Ass5

```
# ============================================================
# BE(IT)/2019 Pattern/LP-IV | Chit #4
# Continuous Bag of Words (CBOW) Model Implementation
# ============================================================

# a. Import required libraries
import numpy as np
import re
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Embedding, Lambda
from tensorflow.keras.preprocessing.text import Tokenizer
import matplotlib.pyplot as plt
import seaborn as sns

# b. Data Preparation
data = """Deep learning (also known as deep structured learning) is part of a broader family of machine learning methods based on artificial neural networks with representation learning. Learning can be supervised, semi-supervised or unsupervised. Deep-learning architectures such as deep neural networks, deep belief networks, deep reinforcement learning, recurrent neural networks, convolutional neural networks and Transformers have been applied to fields including computer vision, speech recognition, natural language processing, machine translation, bioinformatics, drug design, medical image analysis, climate science, material inspection and board game programs, where they have produced results comparable to and in some cases surpassing human expert performance."""

# Split into sentences
sentences = data.split('.')

# Clean sentences
clean_sent = []
for sentence in sentences:
    if sentence.strip() == "":
        continue
    sentence = re.sub('[^A-Za-z0-9]+', ' ', sentence)
    sentence = re.sub(r'(?:^| )\w (?:$| )', ' ', sentence).strip()
    sentence = sentence.lower()
    clean_sent.append(sentence)

# c. Tokenization
tokenizer = Tokenizer()
tokenizer.fit_on_texts(clean_sent)
sequences = tokenizer.texts_to_sequences(clean_sent)
word_to_index = tokenizer.word_index
index_to_word = tokenizer.index_word
vocab_size = len(word_to_index) + 1

# d. Generate training data (context-target pairs)
context_size = 2
contexts, targets = [], []

for sequence in sequences:
    for i in range(context_size, len(sequence) - context_size):
        target = sequence[i]
        context = [sequence[i - 2], sequence[i - 1], sequence[i + 1], sequence[i + 2]]
        contexts.append(context)
        targets.append(target)

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

print(f"✅ Vocabulary Size: {vocab_size}")
print(f"✅ Total Training Samples: {len(X)}")

# Display first few training samples
print("\nSample Context-Target Pairs:\n")
for i in range(5):
    ctx_words = [index_to_word[j] for j in X[i]]
    tgt_word = index_to_word[Y[i]]
    print(f"{ctx_words}  -->  {tgt_word}")

# e. Build the CBOW Model
emb_size = 10
model = Sequential([
    Embedding(input_dim=vocab_size, output_dim=emb_size, input_length=2 * context_size),
```

```
    Lambda(lambda x: tf.reduce_mean(x, axis=1)),
    Dense(256, activation='relu'),
    Dense(512, activation='relu'),
    Dense(vocab_size, activation='softmax')
])

model.compile(loss='sparse_categorical_crossentropy', optimizer='adam', metrics=['accuracy'])

# f. Train the Model
history = model.fit(X, Y, epochs=80, verbose=1)

# g. Visualize training loss and accuracy
plt.figure(figsize=(8, 5))
sns.lineplot(data=history.history)
plt.title("Training Progress (Loss & Accuracy)")
plt.xlabel("Epochs")
plt.ylabel("Values")
plt.grid(True)
plt.show()

# h. Test the model on new sentences
test_sentences = [
    "known as structured learning",
    "transformers have applied to",
    "where they produced results",
    "cases surpassing expert performance",
    "convolutional neural where"
]

print("\n--- CBOW Predictions ---\n")
for sent in test_sentences:
    test_words = sent.lower().split()
    x_test = [word_to_index.get(w, 0) for w in test_words]  # handle missing words
    x_test = np.array([x_test])

    pred = model.predict(x_test, verbose=0)
    pred_word = index_to_word.get(np.argmax(pred), "unknown")

    print(f"Context: {test_words}")
    print(f"Predicted Center Word: {pred_word}\n")
```

---

ass5B

```
# ✅ Continuous Bag of Words (CBOW) Model Implementation

# a. Data Preparation
import matplotlib.pyplot as plt
import numpy as np
import re
%matplotlib inline

sentences = """We are about to study the idea of a computational process.
Computational processes are abstract beings that inhabit computers.
As they evolve, processes manipulate other abstract things called data.
The evolution of a process is directed by a pattern of rules
called a program. People create programs to direct processes. In effect,
we conjure the spirits of the computer with our spells."""

# Clean the data
sentences = re.sub('[^A-Za-z0-9]+', ' ', sentences)
sentences = re.sub(r'(?:^| )\w(?:$| )', ' ', sentences).strip()
sentences = sentences.lower()

# Create vocabulary
words = sentences.split()
vocab = set(words)
vocab_size = len(vocab)
embed_dim = 10
context_size = 2

word_to_ix = {word: i for i, word in enumerate(vocab)}
ix_to_word = {i: word for i, word in enumerate(vocab)}
```

```
# b. Generate Training Data
data = []
for i in range(2, len(words) - 2):
    context = [words[i - 2], words[i - 1], words[i + 1], words[i + 2]]
    target = words[i]
    data.append((context, target))
print("Sample training data:", data[:5])


# c. Train Model
embeddings = np.random.random_sample((vocab_size, embed_dim))
theta = np.random.uniform(-1, 1, (2 * context_size * embed_dim, vocab_size))

def linear(m, theta):
    return m.dot(theta)

def log_softmax(x):
    e_x = np.exp(x - np.max(x))
    return np.log(e_x / e_x.sum())

def NLLLoss(logs, targets):
    out = logs[range(len(targets)), targets]
    return -out.sum() / len(out)

def log_softmax_crossentropy_with_logits(logits, target):
    out = np.zeros_like(logits)
    out[np.arange(len(logits)), target] = 1
    softmax = np.exp(logits) / np.exp(logits).sum(axis=-1, keepdims=True)
    return (-out + softmax) / logits.shape[0]

def forward(context_idxs, theta):
    m = embeddings[context_idxs].reshape(1, -1)
    n = linear(m, theta)
    o = log_softmax(n)
    return m, n, o

def backward(preds, theta, target_idxs):
    m, n, o = preds
    dlog = log_softmax_crossentropy_with_logits(n, target_idxs)
    dw = m.T.dot(dlog)
    return dw

def optimize(theta, grad, lr=0.03):
    theta -= grad * lr
    return theta

# Training loop
epoch_losses = {}
for epoch in range(80):
    losses = []
    for context, target in data:
        context_idxs = np.array([word_to_ix[w] for w in context])
        preds = forward(context_idxs, theta)
        target_idxs = np.array([word_to_ix[target]])
        loss = NLLLoss(preds[-1], target_idxs)
        losses.append(loss)
        grad = backward(preds, theta, target_idxs)
        theta = optimize(theta, grad, lr=0.03)
    epoch_losses[epoch] = losses

# Plot training loss
ix = np.arange(0, 80)
plt.figure(figsize=(6,4))
plt.title('Epoch vs Loss', fontsize=14)
plt.plot(ix, [epoch_losses[i][0] for i in ix])
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.show()


# d. Output
def predict(words):
    context_idxs = np.array([word_to_ix[w] for w in words])
    preds = forward(context_idxs, theta)
    word = ix_to_word[np.argmax(preds[-1])]
```

```
    return word

def accuracy():
    wrong = 0
    for context, target in data:
        if predict(context) != target:
            wrong += 1
    return (1 - (wrong / len(data)))

print("Predicted word:", predict(['we', 'are', 'to', 'study']))
print("Model Accuracy:", accuracy())
```