6.3 语言模型数据集（周杰伦专辑歌词）

本节将介绍如何预处理一个语言模型数据集，并将其转换成字符级循环神经网络所需要的输入格式。为此，我们收集了周杰伦从第一张专辑《Jay》到第十张专辑《跨时代》中的歌词，并在后面几节里应用循环神经网络来训练一个语言模型。当模型训练好后，我们就可以用这个模型来创作歌词。

6.3.1 读取数据集

首先读取这个数据集，看看前40个字符是什么样的。

```
import torch
import random
import zipfile

with zipfile.ZipFile('../../data/jaychou_lyrics.txt.zip') as zin:
    with zin.open('jaychou_lyrics.txt') as f:
        corpus_chars = f.read().decode('utf-8')
corpus_chars[:40]
```

输出：

```
'想要有直升机\n想要和你飞到宇宙去\n想要和你融化在一起\n融化在宇宙里\n我每天每天每'
```

这个数据集有6万多个字符。为了打印方便，我们把换行符替换成空格，然后仅使用前1万个字符来训练模型。

```
corpus_chars = corpus_chars.replace('\n', ' ').replace('\r', ' ')
corpus_chars = corpus_chars[0:10000]
```

6.3.2 建立字符索引

我们将每个字符映射成一个从0开始的连续整数，又称索引，来方便之后的数据处理。为了得到索引，我们将数据集里所有不同字符取出来，然后将其逐一映射到索引来构造词典。接着，打印 `vocab_size` ，即词典中不同字符的个数，又称词典大小。

```
idx_to_char = list(set(corpus_chars))
char_to_idx = dict([(char, i) for i, char in enumerate(idx_to_char)])
vocab_size = len(char_to_idx)
vocab_size # 1027
```

之后，将训练数据集中每个字符转化为索引，并打印前20个字符及其对应的索引。

```
corpus_indices = [char_to_idx[char] for char in corpus_chars]
sample = corpus_indices[:20]
print('chars:', ''.join([idx_to_char[idx] for idx in sample]))
print('indices:', sample)
```

输出：

```
chars: 想要有直升机 想要和你飞到宇宙去 想要和
indices: [250, 164, 576, 421, 674, 653, 357, 250, 164, 850, 217, 910, 1012, 261, 275,
```

我们将以上代码封装在 `d2lzh_pytorch` 包里的 `load_data_jay_lyrics` 函数中，以方便后面章节调用。调用该函数后会依次得到 `corpus_indices` 、 `char_to_idx` 、 `idx_to_char` 和 `vocab_size` 这4个变量。

6.3.3 时序数据的采样

在训练中我们需要每次随机读取小批量样本和标签。与之前章节的实验数据不同的是，时序数据的一个样本通常包含连续的字符。假设时间步数为5，样本序列为5个字符，即“想”“要”“有”“直”“升”。该样本的标签序列为这些字符分别在训练集中的下一个字符，即“要”“有”“直”“升”“机”。我们有两种方式对时序数据进行采样，分别是随机采样和相邻采样。

6.3.3.1 随机采样

下面的代码每次从数据里随机采样一个小批量。其中批量大小 `batch_size` 指每个小批量的样本数， `num_steps` 为每个样本所包含的时间步数。 在随机采样中，每个样本是原始序列上任意截取的一段序列。相邻的两个随机小批量在原始序列上的位置不一定相毗邻。因此，我们无法用一个小批量最终时间步的隐藏状态来初始化下一个小批量的隐藏状态。在训练模型时，每次随机采样前都需要重新初始化隐藏状态。

```
# 本函数已保存在d2lzh_pytorch包中方便以后使用
def data_iter_random(corpus_indices, batch_size, num_steps, device=None):
    # 减1是因为输出的索引x是相应输入的索引y加1
    num_examples = (len(corpus_indices) - 1) // num_steps
    epoch_size = num_examples // batch_size
    example_indices = list(range(num_examples))
    random.shuffle(example_indices)

    # 返回从pos开始的长为num_steps的序列
    def _data(pos):
        return corpus_indices[pos: pos + num_steps]
    if device is None:
        device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')

    for i in range(epoch_size):
        # 每次读取batch_size个随机样本
        i = i * batch_size
        batch_indices = example_indices[i: i + batch_size]
        X = [_data(j * num_steps) for j in batch_indices]
        Y = [_data(j * num_steps + 1) for j in batch_indices]
        yield torch.tensor(X, dtype=torch.float32, device=device), torch.tensor(Y, dt
```

让我们输入一个从0到29的连续整数的人工序列。设批量大小和时间步数分别为2和6。打印随机采样每次读取的小批量样本的输入 `X` 和标签 `Y` 。可见，相邻的两个随机小批量在原始序列上的位置不一定相毗邻。

```
my_seq = list(range(30))
for X, Y in data_iter_random(my_seq, batch_size=2, num_steps=6):
    print('X: ', X, '\nY:', Y, '\n')
```

输出：

```
X:  tensor([[18., 19., 20., 21., 22., 23.],
        [12., 13., 14., 15., 16., 17.]])
Y: tensor([[19., 20., 21., 22., 23., 24.],
        [13., 14., 15., 16., 17., 18.]])

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

```
        [ 6.,  7.,  8.,  9., 10., 11.]])
Y: tensor([[ 1.,  2.,  3.,  4.,  5.,  6.],
        [ 7.,  8.,  9., 10., 11., 12.]])
```

6.3.3.2 相邻采样

除对原始序列做随机采样之外，我们还可以令相邻的两个随机小批量在原始序列上的位置相毗邻。这时候，我们就可以用一个小批量最终时间步的隐藏状态来初始化下一个小批量的隐藏状态，从而使下一个小批量的输出也取决于当前小批量的输入，并如此循环下去。这对实现循环神经网络造成了两方面影响：一方面， 在训练模型时，我们只需在每一个迭代周期开始时初始化隐藏状态；另一方面，当多个相邻小批量通过传递隐藏状态串联起来时，模型参数的梯度计算将依赖所有串联起来的小批量序列。同一迭代周期中，随着迭代次数的增加，梯度的计算开销会越来越大。 为了使模型参数的梯度计算只依赖一次迭代读取的小批量序列，我们可以在每次读取小批量前将隐藏状态从计算图中分离出来。我们将在下一节（循环神经网络的从零开始实现）的实现中了解这种处理方式。

```
# 本函数已保存在d2lzh_pytorch包中方便以后使用
def data_iter_consecutive(corpus_indices, batch_size, num_steps, device=None):
    if device is None:
        device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
    corpus_indices = torch.tensor(corpus_indices, dtype=torch.float32, device=device)
    data_len = len(corpus_indices)
    batch_len = data_len // batch_size
    indices = corpus_indices[0: batch_size*batch_len].view(batch_size, batch_len)
    epoch_size = (batch_len - 1) // num_steps
    for i in range(epoch_size):
        i = i * num_steps
        X = indices[:, i: i + num_steps]
        Y = indices[:, i + 1: i + num_steps + 1]
        yield X, Y
```

同样的设置下，打印相邻采样每次读取的小批量样本的输入 `X` 和标签 `Y` 。相邻的两个随机小批量在原始序列上的位置相毗邻。

```
for X, Y in data_iter_consecutive(my_seq, batch_size=2, num_steps=6):
    print('X: ', X, '\nY:', Y, '\n')
```

输出：

Copy to clipboard

```
X:  tensor([[ 0.,  1.,  2.,  3.,  4.,  5.],
        [15., 16., 17., 18., 19., 20.]])
Y: tensor([[ 1.,  2.,  3.,  4.,  5.,  6.],
        [16., 17., 18., 19., 20., 21.]])

X:  tensor([[ 6.,  7.,  8.,  9., 10., 11.],
        [21., 22., 23., 24., 25., 26.]])
Y: tensor([[ 7.,  8.,  9., 10., 11., 12.],
        [22., 23., 24., 25., 26., 27.]])
```

小结

- 时序数据采样方式包括随机采样和相邻采样。使用这两种方式的循环神经网络训练在实现上略有不同。