

实验四 机器翻译

下载和预处理数据集

1. 下载数据集;

```
# 下载和预处理数据集
#@save
d2l.DATA_HUB['fra-eng']=(d2l.DATA_URL+'fra-eng.zip',
                          '94646ad1522d915e7b0f9296181140edcf86a4f5')

# 读取数据集
#@save
def read_data_nmt():
    """载入英语-法语数据集"""
    data_dir=d2l.download_extract('fra-eng')
    with open(os.path.join(data_dir,'fra.txt'),'r',encoding='utf8') as f:
        return f.read()

raw_text=read_data_nmt()
# 输出数据集的前75个字符
print(raw_text[:75])
```

```
Go.      Va !
Hi.      Salut !
Run!     Cours !
Run!     Courez !
Who?     Qui ?
Wow!     Ça alors !
```

2. 数据预处理;

```
# 预处理数据集
#@save
def preprocess_nmt(text):
    def no_space(char, prev_char):
        return char in set(',.!?') and prev_char != ' '
    # 使用空格替换不间断空格
    # 使用小写字母替换大写字母
    text = text.replace('\u202f', ' ').replace('\xa0', ' ').lower()
    # 在单词和标点符号之间插入空格
    out = [' ' + char if i > 0 and no_space(char, text[i - 1]) else char
           for i, char in enumerate(text)]
    return ''.join(out)
```

```
text=preprocess_nmt(raw_text)
print(text[:80])
```

```
go .      va !
hi .      salut !
run !     cours !
run !     courez !
who ?     qui ?
wow !     ça alors !
```

词元化

1. 将数据集中的每一行分割为源语言和目的语言；

```
# 词元化
#@save
def tokenize_nmt(text, num_examples=None):
    source, target=[], []
    for i, line in enumerate(text.split('\n')):
        if num_examples and i>num_examples:
            break
        parts=line.split('\t')
        if len(parts)==2:
            source.append(parts[0].split(' '))
            target.append(parts[1].split(' '))
    return source, target

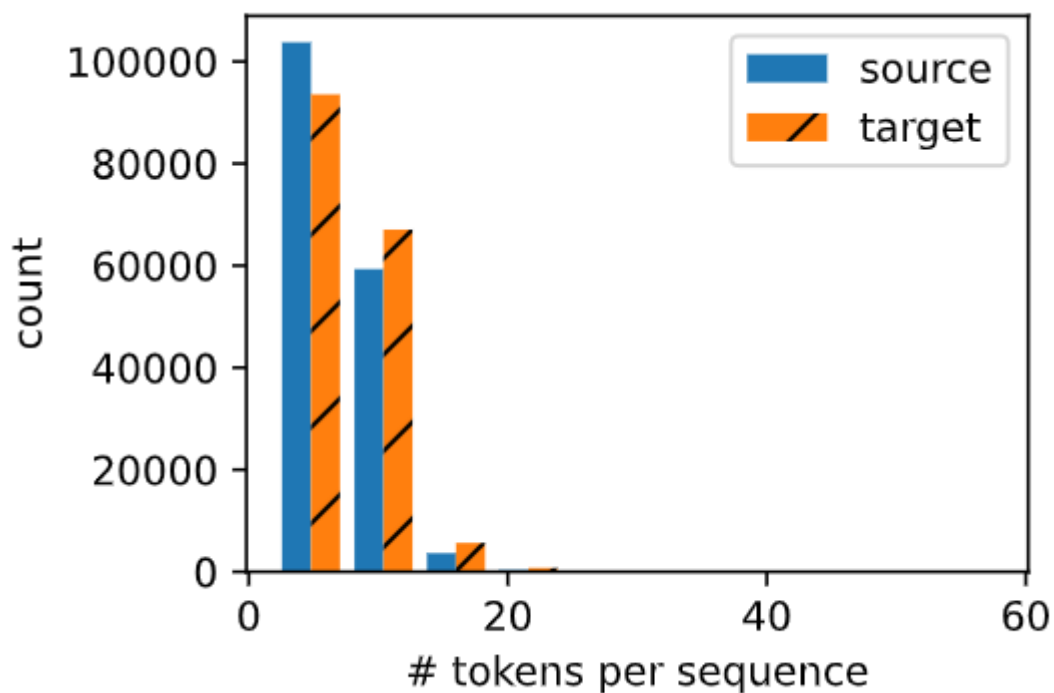
source, target=tokenize_nmt(text)
source[:6], target[:6]
```

```
Out[9]: ([[ 'go', '.'],
          [ 'hi', '.'],
          [ 'run', '!'],
          [ 'run', '!'],
          [ 'who', '?'],
          [ 'wow', '!']],
          [[ 'va', '!'],
          [ 'salut', '!'],
          [ 'cours', '!'],
          [ 'courez', '!'],
          [ 'qui', '?'],
          [ 'ça', 'alors', '!']])
```

2. 绘制词元长度对直方图

```
# 绘制词元长度对直方图
#@save
def show_list_len_pair_hist(legend, xlabel, ylabel, xlist, ylist):
    d2l.set_figsize()
    _,_,patches=d2l.plt.hist([
        [len(l) for l in xlist], [len(l) for l in ylist]
    ])
    d2l.plt.xlabel(xlabel)
    d2l.plt.ylabel(ylabel)
    for patch in patches[1].patches:
        patch.set_hatch('/')
    d2l.plt.legend(legend)

show_list_len_pair_hist(['source', 'target'], '# tokens per sequence',
                        'count', source, target)
```



构建词表

```
src_vocab = d2l.Vocab(source, min_freq=2,
                      reserved_tokens=['<pad>', '<bos>', '<eos>'])

len(src_vocab)
```

Vocab函数用于构建和管理词汇表，可以帮助我们将文本数据中的词汇映射到数字索引。

```
# 查看词汇表中的词汇及对应的索引
print(src_vocab.token_to_idx)
```

```
{' <unk>': 0, ' <pad>': 1, ' <bos>': 2, ' <eos>': 3, '.': 4, 'i': 5,
'you': 6, 'to': 7, 'the': 8, '?': 9, 'a': 10, 'is': 11, 'tom': 1
2, 'that': 13, 'he': 14, 'do': 15, 'of': 16, 'it': 17, 'this': 1
8, 'in': 19, 'me': 20, 'have': 21, "don't": 22, ',': 23, 'was':
24, 'my': 25, 'are': 26, 'for': 27, 'your': 28, 'what': 29,
'i'm': 30, 'we': 31, 'be': 32, 'want': 33, 'she': 34, 'not': 35,
'know': 36, 'like': 37, 'on': 38, 'with': 39, 'can': 40, 'his':
41, 'all': 42, 'did': 43, 'at': 44, "you're": 45, 'how': 46, 'g
o': 47, 'they': 48, 'him': 49, 'think': 50, 'and': 51, "it's": 5
2, 'about': 53, 'time': 54, "can't": 55, 'here': 56, 'very': 57,
"didn't": 58, 'get': 59, 'there': 60, 'her': 61, 'were': 62, 'a
s': 63, 'will': 64, 'had': 65, 'if': 66, 'why': 67, 'just': 68,
'up': 69, 'out': 70, 'no': 71, 'has': 72, 'one': 73, 'going': 7
4, 'would': 75, 'so': 76, 'good': 77, 'need': 78, 'tell': 79, 'a
n': 80, 'see': 81, "i'll": 82, 'come': 83, 'when': 84, 'from': 8
5, 'by': 86, 'really': 87, 'mary': 88, 'help': 89, 'who': 90, 'p
lease': 91, 'us': 92, "that's": 93, 'should': 94, 'could': 95,
'been': 96, 'i've': 97, 'never': 98, 'more': 99, 'now': 100, 'wh
ere': 101, 'take': 102, 'something': 103, 'got': 104, 'too': 10
5, 'that': 106, 'which': 107, 'about': 108, 'how': 109, 'if': 110, 'and': 111, 'or': 112, 'but': 113, 'so': 114, 'because': 115, 'as': 116, 'for': 117, 'with': 118, 'from': 119, 'to': 120, 'at': 121, 'on': 122, 'in': 123, 'by': 124, 'of': 125, 'the': 126, 'a': 127, 'an': 128, 'the': 129, 'and': 130, 'but': 131, 'or': 132, 'so': 133, 'because': 134, 'as': 135, 'for': 136, 'with': 137, 'from': 138, 'to': 139, 'at': 140, 'on': 141, 'in': 142, 'by': 143, 'of': 144, 'the': 145, 'a': 146, 'an': 147, 'the': 148, 'and': 149, 'but': 150, 'or': 151, 'so': 152, 'because': 153, 'as': 154, 'for': 155, 'with': 156, 'from': 157, 'to': 158, 'at': 159, 'on': 160, 'in': 161, 'by': 162, 'of': 163, 'the': 164, 'a': 165, 'an': 166, 'the': 167, 'and': 168, 'but': 169, 'or': 170, 'so': 171, 'because': 172, 'as': 173, 'for': 174, 'with': 175, 'from': 176, 'to': 177, 'at': 178, 'on': 179, 'in': 180, 'by': 181, 'of': 182, 'the': 183, 'a': 184, 'an': 185, 'the': 186, 'and': 187, 'but': 188, 'or': 189, 'so': 190, 'because': 191, 'as': 192, 'for': 193, 'with': 194, 'from': 195, 'to': 196, 'at': 197, 'on': 198, 'in': 199, 'by': 200, 'of': 201, 'the': 202, 'a': 203, 'an': 204, 'the': 205, 'and': 206, 'but': 207, 'or': 208, 'so': 209, 'because': 210, 'as': 211, 'for': 212, 'with': 213, 'from': 214, 'to': 215, 'at': 216, 'on': 217, 'in': 218, 'by': 219, 'of': 220, 'the': 221, 'a': 222, 'an': 223, 'the': 224, 'and': 225, 'but': 226, 'or': 227, 'so': 228, 'because': 229, 'as': 230, 'for': 231, 'with': 232, 'from': 233, 'to': 234, 'at': 235, 'on': 236, 'in': 237, 'by': 238, 'of': 239, 'the': 240, 'a': 241, 'an': 242, 'the': 243, 'and': 244, 'but': 245, 'or': 246, 'so': 247, 'because': 248, 'as': 249, 'for': 250, 'with': 251, 'from': 252, 'to': 253, 'at': 254, 'on': 255, 'in': 256, 'by': 257, 'of': 258, 'the': 259, 'a': 260, 'an': 261, 'the': 262, 'and': 263, 'but': 264, 'or': 265, 'so': 266, 'because': 267, 'as': 268, 'for': 269, 'with': 270, 'from': 271, 'to': 272, 'at': 273, 'on': 274, 'in': 275, 'by': 276, 'of': 277, 'the': 278, 'a': 279, 'an': 280, 'the': 281, 'and': 282, 'but': 283, 'or': 284, 'so': 285, 'because': 286, 'as': 287, 'for': 288, 'with': 289, 'from': 290, 'to': 291, 'at': 292, 'on': 293, 'in': 294, 'by': 295, 'of': 296, 'the': 297, 'a': 298, 'an': 299, 'the': 300, 'and': 301, 'but': 302, 'or': 303, 'so': 304, 'because': 305, 'as': 306, 'for': 307, 'with': 308, 'from': 309, 'to': 310, 'at': 311, 'on': 312, 'in': 313, 'by': 314, 'of': 315, 'the': 316, 'a': 317, 'an': 318, 'the': 319, 'and': 320, 'but': 321, 'or': 322, 'so': 323, 'because': 324, 'as': 325, 'for': 326, 'with': 327, 'from': 328, 'to': 329, 'at': 330, 'on': 331, 'in': 332, 'by': 333, 'of': 334, 'the': 335, 'a': 336, 'an': 337, 'the': 338, 'and': 339, 'but': 340, 'or': 341, 'so': 342, 'because': 343, 'as': 344, 'for': 345, 'with': 346, 'from': 347, 'to': 348, 'at': 349, 'on': 350, 'in': 351, 'by': 352, 'of': 353, 'the': 354, 'a': 355, 'an': 356, 'the': 357, 'and': 358, 'but': 359, 'or': 360, 'so': 361, 'because': 362, 'as': 363, 'for': 364, 'with': 365, 'from': 366, 'to': 367, 'at': 368, 'on': 369, 'in': 370, 'by': 371, 'of': 372, 'the': 373, 'a': 374, 'an': 375, 'the': 376, 'and': 377, 'but': 378, 'or': 379, 'so': 380, 'because': 381, 'as': 382, 'for': 383, 'with': 384, 'from': 385, 'to': 386, 'at': 387, 'on': 388, 'in': 389, 'by': 390, 'of': 391, 'the': 392, 'a': 393, 'an': 394, 'the': 395, 'and': 396, 'but': 397, 'or': 398, 'so': 399, 'because': 400, 'as': 401, 'for': 402, 'with': 403, 'from': 404, 'to': 405, 'at': 406, 'on': 407, 'in': 408, 'by': 409, 'of': 410, 'the': 411, 'a': 412, 'an': 413, 'the': 414, 'and': 415, 'but': 416, 'or': 417, 'so': 418, 'because': 419, 'as': 420, 'for': 421, 'with': 422, 'from': 423, 'to': 424, 'at': 425, 'on': 426, 'in': 427, 'by': 428, 'of': 429, 'the': 430, 'a': 431, 'an': 432, 'the': 433, 'and': 434, 'but': 435, 'or': 436, 'so': 437, 'because': 438, 'as': 439, 'for': 440, 'with': 441, 'from': 442, 'to': 443, 'at': 444, 'on': 445, 'in': 446, 'by': 447, 'of': 448, 'the': 449, 'a': 450, 'an': 451, 'the': 452, 'and': 453, 'but': 454, 'or': 455, 'so': 456, 'because': 457, 'as': 458, 'for': 459, 'with': 460, 'from': 461, 'to': 462, 'at': 463, 'on': 464, 'in': 465, 'by': 466, 'of': 467, 'the': 468, 'a': 469, 'an': 470, 'the': 471, 'and': 472, 'but': 473, 'or': 474, 'so': 475, 'because': 476, 'as': 477, 'for': 478, 'with': 479, 'from': 480, 'to': 481, 'at': 482, 'on': 483, 'in': 484, 'by': 485, 'of': 486, 'the': 487, 'a': 488, 'an': 489, 'the': 490, 'and': 491, 'but': 492, 'or': 493, 'so': 494, 'because': 495, 'as': 496, 'for': 497, 'with': 498, 'from': 499, 'to': 500, 'at': 501, 'on': 502, 'in': 503, 'by': 504, 'of': 505, 'the': 506, 'a': 507, 'an': 508, 'the': 509, 'and': 510, 'but': 511, 'or': 512, 'so': 513, 'because': 514, 'as': 515, 'for': 516, 'with': 517, 'from': 518, 'to': 5
```

加载数据集

语言模型的序列样本必须有一个固定的长度，我们使用截断和填充的方式实现处理一个小批量的文本序列。

```
# 截断或填充文本序列
#@save
def truncate_pad(line, num_steps, padding_token):
    if len(line)>num_steps:
        return line[:num_steps] # 截断
    return line+[padding_token]*(num_steps-len(line)) # 填充

truncate_pad(src_vocab[source[0]], 10, src_vocab['<pad>'])
```

```
Out[49]: [47, 4, 1, 1, 1, 1, 1, 1, 1, 1]
```

将文本序列转换成小批量数据集进行训练。

```
# 将机器翻译的文本序列转换成小批量
#@save
def build_array_nmt(lines, vocab, num_steps):
    lines = [vocab[l] for l in lines]
    lines = [l + [vocab['<eos>']] for l in lines]
    array = torch.tensor([truncate_pad(
        l, num_steps, vocab['<pad>']) for l in lines])
    valid_len = (array != vocab['<pad>']).type(torch.int32).sum(1)
    return array, valid_len
```

训练模型

```

#@save
def load_data_nmt(batch_size, num_steps, num_examples=600):
    """返回翻译数据集的迭代器和词表"""
    text = preprocess_nmt(read_data_nmt())
    source, target = tokenize_nmt(text, num_examples)
    src_vocab = d2l.Vocab(source, min_freq=2,
                          reserved_tokens=['<pad>', '<bos>', '<eos>'])
    tgt_vocab = d2l.Vocab(target, min_freq=2,
                          reserved_tokens=['<pad>', '<bos>', '<eos>'])
    src_array, src_valid_len = build_array_nmt(source, src_vocab, num_steps)
    tgt_array, tgt_valid_len = build_array_nmt(target, tgt_vocab, num_steps)
    data_arrays = (src_array, src_valid_len, tgt_array, tgt_valid_len)
    data_iter = d2l.load_array(data_arrays, batch_size)
    return data_iter, src_vocab, tgt_vocab

```

读取数据集的第一个小批量数据。

```

train_iter, src_vocab, tgt_vocab = load_data_nmt(batch_size=2, num_steps=8)
for X, X_valid_len, Y, Y_valid_len in train_iter:
    print('X:', X.type(torch.int32))
    print('x的有效长度:', X_valid_len)
    print('Y:', Y.type(torch.int32))
    print('y的有效长度:', Y_valid_len)
    break

```

```

X: tensor([[ 9, 173, 118,  4,  3,  1,  1,  1],
           [39,  0,  5,  3,  1,  1,  1,  1]], dtype=torch.int32)
x的有效长度: tensor([5, 4])
Y: tensor([[45, 46, 151,  5,  3,  1,  1,  1],
           [47, 12, 106,  5,  3,  1,  1,  1]], dtype=torch.int32)
y的有效长度: tensor([5, 5])

```