1. OBJECTIVE OF THE PROGRAM

This program trains a Word2Vec Skip-Gram model on a small custom corpus, extracts word embeddings, computes semantic similarity between words using cosine similarity, retrieves the most similar words, computes an average similarity score for selected word pairs, and finally visualizes the learned embeddings in 2D using t-SNE.

2. KEY TERMS

2.1 Corpus

Definition: A corpus is a collection of text data used for training NLP models.

2.2 Tokenization

Definition: Tokenization is the process of splitting a sentence into smaller units called tokens (usually words).
Example: "Healthcare systems rely on data" becomes
["healthcare", "systems", "rely", "on", "data"]
Word2Vec requires tokenized sentences as input.

2.3 Word Embedding

Definition: A word embedding is a dense numerical vector representation of a word such that words with similar meaning/context have vectors close to each other.
Example:

$$\text{data} \rightarrow [0.21, -0.43, 0.77, \ldots, 0.09]$$

Dimension
The embedding size is defined by $vector\_size = 100$, so each word is represented as a 100-dimensional vector.

2.4 Word2Vec

Definition: Word2Vec is a neural network-based technique that learns word embeddings by analysing the context in which words appear.

It has two architectures:

- CBOW (Continuous Bag of Words)
- Skip-Gram

2.5 Skip-Gram Model

Definition: Skip-Gram predicts surrounding context words given a target word.
If the sentence is:

$$\mathrm{w}_{t-2}, \mathrm{w}_{t-1}, \mathrm{w}_{t}, \mathrm{w}_{t+1}, \mathrm{w}_{t+2}$$

Skip-Gram uses w_t to predict w_{t-k} and w_{t+k}.
Training Objective
The Skip-Gram model maximizes:

$$\sum_{t=1}^{T} \sum_{-c<=j<=c, j\neq 0} log\, \mathrm{P}(\mathrm{w}_{t+j} \mid \mathrm{w}_{t})$$

Where:
T = total words

c = window size
$\mathrm{P}(\mathrm{w}_{t+j} \mid \mathrm{w}_t)$ = probability of a context word given target word

2.6 SoftMax Probability (Core Formula)
Skip-Gram uses SoftMax:

$$\mathrm{P}(\mathrm{w_o}|\mathrm{w_i}) = \frac{\exp(\mathrm{v}'_{\mathrm{wo}} \cdot \mathrm{v}_{\mathrm{wi}})}{\sum_{w=1}^{v} \exp(\mathrm{v}'_{\mathrm{w}} \cdot \mathrm{v}_{\mathrm{wi}})}$$

Where:

w_i = input word (target)
w_o = output word (context)
v = vocabulary size
v_{w_i} = vector of target word
v'_{w_o} = output-context vector
· = dot product

This is computationally expensive for large vocabularies, so real Word2Vec implementations typically use approximations like Negative Sampling.

2.7 Negative Sampling (Used Internally by Word2Vec)
Instead of full SoftMax, negative sampling updates only a few negative examples.
Objective:

$$log\, \sigma(v'_{w_o} \cdot v_{w_i}) + \sum_{k=1}^{K} log\, \sigma(-v'_{wk} \cdot v_{wi})$$

Where:
$\sigma(x)$ = sigmoid function
w_k = negative sampled word
K = number of negative samples

Sigmoid function:

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

2.8 Context Window
Definition: The window defines how many words around the target word are considered context.
In this code:

$$window = 3$$

So, for a target word w_t, context includes up to 3 words before and after.

2.9 Vocabulary
Definition: Vocabulary is the set of all unique words in the corpus after preprocessing.
In gensim, vocabulary is accessible via:

```
skipgram_model.wv.index_to_key
```

2.10 Cosine Similarity
Definition: Cosine similarity measures similarity between two vectors based on the cosine of the angle between them.
Formula:

$$cos\theta = \frac{(A \cdot B)}{|A||B|}$$

Where:

$A \cdot B$ is dot product

$|A|$ is magnitude of vector A

Range:

1 = identical direction (high similarity)

0 = no relationship

-1 = opposite direction

This program uses cosine similarity to measure semantic closeness between word embeddings.

2.11 t-SNE (t-distributed Stochastic Neighbour Embedding)

Definition: t-SNE is a dimensionality reduction algorithm that maps high-dimensional data (100D) into low-dimensional space (2D) for visualization while preserving neighbourhood relationships.

t-SNE converts distances into probabilities.

High-dimensional probability:

$$p_{(j|i)} = \frac{exp\left(-\frac{|x_i - x_j|^2}{2\sigma_i^2}\right)}{\sum_{k\neq i} exp\left(-\frac{|x_i - x_k|^2}{(2\sigma_i^2)}\right)}$$

Low-dimensional probability:

$$q_{ij} = \frac{\left(1 + \left|y_i - y_j\right|^2\right)^{-1}}{\sum_{k\neq l}(1 + |y_k - y_l|^2)^{-1}}$$

t-SNE minimizes KL divergence:

$$KL(P||Q) = \sum_i \sum_j log\left(\frac{p_{ij}}{q_{ij}}\right)$$

This ensures similar points remain close in 2D.

2.12 Perplexity (t-SNE Parameter)

Definition: Perplexity controls how many neighbours t-SNE considers important.

Given perplexity P, it approximates an effective neighbour count.

In the code:

perplexity=5

Small perplexity is appropriate because the corpus vocabulary is small.

3. CODE EXPLANATION

3.1 Install and Import Libraries

```
!pip install gensim
```

Installs the gensim library which contains an optimized Word2Vec implementation.

```
import gensim
```

Loads gensim into runtime so its modules can be accessed.

```
from gensim.models import Word2Vec
```
Imports the Word2Vec class used to train embeddings.

```
from sklearn.metrics.pairwise import cosine_similarity
```
Imports cosine similarity function. This import is not required because gensim already provides. `similarity()`. It is included as an alternative method.

```
from sklearn.manifold import TSNE
```
Imports t-SNE algorithm required for dimensionality reduction of embeddings.

```
import matplotlib.pyplot as plt
```
Imports plotting library to visualize word embeddings.

```
import numpy as np
```
Imports NumPy for vector/matrix handling and mean calculation.

3.2 Define the Corpus

```
sentences = [...]
```
Creates the training dataset. Each sentence contributes co-occurrence information. Word2Vec learns relationships based on which words repeatedly appear near each other.

3.3 Tokenization

```
tokenized_sentences = [sentence.lower().split() for sentence in
sentences]
```
Transforms each sentence into a list of lowercase words.
Lowercasing prevents duplication of tokens (Data vs data).
Splitting converts raw text into tokens required for Word2Vec.

3.4 Train Skip-Gram Model

```
skipgram_model = Word2Vec(
    sentences=tokenized_sentences,
    vector_size=100,
    window=3,
    min_count=1,
    sg=1,
    epochs=100
)
```

```
sentences=tokenized_sentences
```
Supplies the tokenized corpus to the training algorithm.

```
vector_size=100
```
Fixes embedding dimension. Each word is mapped to a 100-length numeric vector.

```
window=3
```
Uses a context window of ±3 words around the target word. Larger window captures broader context.

```
min_count=1
```
Keeps all words, including rare ones. This is required because the corpus is small; otherwise, many words would be removed.

```
sg=1
```
Selects Skip-Gram architecture. Skip-Gram works better than CBOW on smaller datasets and rare words.

```
epochs=100
```
Repeats training over the dataset 100 times. Small corpora need many epochs to stabilize embeddings.

3.5 Inspect Word Vector

```
word = "data"
vector = skipgram_model.wv[word]
print(f"Vector size for '{word}':", len(vector))
```

```
word = "data"
```
Sets target word to extract its embedding.

```
vector = skipgram_model.wv[word]
```
Fetches the 100-dimensional embedding for the word "data" from the trained model.

```
len(vector)
```
Verifies the dimension of the embedding.

3.6 Compute Similarity Between Two Words

```
similarity_score = skipgram_model.wv.similarity("data", "health")
print("Similarity between 'data' and 'health':", similarity_score)
.similarity("data","health")
```
Computes cosine similarity between embeddings of "data" and "health". A higher score implies these words appear in similar contexts in the corpus.

3.7 Find Most Similar Words

```
similar_words = skipgram_model.wv.most_similar("data", topn=5)
print("Words similar to 'data':")
for word, score in similar_words:
    print(word, ":", score)
.most_similar("data", topn=5)
```
Searches vocabulary and returns the 5 words whose embeddings have the highest cosine similarity with "data".

```
for word, score in similar_words
```
Iterates through returned list of tuples and prints each similar word with its similarity score.

3.8 Compute Average Similarity for Selected Pairs

```
pairs = [
    ("data", "health"),
    ("machine", "learning"),
    ("sports", "analytics")
]

scores = [skipgram_model.wv.similarity(w1, w2) for w1, w2 in pairs]
print("Average Similarity Score:", np.mean(scores))
pairs = [...]
```

Defines specific semantic pairs to evaluate.
List comprehension
Computes similarity score for each pair.

```
np.mean(scores)
```
Computes arithmetic mean of similarity scores, giving a single aggregated measure.

3.9 Prepare Embeddings for t-SNE Visualization

```
words = list(skipgram_model.wv.index_to_key)
vectors = np.array([skipgram_model.wv[word] for word in words])
index_to_key
```
Extracts the vocabulary list from the trained model.

```
vectors = np.array([...])
```
Builds a matrix where each row is the embedding vector of a word.
If vocabulary size is (N), then the matrix shape becomes:

$$(N, 100)$$

3.10 Apply t-SNE Dimensionality Reduction

```
tsne = TSNE(n_components=2, random_state=42, perplexity=5)
reduced_vectors = tsne.fit_transform(vectors)
n_components=2
```
Reduces from 100D embeddings to 2D coordinates.

```
random_state=42
```
Ensures reproducible results.

```
perplexity=5
```
Chooses a small neighborhood size, suitable for small vocabulary.

```
fit_transform(vectors)
```
Runs the t-SNE optimization and returns a 2D matrix:

$$(N, 2)$$

3.11 Plot Word Embeddings

```
plt.figure(figsize=(8, 6))

for i, word in enumerate(words):
    plt.scatter(reduced_vectors[i, 0], reduced_vectors[i, 1])
    plt.annotate(word, (reduced_vectors[i, 0], reduced_vectors[i, 1]))

plt.title("Skip-Gram Word Embedding Visualization (t-SNE)")
plt.xlabel("Dimension 1")
plt.ylabel("Dimension 2")
plt.grid(True)
plt.show()

plt.figure(figsize=(8,6))
```
Creates a plot window of fixed size.

```
enumerate(words)
```
Provides index-word mapping so the correct 2D vector is used for each word.

```
plt.scatter(x,y)
```
Plots a point representing a word embedding in 2D.

```
plt.annotate(word, (x,y))
```
Labels each point with its corresponding word.

```
plt.title / plt.xlabel / plt.ylabel
```
Adds plot metadata for interpretability.

```
plt.grid(True)
```
Improves readability by adding grid lines.

```
plt.show()
```
Displays the visualization.

4. OUTPUT EXPLANATION

4.2 Vector Size Output

```
Vector size for 'data': 100
```
Meaning: The model successfully generated a 100-dimensional embedding for "data", matching `vector_size=100`.

4.3 Similarity Output

```
Similarity between 'data' and 'health': 0.48025241
```
Meaning: "data" and "health" share moderate contextual similarity in the corpus, because multiple sentences contain health-data relations.

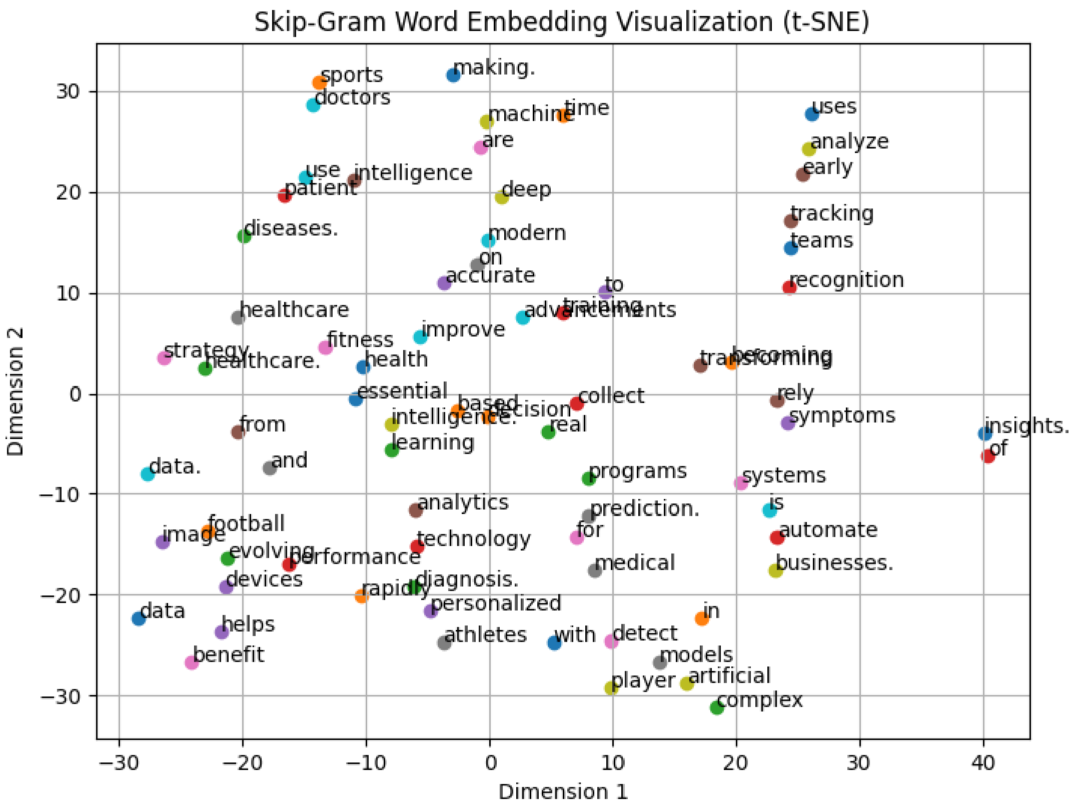
4.4 Most Similar Words Output

Example output shown:
```
Words similar to 'data':
benefit : 0.5446...
evolving : 0.5314...
for : 0.5287...
to : 0.5243...
on : 0.4924...
```
Interpretation:
The model incorrectly treats common function words ("for", "to", "on") as semantically similar.
This happens because the corpus is too small and stopwords were not removed.
Word2Vec learns statistical co-occurrence, not linguistic meaning directly.

4.5 Average Similarity Output

```
Average Similarity Score: 0.33346006
```
Meaning: On average, the three chosen word pairs show moderate similarity in this embedding space.

4.6 Visualizing t-SNE Plot



4.7 What the Plot Represents

Each labelled point is a word from the vocabulary.
The position is derived from its 100D embedding compressed into 2D using t-SNE.
Nearby points indicate words that appear in similar contexts.

4.8 Why Some Points May Look Random

t-SNE preserves local neighbour structure but can distort global structure.
Additionally, small corpus size produces weak embeddings, so clusters may not be clean.

5. COMPLETE SYSTEM-LEVEL EXPLANATION

This program illustrates the entire process of training and analysing Skip-Gram Word2Vec embeddings using the gensim library. It starts by importing the necessary libraries for training Word2Vec, calculating similarities, processing numerical data, and visualizing results. A custom set of 10 sentences is created, featuring vocabulary related to healthcare, sports analytics, and artificial intelligence. The sentences are pre-processed through tokenization, where each sentence is converted to lowercase and divided into word tokens. This tokenized data is then input into a Word2Vec model set with sg=1, enabling the Skip-Gram method. The Skip-Gram approach learns by predicting context words based on a target word, aiming to enhance the probability of co-occurrence patterns. Each word is represented in a 100-dimensional embedding space, with `vector_size = 100` and `window=3` determining the number of nearby words considered as context. The parameter `min_count=1` retains all words because of the small dataset, and `epochs=100` ensures multiple rounds of training to stabilize the embeddings.

Following training, the program extracts the embedding vector for the term "data" and verifies its dimensionality as 100. It then calculates the cosine similarity between "data" and "health" using gensim’s similarity function, resulting in a moderate similarity score around 0.48, which suggests a contextual connection within the dataset. The program identifies the five words most similar to "data" based on cosine similarity; however, due to the limited dataset and the absence of stopword removal, common function words like "for," "to," and "on" appear in the similarity list, highlighting a recognized drawback of training Word2Vec on small corpora. The code further assesses semantic connection by calculating similarity scores for pairs of three words and then averaging these scores to yield a mean similarity of about 0.333. In addition, the program visualizes the embedding space. Since the embeddings are in 100 dimensions, t-SNE is utilized to condense them into 2 dimensions while maintaining the local neighbourhood structure. These condensed 2D coordinates are plotted with matplotlib, and each point is identified by its corresponding word. The resulting scatter plot illustrates the learned semantic geometry of the vocabulary, showing that words with similar contexts tend to be closer together. This visualization can be found in the output plot displayed in the PDF. In summary, the program illustrates the training of Skip-Gram embeddings, the mathematical assessment of semantic similarity through cosine similarity, and the visualization of high-dimensional word vectors using t-SNE.