5: Continuous Bag of Words (CBOW) Model Implementation for text recognition # --- # a) Data preparation # b) Generate training data # c) Train model # d) Output # --

In [1]:
```
# a) Data Preparation

import numpy as np
import tensorflow as tf
from tensorflow.keras import layers, models, optimizers
print("Imports complete!\n")

# Demo text
raw_text = """Machine learning enables computers to discover patterns from data.
Neural networks are powerful models that learn representations automatically.
Natural language processing allows machines to understand human communication.
Deep learning methods are transforming artificial intelligence research and appl

vocab = sorted(set(raw_text))
vocab_size = len(vocab)
word_to_ix = {word: ix for ix, word in enumerate(vocab)}
ix_to_word = {ix: word for word, ix in word_to_ix.items()}

print("Vocabulary size:", vocab_size)
print("First 5 word-index pairs:", list(word_to_ix.items())[:5])
```

```
Imports complete!

Vocabulary size: 34
First 5 word-index pairs: [('Deep', 0), ('Machine', 1), ('Natural', 2), ('Neura
l', 3), ('allows', 4)]
```

In [2]:
```
# b) Generate Training Data

CONTEXT_SIZE = 2          # Two words before, two after
EMBEDDING_DIM = 50        # Size of word embedding vectors

def make_context_vector(context, word_to_ix):
    return [word_to_ix[w] for w in context]

data = []
for i in range(CONTEXT_SIZE, len(raw_text) - CONTEXT_SIZE):
    context = [
        raw_text[i - 2], raw_text[i - 1],
        raw_text[i + 1], raw_text[i + 2]
    ]
    target = raw_text[i]
    data.append((context, target))

X = np.array([make_context_vector(ctx, word_to_ix) for ctx, _ in data])
y = np.array([word_to_ix[target] for _, target in data])

print("Shape of X (contexts):", X.shape)
print("Shape of y (targets):", y.shape)
print("Sample context/target:\n  Context:", [vocab[ix] for ix in X[0]], "\n  Tar
```

```
Shape of X (contexts): (33, 4)
Shape of y (targets): (33,)
Sample context/target:
  Context: ['Machine', 'learning', 'computers', 'to']
  Target: enables
```

In [3]:
```python
# c) Define CBOW model in Keras

inputs = layers.Input(shape=(4,), dtype="int32")  # 4 context words (indices)
embed = layers.Embedding(input_dim=vocab_size, output_dim=EMBEDDING_DIM)(inputs)
avg_embed = layers.Lambda(lambda x: tf.reduce_mean(x, axis=1))(embed)
dense1 = layers.Dense(128, activation="relu")(avg_embed)
outputs = layers.Dense(vocab_size, activation="softmax")(dense1)

cbow_model = models.Model(inputs=inputs, outputs=outputs)
cbow_model.compile(
    optimizer=optimizers.Adam(learning_rate=0.01),
    loss="sparse_categorical_crossentropy",
    metrics=["accuracy"]
)
print("CBOW model summary:")
cbow_model.summary()
```

```
WARNING:tensorflow:From C:\Users\kusha\AppData\Local\Packages\PythonSoftwareFound
ation.Python.3.12_qbz5n2kfra8p0\LocalCache\local-packages\Python312\site-packages
\keras\src\backend\tensorflow\core.py:219: The name tf.placeholder is deprecated.
Please use tf.compat.v1.placeholder instead.

CBOW model summary:
```

Model: "functional"

Layer (type)	Output Shape
input_layer (InputLayer)	(None, 4)
embedding (Embedding)	(None, 4, 50)
lambda (Lambda)	(None, 50)
dense (Dense)	(None, 128)
dense_1 (Dense)	(None, 34)

Total params: 12,614 (49.27 KB)

Trainable params: 12,614 (49.27 KB)

Non-trainable params: 0 (0.00 B)

In [4]:
```python
# d) Train the model

history = cbow_model.fit(
    X, y,
    epochs=100,
    batch_size=8,
    verbose=1
)
print("Training done!\n")
```

```
Epoch 1/100
5/5 ━━━━━━━━━━━━━━━━━━━━ 1s 8ms/step - accuracy: 0.0000e+00 - loss: 3.5421
Epoch 2/100
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.1717 - loss: 3.4669
Epoch 3/100
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.2524 - loss: 3.3259
Epoch 4/100
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.3542 - loss: 3.0758
Epoch 5/100
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.3389 - loss: 2.6599
Epoch 6/100
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.4129 - loss: 2.1166
Epoch 7/100
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.5843 - loss: 1.6756
Epoch 8/100
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.6282 - loss: 1.1565
Epoch 9/100
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.8450 - loss: 0.7872
Epoch 10/100
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.9676 - loss: 0.5244
Epoch 11/100
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.9673 - loss: 0.3197
Epoch 12/100
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.9777 - loss: 0.1864
Epoch 13/100
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 1.0000 - loss: 0.1278
Epoch 14/100
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 1.0000 - loss: 0.0686
Epoch 15/100
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 1.0000 - loss: 0.0391
Epoch 16/100
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 1.0000 - loss: 0.0324
Epoch 17/100
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 1.0000 - loss: 0.0211
Epoch 18/100
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 1.0000 - loss: 0.0157
Epoch 19/100
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 1.0000 - loss: 0.0134
Epoch 20/100
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 1.0000 - loss: 0.0116
Epoch 21/100
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 1.0000 - loss: 0.0100
Epoch 22/100
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 1.0000 - loss: 0.0069
Epoch 23/100
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 1.0000 - loss: 0.0067
Epoch 24/100
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 1.0000 - loss: 0.0063
Epoch 25/100
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 1.0000 - loss: 0.0052
Epoch 26/100
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 1.0000 - loss: 0.0049
Epoch 27/100
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 1.0000 - loss: 0.0044
Epoch 28/100
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 1.0000 - loss: 0.0039
Epoch 29/100
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 1.0000 - loss: 0.0035
Epoch 30/100
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 1.0000 - loss: 0.0037
```

```
Epoch 31/100
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 1.0000 - loss: 0.0033
Epoch 32/100
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 1.0000 - loss: 0.0029
Epoch 33/100
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 1.0000 - loss: 0.0027
Epoch 34/100
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 1.0000 - loss: 0.0030
Epoch 35/100
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 1.0000 - loss: 0.0026
Epoch 36/100
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 1.0000 - loss: 0.0024
Epoch 37/100
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 1.0000 - loss: 0.0024
Epoch 38/100
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 1.0000 - loss: 0.0021
Epoch 39/100
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 1.0000 - loss: 0.0022
Epoch 40/100
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 1.0000 - loss: 0.0021
Epoch 41/100
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 1.0000 - loss: 0.0020
Epoch 42/100
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 1.0000 - loss: 0.0019
Epoch 43/100
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 1.0000 - loss: 0.0019
Epoch 44/100
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 1.0000 - loss: 0.0017
Epoch 45/100
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 1.0000 - loss: 0.0017
Epoch 46/100
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 1.0000 - loss: 0.0016
Epoch 47/100
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 1.0000 - loss: 0.0015
Epoch 48/100
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 1.0000 - loss: 0.0015
Epoch 49/100
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 1.0000 - loss: 0.0013
Epoch 50/100
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 1.0000 - loss: 0.0014
Epoch 51/100
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 1.0000 - loss: 0.0014
Epoch 52/100
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 1.0000 - loss: 0.0013
Epoch 53/100
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 1.0000 - loss: 0.0012
Epoch 54/100
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 1.0000 - loss: 0.0011
Epoch 55/100
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 1.0000 - loss: 0.0011
Epoch 56/100
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 1.0000 - loss: 0.0011
Epoch 57/100
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 1.0000 - loss: 0.0011
Epoch 58/100
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 1.0000 - loss: 0.0011
Epoch 59/100
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 1.0000 - loss: 9.6663e-04
Epoch 60/100
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 1.0000 - loss: 0.0010
```

```
Epoch 61/100
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 1.0000 - loss: 9.5311e-04
Epoch 62/100
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 1.0000 - loss: 9.1310e-04
Epoch 63/100
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 1.0000 - loss: 8.3330e-04
Epoch 64/100
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 1.0000 - loss: 8.5978e-04
Epoch 65/100
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 1.0000 - loss: 7.8530e-04
Epoch 66/100
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 1.0000 - loss: 8.6450e-04
Epoch 67/100
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 1.0000 - loss: 7.9345e-04
Epoch 68/100
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 1.0000 - loss: 7.8645e-04
Epoch 69/100
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 1.0000 - loss: 7.2354e-04
Epoch 70/100
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 1.0000 - loss: 7.3858e-04
Epoch 71/100
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 1.0000 - loss: 7.4215e-04
Epoch 72/100
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 1.0000 - loss: 7.1911e-04
Epoch 73/100
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 1.0000 - loss: 6.3773e-04
Epoch 74/100
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 1.0000 - loss: 6.3913e-04
Epoch 75/100
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 1.0000 - loss: 6.2974e-04
Epoch 76/100
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 1.0000 - loss: 6.4009e-04
Epoch 77/100
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 1.0000 - loss: 6.1528e-04
Epoch 78/100
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 1.0000 - loss: 5.9539e-04
Epoch 79/100
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 1.0000 - loss: 5.4058e-04
Epoch 80/100
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 1.0000 - loss: 5.6306e-04
Epoch 81/100
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 1.0000 - loss: 5.9318e-04
Epoch 82/100
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 1.0000 - loss: 5.3875e-04
Epoch 83/100
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 1.0000 - loss: 5.0996e-04
Epoch 84/100
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 1.0000 - loss: 5.4855e-04
Epoch 85/100
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 1.0000 - loss: 4.8467e-04
Epoch 86/100
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 1.0000 - loss: 4.9387e-04
Epoch 87/100
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 1.0000 - loss: 4.6639e-04
Epoch 88/100
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 1.0000 - loss: 4.6948e-04
Epoch 89/100
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 1.0000 - loss: 4.4877e-04
Epoch 90/100
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 1.0000 - loss: 4.3608e-04
```

```
Epoch 91/100
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 1.0000 - loss: 4.4905e-04
Epoch 92/100
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 1.0000 - loss: 4.5017e-04
Epoch 93/100
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 1.0000 - loss: 4.0377e-04
Epoch 94/100
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 1.0000 - loss: 4.3544e-04
Epoch 95/100
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 1.0000 - loss: 3.8707e-04
Epoch 96/100
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 1.0000 - loss: 3.9527e-04
Epoch 97/100
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 1.0000 - loss: 4.0421e-04
Epoch 98/100
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 1.0000 - loss: 3.8319e-04
Epoch 99/100
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 1.0000 - loss: 3.6891e-04
Epoch 100/100
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 1.0000 - loss: 3.7637e-04
Training done!
```

In [5]:

```python
# e) Output: Test context for prediction

test_context = ['Neural', 'networks', 'that', 'learn']
test_input = np.array([make_context_vector(test_context, word_to_ix)])
pred_probs = cbow_model.predict(test_input)
pred_idx = np.argmax(pred_probs[0])
pred_word = ix_to_word[pred_idx]

print("\n--------------------------------------------")
print("Test Context Words:", test_context)
print("Predicted Target Word:", pred_word)
print("--------------------------------------------")
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 74ms/step

--------------------------------------------
Test Context Words: ['Neural', 'networks', 'that', 'learn']
Predicted Target Word: models
--------------------------------------------
```