1. Design and implement a neural based network for generating word embedding for words in a document corpus.

```
import torch
import torch.nn as nn
import torch.optim as optim
from collections import Counter
import random
import numpy as np

# Sample corpus (list of sentences; replace with your document corpus)
corpus = [
    "the quick brown fox jumps over the lazy dog",
    "word embeddings capture semantic meanings in vectors",
    "neural networks are powerful for natural language processing",
    "fork is built by xai to answer questions"
]

# Preprocessing: Tokenize and build vocabulary
tokens = [word for sentence in corpus for word in sentence.split()]
vocab = list(set(tokens))  # Unique words
vocab_size = len(vocab)
word_to_idx = {word: i for i, word in enumerate(vocab)}
idx_to_word = {i: word for word, i in word_to_idx.items()}

# Generate training data (skip-gram pairs)
def generate_pairs(tokens, window_size=2):
    pairs = []
    for i, target in enumerate(tokens):
        context = tokens[max(0, i - window_size):i] + tokens[i + 1:i + window_size + 1]
        for ctx in context:
            pairs.append((word_to_idx[target], word_to_idx[ctx]))
    return pairs

data = generate_pairs(tokens)

# Negative sampling: Build frequency-based sampler for negatives
word_freq = Counter(tokens)
word_freq = {word: freq ** 0.75 for word, freq in word_freq.items()}  # Unigram^0.75
total_freq = sum(word_freq.values())
neg_sampling_probs = {word: freq / total_freq for word, freq in word_freq.items()}

def get_negatives(target, num_neg=5):
    negatives = []
    while len(negatives) < num_neg:
        neg = random.choices(list(neg_sampling_probs.keys()),
weights=list(neg_sampling_probs.values()))[0]
        if neg != target:
            negatives.append(word_to_idx[neg])
    return negatives
```

```
# Model Definition
class SkipGramModel(nn.Module):
    def __init__(self, vocab_size, embed_dim):
        super(SkipGramModel, self).__init__()
        self.embeddings = nn.Embedding(vocab_size, embed_dim)  # Target embeddings
        self.context_embeddings = nn.Embedding(vocab_size, embed_dim)  # Context embeddings

    def forward(self, target, context):
        target_emb = self.embeddings(target)
        context_emb = self.context_embeddings(context)
        return target_emb, context_emb

# Training
embed_dim = 100
model = SkipGramModel(vocab_size, embed_dim)
optimizer = optim.SGD(model.parameters(), lr=0.001)
loss_fn = nn.BCEWithLogitsLoss()  # For negative sampling

epochs = 10
for epoch in range(epochs):
    total_loss = 0
    for target_idx, context_idx in data:
        # Positive sample
        target = torch.tensor([target_idx], dtype=torch.long)
        context_pos = torch.tensor([context_idx], dtype=torch.long)
        target_emb, context_pos_emb = model(target, context_pos)
        pos_score = torch.sum(target_emb * context_pos_emb, dim=1)
        pos_label = torch.ones_like(pos_score)

        # Negative samples
        negs = get_negatives(idx_to_word[target_idx])
        context_neg = torch.tensor(negs, dtype=torch.long)
        _, context_neg_emb = model(target, context_neg)  # Reuse target_emb
        neg_score = torch.matmul(target_emb, context_neg_emb.t()).squeeze(0)
        neg_label = torch.zeros_like(neg_score)

        # Combined loss
        scores = torch.cat([pos_score, neg_score])
        labels = torch.cat([pos_label, neg_label])
        loss = loss_fn(scores, labels)
        total_loss += loss.item()

        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

    print(f"Epoch {epoch + 1}, Loss: {total_loss / len(data):.4f}")

# Extract and print sample embeddings
embeddings = model.embeddings.weight.data.numpy()
print("\nSample Word Embeddings:")
```

```
for word in ["the", "word", "neural", "fork"]:
    if word in word_to_idx:
        idx = word_to_idx[word]
        print(f"{word}: {embeddings[idx][:5]}... (first 5 dims)")
    else:
        print(f"{word}: Not in vocabulary")
```

```
Epoch 1, Loss: 4.2422
Epoch 2, Loss: 3.9080
Epoch 3, Loss: 4.2143
Epoch 4, Loss: 3.9021
Epoch 5, Loss: 3.7951
Epoch 6, Loss: 3.9649
Epoch 7, Loss: 4.0796
Epoch 8, Loss: 3.8775
Epoch 9, Loss: 3.9534
Epoch 10, Loss: 3.8556

Sample Word Embeddings:
the: [-0.8851693  -0.49322948 -0.03846395  0.919265    0.07567823]... (first 5 d
ims)
word: [ 1.4223266  -1.5130905  -0.8228552   0.14413585  0.8477999 ]... (first 5
dims)
neural: [ 0.44077718 -1.2260877   1.6728966   0.7193192   0.15576063]... (first
5 dims)
fork: [ 1.324665    0.12992626 -0.05621464  1.2088064   0.02148499]... (first 5
dims)
```