10.7 文本情感分类：使用循环神经网络

文本分类是自然语言处理的一个常见任务，它把一段不定长的文本序列变换为文本的类别。本节关注它的一个子问题：使用文本情感分类来分析文本作者的情绪。这个问题也叫情感分析，并有着广泛的应用。例如，我们可以分析用户对产品的评论并统计用户的满意度，或者分析用户对市场行情的情绪并用以预测接下来的行情。

同搜索近义词和类比词一样，文本分类也属于词嵌入的下游应用。在本节中，我们将应用预训练的词向量和含多个隐藏层的双向循环神经网络，来判断一段不定长的文本序列中包含的是正面还是负面的情绪。

在实验开始前，导入所需的包或模块。

```
import collections
import os
import random
import tarfile
import torch
from torch import nn
import torchtext.vocab as Vocab
import torch.utils.data as Data

import sys
sys.path.append("..")
import d2lzh_pytorch as d2l

os.environ["CUDA_VISIBLE_DEVICES"] = "0"
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')

DATA_ROOT = "/S1/CSCL/tangss/Datasets"
```

10.7.1 文本情感分类数据

我们使用斯坦福的IMDb数据集（Stanford's Large Movie Review Dataset）作为文本情感分类的数据集[1]。这个数据集分为训练和测试用的两个数据集，分别包含25,000条从IMDb下载的关于电影的评论。在每个数据集中，标签为“正面”和“负面”的评论数量相等。

10.7.1.1 读取数据

首先下载这个数据集到 `DATA_ROOT` 路径下，然后解压。

```
fname = os.path.join(DATA_ROOT, "aclImdb_v1.tar.gz")
if not os.path.exists(os.path.join(DATA_ROOT, "aclImdb")):
    print("从压缩包解压...")
    with tarfile.open(fname, 'r') as f:
        f.extractall(DATA_ROOT)
```

接下来，读取训练数据集和测试数据集。每个样本是一条评论及其对应的标签：1表示“正面”，0表示“负面”。

```
from tqdm import tqdm
# 本函数已保存在d2lzh_pytorch包中方便以后使用
def read_imdb(folder='train', data_root="/S1/CSCL/tangss/Datasets/aclImdb"):
    data = []
    for label in ['pos', 'neg']:
        folder_name = os.path.join(data_root, folder, label)
        for file in tqdm(os.listdir(folder_name)):
            with open(os.path.join(folder_name, file), 'rb') as f:
                review = f.read().decode('utf-8').replace('\n', '').lower()
                data.append([review, 1 if label == 'pos' else 0])
    random.shuffle(data)
    return data

train_data, test_data = read_imdb('train'), read_imdb('test')
```

10.7.1.2 预处理数据

我们需要对每条评论做分词，从而得到分好词的评论。这里定义的 `get_tokenized_imdb` 函数使用最简单的方法：基于空格进行分词。

```
# 本函数已保存在d2lzh_pytorch包中方便以后使用
def get_tokenized_imdb(data):
    """
    data: list of [string, label]
    """
    def tokenizer(text):
```

```
            return [tok.lower() for tok in text.split(' ')]
        return [tokenizer(review) for review, _ in data]
```

现在，我们可以根据分好词的训练数据集来创建词典了。我们在这里过滤掉了出现次数少于5的词。

```
# 本函数已保存在d2lzh_pytorch包中方便以后使用
def get_vocab_imdb(data):
    tokenized_data = get_tokenized_imdb(data)
    counter = collections.Counter([tk for st in tokenized_data for tk in st])
    return Vocab.Vocab(counter, min_freq=5)

vocab = get_vocab_imdb(train_data)
'# words in vocab:', len(vocab)
```

输出：

```
('# words in vocab:', 46151)
```

因为每条评论长度不一致所以不能直接组合成小批量，我们定义 `preprocess_imdb` 函数对每条评论进行分词，并通过词典转换成词索引，然后通过截断或者补0来将每条评论长度固定成500。

```
# 本函数已保存在d2lzh_torch包中方便以后使用
def preprocess_imdb(data, vocab):
    max_l = 500  # 将每条评论通过截断或者补0，使得长度变成500

    def pad(x):
        return x[:max_l] if len(x) > max_l else x + [0] * (max_l - len(x))

    tokenized_data = get_tokenized_imdb(data)
    features = torch.tensor([pad([vocab.stoi[word] for word in words]) for words in t
    labels = torch.tensor([score for _, score in data])
    return features, labels
```

10.7.1.3 创建数据迭代器

现在，我们创建数据迭代器。每次迭代将返回一个小批量的数据。

```
batch_size = 64
train_set = Data.TensorDataset(*preprocess_imdb(train_data, vocab))
test_set = Data.TensorDataset(*preprocess_imdb(test_data, vocab))
train_iter = Data.DataLoader(train_set, batch_size, shuffle=True)
test_iter = Data.DataLoader(test_set, batch_size)
```

打印第一个小批量数据的形状以及训练集中小批量的个数。

```
for X, y in train_iter:
    print('X', X.shape, 'y', y.shape)
    break
'#batches:', len(train_iter)
```

输出：

```
X torch.Size([64, 500]) y torch.Size([64])
('#batches:', 391)
```

10.7.2 使用循环神经网络的模型

在这个模型中，每个词先通过嵌入层得到特征向量。然后，我们使用双向循环神经网络对特征序列进一步编码得到序列信息。最后，我们将编码的序列信息通过全连接层变换为输出。具体来说，我们可以将双向长短期记忆在最初时间步和最终时间步的隐藏状态连结，作为特征序列的表征传递给输出层分类。在下面实现的 `BiRNN` 类中， `Embedding` 实例即嵌入层， `LSTM` 实例即为序列编码的隐藏层， `Linear` 实例即生成分类结果的输出层。

```
class BiRNN(nn.Module):
    def __init__(self, vocab, embed_size, num_hiddens, num_layers):
        super(BiRNN, self).__init__()
        self.embedding = nn.Embedding(len(vocab), embed_size)
        # bidirectional设为True即得到双向循环神经网络
        self.encoder = nn.LSTM(input_size=embed_size,
                               hidden_size=num_hiddens,
```

```
                                num_layers=num_layers,
                                bidirectional=True)
        # 初始时间步和最终时间步的隐藏状态作为全连接层输入
        self.decoder = nn.Linear(4*num_hiddens, 2)

    def forward(self, inputs):
        # inputs的形状是(批量大小, 词数)，因为LSTM需要将序列长度(seq_len)作为第一维，所以
        # 再提取词特征，输出形状为(词数, 批量大小, 词向量维度)
        embeddings = self.embedding(inputs.permute(1, 0))
        # rnn.LSTM只传入输入embeddings，因此只返回最后一层的隐藏层在各时间步的隐藏状态。
        # outputs形状是(词数, 批量大小, 2 * 隐藏单元个数)
        outputs, _ = self.encoder(embeddings) # output, (h, c)
        # 连结初始时间步和最终时间步的隐藏状态作为全连接层输入。它的形状为
        # (批量大小, 4 * 隐藏单元个数)。
        encoding = torch.cat((outputs[0], outputs[-1]), -1)
        outs = self.decoder(encoding)
        return outs
```

创建一个含两个隐藏层的双向循环神经网络。

```
embed_size, num_hiddens, num_layers = 100, 100, 2
net = BiRNN(vocab, embed_size, num_hiddens, num_layers)
```

10.7.2.1 加载预训练的词向量

由于情感分类的训练数据集并不是很大，为应对过拟合，我们将直接使用在更大规模语料上预训练的词向量作为每个词的特征向量。这里，我们为词典 `vocab` 中的每个词加载100维的GloVe词向量。

```
glove_vocab = Vocab.GloVe(name='6B', dim=100, cache=os.path.join(DATA_ROOT, "glove"))
```

然后，我们将用这些词向量作为评论中每个词的特征向量。注意，预训练词向量的维度需要与创建的模型中的嵌入层输出大小 `embed_size` 一致。此外，在训练中我们不再更新这些词向量。

```
# 本函数已保存在d2lzh_torch包中方便以后使用
def load_pretrained_embedding(words, pretrained_vocab):
```

```
    """从预训练好的vocab中提取出words对应的词向量"""
    embed = torch.zeros(len(words), pretrained_vocab.vectors[0].shape[0]) # 初始化为0
    oov_count = 0 # out of vocabulary
    for i, word in enumerate(words):
        try:
            idx = pretrained_vocab.stoi[word]
            embed[i, :] = pretrained_vocab.vectors[idx]
        except KeyError:
            oov_count += 1
    if oov_count > 0:
        print("There are %d oov words." % oov_count)
    return embed

net.embedding.weight.data.copy_(
    load_pretrained_embedding(vocab.itos, glove_vocab))
net.embedding.weight.requires_grad = False # 直接加载预训练好的, 所以不需要更新它
```

输出:

```
There are 21202 oov words.
```

10.7.2.2 训练并评价模型

这时候就可以开始训练模型了。

```
lr, num_epochs = 0.01, 5
# 要过滤掉不计算梯度的embedding参数
optimizer = torch.optim.Adam(filter(lambda p: p.requires_grad, net.parameters()), lr=
loss = nn.CrossEntropyLoss()
d2l.train(train_iter, test_iter, net, loss, optimizer, device, num_epochs)
```

输出:

```
training on  cuda
epoch 1, loss 0.5759, train acc 0.666, test acc 0.832, time 250.8 sec
```

```
epoch 2, loss 0.1785, train acc 0.842, test acc 0.852, time 253.3 sec
epoch 3, loss 0.1042, train acc 0.866, test acc 0.856, time 253.7 sec
epoch 4, loss 0.0682, train acc 0.888, test acc 0.868, time 254.2 sec
epoch 5, loss 0.0483, train acc 0.901, test acc 0.862, time 251.4 sec
```

最后，定义预测函数。

```
# 本函数已保存在d2lzh_pytorch包中方便以后使用
def predict_sentiment(net, vocab, sentence):
    """sentence是词语的列表"""
    device = list(net.parameters())[0].device
    sentence = torch.tensor([vocab.stoi[word] for word in sentence], device=device)
    label = torch.argmax(net(sentence.view((1, -1))), dim=1)
    return 'positive' if label.item() == 1 else 'negative'
```

下面使用训练好的模型对两个简单句子的情感进行分类。

```
predict_sentiment(net, vocab, ['this', 'movie', 'is', 'so', 'great']) # positive
```

```
predict_sentiment(net, vocab, ['this', 'movie', 'is', 'so', 'bad']) # negative
```

小结

- 文本分类把一段不定长的文本序列变换为文本的类别。它属于词嵌入的下游应用。
- 可以应用预训练的词向量和循环神经网络对文本的情感进行分类。