自然语言处理研讨课
（实践课）
第4章 词向量实践

赵 阳
中国科学院自动化研究所
yang.zhao@nlpr.ia.ac.cn

本章内容

1. 词向量理论介绍
2. 实践基础
3. CBOW实践
4. 本章实践

1. 词向量介绍

■文本

- 文本是由文字和标点组成的字符串，短语、句子、段落和篇章都是不同粒度的文本
- 字或字符组成词、短语，进而形成句子、段落和篇章

■文本表示的必要性

- 计算机进行文本理解，必须知道文本长什么样
- 文本的形式化表示是反映文本内容和区分不同文本的有效途径

1. 词向量介绍

文本的one-hot表示

- 文本的One-hot（独热）表示是一种将文本数据（如单词、字符等）转换为数值向量的方法。
- 每个独特的单词或字符都被分配一个唯一的索引，并且这个索引在向量中对应的位置被设置为1，而向量中的其他所有位置都被设置为0。

1. 词向量介绍

- 文本的one-hot表示

该 电影 很 枯燥， 大家 觉得 很 无聊 。

枯燥

$$\begin{bmatrix} 0 \\ 1 \\ 0 \\ \vdots \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

无聊

$$\begin{bmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ 0 \\ 1 \\ 0 \end{bmatrix}$$

1. 词向量介绍

文本的one-hot表示的问题

- **维度灾难**：当词汇表较大时，维度会较高，导致计算效率低下，且大多数位置都是0，非常稀疏。
- **语义鸿沟**：无法体现单词之间的语义关系，因为任何两个单词的One-hot向量之间的欧氏距离都是相同的。

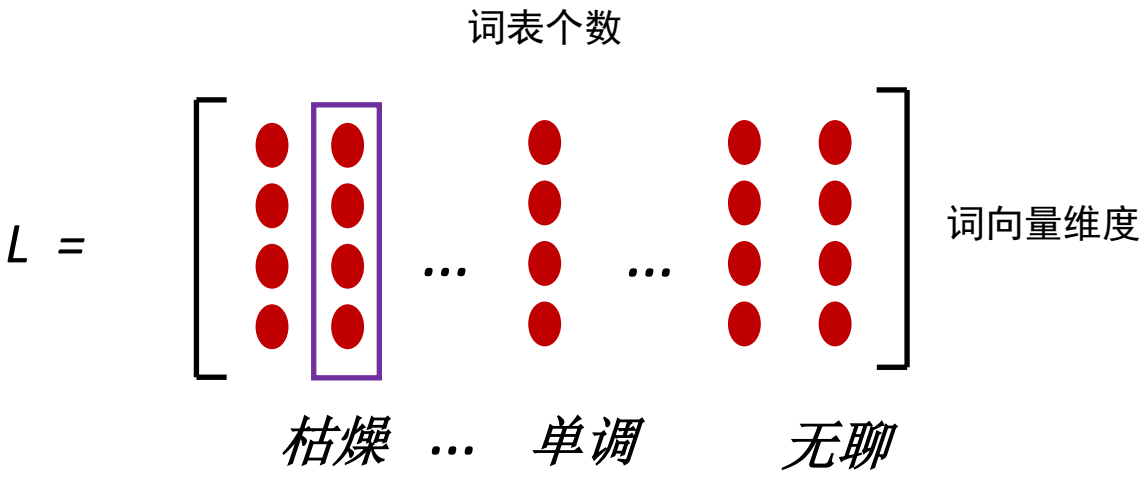
枯燥 ⊗ 无聊

$$\begin{bmatrix}0\\1\\0\\\vdots\\0\\0\\0\end{bmatrix} \times \begin{bmatrix}0\\0\\0\\\vdots\\0\\1\\0\end{bmatrix} = 0$$

任意两个词之间的相似度都为0！

1. 词向量介绍

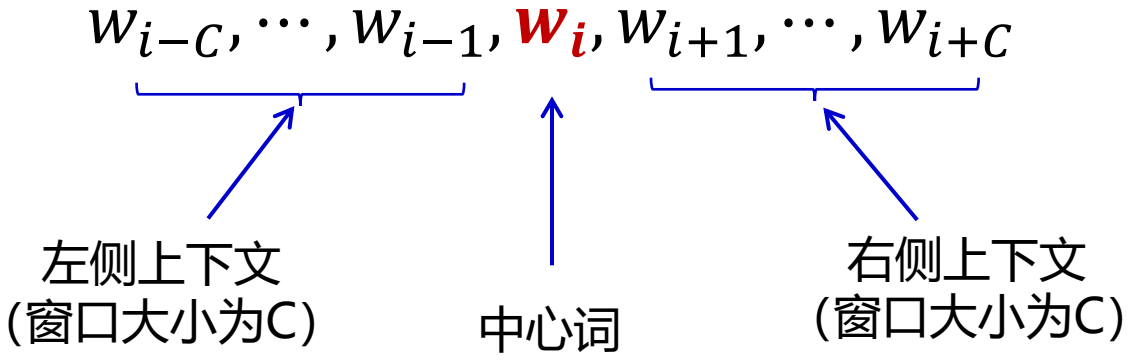
- 文本的低维实值向量表示方法
  - 将文本的单词表示为D维（远远低于词表个数）的稠密向量



1. 词向量介绍

■ CBOW（Continuous Bag-of-Words）

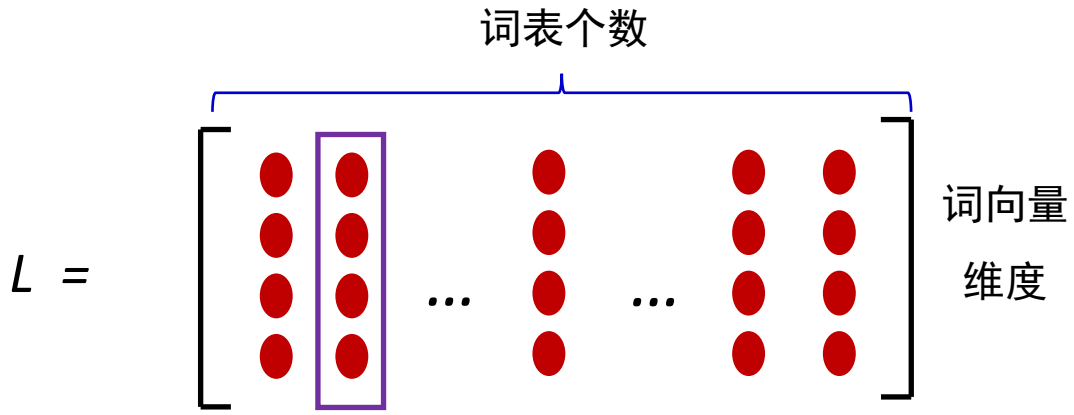
- 词袋（Bag-of-Words）：词序不影响预测
- 利用上下文词语预测中心词语



1. 词向量介绍

■ CBOW（Continuous Bag-of-Words）

Step 1：初始化向量矩阵L

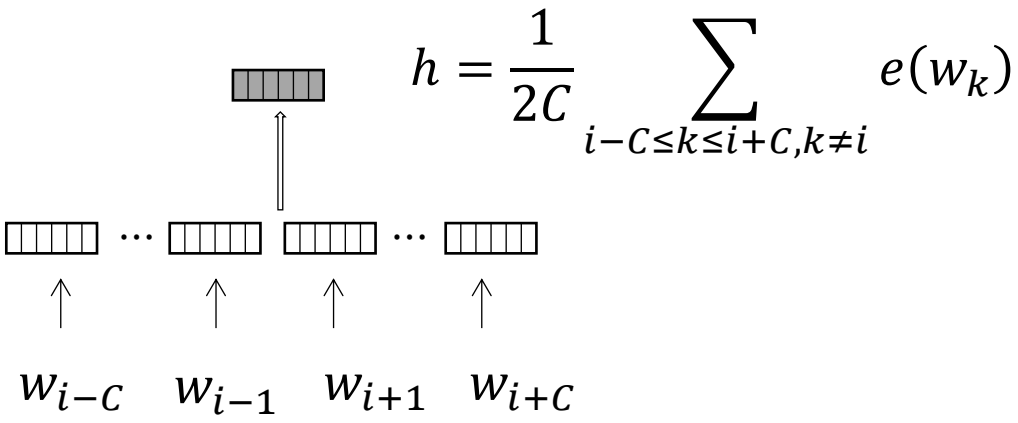


- **词表个数**：选择语料中最高频的N个或者频数超过K个的单词
- **词向量维度**：超参数，自行决定

1. 词向量介绍

- CBOW（Continuous Bag-of-Words）

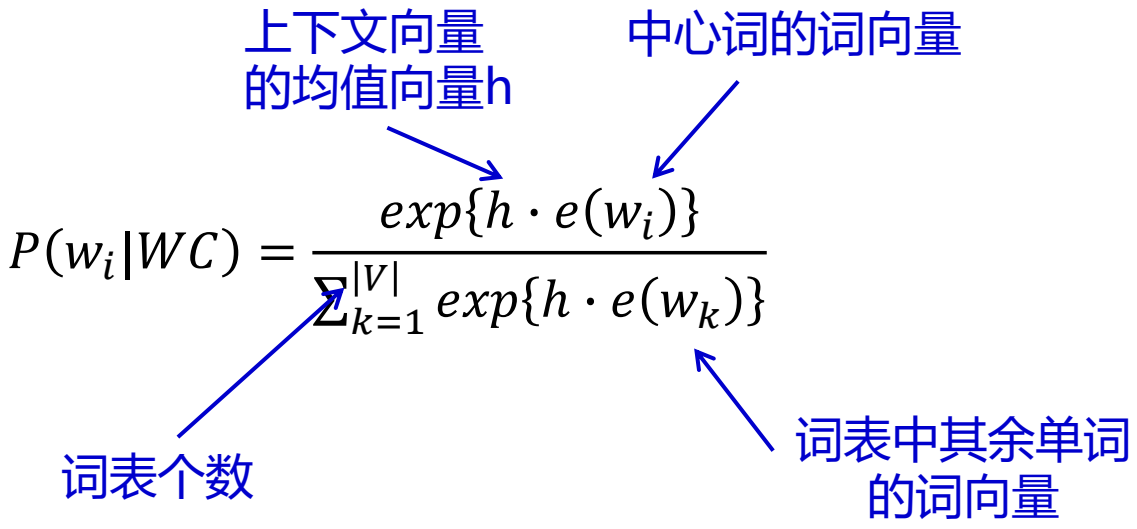
  Step 2：求上下文向量的均值向量h



1. 词向量介绍

- CBOW（Continuous Bag-of-Words）

  Step 3：softmax函数求中心词的概率



1. 词向量介绍

- CBOW（Continuous Bag-of-Words）

  Step 4：目标函数

$$L = \sum_{w_i \text{ in } D} log\, P(w_i|WC)$$

整个训练数据

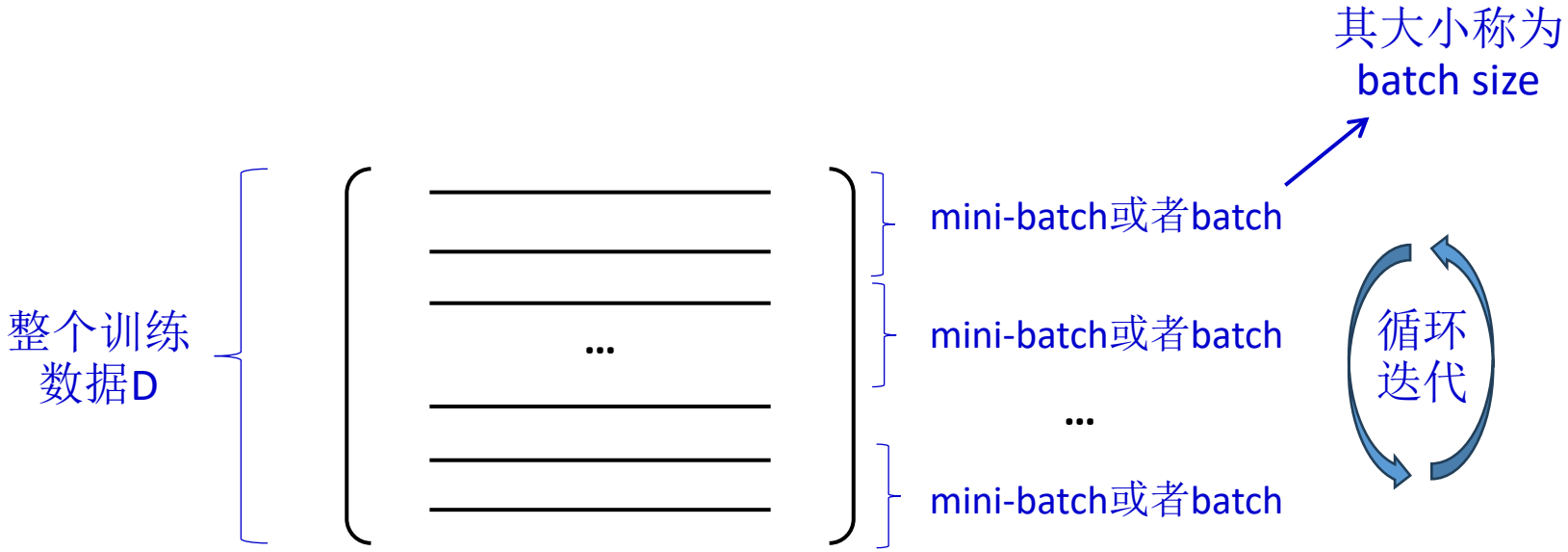
本章内容

1. 词向量理论介绍
2. 实践基础
3. CBOW实践
4. 本章实践

2. 实践基础

■批量（mini-batch）处理

- 将整个训练数据集划分为多个较小的数据子集（mini-batch）
- 每个mini-batch包含一定数量的训练样本。
- 模型使用每个mini-batch的样本进行模型参数的更新，并不断迭代。



2. 实践基础

■ 思考

- 为什么不采用全批量数据进行参数更新？
- 为什么不采用单个数据进行参数更新？

本次实践中需要大家自行构建批数据。

下一次实践中介绍PyTorch提供的批量处理方法。

2. 实践基础

- 负采样技术

$$\boldsymbol{L} = \sum_{\boldsymbol{w_i}\ \text{in}\ \boldsymbol{B}} \boldsymbol{log}\ \boldsymbol{P}(\boldsymbol{w_i}|\boldsymbol{WC}) = \sum_{\boldsymbol{w_i}\ \text{in}\ \boldsymbol{B}} \boldsymbol{log}\ \frac{exp\{h \cdot e(w_i)\}}{\sum_{k=1}^{|V|} exp\{h \cdot e(w_k)\}}$$

词表个数

当词表个数|V|过大时，计算复杂度太高！

2. 实践基础

■ 负采样技术

$$\boldsymbol{L} = \sum_{\boldsymbol{w_i} \text{ in } \boldsymbol{B}} \boldsymbol{log}\, \boldsymbol{P}(\boldsymbol{w_i}|\boldsymbol{WC}) = \sum_{\boldsymbol{w_i} \text{ in } \boldsymbol{B}} \boldsymbol{log} \frac{exp\{h \cdot e(w_i)\}}{\sum_{k=1}^{|V|} exp\{h \cdot e(w_k)\}}$$

词表个数

仅采样K个负样本，代表整个词表

负样本个数

$$\boldsymbol{L} = \sum_{\boldsymbol{w_i} \text{ in } \boldsymbol{B}} \boldsymbol{log}\, \boldsymbol{P}(\boldsymbol{w_i}|\boldsymbol{WC}) = \sum_{\boldsymbol{w_i} \text{ in } \boldsymbol{B}} \boldsymbol{log} \frac{exp\{h \cdot e(w_i)\}}{\sum_{k=1}^{\boldsymbol{K}} exp\{h \cdot e(w_k)\}}$$

$$= \sum_{\boldsymbol{w_i} \text{ in } \boldsymbol{B}} \{\, \boldsymbol{log}\, exp\{h \cdot e(w_i)\} - \boldsymbol{log} \sum_{k=1}^{\boldsymbol{K}} exp\{h \cdot e(w_k)\} \,\}$$

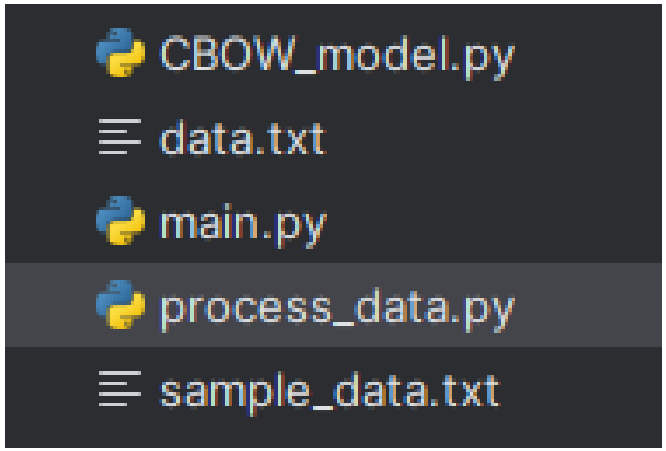
本章内容

1. 词向量理论介绍
2. 实践基础
3. CBOW实践
4. 本章实践

1. 词向量介绍

■ 基本实现思路

- main.py
  #主函数，实现程序逻辑控制
  word2vec类
- process_data.py
  # 数据处理，实现数据处理
  # 构建正负样本
- CBOW_model.py
  # 实现CBOW模型
  初始化词向量
  计算损失



1. 词向量介绍

- 主函数（main.py）基本实现思路

超参数

word2vec类

实例化类

训练

```
from CBOW_model import CBOWModel
from process_data import ProcessData
import torch.optim as optim
from tqdm import tqdm
BATCH_SIZE = 8  # mini-batch
WINDOW_SIZE = 4 # 上下文窗口
MIN_COUNT = 0  # 可有可无
EMB_DIMENSION = 100  # embedding维度
LR = 0.02  # 学习率
NEG_COUNT = 4  # 负采样数

class Word2Vec:
    def __init__(self, input_file_name, output_file_name):

    def train(self):

if __name__ == '__main__':
    word2vec = Word2Vec(input_file_name='./sample_data.txt',
output_file_name="word_embedding.txt")
    word2vec.train()
```

1. 词向量介绍

- Word2vec类的实现思路
  - __init__()函数

处理数据

实例化模型

```
class Word2Vec:
    def __init__(self, input_file_name, output_file_name):
        self.output_file_name = output_file_name
        self.data = ProcessData(input_file_name, MIN_COUNT)
        self.model = CBOWModel(self.data.word_count, EMB_DIMENSION)
        self.lr = LR
        self.optimizer = optim.SGD(self.model.parameters(), lr=self.lr)
```

1. 词向量介绍

■ Word2vec类的实现思路

- train() 函数

显示数据
完成进度

正样本
pos_u：上下文样本
pos_w：中心词

负样本 →

利用CBOW预测损失 →

随着训练进行，
降低学习率

保存词向量 →

```
def train(self):
    pairs_count = self.data.evaluate_pairs_count(WINDOW_SIZE)
    batch_count = pairs_count / BATCH_SIZE
    process_bar = tqdm(range(int(batch_count)))
    for i in process_bar:
        pos_pairs = self.data.get_batch_pairs(BATCH_SIZE, WINDOW_SIZE)
        pos_u = [pair[0] for pair in pos_pairs]
        pos_w = [int(pair[1]) for pair in pos_pairs]
        neg_w = self.data.get_negative_sampling(pos_pairs, NEG_COUNT)
        self.optimizer.zero_grad()
        loss = self.model.forward(pos_u, pos_w, neg_w)
        loss.backward()
        self.optimizer.step()
        if i * BATCH_SIZE % 100000 == 0:
            self.lr = self.lr * (1.0 - 1.0 * i / batch_count)
            for param_group in self.optimizer.param_groups:
                param_group['lr'] = self.lr
    self.model.save_embedding(self.data.id2word_dict, self.output_file_name)
```

1. 词向量介绍

- Word2vec类的实现思路
  - 正样本和负样本示例

上下文的ID

正确单词的ID

- pos_pair

[([0, 1, 3, 4, 5], 2), ([0, 1, 2, 4, 5, 6], 3), ([1, 2, 3, 5, 6, 5], 4), ([3, 4, 5, 5, 6, 7], 5), ([5, 6, 5, 7, 8], 6), ([5, 6, 7], 5), ([7, 8], 6), ([1, 2, 3], 0)]

- neg_w

[[545, 230, 305, 436], [185, 184, 327, 618], [499, 246, 398, 111], [303, 598, 143, 548], [572, 465, 107, 195], [580, 326, 546, 190], [390, 155, 49, 150], [205, 618, 523, 217]]

负样本的ID

1. 词向量介绍

- Process_data.py

三个字典

双向队列

```
class ProcessData:
    def __init__(self, input_file_name, min_count):
        self.input_file_name = input_file_name
        self.input_file = open(self.input_file_name,encoding="utf-8") # 数据文件
        self.min_count = min_count # 要淘汰的低频数据的频度
        self.word_count = 0 # 单词数（重复的词只算1个）
        self.word_count_sum = 0 # 单词总数
        self.sentence_count = 0 # 句子数
        self.id2word_dict = dict() # 词id-词 dict
        self.word2id_dict = dict() # 词-词id dict
        self.wordId_frequency_dict = dict() # 词id-出现次数 dict
        self._init_dict() # 初始化字典
        self.word_pairs_queue = deque()
    def _init_dict(self):

    def get_batch_pairs(self, batch_size, window_size):

    def get_negative_sampling(self, positive_pairs, neg_count):

    def evaluate_pairs_count(self, window_size):
```

1. 词向量介绍

- Process_data.py

根据训练数据，构建

id2word_dict

word2id_dict

wordId_frequency_dict

```
def _init_dict(self):
    word_freq = dict()
    # 统计 word_frequency
    for line in self.input_file:
        line = line.strip().split(' ')  # 去首尾空格
            self.word_count_sum += len(line)
        self.sentence_count += 1
        for word in line:
            try:
                word_freq[word] += 1
            except:
                word_freq[word] = 1
    word_id = 0
    # 初始化 word2id_dict,id2word_dict, wordId_frequency_dict字典
       for per_word, per_count in word_freq.items():
        if per_count < self.min_count:  # 去除低频
                    self.word_count_sum -= per_count
            continue
        self.id2word_dict[word_id] = per_word
        self.word2id_dict[per_word] = word_id
        self.wordId_frequency_dict[word_id] = per_count
        word_id += 1
    self.word_count = len(self.word2id_dict)
```

1. 词向量介绍

■ Process_data.py

大家自行完成

```
# 获取mini-batch大小的正采样对 (Xw,w)， Xw为上下文id数组，w为目标词id。
窗口大小为window_size，即2c = 2*window_size
def get_batch_pairs(self, batch_size, window_size):
...
return result_pairs
```

■ 提示1：输出格式

上下文的ID

正确单词的ID

- pos_pair

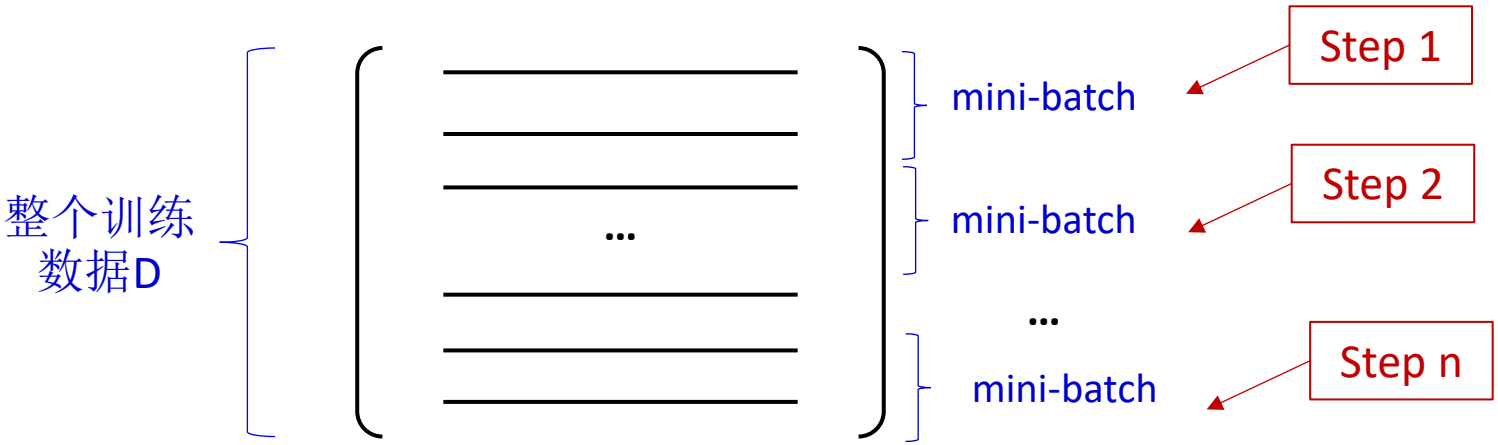
[([0, 1, 3, 4, 5], 2), ([0, 1, 2, 4, 5, 6], 3), ([1, 2, 3, 5, 6, 5], 4), ([3, 4, 5, 5, 6, 7], 5), ([5, 6, 5, 7, 8], 6), ([5, 6, 7], 5), ([7, 8], 6), ([1, 2, 3], 0)]

1. 词向量介绍

大家自行完成

```
# 获取mini-batch大小的正采样对 (Xw,w)， Xw为上下文id数组，w为目标词id。
窗口大小为window_size，即2c = 2*window_size
def get_batch_pairs(self, batch_size, window_size):
…
return result_pairs
```

■ 提示2：双端队列(deque)



- 位于collections模块中
- 允许从两端快速地添加（append）和弹出（pop）元素。

1. 词向量介绍

- Process_data.py

最简单的负采样方法

```
def get_negative_sampling(self, positive_pairs, neg_count):
    neg_u = np.random.choice(self.word_count, size=(len(positive_pairs), neg_count)).tolist()
    return neg_u

# 估计数据中正采样对数
def evaluate_pairs_count(self, window_size):
```

大家自行完成

1. 词向量介绍

■ CBOW_model.py的实现思路

- __init__() 函数

```
import torch
import torch.nn as nn
import torch.nn.functional as F
import numpy as np
class CBOWModel(nn.Module):
    def __init__(self, emb_size, emb_dimension):
        super(CBOWModel, self).__init__()
        self.emb_size = emb_size
        self.emb_dimension = emb_dimension
        self.u_embeddings = nn.Embedding(self.emb_size, self.emb_dimension)
        self.w_embeddings = nn.Embedding(self.emb_size, self.emb_dimension)
        self._init_embedding()  # 初始化

    def _init_embedding(self):
        self.u_embeddings.weight.data.uniform_(0, 1)
        self.w_embeddings.weight.data.uniform_(0, 1)
```

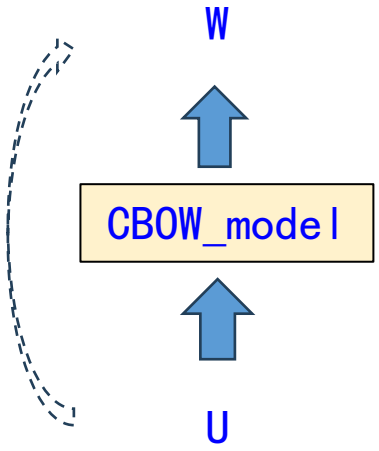
- 上下文的词向量U
- 预测词的向量W

（0，1）均匀分布

1. 词向量介绍

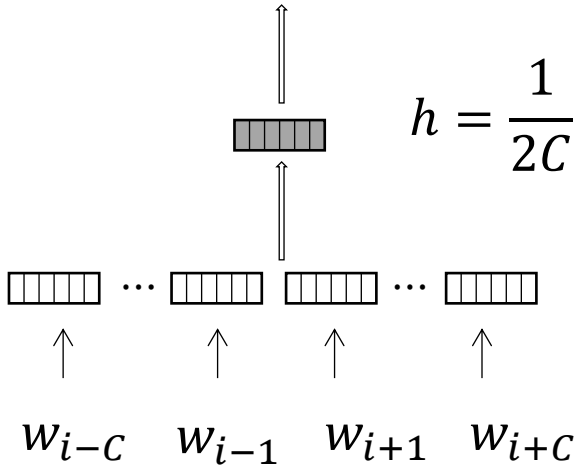
- CBOW_model.py的实现思路
  - __init__() 函数

可以共享词向量
也可以不共享



$$P(w_i|WC) = \frac{exp\{h \cdot e(w_i)\}}{\sum_{k=1}^{|V|} exp\{h \cdot e(w_k)\}}$$

$$h = \frac{1}{2C} \sum_{i-C \le k \le i+C, k \ne i} e(w_k)$$

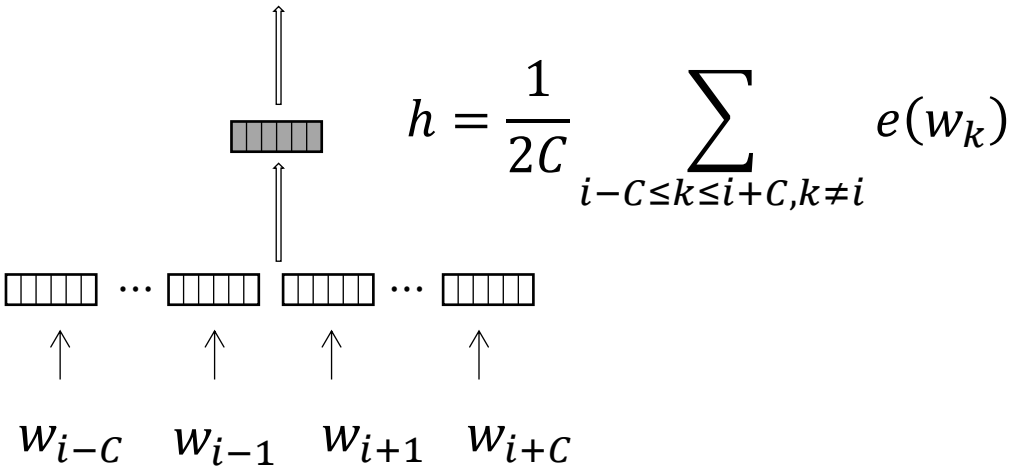


1. 词向量介绍

■ CBOW_Model.py的实现思路

- forward （）函数

$$P(w_i|WC) = \frac{exp\{h \cdot e(w_i)\}}{\sum_{k=1}^{|V|} exp\{h \cdot e(w_k)\}}$$



1. 词向量介绍

上下文的词向量并求和

根据索引，取词向量embeddings

正样本打分

负样本打分

```
def forward(self, pos_u, pos_w, neg_w):
    pos_u_emb = []  # 上下文embedding
    for per_Xw in pos_u:
        per_u_emb = self.u_embeddings(torch.LongTensor(per_Xw))  # 上下文
embedding
        per_u_numpy = per_u_emb.data.numpy()
        per_u_numpy = np.sum(per_u_numpy, axis=0)
        per_u_list = per_u_numpy.tolist
        pos_u_emb.append(per_u_list)  # 放回数组
    pos_u_emb = torch.FloatTensor(pos_u_emb)  # [batch_size * emb_dimension ]
    pos_w_emb = self.w_embeddings(torch.LongTensor(pos_w))  # [batch_size *
emb_dimension ]
    neg_w_emb = self.w_embeddings(torch.LongTensor(neg_w))  #
[batch_size*negative_number * emb_dimension ]
    score_1 = torch.mul(pos_u_emb, pos_w_emb).squeeze()
    score_2 = torch.sum(score_1, dim=1)
    score_3 = F.logsigmoid(score_2)  #
    neg_score_1 = torch.bmm(neg_w_emb, pos_u_emb.unsqueeze(2)).squeeze()
    neg_score_2 = torch.sum(neg_score_1, dim=1)  #
    neg_score_3 = F.logsigmoid((-1) * neg_score_2)  #
    loss = torch.sum(score_3) + torch.sum(neg_score_3)
    return -1 * loss
```

1. 词向量介绍

■ PyTorch中的维度变化

- 张量的维度变化是深度学习中常见的操作，要求我们重新组织张量以满足模型或函数的需求
- PyTorch提供了多种方法改变张量（tensor）的维度，例如：reshape、view、squeeze、unsqueeze、permute、expand等
- 这里用到了unsqueeze

```
neg_score_1 = torch.bmm(neg_w_emb, pos_u_emb.unsqueeze(2)).squeeze()
```

1. 词向量介绍

■ PyTorch中的维度变化

```
neg_score_1 = torch.bmm(neg_w_emb, pos_u_emb.unsqueeze(2)).squeeze()
```

- neg_w_emb维度 [batch_size*negative_number * emb_dimension]
- pos_u_emb维度 [batch_size * emb_dimension]
- pos_u_emb.unsqueeze(2) 维度[batch_size * emb_dimension* 1] #指定的位置插入一个新的维度，其大小为1
- torch.bmm执行批量矩阵乘法（batch matrix multiplication）
  - 输入：两个三维张量，形状满足（b，n，m）和（b，m，p），其中 b 是批次大小，n、m 和 p 是矩阵的维度。
  - 输出：新的三维张量，形状为（b，n，p）
- torch.bmm(neg_w_emb, pos_u_emb.unsqueeze(2))维度 [batch_size*negative_number * 1]
- torch.bmm(neg_w_emb, pos_u_emb.unsqueeze(2)).squeeze()的维度[batch_size*negative_number]

本章内容

1. 词向量理论介绍
2. 实践基础
3. CBOW实践
4. 本章实践

本节实践

文本表示实践

- 利用上节课抽取的文本数据，实现至少一种词向量方法（CBOW和skip-gram等）
- 分别在中文和英文上训练词向量。
- 给定20个单词，利用词向量找到与之最相似的top 10个单词

注意：

可以按照PPT上的思路实现，也可以不按照思路实现。

事实上有更简写、更高效的写法。

本节实践

■ **要求和评分准则：**

- 报告需要详细说明爬取和处理的基本方法、数据来源、算法流程、分析结果、结论等；
- 不需要把程序大段复制上去，如果有需要，只需要截取和复制关键程序即可；报告页数不需要太长。
- 如果满足了基本要求就可以得到B（80-85分）
- 剩余的15分， 看大家的自己拓展分析和实验

不同主题下词向量的差别

不同负采样的个数和选择方法是否有差别

不同词向量方法的差别

…

本节实践

本次作业需要与上节课实践一同提交，涉及数据爬取、处理和表示

提交时间：10月28号之前

提交内容：实践报告、代码和数据

谢谢！

Thanks!