数据准备和预处理

1. 下载并提取IMDb评论数据集。

```
# 下载数据集
#@save
d2l.DATA_HUB['aclImdb'] = (
    'http://ai.stanford.edu/~amaas/data/sentiment/aclImdb_v1.tar.gz',
    '01ada507287d82875905620988597833ad4e0903')
```

2. 读取数据集并进行分词处理。

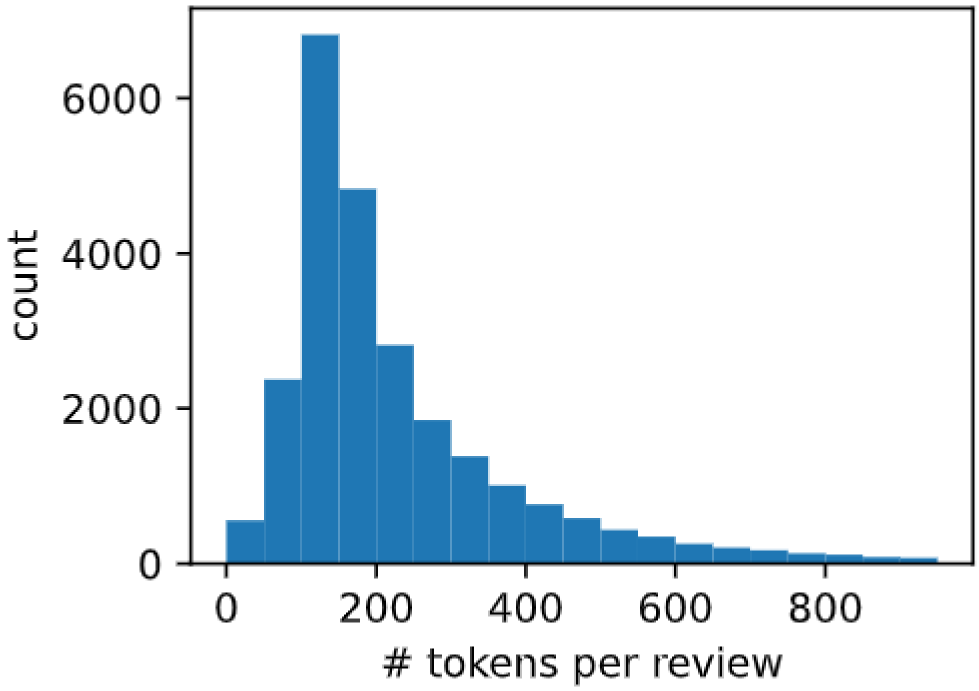
```
#@save
def read_imdb(data_dir, is_train):
    """读取IMDb评论数据集文本序列和标签"""
    data, labels = [], []
    for label in ('pos', 'neg'):
        folder_name = os.path.join(data_dir, 'train' if is_train else 'test',
                                   label)
        for file in os.listdir(folder_name):
            with open(os.path.join(folder_name, file), 'rb') as f:
                review = f.read().decode('utf-8').replace('\n', '')
                data.append(review)
                labels.append(1 if label == 'pos' else 0)
    return data, labels

# 读取训练集数据
train_data = read_imdb(data_dir, is_train=True)
print('训练集数目: ', len(train_data[0]))
for x, y in zip(train_data[0][:3], train_data[1][:3]):
    print('标签: ', y, 'review:', x[0:60])
```

3. 删除低频词

```
# 每个单词作为一个词元，过滤掉出现不到五次的单词
train_tokens = d2l.tokenize(train_data[0], token='word')
vocab = d2l.Vocab(train_tokens, min_freq=5, reserved_tokens=['<pad>'])
```

可视化词频

```
# 绘制词元长度的直方图
d2l.set_figsize()
d2l.plt.xlabel('# tokens per review')
d2l.plt.ylabel('count')
d2l.plt.hist([len(line) for line in train_tokens], bins=range(0, 1000, 50));
```



4. 将文本序列转化为固定长度的索引序列，并进行填充

```
# 整合代码
#@save
def load_data_imdb(batch_size, num_steps=500):
    """返回数据迭代器和IMDb评论数据集的词表"""
    data_dir = d2l.download_extract('aclImdb', 'aclImdb')
    train_data = read_imdb(data_dir, True)
    test_data = read_imdb(data_dir, False)
    train_tokens = d2l.tokenize(train_data[0], token='word')
    test_tokens = d2l.tokenize(test_data[0], token='word')
    vocab = d2l.Vocab(train_tokens, min_freq=5)
    train_features = torch.tensor([d2l.truncate_pad(
        vocab[line], num_steps, vocab['<pad>']) for line in train_tokens])
    test_features = torch.tensor([d2l.truncate_pad(
        vocab[line], num_steps, vocab['<pad>']) for line in test_tokens])
    train_iter = d2l.load_array((train_features, torch.tensor(train_data[1])),
                                batch_size)
    test_iter = d2l.load_array((test_features, torch.tensor(test_data[1])),
                               batch_size,
                               is_train=False)
    return train_iter, test_iter, vocab
```

模型定义

定义双向LSTM网络。

```
class BiRNN(nn.Module):
    def __init__(self, vocab_size, embed_size, num_hiddens,
                 num_layers, **kwargs):
        super(BiRNN, self).__init__(**kwargs)
```

```python
        # 嵌入层
        self.embedding = nn.Embedding(vocab_size, embed_size)
        # 双向LSTM层
        self.encoder = nn.LSTM(embed_size, num_hiddens, num_layers=num_layers,
                               bidirectional=True)
        # 全连接层
        self.decoder = nn.Linear(4 * num_hiddens, 2)

    def forward(self, inputs):
        # inputs的形状是（批量大小，时间步数）
        # 输出形状为（时间步数，批量大小，词向量维度）
        embeddings = self.embedding(inputs.T)
        self.encoder.flatten_parameters()
        # 返回上一个隐藏层在不同时间步的隐状态，
        # outputs的形状是（时间步数，批量大小，2*隐藏单元数）
        outputs, _ = self.encoder(embeddings)
        # 连结初始和最终时间步的隐状态，作为全连接层的输入，
        # 其形状为（批量大小，4*隐藏单元数）
        encoding = torch.cat((outputs[0], outputs[-1]), dim=1)
        outs = self.decoder(encoding)
        return outs
```

`BiRNN`类包含嵌入层、双向LSTM层和全连接层。

- 嵌入层将词索引转换为词向量。
- 双向LSTM层处理嵌入序列，输出最后一个时间步的隐藏状态。
- 全连接层将隐藏状态映射到分类结果。

模型初始化和参数设置

1. 初始化模型参数

```python
embed_size, num_hiddens, num_layers = 100, 100, 2
devices = d2l.try_all_gpus()
net = BiRNN(len(vocab), embed_size, num_hiddens, num_layers)

def init_weights(m):
    if type(m) == nn.Linear:
        nn.init.xavier_uniform_(m.weight)
    if type(m) == nn.LSTM:
        for param in m._flat_weights_names:
            if "weight" in param:
                nn.init.xavier_uniform_(m._parameters[param])
net.apply(init_weights)
```

- 设置嵌入维度、隐藏单元数和LSTM层数。
- 初始化模型参数，使用Xavier初始化方法。

2. 加载预训练的GloVe词向量

```
glove_embedding = d2l.TokenEmbedding('glove.6b.100d')
embeds = glove_embedding[vocab.idx_to_token]
net.embedding.weight.data.copy_(embeds)
net.embedding.weight.requires_grad = False
```

- 加载预训练的GloVe词向量，并将其复制到模型的嵌入层中。
- 冻结嵌入层的权重，不参与训练。

模型训练

1. 定义训练函数

```
# 用于在单个批量数据上训练模型
def train_batch_ch13(net, X, y, loss, trainer, devices):
    if isinstance(X, list):
        X=[x.to(devices[0]) for x in X]
    else:
        X=X.to(devices[0])
    y=y.to(devices[0])
    net.train()
    trainer.zero_grad()
    pred=net(X)
    l=loss(pred,y)
    l.sum().backward()
    trainer.step()
    train_loss_sum=l.sum()
    train_acc_sum=d2l.accuracy(pred,y)
    return train_loss_sum, train_acc_sum

# 整个训练过程
def train_ch13(net, train_iter, test_iter, loss, trainer, num_epoch,
devices=d2l.try_all_gpus()):
    timer, num_batches=d2l.Timer(), len(train_iter)
    animator=d2l.Animator(xlabel='epoch',xlim=[1,num_epoch], ylim=
[0,1],legend=['train loss','train acc','test acc'])
    net=nn.DataParallel(net,device_ids=devices).to(devices[0])
    for epoch in range(num_epoch):
        metric=d2l.Accumulator(4)
        for i,(features, labels) in enumerate(train_iter):
            timer.start()
            l,
acc=train_batch_ch13(net,features,labels,loss,trainer,devices)
            metric.add(l,acc,labels.shape[0],labels.numel())
            timer.stop()
            if (i + 1) % (num_batches // 5) == 0 or i == num_batches -
1:
                animator.add(epoch + (i + 1) / num_batches, (metric[0]
/ metric[2], metric[1] / metric[3], None))
        test_acc = d2l.evaluate_accuracy_gpu(net, test_iter)
        animator.add(epoch + 1, (None, None, test_acc))
```

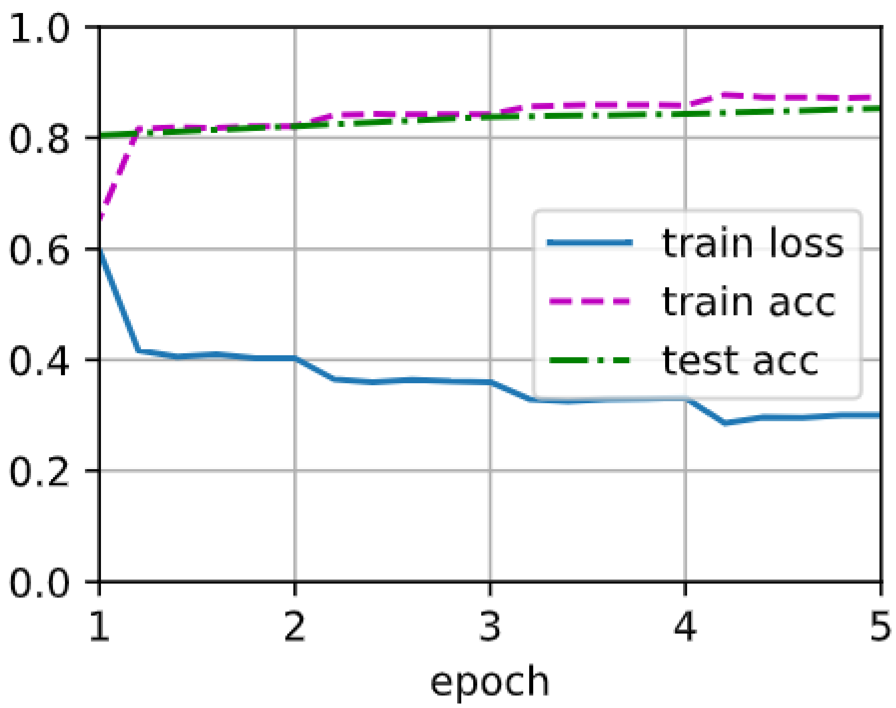
```
    print(f'loss {metric[0] / metric[2]:.3f}, train acc {metric[1] /
metric[3]:.3f}, test acc {test_acc:.3f}')
    print(f'{metric[2] * num_epochs / timer.sum():.1f} examples/sec on
{str(devices)}')
```

开始训练

```
lr, num_epochs = 0.01, 5
trainer = torch.optim.Adam(net.parameters(), lr=lr)
loss = nn.CrossEntropyLoss(reduction="none")
train_ch13(net, train_iter, test_iter, loss, trainer, num_epochs,
devices)
```

- 设置学习率和训练epoch数。
- 使用Adam优化器和交叉熵损失函数。

模型训练过程为:



预测结果为:

```
predict_sentiment(net, vocab, 'this movie is so great')
```

: 'positive'

```
predict_sentiment(net, vocab, 'this movie is so bad')
```

: 'negtive'