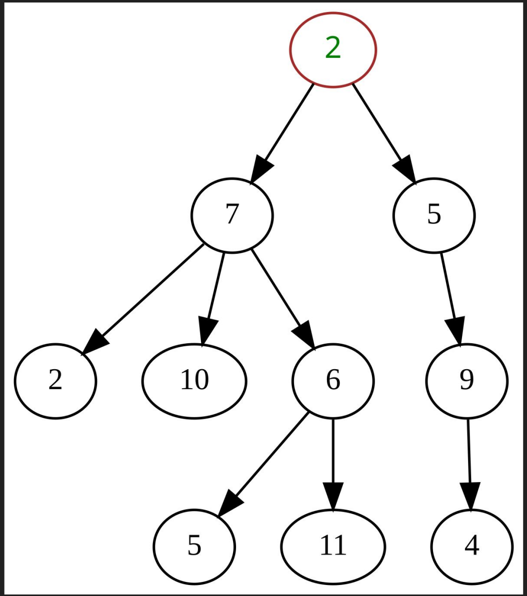
Hyperbolic Restricted Boltzmann Machines for Hierarchical Learning

Kylie Hefner
Chapman University: Computational and Data Sciences
CS 595: Computational Science Seminars

Overview

- Problem:
  - Traditional RBMs use Euclidean geometry, which distorts hierarchical/tree-like data.
  - Key Gap: Can hyperbolic geometry improve RBMs for hierarchical structures?
- Solution:
  - A **Hyperbolic RBM** with adjusted energy function and sampling.
- Impact:
  - Better unsupervised learning for NLP, biology, or knowledge graphs.

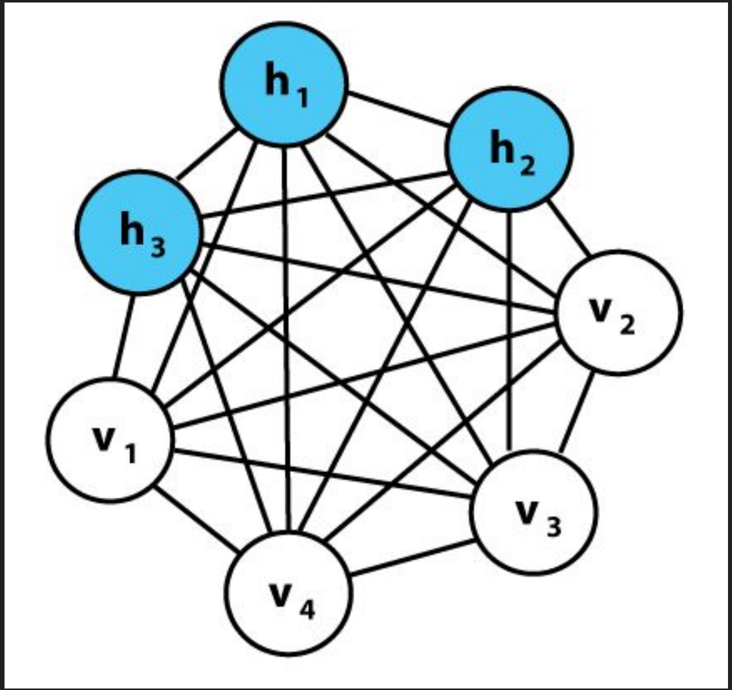


Literature Review

- Masaki Kobayashi
  - Researches the use of non-Euclidean geometry in Hopfield Neural Networks (complex, hyperbolic, quaternion, rotor, etc.)
- John J. Hopfield
  - Invented the Hopfield Neural Network (1982)
  - 2024 Nobel Prize in Physics
- Geoffrey Hinton
  - Invented the Boltzmann Machine (1983-1985)
  - 2024 Nobel Prize in Physics

What is a Boltzmann Machine

- Stochastic, generative neural networks that learn patterns by minimizing an energy function
- Inspired by statistical mechanics (Boltzmann distribution)
- Why they matter:
  - Foundation for energy-based models, latent variable learning, and modern generative AI.

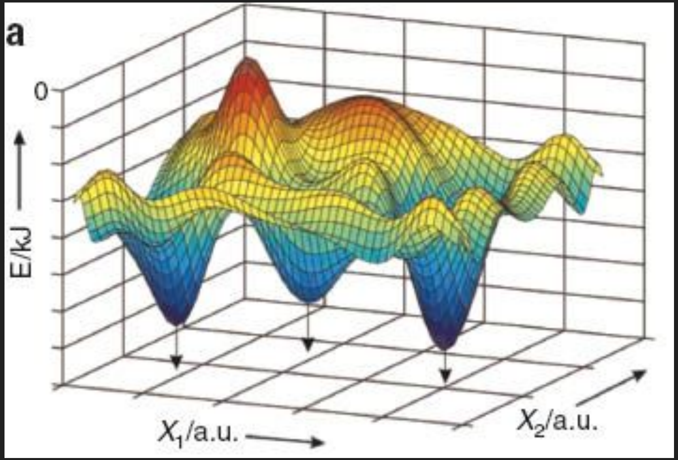


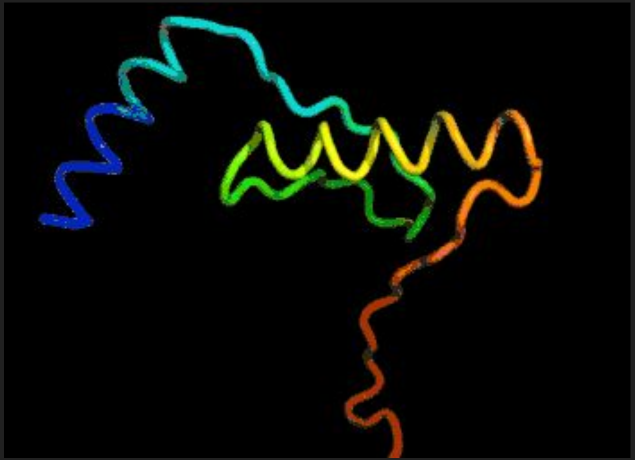
Energy Computation

$$E(\mathbf{v}, \mathbf{h}) = -\mathbf{v}^T W \mathbf{h} - \mathbf{a}^T \mathbf{v} - \mathbf{b}^T \mathbf{h}$$

Measures the "goodness" of a configuration (lower energy = higher probability)

Learning involves adjusting weights to reduce energy for training data



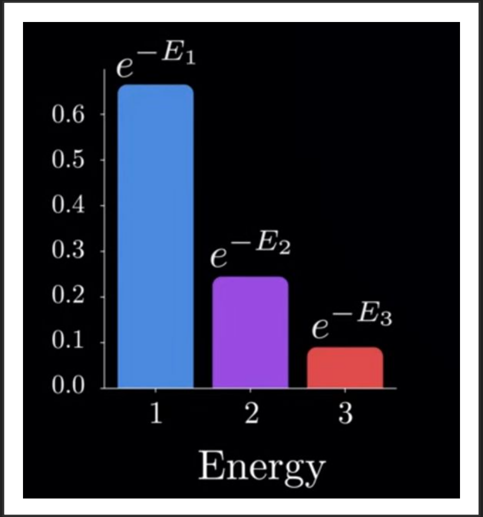


Energy Computation

- The energy function is used to compute the **probability distribution** over all possible configurations of the units.
- Specifically, the probability of a configuration is given by the Boltzmann distribution:

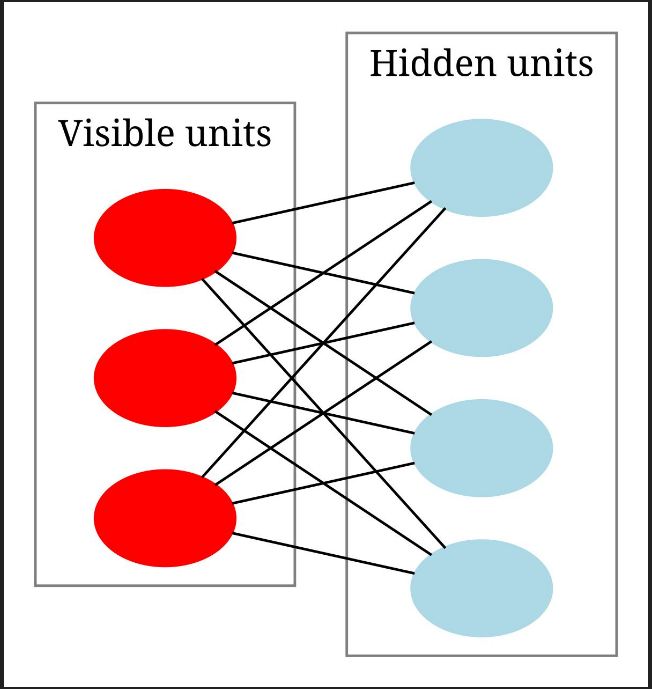
$$p(\mathbf{v}, \mathbf{h}) = \frac{\exp(-E(\mathbf{v}, \mathbf{h}))}{Z}$$

(*Z* is a normalizing function, the sum of all possible state probabilities)



Restricted Boltzmann Machines

- Structure:
  - A simplified Boltzmann Machine with restricted architecture
  - No intra-layer connections: only visible-to-hidden links
- Training:
  - Contrastive Divergence (CD-k): Approximate gradient using Gibbs sampling (fast, but biased)



Boltzmann Machine Simple Example

Let's say you have a dataset of movie preferences for three people. Each person rates whether they like (1) or dislike (0) two movies.

The dataset consists of the following vectors:

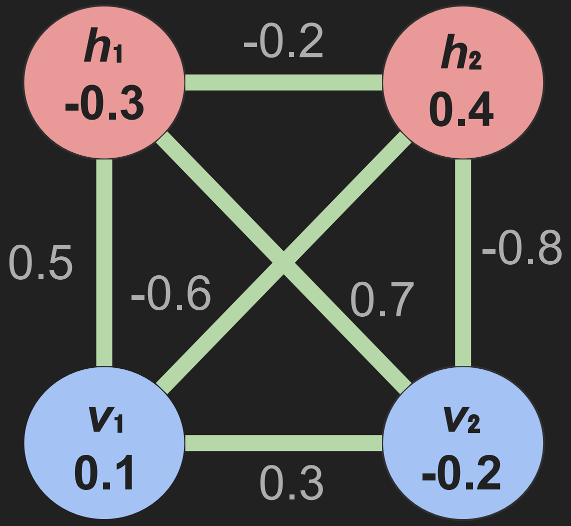
- $x_1 = [1, 0]$ (Person 1 likes Movie A and dislikes Movie B)
- $x_2 = [0, 1]$ (Person 2 dislikes Movie A and likes Movie B)
- $x_3 = [1, 1]$ (Person 3 likes Movie A and Movie B)

Our goal is to train a Boltzmann Machine to learn the underlying patterns in this dataset.

Boltzmann Machine Simple Example

We create the architecture to represent this data:

- Visible units represent the movie preferences (movie A/B)
- Hidden units capture latent (unmeasured) features
- Weights connect units (pos/neg relationship between units)
- Visible and hidden units have biases that measure individual probability of being on or off

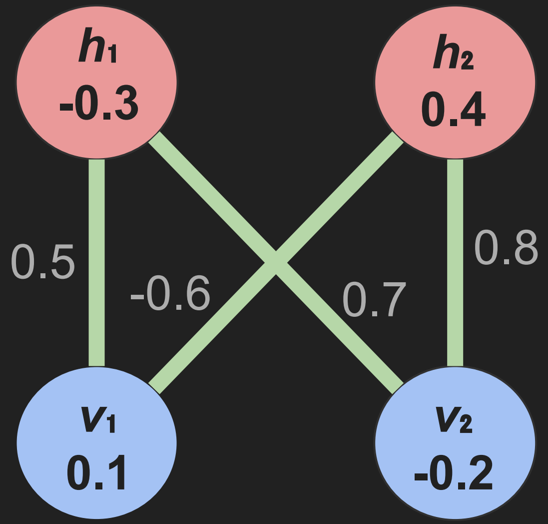


Initial values are chosen randomly or from domain knowledge

Restricted Boltzmann Machine Simple Example

We create the architecture to represent this data:

- Visible units represent the movie preferences (movie A/B)
- Hidden units capture latent (unmeasured) features
- Weights connect units (pos/neg relationship between units)
- Visible and hidden units have biases that measure individual probability of being on or off



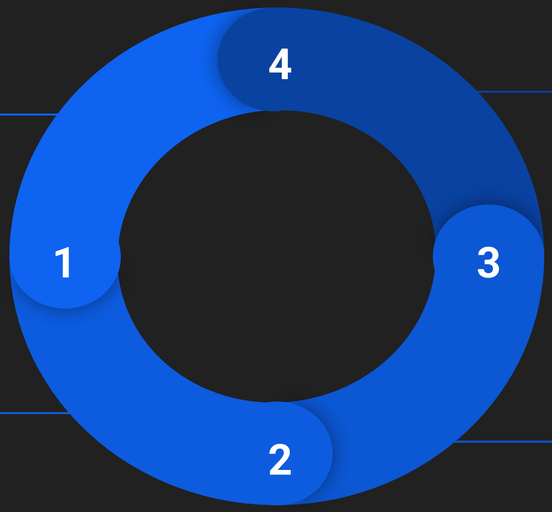
Initial values are chosen randomly or from domain knowledge

Training RBMs: Steps

0. Initialize weights and biases

1. Forward Pass (Visible → Hidden)

2. Reconstruct Input (Hidden → Visible)



4. Update weights and biases

3. Forward Pass Again (Reconstructed Visible → Hidden)

5. Repeat steps 1-4 for all data points and multiple epochs until convergence

Training RBMs: Approximation

Theoretical:

- Goal: Maximize the likelihood of the training data under the RBM's energy-based model
- Problem: Requires the partition function Z, which sums up all possible configurations of the model. Too many calculations for large models.

Contrastive Divergence:

- Solution: Approximate the gradient by running Gibbs Sampling and stopping before equilibrium
- Why it works: Just a few steps of Gibbs tells us the direction of gradient needed to reduce energy.

Step 1: Forward Pass

Sampling is used to infer hidden states given visible states

$$p(h_j = 1|\mathbf{v}) = \sigma\left(b_j + \sum_i v_i w_{ij}\right)$$

$$\sigma(x) = \frac{1}{1+e^{-x}}$$

Sampling for v = [1, 0] (person 1):

h1: $\sigma(-0.3 + 1 \cdot 0.5 + 0 \cdot 0.7) = \sigma(0.2) \approx 0.5498$. Sampled $h_1 = 1$.

h2: $\sigma(0.4 + 1 \cdot (-0.6) + 0 \cdot (-0.8)) = \sigma(-0.2) \approx 0.4502$. Sampled $h_2 = 0$.

Therefore, our hidden activation is h = [1, 0] for this visible state.

Step 2: Reconstruct Input

Now, we sample visible nodes (input) using *h* = [1, 0] from previous step:

$$P(v_i = 1|h) = \sigma(a_i + \textstyle\sum_j h_j W_{ij})$$

v1: $\sigma(0.1 + 1 \cdot 0.5 + 0 \cdot (-0.6)) = \sigma(0.6) \approx 0.6457$. Sampled $v_1' = 1$.

v2: $\sigma(-0.2 + 1 \cdot 0.7 + 0 \cdot (-0.8)) = \sigma(0.5) \approx 0.6225$. Sampled $v_2' = 1$.

Therefore, our reconstructed input is *v'* = [1, 1].

Step 3: Forward Pass (again)

$$p(h_j = 1|\mathbf{v}) = \sigma\left(b_j + \sum_i v_i w_{ij}\right)$$

$$\sigma(x) = \frac{1}{1+e^{-x}}$$

Sampling for *v'* = [1, 1]:

h1': $\sigma(-0.3 + 1 \cdot 0.5 + 1 \cdot 0.7) = \sigma(0.9) \approx 0.7109$. Sampled $h'_1 = 1$.

h2': $\sigma(0.4 + 1 \cdot (-0.6) + 1 \cdot (-0.8)) = \sigma(-1.0) \approx 0.2689$. Sampled $h'_2 = 0$.

Therefore, our reconstructed hidden activation is *h'* = [1, 0].

Step 4: Update Parameters

Goal: Adjust weights and biases to minimize difference between *v* and *v'*

$$\Delta W = \epsilon \left(\mathbf{v}^T\mathbf{h} - \mathbf{v}'^T\mathbf{h}'\right)$$

$$= \epsilon \left(\begin{bmatrix} 1 \\ 0 \end{bmatrix} \begin{bmatrix} 1 & 0 \end{bmatrix} - \begin{bmatrix} 1 \\ 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \end{bmatrix} \right)$$

$$= 0.1 \left(\begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} - \begin{bmatrix} 1 & 0 \\ 1 & 0 \end{bmatrix} \right)$$

$$= \begin{bmatrix} 0 & 0 \\ -0.1 & 0 \end{bmatrix}$$

$$\Delta \mathbf{a} = \epsilon(\mathbf{v} - \mathbf{v}')$$

$$= 0.1 \left(\begin{bmatrix} 1 & 0 \end{bmatrix} - \begin{bmatrix} 1 & 1 \end{bmatrix} \right)$$

$$= \begin{bmatrix} 0 & -0.1 \end{bmatrix}$$

$$\Delta \mathbf{b} = \epsilon(\mathbf{h} - \mathbf{h}')$$

$$= 0.1 \left(\begin{bmatrix} 1 & 0 \end{bmatrix} - \begin{bmatrix} 1 & 0 \end{bmatrix} \right)$$

$$= \begin{bmatrix} 0 & 0 \end{bmatrix}$$

Step 4: Update Parameters

Now, we can update the original parameters

$$W_{\text{new}} = W + \Delta W = \begin{bmatrix} 0.5 & -0.6 \\ 0.7 & -0.8 \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ -0.1 & 0 \end{bmatrix} = \begin{bmatrix} 0.5 & -0.6 \\ 0.6 & -0.8 \end{bmatrix}$$

$$\mathbf{a}_{\text{new}} = \mathbf{a} + \Delta \mathbf{a} = \begin{bmatrix} 0.1 & -0.2 \end{bmatrix} + \begin{bmatrix} 0 & -0.1 \end{bmatrix} = \begin{bmatrix} 0.1 & -0.3 \end{bmatrix}$$

$$\mathbf{b}_{\text{new}} = \mathbf{b} + \Delta \mathbf{b} = \begin{bmatrix} -0.3 & 0.4 \end{bmatrix} + \begin{bmatrix} 0 & 0 \end{bmatrix} = \begin{bmatrix} -0.3 & 0.4 \end{bmatrix} \quad \text{(no change)}$$

Step 5: Repeat steps 1-4 for all data points and multiple epochs until convergence

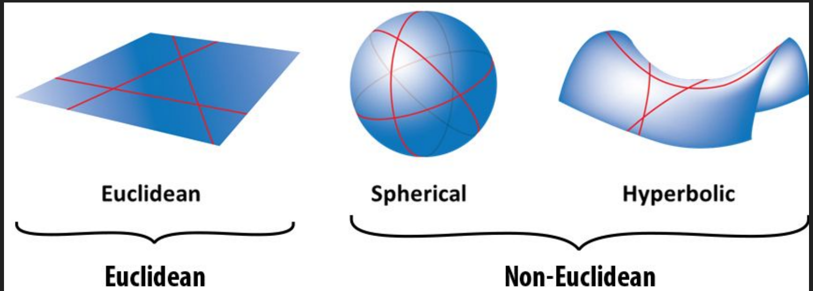
Legacy in Modern Machine Learning

- Deep Belief Networks (DBNs):
  - Stacked RBMs → Unsupervised pre-training → Fine-tuning with backprop
  - Pioneered the "deep learning revolution" (2006–2012)
- Connections to Modern Models:
  - Energy-based models (e.g., VAEs, diffusion models)
  - Stochastic units in GANs/VAEs vs. BM’s Gibbs sampling
- Why They Faded:
  - Computationally expensive vs. backpropagation-friendly architectures (CNNs, Transformers)

Key Takeaway: Understanding Boltzmann Machines reveals the DNA of modern generative AI

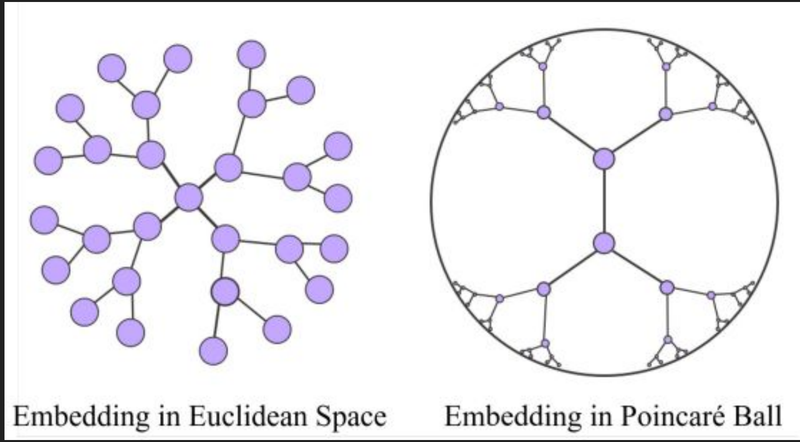
A Primer on Hyperbolic Geometry

- A non-Euclidean geometry with constant negative curvature (like a saddle or pringle shape).
- Contrast with:
  - Euclidean (flat) space: Zero curvature.
  - Spherical geometry: Positive curvature (like a sphere).
- In hyperbolic space, distances grow exponentially as you move outward.



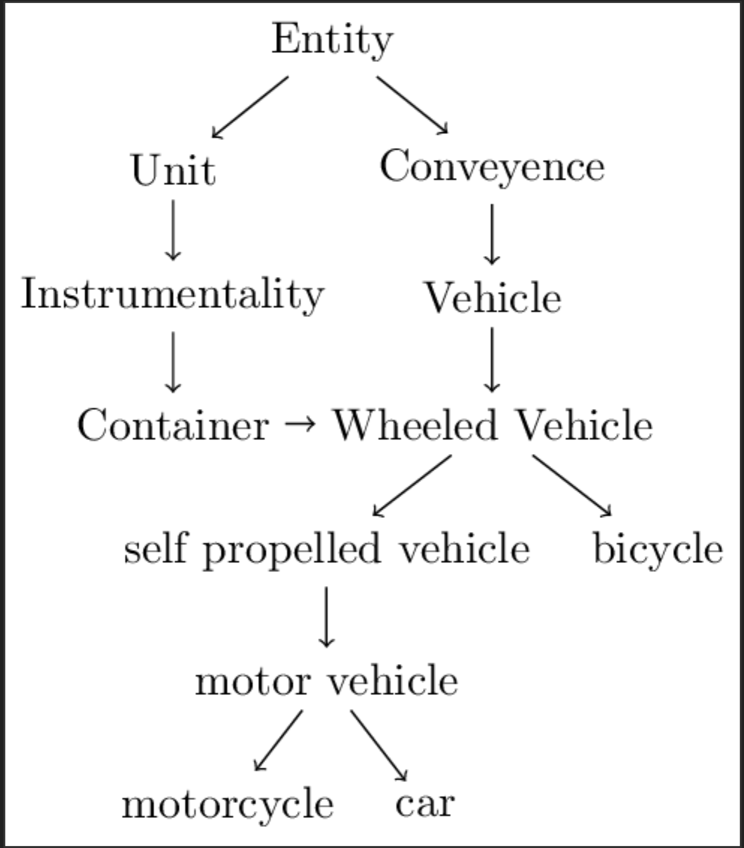
Hyperbolic Models: A Tool for Hierarchical Data

- Represent data points in hyperbolic space (e.g., Poincaré ball)
- Distance between points: $d(x,y) = \operatorname{arcosh}\left(1 + 2\frac{\|x-y\|^2}{(1-\|x\|^2)(1-\|y\|^2)}\right)$
- Efficient representation of hierarchies (fewer dimensions, lower distortion).



WordNet Hierarchy

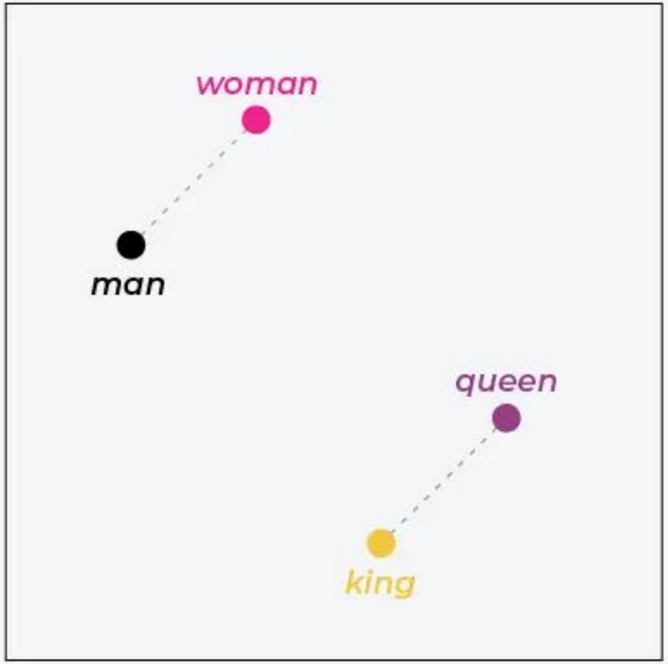
- A lexical database where nouns are organized in a hierarchical taxonomy (e.g., "animal" → "mammal" → "dog").
- Tree-like structure with clear parent-child relationships.
- Access: nltk.corpus.wordnet



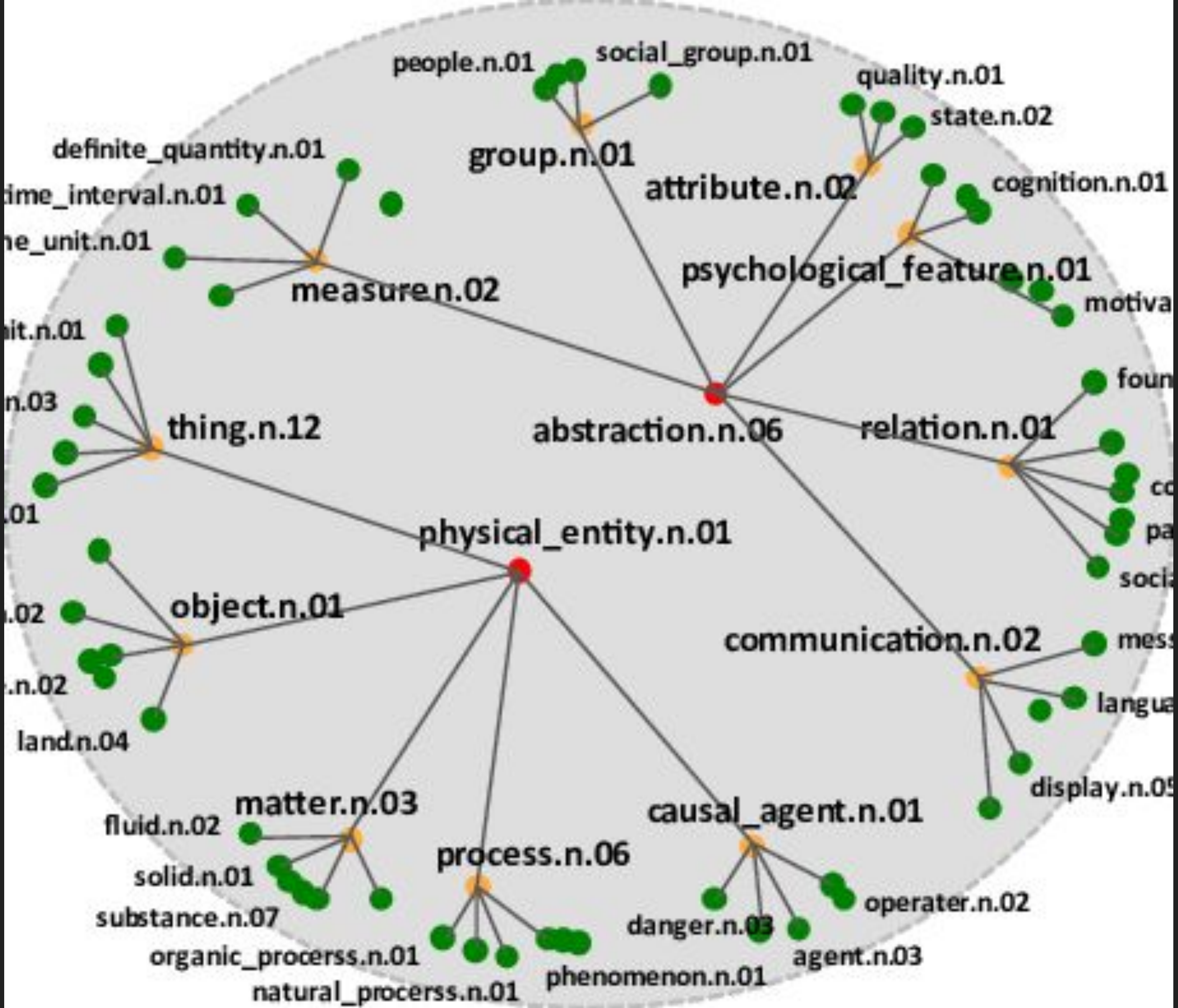
word	living being	feline	human	gender	royalty	verb	plural
man	0.6	-0.2	0.8	0.9	-0.1	-0.9	-0.7
woman	0.7	0.3	0.8	-0.7	0.1	-0.5	-0.4
king	0.5	-0.4	0.7	0.8	0.9	-0.7	-0.6
queen	0.8	-0.1	0.8	-0.9	0.8	-0.5	-0.9

word

Word embedding



Visualization of word embedding


people.n.01
social_group.n.01
quality.n.01
state.n.02
group.n.01
definite_quantity.n.01
attribute.n.02
cognition.n.01
ime_interval.n.01
ne_unit.n.01
psychological_feature.n.01
measure.n.02
motiva
it.n.01
foun
n.03
thing.n.12
abstraction.n.06
relation.n.01
co
.01
pa
physical_entity.n.01
socia
object.n.01
.n.02
communication.n.02
mess
.n.02
langua
land.n.04
display.n.05
matter.n.03
causal_agent.n.01
fluid.n.02
process.n.06
solid.n.01
operater.n.02
substance.n.07
danger.n.03
organic_procerss.n.01
agent.n.03
phenomenon.n.01
natural_procerss.n.01

Model Design: Energy Function

Euclidean RBM:

$$E(\mathbf{v}, \mathbf{h}) = -\mathbf{v}^T W \mathbf{h} - \mathbf{a}^T \mathbf{v} - \mathbf{b}^T \mathbf{h}$$

- $\mathbf{v}$: Visible units.
- $\mathbf{h}$: Hidden units.
- W: Weight matrix.
- $\mathbf{a}, \mathbf{b}$: Bias vectors.

Hyperbolic RBM:

$$E(\mathbf{v}, \mathbf{h}) = -\beta \cdot d_{\mathbb{H}}(\mathbf{v}, W \otimes \mathbf{h})^2 - \mathbf{a}^T \otimes \mathbf{v} - \mathbf{b}^T \otimes \mathbf{h}$$

- $d_{\mathbb{H}}$: Hyperbolic distance in the Poincaré ball.
- $\otimes$: Lorentzian inner product (analog of dot product).
- β: Scaling factor for curvature.

Model Design: Sampling Methods

Euclidean RBM:

- Gibbs Sampling: Sample $h|v$ and $v|h$ using logistic activation
- Markov chain typically converges quickly

$$p(h_j = 1|\mathbf{v}) = \sigma\left(b_j + \sum_i v_i w_{ij}\right)$$

$$p(v_i = 1|\mathbf{h}) = \sigma\left(a_i + \sum_j h_j w_{ij}\right)$$

Hyperbolic RBM:

- Gibbs Sampling: Sample h|v and v|h using hyperbolic probabilities
- Markov chain may take longer to converge due to curvature

$$p(h_j = 1|\mathbf{v}) = \sigma\left(\mathrm{logit}_{\mathbb{H}}(b_j \oplus \sum_i v_i \otimes w_{ij})\right)$$

$$p(v_i = 1|\mathbf{h}) = \sigma\left(\mathrm{logit}_{\mathbb{H}}(a_i \oplus \sum_j h_j \otimes w_{ij})\right)$$

- $\oplus$: Möbius addition.
- $\mathrm{logit}_{\mathbb{H}}$: Hyperbolic logit function.

Model Design: Training

Euclidean RBM:

- Update weights and biases using gradient ascent:

$$\Delta w_{ij} = \epsilon \left(\langle v_i h_j \rangle_{\text{data}} - \langle v_i h_j \rangle_{\text{model}} \right)$$

ϵ: Learning rate.

- Key Properties:
  - Gradients are computed using standard backpropagation.
  - Optimization is stable and efficient.

Hyperbolic RBM:

- Update weights and biases using Riemannian gradient ascent:

$$\Delta w_{ij} = \epsilon \cdot \mathrm{proj}_{\mathbb{H}} \left(\langle v_i h_j \rangle_{\text{data}} - \langle v_i h_j \rangle_{\text{model}} \right)$$

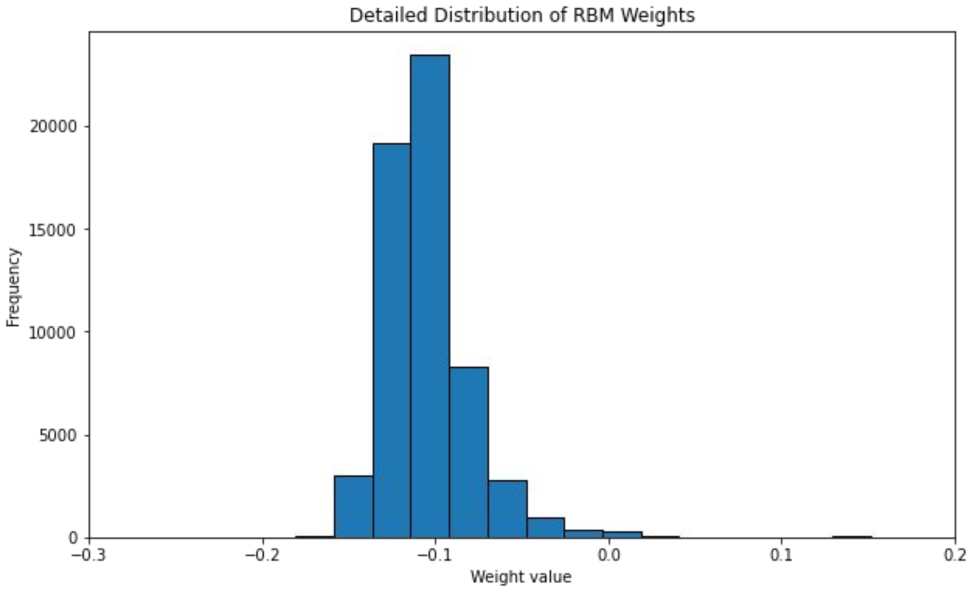
$\mathrm{proj}_{\mathbb{H}}$: Projection onto the hyperbolic manifold.

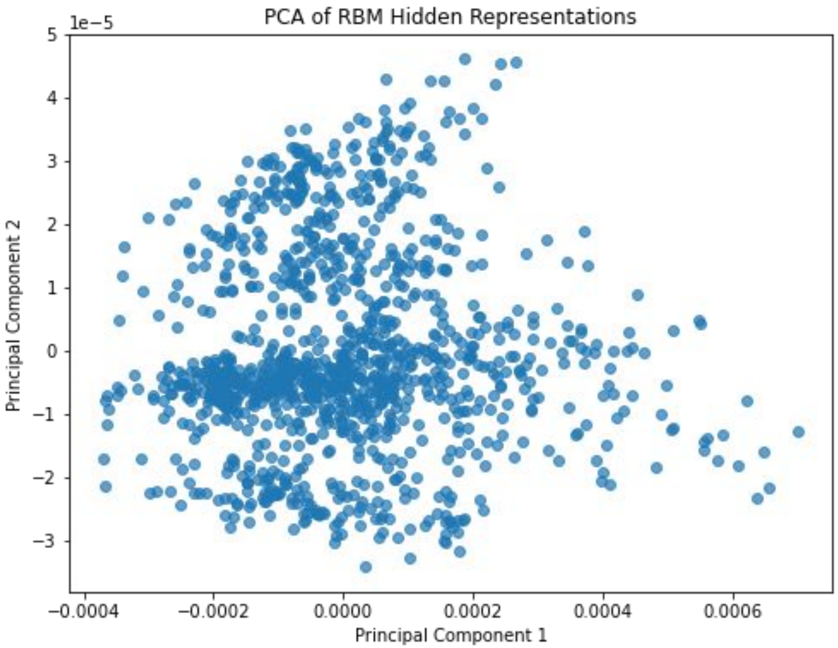
- Key Properties:
  - Gradients must respect the hyperbolic geometry (e.g., Riemannian optimization).
  - Optimization is more challenging due to curvature and numerical instability.

Implementation

Jupyter notebook, nltk wordnet, numpy, sklearn (for BernoulliRBM), matplotlib, networkx (for visualization)

I ran both models on the 1.2k synset “mammal” tree





Results

<u>Euclidean</u>

50 hidden layers, 50 epochs

```
Chosen Sample Node:
pug.n.01

Original active synsets:
['canine.n.02', 'carnivore.n.01', 'dog.n.01', 'mammal.n.01', 'placental.n.0
1', 'pug.n.01']

Reconstructed active synsets:
['ape.n.01', 'equine.n.01', 'even-toed_ungulate.n.01', 'horse.n.01', 'mamma
l.n.01', 'odd-toed_ungulate.n.01', 'placental.n.01']
```

```
Reconstruction error (mean absolute difference): 0.0077
```

```
Precision=0.286, Recall=0.333, F1=0.308
```

<u>Hyperbolic</u>

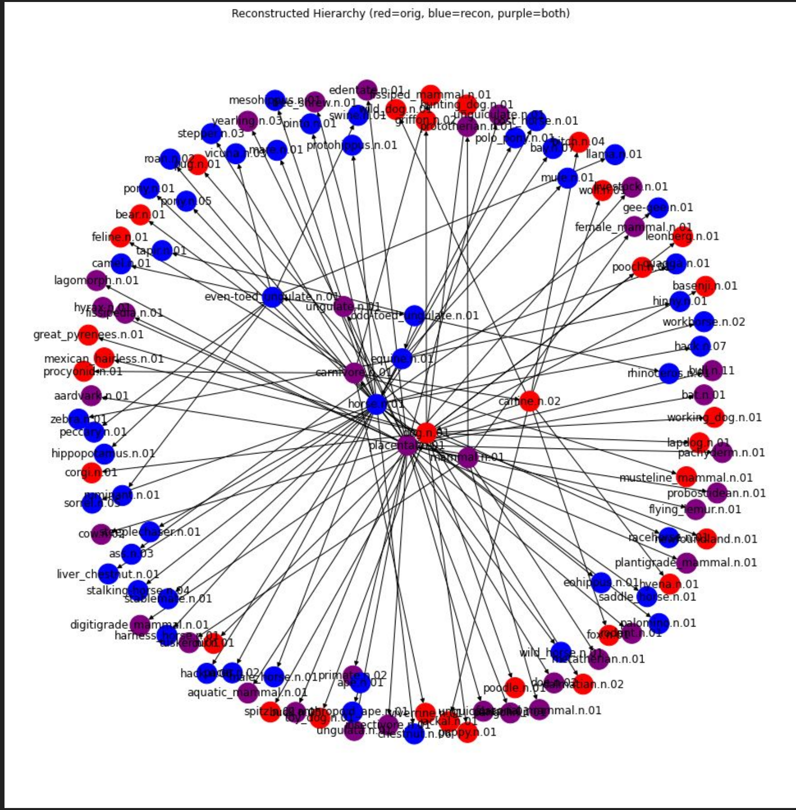
50 hidden layers, 3 epochs (~60 min)

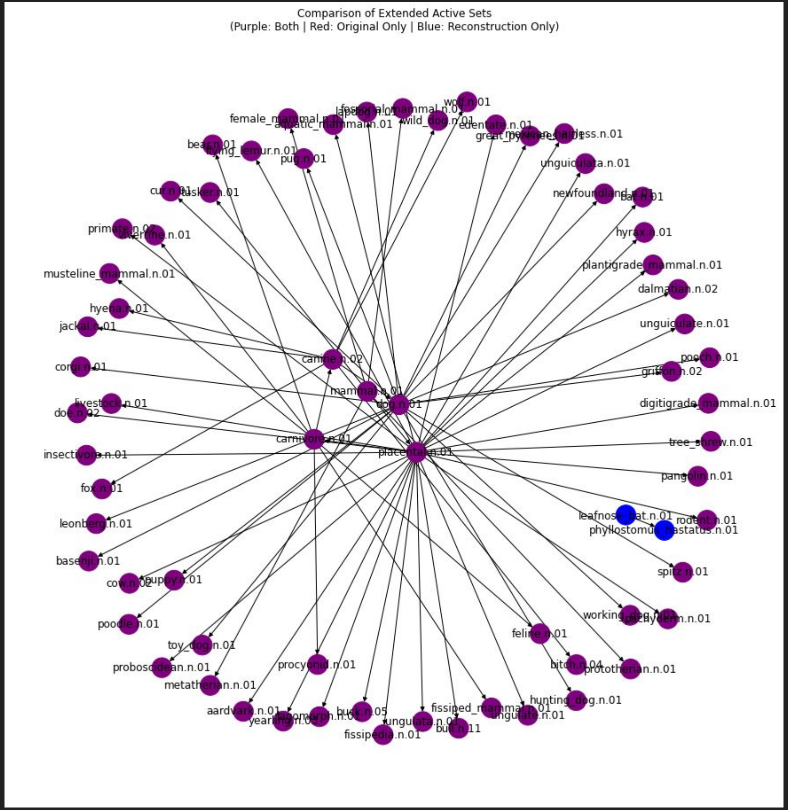
```
Chosen Sample Node:
pug.n.01

Original active synsets:
['canine.n.02', 'carnivore.n.01', 'dog.n.01', 'mammal.n.01', 'placental.n.0
1', 'pug.n.01']

Reconstructed active synsets:
['canine.n.02', 'carnivore.n.01', 'dog.n.01', 'mammal.n.01', 'phyllostomus_
hastatus.n.01', 'placental.n.01']
```

```
Reconstruction error (mean absolute difference): 0.0017
```

```
Precision=0.833, Recall=0.833, F1=0.833
```

Reconstruction Graphs (2 layers deep)





Results

Now reconstructing ‘dog.n.01’

Euclidean

```
Chosen Sample Node:
dog.n.01

Original active synsets:
['canine.n.02', 'carnivore.n.01', 'dog.n.01', 'mammal.n.01', 'placental.n.0
1']

Reconstructed active synsets:
['bovid.n.01', 'dog.n.01', 'mammal.n.01', 'odd-toed_ungulate.n.01', 'placen
tal.n.01', 'primate.n.02', 'rabbit-eared_bandicoot.n.01']
```

```
Reconstruction error (mean absolute difference): 0.0051
```

```
Precision=0.429, Recall=0.600, F1=0.500
```

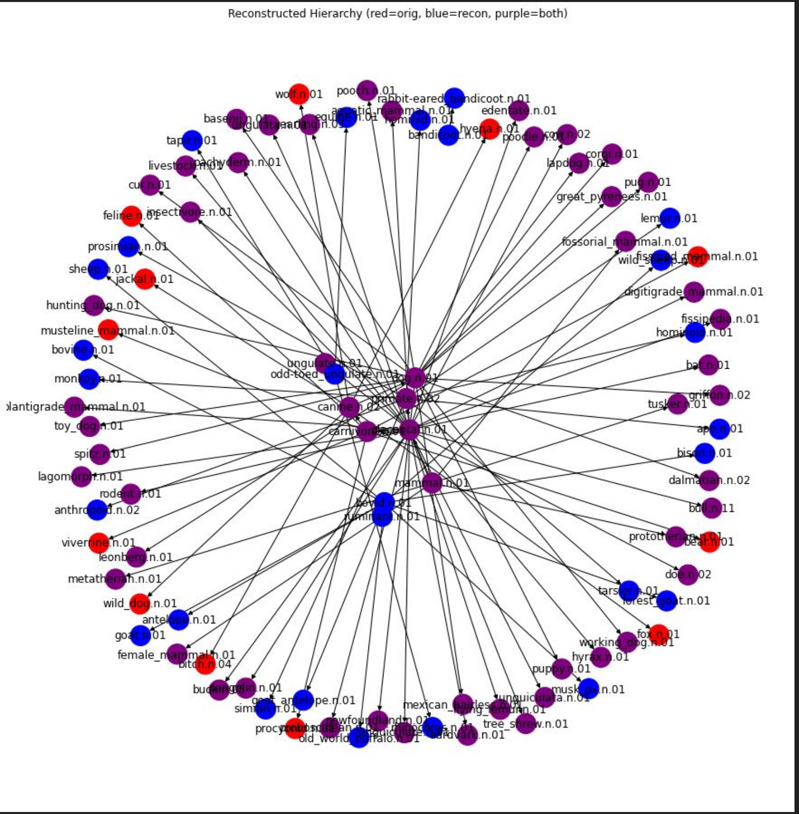
Hyperbolic

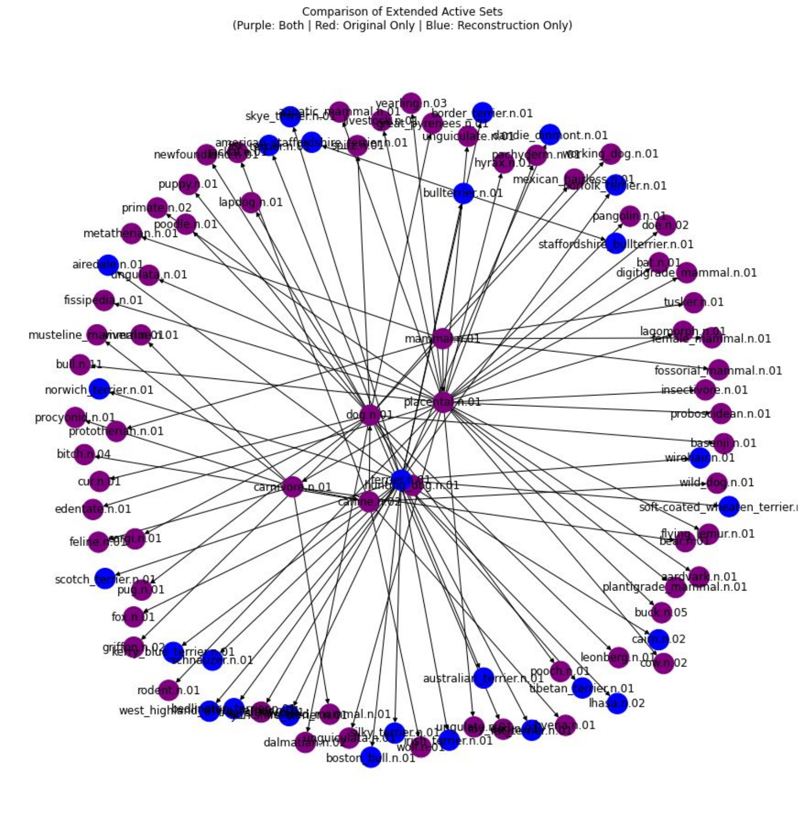
```
Chosen Sample Node:
dog.n.01

Original active synsets:
['canine.n.02', 'carnivore.n.01', 'dog.n.01', 'mammal.n.01', 'placental.n.0
1']

Reconstructed active synsets:
['canine.n.02', 'carnivore.n.01', 'dog.n.01', 'hunting_dog.n.01', 'mammal.
n.01', 'placental.n.01']
```

```
Reconstruction error (mean absolute difference): 0.0009
```

```
Precision=0.833, Recall=1.000, F1=0.909
```

Reconstruction Graphs (2 layers deep)





Challenges and Future Work

- Training Time
  - Hyperbolic arithmetic is more computationally intensive than standard float operations.
  - Future work: explore optimized implementations or approximations -- rewriting the model using PyTorch would enable GPU acceleration
- Future models could treat curvature as a hyperparameter or optimize it jointly with weights.
- A larger evaluation—e.g., across full WordNet, biomedical ontologies, or hierarchical image taxonomies—would provide stronger evidence of HRBM effectiveness.