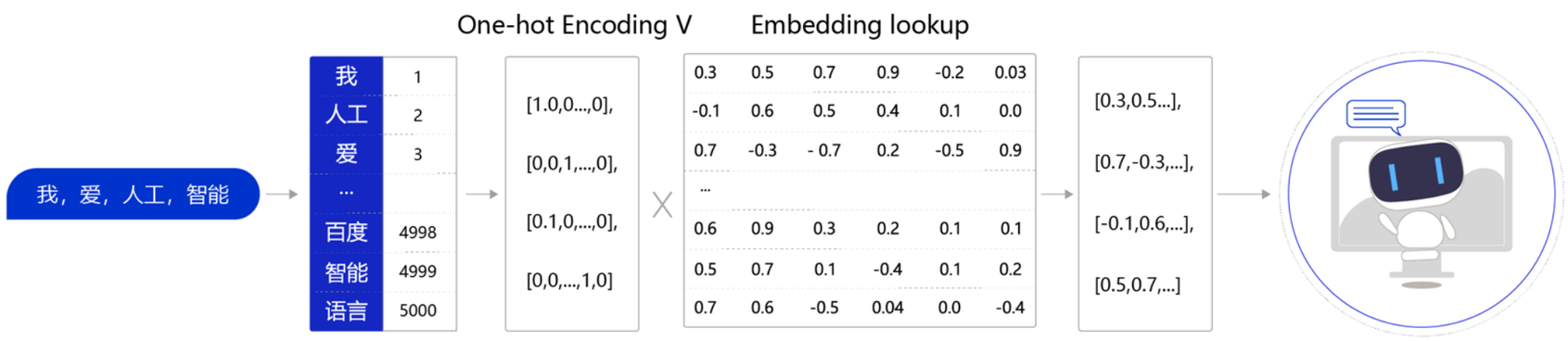
第3章 Word2Vec入门



In [1]:
```
import re
import torch
import torch.nn as nn
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from collections import Counter
from sklearn.decomposition import PCA
%pprint

import sys
sys.path.append(r"C:\Users\Administrator\Desktop\Open1507Lab")
import nndl

 # 创建浮点数时默认设为float64，保证数据x和模型参数为同一类型(labels必须为int64)
torch.set_default_dtype(torch.float64)

 # 在notebook中运行如果提示"The kernel appears to have died. It will restart automatically."
import os
os.environ["KMP_DUPLICATE_LIB_OK"]="TRUE"

# 数据打乱和模型参数初始化时用到了随机，设置随机种子保证模型结果可复现
torch.manual_seed(102)
```

```
Pretty printing has been turned OFF
```

Out[1]:
```
<torch._C.Generator object at 0x000001B73B511C90>
```

3.1 CBOW模型的简单实现思路

3.1.1 模型思路：通过上下文单词预测中间词

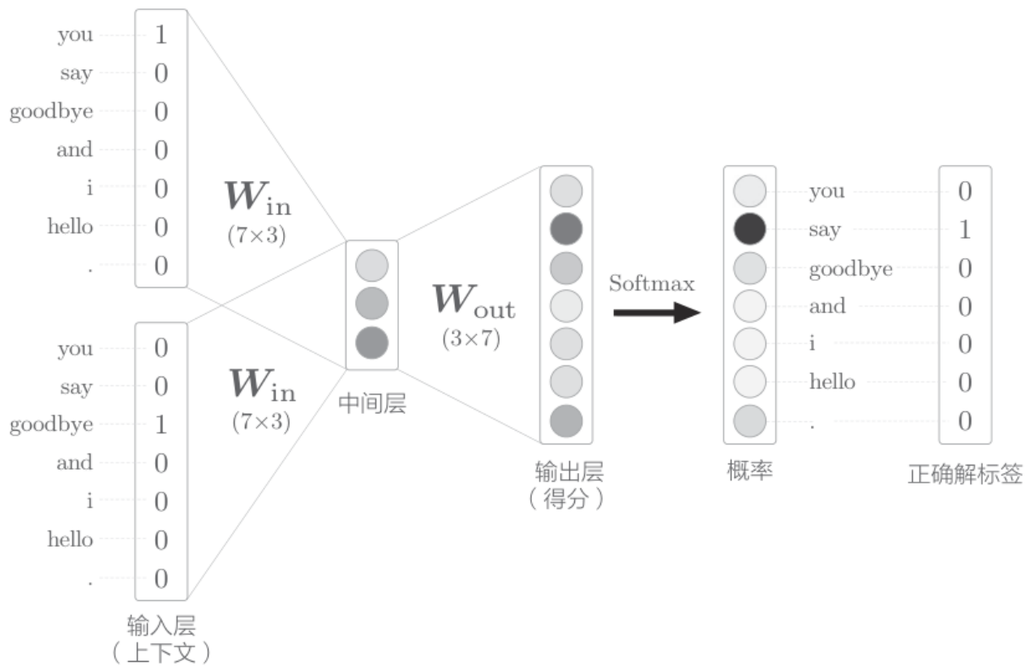
为了简单起见，我们这里上下文单词各取1个

模型搭建：

比如上面一句话，我们使用单词you和goodbye来预测中间词say，我们构建如下简单的全连接神经网络模型，
把you和goodbye的one-hot向量分别乘以W_{in}获得两个向量，求两个向量的平均值作为中间层，然后乘以W_{out}
获得输出，将输出经过softmax层变成概率分布，计算交叉熵损失。

单词编码：

我们最后可以拿训练好后的W_{in}或者W_{out}的行向量作为单词的编码，也可以都使用，简单起见我们拿W_{in}的每一行
作为单词的编码，可以看标题中那张图，实际上one-hot向量乘以W_{in}后出来的编码向量就是W_{in}的一个行向量



3.1.2 数据集预处理

读取文件，获得需要的句子，转换成list of list格式

第1步：获取所有句子放在一个列表中，格式为list of list

我们抽出部分句子作为全部的训练语料，数据越多越好，这里仅仅是为了演示这个word2vec的操作流程，选少一点计算速度更快

In [2]:
```
data=pd.read_csv("../data/dataset/imdb/labeledTrainData.tsv",sep="\t")
seqs=[str(seq) for seq in data.review][:10]
print("长度：",len(seqs))
print("第一条句子：","\n",seqs[0])
```

```
长度： 10
第一条句子：
 With all this stuff going down at the moment with MJ i've started listening to his music, watching the odd documentary here and there, watched The Wiz and watched Moonwalker again. Maybe i just want to get a certain insight into this guy who i thought was really cool in the eighties just to maybe make up my mind whether he is guilty or innocent. Moonwalker is part biography, part feature film which i remember going to see at the cinema when it was originally released. Some of it has subtle messages about MJ's feeling towards the press and also the obvious message of drugs are bad m'kay.<br /><br />Visually impressive but of course this is all about Michael Jackson so unless you remotely like MJ in anyway then you are going to hate this and find it boring. Some may call MJ an egotist for consenting to the making of this movie BUT MJ and most of his fans would say that he made it for the fans which if true is really nice of him.<br /><br />The actual feature film bit when it finally starts is only on for 20 minutes or so excluding the Smooth Criminal sequence and Joe Pesci is convincing as a psychopathic all powerful drug lord. Why he wants MJ dead so bad is beyond me. Because MJ overheard his plans? Nah, Joe Pesci's character ranted that he wanted people to know it is he who is supplying drugs etc so i dunno, maybe he just hates MJ's music.<br /><br />Lots of cool things in this like MJ turning into a car and a robot and the whole Speed Demon sequence. Also, the director must have had the patience of a saint when it came to filming the kiddy Bad sequence as usually directors hate working with one kid let alone a whole bunch of them performing a complex dance scene.<br /><br />Bottom line, this movie is for people who like MJ on one level or another (which i think is most people). If not, then stay away. It does try and give off a wholesome message and ironically MJ's bestest buddy in this movie is a girl! Michael Jackson is truly one of the most talented people ever to grace this planet but is he guilty? Well, with all the attention i've gave this subject....hmmm well i don't know because people can be different behind closed doors, i know this for a fact. He is either an extremely nice but stupid guy or one of the most sickest liars. I hope he is not the latter.
```

第2步：对str分词，获得list of list(word str)

In [3]:
```
def tokenize(seqs):
    patten=re.compile(r"[a-zA-Z]+")
    return [re.findall(patten,seq.lower()) for seq in seqs]
seqs=tokenize(seqs)
seqs[0]
```

Out[3]:
```
['with', 'all', 'this', 'stuff', 'going', 'down', 'at', 'the', 'moment', 'with', 'mj', 'i', 've', 'started', 'listening', 'to', 'his', 'music', 'watching', 'the', 'odd', 'documentary', 'here', 'and', 'there', 'watched', 'the',
'wiz', 'and', 'watched', 'moonwalker', 'again', 'maybe', 'i', 'just', 'want', 'to', 'get', 'a', 'certain', 'insight', 'into', 'this', 'guy', 'who', 'i', 'thought', 'was', 'really', 'cool', 'in', 'the', 'eighties', 'just', 'to',
'maybe', 'make', 'up', 'my', 'mind', 'whether', 'he', 'is', 'guilty', 'or', 'innocent', 'moonwalker', 'is', 'part', 'biography', 'part', 'feature', 'film', 'which', 'i', 'remember', 'going', 'to', 'see', 'at', 'the', 'cinema',
'when', 'it', 'was', 'originally', 'released', 'some', 'of', 'it', 'has', 'subtle', 'messages', 'about', 'mj', 's', 'feeling', 'towards', 'the', 'press', 'and', 'also', 'the', 'obvious', 'message', 'of', 'drugs', 'are', 'bad',
'm', 'kay', 'br', 'br', 'visually', 'impressive', 'but', 'of', 'course', 'this', 'is', 'all', 'about', 'michael', 'jackson', 'so', 'unless', 'you', 'remotely', 'like', 'mj', 'in', 'anyway', 'then', 'you', 'are', 'going', 'to',
'hate', 'this', 'and', 'find', 'it', 'boring', 'some', 'may', 'call', 'mj', 'an', 'egotist', 'for', 'consenting', 'to', 'the', 'making', 'of', 'this', 'movie', 'but', 'mj', 'and', 'most', 'of', 'his', 'fans', 'would', 'say', 't
hat', 'he', 'made', 'it', 'for', 'the', 'fans', 'which', 'if', 'true', 'is', 'really', 'nice', 'of', 'him', 'br', 'br', 'the', 'actual', 'feature', 'film', 'bit', 'when', 'it', 'finally', 'starts', 'is', 'only', 'on', 'for', 'm
inutes', 'or', 'so', 'excluding', 'the', 'smooth', 'criminal', 'sequence', 'and', 'joe', 'pesci', 'is', 'convincing', 'as', 'a', 'psychopathic', 'all', 'powerful', 'drug', 'lord', 'why', 'he', 'wants', 'mj', 'dead', 'so', 'ba
d', 'is', 'beyond', 'me', 'because', 'mj', 'overheard', 'his', 'plans', 'nah', 'joe', 'pesci', 's', 'character', 'ranted', 'that', 'he', 'wanted', 'people', 'to', 'know', 'it', 'is', 'he', 'who', 'is', 'supplying', 'drugs', 'et
c', 'so', 'i', 'dunno', 'maybe', 'he', 'just', 'hates', 'mj', 's', 'music', 'br', 'br', 'lots', 'of', 'cool', 'things', 'in', 'this', 'like', 'mj', 'turning', 'into', 'a', 'car', 'and', 'a', 'robot', 'and', 'the', 'whole', 'spe
ed', 'demon', 'sequence', 'also', 'the', 'director', 'must', 'have', 'had', 'the', 'patience', 'of', 'a', 'saint', 'when', 'it', 'came', 'to', 'filming', 'the', 'kiddy', 'bad', 'sequence', 'as', 'usually', 'directors', 'hate',
'working', 'with', 'one', 'kid', 'let', 'alone', 'a', 'whole', 'bunch', 'of', 'them', 'performing', 'a', 'complex', 'dance', 'scene', 'br', 'br', 'bottom', 'line', 'this', 'movie', 'is', 'for', 'people', 'who', 'like', 'mj', 'o
n', 'one', 'level', 'or', 'another', 'which', 'i', 'think', 'is', 'most', 'people', 'if', 'not', 'then', 'stay', 'away', 'it', 'does', 'try', 'and', 'give', 'off', 'a', 'wholesome', 'message', 'and', 'ironically', 'mj', 's', 'b
estest', 'buddy', 'in', 'this', 'movie', 'is', 'a', 'girl', 'michael', 'jackson', 'is', 'truly', 'one', 'of', 'the', 'most', 'talented', 'people', 'ever', 'to', 'grace', 'this', 'planet', 'but', 'is', 'he', 'guilty', 'well', 'w
ith', 'all', 'the', 'attention', 'i', 've', 'gave', 'this', 'subject', 'hmmm', 'well', 'i', 'don', 't', 'know', 'because', 'people', 'can', 'be', 'different', 'behind', 'closed', 'doors', 'i', 'know', 'this', 'for', 'a', 'fac
t', 'he', 'is', 'either', 'an', 'extremely', 'nice', 'but', 'stupid', 'guy', 'or', 'one', 'of', 'the', 'most', 'sickest', 'liars', 'i', 'hope', 'he', 'is', 'not', 'the', 'latter']
```

第3步：统计词频，制作**word2id**词典，再根据词典把每个单词映射成id，获得**list of list(int)**

In [4]:
```
def word_freq_count(seqs):
    words_all=[]
    for seq in seqs:
        words_all+=seq
    return Counter(words_all)
```

In [5]:
```
word_freq_dict=word_freq_count(seqs)
for i,item in enumerate(word_freq_dict.items()):
    print(item)
    if i==5: break
```

```
('with', 28)
('all', 7)
('this', 31)
('stuff', 1)
('going', 3)
('down', 1)
```

In [6]:
```
def freqdict2vocab(word_freq_dict):
    word_dict=sorted(word_freq_dict.items(),key=lambda x:x[1],reverse=True)
    vocab={"PAD":0,"UNK":1}
    for word,freq in word_dict:
        id=len(vocab)
        vocab[word]=id
    return vocab
```

In [7]:
```
word2id_dict=freqdict2vocab(word_freq_dict)
embedding_dim=len(word2id_dict)
embedding_dim
```

Out[7]:
```
1003
```

In [8]:
```
def word2id(seqs,vocab):
    out=[]
    for seq in seqs:
        out.append([vocab.get(word,1) for word in seq])
    return out
```

In [9]:
```
seqs=word2id(seqs,word2id_dict)
seqs[0]
```

Out[9]:
```
[10, 41, 9, 272, 85, 273, 42, 2, 274, 10, 27, 13, 136, 275, 276, 7, 21, 56, 277, 2, 278, 279, 280, 3, 86, 66, 2, 281, 3, 66, 137, 282, 87, 13, 48, 283, 7, 138, 4, 284, 285, 57, 9, 139, 22, 13, 286, 43, 88, 140, 8, 2, 287, 48,
7, 87, 141, 67, 142, 143, 288, 24, 6, 144, 20, 289, 137, 6, 89, 290, 89, 145, 23, 35, 13, 291, 85, 7, 146, 42, 2, 147, 36, 12, 43, 292, 293, 37, 5, 12, 68, 294, 295, 34, 27, 15, 148, 296, 2, 297, 3, 90, 2, 298, 149, 5, 150, 25,
49, 151, 299, 17, 17, 300, 301, 18, 5, 152, 9, 6, 41, 34, 153, 154, 30, 302, 44, 303, 38, 27, 8, 155, 58, 44, 25, 85, 7, 156, 9, 3, 304, 12, 157, 37, 305, 306, 27, 31, 307, 14, 308, 7, 2, 309, 5, 9, 16, 18, 27, 3, 32, 5, 21, 15
8, 91, 159, 19, 24, 59, 12, 14, 2, 158, 35, 50, 310, 6, 88, 160, 5, 311, 17, 17, 2, 161, 145, 23, 312, 36, 12, 313, 92, 6, 69, 45, 14, 93, 20, 30, 314, 2, 315, 316, 94, 3, 162, 163, 6, 164, 11, 4, 317, 41, 318, 319, 320, 70, 2
4, 165, 27, 95, 30, 49, 6, 166, 60, 61, 27, 321, 21, 322, 323, 162, 163, 15, 96, 324, 19, 24, 325, 39, 7, 71, 12, 6, 24, 22, 6, 326, 150, 327, 30, 13, 328, 87, 24, 48, 329, 27, 15, 56, 17, 17, 167, 5, 140, 97, 8, 9, 38, 27, 33
0, 57, 4, 331, 3, 4, 332, 3, 2, 98, 333, 334, 94, 90, 2, 168, 169, 51, 62, 2, 335, 5, 4, 336, 36, 12, 337, 7, 338, 2, 339, 49, 94, 11, 170, 340, 156, 171, 10, 26, 341, 342, 343, 4, 98, 344, 5, 172, 345, 4, 346, 347, 72, 17, 17,
99, 348, 9, 16, 6, 14, 39, 22, 38, 27, 45, 26, 349, 20, 100, 35, 13, 101, 6, 32, 39, 50, 28, 58, 350, 351, 12, 102, 352, 3, 103, 104, 4, 353, 149, 3, 354, 27, 15, 355, 356, 8, 9, 16, 6, 4, 357, 153, 154, 6, 358, 26, 5, 2, 32, 1
73, 39, 105, 7, 359, 9, 360, 18, 6, 24, 144, 106, 10, 41, 2, 361, 13, 136, 362, 9, 363, 364, 106, 13, 73, 33, 71, 61, 39, 74, 52, 174, 365, 366, 367, 13, 71, 9, 14, 4, 175, 24, 6, 75, 31, 176, 160, 18, 368, 139, 20, 26, 5, 2, 3
2, 369, 370, 13, 371, 24, 6, 28, 2, 177]
```

第4步：将上述预处理过程封装成类

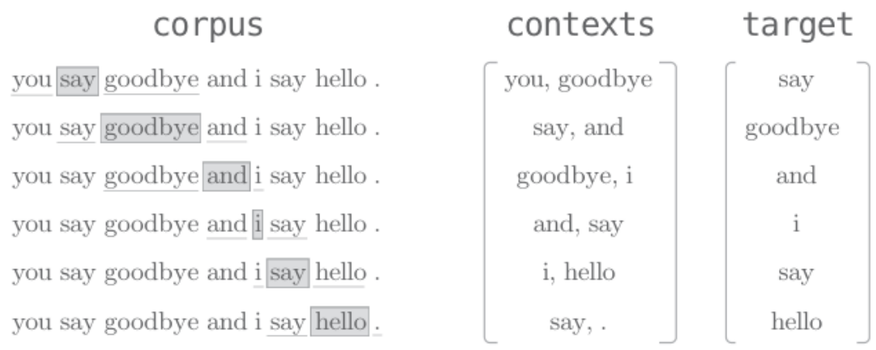
In [10]:
```
class IMDB_Preprocessing:
    def __init__(self,path=None,nums_seq=20):
        df=self.load(path)     # dataframe
        seqs=self.get_seqs(df,nums_seq)  # list of str
        seqs=self.tokenize(seqs)      # list of list-str
        self.word_freq_dict=self.word_freq_count(seqs)  # 词频字典
        self.vocab=self.freqdict2vocab(self.word_freq_dict) # word2id 词典
        self.seqs=self.word2id(seqs,self.vocab)

    def load(self,path):
        return pd.read_csv(path,sep="\t")
    def get_seqs(self,df,nums_seq):
        return [str(seq) for seq in df.review][:nums_seq]
    def tokenize(self,seqs):
        patten=re.compile(r"[a-zA-Z]+")
        return [re.findall(patten,seq.lower()) for seq in seqs]
    def word_freq_count(self,seqs):
        words_all=[]
        for seq in seqs:
            words_all+=seq
        return Counter(words_all)
    def freqdict2vocab(self,word_freq_dict):
        word_dict=sorted(word_freq_dict.items(),key=lambda x:x[1],reverse=True)
        vocab={"PAD":0,"UNK":1}
        for word,freq in word_dict:
            id=len(vocab)
            vocab[word]=id
        return vocab
    def word2id(self,seqs,vocab):
        out=[]
        for seq in seqs:
            out.append([vocab.get(word,1) for word in seq])
        return out
```

In [11]:
```
# test
path="../data/dataset/imdb/labeledTrainData.tsv"
data=IMDB_Preprocessing(path=path,nums_seq=20)
data.seqs[0]
```

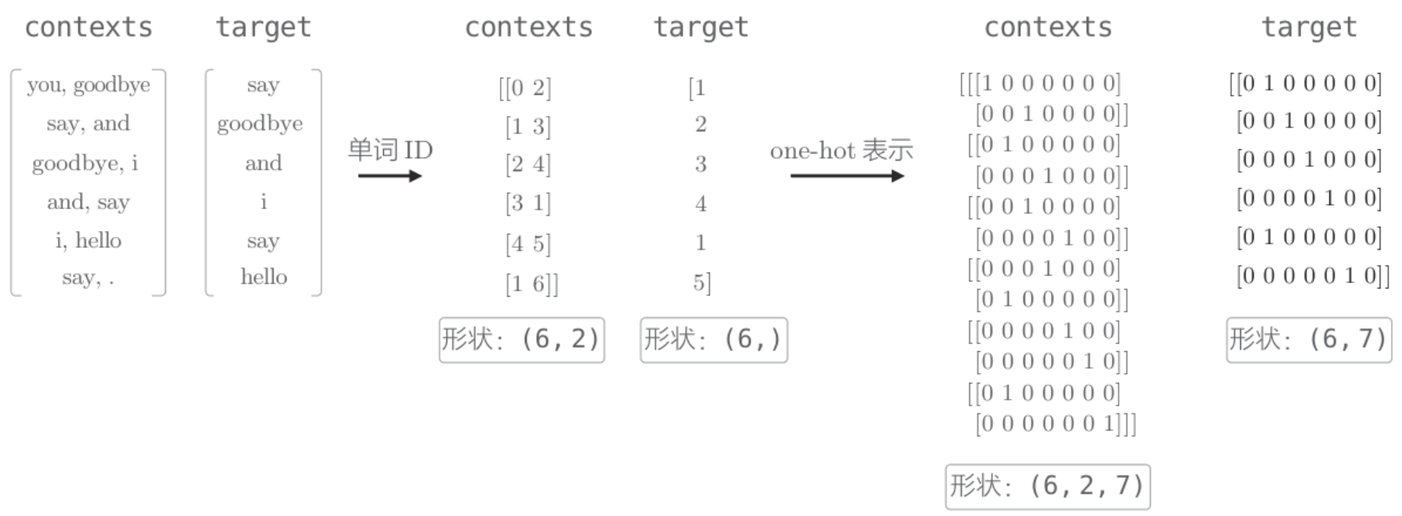
Out[11]:
```
[17, 41, 11, 509, 178, 134, 30, 2, 270, 17, 54, 13, 179, 510, 511, 7, 25, 78, 512, 2, 513, 514, 135, 3, 57, 136, 2, 515, 3, 136, 271, 516, 180, 13, 47, 517, 7, 272, 4, 273, 518, 58, 11, 274, 31, 13, 275, 22, 79, 276, 9, 2, 519,
47, 7, 180, 73, 89, 90, 277, 520, 32, 6, 278, 26, 521, 271, 6, 91, 522, 91, 279, 18, 48, 13, 523, 178, 7, 112, 30, 2, 92, 37, 10, 22, 524, 525, 34, 5, 10, 42, 526, 527, 33, 54, 12, 181, 280, 2, 281, 3, 74, 2, 528, 282, 5, 283,
27, 59, 284, 529, 8, 8, 285, 530, 19, 5, 113, 11, 6, 41, 33, 286, 287, 38, 288, 43, 182, 36, 54, 9, 183, 80, 43, 27, 178, 7, 289, 11, 3, 290, 10, 137, 34, 184, 531, 54, 28, 532, 16, 533, 7, 2, 291, 5, 11, 20, 19, 54, 3, 39, 5,
25, 185, 75, 138, 14, 32, 114, 10, 16, 2, 185, 48, 49, 534, 6, 79, 292, 5, 293, 8, 8, 2, 186, 279, 18, 187, 37, 10, 294, 139, 6, 60, 44, 16, 140, 26, 38, 535, 2, 536, 537, 141, 3, 295, 296, 6, 297, 15, 4, 538, 41, 539, 540, 54
1, 61, 32, 298, 54, 188, 38, 59, 6, 189, 93, 94, 54, 542, 25, 543, 544, 295, 296, 12, 81, 545, 14, 32, 546, 45, 7, 115, 10, 6, 32, 31, 6, 547, 283, 190, 38, 13, 548, 180, 32, 47, 549, 54, 12, 78, 8, 8, 299, 5, 276, 191, 9, 11,
36, 54, 550, 58, 4, 300, 3, 4, 551, 3, 2, 142, 301, 552, 141, 74, 2, 116, 143, 40, 50, 2, 553, 5, 4, 554, 37, 10, 302, 7, 303, 2, 555, 59, 141, 15, 144, 556, 289, 304, 17, 21, 557, 305, 145, 4, 142, 558, 5, 192, 559, 4, 560, 30
6, 76, 8, 8, 193, 561, 11, 20, 6, 16, 45, 31, 36, 54, 44, 21, 307, 26, 82, 48, 13, 117, 6, 39, 45, 49, 23, 80, 562, 194, 10, 146, 563, 3, 147, 195, 4, 564, 282, 3, 565, 54, 12, 566, 567, 9, 11, 20, 6, 4, 568, 286, 287, 6, 569,
21, 5, 2, 39, 196, 45, 83, 7, 570, 11, 571, 19, 6, 32, 278, 62, 17, 41, 2, 308, 13, 179, 572, 11, 573, 574, 62, 13, 118, 29, 115, 94, 45, 84, 35, 309, 310, 575, 576, 13, 115, 11, 16, 4, 148, 32, 6, 95, 28, 311, 292, 19, 577, 27
4, 26, 21, 5, 2, 39, 578, 579, 13, 580, 32, 6, 23, 2, 312]
```

3.1.3 CBOW数据集的构建

首先，我们从一句话里面基于长度为3的窗口进行滑动，把窗口中的3个词拿出来，
中间的词作为label，前后两个词作为input
这样，下面单独一句话我们就构建了6个样本标签对，以此类推，我们需要把所有的
句子拿出来，同样处理，把得到的样本标签对放在一起组成一个完整的训练集



上面的数据集我们还不能直接使用，需要先将word转换为id，再将id转换为one-hot向量，才能丢到模型中训练
注意不同框架里面labels的格式，大部分框架labels不是one-hot向量，直接是标签数字，下面这张图是课本作者用numpy实现的交叉熵损失，所以他把labels也都转成了one-hot向量，我们使用框架交叉熵损失，就不用转换了



批量计算过程：上面是6个样本，假设batch_size=4，我们一次性拿出4个样本的立方体，立方体形状（4，2，7）
假设W_{in}形状为（7，3），就是把7维向量转换为3维，将立方体与W_{in}相乘，输出形状（4，2，3）
表示4个样本的输出，每个样本产成2个3维的输出向量，正是上下文单词分别编码的结果，然后我们按照dim=1这个维度把两个向量相加，获得这批样本的中间层向量（4，1，3），再squeeze变成（4，3），这个形状表示4批样本的特征，每个特征占一行，这正是最基本的批运算的格式，假设是10分类问题，然后我们最终得到预测概率分布（4，10），表示4个样本的输出分布。

第1步：基于长度为3的窗口在每个句子上滑动取样本和标签

In [12]:
```
def seq2sample(seq,window_size=3): # list of tensor
    x,labels=[],[]
    n=len(seq)
    seq=torch.tensor(seq,dtype=torch.int64)
    for i in range(n-window_size+1):
        window=seq[i:i+window_size]
        mid=window_size//2
        x.append(torch.concat([window[:mid],window[mid+1:]]))
        labels.append(window[mid])
    return x,labels
```

In [13]:
```
x,labels=seq2sample(data.seqs[0])
print("前5个样本：","\n",x[:5])
print("前5个标签：","\n",labels[:5])
```

```
前5个样本：
 [tensor([17, 11]), tensor([ 41, 509]), tensor([ 11, 178]), tensor([509, 134]), tensor([178,  30])]
前5个标签：
 [tensor(41), tensor(11), tensor(509), tensor(178), tensor(134)]
```

In [14]:
```
def creat_samples(seqs):
    x=[]
    labels=[]
    for seq in seqs:
        a,b=seq2sample(seq)
        x+=a
        labels+=b
    return x,labels
```

In [15]:
```
x,labels=creat_samples(data.seqs)
len(x),len(labels)
```

Out[15]:
```
(4575, 4575)
```

第2步：定义一个变换，能把x转为one hot vector,需要事先指定one hot vector的维数

In [16]:
```
def seq2onehot(seq,embedding_dim):
    if seq.dim()==1:
        L=seq.shape[0]
        D=embedding_dim
        one_hot=torch.zeros(size=[L,D])
        for i in range(L):
            one_hot[i,seq[i]]=1
    elif seq.dim()==2:
        B=seq.shape[0]
        L=seq.shape[1]
        D=embedding_dim
        one_hot=torch.zeros(size=[B,L,D])
        for i in range(B):
            for j in range(L):
                one_hot[i,j,seq[i,j]]=1
    return one_hot
```

In [17]:
```
seq2onehot(x[0],30)
```

Out[17]:
```
tensor([[0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1.,
         0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
        [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0.,
         0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.]])
```

第3步：制作Dataset类

In [18]:
```
class Build_CBOW_Dataset((nndl.utils.Dataset)):
    def __init__(self,seqs=None,vocab=None):
        self.seqs=seqs
        self.vocab=vocab
        self.embedding_dim=len(vocab)
        x,labels=self.creat_datasets(self.seqs)  # list of tensor

        self.x=self.dtype_transform(x)             # tensor,dim=2
        self.labels=self.dtype_transform(labels)   # tensor,dim=1
    def __len__(self):
        return self.labels.shape[0]
    def __getitem__(self,idx):
        return self.seq2onehot(self.x[idx],self.embedding_dim),self.labels[idx]

    def seq2sample(self,seq,window_size=3): # list of tensor
        x,labels=[],[]
        n=len(seq)
        seq=torch.tensor(seq,dtype=torch.int64)
        for i in range(n-window_size+1):
            window=seq[i:i+window_size]
            mid=window_size//2
            x.append(torch.concat([window[:mid],window[mid+1:]]))
            labels.append(window[mid])
        return x,labels
```

```
    def creat_datasets(self,seqs):  # list of tensor
        x,labels=[],[]
        for seq in seqs:
            a,b=self.seq2sample(seq)
            x+=a
            labels+=b
        return x,labels
    def seq2onehot(self,seq,embedding_dim):
        if seq.dim()==1:
            L=seq.shape[0]
            D=embedding_dim
            one_hot=torch.zeros(size=[L,D])
            for i in range(L):
                one_hot[i,seq[i]]=1
        elif seq.dim()==2:
            B=seq.shape[0]
            L=seq.shape[1]
            D=embedding_dim
            one_hot=torch.zeros(size=[B,L,D])
            for i in range(B):
                for j in range(L):
                    one_hot[i,j,seq[i,j]]=1
        return one_hot
    def dtype_transform(self,input):
        return torch.tensor([x.tolist() for x in input])
```

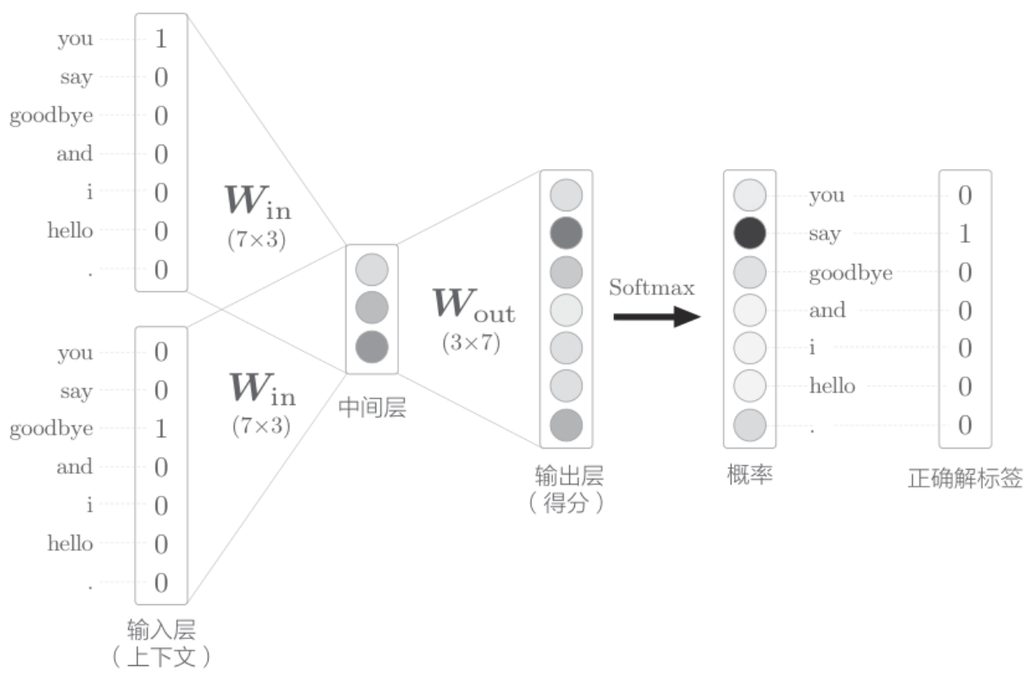
In [19]:
```
trainset=Build_CBOW_Dataset(seqs=data.seqs,vocab=data.vocab)
trainloader=nndl.utils.DataLoader(trainset,batch_size=16)
len(trainset),len(trainloader)
```

Out[19]: (4575, 286)

In [20]:
```
for a,b in trainloader:
    print(a.shape)
    print(b)
    break
```

```
torch.Size([16, 2, 1636])
tensor([ 41,  11, 509, 178, 134,  30,   2, 270,  17,  54,  13, 179, 510, 511,
          7,  25])
```

3.1.4 SimpleCBOW模型的搭建与训练



定义模型

In [21]:
```
class SimpleCBOW(nn.Module):
    def __init__(self,num_embeddings,embedding_dim):
        super(SimpleCBOW,self).__init__()
        self.num_embeddings=num_embeddings
        self.embedding_dim=embedding_dim
        self.w_in=nn.Parameter(torch.randn(size=[num_embeddings,embedding_dim]))
        # self.bn=nn.BatchNorm1d(num_features=embedding_dim)
        self.w_out=nn.Parameter(torch.randn(size=[embedding_dim,num_embeddings]))
        self.softmax=nn.Softmax(dim=1)
    # def __call__(self,x):
        # return self.forward(x)
    def forward(self,x):
        x=torch.matmul(x,self.w_in)  # (16,2,d)
        x=x.sum(dim=1)*torch.tensor(0.5) # (16,1,d)
        x=x.squeeze(1) # (16,d)
        # x=self.bn(x)
        x=torch.matmul(x,self.w_out) # (16,max_num)
        x=self.softmax(x)
        return x
```

批量计算过程：上面是N个样本，batch_size=16，我们一次性拿出16个样本的立方体，立方体形状（16，2，max_num）
假设W_{in}形状为（max_num，d），就是把max_num维向量转换为d维，将立方体与W_{in}相乘，输出形状（16，2，d）
表示16个样本的输出，每个样本产成2个d维的输出向量，正是上下文单词分别编码的结果，然后我们按照dim=1
这个维度把两个向量相加，获得这批样本的中间层向量（16，1，d），再squeeze变成（16，d），这个形状表示
16批样本的特征，每个特征占一行，这正是最基本的批运算的格式，这里是max_num分类问题，然后我们最终得到预测
概率分布（16，max_num），表示16个样本的输出分布。

实例化模型，损失函数，优化器，评估器

这里损失函数，优化器，评估器就用自己写的了

In [22]:
```
model=SimpleCBOW(num_embeddings=len(data.vocab),embedding_dim=16)
loss_fn=nndl.CrossEntropyLoss()
metric=nndl.Accuracy()
```

实例化Runner，开始训练

In [23]:
```
# opt=nndl.optim.SGD(model.parameters(),lr=0.6)
# runner=nndl.RunnerV2(model=model,loss_fn=loss_fn,optimizer=opt,metric=metric)
# runner.train(train_loader=trainloader,dev_loader=trainloader,num_epochs=64,log_stride=1024)
```

模型参数保存

In [24]:
```
# w_in_save=model.w_in
# w_out_save=model.w_out
# np.savetxt("./losslandscape/w_in.txt",w_in_save.detach().numpy())
# np.savetxt("./losslandscape/w_out.txt",w_out_save.detach().numpy())
# torch.save(list(model.parameters()),"./losslandscape/last_weights.pt")
```

模型可视化

初始化LossLandscape对象

In [25]:
```
# losslandscape=nndl.utils.LossLandscape(model=model,loss_fn=loss_fn,dataloader=trainloader,runner=runner)
```

加载之前跑过的比较好的最优点作为坐标中心

In [26]:
```
# losslandscape.load("./losslandscape/weights_star.pt")
```

开始运行作图

In [27]:
```
# losslandscape.run(range=1,nums=40,mode="pca")
```

模型效果分析

结论：效果一般般

原因1：数据问题，这是如下表中的数据集的800分类问题，2000个样本。
把2000个样本线性投射两次，然后分成800个类别，可能类别太多，两次线性映射实在是分不开

原因2：由于只有W_{in}和W_{out}两个线性层，这个简单模型基本可以断定就是一个大碗，在这组少量样本下，
最优点顶多就那样了，模型复杂度就这样了。

原因3：把W_{out}的800个列向量看成800个类别轴，交叉熵损失只是增大中间层向量与某条轴的内积大小，
只是拉近了样本与"正例"的距离，但是没有排斥与其他轴的距离，导致优化过程中，样本与代表"负例"的轴之间
的关系是不确定的，从而很难优化到很好的结果

原因4：交叉熵损失确实增大了样本与"正例"的距离，也拉远了与其他轴的距离，但是其他轴选的太多，这个拉远
效果被平摊后，跟没拉一样，需要减少"负例"

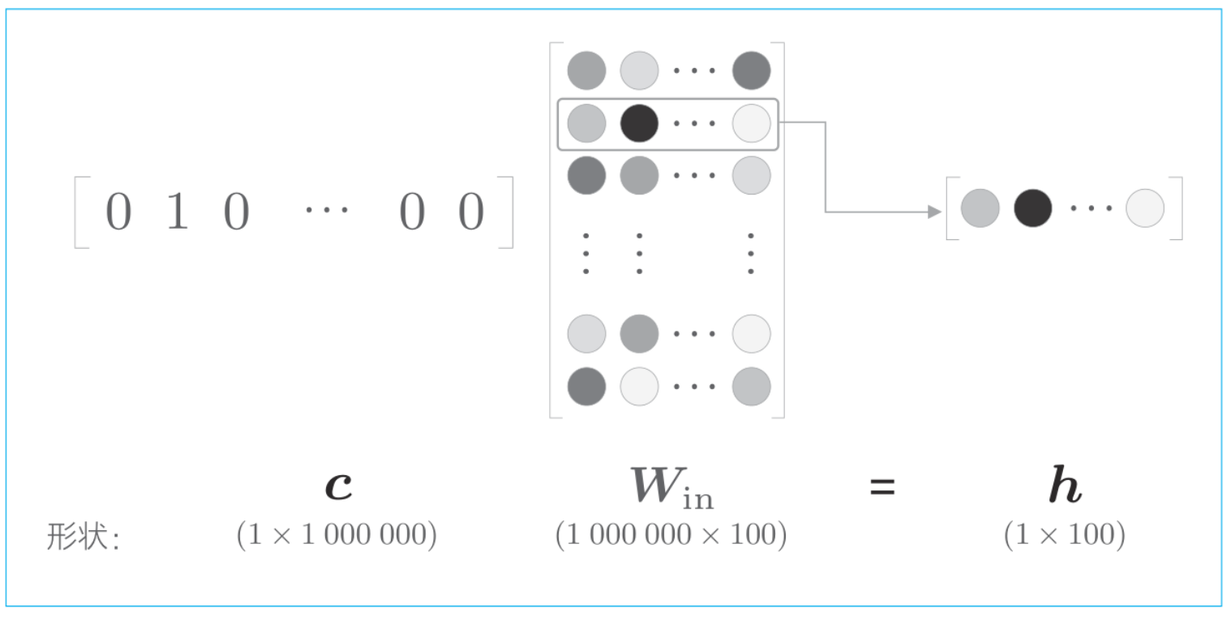
上面只是粗略找了一下可能的原因，不一定准确，有兴趣的可以慢慢做实验一个一个试

Dataset	features		labels
	左边的单词	右边的单词	中间的单词
x_1	[0, 0, 0, . . . , 1, 0, . . . , 0, . . .]	[0, 0, 0, . . . , 1, 0, . . . , 1, . . .]	285
x_2	[0, 0, 0, . . . , 1, 0, . . . , 0, . . .]	[0, 0, 0, . . . , 1, 0, . . . , 1, . . .]	633
x_3	[0, 0, 0, . . . , 1, 0, . . . , 0, . . .]	[0, 0, 0, . . . , 1, 0, . . . , 1, . . .]	800
x_4	[0, 0, 0, . . . , 1, 0, . . . , 0, . . .]	[0, 0, 0, . . . , 1, 0, . . . , 1, . . .]	12
x_5	[0, 0, 0, . . . , 1, 0, . . . , 0, . . .]	[0, 0, 0, . . . , 1, 0, . . . , 1, . . .]	76
x_6	[0, 0, 0, . . . , 1, 0, . . . , 0, . . .]	[0, 0, 0, . . . , 1, 0, . . . , 1, . . .]	55
x_7	[0, 0, 0, . . . , 1, 0, . . . , 0, . . .]	[0, 0, 0, . . . , 1, 0, . . . , 1, . . .]	63
x_8	[0, 0, 0, . . . , 1, 0, . . . , 0, . . .]	[0, 0, 0, . . . , 1, 0, . . . , 1, . . .]	257
x_9	[0, 0, 0, . . . , 1, 0, . . . , 0, . . .]	[0, 0, 0, . . . , 1, 0, . . . , 1, . . .]	346
x_10	[0, 0, 0, . . . , 1, 0, . . . , 0, . . .]	[0, 0, 0, . . . , 1, 0, . . . , 1, . . .]	417
x_11	[0, 0, 0, . . . , 1, 0, . . . , 0, . . .]	[0, 0, 0, . . . , 1, 0, . . . , 1, . . .]	527
x_12	[0, 0, 0, . . . , 1, 0, . . . , 0, . . .]	[0, 0, 0, . . . , 1, 0, . . . , 1, . . .]	511
x_13	[0, 0, 0, . . . , 1, 0, . . . , 0, . . .]	[0, 0, 0, . . . , 1, 0, . . . , 1, . . .]	38
x_14	[0, 0, 0, . . . , 1, 0, . . . , 0, . . .]	[0, 0, 0, . . . , 1, 0, . . . , 1, . . .]	600
x_15	[0, 0, 0, . . . , 1, 0, . . . , 0, . . .]	[0, 0, 0, . . . , 1, 0, . . . , 1, . . .]	315
x_16	[0, 0, 0, . . . , 1, 0, . . . , 0, . . .]	[0, 0, 0, . . . , 1, 0, . . . , 1, . . .]	123
. . .	. . . . .	. . . . .	. . .
x_2000	[0, 0, 0, . . . , 1, 0, . . . , 0, . . .]	[0, 0, 0, . . . , 1, 0, . . . , 0, . . .]	486

3.2 CBOW模型的改进

3.2.1 CBOW的改进①：

x*W_in是one-hot向量乘以一个矩阵，没必要按照矩阵乘法全部元素进行运算
实际上只要取出W_in的相应的行即可，为此我们实现一个特殊的层完成这种乘法
注：取出一行的操作计算速度比向量乘以矩阵的每一列快多了，改进体现在计算速度上



In [28]:
```
class SuperW(nn.Module):
    def __init__(self,size,dim=0):
        # dim参数表示对取行还是取列
        super(SuperW,self).__init__()
        self.dim=dim
        self.W=nn.Parameter(torch.randn(size=size))
    def __call__(self,idx):
        return self.forward(idx)
    def forward(self,idx):
        if self.dim==0:
            return self.W[idx]
        elif self.dim==1:
            if idx.dim()==1:
                return self.W.T[idx].T
            elif idx.dim()==2:
                return self.W.T[idx].transpose(1,2)
            else:
                raise "dim error"
        else:
            raise "dim error"
```

In [29]:
```
# test1
w=SuperW(size=[8,4])
idx=torch.tensor([0,7])
print(w.W)
print(idx)
print(w(idx))
```

```
Parameter containing:
tensor([[-1.3916,  0.2485, -0.2912,  1.7717],
        [ 0.6727, -0.4433, -0.5900, -0.8899],
        [ 0.1593,  1.3709, -0.1153, -0.6637],
        [ 1.8539,  0.4411,  1.7560, -0.9298],
        [ 0.2277, -0.8553,  0.4410,  0.9782],
        [ 0.6290, -1.4591, -0.6601, -0.8451],
        [-0.0282,  0.3730,  0.8637, -1.3661],
        [ 1.5437, -0.0300,  1.3951,  0.8590]], requires_grad=True)
tensor([0, 7])
tensor([[-1.3916,  0.2485, -0.2912,  1.7717],
        [ 1.5437, -0.0300,  1.3951,  0.8590]], grad_fn=<IndexBackward0>)
```

In [30]:
```
# test2,行批采样
w=SuperW(size=[8,4])
idx=torch.tensor([[0,7],[1,6]])
print(w.W)
print(idx)
print(w(idx))
```

```
Parameter containing:
tensor([[ 0.6843, -0.6288,  1.1483, -1.2787],
        [ 0.4187,  1.8975,  0.6673,  0.0796],
        [ 1.2347,  0.5808, -0.4587,  0.4245],
        [-0.5086,  1.1555,  0.4066, -1.1242],
        [ 0.0134, -0.6531, -0.8338,  0.1273],
        [-0.4157, -1.6623, -0.5367, -0.0213],
        [-1.9127, -1.5569, -1.4756,  1.3046],
        [-0.8124, -0.1626, -1.0284, -0.1196]], requires_grad=True)
tensor([[0, 7],
        [1, 6]])
tensor([[[ 0.6843, -0.6288,  1.1483, -1.2787],
         [-0.8124, -0.1626, -1.0284, -0.1196]],

        [[ 0.4187,  1.8975,  0.6673,  0.0796],
         [-1.9127, -1.5569, -1.4756,  1.3046]]], grad_fn=<IndexBackward0>)
```

In [31]:
```
# test3 列采样
w=SuperW(size=[8,4],dim=1)
idx=torch.tensor([0,2])
print(w.W)
print(idx)
print(w(idx))
```

```
Parameter containing:
tensor([[-0.1190,  0.1744,  0.1832, -0.8257],
        [ 0.2896,  0.1395,  1.3413, -0.1646],
        [-0.7923, -0.1550, -0.5213,  0.5936],
        [ 0.0176,  0.6556,  2.8821, -0.8821],
        [-0.1249,  1.1001, -0.1824, -0.4757],
        [-1.2422,  0.8956,  0.3325,  0.1822],
        [ 0.2024,  0.1500,  0.9495,  1.1472],
        [-0.4680,  1.9373, -0.4617,  1.0300]], requires_grad=True)
tensor([0, 2])
tensor([[-0.1190,  0.1832],
        [ 0.2896,  1.3413],
        [-0.7923, -0.5213],
        [ 0.0176,  2.8821],
        [-0.1249, -0.1824],
        [-1.2422,  0.3325],
        [ 0.2024,  0.9495],
        [-0.4680, -0.4617]], grad_fn=<PermuteBackward0>)
```

In [32]:
```
# test4 列批采样
w=SuperW(size=[8,4],dim=1)
idx=torch.tensor([[0,2],[1,3]])
print(w.W)
print(idx)
print(w(idx))
```

```
Parameter containing:
tensor([[-1.3421, -1.1918, -0.5107,  0.4092],
        [-0.5589,  1.5297, -0.3791,  0.3314],
        [-0.1975, -0.4109,  0.1915, -0.5284],
        [-1.3283, -1.0868,  0.0094,  1.5266],
        [ 1.0874,  1.1190, -1.5466, -0.1901],
        [ 0.5937,  1.7216,  0.7528,  0.6584],
        [ 1.1770,  1.1499,  1.1301, -0.2430],
        [ 0.3912,  0.7581,  0.8894, -1.7047]], requires_grad=True)
tensor([[0, 2],
        [1, 3]])
tensor([[[-1.3421, -0.5107],
         [-0.5589, -0.3791],
         [-0.1975,  0.1915],
         [-1.3283,  0.0094],
         [ 1.0874, -1.5466],
         [ 0.5937,  0.7528],
         [ 1.1770,  1.1301],
         [ 0.3912,  0.8894]],

        [[-1.1918,  0.4092],
         [ 1.5297,  0.3314],
         [-0.4109, -0.5284],
         [-1.0868,  1.5266],
         [ 1.1190, -0.1901],
         [ 1.7216,  0.6584],
         [ 1.1499, -0.2430],
         [ 0.7581, -1.7047]]], grad_fn=<TransposeBackward0>)
```

3.2.2 CBOW的改进②：

h*W_out=p,其中p是一个800维的向量，是输出的概率分布，其中一个概率是分类正确的概率，
其他全都是分类错误的概率。
一、我们可以考虑将其变成一个二分类问题，直接用h乘以W_out正确类别那一列，然后
使用logistic将其转换为概率，但是这样对错误样本的的判断就比较粗略，效果可能不会太好。
二、我们依旧使用Softmax多分类，因为这样错误类别的概率会出现在分母，就会考虑到分类
错误的概率，只不过我们不用所有剩下的样本都来计算概率，我们按数据集分布随机取出比如19个负例，
然后h*W_out[正例的idx，负例的idx]=p，此时p是一个20维向量，就可以当成一个20个类别的分类问题。

负例采样器

在样本足够多的情况下，我们按照词频对样本进行采样，有时候为了增加低频
词汇被抽中的可能性，会对对词频进行log缩放，或者对将概率进行0.75的乘方，
但是我们这里为了简单，就不这么处理了，直接按词频采样。

In [33]:
```
def get_word_distribution(word_freq_dict,vocab):
    # pad和unk不在原始词典中，词频按0处理
    weights=[word_freq_dict.get(key,0) for key in vocab.keys()]
    return torch.tensor(weights)
```

In [34]:
```
word_distribution=get_word_distribution(word_freq_dict,trainset.vocab)
word_distribution[:10]
```

Out[34]:
```
tensor([  0,   0, 125,  78,  59,  55,  47,  43,  20,  32])
```

In [35]:
```
def sampler(word_distribution,n,label):
    word_weights=word_distribution.detach().clone().to(torch.float)
    label=label.detach().clone().to(torch.int64)
    if label.dim()==0:
        sample=torch.multinomial(word_weights, n)
        while label in sample:
            sample=torch.multinomial(word_weights, n)
    elif (label.dim()==1 and label.shape[0]==1):
        sample=torch.multinomial(word_weights, n)
        while label in sample:
            sample=torch.multinomial(word_weights, n)
    elif label.dim()==1:
        k=label.shape[0]
        batch_word_weights=[word_weights for i in range(k)]
        labels=[label[i] for i in range(k)]
        sample=[sampler(w,n,z) for (w,z) in zip(batch_word_weights,labels)]
        sample=torch.tensor([x.tolist() for x in sample])
    else:
        raise "label dim error"
    return sample
```

In [36]:
```
# test1
sampler(word_distribution,5,torch.tensor(255))
```

Out[36]:
```
tensor([  7, 280, 214, 607, 139])
```

In [37]:
```
# test2
sampler(word_distribution,10,torch.tensor([5,12,255]))
```

Out[37]:
```
tensor([[361,   2,   4,  21, 118, 751, 742, 697, 277,  12],
        [ 15, 213,  14,  50,  51, 602,  28,  70, 295, 724],
        [650,  93,  51,  29,   6, 415, 167, 429,   2, 154]])
```

In [38]:
```
# 正负例id合并
positive_label=torch.tensor([5,12,255]).reshape(-1,1)
negative_label=sampler(word_distribution,10,torch.tensor([5,12,255]))
idx=torch.concat([positive_label,negative_label],dim=1)
```

```
print("正例id: ",positive_label)
print("负例id: ",negative_label)
print("合并id: ",idx)
```

```
正例id:  tensor([[  5],
        [ 12],
        [255]])
负例id:  tensor([[  43, 1061,  186,  590,  208,    2,  306, 1010,    3,  798],
        [1099,  147,  300,   17,  131,  368,  337,  847,    6,  182],
        [ 270,  568,    2,    7,  313,  141,    3,  929,   94,    9]])
合并id:  tensor([[   5,   43, 1061,  186,  590,  208,    2,  306, 1010,    3,  798],
        [  12, 1099,  147,  300,   17,  131,  368,  337,  847,    6,  182],
        [ 255,  270,  568,    2,    7,  313,  141,    3,  929,   94,    9]])
```

3.2.3 重构preprocessing、dataset与dataloader类

预处理

In [39]:
```
class IMDB_Preprocessing:
    """
    输入：imdb文件路径，取多少条句子
    重要属性：
        seqs：list of list
        word_freq_dict、vocab
    """
    def __init__(self,path=None,nums_seq=20):
        df=self.load(path)     # dataframe
        seqs=self.get_seqs(df,nums_seq)  # list of str
        seqs=self.tokenize(seqs)      # list of list-str
        self.word_freq_dict=self.word_freq_count(seqs)  # 词频字典
        self.vocab=self.freqdict2vocab(self.word_freq_dict) # word2id 词典
        self.seqs=self.word2id(seqs,self.vocab)

    def load(self,path):
        return pd.read_csv(path,sep="\t")
    def get_seqs(self,df,nums_seq):
        return [str(seq) for seq in df.review][:nums_seq]
    def tokenize(self,seqs):
        patten=re.compile(r"[a-zA-Z]+")
        return [re.findall(patten,seq.lower()) for seq in seqs]
    def word_freq_count(self,seqs):
        words_all=[]
        for seq in seqs:
            words_all+=seq
        return Counter(words_all)
    def freqdict2vocab(self,word_freq_dict):
        word_dict=sorted(word_freq_dict.items(),key=lambda x:x[1],reverse=True)
        vocab={"PAD":0,"UNK":1}
        for word,freq in word_dict:
            id=len(vocab)
            vocab[word]=id
        return vocab
    def word2id(self,seqs,vocab):
        out=[]
        for seq in seqs:
            out.append([vocab.get(word,1) for word in seq])
        return out
```

In [40]:
```
# test
path="../data/dataset/imdb/labeledTrainData.tsv"
data=IMDB_Preprocessing(path=path,nums_seq=100)
seqs=data.seqs
word_freq_dict=data.word_freq_dict
vocab=data.vocab
```

dataset类

In [41]:
```
class Build_CBOW_Dataset((nndl.utils.Dataset)):
    def __init__(self,data=None,window_size=3):
        self.seqs=data.seqs
        self.vocab=data.vocab
        self.word_freq_dict=data.word_freq_dict
        self.embedding_dim=len(vocab)
        self.x,self.labels=self.creat_datasets(self.seqs,window_size)

        self.word_distribution=self.get_word_distribution(self.word_freq_dict,self.vocab)

    def seq2sample(self,seq,window_size):
        x,labels=[],[]
        n=len(seq)
        seq=torch.tensor(seq,dtype=torch.int64)
        for i in range(n-window_size+1):
            window=seq[i:i+window_size]
            mid=window_size//2
            x.append(torch.concat([window[:mid],window[mid+1:]]))
            labels.append(window[mid])
        return x,labels  # list of tensor
    def creat_datasets(self,seqs,window_size):  # list of tensor
        x,labels=[],[]
        for seq in seqs:
            a,b=self.seq2sample(seq,window_size)
            x+=a
            labels+=b
        x=torch.tensor([xx.tolist() for xx in x])
        labels=torch.tensor([yy.tolist() for yy in labels])
        return x,labels # 2 dim tensor , 1 dim tensor

    def get_word_distribution(self,word_freq_dict,vocab):
        weights=[word_freq_dict.get(key,0) for key in vocab.keys()]
        return torch.tensor(weights)

    def __len__(self):
        return self.labels.shape[0]
    def __getitem__(self,idx):
        return self.x[idx],self.labels[idx]
```

dataloader类

说明：这里每次取出一个batch的idx都是随机的，如果想固定下来，可以遍历一个epoch，
把第一次遍历的数据存下来，后面就在这个固定的数据上训练即可

第一种写法，在iter中直接从dataset中获取batch数据，然后处理，通过yield返回

In [42]:
```
class IMDB_DataLoader(nndl.utils.DataLoader):
    def __init__(self,dataset=None,batch_size=16,c=20):
        super().__init__(dataset=trainset,batch_size=batch_size)
        self.c=c
        self.word_distribution=dataset.word_distribution

    def get_shuffled_index(self):
        if self.shuffle:
            index = self.random_sample()
        else:
            index = self.sequential_sample()
        return index

    def __iter__(self):
        N=len(self)
        index=self.get_shuffled_index()
        n = len(self.dataset)
        stride = self.batch_size
        for i in range(N):
            batch_index = index[i * stride:min(i * stride + stride, n)]
            x,y = self.dataset[batch_index]
            positive_label=y.reshape(-1,1)
            negative_label=self.sampler(self.word_distribution,self.c-1,y)
            idx=torch.concat([positive_label,negative_label],dim=1)
            yield x,idx

    def sampler(self,word_distribution,n,label):
        word_weights=word_distribution.detach().clone().to(torch.float)
        label=label.detach().clone().to(torch.int64)
        if label.dim()==0:
```

```
            sample=torch.multinomial(word_weights, n)
            while label in sample:
                sample=torch.multinomial(word_weights, n)
        elif (label.dim()==1 and label.shape[0]==1):
            sample=torch.multinomial(word_weights, n)
            while label in sample:
                sample=torch.multinomial(word_weights, n)
        elif label.dim()==1:
            k=label.shape[0]
            batch_word_weights=[word_weights for i in range(k)]
            labels=[label[i] for i in range(k)]
            sample=[sampler(w,n,z) for (w,z) in zip(batch_word_weights,labels)]
            sample=torch.tensor([x.tolist() for x in sample])
        else:
            raise "label dim error"
        return sample
```

第二种写法：在外面定义好collate_fn函数，然后交给dataloader即可

In [43]:
```
def sampler(word_distribution,n,label):
    word_weights=word_distribution.detach().clone().to(torch.float)
    label=label.detach().clone().to(torch.int64)
    if label.dim()==0:
        sample=torch.multinomial(word_weights, n)
        while label in sample:
            sample=torch.multinomial(word_weights, n)
    elif (label.dim()==1 and label.shape[0]==1):
        sample=torch.multinomial(word_weights, n)
        while label in sample:
            sample=torch.multinomial(word_weights, n)
    else:
        raise "label dim error"
    return sample
def collate_fn(batch_data=None,word_distribution=None,n=None,label=None):
    input,positive_label=batch_data
    k=positive_label.shape[0]
    batch_word_weights=[word_distribution for i in range(k)]
    label=[positive_label[i] for i in range(k)]
    negative_label=[sampler(w,n,z) for (w,z) in zip(batch_word_weights,label)]
    negative_label=torch.tensor([x.tolist() for x in negative_label])
    positive_label=positive_label.reshape(-1,1)
    idx=torch.concat([positive_label,negative_label],dim=1)
    return input,idx
```

实例化

In [44]:
```
from functools import partial
trainset=Build_CBOW_Dataset(data=data,window_size=9)
collate_fn=partial(collate_fn,word_distribution=trainset.word_distribution,n=19,label=None)
trainloader=nndl.utils.DataLoader(trainset,batch_size=16,collate_fn=collate_fn)
len(trainset),len(trainloader)
```

Out[44]:
```
(20028, 1252)
```

In [45]:
```
for a,b in trainloader:
    print(a.shape,b.shape)
    break
```

```
torch.Size([16, 8]) torch.Size([16, 20])
```

3.2.4 重新搭建模型

In [46]:
```
class SuperW(nn.Module):
    def __init__(self,size,dim=0):
        # dim参数表示取行还是取列
        super(SuperW,self).__init__()
        self.dim=dim
        self.W=nn.Parameter(torch.randn(size=size))
    def __call__(self,idx):
        return self.forward(idx)
    def forward(self,idx):
        if self.dim==0:
            return self.W[idx]
        elif self.dim==1:
            if idx.dim()==1: # 索引是一个列表
                return self.W.T[idx].T
            elif idx.dim()==2: # 索引是多个列表
                return self.W.T[idx].transpose(1,2)
            else:
                raise "dim error"
        else:
            raise "dim error"

class CBOW(nn.Module):
    def __init__(self,num_embeddings,embedding_dim):
        super(CBOW,self).__init__()
        self.num_embeddings=num_embeddings
        self.embedding_dim=embedding_dim
        self.w_in=SuperW(size=[num_embeddings,embedding_dim],dim=0)
        self.w_out=SuperW(size=[embedding_dim,num_embeddings],dim=1)
        self.softmax=nn.Softmax(dim=1)

    def forward(self,x,idx):        # x=(B,L) idx=(B,c)
        x=self.w_in(x)              # (B,L)->(B,L,D)
        x=x.sum(dim=1)/x.shape[1]   # (B,L,D)->(B,1,D)
        new_w_out=self.w_out(idx)   #  new_w_out=(B,D,c)
        x=torch.matmul(x,new_w_out) # (B,1,D)x(B,D,c)->(B,1,c)
        x=x.squeeze(1)              # (B,1,c)->(B,c)
        x=self.softmax(x)           # (B,c)
        return x
```

3.2.5 重构Runner类

In [47]:
```
class Runner():
    def __init__(self, model=None, loss_fn=None, optimizer=None):

        self.model = model
        self.loss_fn = loss_fn
        self.optimizer = optimizer

        self.train_epoch_loss = []
        self.train_step_loss = []


    def train(self, train_loader,  num_epochs=1, log_stride=1):
        step = 0
        total_steps = num_epochs * len(train_loader)

        for epoch in range(num_epochs):
            total_loss = 0
            for x, idx in train_loader:
                self.model.train()
                predicts = self.model(x,idx)    # 这里改动过
                loss = self.loss_fn(predicts)   # 这里改动过
                total_loss += loss.item()
                self.train_step_loss.append((step,loss.item()))
                if step % log_stride == 0 or step == total_steps - 1:
                    print("[Train]  epoch:{}/{} step:{}/{} loss:{:.4f}".format(epoch, num_epochs, step, total_steps,
                                                                               loss.item()))

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

                step += 1
            self.train_epoch_loss.append(total_loss / len(train_loader))


    def plot(self,plot_stride=32):

        plt.figure(figsize=(12, 4))
        plt.clf()
        plt.xlabel("step")
        plt.ylabel("loss")
        train_items=self.train_step_loss[::plot_stride]
```

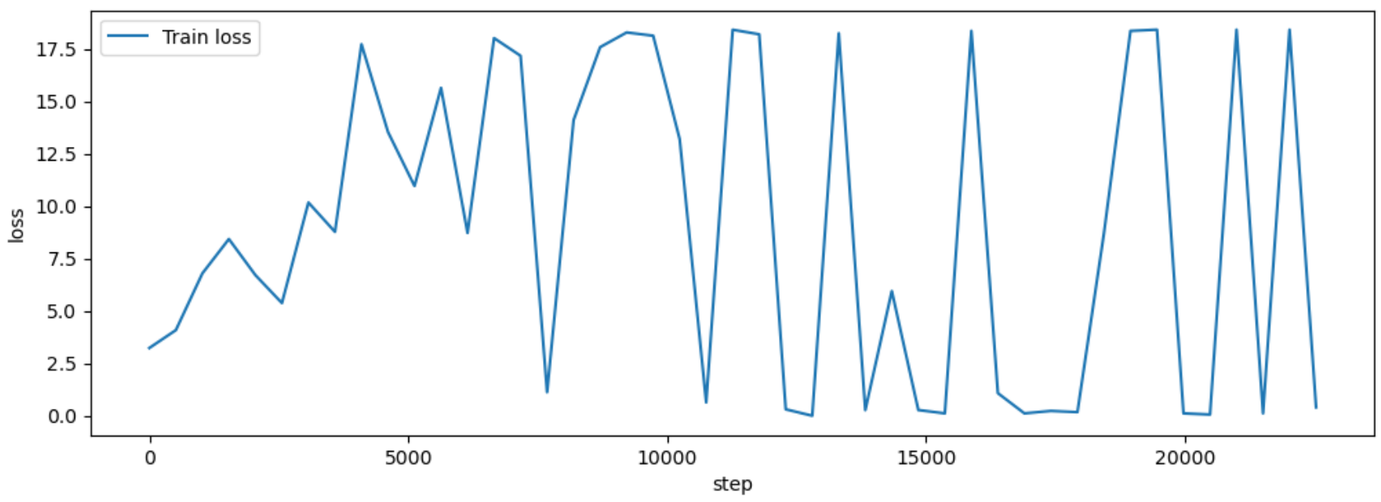
```
        train_x=[x[0] for x in train_items]
        train_loss=[x[1] for x in train_items]
        plt.plot(train_x,train_loss,label="Train loss")
        plt.legend()
        plt.draw()


    def save(self, path):
        torch.save(self.model.state_dict(), path)

    def load(self, path):
        state_dict = torch.load(path)
        self.model.set_state_dict(state_dict)
```

In [48]:
```
model=CBOW(num_embeddings=len(trainset.vocab),embedding_dim=16)
loss_fn=nndl.CrossEntropyLoss_V2()
```

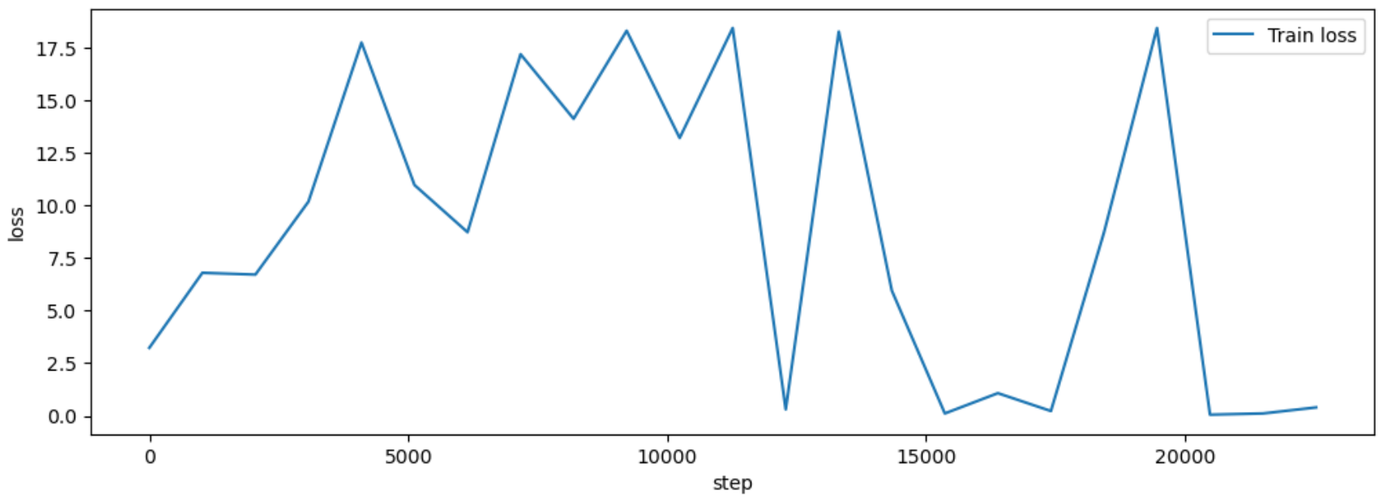
In [49]:
```
opt=torch.optim.Adam(model.parameters(),lr=0.2)
runner=Runner(model=model,loss_fn=loss_fn,optimizer=opt)
runner.train(train_loader=trainloader,num_epochs=18,log_stride=1800)
```

```
[Train]  epoch:0/18 step:0/22536 loss:3.2333
[Train]  epoch:1/18 step:1800/22536 loss:5.3327
[Train]  epoch:2/18 step:3600/22536 loss:7.9600
[Train]  epoch:4/18 step:5400/22536 loss:2.6731
[Train]  epoch:5/18 step:7200/22536 loss:15.4739
[Train]  epoch:7/18 step:9000/22536 loss:4.7157
[Train]  epoch:8/18 step:10800/22536 loss:0.5181
[Train]  epoch:10/18 step:12600/22536 loss:0.1151
[Train]  epoch:11/18 step:14400/22536 loss:2.1807
[Train]  epoch:12/18 step:16200/22536 loss:18.4207
[Train]  epoch:14/18 step:18000/22536 loss:0.4271
[Train]  epoch:15/18 step:19800/22536 loss:0.0576
[Train]  epoch:17/18 step:21600/22536 loss:-0.0000
[Train]  epoch:17/18 step:22535/22536 loss:0.1535
```

In [50]:
```
runner.plot(plot_stride=512)
```



In [51]:
```
runner.plot(plot_stride=1024)
```



3.3 设计word2vec模块

Vocab类

就是word2id的词典

In [52]:
```
class Vocab:
    pass
```

Embedding类

就是CBOW，skip-gram，GloVe等模型的第一层那个W，可以随机初始化获得一个W

In [55]:
```
class Embedding:
    def __init__(self,num_embeddings,embedding_dim):
        self.weight=None
    def forward(self):
        pass
```

TokenEmbedding类

1.继承Embedding类，可以直接训练
2.可以加载CBOW,skip-gram,GloVe等模型训练好的参数给weight
3.需要加载词典进来
4.提供了计算相似度等方法

In [54]:
```
class TokenEmbedding(Embedding):
    pass
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]: