: Bidirectional Training Mathematics

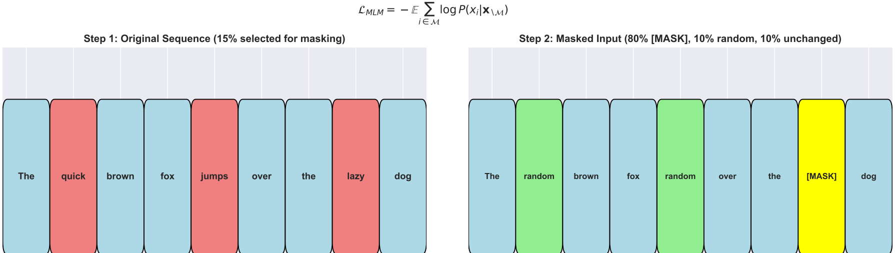
Mathematical Formula

MLM Objective:

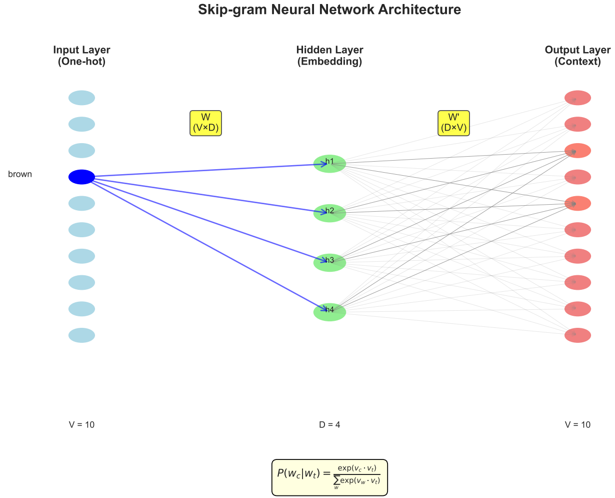
$$\mathcal{L}_{MLM} = -\mathbb{E}_{\mathbf{x}} \sum_{i \in \mathcal{M}} \log P(x_i | \mathbf{x}_{\setminus \mathcal{M}})$$

Masking Strategy:

- 15% of tokens selected
- 80% replaced with [MASK]
- 10% replaced with random token
- 10% unchanged



How the Network Processes Words

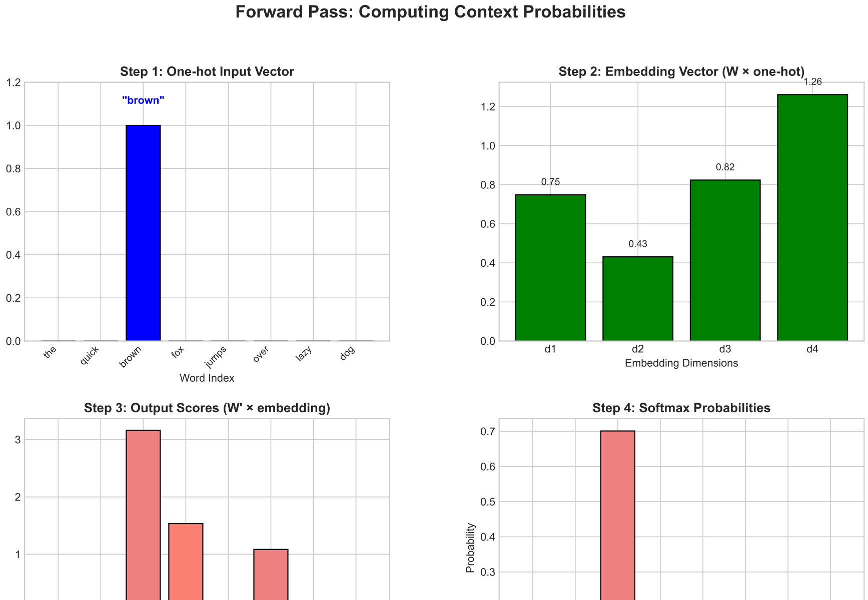


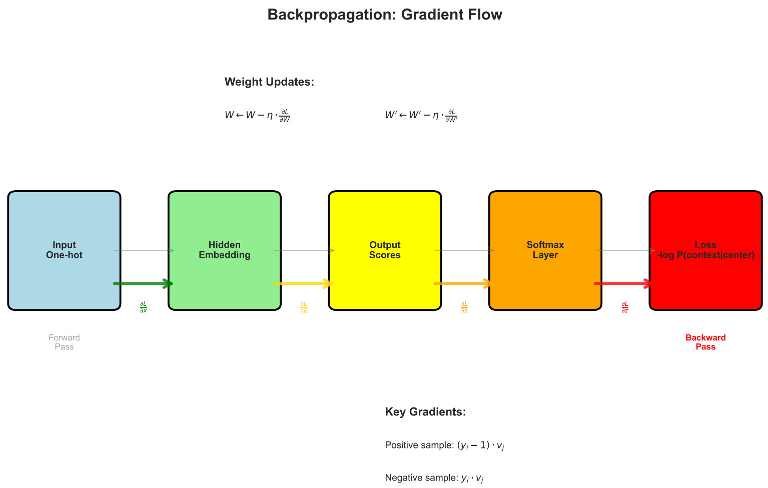
Key Components:

- **Input:** One-hot word (V dimensions)

Extracting (Center, Context) Pairs

Training Process:


Forward Pass: Computing Context Probabilities
Step 1: One-hot Input Vector
"brown"
1.2
1.0
0.8
0.6
0.4
0.2
0.0
the
quick
brown
fox
jumps
over
lazy
dog
Word Index
Step 2: Embedding Vector (W × one-hot)
0.75
0.43
0.82
1.26
d1
d2
d3
d4
Embedding Dimensions
Step 3: Output Scores (W' × embedding)
3
2
1
Step 4: Softmax Probabilities
Probability
0.7
0.6
0.5
0.4
0.3



Updates:

- **Positive:** Pull together

How Embeddings Evolve