Prbolem Statement 5: Implement the Continuous Bag of Words (CBOW) Model. Stages can be: Data preparation Generate training data Train model Output

```
# Import necessary libraries
import numpy as np
import pandas as pd
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, Lambda, Dense
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences

# Sample corpus
corpus = [
    "the cat sat on the mat",
    "the dog sat on the log",
    "cats and dogs are great pets",
    "the mat is soft and warm"
]

# Preprocess text: Tokenization and lowercasing
tokenizer = Tokenizer()
tokenizer.fit_on_texts(corpus)
total_words = len(tokenizer.word_index) + 1  # +1 for padding

# Convert text to sequences
sequences = tokenizer.texts_to_sequences(corpus)



## Stage b: Generate Training Data
def generate_training_data(sequences, window_size=2):
    contexts = []
    targets = []

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

    return np.array(contexts), np.array(targets)

X, y = generate_training_data(sequences)

# Pad sequences for consistent input shape
X = pad_sequences(X, maxlen=4)  # Adjust maxlen based on context size



## Stage c: Train Model
# Define CBOW model architecture
model = Sequential()
model.add(Embedding(input_dim=total_words, output_dim=10, input_length=4))
model.add(Lambda(lambda x: tf.reduce_mean(x, axis=1)))  # Average embeddings
model.add(Dense(total_words, activation='softmax'))

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

# Train the model
model.fit(X, y, epochs=100)
```

```
Epoch 82/100
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 60ms/step - accuracy: 0.8750 - loss: 2.4589
Epoch 83/100
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 57ms/step - accuracy: 0.8750 - loss: 2.4528
Epoch 84/100
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 37ms/step - accuracy: 0.8750 - loss: 2.4467
Epoch 85/100
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 35ms/step - accuracy: 0.8750 - loss: 2.4406
Epoch 86/100
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 58ms/step - accuracy: 0.8750 - loss: 2.4345
Epoch 87/100
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 41ms/step - accuracy: 0.8750 - loss: 2.4283
Epoch 88/100
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 50ms/step - accuracy: 0.8750 - loss: 2.4220
Epoch 89/100
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 38ms/step - accuracy: 0.8750 - loss: 2.4157
Epoch 90/100
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 52ms/step - accuracy: 0.8750 - loss: 2.4094
Epoch 91/100
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 30ms/step - accuracy: 0.8750 - loss: 2.4031
Epoch 92/100
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 61ms/step - accuracy: 0.8750 - loss: 2.3967
Epoch 93/100
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 53ms/step - accuracy: 0.8750 - loss: 2.3902
Epoch 94/100
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 55ms/step - accuracy: 0.8750 - loss: 2.3838
Epoch 95/100
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 56ms/step - accuracy: 0.8750 - loss: 2.3773
Epoch 96/100
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 38ms/step - accuracy: 0.8750 - loss: 2.3707
Epoch 97/100
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 43ms/step - accuracy: 0.8750 - loss: 2.3641
Epoch 98/100
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 58ms/step - accuracy: 0.8750 - loss: 2.3575
Epoch 99/100
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 41ms/step - accuracy: 0.8750 - loss: 2.3509
Epoch 100/100
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 56ms/step - accuracy: 0.8750 - loss: 2.3442
<keras.src.callbacks.history.History at 0x78040743e740>
```

```
## Stage d: Output
# Get word embeddings from the trained model
word_embeddings = model.layers[0].get_weights()[0]

# Create a mapping of words to their embeddings
word_index = tokenizer.word_index


print('Vocabulary Size:', len(word_index))
print('Vocabulary Sample:', list(word_index.items())[:10],"\n\n")


embeddings_dict = {word: word_embeddings[idx] for word, idx in word_index.items()}

# Output the embeddings for each word in a structured format
print("{:<10} | {}".format("Word", "Embedding"))
print("-" * 40)
for word, embedding in embeddings_dict.items():
    print("{:<10} | {}".format(word, np.round(embedding, 3)))
```

```
Vocabulary Size: 16
Vocabulary Sample: [('the', 1), ('sat', 2), ('on', 3), ('mat', 4), ('and', 5), ('cat', 6), ('dog', 7), ('log', 8), ('cats', 9), ('dc


Word       | Embedding
----------------------------------------
the        | [-0.082  0.096  0.303 -0.273  0.116 -0.122  0.018 -0.059 -0.243 -0.128]
sat        | [-0.077 -0.15   0.139 -0.22   0.145 -0.109 -0.176  0.183 -0.189 -0.202]
on         | [-0.14   0.124  0.14  -0.164  0.105 -0.131  0.137 -0.172 -0.157 -0.136]
mat        | [ 0.116 -0.124  0.093  0.051  0.024  0.142 -0.173  0.181 -0.105  0.054]
and        | [ 0.129  0.026 -0.06  -0.052 -0.207  0.022 -0.2    0.141  0.021  0.021]
cat        | [-0.1   -0.044  0.165 -0.111  0.129 -0.119 -0.028 -0.02  -0.166 -0.138]
dog        | [-0.078  0.037  0.183 -0.164  0.142 -0.103 -0.012 -0.048 -0.099 -0.179]
log        | [-0.08  -0.063  0.086 -0.159  0.111 -0.15   -0.102  0.071 -0.146 -0.097]
cats       | [-0.093 -0.105 -0.092 -0.082 -0.127 -0.127 -0.137 -0.141  0.108 -0.126]
dogs       | [ 0.08   0.119  0.116 -0.151 -0.169 -0.112 -0.134  0.12  -0.1   -0.052]
are        | [-0.143 -0.056 -0.128 -0.086 -0.076 -0.101 -0.091 -0.155  0.072 -0.144]
great      | [ 0.02   0.127 -0.015 -0.135 -0.125 -0.148 -0.181  0.117  0.007 -0.103]
pets       | [ 0.145  0.135  0.136 -0.155 -0.164 -0.122 -0.156  0.13  -0.148 -0.066]
is         | [ 0.095 -0.088 -0.086  0.065 -0.1    0.123 -0.118  0.089  0.15   0.079]
soft       | [ 0.133 -0.056  0.15   0.017 -0.109  0.068 -0.154  0.152 -0.103  0.049]
warm       | [ 0.085 -0.152 -0.109  0.138 -0.116  0.099 -0.097  0.139  0.158  0.086]
```

Start coding or generate with AI.