```
# 📘 Continuous Bag of Words (CBOW) Implementation in NumPy
# Works perfectly in Google Colab
# Stages:
# a) Data Preparation
# b) Generate Training Data
# c) Train Model
# d) Output Word Embeddings and Predictions

import numpy as np
from collections import defaultdict

# ---------- a) Data Preparation ----------
corpus = """we are what we repeatedly do excellence then is not an act but a habit"""
corpus = corpus.lower().split()

# Build vocabulary
vocab = sorted(set(corpus))
vocab_size = len(vocab)
word2idx = {w: i for i, w in enumerate(vocab)}
idx2word = {i: w for w, i in word2idx.items()}

print("Vocabulary:", vocab)
print("Vocab size:", vocab_size)

# ---------- b) Generate Training Data ----------
window_size = 2  # number of context words on each side
```

```
data = []

for i in range(len(corpus)):
    target = corpus[i]
    context = []
    for j in range(i - window_size, i + window_size + 1):
        if j != i and 0 <= j < len(corpus):
            context.append(corpus[j])
    data.append((context, target))

print("\nSample training pairs (context -> target):")
for context, target in data[:5]:
    print(context, "->", target)

# One-hot encoding
def one_hot(index, size=vocab_size):
    vec = np.zeros(size)
    vec[index] = 1
    return vec

# Prepare training data
X = []  # context inputs (average one-hots)
Y = []  # target outputs (one-hot)
for context_words, target_word in data:
    x = np.mean([one_hot(word2idx[w]) for w in context_words], axis=0)
    y = one_hot(word2idx[target_word])
```

```
    X.append(x)
    Y.append(y)

X = np.array(X)
Y = np.array(Y)

print("\nShapes:")
print("X:", X.shape)
print("Y:", Y.shape)

# -------------------- c) Train Model --------------------
np.random.seed(42)
D = 10  # embedding dimension
W1 = np.random.randn(vocab_size, D) * 0.01  # input -> hidden
W2 = np.random.randn(D, vocab_size) * 0.01  # hidden -> output

def softmax(x):
    e_x = np.exp(x - np.max(x, axis=1, keepdims=True))
    return e_x / np.sum(e_x, axis=1, keepdims=True)

def cross_entropy(pred, true):
    return -np.mean(np.sum(true * np.log(pred + 1e-10), axis=1))

learning_rate = 0.1
epochs = 1000
```

```
for epoch in range(1, epochs + 1):
    # Forward pass
    h = X.dot(W1)
    u = h.dot(W2)
    y_pred = softmax(u)

    loss = cross_entropy(y_pred, Y)

    # Backpropagation
    dL_du = (y_pred - Y) / X.shape[0]
    dW2 = h.T.dot(dL_du)
    dW1 = X.T.dot(dL_du.dot(W2.T))

    # Update weights
    W1 -= learning_rate * dW1
    W2 -= learning_rate * dW2

    if epoch % 200 == 0 or epoch == 1:
        print(f"Epoch {epoch:4d} | Loss: {loss:.4f}")

print("\nTraining complete!")

# ---------- d) Output ----------
# Word embeddings
embeddings = W1
```

```
def cosine_similarity(vecs, v):
    return np.dot(vecs, v) / (np.linalg.norm(vecs, axis=1) * np.linalg.norm(v) + 1e-10)

def nearest(word, top_n=3):
    idx = word2idx[word]
    v = embeddings[idx]
    sims = cosine_similarity(embeddings, v)
    nearest_ids = np.argsort(-sims)[1:top_n+1]
    return [idx2word[i] for i in nearest_ids]

print("\nNearest words by similarity:")
for w in ["we", "do", "habit", "excellence"]:
    print(f"{w:12s} -> {nearest(w)}")

# Predict the center word given context
def predict_center(context_words, top=3):
    context_vec = np.mean([one_hot(word2idx[w]) for w in context_words], axis=0).reshape(1, -
1)
    h = context_vec.dot(W1)
    u = h.dot(W2)
    probs = softmax(u)[0]
    top_ids = np.argsort(-probs)[:top]
    return [(idx2word[i], probs[i]) for i in top_ids]

print("\nPrediction examples:")
print("Context ['we', 'repeatedly'] ->", predict_center(['we', 'repeatedly']))
```

```
print("Context ['not', 'an'] ->", predict_center(['not', 'an']))
```