Holographic Reduced Representations

CS 135 - Brandeis University
Fall 2024
James Pustejovsky

November 22, 2024

Introduction

We explore three methods to represent sentences as vectors:

- Conventional vector composition
- Transformer-based contextualized embeddings
- Holographic Reduced Representations (HRRs)

We explore some semantic problems:

- Nouns as Vectors / Adjectives as Matrices
- Generative Lexicon encoded as Vector Binding
- Type Coercion in Vector Semantics

What Are One-Hot Vectors?

- A one-hot vector is a binary vector used to represent categorical data.
- For a vocabulary of size V, each word is assigned a unique index i where:

$$\mathbf{v}_i = [0, 0, \ldots, 1, \ldots, 0]$$

- Example:
  - Vocabulary: ["cat", "dog", "fish"]
  - "cat" $\rightarrow [1, 0, 0]$, "dog" $\rightarrow [0, 1, 0]$, "fish" $\rightarrow [0, 0, 1]$

Limitations of One-Hot Vectors

- High Dimensionality:

$$\text{Dimension of vector} = V$$

  For large vocabularies ($V > 100,000$), the vectors become inefficient.
- Lack of Semantic Information:
  - No similarity between "cat" and "dog".
  - All vectors are orthogonal.
- Solution: Use Word2Vec to map one-hot vectors into dense, low-dimensional embeddings.

From One-Hot Vectors to Word Embeddings (Word2Vec)

Word2Vec learns dense vector representations for words by analyzing their context in a corpus.

- Input: One-hot vector for each word.
- Output: Dense, low-dimensional embedding ($\mathbf{w} \in \mathbb{R}^d$).

Key Idea: Distributional Hypothesis

Words that appear in similar contexts have similar meanings.

Two training methods:

- Skip-gram: Predict context words from a target word.
- CBOW (Continuous Bag of Words): Predict a target word from context words.

Example: Word2Vec Conversion

- Vocabulary: ["cat", "dog", "fish"]
- One-hot vectors:

$$\text{"cat"} = [1,0,0], \quad \text{"dog"} = [0,1,0], \quad \text{"fish"} = [0,0,1]$$

- Dense embeddings (Word2Vec output):

$$\text{"cat"} = [0.5, 0.1, 0.3], \text{"dog"} = [0.4, 0.2, 0.5], \text{"fish"} = [0.3, 0.8, 0.2]$$

- These embeddings capture semantic similarity:

$$\text{Similarity("cat", "dog")} > \text{Similarity("cat", "fish")}$$

What is Skip-gram?

- Word2Vec learns dense vector representations for words by predicting their context in a corpus.
- Skip-gram Model:
  - Predicts context words (w_c) given a target word (w_t).
  - Objective: Maximize the probability of context words given the target word:

$$P(w_c|w_t)$$

- Embedding Space:
  - Each word is mapped to a dense vector ($d \sim 100 - 300$).
  - Vectors capture semantic similarity (e.g., "king" and "queen").

Details of the Skip-gram Model

Objective Function

For a given corpus, the Skip-gram model maximizes the conditional probability:

$$\prod_{t=1}^{T} \prod_{-c \leq j \leq c, j \neq 0} P(w_{t+j} | w_t)$$

where:

- w_t: Target word.
- w_{t+j}: Context words within a window of size c.

Details of the Skip-gram Model

Log-Likelihood

Taking the logarithm, the objective becomes:

$$\mathcal{L} = \sum_{t=1}^{T} \sum_{-c \leq j \leq c, j \neq 0} \log P(w_{t+j} | w_t)$$

Conditional Probability

The probability $P(w_{t+j}|w_t)$ is modeled using softmax:

$$P(w_{t+j}|w_t) = \frac{\exp(\mathbf{u}_{w_{t+j}}^\top \mathbf{v}_{w_t})}{\sum_{w \in V} \exp(\mathbf{u}_w^\top \mathbf{v}_{w_t})}$$

- $\mathbf{v}_{w_t}$: Embedding for the target word.
- $\mathbf{u}_{w_{t+j}}$: Embedding for the context word.

Skip-gram Training Steps

1. Initialize two embedding matrices:
   - W (target embeddings): $|V| \times d$
   - W' (context embeddings): $|V| \times d$
2. For each target word w_t:
   - Predict each context word w_c in the window $[-c, c]$.
3. Compute the loss (negative log-likelihood):

$$\mathcal{L} = -\log P(w_c | w_t)$$

4. Update W and W' using stochastic gradient descent (SGD).

Computing Gradients for Skip-gram

For a single pair (w_t, w_c), the loss is:

$$\mathcal{L} = -\log P(w_c | w_t)$$

Gradient for Target Embedding ($\mathbf{v}_{w_t}$)

$$\frac{\partial \mathcal{L}}{\partial \mathbf{v}_{w_t}} = \mathbf{u}_{w_c} - \sum_{w \in V} P(w|w_t)\mathbf{u}_w$$

Gradient for Context Embedding ($\mathbf{u}_{w_c}$)

$$\frac{\partial \mathcal{L}}{\partial \mathbf{u}_{w_c}} = \mathbf{v}_{w_t} - \sum_{w \in V} P(w|w_t)\mathbf{v}_{w_t}$$

The gradients are used to update the embeddings via SGD.

Negative Sampling: Reducing Computation

- Problem: Softmax requires summing over all words in the vocabulary ($|V|$).
- Solution: Use Negative Sampling to approximate softmax.

$$\log P(w_c|w_t) \approx \log \sigma(\mathbf{u}_{w_c}^\top \mathbf{v}_{w_t}) + \sum_{i=1}^{k} \mathbb{E}_{w \sim P_n(w)} \left[\log \sigma(-\mathbf{u}_w^\top \mathbf{v}_{w_t}) \right]$$

- $P_n(w)$: Noise distribution for negative samples.
- k: Number of negative samples per positive pair.

Advantages

- Reduces computation from $O(|V|)$ to $O(k)$.
- Focuses on distinguishing the target-context pairs from random noise.

Worked Example: Skip-gram Training Step

Consider a toy vocabulary: $V = \{\text{cat}, \text{dog}, \text{fish}\}$.

- Target word: $w_t = \text{cat}$.
- Context words: $w_c \in \{\text{dog}, \text{fish}\}$.
- Embedding dimension: $d = 2$.
- Initialize embeddings:

$$\mathbf{v}_{\text{cat}} = [0.1, 0.3], \quad \mathbf{u}_{\text{dog}} = [0.2, 0.4], \quad \mathbf{u}_{\text{fish}} = [0.3, 0.1]$$

Compute $P(w_c|w_t)$ for $w_c = \text{dog}$:

$$P(w_c|w_t) = \frac{\exp(\mathbf{u}_{\text{dog}}^\top \mathbf{v}_{\text{cat}})}{\sum_{w \in V} \exp(\mathbf{u}_w^\top \mathbf{v}_{\text{cat}})}$$

Numerator:

$$\mathbf{u}_{\text{dog}}^\top \mathbf{v}_{\text{cat}} = (0.2)(0.1) + (0.4)(0.3) = 0.14$$

Denominator:

$$\text{Sum} = \exp(0.14) + \exp(0.11) + \exp(0.03)$$

Result:

$$P(\text{dog}|\text{cat}) = \frac{\exp(0.14)}{\exp(0.14) + \exp(0.11) + \exp(0.03)} = \ldots$$

Final Word Embeddings

- After training, each word has two embeddings:
  - $\mathbf{v}_{w_t}$: Represents the word as a target.
  - $\mathbf{u}_{w_c}$: Represents the word as a context.
- Combine these embeddings (e.g., by averaging) to create the final word vector:

$$\mathbf{w} = \frac{\mathbf{v}_{w_t} + \mathbf{u}_{w_c}}{2}$$

- These embeddings capture semantic similarity and are used in downstream tasks.

Using Word2Vec for Analogical Reasoning

- Analogical reasoning involves finding relationships between pairs of words or concepts.

"king is to queen as man is to woman."

- Represented mathematically as:

$$\mathbf{king} - \mathbf{man} \approx \mathbf{queen} - \mathbf{woman}.$$

- Captures relational patterns in vector space.

Latent Semantic Analysis (LSA)

- LSA uses singular value decomposition (SVD) to reduce the dimensionality of term-document matrices.
- Represents words and documents as vectors in a semantic space:

$$\mathbf{M} \in \mathbb{R}^{|V| \times |D|} \quad \rightarrow \quad \mathbf{M}_{\text{reduced}} \in \mathbb{R}^{|V| \times k}.$$

- Captures semantic relationships:

  cosine similarity between word vectors reflects semantic similarity.

Limitations

- LSA focuses on co-occurrence, not relational patterns.
- Cannot explicitly represent analogies.

The Parallelogram Hypothesis

- Hypothesis: Analogical reasoning can be represented as geometric relationships in vector space.
- Example:

$$\textbf{king} - \textbf{man} + \textbf{woman} \approx \textbf{queen}.$$

- Geometric Interpretation:
  - The vector from **man** to **king** is parallel to the vector from **woman** to **queen**.
- Visualized as a parallelogram:
  Given three points, solve for the fourth:

$$\textbf{queen} = \textbf{king} - \textbf{man} + \textbf{woman}.$$

How Word2Vec Derives Analogies

- Word2Vec learns dense word embeddings that capture semantic and syntactic relationships.
- Relationships are encoded in the directions between vectors.
- Analogy-solving formula:

$$\mathbf{w}_4 = \arg\max_{\mathbf{w} \in V} \cos\left(\mathbf{w}, \mathbf{w}_2 - \mathbf{w}_1 + \mathbf{w}_3\right),$$

where:

  - $\mathbf{w}_1 = \text{man}, \mathbf{w}_2 = \text{king}, \mathbf{w}_3 = \text{woman},$
  - $\mathbf{w}_4 = \text{queen}.$

Worked Example: Word2Vec Analogy

Example: Solve "king is to queen as man is to woman":

- Vectors:

$$\mathbf{king} = [0.8, 0.6], \quad \mathbf{queen} = [0.9, 0.7],$$

$$\mathbf{man} = [0.2, 0.4], \quad \mathbf{woman} = [0.3, 0.5].$$

- Compute:

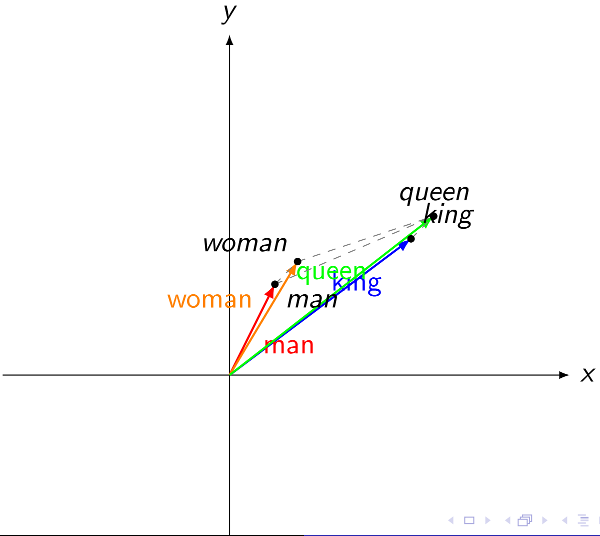
$$\mathbf{queen} = \mathbf{king} - \mathbf{man} + \mathbf{woman}.$$

$$\mathbf{queen} = [0.8, 0.6] - [0.2, 0.4] + [0.3, 0.5].$$

Result:

$$\mathbf{queen} = [0.9, 0.7].$$

Vector Analogy with Parallelogram Visualization



Why Does Word2Vec Work?

- Co-occurrence Modeling:
  - Word2Vec captures context relationships via training on skip-grams.
- Semantic Directionality:
  - Embeddings encode directional relationships (e.g., gender, tense).
- Vector Arithmetic:
  - The geometry of word embeddings allows analogical reasoning through addition and subtraction.

What Word2Vec Does Not Explain

- Syntax-Semantics Interface:
  - Analogies focus on semantics; no explicit representation of syntactic structure.
- Complex Analogies:
  - Cannot handle multi-step or hierarchical relationships.
- Context Dependence:
  - Word2Vec embeddings are static, ignoring polysemy and contextual nuances.
- Empirical Limitations:
  - Only works well for analogies seen in training or closely related domains.

Key Takeaways

- Analogical reasoning is a fundamental capability of word embeddings like Word2Vec.
- The parallelogram hypothesis explains how analogies are geometrically encoded in vector space.
- Limitations:
  - Word2Vec does not capture syntax or hierarchical relationships.
  - Contextualized embeddings (e.g., BERT) address some limitations but are less interpretable.
- Analogical reasoning with vectors demonstrates the power and constraints of distributional semantics.

Transformer-Based Contextualized Embeddings

Overview of Self-Attention in Transformers

- Self-attention computes relationships between tokens in a sentence.
- Outputs contextualized representations for each token.

Self-Attention Formula

For query (Q), key (K), and value (V) matrices:

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

- $Q, K, V \in \mathbb{R}^{n \times d_k}$, where n is the number of tokens and d_k is the embedding size.
- Each token generates its own query, key, and value vectors.

Step 1: Input to Embeddings

Given a sentence:

"The bank will not approve the loan."

- Tokens: [The, bank, will, not, approve, the, loan].
- Embedding dimension: $d_k = 4$ (for simplicity).

Token embeddings (random initialization for this example):

$$\mathbf{x}_1 = [1, 0, 1, 0],\ \mathbf{x}_2 = [0, 1, 0, 1], \ldots$$

Step 2: Compute Query, Key, and Value Matrices

Each token embedding is projected into query, key, and value spaces using learned weight matrices:

$$Q = XW_Q, \quad K = XW_K, \quad V = XW_V$$

Example weights ($W_Q, W_K, W_V \in \mathbb{R}^{d_k \times d_k}$):

$$W_Q = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \end{bmatrix}, \; W_K = \begin{bmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 \end{bmatrix}, \ldots$$

For token 1 ($\mathbf{x}_1 = [1, 0, 1, 0]$):

$$Q_1 = \mathbf{x}_1 W_Q = [1, 0, 1, 0] \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \end{bmatrix} = [2, 0, 1, 0]$$

Step 3: Compute Attention Scores

Compute scaled dot-product attention:

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

For Q_1 and K_2:

$$Q_1 \cdot K_2 = [2, 0, 1, 0] \cdot [0, 1, 0, 1] = 0$$

Attention scores matrix:

$$\text{Scores}_{i,j} = \frac{Q_i \cdot K_j}{\sqrt{d_k}}, \quad i, j \in \{1, \ldots, n\}$$

Normalize scores using softmax:

$$\text{softmax}(z_i) = \frac{\exp(z_i)}{\sum_j \exp(z_j)}$$

Step 4: Compute Weighted Values

For token 1:

$$\text{Attention}(Q_1, K, V) = \text{softmax}\left(\frac{Q_1 K^T}{\sqrt{d_k}}\right) V$$

Example:

$$\text{Scores}_{1,} = \text{softmax}\left(\frac{[0, 1, 2]}{\sqrt{4}}\right) = [0.04, 0.11, 0.85]$$

Use scores to compute weighted sum:

$$\mathbf{z}_1 = \sum_{j=1}^{n} \text{Scores}_{1,j} \cdot V_j$$

Worked Example: Final Contextualized Embeddings

For each token i, compute:

$$\mathbf{z}_i = \sum_{j=1}^{n} \text{Attention}(Q_i, K_j, V_j)$$

Result:

$$Z = \begin{bmatrix} \mathbf{z}_1 \\ \mathbf{z}_2 \\ \vdots \\ \mathbf{z}_n \end{bmatrix}, \quad Z \in \mathbb{R}^{n \times d_k}$$

Aggregated sentence embedding:

$$\mathbf{S}_{\text{Transformer}} = \text{Mean}(Z)$$

or use special token [CLS].

Motivation for Hypervectors and Hyperdimensional Computing

- Hypervectors: High-dimensional vectors ($d \gg 1000$) used to represent information in a distributed manner.
- Inspired by the properties of the brain:
  - Robustness to noise.
  - Ability to store and retrieve large amounts of information.
- Key idea:
  Complex structures can be represented as combinations of simple high-dimensional vectors.

What is a Hyperdimensional Vector Space?

- A hyperdimensional vector space is a high-dimensional space ($d \gg 1000$) used to represent information.
- Hypervectors ($\mathbf{v} \in \mathbb{R}^d$):
  - Randomly initialized.
  - High dimensionality ensures approximate orthogonality between vectors.

Properties of High-Dimensional Spaces

- Orthogonality:

$$\text{For random vectors } \mathbf{v}, \mathbf{w} \in \mathbb{R}^d : \quad \mathbf{v} \cdot \mathbf{w} \approx 0 \quad (\text{if } \mathbf{v} \neq \mathbf{w}).$$

- Stability:

$$\|\mathbf{v}_1 + \mathbf{v}_2 + \cdots + \mathbf{v}_n\| \text{ grows with } n.$$

- Capacity:

  Large spaces can encode exponentially many distinct patterns.

How do Hyperdimensional Spaces Encode Patterns"

- Key Property: In a hyperdimensional space $\mathbb{R}^d$ with $d \gg 1000$:

$$\text{Number of distinct patterns is exponential in } d.$$

- Intuition:
  - A random hypervector $\mathbf{v} \in \mathbb{R}^d$ has d components.
  - Each component can take on many possible values (e.g., $[-1, 1]$ for bipolar vectors or $\mathbb{R}$ for real-valued vectors).
- Mathematical Argument:
  - Consider d-dimensional binary vectors $\{0, 1\}^d$:

$$\text{Total number of distinct vectors: } 2^d.$$

  - For real-valued or bipolar vectors, the number of distinct patterns grows even faster.

Geometric Perspective: Orthogonality in High Dimensions

- High-dimensional spaces have the property that random vectors are nearly orthogonal:

$$\mathbf{v}_1 \cdot \mathbf{v}_2 \approx 0, \quad \text{if } \mathbf{v}_1, \mathbf{v}_2 \text{ are random.}$$

- Implication:
  - You can generate exponentially many random hypervectors that are distinguishable (linearly independent or approximately orthogonal).
- Example:
  - In $\mathbb{R}^{10,000}$, billions of random vectors will have dot products close to zero.

Superposition and Binding in Hyperdimensional Spaces

- Superposition (addition):

$$\mathbf{v} = \mathbf{v}_1 + \mathbf{v}_2 + \cdots + \mathbf{v}_n.$$

- Even with $n \gg 1$, the resulting vector is still distinguishable due to high dimensionality.
- Binding (e.g., circular convolution):

$$\mathbf{c} = \mathbf{a} \circledast \mathbf{b}.$$

- Each binding operation produces a new, distinguishable pattern.
- Exponentially Growing Combinations:
  - n hypervectors can generate:

$$2^n \quad \text{combinations via binding and superposition.}$$

Worked Example: Exponential Growth of Patterns

Consider $d = 10,000$ and binary vectors $\{0,1\}^d$:

- Total possible distinct vectors:

$$2^{10,000} \quad \text{(an astronomically large number).}$$

- Now allow for superposition and binding:
  - Superposition combines n vectors into a unique vector.
  - Binding generates entirely new patterns:

$$\mathbf{a} \circledast \mathbf{b} \quad \text{is unique for any } \mathbf{a}, \mathbf{b}.$$

- Result:
  With high-dimensional vectors, you can encode exponentially many relationships.

Key Benefits of Exponentially Large Spaces

- Robustness:
  - Small errors (noise) in the components of hypervectors do not significantly affect overall distinguishability.
- Scalability:
  - Exponentially large capacity ensures scalability for encoding large vocabularies, complex patterns, and relationships.
- Expressiveness:
  - Binding and superposition operations allow for compositional representations (e.g., hierarchical structures or analogies).
- Similarity Preservation:
  - High-dimensional vectors can preserve similarity in the space (e.g., similar words have closer embeddings).

Comparison: One-Hot Vectors vs. Hypervectors

- Dimensionality:
  - One-hot: $|V|$ (grows with vocabulary size).
  - Hypervectors: d (fixed, large dimensionality, e.g., $d = 10,000$).
- Orthogonality:
  - One-hot:
$$\mathbf{v}_i \cdot \mathbf{v}_j = 0 \quad \text{for } i \neq j.$$
  - Hypervectors:
$$\mathbf{v}_i \cdot \mathbf{v}_j \approx 0 \quad \text{(approximate for random vectors).}$$
- Representation Power:
  - One-hot: Only encodes identity.
  - Hypervectors: Encodes identity, similarity, and relationships.

Computational Distinctions

- Storage Requirements:
  - One-hot: Requires a vector of size $|V|$ for each token.
  - Hypervectors: Fixed size d, independent of vocabulary size.
- Operations:
  - One-hot: No meaningful operations (e.g., addition, multiplication).
  - Hypervectors: Supports binding, superposition, and correlation.

$$\mathbf{a} \circledast \mathbf{b}, \quad \mathbf{a} + \mathbf{b}, \quad \mathbf{a} \circledast \mathbf{b}^{-1}.$$

- Scalability:
  - One-hot: Becomes infeasible for large vocabularies ($|V| \gg 10^6$).
  - Hypervectors: Efficient for large vocabularies due to fixed dimensionality.

Advantages of Hypervector Encodings

- Compositionality:
  - Represent relationships through binding:

$$\mathbf{relation} = \mathbf{a} \circledast \mathbf{b}.$$

  - Combine multiple pieces of information:

$$\mathbf{context} = \mathbf{v}_1 + \mathbf{v}_2 + \mathbf{v}_3.$$

- Noise Tolerance:
  - Small changes to components do not disrupt overall encoding.
- Similarity Preservation:
  - Similar inputs produce similar hypervectors, enabling clustering and matching.
- Scalability:
  - Fixed-dimensional encoding handles large vocabularies and complex structures.

Comparison: One-Hot Vectors vs. Hypervectors

Feature	One-Hot Vectors	Hypervectors
Dimensionality	$\|V\|$ (vocab size)	Fixed d (e.g., $10,000$)
Orthogonality	Exact	Approximate
Representation	Identity only	Identity + relationships
Operations	None	Binding, superposition, correlation
Storage	Large for large $\|V\|$	Fixed-size
Scalability	Limited	High

Why Orthonormality is Important

- In high-dimensional spaces, random hypervectors are approximately orthonormal:

$$\mathbf{v} \cdot \mathbf{w} \approx 0, \quad \|\mathbf{v}\| = \|\mathbf{w}\| = 1.$$

- Key Implication: Vectors do not interfere with each other in superposition or binding.
  - Superposition: Combines multiple vectors while keeping them distinguishable.

$$\mathbf{s} = \mathbf{v}_1 + \mathbf{v}_2 + \cdots + \mathbf{v}_n$$

  - Binding: Combines vectors into unique encodings using circular convolution.

$$\mathbf{b} = \mathbf{v}_1 \circledast \mathbf{v}_2$$

Circular Convolution: Binding Vectors

Circular convolution is defined as:

$$(\mathbf{a} \circledast \mathbf{b})_i = \sum_{j=0}^{d-1} a_j \cdot b_{(i-j)} \mod d \cdot$$

Properties of Circular Convolution

- Dimensionality:

$$\mathbf{a} \circledast \mathbf{b} \in \mathbb{R}^d$$

- Uniqueness: Produces a distinct vector for each pair of inputs.
- Approximate Inverse:

$$\mathbf{a} \circledast \mathbf{b} \circledast \mathbf{b}^{-1} \approx \mathbf{a}$$

Correlation: Unbinding Vectors

Circular correlation retrieves one vector from a bound pair:

$$(\mathbf{a} \circledast \mathbf{b}) \circledast \mathbf{b}^{-1} \approx \mathbf{a}.$$

Definition of Circular Correlation

Circular correlation is defined as:

$$(\mathbf{c} \circledast \mathbf{b}^{-1})_i = \sum_{j=0}^{d-1} c_j \cdot b_{(j-i)} \mod d.$$

Key Insights

- Uses the approximate orthonormality of random vectors.
- Recovers the original vector when the bound pair is unbound.

Approximate Orthogonality in High Dimensions

For two random hypervectors $\mathbf{v}, \mathbf{w} \in \mathbb{R}^d$:

$$\mathbf{v} \cdot \mathbf{w} = \sum_{i=1}^{d} v_i w_i.$$

If $\mathbf{v}_i$ and $\mathbf{w}_i$ are independent and zero-mean:

- Expected value:
$$\mathbb{E}[\mathbf{v} \cdot \mathbf{w}] = 0.$$
- Variance decreases with dimensionality:
$$\mathrm{Var}[\mathbf{v} \cdot \mathbf{w}] = O\left(\frac{1}{d}\right).$$

For large d, the dot product is negligibly small:

$$\mathbf{v} \cdot \mathbf{w} \approx 0.$$

Implications for Encoding in NLP

- Superposition: Adding hypervectors preserves distinguishability:

$$\mathbf{s} = \mathbf{v}_1 + \mathbf{v}_2 \quad \Rightarrow \quad \mathbf{s} \cdot \mathbf{v}_1 \gg 0.$$

- Binding: Convolution produces unique encodings:

$$\mathbf{b} = \mathbf{v}_1 \circledast \mathbf{v}_2.$$

  Since $\mathbf{v}_1 \cdot \mathbf{v}_2 \approx 0$, the result is not confounded by interference.

- Unbinding: Correlation retrieves components reliably:

$$(\mathbf{b} \circledast \mathbf{v}_2^{-1}) \approx \mathbf{v}_1.$$

Example: Semantic Role Binding

Represent the sentence "The dog chased the ball":

- Words: **dog**, **chased**, **ball**.
- Roles: **subject**, **verb**, **object**.

Encoding:

$$\mathbf{S} = (\mathbf{dog} \circledast \mathbf{subject}) + (\mathbf{chased} \circledast \mathbf{verb}) + (\mathbf{ball} \circledast \mathbf{object}).$$

Retrieval:

- Retrieve the subject:

$$\mathbf{dog} \approx \mathbf{S} \circledast \mathbf{subject}^{-1}.$$

Example: Sequential Encoding

Encode "The dog sleeps":

- Words: **the**, **dog**, **sleeps**.
- Positional encoding:

$$\mathbf{sequence} = \mathbf{the} + \text{Perm}(\mathbf{dog}) + \text{Perm}^2(\mathbf{sleeps}).$$

Retrieval:

- Retrieve "dog" by reversing the permutation:

$$\mathbf{dog} \approx \mathbf{sequence} \circledast \text{Perm}^{-1}.$$

Vector Algebraic Composition Operations

Three primary operations are used in hyperdimensional computing:

- Superposition (Addition):

  $$\mathbf{c} = \mathbf{a} + \mathbf{b}$$

  Combines vectors while preserving their individual contributions.
- Binding (Multiplication or Convolution):

  $$\mathbf{c} = \mathbf{a} \circledast \mathbf{b}$$

  Creates a unique composite vector that is distinct from the inputs.
- Permutation:
  Random reordering of vector components to represent positional information.

Convolution as Binding

Circular Convolution

The binding operation in HRRs is defined as circular convolution:

$$c_i = \sum_{j=0}^{d-1} a_j \cdot b_{(i-j) \mod d}$$

Here:

- $\mathbf{a} = [a_0, a_1, \ldots, a_{d-1}]$
- $\mathbf{b} = [b_0, b_1, \ldots, b_{d-1}]$

- Properties:
  - Produces a vector of the same dimension d.
  - Distributes information of **a** and **b** across **c**.
  - Approximately invertible.

Worked Example: Circular Convolution

Given:

$$\mathbf{a} = [1, 2, 3], \quad \mathbf{b} = [0, 1, 0]$$

Compute:

$$c_0 = a_0 \cdot b_0 + a_1 \cdot b_2 + a_2 \cdot b_1 = 1 \cdot 0 + 2 \cdot 0 + 3 \cdot 1 = 3$$

$$c_1 = a_0 \cdot b_1 + a_1 \cdot b_0 + a_2 \cdot b_2 = 1 \cdot 1 + 2 \cdot 0 + 3 \cdot 0 = 1$$

$$c_2 = a_0 \cdot b_2 + a_1 \cdot b_1 + a_2 \cdot b_0 = 1 \cdot 0 + 2 \cdot 1 + 3 \cdot 0 = 2$$

Result:

$$\mathbf{c} = [3, 1, 2]$$

Deconvolution as Unbinding

Deconvolution

To retrieve **a** from **c** and **b**, perform circular correlation:

$$a_i = \sum_{j=0}^{d-1} c_j \cdot b_{(i-j) \mod d}$$

- Inverse Property:

$$\mathbf{a} \approx \mathbf{c} \circledast \mathbf{b}^{-1}$$

- Enables retrieval of the original components bound together.

Summary of HRR Operations

- Binding (Encoding):

$$\mathbf{binding} = \mathbf{w}_{\text{word}} \circledast \mathbf{r}_{\text{role}}$$

- Superposition:

$$\mathbf{S}_{\text{HRR}} = \mathbf{S} + \mathbf{V} + \mathbf{Neg} + \mathbf{O}$$

- Unbinding (Decoding):

$$\mathbf{w}_{\text{word}} \approx \mathbf{S}_{\text{HRR}} \circledast \mathbf{r}_{\text{role}}^{-1}$$

HRRs provide a robust framework for representing and manipulating structured information in high-dimensional spaces.

Comparison of Methods

A comparison of the three methods:

Feature	Transformer	Conventional	HRR
Context-Sensitivity	High	None	Moderate
Syntactic Structure	Implicit	Ignored	Explicit
Polysemy Handling	Excellent	Poor	Limited
Invertibility	No	No	Yes

Overview of Baroni & Zamparelli's Theory

- Nouns: Represented as dense vectors.

$$\mathbf{n} \in \mathbb{R}^d$$

- Adjectives: Represented as linear transformations (matrices).

$$\mathbf{A} \in \mathbb{R}^{d \times d}$$

- Modification: Apply the adjective to the noun using matrix-vector multiplication.

$$\text{Modified noun: } \mathbf{n}' = \mathbf{A} \cdot \mathbf{n}$$

Worked Example: Baroni & Zamparelli's Theory

Let:

- Noun: $\mathbf{n} = [1, 0, 1]^T$
- Adjective: $\mathbf{A} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix}$

Compute:

$$\mathbf{n}' = \mathbf{A} \cdot \mathbf{n} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

Interpretation:

- Adjective transforms the noun's semantic space.
- Matrix captures how adjectives modify meaning (e.g., "red" or "big").

Strengths and Limitations of Baroni & Zamparelli's Theory

- Strengths:
  - Captures compositional semantics via linear transformations.
  - Allows for a systematic representation of adjective effects.
- Limitations:
  - High parameter cost (d^2 parameters per adjective).
  - Limited interpretability of learned matrices.
  - Ignores distributed binding (no explicit roles or structure).

Overview of HRR Approach

- Nouns: Represented as high-dimensional hypervectors.

$$\mathbf{n} \in \mathbb{R}^d, \quad d \gg 1000$$

- Adjectives: Represented as hypervectors.

$$\mathbf{a} \in \mathbb{R}^d$$

- Binding: Adjectives bind to nouns using circular convolution.

$$\mathbf{n}' = \mathbf{a} \circledast \mathbf{n}$$

- Superposition: Combine multiple adjective-noun pairs.

$$\mathbf{S} = \mathbf{n}'_1 + \mathbf{n}'_2 + \cdots$$

Worked Example: HRR Approach

Let:

- Noun hypervector: $\mathbf{n} = [1, 0, 1]$
- Adjective hypervector: $\mathbf{a} = [0, 1, 0]$

Compute binding via circular convolution:

$$c_0 = a_0 \cdot n_0 + a_1 \cdot n_2 + a_2 \cdot n_1 = 0 \cdot 1 + 1 \cdot 1 + 0 \cdot 0 = 1$$

$$c_1 = a_0 \cdot n_1 + a_1 \cdot n_0 + a_2 \cdot n_2 = 0 \cdot 0 + 1 \cdot 1 + 0 \cdot 1 = 1$$

$$c_2 = a_0 \cdot n_2 + a_1 \cdot n_1 + a_2 \cdot n_0 = 0 \cdot 1 + 1 \cdot 0 + 0 \cdot 1 = 0$$

Resulting vector:

$$\mathbf{n}' = [1, 1, 0]$$

Comparison: Baroni & Zamparelli vs. HRR

Feature	Baroni & Zamparelli	HRR
Representation of Adjectives	Matrices ($d \times d$)	Hypervectors (d)
Composition Operation	Matrix-Vector Multiplication	Circular Convolution
Dimensionality	$O(d^2)$ (scaling issue)	Fixed (d)
Interpretability	Low	Moderate
Flexibility (e.g., roles)	Limited	High (binding and superposition)

The Role of Telic in Generative Lexicon

- The Telic role in GL Theory captures the purpose or function of an entity.

$$\text{Telic(pen)} = \text{write-with}, \quad \text{Telic(car)} = \text{drive}$$

- Adjectives modify nouns by binding to specific Qualia roles, including the Telic role.
- Disambiguation occurs when an adjective aligns with the Telic role of the noun.

Examples of Telic-Driven Disambiguation

- "Fast car" → Telic: drive (interpreted as speed when driving).
- "Good pen" → Telic: write-with (interpreted as quality in writing).
- "Loud speaker" → Telic: produce-sound (interpreted as volume of sound production).

Selective Binding with Telic Role

- Adjective-noun composition involves selective binding:

$$\text{Adjective} \circledast \text{Telic}(\text{Noun})$$

- This highlights the Telic role of the noun as the locus of modification.
- Formalization:

$$\mathbf{a} \circledast \mathbf{r}_{\text{Telic}} \circledast \mathbf{n}$$

Interpretation of Adjective Modification

The adjective binds to the Telic role, influencing how the noun is interpreted in context.

$$\text{"Fast car"} = \text{fast} \circledast \text{drive}(\text{car})$$

HRR Representation of Telic Role

- Nouns: Represented as hypervectors ($\mathbf{n}$).
- Adjectives: Represented as hypervectors ($\mathbf{a}$).
- Telic Role: Represented as a hypervector ($\mathbf{r}_{\text{Telic}}$).
- Binding: Use circular convolution to encode adjective modification of the Telic role.

$$\mathbf{composition} = \mathbf{a} \circledast \mathbf{r}_{\text{Telic}} \circledast \mathbf{n}$$

- Superposition: Combine multiple adjective-noun pairs for broader contexts.

Worked Example: "Fast Car"

Given:

- Noun: "car" ($\mathbf{n} = [1, 0, 1]$).
- Adjective: "fast" ($\mathbf{a} = [0, 1, 0]$).
- Telic Role: "drive" ($\mathbf{r}_{\text{Telic}} = [1, 1, 0]$).

Compute:

$$\mathbf{binding} = \mathbf{a} \circledast \mathbf{r}_{\text{Telic}} \circledast \mathbf{n}$$

Step 1 (Adjective-Telic binding):

$$c_0 = a_0 r_0 + a_1 r_2 + a_2 r_1 = 0 \cdot 1 + 1 \cdot 0 + 0 \cdot 1 = 0$$

Step 2 (Adjective-Telic-Noun binding):

$$c_0' = c_0 n_0 + c_1 n_2 + c_2 n_1 = \ldots$$

Result: $\mathbf{binding} = \ldots$

The final vector encodes "fast car" in terms of its Telic role, focusing on the meaning of "fast" in the context of driving.

Worked Example: "Good Pen"

Given:

- Noun: "pen" ($\mathbf{n} = [0, 1, 0]$).
- Adjective: "good" ($\mathbf{a} = [1, 0, 1]$).
- Telic Role: "write-with" ($\mathbf{r}_{\text{Telic}} = [0, 1, 1]$).

Compute:

$$\textbf{binding} = \mathbf{a} \circledast \mathbf{r}_{\text{Telic}} \circledast \mathbf{n}$$

Result:

$$\textbf{binding} = \dots$$

This vector captures the modification of "pen" by "good" with respect to the Telic role, emphasizing its quality in writing.

What is Type Coercion?

- Type Coercion occurs when a verb's argument requires a type mismatch to be resolved.
- The mismatched argument is coerced into the required type using its Qualia Structure.

Example: "Mary enjoyed a coffee"

- Verb: "enjoy" requires an event as its object.
- Noun: "a coffee" is an entity, not an event.
- Coercion: The Telic role of "coffee" provides the event "drink a coffee".

"Mary enjoyed a coffee" $\Rightarrow$ "Mary enjoyed drinking a coffee"

Type Coercion Using Qualia Structure

- The Qualia Structure of "coffee":
  - Formal: *beverage*.
  - Constitutive: *made-of-water*.
  - Telic: *drink*.
  - Agentive: *brewed*.
- The Telic role provides the required event for coercion:

$$\text{Telic}(\text{coffee}) = \text{drink}(\text{coffee})$$

Coercion Process

1. Verb identifies a type mismatch (*entity* vs. *event*). 2. Use the Qualia Structure to resolve the mismatch. 3. Bind the Telic role to the object and recompose:

$$\text{enjoy} \circledast \text{Telic}(\text{coffee}) = \text{enjoy}(\text{drink}(\text{coffee}))$$

HRR Representation of Coercion

- Nouns: Represented as hypervectors ($\mathbf{n}$).
- Telic Role: Represented as a hypervector ($\mathbf{r}_{\text{Telic}}$).
- Verb: Represented as a hypervector ($\mathbf{v}_{\text{enjoy}}$).
- Coercion: Bind the verb to the Telic role of the object:

$$\textbf{composition} = \mathbf{v}_{\text{enjoy}} \circledast \mathbf{r}_{\text{Telic}} \circledast \mathbf{n}_{\text{coffee}}$$

Worked Example: "Mary Enjoyed a Coffee"

Given:

- Noun hypervector: $\mathbf{n}_{\text{coffee}} = [1, 0, 1]$.
- Telic Role: $\mathbf{r}_{\text{Telic}} = [0, 1, 0]$.
- Verb: $\mathbf{v}_{\text{enjoy}} = [1, 1, 0]$.

Compute coercion:

$$\textbf{composition} = \mathbf{v}_{\text{enjoy}} \circledast \mathbf{r}_{\text{Telic}} \circledast \mathbf{n}_{\text{coffee}}$$

Step 1 (Verb-Telic binding):

$$c_0 = v_0 r_0 + v_1 r_2 + v_2 r_1 = 1 \cdot 0 + 1 \cdot 0 + 0 \cdot 1 = 0$$

Step 2 (Verb-Telic-Noun binding):

$$c_0' = c_0 n_0 + c_1 n_2 + c_2 n_1 = \dots$$

Result:

$$\textbf{composition} = [\dots]$$

This vector encodes the coerced meaning "enjoy drinking a coffee."

Denotational Semantics and Type-Theoretic Interpretation

- $[\![\text{expression}]\!] \to$ mathematical structure
- Examples:
  - Words: $[\![\text{dog}]\!] =$ vector in $\mathbb{R}^d$.
  - Sentences: $[\![\text{"Dogs bark"}]\!] =$ truth value $\{0, 1\}$.
- In vector-based semantics:
  - Nouns: $[\![N]\!] : \mathbb{R}^d$
  - Adjectives: $[\![\text{Adj}]\!] : \mathbb{R}^d \to \mathbb{R}^d$
  - Verbs: $[\![V]\!] : \mathbb{R}^d \to \mathbb{R}^d$
- Sentence composition:

$$[\![\text{"Dogs bark"}]\!] = \text{function application or combination of vectors.}$$

Conventional Vector Composition

- Words are vectors: $[\![w]\!] \in \mathbb{R}^d$.
- Composition is performed using vector addition or pointwise multiplication.

Denotational Semantics

Let $w_1, w_2 \in \mathbb{R}^d$, then:

$$[\![\text{"fast car"}]\!] = \mathbf{v}_{\text{fast}} + \mathbf{v}_{\text{car}}.$$

Type-Theoretic Interpretation

- Nouns: $e : \mathbb{R}^d$
- Adjectives: $e \rightarrow e : \mathbb{R}^d \rightarrow \mathbb{R}^d$

Composition:

$$\text{Adj(N)} : \mathbb{R}^d \rightarrow \mathbb{R}^d$$

$$[\![\text{"fast"}]\!]([\![\text{"car"}]\!]) = \mathbf{v}_{\text{fast}} + \mathbf{v}_{\text{car}}.$$

Baroni and Zamparelli: Adjectives as Matrices

- Words are assigned different types:
  - Nouns: $\mathbb{R}^d$
  - Adjectives: $\mathbb{R}^d \to \mathbb{R}^d$ (matrices)
- Composition uses matrix-vector multiplication.

Denotational Semantics

Let:

- $\mathbf{v}_{\text{car}} \in \mathbb{R}^d$
- $\mathbf{M}_{\text{fast}} \in \mathbb{R}^{d \times d}$

Then:

$$[\![\text{"fast car"}]\!] = \mathbf{M}_{\text{fast}} \cdot \mathbf{v}_{\text{car}}.$$

Baroni and Zamparelli: Adjectives as Matrices

Type-Theoretic Interpretation

- Nouns: $e : \mathbb{R}^d$
- Adjectives: $e \to e : \mathbb{R}^d \to \mathbb{R}^d$ (linear maps)

Composition:

$$\text{Adj(N)} : \mathbb{R}^d \to \mathbb{R}^d$$

$$[\![\text{"fast"}]\!]([\![\text{"car"}]\!]) = \mathbf{M}_{\text{fast}} \cdot \mathbf{v}_{\text{car}}.$$

HRR: Binding with Circular Convolution

- Words and roles are hypervectors ($\mathbb{R}^d$).
- Binding is performed using circular convolution.

Denotational Semantics

Let:

- $\mathbf{v}_{\text{car}}, \mathbf{v}_{\text{fast}} \in \mathbb{R}^d$

Then: $[\![\text{"fast car"}]\!] = \mathbf{v}_{\text{fast}} \circledast \mathbf{v}_{\text{car}}$, where:

$$(\mathbf{v}_{\text{fast}} \circledast \mathbf{v}_{\text{car}})_i = \sum_{j=0}^{d-1} v_{\text{fast},j} \cdot v_{\text{car},(i-j) \mod d}.$$

HRR: Binding with Circular Convolution

Type-Theoretic Interpretation

- Nouns: $e : \mathbb{R}^d$
- Adjectives: $e \to e : \mathbb{R}^d \to \mathbb{R}^d$ (convolution operators)

Composition:

$$\text{Adj(N)} : \mathbb{R}^d \to \mathbb{R}^d$$

$$[\![\text{"fast"}]\!]([\![\text{"car"}]\!]) = \mathbf{v}_{\text{fast}} \circledast \mathbf{v}_{\text{car}}.$$