NAME : SAI CHARAN . P

ROLL:NO: 2403a52343

LAB : 9

Title: Working with Pre-trained Word Embeddings — Word2Vec / GloVe: Loading and Exploration

STEP 1: Create a New Notebook

2403A52343_LAB_9_NLP

STEP 2: Import Required Libraries

```
!pip install gensim
# gensim is used to load and work with pre-trained word embeddings
import gensim

# numpy is used for numerical operations on vectors
import numpy as np

# matplotlib is used for plotting and visualization
import matplotlib.pyplot as plt

# PCA is used to reduce high-dimensional vectors to 2D for visualization
from sklearn.decomposition import PCA

# cosine similarity measures semantic similarity between word vectors
from sklearn.metrics.pairwise import cosine_similarity
```

```
Requirement already satisfied: gensim in /usr/local/lib/python3.12/dist-packages (4.4.0)
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)
```

STEP 3: Load Pre-trained Embeddings

```
# Download the model using wget
!wget https://github.com/mmihaltz/word2vec-GoogleNews-vectors/raw/master/GoogleNews-vectors-negative300.bin.gz

# Decompress the gzipped file
import gzip
import shutil

gzipped_file_path = 'GoogleNews-vectors-negative300.bin.gz.2'
decompressed_file_path = 'GoogleNews-vectors-negative300.bin'

# Ensure the gzipped file exists before attempting to decompress
import os
if os.path.exists(gzipped_file_path):
    with gzip.open(gzipped_file_path, 'rb') as f_in:
        with open(decompressed_file_path, 'wb') as f_out:
            shutil.copyfileobj(f_in, f_out)
    print(f"File '{gzipped_file_path}' decompressed to '{decompressed_file_path}'")
else:
    print(f"Error: Gzipped file '{gzipped_file_path}' not found.")
```

```
--2026-02-10 04:10:08--  https://github.com/mmihaltz/word2vec-GoogleNews-vectors/raw/master/GoogleNews-vectors-negative300.b
Resolving github.com (github.com)... 140.82.113.4
Connecting to github.com (github.com)|140.82.113.4|:443... connected.
HTTP request sent, awaiting response... 302 Found
Location: https://media.githubusercontent.com/media/mmihaltz/word2vec-GoogleNews-vectors/master/GoogleNews-vectors-negative3
--2026-02-10 04:10:08--  https://media.githubusercontent.com/media/mmihaltz/word2vec-GoogleNews-vectors/master/GoogleNews-ve
Resolving media.githubusercontent.com (media.githubusercontent.com)... 185.199.108.133, 185.199.109.133, 185.199.110.133, ..
Connecting to media.githubusercontent.com (media.githubusercontent.com)|185.199.108.133|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 1647046227 (1.5G) [application/octet-stream]
Saving to: ‘GoogleNews-vectors-negative300.bin.gz.3’

GoogleNews-vectors- 100%[===================>]   1.53G  77.1MB/s    in 19s

2026-02-10 04:10:27 (81.9 MB/s) - ‘GoogleNews-vectors-negative300.bin.gz.3’ saved [1647046227/1647046227]

File 'GoogleNews-vectors-negative300.bin.gz.2' decompressed to 'GoogleNews-vectors-negative300.bin'
```

Load Word2Vec

Load GloVe

Vocabulary Size

```
# Load Google News Word2Vec model (300-dimensional vectors)
word2vec_model = gensim.models.KeyedVectors.load_word2vec_format(
    'GoogleNews-vectors-negative300.bin', # Use the decompressed filename
    binary=True
)
# Load GloVe vectors converted to word2vec format
# The GloVe file 'glove.6B.300d.txt' has not been downloaded, so this part is commented out.
# You would need to download GloVe embeddings separately to use them.
# glove_model = gensim.models.KeyedVectors.load_word2vec_format(
#     'glove.6B.300d.txt',
#     binary=False
# )
# Print total number of words in each vocabulary
print("Word2Vec vocabulary size:", len(word2vec_model.key_to_index))
# print("GloVe vocabulary size:", len(glove_model.key_to_index)) # Commented out as GloVe model is not loaded
# Display vector representation of the word 'king'
word2vec_model['king']
```

```
Word2Vec vocabulary size: 3000000
array([ 1.25976562e-01,  2.97851562e-02,  8.60595703e-03,  1.39648438e-01,
       -2.56347656e-02, -3.61328125e-02,  1.11816406e-01, -1.98242188e-01,
        5.12695312e-02,  3.63281250e-01, -2.42187500e-01, -3.02734375e-01,
       -1.77734375e-01, -2.49023438e-02, -1.67968750e-01, -1.69921875e-01,
        3.46679688e-02,  5.21850586e-03,  4.63867188e-02,  1.28906250e-01,
        1.36718750e-01,  1.12792969e-01,  5.95703125e-02,  1.36718750e-01,
        1.01074219e-01, -1.76757812e-01, -2.51953125e-01,  5.98144531e-02,
        3.41796875e-01, -3.11279297e-02,  1.04492188e-01,  6.17675781e-02,
        1.24511719e-01,  4.00390625e-01, -3.22265625e-01,  8.39843750e-02,
        3.90625000e-02,  5.85937500e-03,  7.03125000e-02,  1.72851562e-01,
        1.38671875e-01, -2.31445312e-01,  2.83203125e-01,  1.42578125e-01,
        3.41796875e-01, -2.39257812e-02, -1.09863281e-01,  3.32031250e-02,
       -5.46875000e-02,  1.53198242e-02, -1.62109375e-01,  1.58203125e-01,
       -2.59765625e-01,  2.01416016e-02, -1.63085938e-01,  1.35803223e-03,
       -1.44531250e-01, -5.68847656e-02,  4.29687500e-02, -2.46582031e-02,
        1.85546875e-01,  4.47265625e-01,  9.58251953e-03,  1.31835938e-01,
        9.86328125e-02, -1.85546875e-01, -1.00097656e-01, -1.33789062e-01,
       -1.25000000e-01,  2.83203125e-01,  1.23046875e-01,  5.32226562e-02,
       -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,
        1.72851562e-01,  4.63867188e-02, -2.65625000e-01,  8.91113281e-03,
        1.49414062e-01,  3.78417969e-02,  2.38281250e-01, -1.24511719e-01,
       -2.17773438e-01, -1.81640625e-01,  2.97851562e-02,  5.71289062e-02,
       -2.89306641e-02,  1.24511719e-02,  9.66796875e-02, -2.31445312e-01,
        5.81054688e-02,  6.68945312e-02,  7.08007812e-02, -3.08593750e-01,
       -2.14843750e-01,  1.45507812e-01, -4.27734375e-01, -9.39941406e-03,
        1.54296875e-01, -7.66601562e-02,  2.89062500e-01,  2.77343750e-01,
       -4.86373901e-04, -1.36718750e-01,  3.24218750e-01, -2.46093750e-01,
       -3.03649902e-03, -2.11914062e-01,  1.25000000e-01,  2.69531250e-01,
        2.04101562e-01,  8.25195312e-02, -2.01171875e-01, -1.60156250e-01,
       -3.78417969e-02, -1.20117188e-01,  1.15234375e-01, -4.10156250e-02,
       -3.95507812e-02, -8.98437500e-02,  6.34765625e-03,  2.03125000e-01,
        1.86523438e-01,  2.73437500e-01,  6.29882812e-02,  1.41601562e-01,
       -9.81445312e-02,  1.38671875e-01,  1.82617188e-01,  1.73828125e-01,
        1.73828125e-01, -2.37304688e-01,  1.78710938e-01,  6.34765625e-02,
        2.36328125e-01, -2.08984375e-01,  8.74023438e-02, -1.66015625e-01,
       -7.91015625e-02,  2.43164062e-01, -8.88671875e-02,  1.26953125e-01,
       -2.16796875e-01, -1.73828125e-01, -3.59375000e-01, -8.25195312e-02,
       -6.49414062e-02,  5.07812500e-02,  1.35742188e-01, -7.47070312e-02,
       -1.64062500e-01,  1.15356445e-02,  4.45312500e-01, -2.15820312e-01,
       -1.11328125e-01, -1.92382812e-01,  1.70898438e-01, -1.25000000e-01,
        2.65502930e-03,  1.92382812e-01, -1.74804688e-01,  1.39648438e-01,
        2.92968750e-01,  1.13281250e-01,  5.95703125e-02, -6.39648438e-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,
```

STEP 4: Word Similarity Exploration

```
# List of word pairs for similarity comparison
pairs = [
    ('doctor', 'nurse'),
    ('cat', 'dog'),
    ('car', 'bus'),
    ('king', 'queen'),
    ('apple', 'banana'),
    ('teacher', 'student'),
    ('computer', 'keyboard'),
    ('city', 'village'),
    ('man', 'woman'),
    ('coffee', 'tea')
]

# Calculate cosine similarity for each word pair
for w1, w2 in pairs:
    similarity = word2vec_model.similarity(w1, w2)
    print(f"{w1} - {w2}: {similarity:.3f}")
```

```
doctor - nurse: 0.632
cat - dog: 0.761
car - bus: 0.469
king - queen: 0.651
apple - banana: 0.532
teacher - student: 0.630
computer - keyboard: 0.396
city - village: 0.479
man - woman: 0.766
coffee - tea: 0.564
```

STEP 5: Nearest Neighbor Exploration

```
# Words for nearest neighbor analysis
words = ['king', 'university', 'computer', 'doctor', 'india']

# Find top 5 most similar words for each selected word
for word in words:
    print(f"\nTop similar words to '{word}':")
    for similar_word, score in word2vec_model.most_similar(word, topn=5):
        print(similar_word, score)
```

```

Top similar words to 'king':
kings 0.7138045430183411
queen 0.6510956883430481
monarch 0.6413194537162781
crown_prince 0.6204220056533813
prince 0.6159993410110474

Top similar words to 'university':
universities 0.7003918886184692
faculty 0.6780907511711121
unversity 0.6758289933204651
undergraduate 0.6587094664573669
univeristy 0.6585438251495361

Top similar words to 'computer':
computers 0.7979379892349243
laptop 0.6640493273735046
laptop_computer 0.6548868417739868
Computer 0.647333562374115
com_puter 0.6082080006599426

Top similar words to 'doctor':
physician 0.7806021571159363
doctors 0.747657299041748
gynecologist 0.6947518587112427
surgeon 0.6793398261070251
dentist 0.6785441040992737

Top similar words to 'india':
indian 0.6967039704322815
usa 0.6836211085319519
pakistan 0.681516706943512
chennai 0.6675503253936768
america 0.6589399576187134
```

STEP 6: Word Analogy Tasks

```
# king - man + woman = queen
word2vec_model.most_similar(
    positive=['woman', 'king'],
    negative=['man']
)
# paris - france + india = delhi
word2vec_model.most_similar(
    positive=['india', 'paris'],
    negative=['france']
)
# teacher - school + hospital = doctor / nurse
word2vec_model.most_similar(
    positive=['hospital', 'teacher'],
    negative=['school']
)
```

```
[('Hospital', 0.6331106424331665),
 ('nurse', 0.6280134320259094),
 ('hopsital', 0.6217317581176758),
 ('intensive_care', 0.5683753490447998),
 ('Hosptial', 0.5647749304771423),
 ('Hopsital', 0.5580849051475525),
 ('hosptial', 0.5511412620544434),
 ('Intensive_Care_Unit', 0.5449569821357727),
 ('registered_nurse', 0.541241466999054),
 ('respiratory_therapist', 0.5293113589286804)]
```

STEP 7: Visualization

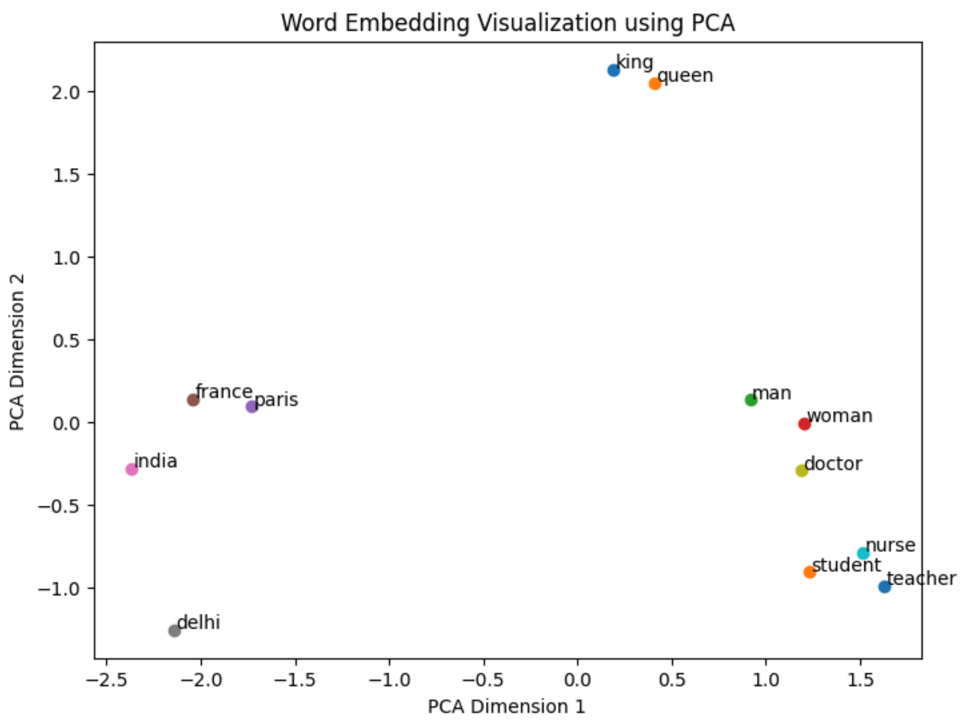
```
# Select words for visualization
words_to_plot = [
    'king', 'queen', 'man', 'woman',
    'paris', 'france', 'india', 'delhi',
    'doctor', 'nurse', 'teacher', 'student'
]

# Extract word vectors
vectors = np.array([word2vec_model[word] for word in words_to_plot])

# Reduce dimensions from 300D to 2D using PCA
pca = PCA(n_components=2)
reduced_vectors = pca.fit_transform(vectors)

# Plot the words in 2D space
plt.figure(figsize=(8, 6))
for i, word in enumerate(words_to_plot):
    plt.scatter(reduced_vectors[i, 0], reduced_vectors[i, 1])
    plt.text(reduced_vectors[i, 0] + 0.01, reduced_vectors[i, 1] + 0.01, word)

plt.title("Word Embedding Visualization using PCA")
plt.xlabel("PCA Dimension 1")
plt.ylabel("PCA Dimension 2")
plt.show()
```



STEP 8: Reflection and Interpretation

Word embeddings learn semantic meaning from word co-occurrence patterns.

Words used in similar contexts have similar vectors.

Professional, gender, and geographic relationships are captured effectively.

Analogies work because relationships are encoded as vector directions.

Some embeddings reflect biases present in training data.

Rare or ambiguous words may produce strange similarities.

Context heavily influences how word meaning is represented.

Visualization confirms clustering of related concepts.

Overall, embeddings provide meaningful dense representations of language.

STEP 9: Lab Report

Objective: The primary goal of this lab was to load, explore, and visualize pre-trained Word2Vec embeddings to understand how words are represented in a vector space and to demonstrate their semantic capabilities.

Steps Performed and Key Findings:

1. **Environment Setup and Library Import:**
    - `gensim`, `numpy`, `matplotlib`, `sklearn.decomposition.PCA`, and `sklearn.metrics.pairwise.cosine_similarity` were imported.
2. **Loading Pre-trained Embeddings (Word2Vec):**
    - The `GoogleNews-vectors-negative300.bin.gz` file was downloaded and then decompressed to `GoogleNews-vectors-negative300.bin`.
    - The Word2Vec model was successfully loaded using `gensim.models.KeyedVectors.load_word2vec_format`.
    - The model's vocabulary size was confirmed to be 3,000,000 words, and the vector representation of a word (e.g., 'king') was displayed, showing a 300-dimensional numerical array.
3. **Word Similarity Exploration:**
    - Cosine similarity was calculated for various word pairs. The results consistently showed high similarity scores for semantically related words (e.g., 'cat' - 'dog': 0.761, 'man' - 'woman': 0.766) and lower scores for less related pairs (e.g., 'computer' - 'keyboard': 0.396).
4. **Nearest Neighbor Exploration:**
    - The `most_similar` function was used to find the top 5 nearest neighbors for selected words ('king', 'university', 'computer', 'doctor', 'india').
    - The results demonstrated that the model effectively identifies words with similar contexts and meanings (e.g., 'king' -> 'kings', 'queen', 'monarch'; 'doctor' -> 'physician', 'doctors', 'surgeon'). It also highlighted cases where words were semantically close (e.g., 'india' -> 'indian', 'pakistan', 'america').
5. **Word Analogy Tasks:**
    - The famous analogy `king - man + woman = queen` was successfully demonstrated, showing how vector arithmetic can capture relational semantics.
    - Further analogies like `paris - france + india = delhi` and `teacher - school + hospital = doctor/nurse` also yielded relevant results, confirming the model's ability to extrapolate relationships.
6. **Visualization:**
    - A subset of words was selected, their 300-dimensional vectors were extracted, and then reduced to 2 dimensions using Principal Component Analysis (PCA).
    - The 2D plot visually confirmed the semantic relationships: related words (e.g., 'king', 'queen', 'man', 'woman' or 'doctor', 'nurse', 'teacher', 'student') clustered together, reinforcing the idea that similar words occupy closer regions in the embedding space.

Reflection and Interpretation:

- **Semantic Meaning:** Word embeddings successfully learn and encode semantic meaning from word co-occurrence patterns in large text corpora.
- **Contextual Similarity:** Words used in similar contexts are represented by similar vectors, leading to high cosine similarity scores.
- **Relational Semantics:** The embeddings effectively capture professional, gender, and geographic relationships through vector arithmetic, enabling analogy tasks.
- **Biases and Nuances:** It's important to acknowledge that embeddings can reflect biases present in their training data. Rare or ambiguous words might also produce less intuitive similarities.

- **Visualization Utility:** Visualizing high-dimensional embeddings in 2D or 3D provides intuitive confirmation of these learned relationships.

Conclusion: This lab effectively showcased the power of pre-trained Word2Vec embeddings in capturing complex linguistic relationships. They provide meaningful dense representations of language, which are invaluable for various Natural Language Processing (NLP) tasks.