Codes:

1)Skipgram code

txt.py:-

```
import re

from collections import Counter

import numpy as np

import torch

import torch.nn as nn

import torch.optim as optim




file_path = "enwik9.txt"



with open(file_path, "r", encoding="utf-8") as f:

    text = f.read().lower()




words = re.findall(r"[a-z]+", text)
```

```
print("Total words:", len(words))

print("Sample words:", words[:20])



word_counts = Counter(words)







vocab = {w:c for w,c in word_counts.items() if c >= 5}



print("Vocabulary size:", len(vocab))



idx2word = list(vocab.keys())

word2idx = {w:i for i,w in enumerate(idx2word)}



print("Index of 'king':", word2idx.get("king"))

print("Word at index 100:", idx2word[100])







words = words[:100_000]
```

```
window_size = 2

training_pairs = []


for i, word in enumerate(words):
    if word not in word2idx:
        continue

    center = word2idx[word]

    start = max(0, i - window_size)
    end   = min(len(words), i + window_size + 1)

    for j in range(start, end):
        if i == j:
            continue
```

```
        context_word = words[j]
        if context_word in word2idx:
            training_pairs.append((center, word2idx[context_word]))
training_pairs = training_pairs[:200_000]


print("Total training pairs:", len(training_pairs))
print("Sample pairs:", training_pairs[:10])


freqs = np.array([vocab[w] for w in idx2word], dtype=np.float32)
unigram_dist = freqs ** 0.75
unigram_dist = unigram_dist / unigram_dist.sum()


def get_negative_samples(pos_index, k=2):
    negatives = []
    while len(negatives) < k:
        neg = np.random.choice(len(idx2word), p=unigram_dist)
        if neg != pos_index:
            negatives.append(neg)
    return negatives
```

```
vocab_size = len(idx2word)
embed_size = 50

input_emb  = nn.Embedding(vocab_size, embed_size)
output_emb = nn.Embedding(vocab_size, embed_size)

optimizer = optim.Adam(list(input_emb.parameters()) +
                       list(output_emb.parameters()), lr=0.001)

criterion = nn.BCEWithLogitsLoss()


def train_skipgram(pairs, epochs=1, batch_size=512):

    for epoch in range(epochs):
        total_loss = 0

        for i in range(0, len(pairs), batch_size):
```

```
            batch = pairs[i:i+batch_size]

            optimizer.zero_grad()



            loss = 0



            for center, context in batch:




                center_vec = input_emb(torch.tensor([center]))

                pos_vec    = output_emb(torch.tensor([context]))



                pos_score = torch.matmul(center_vec, pos_vec.T)

                pos_loss  = criterion(pos_score,
torch.ones_like(pos_score))




                neg_samples = get_negative_samples(context, k=2)

                neg_vecs = output_emb(torch.tensor(neg_samples))



                neg_score = torch.matmul(center_vec, neg_vecs.T)
```

```
                neg_loss  = criterion(neg_score,
torch.zeros_like(neg_score))



                loss += pos_loss + neg_loss



            loss.backward()

            optimizer.step()

            total_loss += loss.item()



        print(f"Epoch {epoch+1} completed | Loss: {total_loss:.2f}")



print("Starting training...")

train_skipgram(training_pairs, epochs=1)

print("Training finished!")
```

Output:

```
PS C:\Users\ruthw\OneDrive\Desktop\assing nlp> python txt.py
```

```
Total words: 141177027

Sample words: ['mediawiki', 'xmlns', 'http', 'www', 'mediawiki', 'org',
'xml', 'export', 'xmlns', 'xsi', 'http', 'www', 'w', 'org', 'xmlschema',
'instance', 'xsi', 'schemalocation', 'http', 'www']

Vocabulary size: 277712

Index of 'king': 170

Word at index 100: diggers

Total training pairs: 200000

Sample pairs: [(0, 1), (0, 2), (1, 0), (1, 2), (1, 3), (2, 0), (2, 1), (2,
3), (2, 0), (3, 1)]

Starting training…
```

Epoch 1 completed | Loss: 2772.223
Training finished!

---

2) compare:

```python
import re
from gensim.models import Word2Vec

with open("enwik9.txt","r",encoding="utf-8") as f:
    text = f.read().lower()

words = re.findall(r"[a-z]+", text)
words = words[:2_000_000]   # small part

sentences = []
for i in range(0,len(words),50):
    sentences.append(words[i:i+50])

model = Word2Vec(
    sentences,
    vector_size=300,
```

```
    window=2,
    min_count=5,
    sg=1,
    negative=2,
    workers=4
)

model.save("my_w2v.model")
print("Training finished!")
```

---

3)import gensim.downloader as api

```
from sklearn.metrics.pairwise import cosine_similarity

pre = api.load("word2vec-google-news-300")
my  = model

words = ["king","queen","man","woman","computer"]

for w in words:
    if w in my.wv and w in pre:
        v1 = my.wv[w].reshape(1,-1)
        v2 = pre[w].reshape(1,-1)
        sim = cosine_similarity(v1,v2)[0][0]
        print(w,"→ cosine similarity:",round(sim,3))
print(my.wv.most_similar(positive=["king","woman"], negative=["man"],
topn=5))
```

Output:

```
king → cosine similarity: -0.077

queen → cosine similarity: -0.019

man → cosine similarity: -0.011

woman → cosine similarity: 0.058

computer → cosine similarity: 0.082

[('emperor', 0.8300297856330872), ('cleopatra', 0.8082027435302734),
('throne', 0.8056543469429016), ('ruler', 0.7952570915222168), ('darius',
0.7798907160758972)]
```

---

4)

```
import numpy as np

def cos(u,v):
    return np.dot(u,v)/(np.linalg.norm(u)*np.linalg.norm(v))

def bias(word,a,b):
    return cos(my.wv[word], my.wv[a]) - cos(my.wv[word], my.wv[b])

for w in ["doctor","nurse","engineer","teacher"]:
    if w in my.wv:
        print(w,"bias (he-she):",round(bias(w,"he","she"),3))
```

Output:

```
doctor bias (he-she): -0.171
nurse bias (he-she): -0.134
engineer bias (he-she): -0.058
teacher bias (he-she): -0.139

Note:
Due to system limitations, the final training and testing were done using
Google Colab.

Embeddings saved  in embeddings.txt
Results saved in results.txt
```