第5章 文本分类实践

In [1]:
```
import torch
import torch.nn as nn
import re
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from collections import Counter
import sys
sys.path.append(r"C:\Users\Administrator\Desktop\Open1507Lab")
import nndl
from torch.nn.utils.rnn import pack_sequence,pack_padded_sequence,pad_sequence,pad_packed_sequence
```

In [2]:
```
%pprint
```

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

In [3]:
```
df=pd.read_csv("../data/dataset/imdb/labeledTrainData.tsv",sep="\t")
df.head()
```

Out[3]:

	id	sentiment	review
0	5814_8	1	With all this stuff going down at the moment w...
1	2381_9	1	\The Classic War of the Worlds\" by Timothy Hi...
2	7759_3	0	The film starts with a manager (Nicholas Bell)...
3	3630_4	0	It must be assumed that those who praised this...
4	9495_8	1	Superbly trashy and wondrously unpretentious 8...

数据处理及计算流程

① 预处理层

【目标】：
1.获得形如[[],[],[],...]这样的list of list，里面的每个list都是一个句子的数字表示，比如[我，叫，张，三]=[12,14,17,44]
2.获得[1,0,1,1,0,1,0,...]这样的标签列表
【处理流程】：
第一步：先把文件读取，把review和sentiment两列数据抽出来放到两个列表中，获得list of int 和 list of str，直接达成目标2
第二步：把list of str里面的每个str句子分词，获得list of list，里面的list是单词组成的list，还不是数字
第三步：制作一个单词到数字映射的词典，就是把上面list of str里面所有出现的单词统计出来，按词频由高到低把单词映射成自然数
第四步：根据第三步制作的词典，将list of str中的str映射为数字的list，从而获得list of list

In [4]:
```
class IMDB_Preprocessing:
    def __init__(self,path=None):
        df=self.load(path)     # dataframe
        seqs,labels=self.get_data(df)  # 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)
        self.labels=torch.from_numpy(labels.values)

    def load(self,path):
        return pd.read_csv(path,sep="\t")
    def get_data(self,df):
        return [str(seq) for seq in df.review],df.sentiment
    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 [5]:
```
# test
path="../data/dataset/imdb/labeledTrainData.tsv"
data=IMDB_Preprocessing(path=path)
len(data.seqs),len(data.labels)
```

Out[5]:
```
(25000, 25000)
```

In [6]:
```
data.seqs[0],data.labels[0]
```

Out[6]:
```
([18, 32, 12, 530, 170, 181, 33, 2, 547, 18, 8965, 11, 139, 636, 2601, 6, 28, 224, 149, 2, 1010, 645, 128, 3, 41, 294, 2, 18843, 3, 294, 11399, 174, 279, 11, 44, 182, 6, 76, 4, 795, 2602, 84, 12, 226, 37, 11, 196, 15, 66, 628,
10, 2, 4221, 44, 6, 279, 96, 57, 61, 328, 712, 26, 7, 2477, 43, 1332, 11399, 7, 173, 4973, 173, 769, 21, 63, 11, 372, 170, 6, 67, 33, 2, 424, 55, 9, 15, 1804, 616, 50, 5, 9, 48, 1280, 3410, 45, 8965, 14, 538, 930, 2, 3487, 3, 8
3, 2, 568, 731, 5, 1641, 27, 77, 142, 4570, 8, 8, 1994, 1135, 20, 5, 260, 12, 7, 32, 45, 467, 1581, 38, 879, 23, 2566, 40, 8965, 10, 545, 94, 23, 27, 170, 6, 771, 12, 3, 169, 9, 354, 50, 200, 670, 8965, 36, 26221, 17, 26222, 6,
2, 229, 5, 12, 19, 20, 8965, 3, 91, 5, 28, 447, 62, 134, 13, 26, 92, 9, 17, 2, 447, 63, 47, 281, 7, 66, 325, 5, 89, 8, 8, 2, 767, 769, 21, 225, 55, 9, 414, 507, 7, 65, 24, 17, 230, 43, 38, 15689, 2, 3461, 1636, 705, 3, 868, 101
66, 7, 1061, 16, 4, 11732, 32, 958, 1377, 1548, 136, 26, 485, 8965, 339, 38, 77, 7, 707, 72, 87, 8965, 22385, 28, 2416, 13583, 868, 10166, 14, 105, 28772, 13, 26, 466, 79, 6, 123, 9, 7, 26, 37, 7, 18844, 1641, 513, 38, 11, 1140
0, 279, 26, 44, 4108, 8965, 14, 224, 8, 8, 762, 5, 628, 183, 10, 12, 40, 8965, 1569, 84, 4, 505, 3, 4, 2305, 3, 2, 223, 2062, 2645, 705, 83, 2, 158, 208, 29, 69, 2, 5022, 5, 4, 5246, 55, 9, 380, 6, 1406, 2, 26223, 77, 705, 16,
622, 880, 771, 766, 18, 31, 519, 278, 573, 4, 223, 747, 5, 97, 3411, 4, 1298, 818, 135, 8, 8, 1309, 342, 12, 19, 7, 17, 79, 37, 40, 8965, 24, 31, 637, 43, 161, 63, 11, 103, 7, 91, 79, 47, 25, 94, 775, 244, 9, 127, 349, 3, 202,
124, 4, 7575, 731, 3, 3584, 8965, 14, 37405, 2006, 10, 12, 19, 7, 4, 238, 467, 1581, 7, 366, 31, 5, 2, 91, 1001, 79, 125, 6, 1628, 12, 1134, 20, 7, 26, 2477, 73, 18, 32, 2, 678, 11, 139, 511, 12, 852, 7070, 73, 11, 90, 22, 123,
87, 79, 52, 30, 272, 487, 4540, 3532, 11, 123, 12, 17, 4, 191, 26, 7, 344, 36, 565, 325, 20, 373, 226, 43, 31, 5, 2, 91, 18845, 18846, 11, 440, 26, 7, 25, 2, 1496], tensor(1))
```

② Dataset和DataLoader层

Dataset层的目标

就是把上面预处理得到的句子（list of list）和标签（list）封装到Dataset类当中

In [7]:
```
class IMDB_Trainset(nndl.utils.Dataset):
    def __init__(self,data):
        super(IMDB_Trainset,self).__init__()
        self.x=data.seqs
        self.labels=data.labels

    def __getitem__(self,idx):
        return self.x[idx],self.labels[idx]
    def __len__(self):
        assert len(self.x)==len(self.labels)
        return len(self.labels)
```

In [8]:
```
trainset=IMDB_Trainset(data=data)
```

In [9]:
```
first_x,first_label=trainset[0]
first_x,first_label
```

Out[9]:
```
([18, 32, 12, 530, 170, 181, 33, 2, 547, 18, 8965, 11, 139, 636, 2601, 6, 28, 224, 149, 2, 1010, 645, 128, 3, 41, 294, 2, 18843, 3, 294, 11399, 174, 279, 11, 44, 182, 6, 76, 4, 795, 2602, 84, 12, 226, 37, 11, 196, 15, 66, 628,
10, 2, 4221, 44, 6, 279, 96, 57, 61, 328, 712, 26, 7, 2477, 43, 1332, 11399, 7, 173, 4973, 173, 769, 21, 63, 11, 372, 170, 6, 67, 33, 2, 424, 55, 9, 15, 1804, 616, 50, 5, 9, 48, 1280, 3410, 45, 8965, 14, 538, 930, 2, 3487, 3, 8
3, 2, 568, 731, 5, 1641, 27, 77, 142, 4570, 8, 8, 1994, 1135, 20, 5, 260, 12, 7, 32, 45, 467, 1581, 38, 879, 23, 2566, 40, 8965, 10, 545, 94, 23, 27, 170, 6, 771, 12, 3, 169, 9, 354, 50, 200, 670, 8965, 36, 26221, 17, 26222, 6,
2, 229, 5, 12, 19, 20, 8965, 3, 91, 5, 28, 447, 62, 134, 13, 26, 92, 9, 17, 2, 447, 63, 47, 281, 7, 66, 325, 5, 89, 8, 8, 2, 767, 769, 21, 225, 55, 9, 414, 507, 7, 65, 24, 17, 230, 43, 38, 15689, 2, 3461, 1636, 705, 3, 868, 101
66, 7, 1061, 16, 4, 11732, 32, 958, 1377, 1548, 136, 26, 485, 8965, 339, 38, 77, 7, 707, 72, 87, 8965, 22385, 28, 2416, 13583, 868, 10166, 14, 105, 28772, 13, 26, 466, 79, 6, 123, 9, 7, 26, 37, 7, 18844, 1641, 513, 38, 11, 1140
0, 279, 26, 44, 4108, 8965, 14, 224, 8, 8, 762, 5, 628, 183, 10, 12, 40, 8965, 1569, 84, 4, 505, 3, 4, 2305, 3, 2, 223, 2062, 2645, 705, 83, 2, 158, 208, 29, 69, 2, 5022, 5, 4, 5246, 55, 9, 380, 6, 1406, 2, 26223, 77, 705, 16,
622, 880, 771, 766, 18, 31, 519, 278, 573, 4, 223, 747, 5, 97, 3411, 4, 1298, 818, 135, 8, 8, 1309, 342, 12, 19, 7, 17, 79, 37, 40, 8965, 24, 31, 637, 43, 161, 63, 11, 103, 7, 91, 79, 47, 25, 94, 775, 244, 9, 127, 349, 3, 202,
124, 4, 7575, 731, 3, 3584, 8965, 14, 37405, 2006, 10, 12, 19, 7, 4, 238, 467, 1581, 7, 366, 31, 5, 2, 91, 1001, 79, 125, 6, 1628, 12, 1134, 20, 7, 26, 2477, 73, 18, 32, 2, 678, 11, 139, 511, 12, 852, 7070, 73, 11, 90, 22, 123,
87, 79, 52, 30, 272, 487, 4540, 3532, 11, 123, 12, 17, 4, 191, 26, 7, 344, 36, 565, 325, 20, 373, 226, 43, 31, 5, 2, 91, 18845, 18846, 11, 440, 26, 7, 25, 2, 1496], tensor(1))
```

上面获取一个样本是（list，tensor），到了dataloader中，通过fetch获得batch_data=[（list，tensor），（list，tensor），（list，tensor）]这种形式，
这种形式的数据需要通过collate_fn函数进行处理，这就是后面要做的内容

DataLoader的目标及处理流程

DataLoader从上述Dataset中一次性取出batch_size个句子，也就是一次性取出多个列表，并放在一个列表中，由于后面放到模型中
并行计算时，需要句子长度保持一致，所以我们需要对取出来的句子统一长度，就按这批句子里面最长的那个统一，不够长度的补充pad字符
另外一点就是，有的句子过长，我们这里设置一个最大句子长度，超过这个长度的统统截断成这个长度
【目标】：返回的一批句子列表时等长的，这批句子是list of list的形式，最终返回的是一个二维列表，每一行就是一个句子，每一个数字就是一个单词
另外，句子在补充pad之前的长度需要记录到一个列表中，一块返回，后面需要用到。labels直接从Dataset中取出返回，不需要处理
【处理流程】：
第一步：获得从Dataset传过来的一个batch_size的数据，就是句子的list of list，和标签的list，这步在DataLoader内部获取
第二步：把数据丢进collate_fn中处理成等长的，然后返回等长句子，句子真实长度，句子标签三个东西，我们只要定义好collate_fn函数，交给DataLoader就行

方法1

In [10]:
```
def collate_fn1(batch_data,pad=0,max_seq_len=256):
    seqs,seq_lens,labels=[],[],[] # 先把输出的坑位留好
    max_len=0 #处理这批数据之前，最大序列长度置0

    # 截断并保存截断后的真实长度
    for seq,label in batch_data:
        labels.append(label) # 标签不需要处理
        # 句子截断，记录长度
        seq=seq[:max_seq_len]
        seqs.append(seq)
        seq_lens.append(len(seq))
        # 更新这批句子的最大长度
        max_len=max(max_len,len(seq))

    # 等长填充
    for i in range(len(seqs)):
        seqs[i]=seqs[i]+[pad]*(max_len-len(seqs[i]))

    # 返回处理好的数据，返回格式 （seqs,seq_lens），labels 因为需要丢给下层也就是embedding层使用，所以转换为tensor类型
    return (torch.tensor(seqs),torch.tensor(seq_lens,dtype=torch.int64)),torch.tensor(labels,dtype=torch.int64)
```

实例化并测试，为了打印输出不占地方，这里batch_size取4，后面训练的时候再改大一点

In [11]:
```
train_loader1=nndl.utils.DataLoader(trainset,batch_size=4,collate_fn=collate_fn1,shuffle=False)
```

In [12]:
```
for batch_data in train_loader1:
    print(batch_data)
    break
```

```
((tensor([[   18,    32,    12,  ..., 11400,   279,    26],
       [    2,   350,   321,  ...,     0,     0,     0],
       [    2,    21,   507,  ...,     9,  6367,     4],
       [    9,   208,    30,  ...,    70,   510,    29]]), tensor([256, 161, 256, 256])), tensor([1, 1, 0, 0]))
```

方法2

In [13]:
```
def collate_fn2(batch_data,max_seq_len=256):
    seq_lens=[]
    seqs,labels=zip(*batch_data)
    temp=[]
    # 截断
    for seq in seqs:
        seq=seq[:max_seq_len]
        temp.append(torch.tensor(seq))
        seq_lens.append(len(seq))
    # padding
    x=pad_sequence(temp,batch_first=True)
    return (x,seq_lens),torch.tensor(labels,dtype=torch.int64)
```

In [14]:
```
train_loader2=nndl.utils.DataLoader(trainset,batch_size=4,collate_fn=collate_fn2,shuffle=False)
```

In [15]:
```
for batch_data in train_loader2:
    print(batch_data)
    break
```

```
((tensor([[   18,    32,    12,  ..., 11400,   279,    26],
       [    2,   350,   321,  ...,     0,     0,     0],
       [    2,    21,   507,  ...,     9,  6367,     4],
       [    9,   208,    30,  ...,    70,   510,    29]]), [256, 161, 256, 256]), tensor([1, 1, 0, 0]))
```

但是这样padding后的序列，不是按照序列由长到短排序的，压缩后无法交给LSTM处理

In [16]:
```
def collate_fn3(batch_data,max_seq_len=256):
    # 排序
    batch_sorted=sorted(batch_data,key=lambda x:len(x[0]),reverse=True)
    seqs=[p[0] for p in batch_sorted]
    labels=[p[1] for p in batch_sorted]

    # 截断
    temp=[]
    seq_lens=[]
    for seq in seqs:
        seq=seq[:max_seq_len]
        temp.append(torch.tensor(seq))
        seq_lens.append(len(seq))
    # padding
    x=pad_sequence(temp,batch_first=True)
    return (x,seq_lens),torch.tensor(labels,dtype=torch.int64)
```

In [17]:
```
train_loader3=nndl.utils.DataLoader(trainset,batch_size=4,collate_fn=collate_fn3,shuffle=False)
```

In [18]:
```
next(iter(train_loader3))
```

Out[18]:
```
((tensor([[   18,    32,    12,  ..., 11400,   279,    26],
       [    9,   208,    30,  ...,    70,   510,    29],
       [    2,    21,   507,  ...,     9,  6367,     4],
       [    2,   350,   321,  ...,     0,     0,     0]]), [256, 256, 256, 161]), tensor([1, 0, 0, 1]))
```

In [19]:
```
for batch_data in train_loader3:
    print(batch_data)
    break
```

```
((tensor([[   18,    32,    12,  ..., 11400,   279,    26],
       [    9,   208,    30,  ...,    70,   510,    29],
       [    2,    21,   507,  ...,     9,  6367,     4],
       [    2,   350,   321,  ...,     0,     0,     0]]), [256, 256, 256, 161]), tensor([1, 0, 0, 1]))
```

③ Embedding层

Embedding层的目标

把dataloader层丢过来的一批句子BxL编码成BxLxd

In [20]:
```
encoding=nn.Embedding(num_embeddings=len(data.vocab),embedding_dim=30,padding_idx=0)
```

In [21]:
```
(batch_seqs,seq_length),labels=next(iter(train_loader3))
input=encoding(batch_seqs)
input.shape
```

Out[21]:
```
torch.Size([4, 256, 30])
```

④ BiLSTM层

BiLSTM层的目标

把embedding层丢过来的一批句子BxLxd先进行pack，形状为：（B_1+B_2+B_3+B_4）xd，
再经过bilstm层处理为len(seq_1+seq_2+seq_3+seq_4)x2D，最后pad回BxLx2D

In [22]:
```
bilstm=nn.LSTM(input_size=30,hidden_size=8,bidirectional=True,batch_first=True)
```

In [23]:
```
h0=torch.zeros(2,4,8)
c0=torch.zeros(2,4,8)
packed_input=pack_padded_sequence(input,lengths=seq_length,batch_first=True)
```

In [24]:
```
out,(h_o,c_o)=bilstm(packed_input,(h0,c0))
out
```

Out[24]:
```
PackedSequence(data=tensor([[-0.2488,  0.2976,  0.1269,  ..., -0.2135,  0.0070,  0.0124],
        [-0.0638, -0.1042,  0.0642,  ..., -0.0309, -0.1741, -0.3677],
        [-0.1456, -0.1479, -0.1965,  ...,  0.0446, -0.1539, -0.3072],
        ...,
        [ 0.0868,  0.3074, -0.1079,  ..., -0.3216, -0.0831, -0.1908],
        [-0.0694,  0.0683, -0.0245,  ...,  0.4558,  0.0742, -0.1007],
        [-0.1552,  0.0083,  0.3315,  ..., -0.0243, -0.0104, -0.0924]],
       grad_fn=<CatBackward0>), batch_sizes=tensor([4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4,
        4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4,
        4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4,
        4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4,
        4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4,
        4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4,
        4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 3, 3, 3, 3, 3, 3, 3,
        3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
        3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
        3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
        3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3]), sorted_indices=None, unsorted_indices=None)
```

In [25]:
```
rebuild_out,lens=pad_packed_sequence(out,batch_first=True)
rebuild_out.shape
```

Out[25]:
```
torch.Size([4, 256, 16])
```

In [26]:
```
rebuild_out
```

Out[26]:
```
tensor([[[-0.2488,  0.2976,  0.1269,  ..., -0.2135,  0.0070,  0.0124],
         [-0.2667,  0.1188,  0.0577,  ...,  0.0030,  0.3130, -0.0260],
         [ 0.1639,  0.0831,  0.5249,  ...,  0.0229, -0.6965,  0.0098],
         ...,
         [-0.0808,  0.1115, -0.0742,  ...,  0.4136, -0.5832,  0.0930],
         [-0.1200,  0.2541, -0.0167,  ...,  0.1344, -0.5098, -0.0189],
         [ 0.0868,  0.3074, -0.1079,  ..., -0.3216, -0.0831, -0.1908]],

        [[-0.0638, -0.1042,  0.0642,  ..., -0.0309, -0.1741, -0.3677],
         [-0.0851, -0.3379,  0.0097,  ...,  0.0751, -0.5199, -0.2356],
         [-0.0428, -0.1080, -0.0458,  ...,  0.2250, -0.3142, -0.0839],
         ...,
         [-0.2397, -0.0901,  0.0552,  ...,  0.2497, -0.1559, -0.1158],
         [-0.3134, -0.1096,  0.0512,  ...,  0.1092, -0.0128, -0.3627],
         [-0.0694,  0.0683, -0.0245,  ...,  0.4558,  0.0742, -0.1007]],

        [[-0.1456, -0.1479, -0.1965,  ...,  0.0446, -0.1539, -0.3072],
         [-0.2259, -0.2712,  0.2299,  ..., -0.0200, -0.0930, -0.1122],
         [ 0.0213, -0.0124,  0.0452,  ...,  0.0840, -0.2274,  0.1213],
         ...,
         [-0.0069, -0.0754,  0.0889,  ..., -0.0616, -0.1814, -0.3487],
         [-0.0815, -0.0580,  0.1012,  ..., -0.1764, -0.2375, -0.1579],
         [-0.1552,  0.0083,  0.3315,  ..., -0.0243, -0.0104, -0.0924]],

        [[-0.1456, -0.1479, -0.1965,  ...,  0.0086, -0.0371, -0.6513],
         [ 0.0727, -0.2408, -0.2728,  ..., -0.1105, -0.0125, -0.4231],
         [-0.0269, -0.1709, -0.0090,  ..., -0.0783, -0.0301, -0.6846],
         ...,
         [ 0.0000,  0.0000,  0.0000,  ...,  0.0000,  0.0000,  0.0000],
         [ 0.0000,  0.0000,  0.0000,  ...,  0.0000,  0.0000,  0.0000],
         [ 0.0000,  0.0000,  0.0000,  ...,  0.0000,  0.0000,  0.0000]]],
       grad_fn=<TransposeBackward0>)
```

⑤ 平均池化层

平均池化层的目标

把bilstm层丢过来的一批句子的输出BxLx2D变成Bx2D，对于每个句子是这样的：
丢进去L个单词，每个单词维度d，出来L个编码，每个编码维度2D，然后将这L个编码求和平均

In [27]:
```
x_=torch.arange(0,24).reshape(4,2,3).to(torch.float)
x_
```

Out[27]:
```
tensor([[[ 0.,  1.,  2.],
         [ 3.,  4.,  5.]],

        [[ 6.,  7.,  8.],
         [ 9., 10., 11.]],

        [[12., 13., 14.],
         [15., 16., 17.]],

        [[18., 19., 20.],
         [21., 22., 23.]]])
```

In [28]:
```
x_.mean(dim=1)
```

Out[28]:
```
tensor([[ 1.5000,  2.5000,  3.5000],
        [ 7.5000,  8.5000,  9.5000],
        [13.5000, 14.5000, 15.5000],
        [19.5000, 20.5000, 21.5000]])
```

In [29]:
```
avg_out=rebuild_out.mean(dim=1)
avg_out.shape
```

Out[29]:
```
torch.Size([4, 16])
```

In [30]:
```
avg_out
```

Out[30]:
```
tensor([[-0.1580, -0.0789,  0.0774,  0.0822,  0.1270,  0.0232,  0.0201, -0.0736,
         -0.0372,  0.0406,  0.0395,  0.0990,  0.1214,  0.0093, -0.1713, -0.1395],
        [-0.1831, -0.0646,  0.0789,  0.1192,  0.1287,  0.0015,  0.0085, -0.0091,
         -0.0348, -0.0041,  0.0482,  0.0704,  0.0677,  0.0683, -0.1051, -0.0929],
        [-0.1343, -0.0809,  0.0271,  0.0953,  0.1308,  0.0424, -0.0148, -0.0335,
         -0.0005,  0.0173,  0.0684,  0.1165,  0.0984,  0.0451, -0.1562, -0.1237],
        [-0.0958, -0.0593,  0.0314,  0.0182,  0.1057,  0.0135, -0.0013, -0.0611,
          0.0013,  0.0007,  0.0041,  0.0605,  0.0665,  0.0168, -0.1061, -0.0680]],
       grad_fn=<MeanBackward1>)
```

最后上面这个avg_out就可以拿来做分类了

开始建模：组装各层组件

由于这里模型组件用的都不是自己写的，不要继承自己的nndl.Module，不然因为组件内部结构问题，
通过自己的parameters方法获取不到参数，进而无法优化更新

In [31]:
```
class mymodel(nn.Module):
    def __init__(self):
        super(mymodel,self).__init__()
        self.encoding=nn.Embedding(num_embeddings=len(data.vocab),embedding_dim=30,padding_idx=0)
        self.bilstm=nn.LSTM(input_size=30,hidden_size=8,bidirectional=True,batch_first=True)
        self.fc=nn.Linear(16,2)
        self.softmax=nn.Softmax(dim=1)

    def forward(self,batch_x):
        x,seq_lens=batch_x                                         # BxL
        B=x.shape[0]
        h0=torch.zeros(2,B,8)                                     # NxBxD
        c0=torch.zeros(2,B,8)
        x=self.encoding(x)                                         # BxLxd
        x=pack_padded_sequence(x,lengths=seq_lens,batch_first=True)  # sum(seq_lens) x d
        x,(h_o,c_o)=self.bilstm(x,(h0,c0))                              # sum(seq_lens) x 2D
        x,seq_lens=pad_packed_sequence(x,batch_first=True)           # B x L x 2D
        x=x.mean(dim=1)                                            # B x 2D
        x=self.fc(x)                                               # B x 2
        x=self.softmax(x)                                          # B x 2
        return x
```

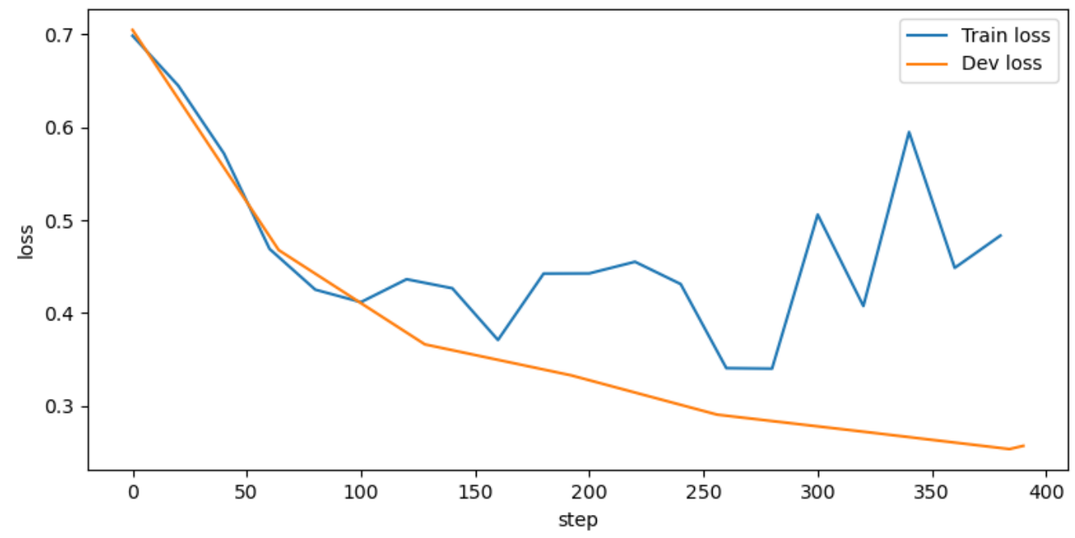
In [32]:
```
model=mymodel()
loss_fn=nndl.CrossEntropyLoss()
metric=nndl.Accuracy()
```

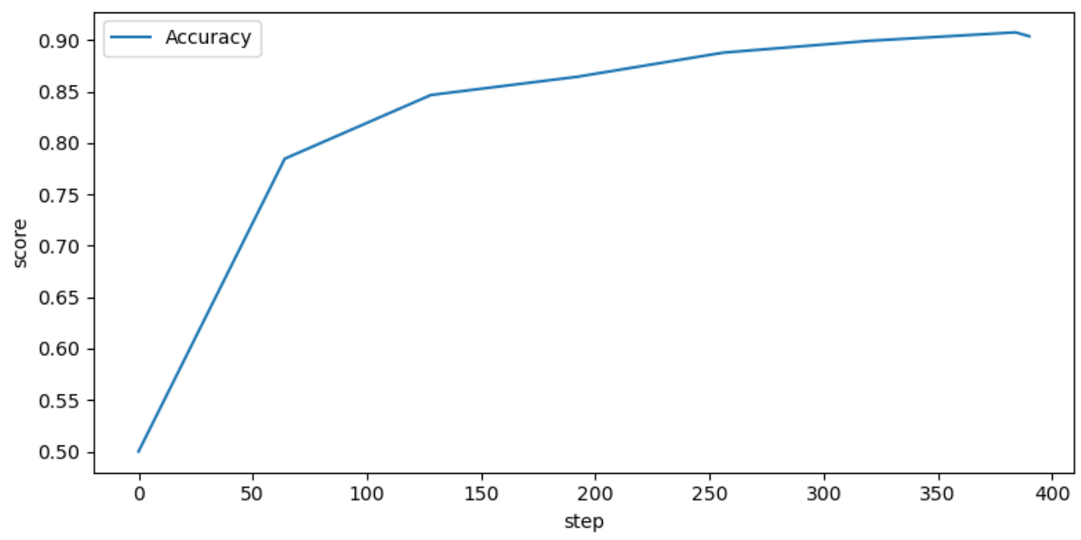
In [33]:
```
opt=nndl.optim.Adam(params=model.parameters(),lr=0.1)
runner=nndl.RunnerV1(model=model,loss_fn=loss_fn,optimizer=opt,metric=metric)
```

In [34]:
```
train_loader4=nndl.utils.DataLoader(trainset,batch_size=64,collate_fn=collate_fn3,shuffle=False)
```

In [35]:
```
runner.train(train_loader=train_loader4,dev_loader=train_loader4,num_epochs=1,log_stride=64)
```

```
[Train]  epoch:0/1 step:0/391 loss:0.6986
[Evaluate]  score:0.5000 loss:0.7049
[Train]  epoch:0/1 step:64/391 loss:0.5038
[Evaluate]  score:0.7846 loss:0.4678
[Train]  epoch:0/1 step:128/391 loss:0.4073
[Evaluate]  score:0.8466 loss:0.3663
[Train]  epoch:0/1 step:192/391 loss:0.3183
[Evaluate]  score:0.8643 loss:0.3329
[Train]  epoch:0/1 step:256/391 loss:0.2131
[Evaluate]  score:0.8878 loss:0.2906
[Train]  epoch:0/1 step:320/391 loss:0.4076
[Evaluate]  score:0.8993 loss:0.2723
[Train]  epoch:0/1 step:384/391 loss:0.3818
[Evaluate]  score:0.9075 loss:0.2535
[Train]  epoch:0/1 step:390/391 loss:0.4050
[Evaluate]  score:0.9037 loss:0.2570
```





In [ ]:

In [ ]:

In [ ]:

In [ ]: