Word Sense Disambiguation (WSD)

About WSD

Word Sense Disambiguation (WSD) is the task where one has to find the correct context or meaning of the word in a sentence , like a sentence can have more than one same spelling words but they are representing different meaning which is easily understood by a layman but not by machines. For example:

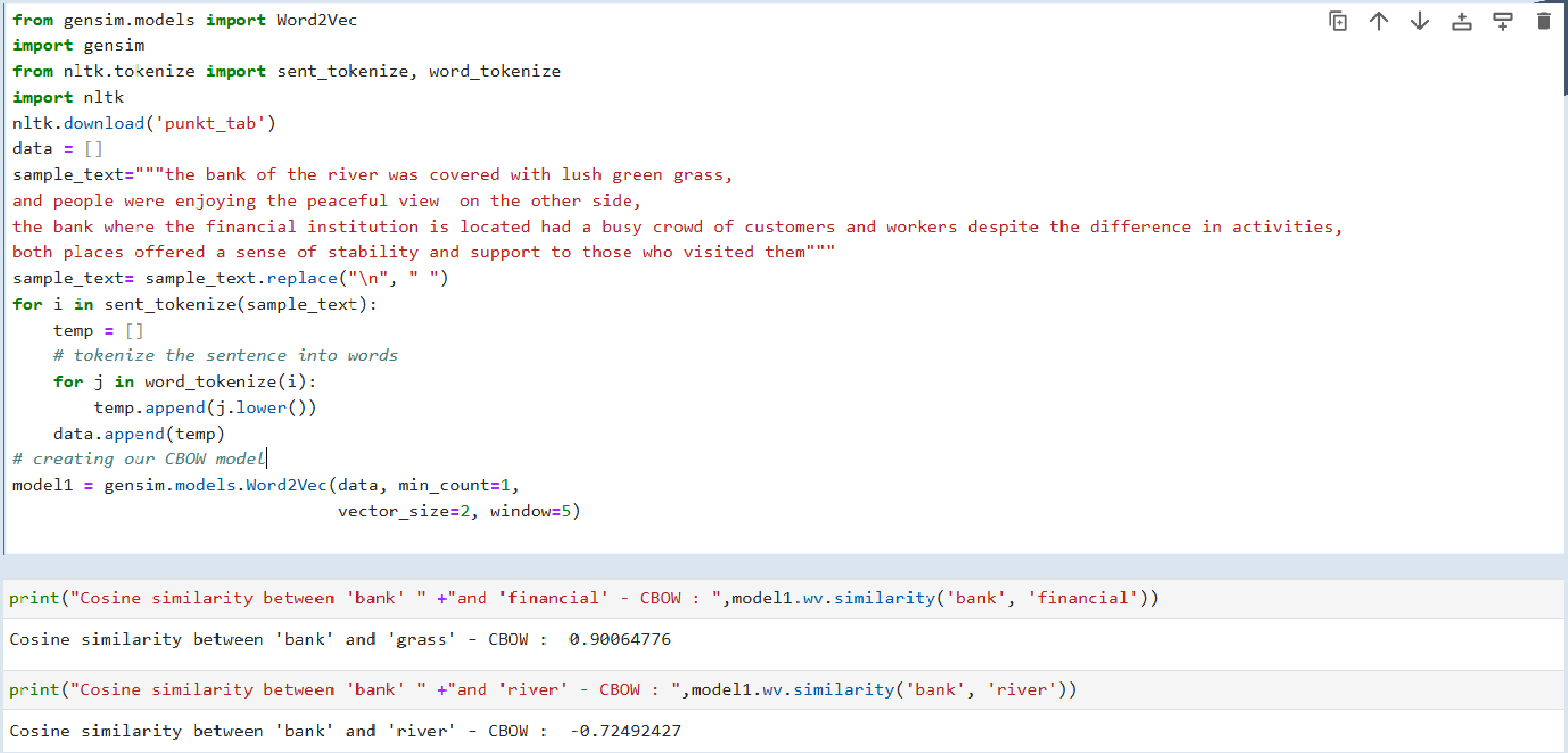
- **"Bank"** can mean:
  - A financial institution: *I deposited money in the bank.*
  - The side of a river: *The fisherman sat on the river bank.*

The proper identification of the intended meaning of such words is crucial in many NLP applications such as **machine translation**, **information retrieval**, and **question answering systems.**

Problem Statement

Traditional approaches to WSD often fail in handling ambiguous words efficiently, especially in highly polysemous contexts where the senses of words overlap significantly. For instance, word embeddings generated by models like **Word2Vec** or **GloVe** do not differentiate between the senses of polysemous words, leading to poor performance in tasks requiring precise word sense identification.

Code for showing the problem using Neural Embeddings



```python
from gensim.models import Word2Vec
import gensim
from nltk.tokenize import sent_tokenize, word_tokenize
import nltk
nltk.download('punkt_tab')
data = []
sample_text="""the bank of the river was covered with lush green grass,
and people were enjoying the peaceful view  on the other side,
the bank where the financial institution is located had a busy crowd of customers and workers despite the difference in activities,
both places offered a sense of stability and support to those who visited them"""
sample_text= sample_text.replace("\n", " ")
for i in sent_tokenize(sample_text):
    temp = []
    # tokenize the sentence into words
    for j in word_tokenize(i):
        temp.append(j.lower())
    data.append(temp)
# creating our CBOW model
model1 = gensim.models.Word2Vec(data, min_count=1,
                                vector_size=2, window=5)
```

```python
print("Cosine similarity between 'bank' " +"and 'financial' - CBOW : ",model1.wv.similarity('bank', 'financial'))
```

```
Cosine similarity between 'bank' and 'grass' - CBOW :  0.90064776
```

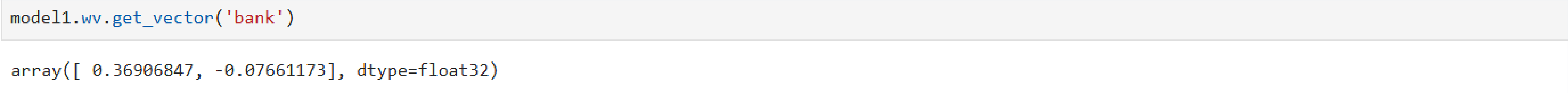
```python
print("Cosine similarity between 'bank' " +"and 'river' - CBOW : ",model1.wv.similarity('bank', 'river'))
```

```
Cosine similarity between 'bank' and 'river' - CBOW :  -0.72492427
```

Analysis of above code

We have trained a Word2Vec model on our sample text and as per output following inference were drawn

- The similarity between the words "River" and "Bank" is negative which implies that these two words are oppositely correlated .
- The words "Bank" and "Financial" are highly correlated since the similarity score is positive.
- But if this common paragraph would have been given to a common layman they would have known the context and use of "bank" in the sentence,also the opposite signs of similarity score suggests that our model is unable to understand the context of word "Bank".



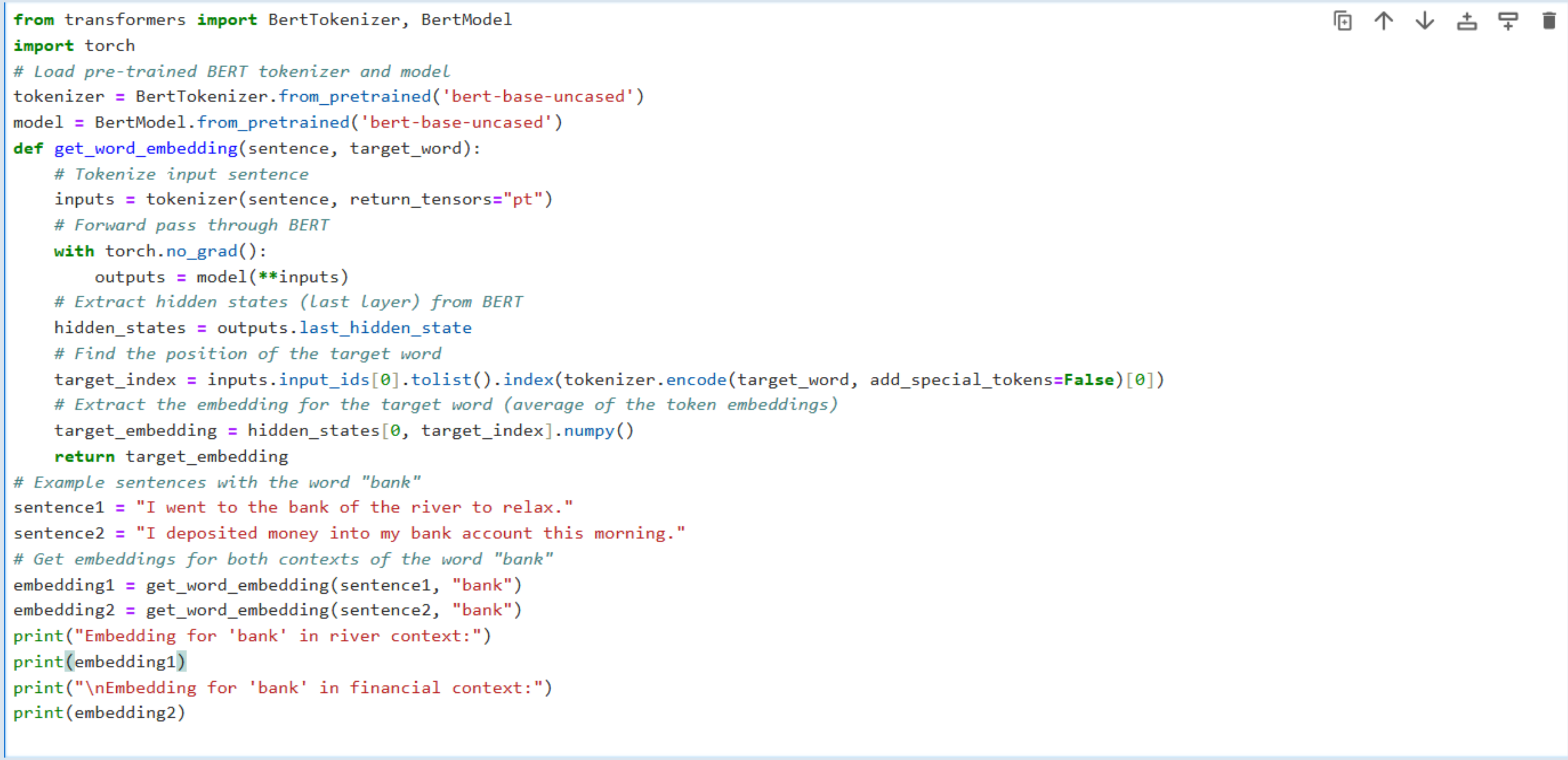
```
model1.wv.get_vector('bank')
```

```
array([ 0.36906847, -0.07661173], dtype=float32)
```

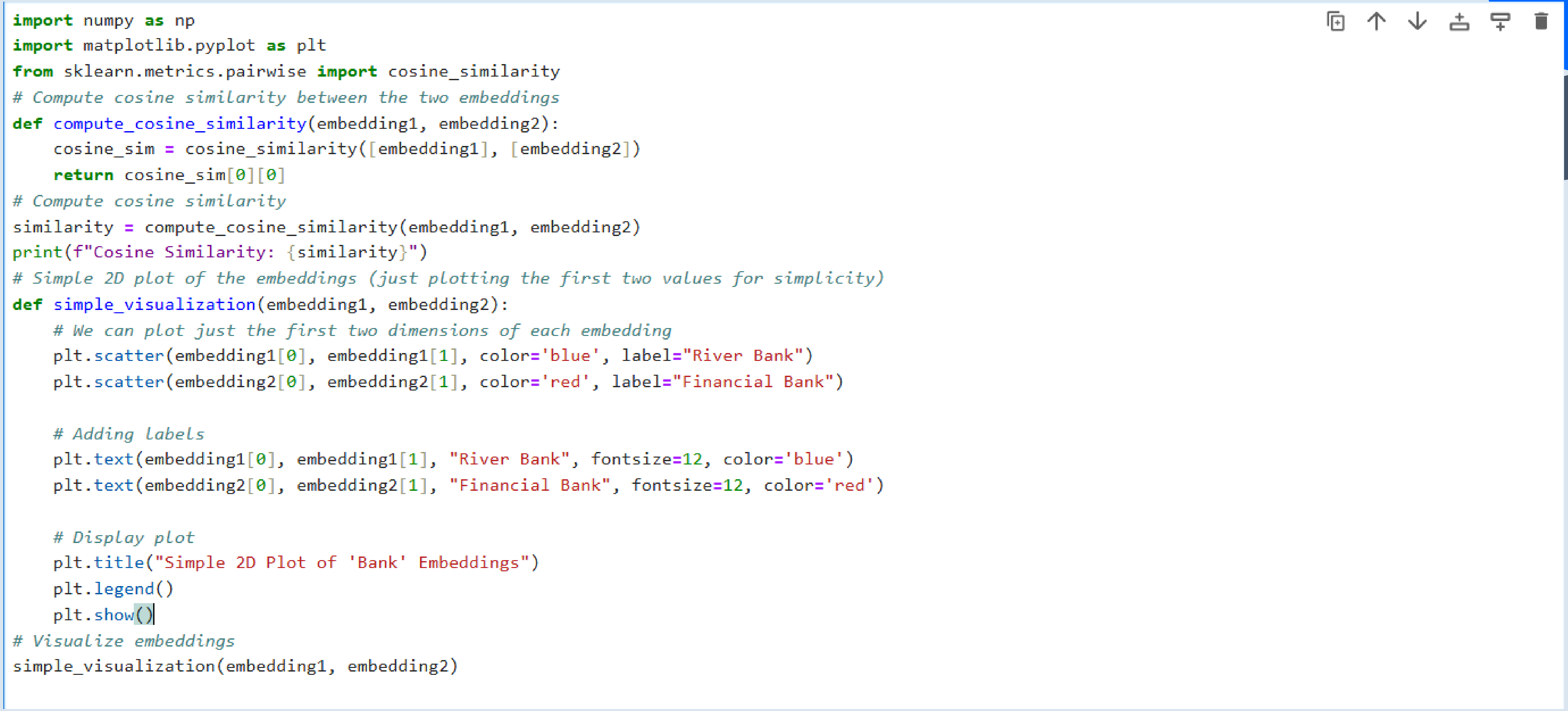
Above code suggests us that vector representation of word "bank" is one only though we would have expected different vectors according to context.

BERT Model

Using modern techniques that can solve the **WSD problem** like **BERT** can help to make contextual meaning of the word and extracting vector representation of the same word ,but still if two same words are used in different sentences the **Cosine Similarity** between two words should be very less or close to –1 so that we can show they are not positively correlated ,but if positive value comes the model is stll unable to make contextual dissimalarity . Below code shows this -



```
from transformers import BertTokenizer, BertModel
import torch
# Load pre-trained BERT tokenizer and model
tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')
model = BertModel.from_pretrained('bert-base-uncased')
def get_word_embedding(sentence, target_word):
    # Tokenize input sentence
    inputs = tokenizer(sentence, return_tensors="pt")
    # Forward pass through BERT
    with torch.no_grad():
        outputs = model(**inputs)
    # Extract hidden states (last layer) from BERT
    hidden_states = outputs.last_hidden_state
    # Find the position of the target word
    target_index = inputs.input_ids[0].tolist().index(tokenizer.encode(target_word, add_special_tokens=False)[0])
    # Extract the embedding for the target word (average of the token embeddings)
    target_embedding = hidden_states[0, target_index].numpy()
    return target_embedding
# Example sentences with the word "bank"
sentence1 = "I went to the bank of the river to relax."
sentence2 = "I deposited money into my bank account this morning."
# Get embeddings for both contexts of the word "bank"
embedding1 = get_word_embedding(sentence1, "bank")
embedding2 = get_word_embedding(sentence2, "bank")
print("Embedding for 'bank' in river context:")
print(embedding1)
print("\nEmbedding for 'bank' in financial context:")
print(embedding2)
```

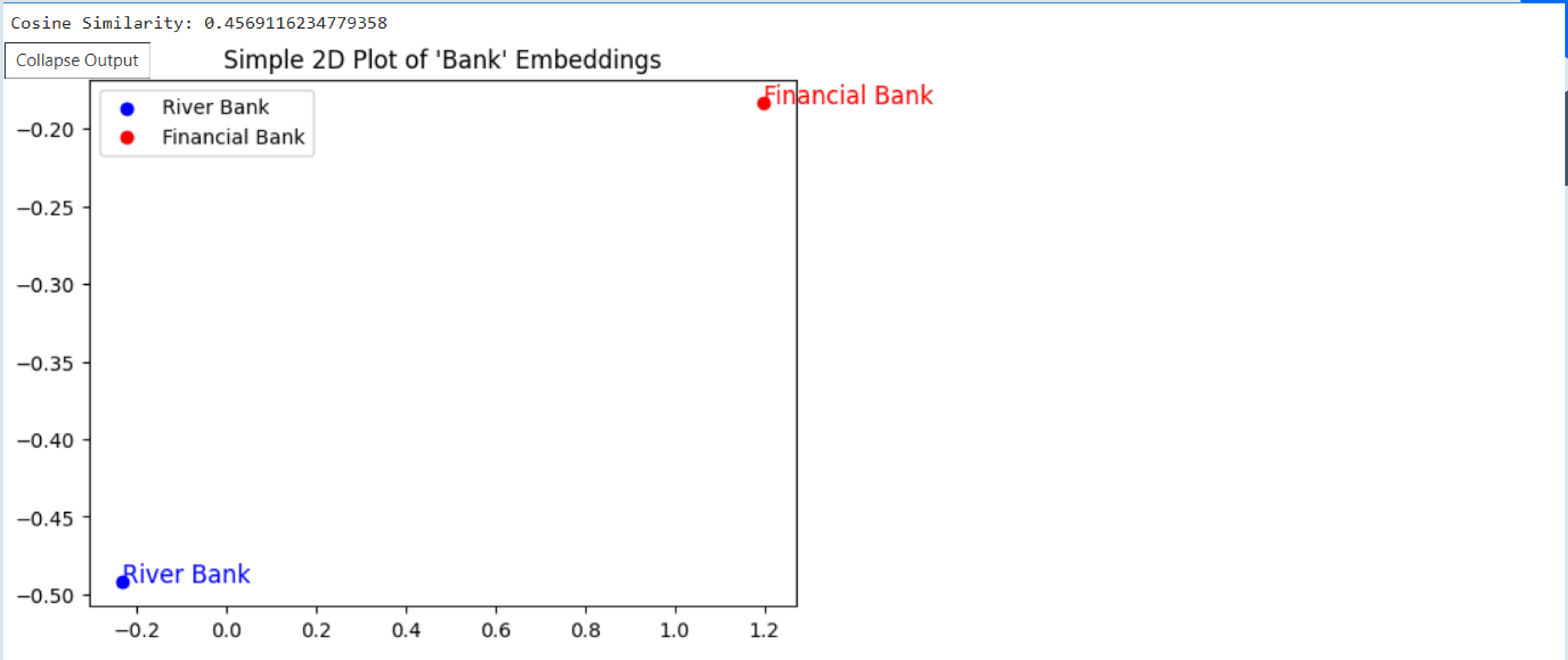


```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics.pairwise import cosine_similarity
# Compute cosine similarity between the two embeddings
def compute_cosine_similarity(embedding1, embedding2):
    cosine_sim = cosine_similarity([embedding1], [embedding2])
    return cosine_sim[0][0]
# Compute cosine similarity
similarity = compute_cosine_similarity(embedding1, embedding2)
print(f"Cosine Similarity: {similarity}")
# Simple 2D plot of the embeddings (just plotting the first two values for simplicity)
def simple_visualization(embedding1, embedding2):
    # We can plot just the first two dimensions of each embedding
    plt.scatter(embedding1[0], embedding1[1], color='blue', label="River Bank")
    plt.scatter(embedding2[0], embedding2[1], color='red', label="Financial Bank")

    # Adding labels
    plt.text(embedding1[0], embedding1[1], "River Bank", fontsize=12, color='blue')
    plt.text(embedding2[0], embedding2[1], "Financial Bank", fontsize=12, color='red')

    # Display plot
    plt.title("Simple 2D Plot of 'Bank' Embeddings")
    plt.legend()
    plt.show()
# Visualize embeddings
simple_visualization(embedding1, embedding2)
```

The above code helps to visualize the **WSD problem** in **BERT**



```
Cosine Similarity: 0.4569116234779358
```

We can see that the **cosine similarity** is **0.45** but this value suggests that they still have some correlation as they are positive ,we wanted them to be close to –1 since cosine similarity has a range **[-1,1]** ,also the grpah presents the distance between two words.

Mathematical modifications to WSD

1. Sense-Specific Projection Matrices

Word embeddings like Word2Vec provide a single vector representation for each word, which fails to distinguish between multiple meanings (senses) of polysemous words. For example, the word *bank* could refer to:

- A financial institution.
- A riverbank.

To overcome this limitation, we use **sense-specific projection matrices**. Each sense of a word is represented in its subspace, which is achieved by learning a unique projection matrix **P(s)** for each sense **s**

By projecting the embeddings of a polysemous word into these sense-specific subspaces, we can separate the meanings into distinct representations, reducing the overlap that arises when a single embedding is used for all senses.

Key Idea

The idea of **sense-specific projection matrices** involves learning a unique matrix **P(s)** for each sense **s** or word

Sense Disambiguation:

- A classifier or a contextual model determines the most likely sense **s** of the word based on the surrounding words (context).

Projection:

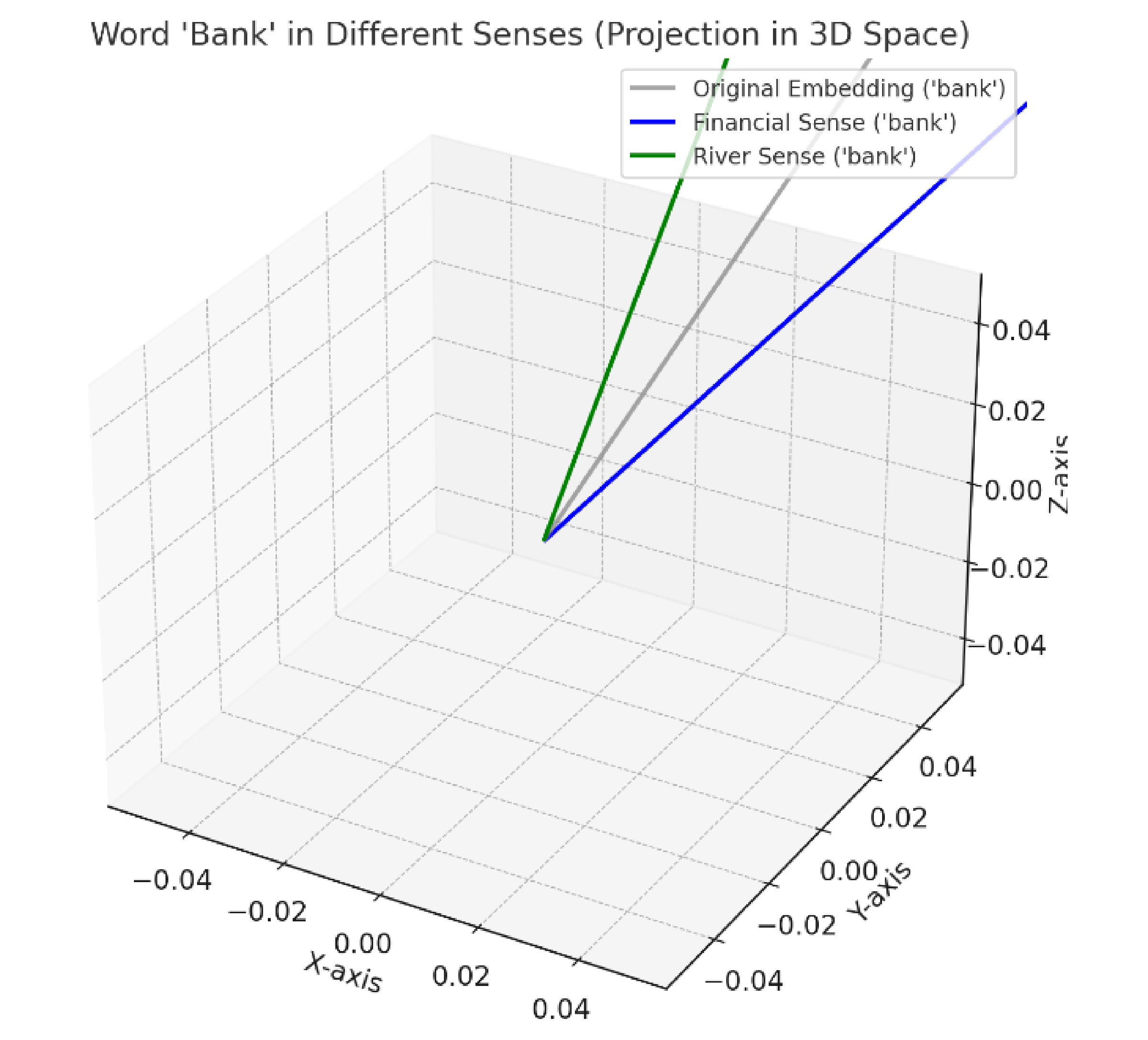
- The original word embedding **w** (shared across all senses) is multiplied by the projection matrix **P(s)** , resulting in **w(s)=P(s)*w,** which represents the word in its sense-specific subspace.

Training:

- The projection matrices **P(s)** are trained using labeled datasets where each instance of the word is annotated with its correct sense. The training optimizes **P(s)** to ensure that embeddings projected into the subspace for s are close to other words with similar meanings.

Analogy: A Prism Splitting Light

Think of the original word embedding as white light containing all possible meanings (senses). The projection matrix acts like a prism that splits the light into its constituent colors (individual senses). Each subspace corresponds to a specific color (sense), providing clarity and separation of meanings.



Using trained models we extract the vectors to specific sense of the main word from our model, which is then taken a **dot product** with the **static vector** representation provided by our **word2vec** model , which projects the specific sense of the word with respect to the context of sentence.

Pseudo Code

Text	Sense-word Vector (Bank)
The river bank is green and pleasant.	B1
The bank faced several financial setbacks.	B2
The bank across the river bank has been closed facing financial issue.	B3

In the above table we have some sentences which has our main word bank but they have different context each. The 2nd column represents the vector representation of "bank" with respect to their context area in respective sentences e.g B1 is the vector representation of word "bank" having context of word "river" and features of the same.

Now B3 is the vector representation of word "bank" having context of both word "river" as well as word "financial". But we want the distinguish between these two different meaning of word "bank" in statement 3. Using **Sense-Specific Projection Matrices** which are **B1** and **B2** we can project **B3** into its constituent context.

```
bank_sense_financial = dot_product(B2,B3)
bank_sense_river= dot_product(B1,B3)
```

WordNet WSD

WordNet is a large lexical database of English that organizes words into sets of synonyms called **synsets**. Each synset represents a distinct concept, meaning, or sense of a word. By leveraging WordNet, we can differentiate between the multiple meanings of a word (e.g., "bank" in different contexts like a **river bank** and a **financial bank**), which is crucial for tasks like **word sense disambiguation (WSD)**.

How WordNet helps ?

Multiple Synsets for Ambiguous Words:

- For an ambiguous word like "bank", WordNet provides multiple synsets, each corresponding to a different sense of the word.
- For example:
  - **bank.n.01**: Refers to the side of a river (i.e., a geographical feature).
  - **bank.n.02**: Refers to a financial institution.

Sense Definitions:

- WordNet includes **definitions** for each synset. These definitions help clarify the meaning of each word sense. For example:
  - **bank.n.01**: "Sloping land beside a body of water."
  - **bank.n.02**: "A financial institution that accepts deposits and makes loans."
- Using these definitions, we can understand that the word "bank" has different meanings based on the context in which it is used.

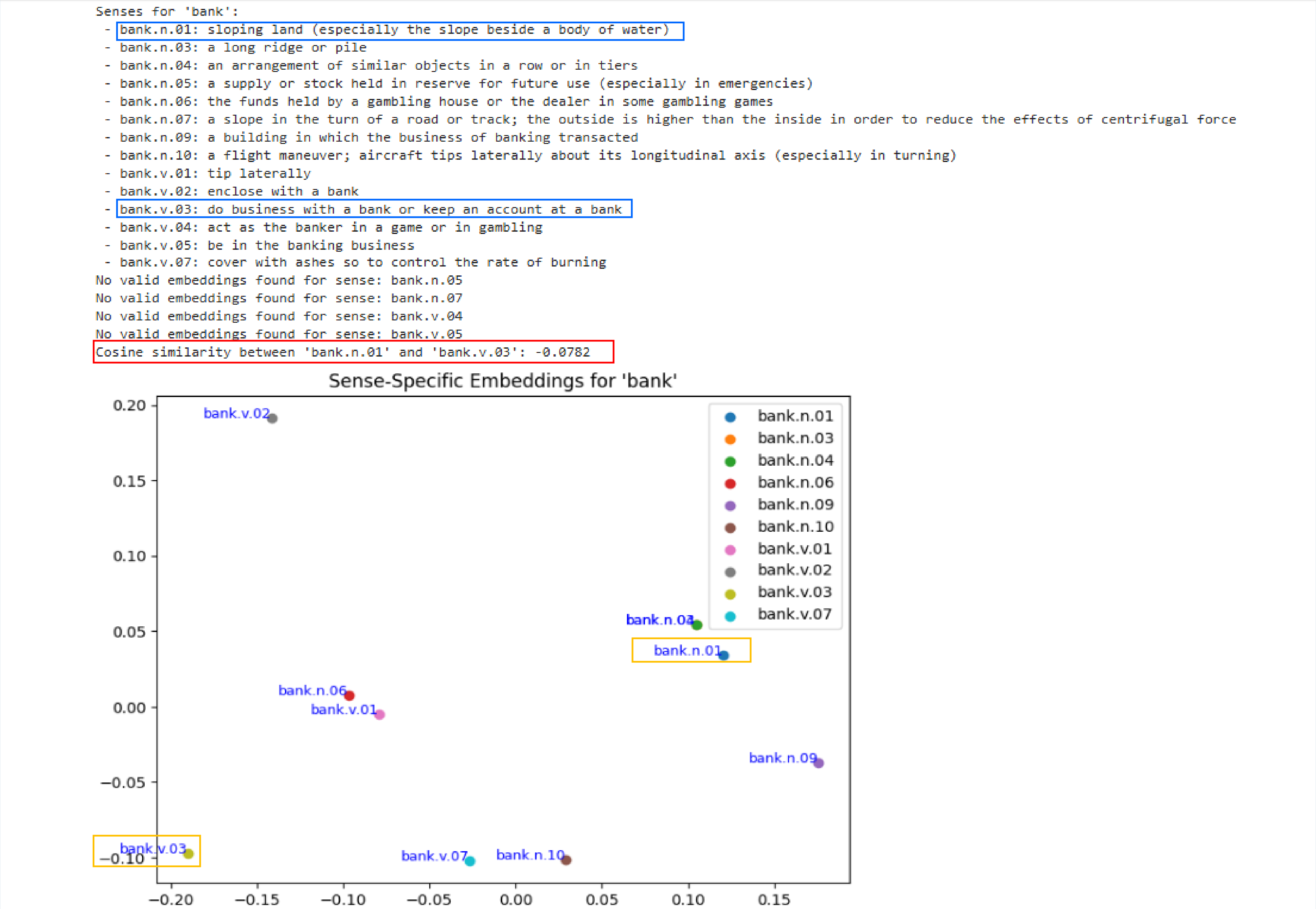
Sense-Specific Word Embeddings:

- In our used codes, we have used **WordNet's synsets** to identify the different senses of the word "bank" and use **contextual examples** to calculate the sense-specific word embeddings.
- For each synset (sense) of the word "bank", we extract example sentences and use them to compute embeddings by averaging the embeddings of words in the example sentences.
- The **projection** of the general word embedding (e.g., for "bank") onto a subspace of the context words in the examples helps create **sense-specific embeddings**.

Sense Differentiation:

- Once we have the **sense-specific embeddings**, we can use techniques like **cosine similarity** and **Euclidean distance** to compare the distance between the embeddings of different senses (e.g., "river bank" vs. "financial bank").
- By comparing the embeddings of different senses, we can quantify how similar or distinct they are. If the embeddings are far apart (high Euclidean distance, low cosine similarity), this indicates that the senses are contextually different.

Using **WordNet** we have generated the following senses and also visualized them pictorially where eac point refers us to a different meaning of the word "bank" , and also we have computed the **cosine similarity** between two human distinguishable examples and analyzed how much well performing our algorithm is.



```
Senses for 'bank':
 - bank.n.01: sloping land (especially the slope beside a body of water)
 - bank.n.03: a long ridge or pile
 - bank.n.04: an arrangement of similar objects in a row or in tiers
 - bank.n.05: a supply or stock held in reserve for future use (especially in emergencies)
 - bank.n.06: the funds held by a gambling house or the dealer in some gambling games
 - bank.n.07: a slope in the turn of a road or track; the outside is higher than the inside in order to reduce the effects of centrifugal force
 - bank.n.09: a building in which the business of banking transacted
 - bank.n.10: a flight maneuver; aircraft tips laterally about its longitudinal axis (especially in turning)
 - bank.v.01: tip laterally
 - bank.v.02: enclose with a bank
 - bank.v.03: do business with a bank or keep an account at a bank
 - bank.v.04: act as the banker in a game or in gambling
 - bank.v.05: be in the banking business
 - bank.v.07: cover with ashes so to control the rate of burning
No valid embeddings found for sense: bank.n.05
No valid embeddings found for sense: bank.n.07
No valid embeddings found for sense: bank.v.04
No valid embeddings found for sense: bank.v.05
Cosine similarity between 'bank.n.01' and 'bank.v.03': -0.0782
```

References

Word2Vec for WSD:
Huang, E. H., Socher, R., Manning, C. D., & Ng, A. Y. (2012). *"Improving Word Representations via Global Context and Multiple Word Prototypes."*

Context-Aware Similarity Measures:
Melamud, O., Goldberger, J., & Dagan, I. (2016). *"context2vec: Learning Generic Context Embedding with Bidirectional LSTM."* Proceedings of The 20th SIGNLL Conference on Computational Natural Language Learning.
Distance Weighted Cosine Similarity Measure for Text Classification Baoli Li and Liping Han

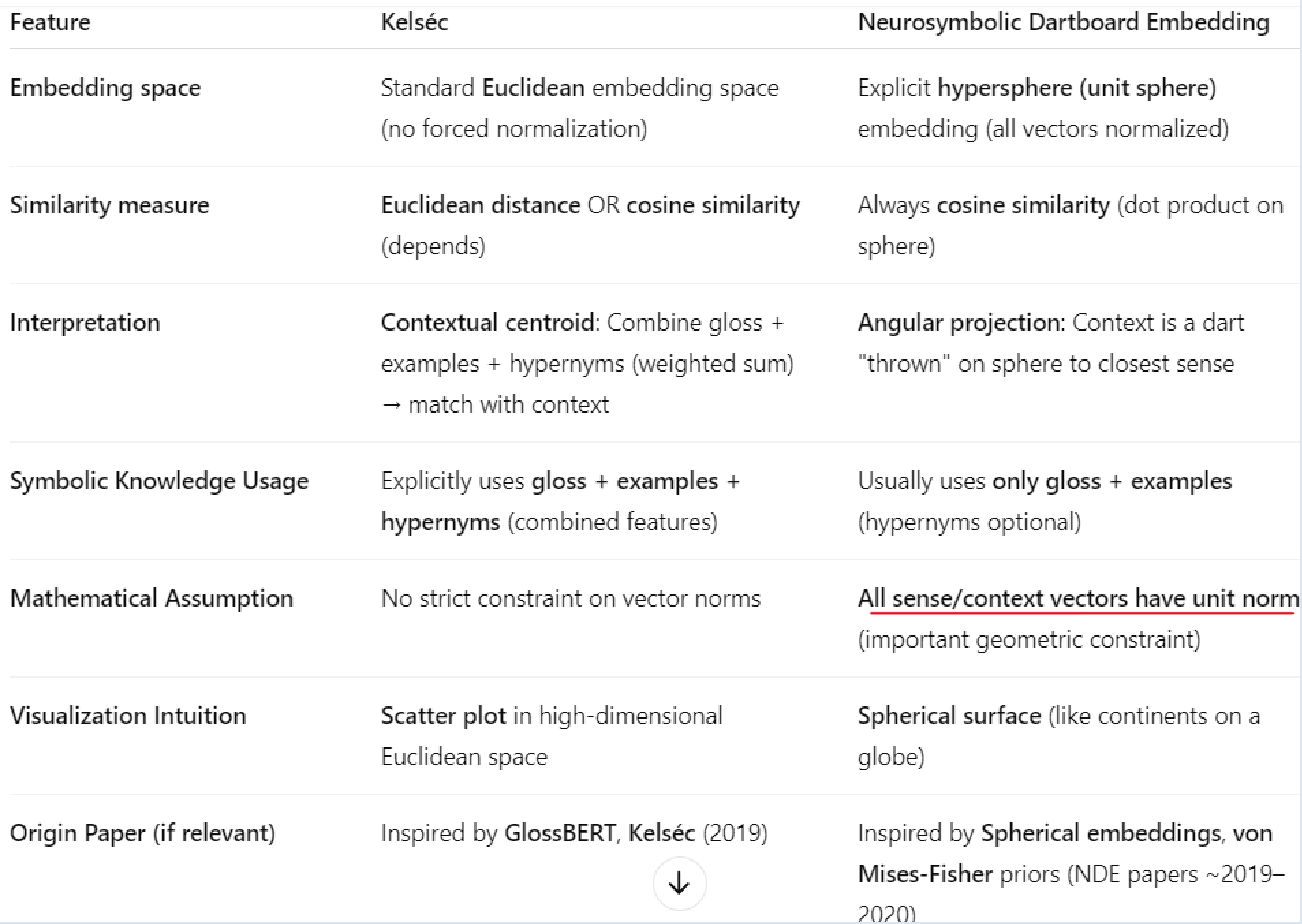
Codes Link :[click here]

Neurosymbolic Dartboard Embedding

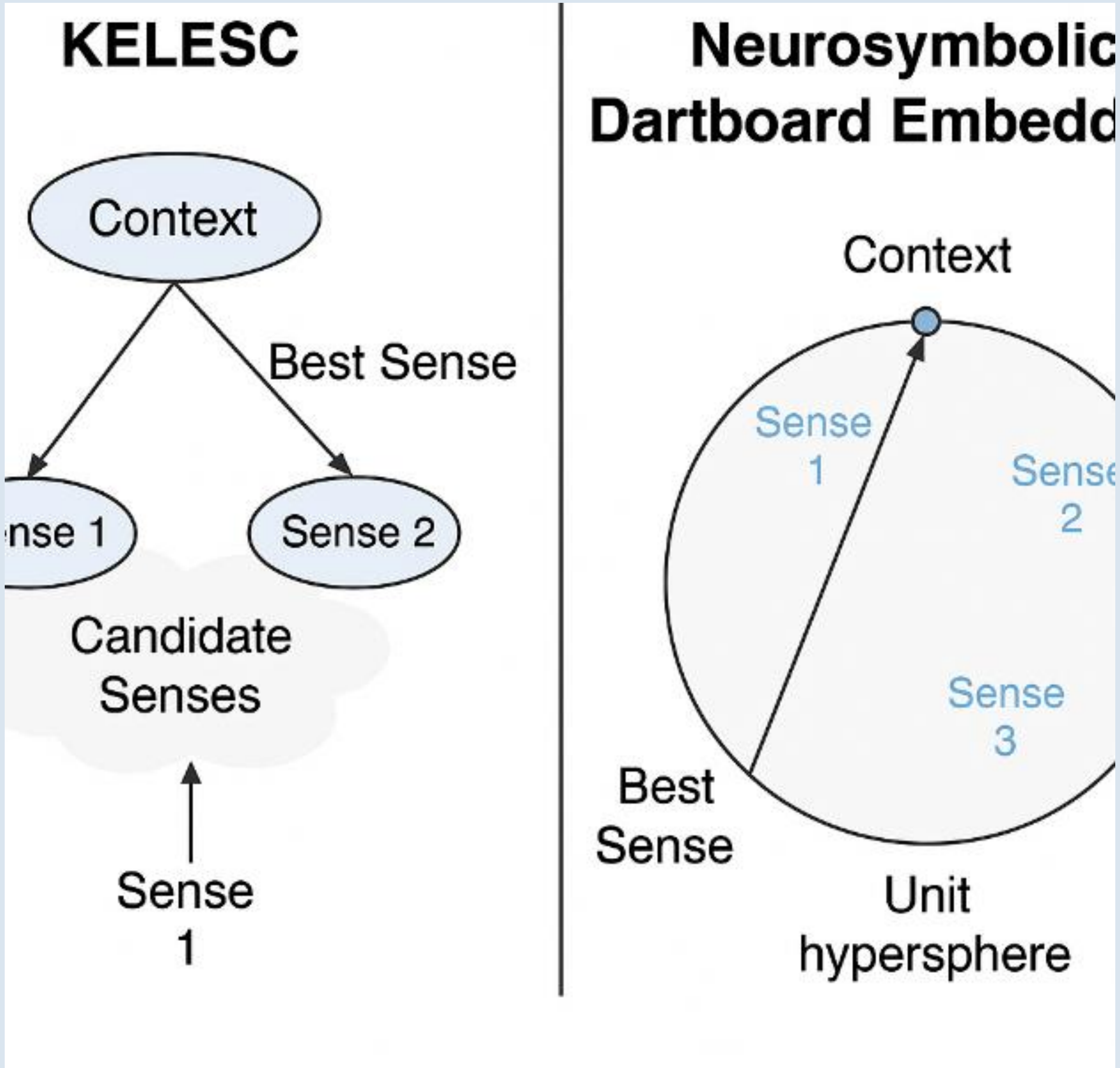
We precisely imposed the taxonomy of classes onto the vector embedding learned by the deep learning networks. This turns vector embeddings into a configuration of nested spheres, which resemble a dart board. Given a word and its context, we build a contextualized vector embedding of this word, as a »dart arrow«. Its word-sense is embedded as a sphere in the neurosymbolic dart board. This process works like shooting a dart arrow into a dart board. Hence, we named our neurosymbolic classifier prototype the `neurosymbolic darter'.

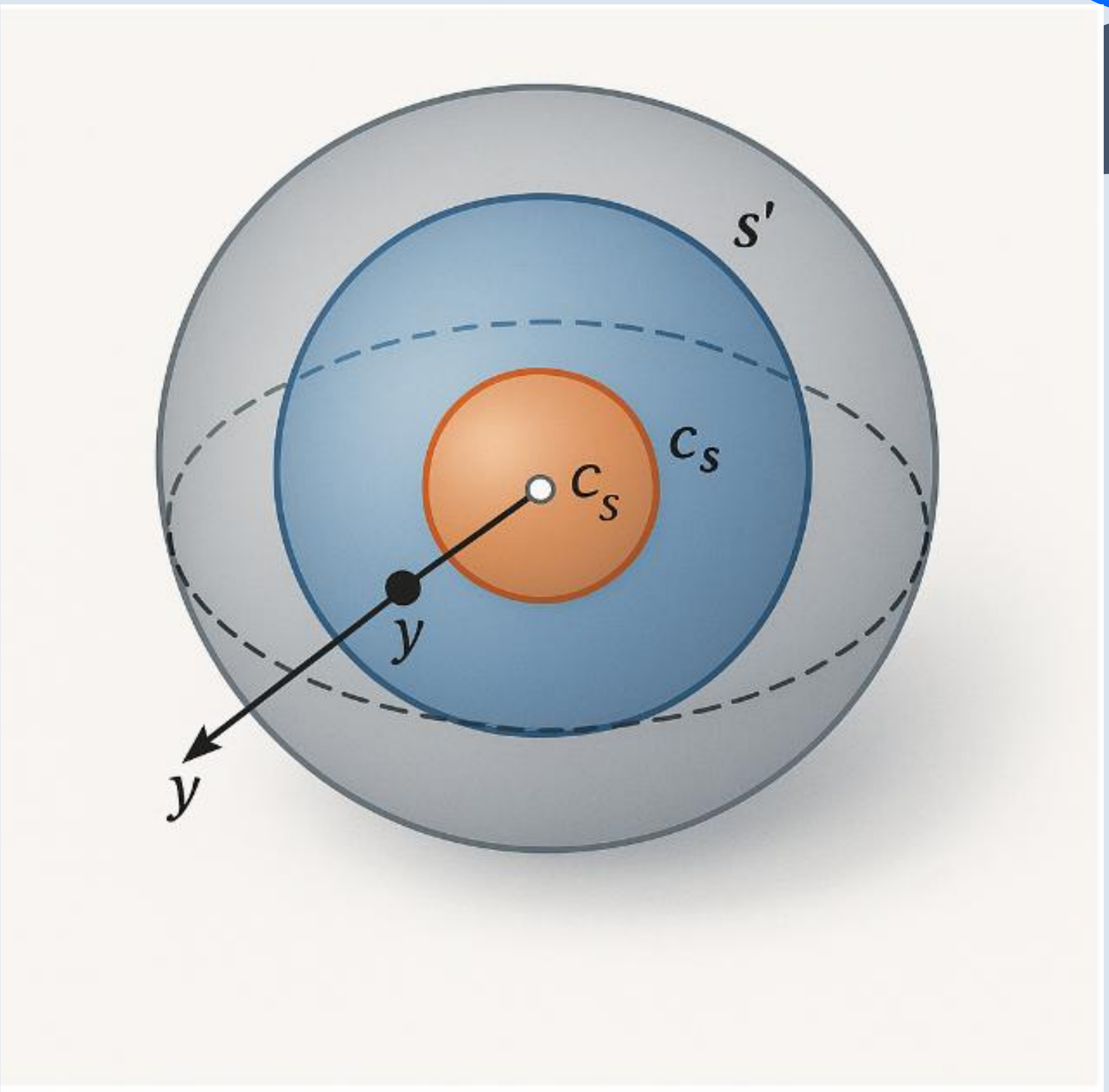
Compared with traditional deep-learning methods, our method has several advantages:
(1) It is much easier to hit a sphere, than hitting a specific point in the vector space, as it is the case with traditional deep-learning methods.
(2) Explicit embedding of the taxonomy enhances the explainability of the results.
(3) the configuration of nested spheres enables logic deduction among the taxonomy of word-senses.

Our experiments show that our »neurosymbolic darter« can break the performance ceiling of pure deep-learning neural-networks. Our neurosymbolic method can push the F1 score above 90%, which breaks the glass ceiling (80%) of pure deep-learning approaches to word-sense disambiguation.



Feature	Kelséc	Neurosymbolic Dartboard Embedding
Embedding space	Standard **Euclidean** embedding space (no forced normalization)	Explicit **hypersphere (unit sphere)** embedding (all vectors normalized)
Similarity measure	**Euclidean distance** OR **cosine similarity** (depends)	Always **cosine similarity** (dot product on sphere)
Interpretation	**Contextual centroid**: Combine gloss + examples + hypernyms (weighted sum) → match with context	**Angular projection**: Context is a dart "thrown" on sphere to closest sense
Symbolic Knowledge Usage	Explicitly uses **gloss** + **examples** + **hypernyms** (combined features)	Usually uses **only gloss** + **examples** (hypernyms optional)
Mathematical Assumption	No strict constraint on vector norms	**All sense/context vectors have unit norm** (important geometric constraint)
Visualization Intuition	**Scatter plot** in high-dimensional Euclidean space	**Spherical surface** (like continents on a globe)
Origin Paper (if relevant)	Inspired by **GlossBERT**, **Kelséc** (2019)	Inspired by **Spherical embeddings**, **von Mises-Fisher** priors (NDE papers ~2019–2020)


KELESC
Context
Best Sense
ense 1
Sense 2
Candidate Senses
Sense 1
Neurosymbolic
Dartboard Embedd
Context
Sense 1
Sense 2
Sense 3
Best Sense
Unit hypersphere


s'
c_s
c_s
y
y

What are the drawbacks with current neurosymbolic dartboard projection ?

The **earlier model** represented **word senses** (or concepts) as **spherical regions** in embedding space:

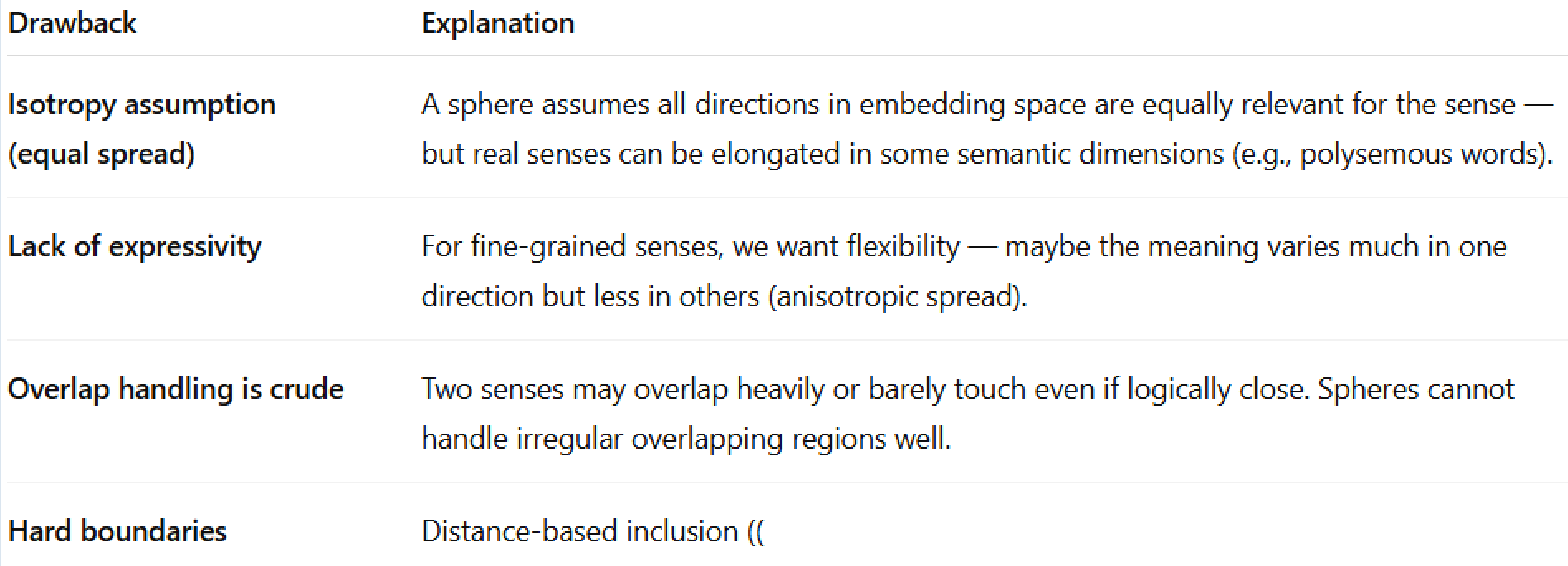
$$\mathcal{E}_s = \{x \in \mathbb{R}^d : \|x - c_s\|^2 \leq r_s^2\}$$

Every sphere of a sense has same property in every direction since the spread is equal everywhere

Here:

- C_s is the center of the sphere = prototype of sense **s**
- R_s is radius (can be fixed or learned)

In this model, all **regions of senses** were **spheres**, i.e., isotropic balls.



Drawback	Explanation
Isotropy assumption (equal spread)	A sphere assumes all directions in embedding space are equally relevant for the sense — but real senses can be elongated in some semantic dimensions (e.g., polysemous words).
Lack of expressivity	For fine-grained senses, we want flexibility — maybe the meaning varies much in one direction but less in others (anisotropic spread).
Overlap handling is crude	Two senses may overlap heavily or barely touch even if logically close. Spheres cannot handle irregular overlapping regions well.
Hard boundaries	Distance-based inclusion ((

Modified Neurosymbolic DartBoard Embedding

The update generalized spherical regions to **ellipsoids**:

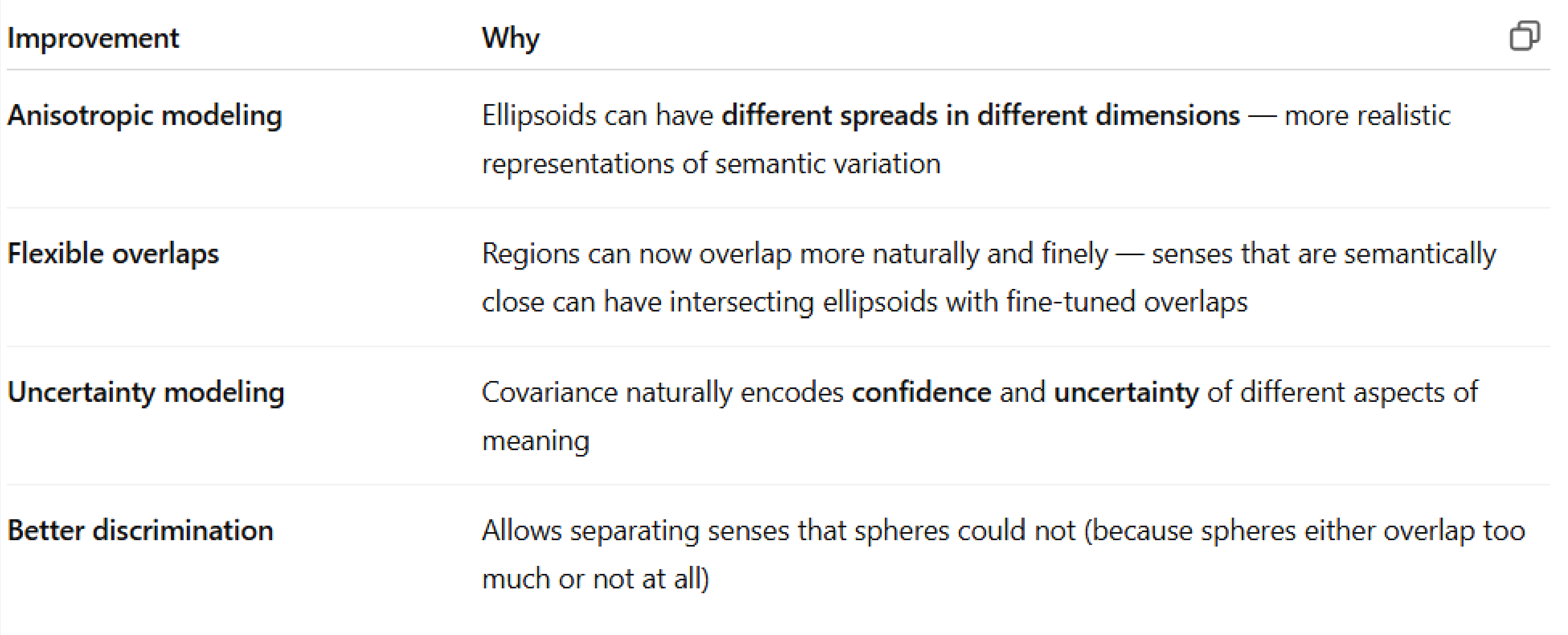
$$\mathcal{E}_s = \{x \in \mathbb{R}^d : (x - c_s)^\top \Sigma_s^{-1}(x - c_s) \leq 1\}$$

Where:

- C_s is still the center

$\Sigma_s \in \mathbb{R}^{d\times d}$ i is a **positive definite covariance matrix** → controls **shape**, **orientation**, and **size**

Improvements



Improvement	Why
Anisotropic modeling	Ellipsoids can have **different spreads in different dimensions** — more realistic representations of semantic variation
Flexible overlaps	Regions can now overlap more naturally and finely — senses that are semantically close can have intersecting ellipsoids with fine-tuned overlaps
Uncertainty modeling	Covariance naturally encodes **confidence** and **uncertainty** of different aspects of meaning
Better discrimination	Allows separating senses that spheres could not (because spheres either overlap too much or not at all)

Some important key points :-

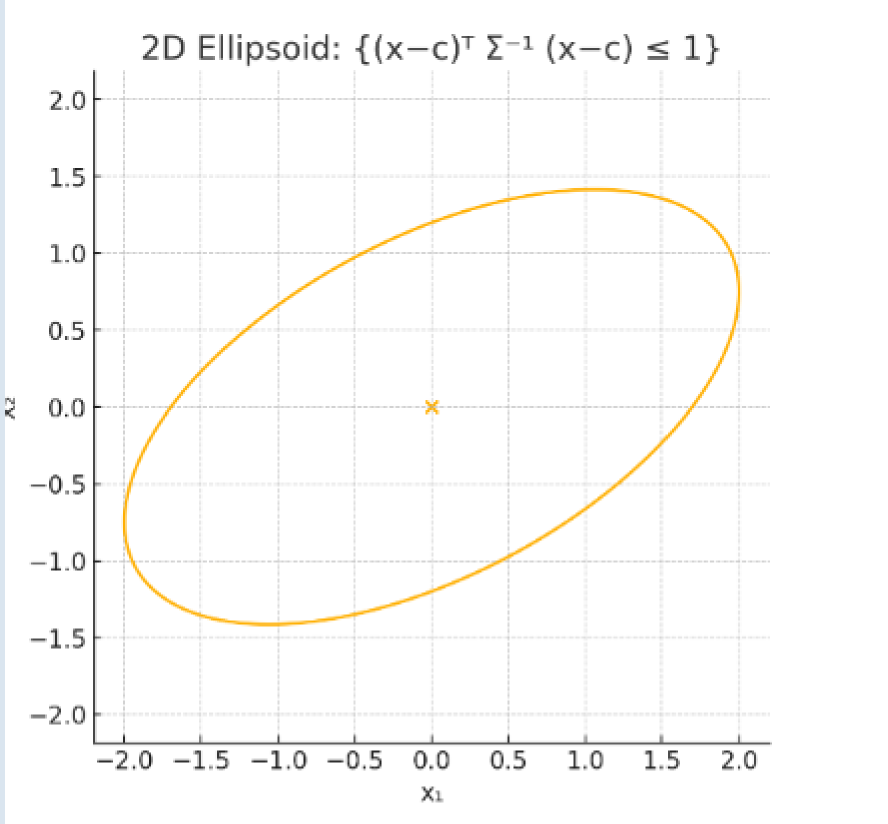
Due to elongated spreadness in ellipsoid multiple sense of a word in same space are distributed , different **gaussian space** for each sense , rather than strict cluster we get a probabilistic cluster for particular projection of target word and we get highest score sense.

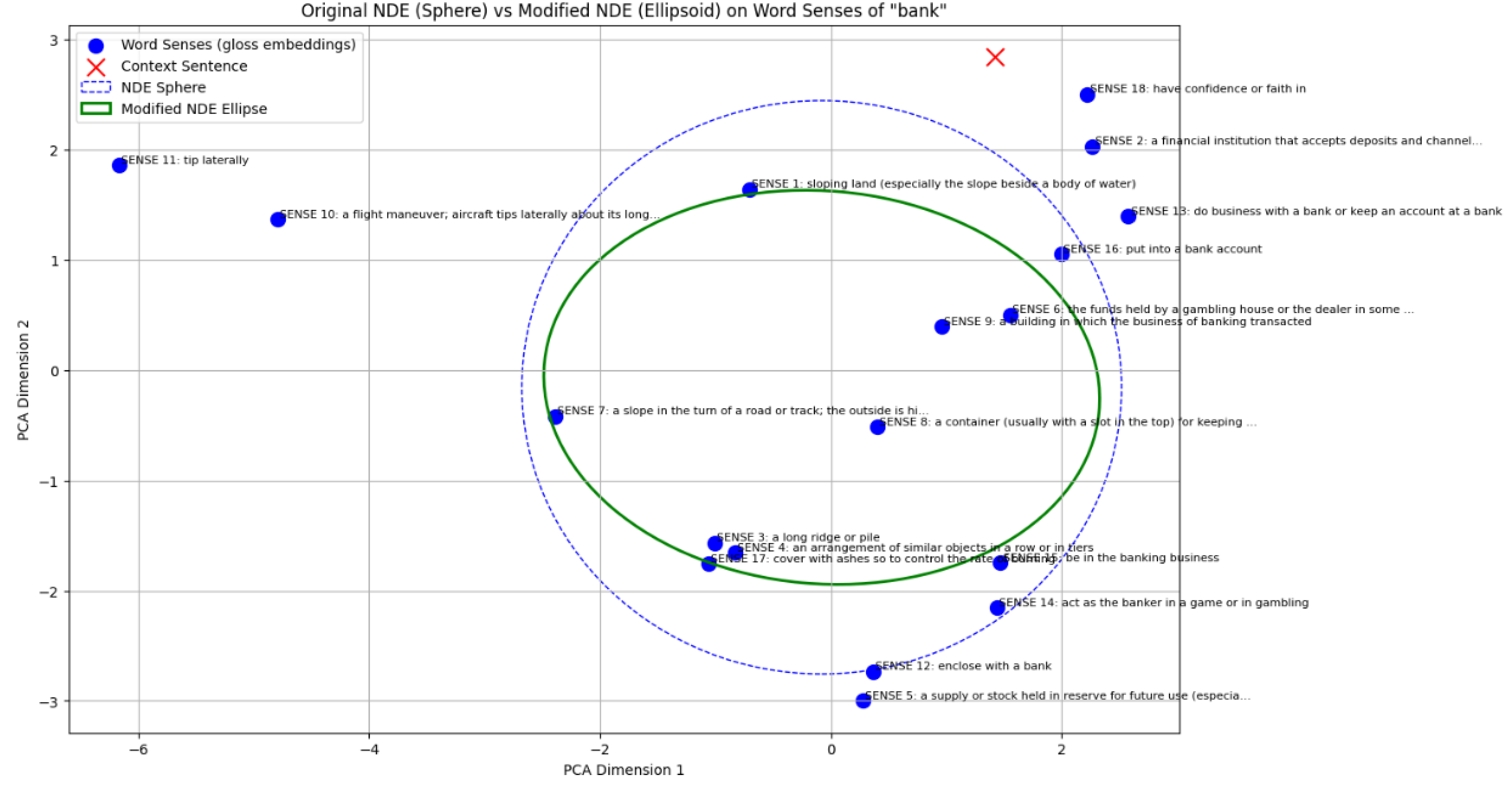
Structured Radii Initialization

Use **sense frequency** or **depth** in the WordNet hypernym tree to set initial r_s:

$$r_s^{(0)} = \alpha \frac{1}{\log(\mathrm{freq}_s + e)} + \beta\, \mathrm{depth}(s).$$

•Scalars α,β controlling trade-off.
•$freq_s$ corpus count; higher frequency → larger radius.
•depth(s) is depth in hypernym graph; deeper → smaller radius.




Original NDE (Sphere) vs Modified NDE (Ellipsoid) on Word Senses of "bank"
Word Senses (gloss embeddings)
Context Sentence
NDE Sphere
Modified NDE Ellipse
PCA Dimension 1
PCA Dimension 2
−6
−4
−2
0
2
3
2
1
0
−1
−2
−3
SENSE 18: have confidence or faith in
SENSE 2: a financial institution that accepts deposits and channel...
SENSE 11: tip laterally
SENSE 1: sloping land (especially the slope beside a body of water)
SENSE 10: a flight maneuver; aircraft tips laterally about its long...
SENSE 13: do business with a bank or keep an account at a bank
SENSE 16: put into a bank account
SENSE 6: the funds held by a gambling house or the dealer in some ...
SENSE 9: a building in which the business of banking transacted
SENSE 7: a slope in the turn of a road or track; the outside is hi...
SENSE 8: a container (usually with a slot in the top) for keeping ...
SENSE 3: a long ridge or pile
SENSE 4: an arrangement of similar objects in a row or in tiers
SENSE 17: cover with ashes so to control the rate of burning
SENSE 15: be in the banking business
SENSE 14: act as the banker in a game or in gambling
SENSE 12: enclose with a bank
SENSE 5: a supply or stock held in reserve for future use (especia...

Contextual Box Embedding (ProtoBox)

It proposes ProtoBox, a novel method to learn contextualized box embeddings. Unlike an ordinary word embedding, which represents a word as a single vector, a box embedding represents the meaning of a word as a box in a high-dimensional space: that is suitable for representing semantic relations between words. In addition, our method aims to obtain a “contextualized” box embedding, which is an abstract representation of a word in a specific context. ProtoBox is based on Prototypical Networks, which is a robust method for classification problems, especially focusing on learning the hypernym–hyponym relation between senses. ProtoBox is evaluated on three tasks: Word Sense Disambiguation (WSD), New Sense Classification (NSC), and Hypernym Identification (HI). Experimental results show that ProtoBox outperforms baselines for the HI task and is comparable for the WSD and NSC tasks.

Box Embeddings: A More Expressive Representation

Instead of representing a concept as a **point**, we represent it as a **box** ***B*** $\in \mathbf{R}^{\mathrm{d}}$ i.e a hyperrectangle.

•A box is defined by its **lower corner I** $\in \mathbf{R}^{\mathrm{d}}$ and **upper corner U** $\in \mathbf{R}^{\mathrm{d}}$, where $\boldsymbol{l}_i < \boldsymbol{u}_i$ for all dimensions I.

•So a concept like "animal" may have a large box, and "cat" a smaller box *inside* it.
This lets us capture **inclusion** (hierarchy) and **intersection** (semantic overlap).

ProtoBox Architecture for WSD

ProtoBox applies this box idea to **contextual word embeddings**, modeling senses as **prototype boxes** and contextual usage of a word as **instance boxes**.

Prototype Boxes for Word Senses

Each word sense s is assigned a prototype box:

$$\boldsymbol{B}_s = [\, \boldsymbol{l}_s, \mathbf{u}_s \,] \text{ where } \boldsymbol{l}_s, \mathbf{u}_s \in \boldsymbol{R}^d \text{ , } \boldsymbol{l}_s \leq \mathbf{u}_s$$

These are trainable parameters and represent *abstract sense concepts*, e.g., "bank" (financial) vs "bank" (river).

Instance Box for Word-in-Context

Given a sentence, we use a contextual encoder (e.g., BERT) to extract a contextual vector $\boldsymbol{h}_w \in \boldsymbol{R}^d$ for word w. This is used to parameterize a box:

$$\boldsymbol{B}_w = [\, \boldsymbol{l}_w, \mathbf{u}_w \,] = \boldsymbol{f}_\theta(\boldsymbol{h}_w)$$

Here, $\boldsymbol{f}_\theta$ is a small neural network that maps the context vector to box corners.

Scoring Function: How Does It Match Boxes?

To determine how well an instance box $\boldsymbol{B}_w$ matches a prototype $\boldsymbol{B}_s$, ProtoBox computes the **Intersection-over-Union (IoU)** between the boxes:

Intersection Box:

For each dimension i :

$$\boldsymbol{l}_i^{\cap} = \max(\boldsymbol{l}_i^{w}, \boldsymbol{l}_i^{s})\,, \boldsymbol{u}_i^{\cap} = \min(\boldsymbol{u}_i^{w}, \boldsymbol{u}_i^{s})$$

If $\boldsymbol{l}_i^{\cap} > \boldsymbol{u}_i^{\cap}$, the intersection is empty in that dimension.

So the **intersection volume**:

$$V_{\text{int}} = \prod_{i=1}^{d} \max(0, u_i^{\cap} - l_i^{\cap})$$

Union Volume:

$$l_i^{\cup} = \min(l_i^{w}, l_i^{s}), \quad u_i^{\cup} = \max(u_i^{w}, u_i^{s})$$

$$V_{\text{union}} = \prod_{i=1}^{d} (u_i^{\cup} - l_i^{\cup})$$

IoU Score:

$$\text{IoU}(\boldsymbol{B}^{\mathrm{w}}, \boldsymbol{B}^{\mathrm{s}}) = \frac{\boldsymbol{V}_{int}}{\boldsymbol{V}_{union} + \in}$$

This score is used as the **similarity between context and sense**, where higher IoU → better match.

Loss Function: Learning to Match Correct Senses

Given training data with labeled senses, we treat this as a **classification problem**.

Cross-Entropy with IoU Logits:

Let $\boldsymbol{S}_w$ be the set of possible senses for word w. For each s ∈ $\boldsymbol{S}_w$ compute IoU$(\boldsymbol{B}^{\mathrm{w}}, \boldsymbol{B}^{\mathrm{s}})$ normalize via softmax:

$$\boldsymbol{P}(\mathrm{s} \mid \mathrm{w}) = \frac{\exp(\mathrm{IoU}(\boldsymbol{B}^{\mathrm{w}}, \boldsymbol{B}^{\mathrm{s}}))}{\sum s\prime \in \boldsymbol{S}_w \exp(\mathrm{IoU}(\boldsymbol{B}^{\mathrm{w}}, \boldsymbol{B}^{\mathrm{s}\prime}))}$$

Then use cross-entropy against the true sense label s*:

$$\mathrm{L} = -\log \mathrm{P}(s^{*} | \mathrm{w})$$

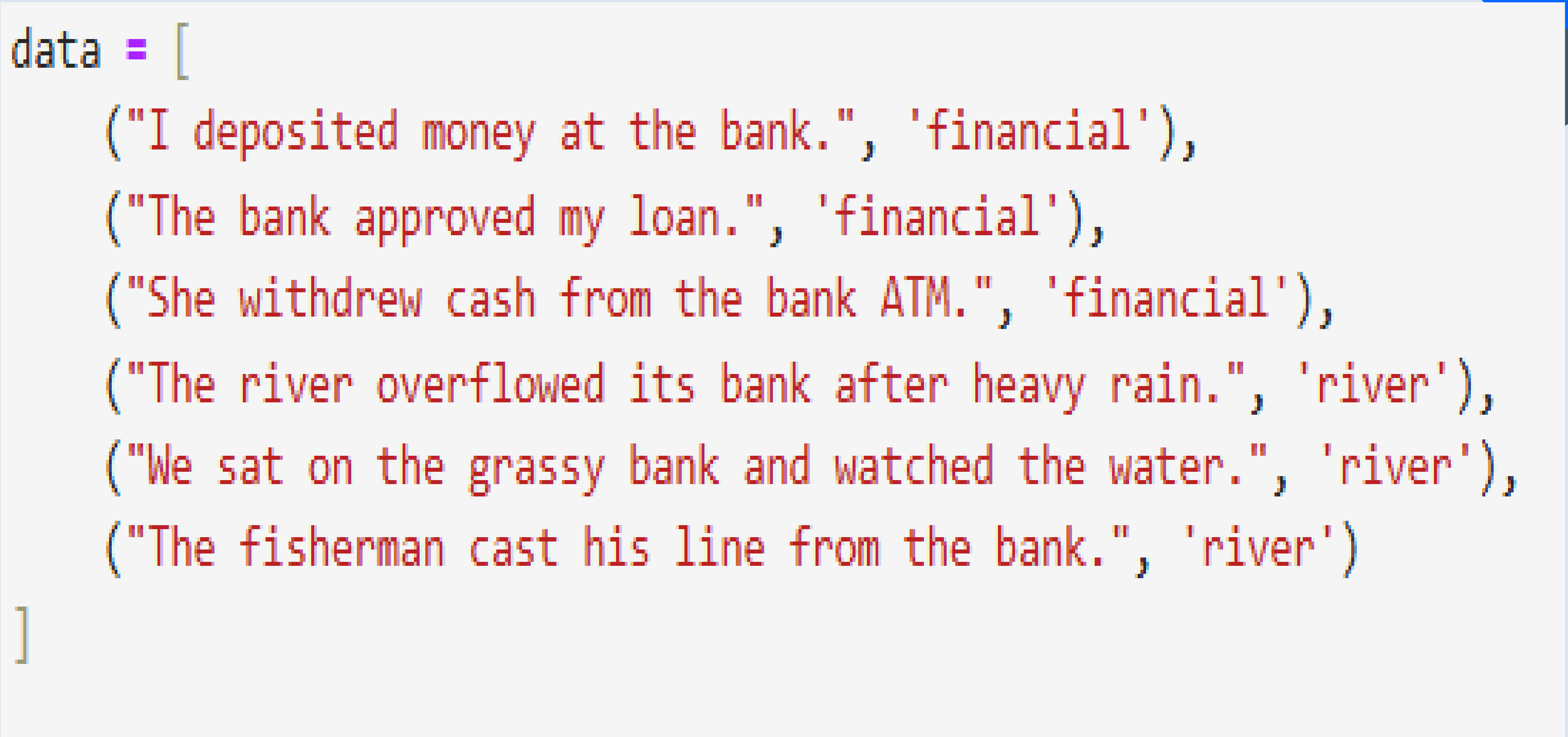
Interpretability: Why Boxes Are Useful

- **Containment**: If box $A \subseteq B$ → A is a *hyponym* of B (e.g., “dog” $\subseteq$ “animal”)
- **Disjointness**: If $A \cap B = \emptyset$ → they’re semantically unrelated
- **Overlap**: Shared context (e.g., “bank” (river) overlaps with “shore”)

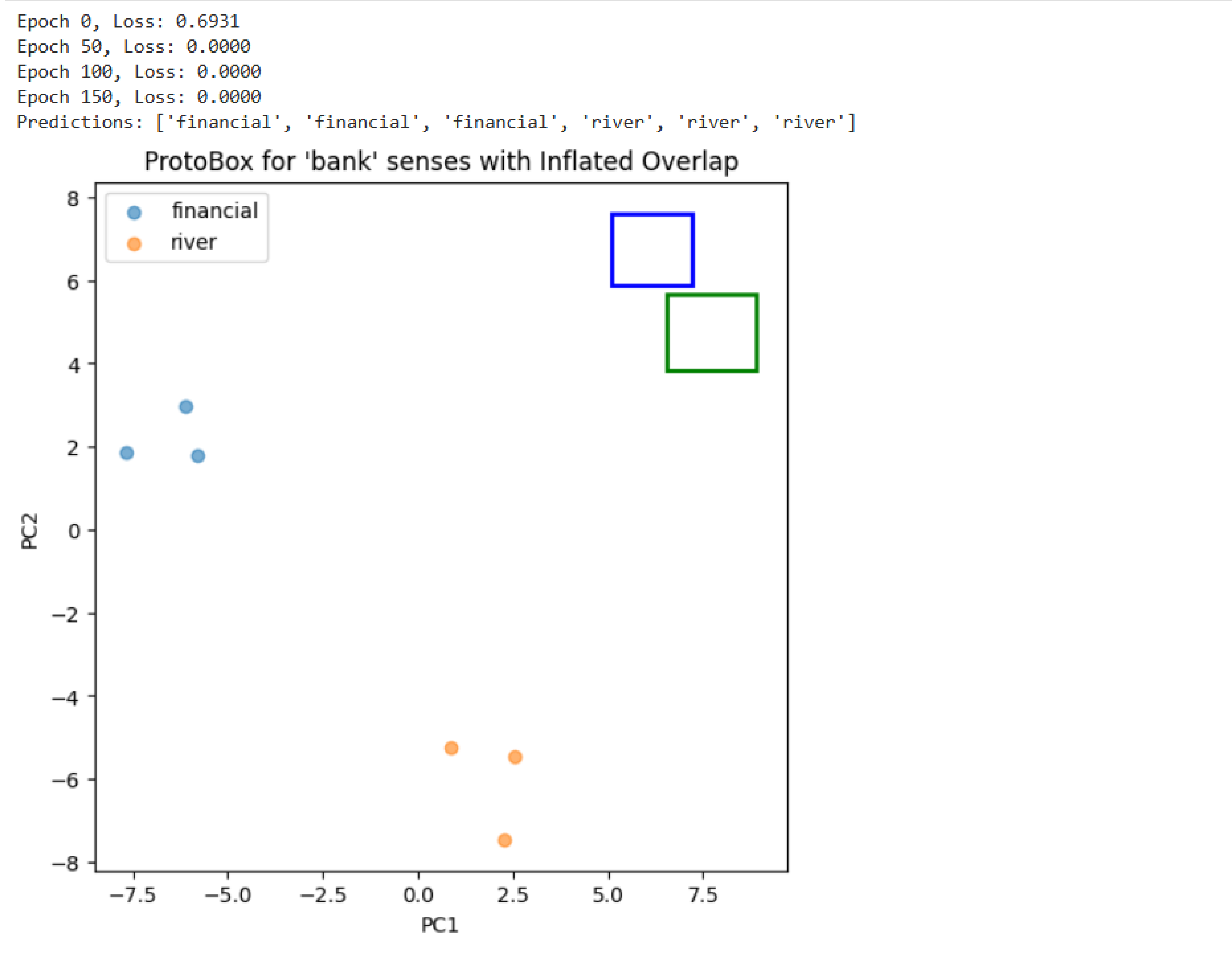
These visualizable properties make ProtoBox **much more interpretable** than point-based BERT or dense vectors.

Experimental Validation

Tasks: Word Sense Disambiguation, Hypernym Prediction, Few-Shot Learning
Datasets: WordNet WSD, HyperLex, WiC
Results: Outperformed BERT baselines in interpretability & structured reasoning, similar accuracy.



```
data = [
    ("I deposited money at the bank.", 'financial'),
    ("The bank approved my loan.", 'financial'),
    ("She withdrew cash from the bank ATM.", 'financial'),
    ("The river overflowed its bank after heavy rain.", 'river'),
    ("We sat on the grassy bank and watched the water.", 'river'),
    ("The fisherman cast his line from the bank.", 'river')
]
```



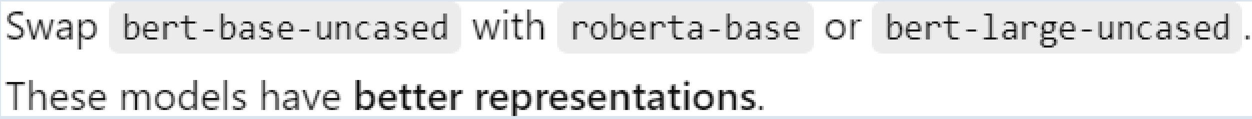
```
Epoch 0, Loss: 0.6931
Epoch 50, Loss: 0.0000
Epoch 100, Loss: 0.0000
Epoch 150, Loss: 0.0000
Predictions: ['financial', 'financial', 'financial', 'river', 'river', 'river']
```

How can we modify ?

Better Contextual Embeddings

Instead of using frozen **BERT** token embeddings, we can:
- Use **larger models** like RoBERTa or BERT-large
- Incorporate **sentence-level context**

BERT has only 110 M parameters which are Limited Was trained with a simple masked language modeling objective and might **not capture all subtle contextual clues** that differentiate senses.



Swap `bert-base-uncased` with `roberta-base` or `bert-large-uncased`.

These models have **better representations**.

Example: concatenate [CLS] embedding + token embedding
- x = [$\boldsymbol{x}_{CLS}, \boldsymbol{X}_{bank}$] (dimension=2d) **Fine-tune** BERT on the task
- **Inject sense definitions** (as in **GlossBERT**)

→ Example: prepend the WordNet gloss to the sentence before embedding

Incorporate sentence-level context (Concatenate [CLS] + token embedding)

In your original model, you used **only the embedding of the word "bank"**
→ It may not capture **global context** (e.g., whether the sentence is talking about money or nature).

Intuition

The **[CLS] token** embedding summarizes the **entire sentence**.
If we **combine [CLS] + token embedding**, the model sees both:
•**Local information** (word-level meaning)
•**Global context** (sentence-level clues)

$$\mathbf{x} = [\mathbf{x}_{\text{CLS}};\ \mathbf{x}_{\text{bank}}] \quad (\text{dimension} = 2d)$$

Inject sense definitions (as in GlossBERT)

Problem
In hard examples, **just the sentence may not be enough** to resolve sense.
→ Example:
Sentence: "He walked along the bank."
This could be both **river** or **institution** without extra info.

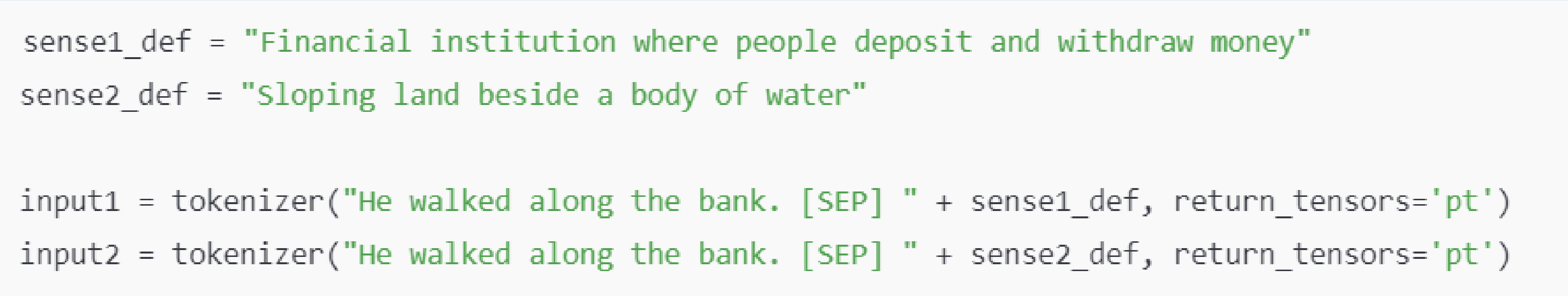
Intuition

If we **provide the model with the WordNet definition (gloss)** of each sense,
it can **match sentence context to sense definition** more explicitly.

Concrete Method

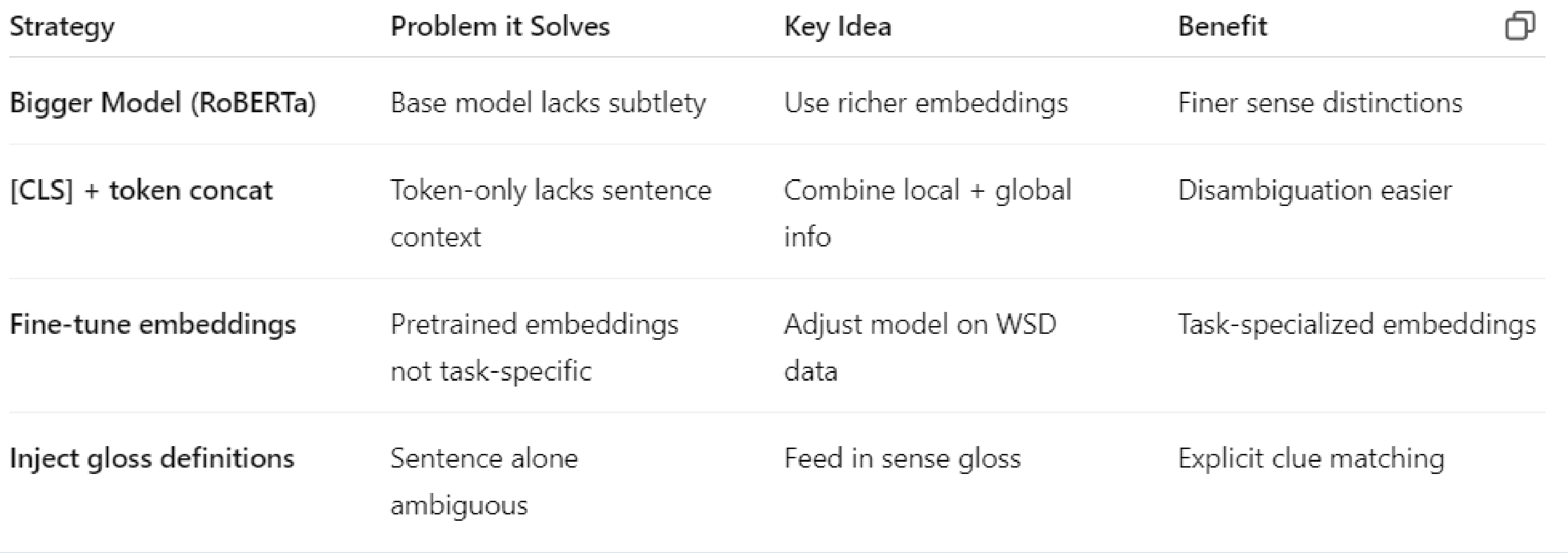
Concatenate gloss text to sentence (like NLI task):

E.G
Input: "He walked along the bank. [SEP] Sense1: Financial institution."
Input: "He walked along the bank. [SEP] Sense2: Sloping land beside river."

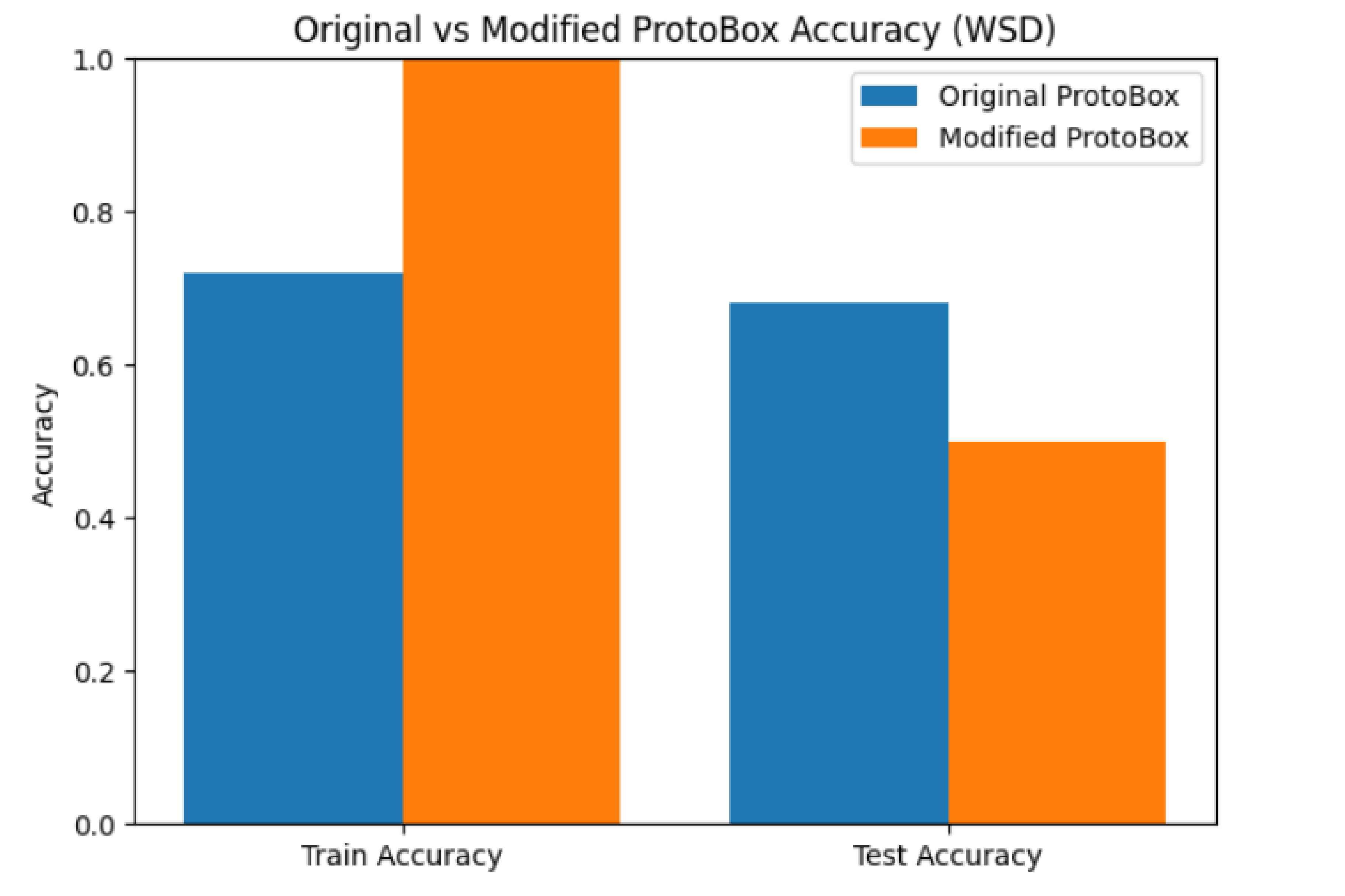


```
sense1_def = "Financial institution where people deposit and withdraw money"
sense2_def = "Sloping land beside a body of water"

input1 = tokenizer("He walked along the bank. [SEP] " + sense1_def, return_tensors='pt')
input2 = tokenizer("He walked along the bank. [SEP] " + sense2_def, return_tensors='pt')
```



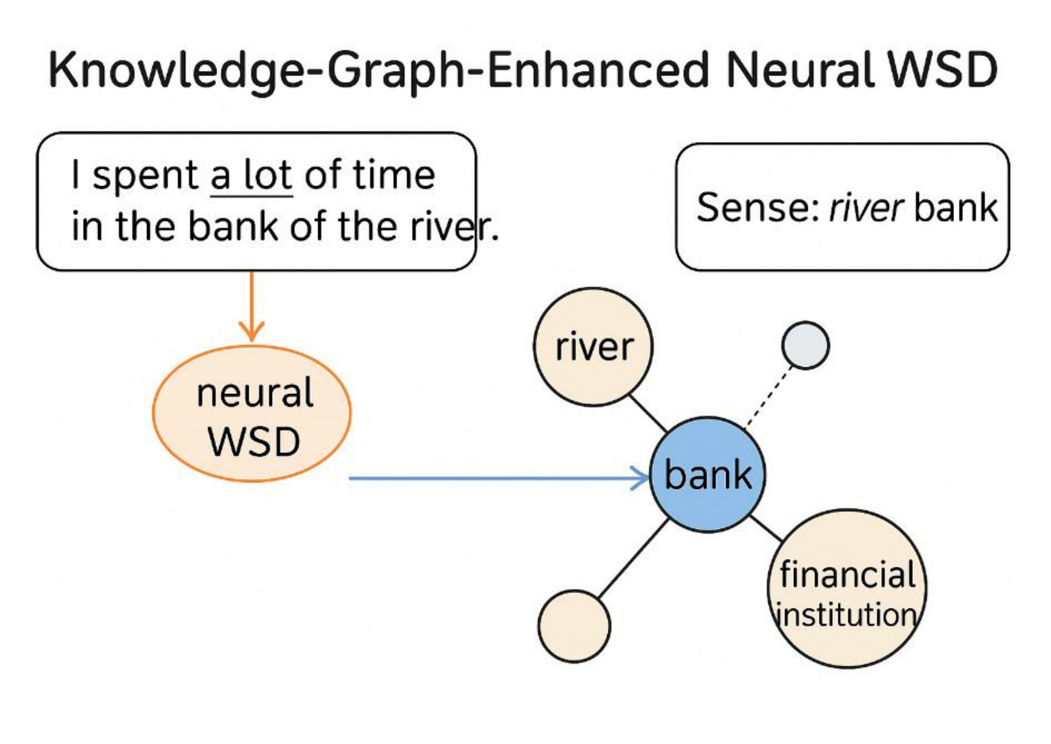
Strategy	Problem it Solves	Key Idea	Benefit
Bigger Model (RoBERTa)	Base model lacks subtlety	Use richer embeddings	Finer sense distinctions
[CLS] + token concat	Token-only lacks sentence context	Combine local + global info	Disambiguation easier
Fine-tune embeddings	Pretrained embeddings not task-specific	Adjust model on WSD data	Task-specialized embeddings
Inject gloss definitions	Sentence alone ambiguous	Feed in sense gloss	Explicit clue matching


Original vs Modified ProtoBox Accuracy (WSD)
Accuracy
1.0
0.8
0.6
0.4
0.2
0.0
Train Accuracy
Test Accuracy
Original ProtoBox
Modified ProtoBox

Knowledge-Graph-Enhanced Neural WSD

A powerful next-step is to integrate lexical knowledge via a graph-based neural model. For example, **EWISER** (Bevilacqua and Navigli 2020) embeds WordNet's graph structure into a neural WSD system aclweb.org. In EWISER each synset (sense) is a node connected by hypernym/antonym edges, and pre-trained synset vectors (from WordNet or BabelNet) serve as features. A transformer (e.g. BERT) computes a contextual vector for the target word, and a Graph Neural Network (GNN) propagates information over the synset graph. The model then scores each candidate sense by combining the context embedding with the graph-derived synset embeddings.

Or in simple words we can say -
Using a smart learning model (neural network) that gets help from a **knowledge graph** to better understand **which meaning** of a word is being used in a sentence.


Knowledge-Graph-Enhanced Neural WSD
I spent a lot of time in the bank of the river.
Sense: *river* bank
neural WSD
river
bank
financial institution

How it works ?

Encode the Context with BERT

Let **C** = $\{\boldsymbol{w}_1, \boldsymbol{w}_2, \dots, \boldsymbol{w}_{t-1}, \boldsymbol{w}_t, \boldsymbol{w}_{t+1}, \dots, \boldsymbol{w}_n\}$ where $\boldsymbol{w}_t$ is the target word ("bank").

We pass the context through a pre-trained transformer (e.g. BERT):

$$H = \text{BERT}(\boldsymbol{w}_1, \ldots, \boldsymbol{w}_2) \in \boldsymbol{R}^{n*d}$$

Let $\boldsymbol{h}_t \in \boldsymbol{R}^d$ be the contextual embedding for $\boldsymbol{w}_t$

Sense Graph and Sense Embeddings

Let:

- $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be the **sense graph** (WordNet or BabelNet), where:
  - $\mathcal{V}$ = all synsets (word senses)
  - $\mathcal{E}$ = edges like hypernym, hyponym, related-to, etc.

Each node (sense) $\boldsymbol{s}_i \in \mathcal{V}$ has an initial **sense embedding** $\mathbf{e}_{\boldsymbol{s}_i}^{(0)} \in \boldsymbol{R}^d$ These can be:

- Pre-trained (e.g., gloss or sense2vec vectors)
- Learned from scratch

Graph Neural Network on the Sense Graph

We now refine each sense embedding using a **GNN** over $\mathcal{G}$

Let’s define one **Graph Convolutional Layer** (GCN):

$$\mathbf{e}_{s_i}^{(l+1)} = \sigma\left(\sum_{s_j \in \mathcal{N}(s_i)} \frac{1}{\sqrt{|\mathcal{N}(s_i)||\mathcal{N}(s_j)|}} \mathbf{W}^{(l)} \mathbf{e}_{s_j}^{(l)}\right)$$

Where:

- $\mathcal{N}(\boldsymbol{s}_i)$ is the set of neighbors of $\boldsymbol{s}_i$
- $\mathbf{W}^{(l)}$ is a trainable weight matrix
- σ is an activation function (e.g., ReLU)

After $\mathcal{L}$ layers, we get **context-independent refined sense embeddings**:

These encode graph structure — e.g., river senses of *bank* will be closer in embedding space than finance senses.

We compute a **compatibility score** between the context vector $\boldsymbol{h}_t$ and each candidate sense $\boldsymbol{s}_i \in \boldsymbol{S}_w$ via:

$$\text{score}(s_i) = \cos(\mathbf{h}_t, \mathbf{e}_{s_i}^{(L)}) = \frac{\mathbf{h}_t^\top \mathbf{e}_{s_i}^{(L)}}{\|\mathbf{h}_t\| \|\mathbf{e}_{s_i}^{(L)}\|}$$

Or, for learnability and richer modeling, we use a **bilinear scoring function**:

$$\text{score}(\boldsymbol{s}_i) = \mathbf{h}_t^{\mathrm{T}} \, \mathbf{W}_s \, \mathbf{e}_{s_i}^{(L)} + \ b_s$$

Where:

- $\mathbf{W}_s \in \ \boldsymbol{R}^{d*d}$ is a learned matrix
- $b_s \in R$ is a bias term

Predict a Probability Distribution Over Senses

We normalize the scores over all candidate senses:

$$P(s_i \mid C, w) = \frac{\exp(\text{score}(s_i))}{\sum_{s_j \in \mathcal{S}_w} \exp(\text{score}(s_j))}$$

This gives a softmax distribution — a **context-conditioned probability over senses**.

Training (Loss Function)

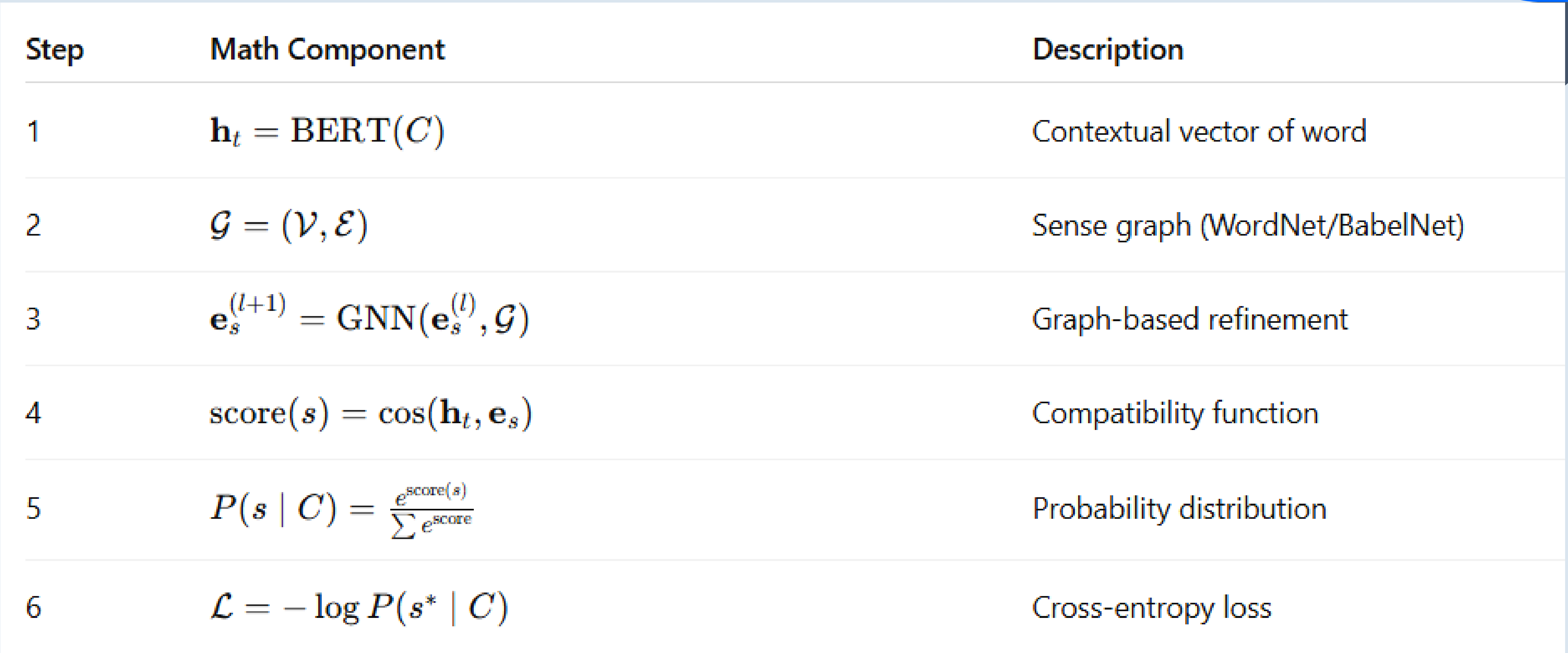
If you have supervised data (e.g., labeled examples of "bank" in context), the training objective is **cross-entropy loss**:

$$\mathcal{L} = -\log P(s^* \mid C, w)$$

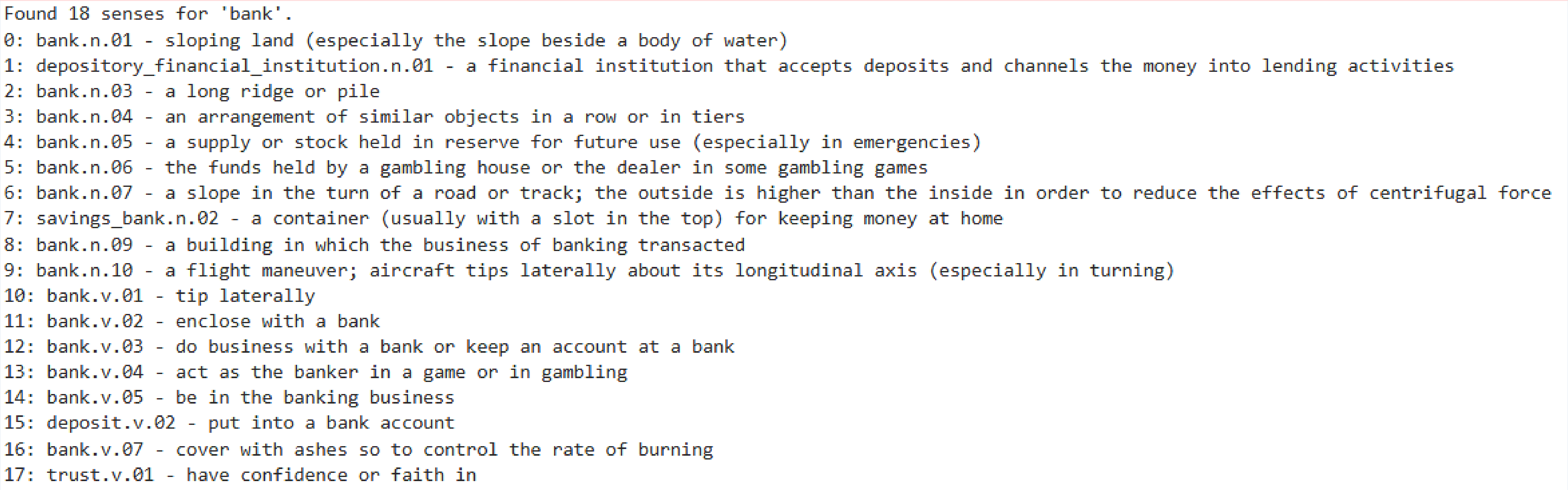
where $\boldsymbol{s}^*$ is the correct sense label.

How It Differentiates Senses

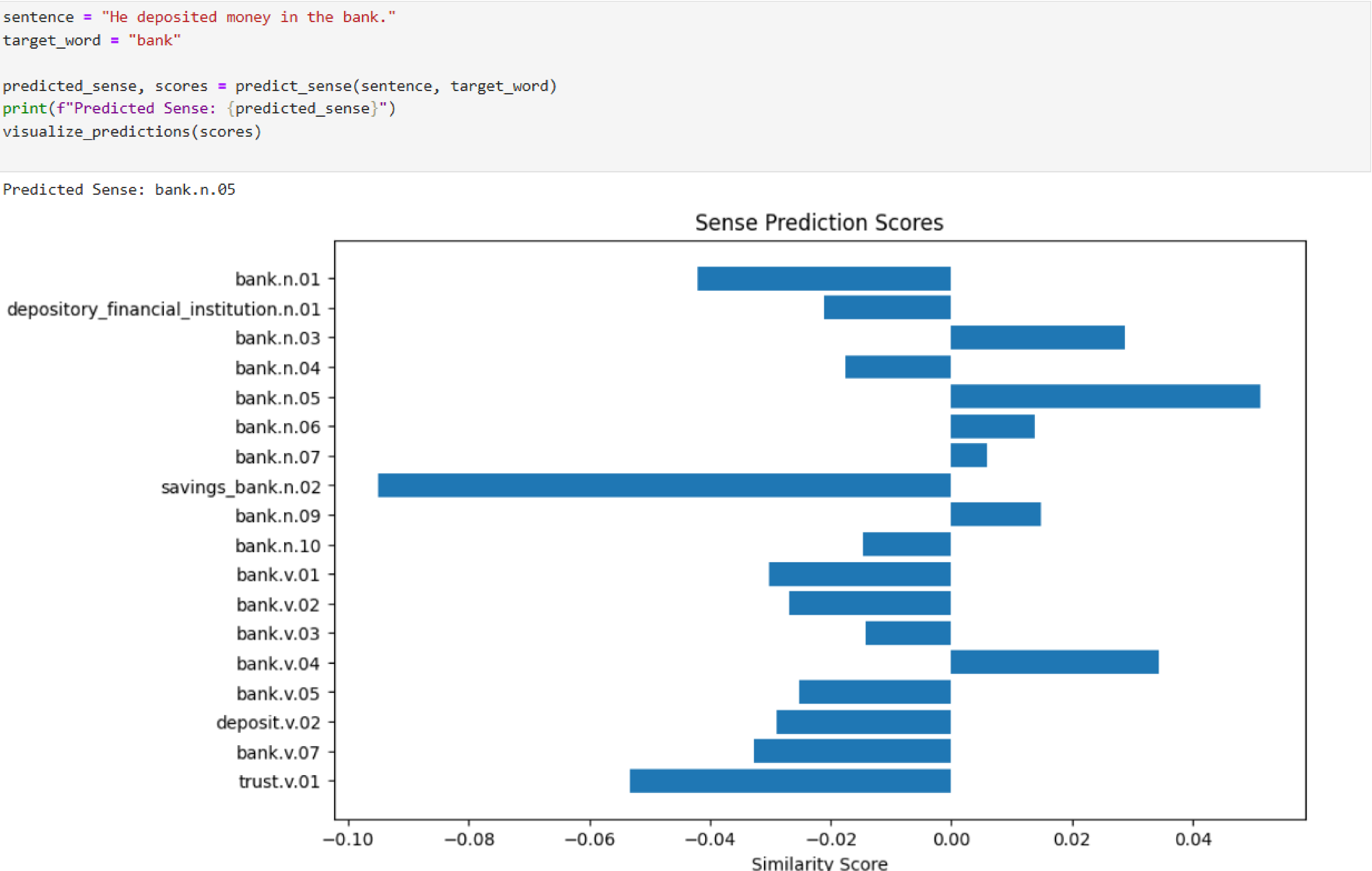
1. **Contextual Encoding**:
    1. $\boldsymbol{h}_t$ encodes *how* "bank" is used: nearby words like "money", "loan" vs. "river", "shore" influence its representation.
    2. This disambiguates polysemy by using the **surrounding sentence structure**.
2. **Graph Refinement**:
    1. Sense embeddings $\mathbf{e}_{s_i}^{(L)}$ encode semantic relationships: senses related via the graph (like hypernyms or related words) become **closer in the embedding space**.
    2. So “river bank” and “floodplain” will have similar vectors, and “savings bank” will be farther.
3. **Scoring**:
    1. The scoring function aligns the **context vector** with the **refined sense vectors**. The sense whose vector most closely matches the context vector (via cosine similarity or bilinear interaction) is chosen.



Step	Math Component	Description
1	$\mathbf{h}_t = \mathrm{BERT}(C)$	Contextual vector of word
2	$\mathcal{G} = (\mathcal{V}, \mathcal{E})$	Sense graph (WordNet/BabelNet)
3	$\mathbf{e}_s^{(l+1)} = \mathrm{GNN}(\mathbf{e}_s^{(l)}, \mathcal{G})$	Graph-based refinement
4	$\mathrm{score}(s) = \cos(\mathbf{h}_t, \mathbf{e}_s)$	Compatibility function
5	$P(s \mid C) = \frac{e^{\mathrm{score}(s)}}{\sum e^{\mathrm{score}}}$	Probability distribution
6	$\mathcal{L} = -\log P(s^* \mid C)$	Cross-entropy loss



```
Found 18 senses for 'bank'.
0: bank.n.01 - sloping land (especially the slope beside a body of water)
1: depository_financial_institution.n.01 - a financial institution that accepts deposits and channels the money into lending activities
2: bank.n.03 - a long ridge or pile
3: bank.n.04 - an arrangement of similar objects in a row or in tiers
4: bank.n.05 - a supply or stock held in reserve for future use (especially in emergencies)
5: bank.n.06 - the funds held by a gambling house or the dealer in some gambling games
6: bank.n.07 - a slope in the turn of a road or track; the outside is higher than the inside in order to reduce the effects of centrifugal force
7: savings_bank.n.02 - a container (usually with a slot in the top) for keeping money at home
8: bank.n.09 - a building in which the business of banking transacted
9: bank.n.10 - a flight maneuver; aircraft tips laterally about its longitudinal axis (especially in turning)
10: bank.v.01 - tip laterally
11: bank.v.02 - enclose with a bank
12: bank.v.03 - do business with a bank or keep an account at a bank
13: bank.v.04 - act as the banker in a game or in gambling
14: bank.v.05 - be in the banking business
15: deposit.v.02 - put into a bank account
16: bank.v.07 - cover with ashes so to control the rate of burning
17: trust.v.01 - have confidence or faith in
```



```
sentence = "He deposited money in the bank."
target_word = "bank"

predicted_sense, scores = predict_sense(sentence, target_word)
print(f"Predicted Sense: {predicted_sense}")
visualize_predictions(scores)
```

```
Predicted Sense: bank.n.05
```

Mathematical Modifications

Context Encoding — $\boldsymbol{h}_c$

We first use a pre-trained language model (like BERT) to encode the entire sentence where the ambiguous word occurs.

Let the sentence be e.g :
“He went to the bank to deposit his paycheck”.

Processing:

- Tokenize and feed into BERT:
- Tokens = $\{\boldsymbol{w}_1, \boldsymbol{w}_2, \ldots, \boldsymbol{w}_{t-1}, \boldsymbol{w}_t, \boldsymbol{w}_{t+1}, \ldots, \boldsymbol{w}_n\}$
- Use mean pooling over BERT outputs:
- $\boldsymbol{h}_c = \frac{1}{n}\sum_{i=1}^{n} \mathbf{e}(\boldsymbol{w}_i)$ where $\mathbf{e}(\boldsymbol{w}_i)$ is the contextualized embedding for token $\boldsymbol{w}_i$.

This vector $\boldsymbol{h}_c$ captures the **overall meaning** of the sentence.

Gloss Encoding — $\boldsymbol{g}_s$

Each possible *sense* of the word "bank" has a **definition (gloss)**:
Examples:
- **bank#1**: "financial institution that accepts deposits"
- **bank#2**: "land alongside a river"

For each gloss s, we compute:
$\boldsymbol{g}_s$ = BERT($\boldsymbol{gloss}_s$) Typically also using mean pooling over token outputs, like:

$$\mathbf{g}_s = \frac{1}{m}\sum_{j=1}^{m}\mathbf{e}(w_j^{(s)})$$

where $w_j^{(s)}$ are tokens in the gloss for sense s.
So now, for each sense s, we have a vector $\boldsymbol{g}_s$ describing its meaning.

Similarity Scoring — score(s)

We now compute how close the sentence meaning is to each gloss:
Core score:

$$\text{score}_{\cos}(s) = \cos(\mathbf{h}_c, \mathbf{g}_s) = \frac{\mathbf{h}_c \cdot \mathbf{g}_s}{\|\mathbf{h}_c\| \cdot \|\mathbf{g}_s\|}$$

This gives us a similarity between 0 and 1 (or -1 to 1), indicating semantic match.

Add Lexical Overlap (Optional) — Lesk

To improve interpretability and boost rare senses, we optionally use a simple **Lesk overlap score**:

Let:

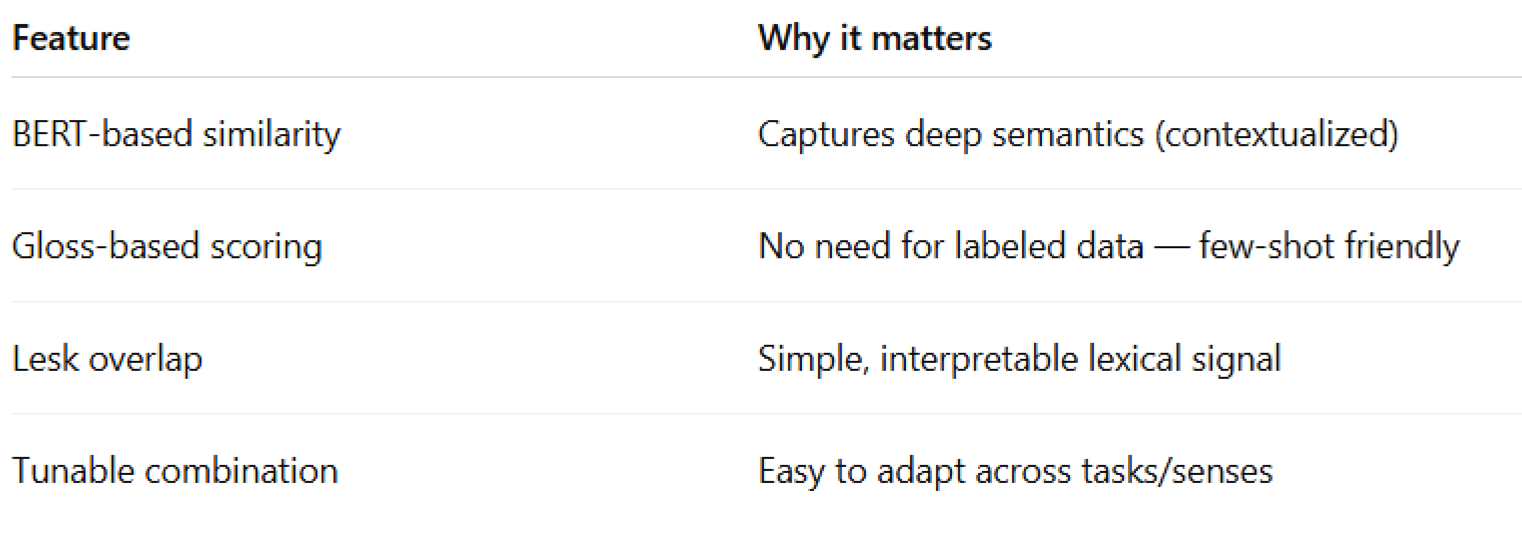
- ContextWords(c) = $\{\boldsymbol{w}_i\}$
- GlossWords(s) = $\{\boldsymbol{w}_j^{(s)}\}$

Then:

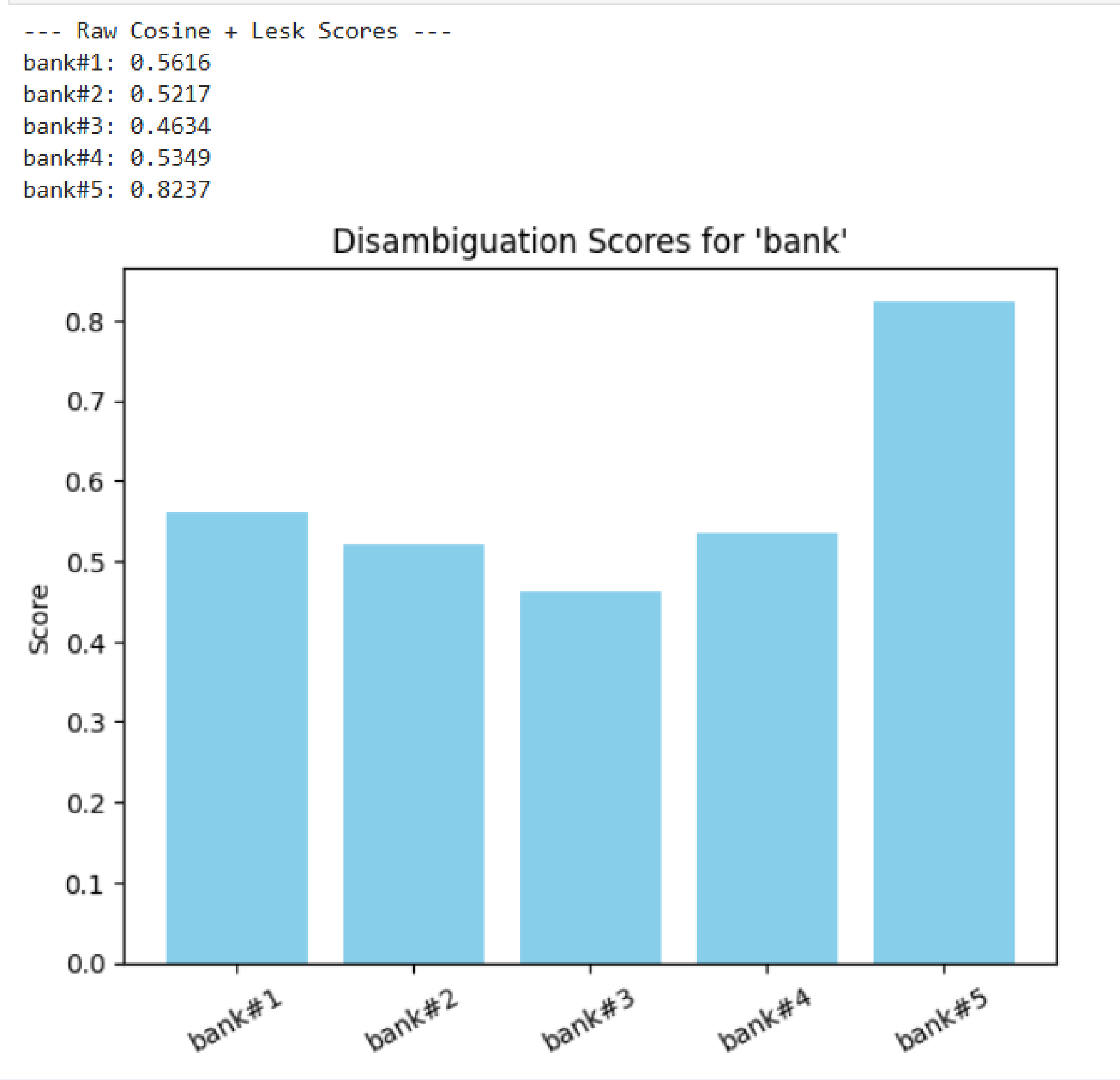
Lesk(s,c) = | ContextWords(c) ∩ GlossWords(s) |

score(s) = cos$(\boldsymbol{h}_c,\ \boldsymbol{g}_s)$ + $\alpha\ .\ Lesk(s, c)$

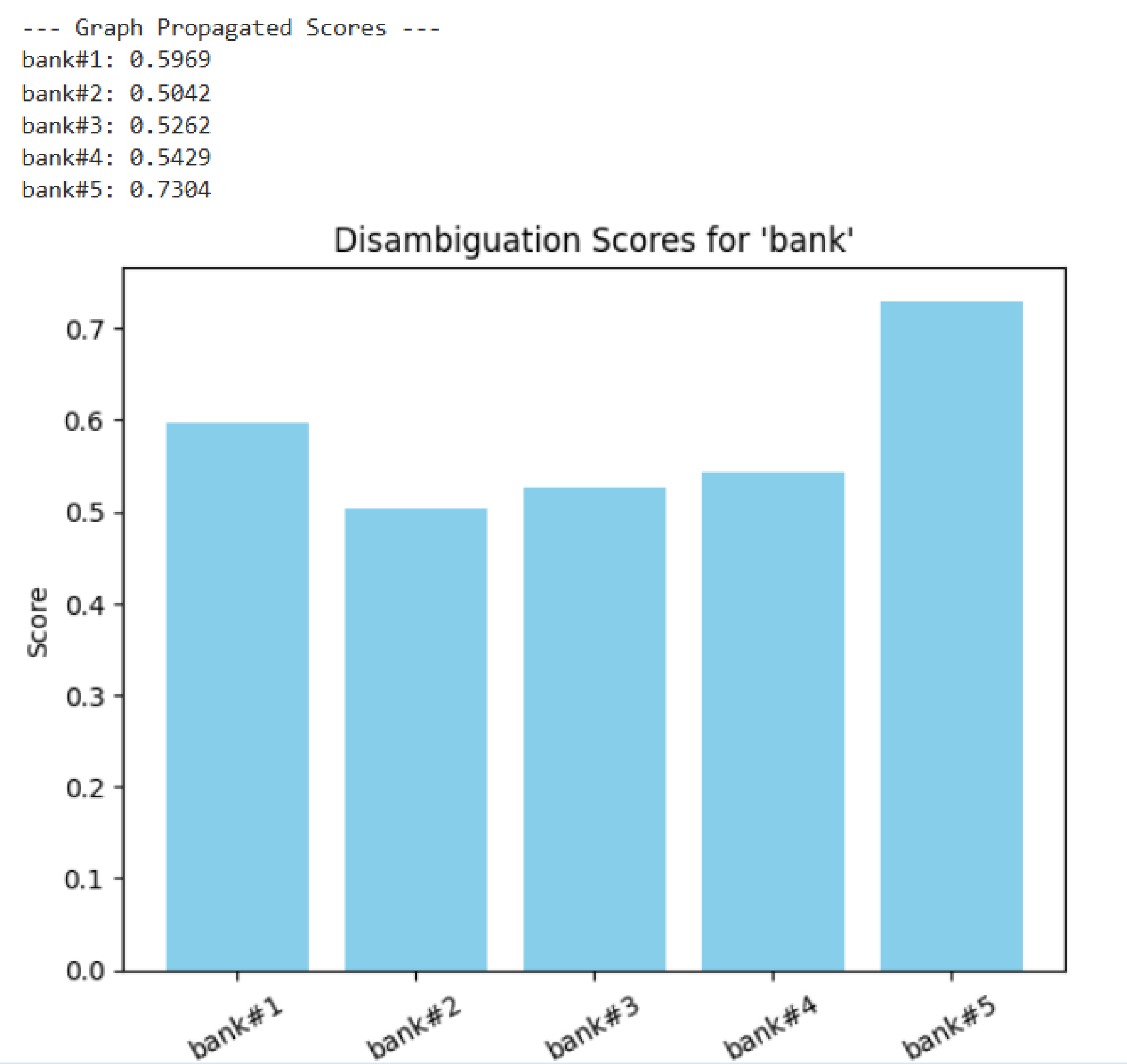
Where $\alpha\ \in\ \boldsymbol{R}_{\succ=0}$



Feature	Why it matters
BERT-based similarity	Captures deep semantics (contextualized)
Gloss-based scoring	No need for labeled data — few-shot friendly
Lesk overlap	Simple, interpretable lexical signal
Tunable combination	Easy to adapt across tasks/senses



```
--- Raw Cosine + Lesk Scores ---
bank#1: 0.5616
bank#2: 0.5217
bank#3: 0.4634
bank#4: 0.5349
bank#5: 0.8237
```



```
--- Graph Propagated Scores ---
bank#1: 0.5969
bank#2: 0.5042
bank#3: 0.5262
bank#4: 0.5429
bank#5: 0.7304
```

References

Word2Vec for WSD:
Huang, E. H., Socher, R., Manning, C. D., & Ng, A. Y. (2012). *"Improving Word Representations via Global Context and Multiple Word Prototypes."*

Context-Aware Similarity Measures:
Melamud, O., Goldberger, J., & Dagan, I. (2016). *"context2vec: Learning Generic Context Embedding with Bidirectional LSTM."* Proceedings of The 20th SIGNLL Conference on Computational Natural Language Learning.
Distance Weighted Cosine Similarity Measure for Text Classification Baoli Li and Liping Han

Word Sense Disambiguation as a Game of Neurosymbolic Darts
Tiansi Dong, Rafet Sifa

Learning Contextualized Box Embeddings with Prototypical Networks
Kohei Oda, Kiyoaki Shirai, Natthawut Kertkeidkachorn

Breaking Through the 80% Glass Ceiling: Raising the State of the Art in Word Sense Disambiguation by Incorporating Knowledge Graph Information
Michele Bevilacqua and Roberto Navigli

Codes Link :click here

Thank you

Ujjwal Gupta

9596357619

ujjwal.gupta.cd.mat22@itbhu.ac.in