Lab 9.2 – Working with Pre-trained Word Embeddings (Word2Vec & GloVe)

Lab9_WordEmbeddings_N.Rukmini_2403a52402.ipynb

STEP 1 — Import Required Libraries

```
# Install gensim library if not already installed
!pip install gensim

# Used to load pre-trained word embeddings like Word2Vec / GloVe
import gensim.downloader as api

# Used for mathematical operations on vectors
import numpy as np

# Used to store similarity results in table format
import pandas as pd

# Used for visualization of embeddings
import matplotlib.pyplot as plt

# Used for dimensionality reduction (optional visualization)
from sklearn.decomposition import PCA
```

```
Collecting gensim
  Downloading gensim-4.4.0-cp312-cp312-manylinux_2_24_x86_64.manylinux_2_28_x86_64.whl.metadata (8.4 kB)
Requirement already satisfied: numpy>=1.18.5 in /usr/local/lib/python3.12/dist-packages (from gensim) (2.0.2)
Requirement already satisfied: scipy>=1.7.0 in /usr/local/lib/python3.12/dist-packages (from gensim) (1.16.3)
Requirement already satisfied: smart_open>=1.8.1 in /usr/local/lib/python3.12/dist-packages (from gensim) (7.5.0)
Requirement already satisfied: wrapt in /usr/local/lib/python3.12/dist-packages (from smart_open>=1.8.1->gensim) (2.1.0)
Downloading gensim-4.4.0-cp312-cp312-manylinux_2_24_x86_64.manylinux_2_28_x86_64.whl (27.9 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 27.9/27.9 MB 55.4 MB/s eta 0:00:00
Installing collected packages: gensim
Successfully installed gensim-4.4.0
```

STEP 2 — Load Pre-trained Embeddings

```
# Load Google News Word2Vec model
model = api.load("word2vec-google-news-300")

# Print vocabulary size
print("Vocabulary Size:", len(model.key_to_index))

# Display vector for sample word
print("\nVector for word 'king':\n")
print(model['king'])
```

```
 -1.77734375e-01  8.59375000e-02 -2.18505859e-02  2.05078125e-02
 -1.39648438e-01  2.51464844e-02  1.38671875e-01 -1.05468750e-01
  1.38671875e-01  8.88671875e-02 -7.51953125e-02 -2.13623047e-02
```

```
  9.96093750e-02 -2.72216797e-02  1.96533203e-02  4.27246094e-02
 -2.46093750e-01  6.39648438e-02 -2.25585938e-01 -1.68945312e-01
  2.89916992e-03  8.20312500e-02  3.41796875e-01  4.32128906e-02
  1.32812500e-01  1.42578125e-01  7.61718750e-02  5.98144531e-02
 -1.19140625e-01  2.74658203e-03 -6.29882812e-02 -2.72216797e-02
 -4.82177734e-03 -8.20312500e-02 -2.49023438e-02 -4.00390625e-01
 -1.06933594e-01  4.24804688e-02  7.76367188e-02 -1.16699219e-01
  7.37304688e-02 -9.22851562e-02  1.07910156e-01  1.58203125e-01
  4.24804688e-02  1.26953125e-01  3.61328125e-02  2.67578125e-01
 -1.01074219e-01 -3.02734375e-01 -5.76171875e-02  5.05371094e-02
  5.26428223e-04 -2.07031250e-01 -1.38671875e-01 -8.97216797e-03
 -2.78320312e-02 -1.41601562e-01  2.07031250e-01 -1.58203125e-01
  1.27929688e-01  1.49414062e-01 -2.24609375e-02 -8.44726562e-02
  1.22558594e-01  2.15820312e-01 -2.13867188e-01 -3.12500000e-01
 -3.73046875e-01  4.08935547e-03  1.07421875e-01  1.06933594e-01
  7.32421875e-02  8.97216797e-03 -3.88183594e-02 -1.29882812e-01
  1.49414062e-01 -2.14843750e-01 -1.83868408e-03  9.91210938e-02
  1.57226562e-01 -1.14257812e-01 -2.05078125e-01  9.91210938e-02
  3.69140625e-01 -1.97265625e-01  3.54003906e-02  1.09375000e-01
  1.31835938e-01  1.66992188e-01  2.35351562e-01  1.04980469e-01
 -4.96093750e-01 -1.64062500e-01 -1.56250000e-01 -5.22460938e-02
  1.03027344e-01  2.43164062e-01 -1.88476562e-01  5.07812500e-02
 -9.37500000e-02 -6.68945312e-02  2.27050781e-02  7.61718750e-02
  2.89062500e-01  3.10546875e-01 -5.37109375e-02  2.28515625e-01
  2.51464844e-02  6.78710938e-02 -1.21093750e-01 -2.15820312e-01
 -2.73437500e-01 -3.07617188e-02 -3.37890625e-01  1.53320312e-01
  2.33398438e-01 -2.08007812e-01  3.73046875e-01  8.20312500e-02
  2.51953125e-01 -7.61718750e-02 -4.66308594e-02 -2.23388672e-02
  2.99072266e-02 -5.93261719e-02 -4.66918945e-03 -2.44140625e-01
 -2.09960938e-01 -2.87109375e-01 -4.54101562e-02 -1.77734375e-01
 -2.79296875e-01 -8.59375000e-02  9.13085938e-02  2.51953125e-01]
```

STEP 3 — Word Similarity

```
word_pairs = [
    ("doctor", "nurse"),
    ("cat", "dog"),
    ("car", "bus"),
    ("king", "queen"),
    ("student", "teacher"),
    ("sun", "moon"),
    ("computer", "laptop"),
    ("apple", "banana"),
    ("river", "water"),
    ("man", "woman")
]

results = []

for w1, w2 in word_pairs:
    similarity = model.similarity(w1, w2)
    results.append([w1, w2, similarity])

df = pd.DataFrame(results, columns=["Word 1", "Word 2", "Similarity"])
print(df)
```

```
     Word 1   Word 2  Similarity
0    doctor    nurse    0.631952
1       cat      dog    0.760946
2       car      bus    0.469337
3      king    queen    0.651096
4   student  teacher    0.630137
5       sun     moon    0.426283
6  computer   laptop    0.664049
7     apple   banana    0.531841
8     river    water    0.576898
9       man    woman    0.766401
```

STEP 4 — Nearest Neighbor Exploration

```
words = ["king", "university", "computer", "doctor", "india"]

for word in words:
    print(f"\nTop similar words to '{word}':")
    print(model.most_similar(word, topn=5))
```

```
Top similar words to 'king':
[('kings', 0.7138045430183411), ('queen', 0.6510956883430481), ('monarch', 0.6413194537162781), ('crown_prince', 0.6204220056533

Top similar words to 'university':
[('universities', 0.7003918886184692), ('faculty', 0.6780907511711121), ('unversity', 0.6758289933204651), ('undergraduate', 0.6

Top similar words to 'computer':
[('computers', 0.7979379892349243), ('laptop', 0.6640493273735046), ('laptop_computer', 0.6548868417739868), ('Computer', 0.6473

Top similar words to 'doctor':
[('physician', 0.7806021571159363), ('doctors', 0.747657299041748), ('gynecologist', 0.6947518587112427), ('surgeon', 0.67933982

Top similar words to 'india':
[('indian', 0.6967039704322815), ('usa', 0.6836211085319519), ('pakistan', 0.681516706943512), ('chennai', 0.6675503253936768),
```

STEP 5 — Word Analogy Tasks

```
print("king - man + woman =", model.most_similar(positive=['king','woman'], negative=['man'], topn=1))

print("paris - france + india =", model.most_similar(positive=['paris','india'], negative=['france'], topn=1))

print("teacher - school + hospital =", model.most_similar(positive=['teacher','hospital'], negative=['school'], topn=1))
```

```
king - man + woman = [('queen', 0.7118193507194519)]
paris - france + india = [('chennai', 0.5442505478858948)]
teacher - school + hospital = [('Hospital', 0.6331106424331665)]
```

STEP 6 — Visualization

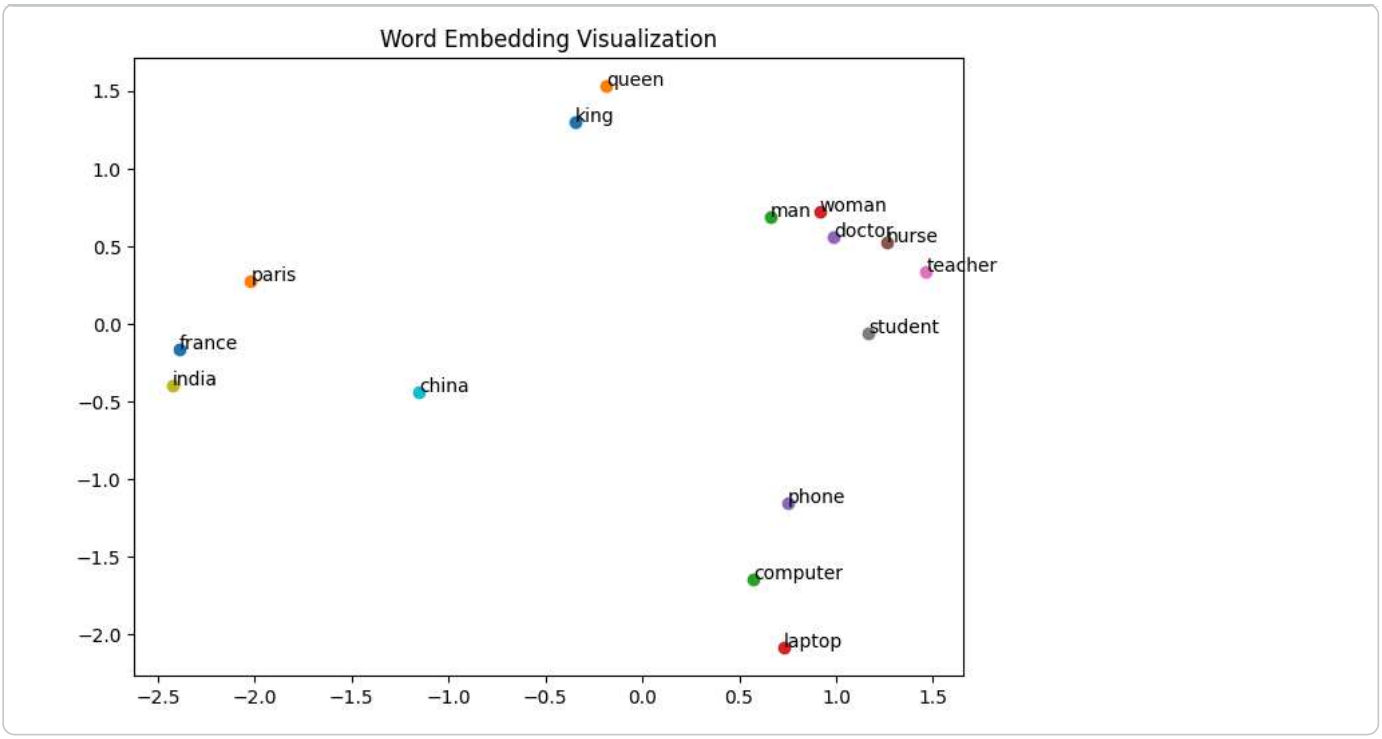
```
words = ["king","queen","man","woman","doctor","nurse","teacher","student",
         "india","china","france","paris","computer","laptop","phone"]

word_vectors = [model[w] for w in words]

# Reduce dimensions
pca = PCA(n_components=2)
result = pca.fit_transform(word_vectors)

# Plot
plt.figure(figsize=(8,6))
for i, word in enumerate(words):
    plt.scatter(result[i,0], result[i,1])
    plt.text(result[i,0], result[i,1], word)

plt.title("Word Embedding Visualization")
plt.show()
```



STEP 7— Reflection

```
#Word embeddings learn semantic meaning by converting words into numerical vectors. Similar words appear closer in vector space
```

STEP 8 — Model / Dataset Description

✅ Word2Vec

Source: Google News dataset

Size: ~3 million words

Dimension: 300

Trained using neural networks to predict surrounding words

Theory Questions Answers

What are word embeddings?

Word embeddings are numerical vector representations of words that capture their meaning and relationships.

Difference between One-Hot and Embeddings? One-Hot Embeddings Large sparse vectors Dense vectors No semantic meaning Captures word meaning High memory usage Efficient

What is Word2Vec?

Word2Vec is a neural network model that learns word meaning by predicting surrounding words.

What is GloVe?

GloVe learns word meaning using global word co-occurrence statistics.

Why Cosine Similarity?

Because it measures similarity based on angle between vectors, not magnitude.

Word Analogy Example

```
print("king - man + woman = queen")
```

```
king - man + woman = queen
```

Real-Life Applications

Chatbots