In [10]:

```
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import Dataset, DataLoader
from collections import Counter
import torch.nn.functional as F
from sklearn.model_selection import train_test_split
import jieba
import numpy as np
import re
```

In [11]:

```
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print("Using device:", device)
```

```
Using device: cuda
```

In [33]:

```
stopwords_file = "./cn_stopwords.txt"
with open(stopwords_file, "r", encoding="utf-8") as file:
    stop_words_list = file.readlines()
stop_words_list = [line.strip() for line in stop_words_list]
```

In [19]:

```
with open("./骆驼祥子.txt", "r", encoding="utf-8") as file:
    text = file.read()

text = re.sub(r'[^\w\s]', '', text)
text = re.sub(r'\s+', ' ', text)
text = text.replace(" ", "")

raw_text_0 = list(jieba.cut(text, cut_all=False))
raw_text = [word for word in raw_text_0 if word not in stop_words_list]

vocab = set(raw_text)
```

In [20]:

```
CONTEXT_SIZE = 2  # 2 words to the left, 2 to the right
EMBEDDING_DIM = 50

# By deriving a set from `raw_text`, we deduplicate the array
vocab_size = len(vocab)

word_to_ix = {word: i for i, word in enumerate(vocab)}
data = []
for i in range(CONTEXT_SIZE, len(raw_text) - CONTEXT_SIZE):
    context = (
        [raw_text[i - j - 1] for j in range(CONTEXT_SIZE)]
        + [raw_text[i + j + 1] for j in range(CONTEXT_SIZE)]
    )
    target = raw_text[i]
    data.append((context, target))
print(data[:5])
```

```
[(['祥子', '骆驼', '祥子', '骆驼'], '介绍'), (['介绍', '祥子', '骆驼', '骆驼'],
'祥子'), (['祥子', '介绍', '骆驼', '外号'], '骆驼'), (['骆驼', '祥子', '外号',
'先'], '骆驼'), (['骆驼', '骆驼', '先', '说祥子'], '外号')]
```

In [21]:
```
class CBOW(nn.Module):

    def __init__(self, vocab_size, embedding_dim, context_size):
        super(CBOW, self).__init__()
        self.embeddings = nn.Embedding(vocab_size, embedding_dim).to(device)
        self.linear1 = nn.Linear(embedding_dim, 128).to(device)
        self.linear2 = nn.Linear(128, vocab_size).to(device)

    def forward(self, inputs):
        embeds = torch.mean(self.embeddings(inputs), dim=0).view((1, -1))
        out = F.relu(self.linear1(embeds))
        out = self.linear2(out)
        log_probs = F.log_softmax(out, dim=1)
        return log_probs

losses = []
loss_function = nn.NLLLoss()
model = CBOW(len(vocab), EMBEDDING_DIM, CONTEXT_SIZE).to(device)
optimizer = optim.SGD(model.parameters(), lr=0.001)
```

In [24]:
```
import time
start_time = time.time()

for epoch in range(10):
    total_loss = 0
    for context, target in data:

        # Step 1. Prepare the inputs to be passed to the model (i.e, turn the wo
        context_idxs = torch.tensor([word_to_ix[w] for w in context], dtype=tor

        # Step 2. Recall that torch *accumulates* gradients. Before passing in a
        # new instance, you need to zero out the gradients from the old instance
        model.zero_grad()

        # Step 3. Run the forward pass, getting log probabilities over next word
        log_probs = model(context_idxs)

        # Step 4. Compute your loss function. (Again, Torch wants the target wor
        loss = loss_function(log_probs, torch.tensor([word_to_ix[target]], dtype

        # Step 5. Do the backward pass and update the gradient
        loss.backward()
        optimizer.step()

        # Get the Python number from a 1-element Tensor by calling tensor.item()
        total_loss += loss.item()
    losses.append(total_loss)
print(losses)  # The loss decreased every iteration over the training data!

end_time = time.time()
print(f"运行时间: {end_time - start_time} ")
```

```
[383040.6752166748, 371139.4347035885, 349080.9028342962, 342351.5752040148, 339795.038377285, 337973.7778482735, 336484.98399430513, 335140.70063331723, 333851.7763360292, 332568.11263199896, 331258.9643229209, 329905.11920079216, 328492.62182867713]
运行时间: 820.6160748004913
```

In [25]:
```python
# Create your model and train. Here are some functions to help you make the data
def make_context_vector(context, word_to_ix):
    idxs = [word_to_ix[w] for w in context]
    return torch.tensor(idxs, dtype=torch.long)

make_context_vector(data[0][0], word_to_ix)  # example
```

Out[25]:
```
tensor([ 764, 2245,  764, 2245])
```

In [ ]:
```python

```

使用gensim中的Word2Vec包实现CBOW

In [43]:
```python
from gensim.models import Word2Vec
```

In [ ]:
```python
"""
- sentences: 可迭代的语句列表，较大的语料库可以考虑从磁盘/IO的形式传输
- vector_size: 单词向量的维数
- window: 句子中当前单词与预测单词的最大距离
- min_count: 忽略总频率低于此值的所有单词
- workers: 使用多个 worker 线程训练模型
- sg: 训练算法，1-> skip-gram 否则 -> CBOW
- hs: 1 -> 分层 softmax 方法，否则 -> 负采样
- negative: >0 则使用负采样，通常推荐距离为 [5-20]，如果设置为0则不适用负采样
- alpha: 初始学习率
- min_alpha: 随着训练进行，学习率将线性下降至 min_alpha
- max_vocab_size: 词库限制，每 1000w 个字类型大约需要1GB的 RAM
- sample: 配置较高频率的单词随机下采样的阈值，生效范围 (0,1e-5)
- epochs 迭代次数
"""
```

In [48]:
```python
with open("三国演义.txt", 'r',encoding='utf-8')as f: # 读入文本
    lines = []
    for line in f: #分别对每段分词
        temp = jieba.lcut(line)  #结巴分词 精确模式
        words = []
        for i in temp:
            #过滤掉所有的标点符号
            i = re.sub("[\s+\.\!\/_,$%^*(+\"\'”” 《》]+|[+——！，。？、~@#￥%·
            if len(i) > 0:
                words.append(i)
        if len(words) > 0:
            lines.append(words)
print(lines[0:5])
```

```
[['三国演义'], ['第一回', '宴', '桃园', '豪杰', '三', '结义', '斩', '黄巾', '英雄', '首', '立功'], ['滚滚', '长江', '东', '逝水', '浪花', '淘尽', '英雄', '是非成败', '转头', '空'], ['青山', '依旧', '在', '几度', '夕阳红', '白发', '渔樵', '江渚上', '惯'], ['看', '秋月春风', '一壶', '浊酒', '喜相逢', '古今', '多少', '事', '都', '付']]
```

In [60]:

```
model = Word2Vec(
    lines,
    vector_size = 100,
    hs = 1,
    sg = 2,
    min_count = 1,
    window = 2,
    workers = 4,
    epochs = 10
)
```

In [61]:

```
print("孔明的词向量：\n",model.wv.get_vector('孔明'))
```

```
孔明的词向量：
 [-0.32365045  0.27591166  0.5305709  -0.23610966 -0.40783113 -0.05334409
 -0.12471253  0.30638564  0.75083137 -0.64840454 -0.0165752   0.40281984
  0.03974082 -0.30338097  0.0661559  -0.32427162 -0.24906923 -0.40847516
 -0.39091098  0.11982058 -0.14609097 -0.0076288   0.3399963   0.00193367
  0.72057104  0.21825542 -0.9662531  -0.35575494 -0.26800779 -0.22006762
 -0.3623526  -0.06418754  0.0697577  -0.2002573  -0.0141564   0.702028
  0.02826011  0.31054723  0.55619836 -0.38314077  0.50334597 -0.4325456
 -0.5552436  -0.50480545  0.12148239 -0.66760194 -0.63635737  0.40741077
  0.39663538 -0.09630173 -0.24927041 -0.04164707 -0.17518984  0.7221643
  0.3903633  -0.00678943  0.5938008  -0.26348865 -0.41011506 -0.40510067
 -0.39398313 -0.26716083  0.5274327   0.37557262  0.47788078  0.1622746
  0.3886236   0.7362393  -0.31729952  0.41644067 -0.6646033   0.21656819
  0.72482973  0.10144351  0.38675648 -0.02992108 -0.25849965 -0.15729703
 -0.58085984  0.19022861 -0.04304889 -0.44869766 -0.03818106  0.08713236
 -0.29699644  0.07344805  0.59316033  0.41191667 -0.12846035 -0.03103376
  0.12596759 -0.10565289  0.05203108 -0.3798121  -0.15744796  0.07548639
  0.02230603 -0.34300146  0.421915   -0.4369814 ]
```

In [62]:

```
print("\n和孔明相关性最高的前20个词语：")
model.wv.most_similar('孔明', topn = 20)# 与孔明最相关的前20个词语
```

```
和孔明相关性最高的前20个词语：
```

Out[62]:

```
[('玄德', 0.7855352759361267),
 ('孟获', 0.7491429448127747),
 ('先主', 0.7122588157653809),
 ('懿', 0.71099853515625),
 ('周瑜', 0.7088468074798584),
 ('鲁肃', 0.7035942673683167),
 ('姜维', 0.6889432072639465),
 ('孙夫人', 0.686983048915863),
 ('郝昭', 0.6833346486091614),
 ('心中', 0.6751553416252136),
 ('孔明来', 0.6660807728767395),
 ('瑜', 0.662392258644104),
 ('庞统', 0.6611930727958679),
 ('孙权', 0.6606889367103577),
 ('后主', 0.6582843661308289),
 ('关公', 0.6560310125350952),
 ('王允', 0.6555484533309937),
 ('马谡', 0.6552057862281799),
 ('孔明回', 0.6539878845214844),
 ('玄德公', 0.6519277095794678)]
```

In [63]:

```python
def getSimilarSeq(key, model, top = 10):
    print("Top For %s ======================" % key)
    sims = model.wv.most_similar(key, topn = top)
    for i in sims:
        print(i)
    print("End Sim For %s ======================" % key)
```

In [64]:

```python
getSimilarSeq("玄德", model)
getSimilarSeq("云长", model)
```

```
Top For 玄德 ======================
('孔明', 0.7855352163314819)
('云长', 0.7299472689628601)
('孙夫人', 0.7004892826080322)
('鲁肃', 0.698726236820221)
('孔明遂', 0.6961550712585449)
('孟获', 0.6815526485443115)
('庞统', 0.6662197113037109)
('诸葛均', 0.6655987501144409)
('孙权', 0.6647616624832153)
('孙乾', 0.6642411351203918)
End Sim For 玄德 ======================
Top For 云长 ======================
('张飞', 0.7685154676437378)
('赵云', 0.7665302753448486)
('关公', 0.7517321109771729)
('岱', 0.7360236644744873)
('玄德', 0.7299474477767944)
('孙夫人', 0.7195055484771729)
('王忠', 0.7171892523765564)
('魏延', 0.7167795896530151)
('翼德', 0.7129519581794739)
('忠', 0.7106291055679321)
End Sim For 云长 ======================
```