```
! pip install gensim
```

```
Requirement already satisfied: gensim in /usr/local/lib/python3.12/dist-packages (4.3.3)
Requirement already satisfied: numpy<2.0,>=1.18.5 in /usr/local/lib/python3.12/dist-packages (from gensim) (1.26.4)
Requirement already satisfied: scipy<1.14.0,>=1.7.0 in /usr/local/lib/python3.12/dist-packages (from gensim) (1.13.1)
Requirement already satisfied: smart-open>=1.8.1 in /usr/local/lib/python3.12/dist-packages (from gensim) (7.3.1)
Requirement already satisfied: wrapt in /usr/local/lib/python3.12/dist-packages (from smart-open>=1.8.1->gensim) (1.17.3)
```

Task 1

```
import numpy as np
import gensim.downloader as api



class WordEmbedder:
    def __init__(self, model_name: str = 'glove-wiki-gigaword-50'):
        print(f"Loading model '{model_name}'")
        self.model = api.load(model_name)
        self.vector_size = self.model.vector_size
        print(f"Model '{model_name}' loaded successfully! Vector size: {self.vector_size}")

    def get_vector(self, word: str):
        if word in self.model.key_to_index:
            return self.model[word]
        else:
            print(f"Warning: '{word}' not in vocabulary (OOV).")
            return None

    def get_similarity(self, word1: str, word2: str):
        if word1 in self.model.key_to_index and word2 in self.model.key_to_index:
            return self.model.similarity(word1, word2)
        else:
            print(f"Warning: One or both words are OOV: '{word1}', '{word2}'.")
            return None

    def get_most_similar(self, word: str, top_n: int = 10):
        if word in self.model.key_to_index:
            return self.model.most_similar(word, topn=top_n)
        else:
            print(f"Warning: '{word}' not in vocabulary (OOV).")
            return []

    def embed_document(self, document: str):
        from nltk.tokenize import word_tokenize
        import nltk
        nltk.download('punkt_tab',quiet=True)

        tokens = word_tokenize(document.lower())
        vectors = []

        for token in tokens:
            if token in self.model.key_to_index:
                vectors.append(self.model[token])

        if not vectors:
            print("No known words in document, returning zero vector.")
            return np.zeros(self.vector_size)

        return np.mean(vectors, axis=0)

```

```
def main():
    embedder = WordEmbedder('glove-wiki-gigaword-50')

    print("\nVector for 'king':")
    print(embedder.get_vector('king'))

    print("\nSimilarity between 'king' and 'queen':", embedder.get_similarity('king', 'queen'))
    print("Similarity between 'king' and 'man':", embedder.get_similarity('king', 'man'))

    print("\nMost similar to 'computer':")
    similar_words = embedder.get_most_similar('computer')
    for word, score in similar_words:
        print(f"{word}: {score:.4f}")
    sentence = "The queen rules the country."
```

```
    print(f"\nEmbedding for sentence: \"{sentence}\"")
    doc_vector = embedder.embed_document(sentence)
    print(doc_vector)
    print("Vector shape:", doc_vector.shape)


if __name__ == "__main__":
    main()

```

```
Loading model 'glove-wiki-gigaword-50'
[==================================================] 100.0% 66.0/66.0MB downloaded
Model 'glove-wiki-gigaword-50' loaded successfully! Vector size: 50

Vector for 'king':
[ 0.50451   0.68607  -0.59517  -0.022801  0.60046  -0.13498  -0.08813
  0.47377  -0.61798  -0.31012  -0.076666  1.493    -0.034189 -0.98173
  0.68229   0.81722  -0.51874  -0.31503  -0.55809   0.66421   0.1961
 -0.13495  -0.11476  -0.30344   0.41177  -2.223    -1.0756   -1.0783
 -0.34354   0.33505   1.9927   -0.04234  -0.64319   0.71125   0.49159
  0.16754   0.34344  -0.25663  -0.8523    0.1661    0.40102   1.1685
 -1.0137   -0.21585  -0.15155   0.78321  -0.91241  -1.6106   -0.64426
 -0.51042 ]

Similarity between 'king' and 'queen': 0.78390425
Similarity between 'king' and 'man': 0.53093773

Most similar to 'computer':
computers: 0.9165
software: 0.8815
technology: 0.8526
electronic: 0.8126
internet: 0.8060
computing: 0.8026
devices: 0.8016
digital: 0.7992
applications: 0.7913
pc: 0.7883

Embedding for sentence: "The queen rules the country."
[ 0.04564168  0.36530998 -0.55974334  0.04014383  0.09655549  0.15623933
 -0.33622834 -0.12495166 -0.01031508 -0.5006717   0.18690467  0.17482166
 -0.268985   -0.03096624  0.36686516  0.29983264  0.01397333 -0.06872118
 -0.3260683  -0.210115    0.16835399 -0.03151734 -0.06204716  0.04301083
 -0.06958768 -1.7792168  -0.54365396 -0.06104483 -0.17618     0.009181
  3.3916333   0.08742473 -0.4675417  -0.213435    0.02391887 -0.04470453
  0.20636833 -0.12902866 -0.28527132 -0.2431805  -0.3114423  -0.03833717
  0.11977985 -0.01418401 -0.37086335  0.22069354 -0.28848937 -0.36188802
 -0.00549529 -0.46997246]
Vector shape: (50,)
```

Task 2 Document embedding

```
sentence = 'The quick brown fox jumps over the lazy dog.'
embedder = WordEmbedder()
doc_vector = embedder.embed_document(sentence)
print(doc_vector)
print("Vector shape:", doc_vector.shape)
```

```
Loading model 'glove-wiki-gigaword-50'
Model 'glove-wiki-gigaword-50' loaded successfully! Vector size: 50
[ 0.018704   -0.01994951 -0.152806   -0.05444079  0.267892    0.27261212
 -0.594531   -0.1579832   0.05074116 -0.22430201 -0.19797495  0.07773571
 -0.18965401  0.23584989  0.2018865  -0.03365446  0.1494343  -0.123215
 -0.77456605 -0.400587   -0.1006896   0.15792981  0.37431902 -0.02621669
  0.20702538 -1.328874   -0.2806339   0.34456852  0.22626872 -0.285655
  2.6676202   0.18003204 -0.23781231  0.09433399 -0.04113727 -0.09008334
 -0.0373017  -0.0350816   0.08468361 -0.4464623  -0.15689898  0.25797132
 -0.33406436  0.11145066  0.040368   -0.18526098  0.17259666 -0.26326978
  0.18413429  0.19813938]
Vector shape: (50,)
```

Task 3

```
from gensim.models import Word2Vec
from gensim.utils import simple_preprocess
from pathlib import Path


class Word2VecTrainer:
    def   init  (self, data path: str, model path: str = "models/word2vec.model"):
```

```
        self.data_path = Path(data_path)
        self.model_path = Path(model_path)
        self.model = None

    def load_data(self):
        print(f"Loading data from: {self.data_path}")
        sentences = []
        with open(self.data_path, "r", encoding="utf-8") as f:
            for line in f:
                tokens = simple_preprocess(line)
                if tokens:
                    sentences.append(tokens)
        print(f"Loaded {len(sentences)} sentences.")
        return sentences

    def train(self, vector_size=50, window=5, min_count=1, workers=4, epochs=10):
        sentences = self.load_data()
        print("Training Word2Vec model...")
        self.model = Word2Vec(
            sentences=sentences,
            vector_size=vector_size,
            window=window,
            min_count=min_count,
            workers=workers,
            epochs=epochs,
        )
        print("Training completed.")

    def save_model(self):
        self.model_path.parent.mkdir(parents=True, exist_ok=True)
        self.model.save(str(self.model_path))
        print(f"Model saved to {self.model_path}")

    def load_model(self):
        print(f"Loading model from {self.model_path}")
        self.model = Word2Vec.load(str(self.model_path))
        print("Model loaded successfully.")
        return self.model

    def get_similarity(self, w1, w2):
        if w1 in self.model.wv and w2 in self.model.wv:
            return self.model.wv.similarity(w1, w2)
        else:
            print(f"One of the words not in vocabulary: {w1}, {w2}")
            return None

    def get_most_similar(self, word, top_n=5):
        if word in self.model.wv:
            return self.model.wv.most_similar(word, topn=top_n)
        else:
            print(f"Word not in vocabulary: {word}")
            return []

```

```
trainer = Word2VecTrainer(data_path="/content/en_ewt-ud-train.txt")
trainer.train(vector_size=50, window=5, min_count=1, epochs=20)

trainer.save_model()

trainer.load_model()
W1 = 'king'
W2 = 'queen'
print(f"\nTừ tương tự '{W1}':")
print(trainer.get_most_similar(W1))

print(f"\nĐộ tương đồng giữa '{W1}' và '{W2}':")
print(trainer.get_similarity(W1, W2))
```

```
Loading data from: /content/en_ewt-ud-train.txt
Loaded 14196 sentences.
Training Word2Vec model...
Training completed.
Model saved to models/word2vec.model
Loading model from models/word2vec.model
Model loaded successfully.

Từ tương tự 'king':
[('barely', 0.9760759472846985), ('goldman', 0.9746447801589966), ('welcome', 0.9742516875267029), ('bowtie', 0.9677373766
```

```
Độ tương đồng giữa 'king' và 'queen':
0.8856816
```

Task 4

```
import os
import re
from pyspark.sql import SparkSession
from pyspark.sql.functions import col, lower, regexp_replace, split
from pyspark.ml.feature import Word2Vec

os.environ["HADOOP_HOME"] = "C:\\hadoop"
os.environ["SPARK_LOCAL_DIRS"] = "C:\\\\spark-temp"
os.makedirs("C:\\hadoop\\bin", exist_ok=True)
os.makedirs("C:\\spark-temp", exist_ok=True)

spark = (
    SparkSession.builder.appName("Lab4_Spark_Word2Vec_C4")
    .config("spark.driver.memory", "4g")
    .config("spark.executor.memory", "4g")
    .getOrCreate()
)

data_path = "/content/c4-train.00000-of-01024-30K.json"

try:
    df = spark.read.json(data_path)
    df_clean = (
        df.select("text")
        .withColumn("text", lower(col("text")))
        .withColumn("text", regexp_replace(col("text"), "[^a-zA-Z\\s]", ""))
        .withColumn("words", split(col("text"), "\\s+"))
        .filter(col("words").isNotNull())
    )

    word2Vec = Word2Vec(
        vectorSize=100,
        minCount=5,
        inputCol="words",
        outputCol="result"
    )

    model = word2Vec.fit(df_clean)

    word = "computer"
    synonyms = model.findSynonyms(word, 5)
    print(f"Các từ giống '{word}' nhất (từ model Spark):")
    synonyms.show()

except Exception as e:
    print(f"Đã có lỗi xảy ra: {e}")

finally:
    spark.stop()

```

```
Các từ giống 'computer' nhất (từ model Spark):
+----------+------------------+
|      word|        similarity|
+----------+------------------+
|stationery|0.6567521691322327|
|  uptodate|0.6258612275123596|
|   desktop| 0.621239423751831|
|   browser|0.6166326403617859|
|    laptop|0.6047263741493225|
+----------+------------------+
```

Task 5

```
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.decomposition import PCA
from sklearn.manifold import TSNE
import numpy as np

words = [
```

```
    'engineer', 'artist',
    'newyork', 'london',
    'facebook', 'tesla',
    'bird', 'horse',
    'love', 'fear',
    'phone', 'computer', 'movie'
]


word_vectors = np.array([embedder.get_vector(w) for w in words])
pca = PCA(n_components=2)
vectors_pca = pca.fit_transform(word_vectors)


tsne = TSNE(n_components=2, random_state=42, perplexity=5)
vectors_tsne = tsne.fit_transform(word_vectors)

plt.style.use('seaborn-v0_8-whitegrid')
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(20, 9))

ax1.scatter(vectors_pca[:, 0], vectors_pca[:, 1], c='blue', edgecolors='k', alpha=0.6)
for i, word in enumerate(words):
    ax1.annotate(word, (vectors_pca[i, 0], vectors_pca[i, 1]), fontsize=12)
ax1.set_title("Trực quan hóa bằng PCA", fontsize=16)
ax1.set_xlabel("Thành phần chính 1", fontsize=12)
ax1.set_ylabel("Thành phần chính 2", fontsize=12)
ax1.grid(True)

ax2.scatter(vectors_tsne[:, 0], vectors_tsne[:, 1], c='green', edgecolors='k', alpha=0.6)
for i, word in enumerate(words):
    ax2.annotate(word, (vectors_tsne[i, 0], vectors_tsne[i, 1]), fontsize=12)
ax2.set_title("Trực quan hóa bằng t-SNE", fontsize=16)
ax2.set_xlabel("t-SNE dimension 1", fontsize=12)
ax2.set_ylabel("t-SNE dimension 2", fontsize=12)
ax2.grid(True)

plt.show()

```

```
/usr/local/lib/python3.12/dist-packages/IPython/core/pylabtools.py:151: UserWarning: Glyph 7847 (\N{LATIN SMALL LETTER A W
  fig.canvas.print_figure(bytes_io, **kw)
/usr/local/lib/python3.12/dist-packages/IPython/core/pylabtools.py:151: UserWarning: Glyph 7921 (\N{LATIN SMALL LETTER U W
  fig.canvas.print_figure(bytes_io, **kw)
/usr/local/lib/python3.12/dist-packages/IPython/core/pylabtools.py:151: UserWarning: Glyph 7857 (\N{LATIN SMALL LETTER A W
  fig.canvas.print_figure(bytes_io, **kw)
```