Word Embeddings: A Visual Deep Dive

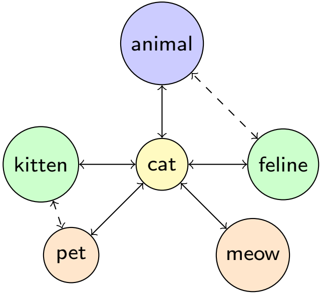
From One-Hot Vectors to Contextual Representations

Joerg R. Osterrieder

www.joergosterrieder.com

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" → [0.2, -0.4, 0.7, ...]
Such that: similar words → similar vectors

Goal: Capture meaning in numbers so computers can process language

Part I

Foundation Concepts

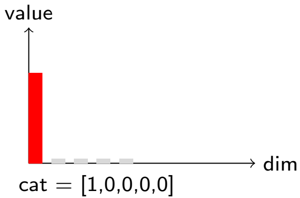
From Basic Representations to Dense Embeddings

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:

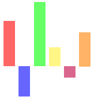
- 100x smaller
- Captures semantics
- Enables arithmetic
- Learned from data

Visual Comparison:

Sparse (One-Hot):

mostly zeros

Dense:



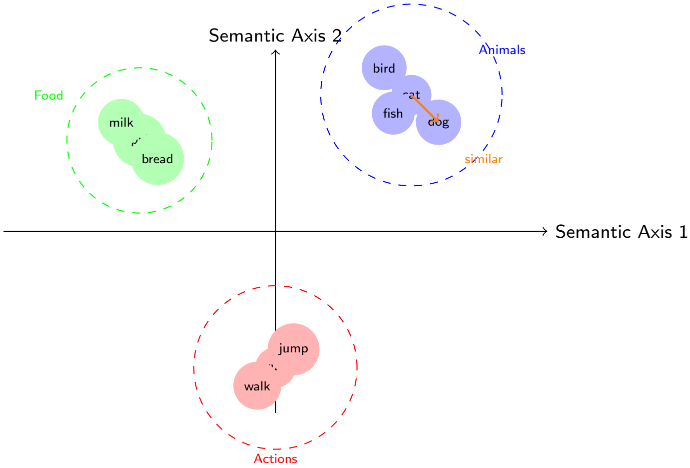
all values meaningful

Example Vector:

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

Each dimension captures some aspect of meaning

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:**

$$\begin{array}{ccc} \text{man} & \longrightarrow & \text{woman} \\ \updownarrow & & \updownarrow \\ \text{king} & \longrightarrow & \text{queen} \end{array}$$

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:

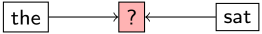
> "You shall know a word by the company it keeps" - Firth (1957)

Two Approaches:

1. **Skip-gram:** Predict context from word

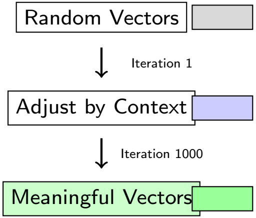
Given "cat", predict context

2. **CBOW:** Predict word from context



Given context, predict "cat"

Training Process Visualization:



Objective Function:

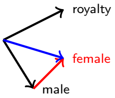
$$\max \sum_{t=1}^{T} \sum_{-c \leq j \leq c} \log P(w_{t+j} | w_t)$$

Maximize probability of context words

Embeddings Capture Analogies!

Famous Examples:

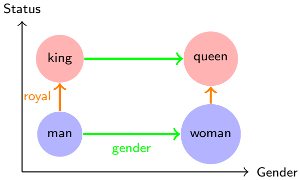
$\boxed{\text{king}} - \boxed{\text{man}} + \boxed{\text{woman}} = \boxed{\text{queen}}$



More Analogies:

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

Why Does This Work?



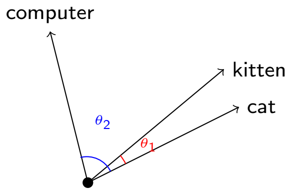
The Pattern:

- 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:

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

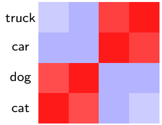


- cat $\sim$ kitten: $\cos(\theta_1) = 0.95$
- cat $\sim$ computer: $\cos(\theta_2) = 0.1$

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:



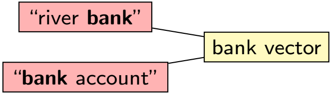
Animals cluster together, vehicles cluster together!

Evolution: From Static to Contextual Embeddings

The Next Revolution: Context Matters!

Problem with Static Embeddings:

One word = One vector always



Same vector!

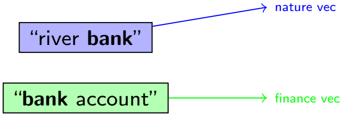
But “bank” has different meanings!

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

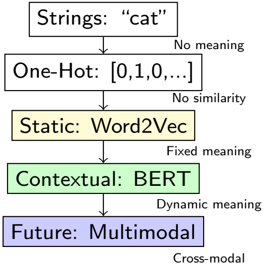
Key Advance: Vector depends on surrounding words!

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!

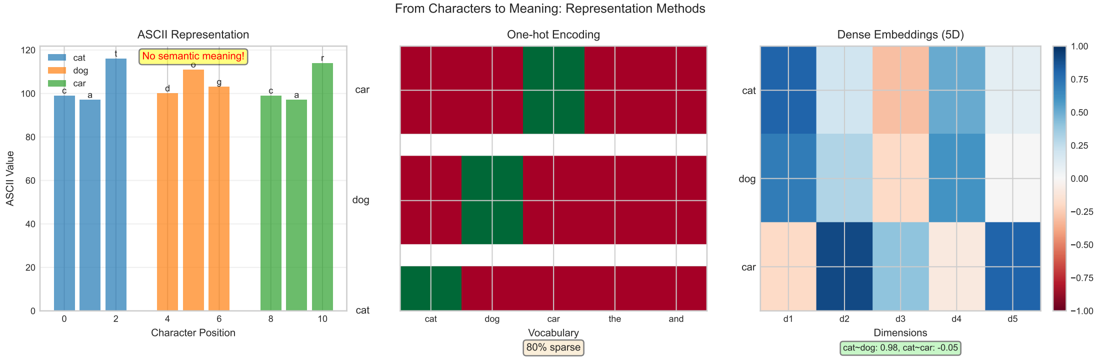
Part II

Advanced Theory & Visualizations

Deeper Insights into Embedding Mathematics

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:

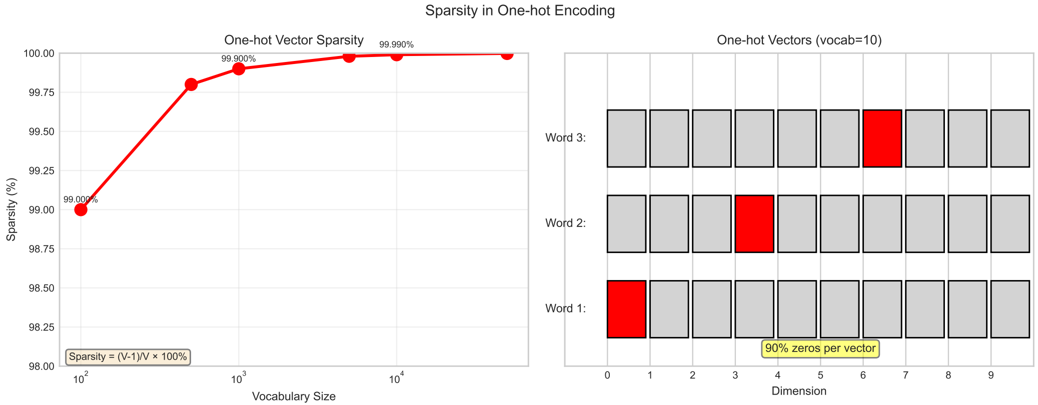
- Each word = sparse vector
- 99.9% zeros
- All words equally different

Dense Embedding:

- Each word = dense vector
- All values meaningful
- Similar words $\rightarrow$ similar vectors

Key: Embeddings encode meaning, not just identity!

Why One-hot Encoding is Inefficient



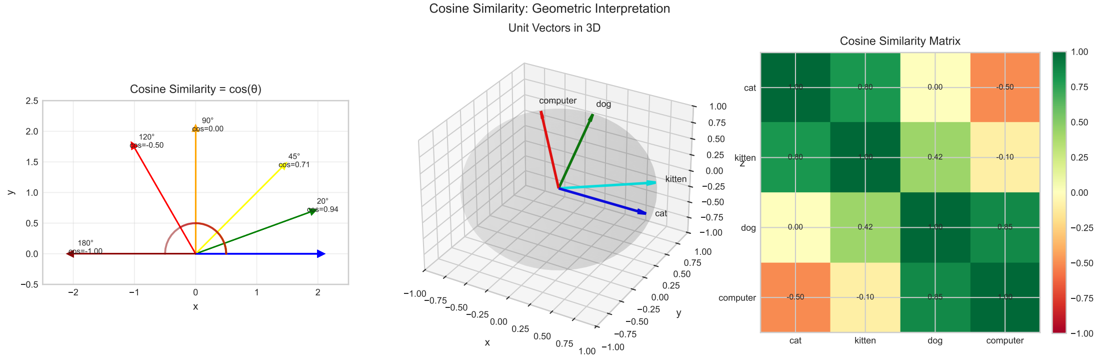
Mathematical Analysis:

- 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:

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

Cosine Similarity: Mathematical Properties

Why Cosine Similarity Works for Embeddings

The Formula:

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

Key Properties:

Scale Invariance:

- $\cos(\vec{a}, \vec{b}) = \cos(k\vec{a}, \vec{b})$
- 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: $\cos(a, b) = \cos(b, a)$

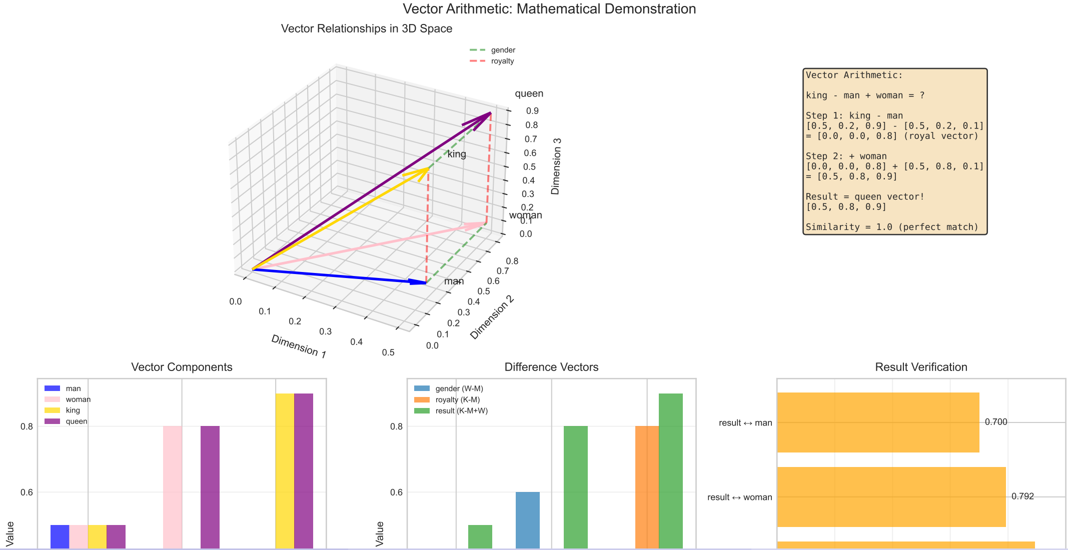
Applications in NLP:

- 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

Embeddings Can Do Analogies!



Vector Arithmetic: Diverse Analogies

The Pattern Works Across Many Domains

Geographic Analogies:

- Paris - France + Germany = **Berlin** (capital relationships)
- Madrid - Spain + Italy = **Rome**
- Tokyo - Japan + UK = **London**

Grammatical Transformations:

Tense Changes:

- walked - walk + run = **ran**
- going - go + eat = **eating**
- saw - see + do = **did**

Comparative/Superlative:

- bigger - big + small = **smaller**
- best - good + bad = **worst**
- faster - fast + slow = **slower**

Semantic Relations:

- Einstein - scientist + artist = **Picasso**
- Microsoft - Gates + Jobs = **Apple**
- nephew - uncle + aunt = **niece**

Success Rates:

- Syntactic analogies: 70-80% accuracy
- Semantic analogies: 60-70% accuracy
- Performance improves with more training data

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: $\vec{g} = \vec{woman} - \vec{man}$
- Royalty vector: $\vec{r} = \vec{king} - \vec{man}$

Step-by-Step Derivation:

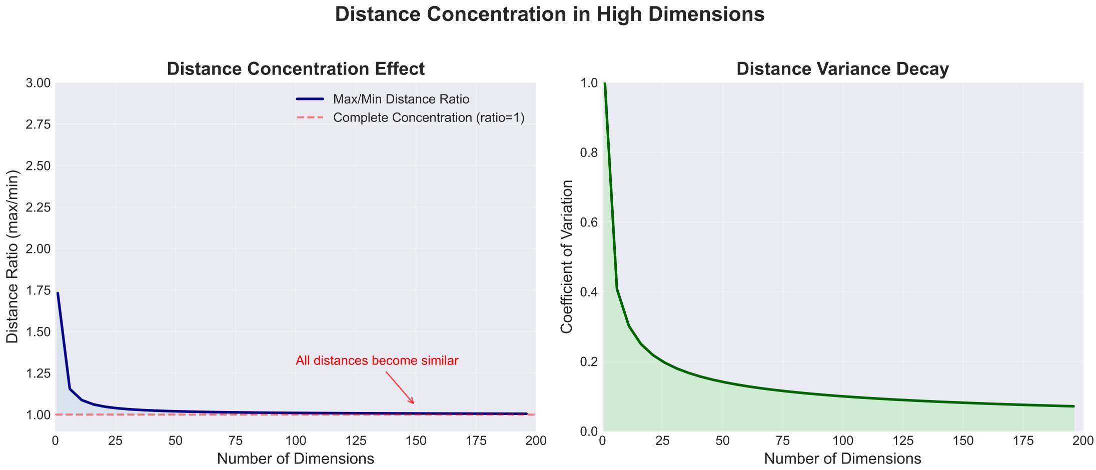
$$\vec{king} = \vec{man} + \vec{r} \quad \text{(man + royalty = king)} \tag{1}$$
$$\vec{queen} = \vec{woman} + \vec{r} \quad \text{(woman + royalty = queen)} \tag{2}$$
$$\therefore \vec{queen} = \vec{woman} + (\vec{king} - \vec{man}) \tag{3}$$
$$= \vec{king} - \vec{man} + \vec{woman} \tag{4}$$

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

What the Visualizations Show
Distance Ratio Convergence:

- $\frac{\text{dist}_{max} - \text{dist}_{min}}{\text{dist}_{mean}} \to 0$ as $d \to \infty$
- For Gaussian points: ratio $\approx \sqrt{1 + 2/d}$
- At d=100: all distances within 10% of mean
- At d=1000: essentially all points equidistant

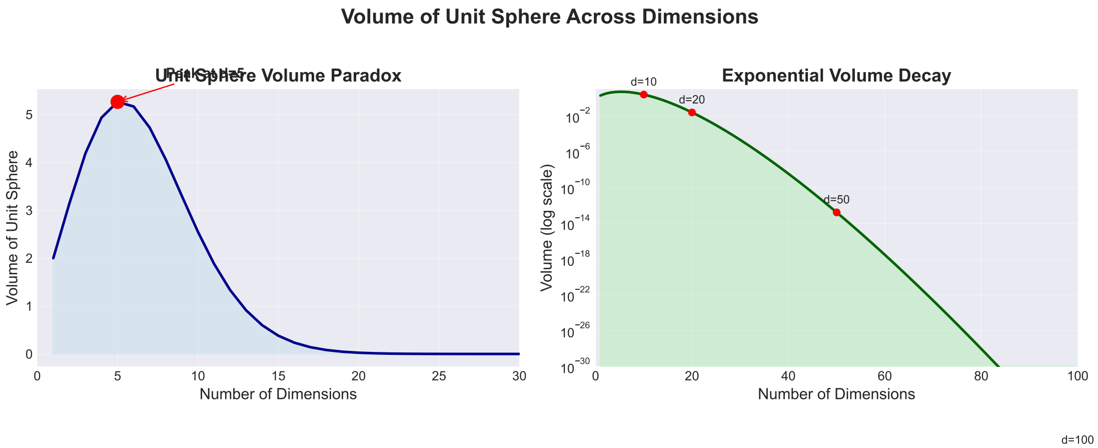
Implications for Machine Learning:

- 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

Key Takeaway: In high dimensions, the concept of "near" and "far" becomes meaningless - all points are approximately the same distance apart!

The Volume Paradox: Visual Evidence

Unit Sphere Volume Across Dimensions



The Volume Formula:

$\pi^{d/2}$

V

Why Volume Goes to Zero: The Mathematics

Understanding the Formula

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

Numerator (Top):

- $\pi^{d/2} \approx (3.14)^{d/2}$
- Grows exponentially
- But base is small: $\sqrt{\pi} \approx 1.77$
- Growth rate: 1.77^d
- Example: $1.77^{100} \approx 10^{25}$

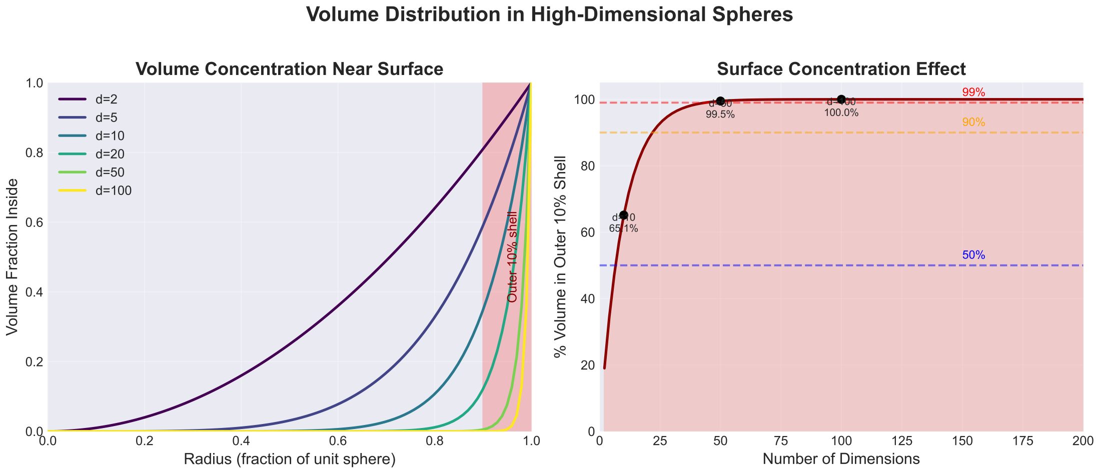
Denominator (Bottom):

- $\Gamma(n+1) = n!$ for integers
- Factorial growth is MUCH faster
- Example: $50! \approx 10^{64}$
- Stirling: $n! \approx \sqrt{2\pi n}(n/e)^n$
- Dominates numerator completely

The Key Mathematical Insight:

Factorial growth beats exponential growth!
$\frac{1.77^d}{(d/2)!} \to 0$ extremely fast as $d \to \infty$
Factorial grows like $(n/e)^n$ while exponential is just a^n

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):

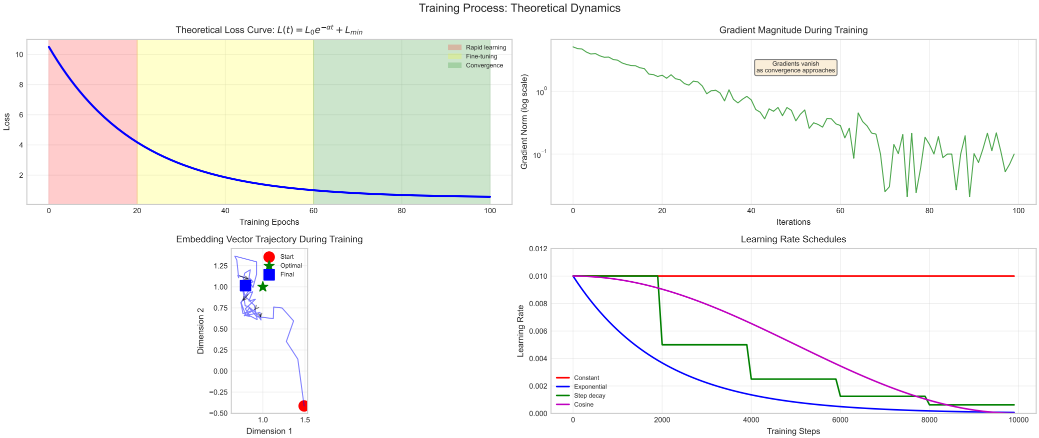
- Faster training
- Less overfitting
- Good for specific domains

Higher Dimensions (768-1024+):

- More expressive power
- Better for transfer learning
- Captures subtle relationships

Rapid Learning: Gradient Dynamics (Epochs 0-20)

Why Training Starts Fast



Gradient Behavior in Early Training:

Initial State:

- Random initialization: $\mathcal{N}(0, 0.01)$

Update Characteristics:

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

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%

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.

Mathematical Foundations: Skip-gram Objective

Formal Skip-gram Model Definition
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 the input vector representation of word w_I
- v'_{w_O} is the output vector representation of word w_O
- W is the vocabulary size

Gradient w.r.t. Input Vector:

$$\frac{\partial J}{\partial v_{w_I}} = \sum_{j=-c}^{c}\left(\sum_{w=1}^{W} p(w|w_I)v'_w - v'_{w_{t+j}}\right)$$

Computational Complexity: $O(W)$ per word - intractable for large vocabularies!

Modified Objective with Negative Sampling
Replace softmax with:

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

where:

- $\sigma(x) = \frac{1}{1+e^{-x}}$ (sigmoid function)
- k is the number of negative samples (typically 5-20)
- $P_n(w)$ is the noise distribution: $P_n(w) = \frac{U(w)^{3/4}}{\sum_{w'} U(w')^{3/4}}$
- $U(w)$ is the unigram distribution

Gradient Update:

$$v_{w_I}^{new} = v_{w_I}^{old} - \eta \left[(\sigma({v'_{w_O}}^T v_{w_I}) - 1) v'_{w_O} + \sum_{i=1}^{k} \sigma({v'_{w_i}}^T v_{w_I}) v'_{w_i} \right]$$

Complexity Reduction: From $O(W)$ to $O(k+1)$ per training example

GloVe: Global Vectors Mathematical Framework

Co-occurrence Matrix and Ratios
Define co-occurrence matrix X where X_{ij} = count of word j appearing in context of word i
Key Insight - Ratio of Probabilities:

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

This ratio encodes semantic relationships!
GloVe Objective Function:

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

where:

- $f(x)$ is a weighting function: $f(x) = \begin{cases} (x/x_{max})^\alpha & \text{if } x < x_{max} \\ 1 & \text{otherwise} \end{cases}$
- $w_i, \tilde{w}_j$ are word and context vectors
- $b_i, \tilde{b}_j$ are bias terms
- Typical: $\alpha = 0.75$, $x_{max} = 100$

Final Embedding: $e_i = w_i + \tilde{w}_i$ (symmetric combination)

Self-Attention: Mathematical Formulation

Scaled Dot-Product Attention
Given queries $Q \in \mathbb{R}^{n \times d_k}$, keys $K \in \mathbb{R}^{m \times d_k}$, values $V \in \mathbb{R}^{m \times d_v}$:

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

Detailed Computation:

1. Score matrix: $S = QK^T \in \mathbb{R}^{n \times m}$
2. Scaled scores: $\tilde{S}_{ij} = \frac{S_{ij}}{\sqrt{d_k}}$ (prevents gradient vanishing)
3. Attention weights: $A_{ij} = \frac{\exp(\tilde{S}_{ij})}{\sum_{j'=1}^{m} \exp(\tilde{S}_{ij'})}$
4. Output: $O = AV \in \mathbb{R}^{n \times d_v}$

Multi-Head Attention:

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \ldots, \text{head}_h) W^O$$

$$\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

where $W_i^Q \in \mathbb{R}^{d_{model} \times d_k}$, $W_i^K \in \mathbb{R}^{d_{model} \times d_k}$, $W_i^V \in \mathbb{R}^{d_{model} \times d_v}$

Positional Encoding: Injecting Order Information

Sinusoidal Position Encoding
For position *pos* and dimension *i*:

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

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

Properties:

- Unique encoding for each position
- Allows model to attend to relative positions
- For any fixed offset k: PE_{pos+k} can be represented as linear function of PE_{pos}

Proof of Relative Position Property:

$$PE_{pos+k,2i} = \sin(\omega_i \cdot pos)\cos(\omega_i \cdot k) + \cos(\omega_i \cdot pos)\sin(\omega_i \cdot k)$$

where $\omega_i = \frac{1}{10000^{2i/d_{model}}}$
This is a linear transformation of PE_{pos}!

BERT: Bidirectional Training Mathematics

Masked Language Model (MLM) Objective
Given input sequence $\mathbf{x} = (x_1, ..., x_n)$, randomly mask 15% of tokens.
MLM Loss:

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

where $\mathcal{M}$ is the set of masked positions.
Next Sentence Prediction (NSP) Loss:

$$\mathcal{L}_{NSP} = -\mathbb{E}_{(A,B)\sim\mathcal{D}}[y \log P(\text{IsNext}|A, B) + (1 - y) \log(1 - P(\text{IsNext}|A, B))]$$

where $y = 1$ if B follows A, else $y = 0$.
Combined Objective:

$$\mathcal{L}_{BERT} = \mathcal{L}_{MLM} + \mathcal{L}_{NSP}$$

Output Probability:

$$P(x_i|\mathbf{x}_{\setminus\mathcal{M}}) = \text{softmax}(W_o h_i + b_o)$$

where h_i is the final hidden state at position i.

Layer Normalization in Transformers

Layer Normalization Mathematics
For hidden state $\mathbf{h} \in \mathbb{R}^d$:
Statistics:

$$\mu = \frac{1}{d}\sum_{i=1}^{d} h_i \qquad \sigma^2 = \frac{1}{d}\sum_{i=1}^{d}(h_i - \mu)^2$$

Normalization:

$$\hat{h}_i = \frac{h_i - \mu}{\sqrt{\sigma^2 + \epsilon}}$$

Affine Transformation:

$$\text{LayerNorm}(\mathbf{h})_i = \gamma_i \hat{h}_i + \beta_i$$

where $\gamma, \beta \in \mathbb{R}^d$ are learned parameters.
Gradient Flow:

$$\frac{\partial \mathcal{L}}{\partial h_i} = \frac{\gamma_i}{\sqrt{\sigma^2 + \epsilon}} \left[\frac{\partial \mathcal{L}}{\partial \hat{h}_i} - \frac{1}{d}\sum_{j=1}^{d} \frac{\partial \mathcal{L}}{\partial \hat{h}_j} - \frac{\hat{h}_i}{d}\sum_{j=1}^{d} \hat{h}_j \frac{\partial \mathcal{L}}{\partial \hat{h}_j} \right]$$

This ensures stable gradients across layers!

Mutual Information Maximization
Embeddings maximize mutual information between words and contexts:

$$I(W;C) = \sum_{w \in \mathcal{W}} \sum_{c \in \mathcal{C}} p(w,c) \log \frac{p(w,c)}{p(w)p(c)}$$

Pointwise Mutual Information (PMI):

$$\mathrm{PMI}(w,c) = \log \frac{p(w,c)}{p(w)p(c)} = \log \frac{p(w|c)}{p(w)}$$

Connection to Skip-gram: Skip-gram with negative sampling implicitly factorizes shifted PMI matrix:

$$\mathbf{w}^T \mathbf{c} \approx \mathrm{PMI}(w,c) - \log k$$

Optimal Embedding Dimension: From Johnson-Lindenstrauss lemma, to preserve pairwise distances with ϵ error:

$$d = O\left(\frac{\log n}{\epsilon^2}\right)$$

where n is vocabulary size, d is embedding dimension.
Entropy of Word Distribution:

$$H(W) = -\sum_{w \in \mathcal{W}} p(w) \log p(w)$$

Higher entropy $\Rightarrow$ need higher dimensional embeddings

Singular Value Decomposition of Co-occurrence
Co-occurrence matrix $\mathbf{X} \in \mathbb{R}^{|V| \times |V|}$ decomposition:

$$\mathbf{X} = \mathbf{U\Sigma V}^T$$

Truncated SVD for Embeddings:

$$\mathbf{W} = \mathbf{U}_k \mathbf{\Sigma}_k^{1/2}$$

where k is the embedding dimension.
Eigenvalue Distribution: For natural language, eigenvalues follow power law:

$$\lambda_i \propto i^{-\alpha}$$

Typically $\alpha \approx 1.5$ for word co-occurrence matrices.
Effective Rank:

$$r_{eff} = \exp\left(-\sum_{i=1}^{n} \frac{\lambda_i}{\sum_j \lambda_j} \log \frac{\lambda_i}{\sum_j \lambda_j}\right)$$

Spectral Norm Regularization:

$$\mathcal{L}_{reg} = \mathcal{L}_{task} + \lambda ||\mathbf{W}||_2$$

where $||\mathbf{W}||_2 = \sigma_{max}(\mathbf{W})$ is the largest singular value.

Non-convex Optimization Problem
Word2Vec optimization:

$$\min_{\mathbf{W},\mathbf{C}} \sum_{(i,j)\in\mathcal{D}} -\log\sigma(\mathbf{w}_i^T\mathbf{c}_j) - \sum_{k\sim P_n} \log\sigma(-\mathbf{w}_i^T\mathbf{c}_k)$$

Critical Points Analysis:

- Saddle points dominate in high dimensions
- Hessian eigenvalue distribution: mostly negative with few positive
- Gradient norm at initialization: $||\nabla\mathcal{L}||_2 \approx \sqrt{d}$

Convergence Rate (SGD):

$$\mathbb{E}[\mathcal{L}(\mathbf{w}_t)] - \mathcal{L}^* \leq \frac{||\mathbf{w}_0 - \mathbf{w}^*||^2}{2\eta t} + \frac{\eta L\sigma^2}{2}$$

where η is learning rate, L is Lipschitz constant, σ^2 is gradient variance.
Adaptive Learning Rate (Adam):

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \eta\frac{\hat{\mathbf{m}}_t}{\sqrt{\hat{\mathbf{v}}_t} + \epsilon}$$

where $\hat{\mathbf{m}}_t$, $\hat{\mathbf{v}}_t$ are bias-corrected moment estimates.

Contextual Embeddings: Mathematical Framework

ELMo: Bidirectional Language Model

Forward LM:

$$p(t_1, t_2, ..., t_N) = \prod_{k=1}^{N} p(t_k | t_1, ..., t_{k-1})$$

Backward LM:

$$p(t_1, t_2, ..., t_N) = \prod_{k=1}^{N} p(t_k | t_{k+1}, ..., t_N)$$

ELMo Representation:

$$\text{ELMo}_k^{task} = \gamma^{task} \sum_{j=0}^{L} s_j^{task} \mathbf{h}_{k,j}^{LM}$$

where:

- $\mathbf{h}_{k,j}^{LM}$ is the j-th layer representation for token k
- s_j^{task} are softmax-normalized weights
- γ^{task} is a task-specific scale parameter

Contextual Variation:

$$\text{Var}(\mathbf{e}_w) = \mathbb{E}_{c \sim \mathcal{C}(w)}[||\mathbf{e}_{w,c} - \bar{\mathbf{e}}_w||^2]$$

where $\mathcal{C}(w)$ is the set of contexts for word w.

Isotropy and Anisotropy
Isotropy Measure:

$$I(\mathbf{W}) = \frac{\min_i \lambda_i}{\max_i \lambda_i}$$

where λ_i are eigenvalues of $\mathbf{W}^T\mathbf{W}$.
Average Cosine Similarity:

$$\bar{\rho} = \frac{2}{n(n-1)} \sum_{i<j} \frac{\mathbf{w}_i^T \mathbf{w}_j}{||\mathbf{w}_i|| \cdot ||\mathbf{w}_j||}$$

Pre-trained embeddings often show $\bar{\rho} > 0.5$ (anisotropic).
Cone Effect: Embeddings often lie in narrow cone with half-angle:

$$\theta = \arccos(\min_{i \neq j} \cos(\mathbf{w}_i, \mathbf{w}_j))$$

Post-processing for Isotropy:

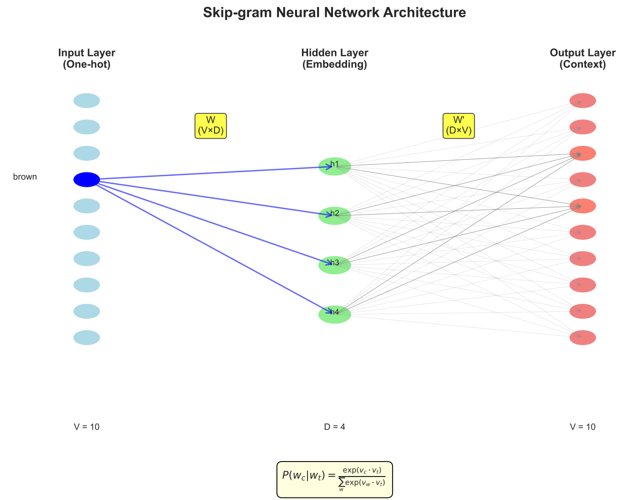
1. Mean centering: $\tilde{\mathbf{w}}_i = \mathbf{w}_i - \bar{\mathbf{w}}$
2. All-but-the-top: Remove top principal components
3. Whitening: $\tilde{\mathbf{W}} = (\mathbf{W} - \mu)\mathbf{\Sigma}^{-1/2}$

Intrinsic Dimension:

$$d_{int} = \frac{(\sum_i \lambda_i)^2}{\sum_i \lambda_i^2}$$

Typically $d_{int} \ll d$ for word embeddings.

How the Network Processes Words

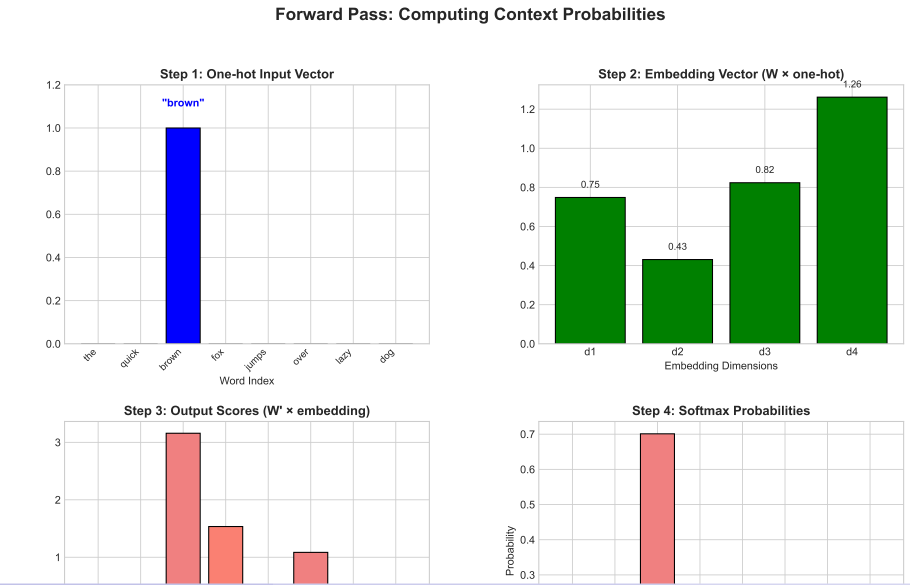


Key Components:

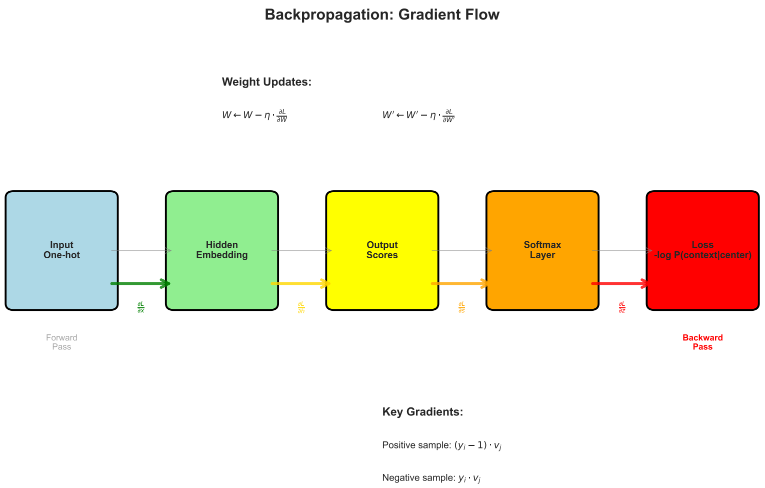
- **Input:** One-hot word (V dimensions)

Extracting (Center, Context) Pairs

Forward Pass: Computing Context Probabilities



Backpropagation: Learning the Embeddings



Updates:

How Embeddings Evolve