EX.NO.: 20
DATE: 17.03.2025

CBOW MODEL USING NN

To implement a Continuous Bag-of-Words (CBOW) model using Neural Networks to learn word embeddings from a given corpus and predict target words based on their surrounding context words.

PROCEDURE:

1. Preprocessing the corpus
2. Prepare the training data
3. Define CBOW model
4. Train model
5. Evaluate model

CODE AND OUTPUT

```
import torch
import torch.nn as nn
import torch.optim as optim
import numpy as np

# Sample corpus
corpus = "The quick brown fox jumps over the lazy dog"

# Preprocess the text
words = corpus.lower().split()
vocab = set(words)
vocab_size = len(vocab)
word_to_idx = {word: idx for idx, word in enumerate(vocab)}
idx_to_word = {idx: word for word, idx in word_to_idx.items()}

# Print Vocabulary
print("\nVocabulary (word to index mapping):")
print(word_to_idx)

# Define CBOW parameters
context_size = 2  # Number of context words on both sides
embedding_dim = 10

# Prepare training data
data = []
for i in range(context_size, len(words) - context_size):
    context = [words[j] for j in range(i - context_size, i)] + [words[j] for j in
range(i + 1, i + context_size + 1)]
    target = words[i]
    data.append((context, target))

# Print Training Data
print("\nTraining Data (Context -> Target):")
for context, target in data:
```

```python
    print(f"{context} -> {target}")

# Convert words to tensor indices
X_train = []
y_train = []
for context, target in data:
    X_train.append([word_to_idx[w] for w in context])
    y_train.append(word_to_idx[target])

X_train = torch.tensor(X_train, dtype=torch.long)
y_train = torch.tensor(y_train, dtype=torch.long)

# Print Input-Output Pairs (Tensors)
print("\nX_train (Context indices):")
print(X_train)
print("\ny_train (Target indices):")
print(y_train)

# Define the CBOW Model
class CBOW(nn.Module):
    def __init__(self, vocab_size, embedding_dim):
        super(CBOW, self).__init__()
        self.embeddings = nn.Embedding(vocab_size, embedding_dim)
        self.linear = nn.Linear(embedding_dim, vocab_size)
        self.activation = nn.LogSoftmax(dim=1)

    def forward(self, context):
        embeds = self.embeddings(context).mean(dim=1)
        out = self.linear(embeds)
        return self.activation(out)

# Initialize the model, loss function, and optimizer
model = CBOW(vocab_size, embedding_dim)
criterion = nn.NLLLoss()
optimizer = optim.SGD(model.parameters(), lr=0.01)

# Print CBOW Model Summary
print("\nCBOW Model Architecture:\n", model)

# Training loop
epochs = 100
for epoch in range(epochs):
    total_loss = 0
    for i in range(len(X_train)):
        context = X_train[i].unsqueeze(0)  # Add batch dimension
        target = y_train[i].unsqueeze(0)

        optimizer.zero_grad()
        output = model(context)
```

```python
        loss = criterion(output, target)
        loss.backward()
        optimizer.step()
        total_loss += loss.item()

    # Print loss every 10 epochs
    if (epoch + 1) % 10 == 0:
        print(f'Epoch [{epoch+1}/{epochs}], Loss: {total_loss:.4f}')

# Testing: Get word embeddings
def get_word_embedding(word):
    word_idx = torch.tensor([word_to_idx[word]], dtype=torch.long)
    return model.embeddings(word_idx).detach().numpy()

# Print Word Embeddings for Each Word in Vocabulary
print("\nWord Embeddings:")
for word in vocab:
    embedding = get_word_embedding(word)
    print(f'{word}: {embedding.flatten()}')
```

```
Vocabulary (word to index mapping):
{'the': 0, 'brown': 1, 'fox': 2, 'dog': 3, 'jumps': 4, 'quick': 5, 'lazy': 6, 'over': 7}

Training Data (Context -> Target):
['the', 'quick', 'fox', 'jumps'] -> brown
['quick', 'brown', 'jumps', 'over'] -> fox
['brown', 'fox', 'over', 'the'] -> jumps
['fox', 'jumps', 'the', 'lazy'] -> over
['jumps', 'over', 'lazy', 'dog'] -> the

X_train (Context indices):
tensor([[0, 5, 2, 4],
        [5, 1, 4, 7],
        [1, 2, 7, 0],
        [2, 4, 0, 6],
        [4, 7, 6, 3]])

y_train (Target indices):
tensor([1, 2, 4, 7, 0])

CBOW Model Architecture:
 CBOW(
  (embeddings): Embedding(8, 10)
  (linear): Linear(in_features=10, out_features=8, bias=True)
  (activation): LogSoftmax(dim=1)
)
Epoch [10/100], Loss: 10.0330
Epoch [20/100], Loss: 9.0920
Epoch [30/100], Loss: 8.3306
```

INFERENCE

The CBOW model effectively learns **context-based word embeddings**, which can enhance various NLP tasks.