获取自然语言推断数据集

下载数据集

```
import os
import re
from torch.utils.data import Dataset
from torch import nn
from d2l import torch as d2l

# 下载SNLI数据集

#@save
d2l.DATA_HUB['SNLI'] = (
    'https://nlp.stanford.edu/projects/snli/snli_1.0.zip',
    '9fcde07509c7e87ec61c640c1b2753d9041758e4')

# data_dir = d2l.download_extract('SNLI')
# 手动解压数据集
data_dir='..\data\snli_1.0'
```

读取数据集

```
# 将数据集分解为前提，假设，标签
def read_snli(data_dir, is_train):
    # 将snli数据集解析为前提 假设和标签
    def extract_text(s):
        # 删除不会使用的信息
        s=re.sub('\\(','',s)
        s=re.sub('\\)','',s)
        # 用一个空格替换两个或连续的空格
        s=re.sub('\\s{2,}',' ',s)
        return s.strip()
    label_set={'entailment':0,'contradiction':1,'neutral':2}
    file_name=os.path.join(data_dir,'snli_1.0_train.txt'
                           if is_train else 'snli_1.0_test.txt')
    with open(file_name,'r') as file:
        rows=[row.split('\t') for row in file.readlines()[1:]]
    premises=[extract_text(row[1]) for row in rows if row[0] in label_set]
    hypotheses=[extract_text(row[2]) for row in rows if row[0] in label_set]
    labels=[label_set[row[0]] for row in rows if row[0] in label_set]
    return premises, hypotheses, labels
```

注意

使用多层文本感知机将文本序列中的词元对齐。

```
def mlp(num_inputs, num_hiddens, flatten):
    net=[]
    net.append(nn.Dropout(0.2))
    net.append(nn.Linear(num_inputs, num_hiddens))
    net.append(nn.ReLU())
    if flatten:
        net.append(nn.Flatten(start_dim=1))
    net.append(nn.Dropout(0.2))
    net.append(nn.Linear(num_hiddens, num_hiddens))
    net.append(nn.ReLU())
    if flatten:
        net.append(nn.Flatten(start_dim=1))
    return nn.Sequential(*net)
```

计算假设中所有词元向量的加权平均值。

```
class Attend(nn.Module):
    def __init__(self, num_inputs, num_hiddens, **kwargs):
        super(Attend, self).__init__(**kwargs)
        self.f=mlp(num_inputs, num_hiddens, flatten=False)
    def forward(self, A, B):
        f_A=self.f(A)
        f_B=self.f(B)
        e=torch.bmm(f_A,f_B.permute(0,2,1))
        beta=torch.bmm(F.softmax(e,dim=-1),B)
        alpha=torch.bmm(F.softmax(e.permute(0,2,1), dim=-1),A)
        return beta, alpha
```

比较

将一个序列中的词元与与该词元软对齐的另一个序列进行比较。

```
class Compare(nn.Module):
    def __init__(self, num_inputs, num_hiddens, **kwargs):
        super(Compare, self).__init__(**kwargs)
        self.g = mlp(num_inputs, num_hiddens, flatten=False)

    def forward(self, A, B, beta, alpha):
        V_A = self.g(torch.cat([A, beta], dim=2))
        V_B = self.g(torch.cat([B, alpha], dim=2))
        return V_A, V_B
```

聚合

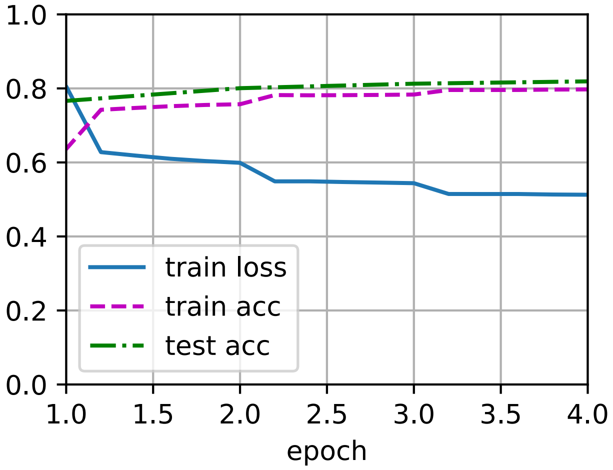
聚合多组求和向量。

```
class Aggregate(nn.Module):
    def __init__(self, num_inputs, num_hiddens, num_outputs, **kwargs):
        super(Aggregate, self).__init__(**kwargs)
        self.h = mlp(num_inputs, num_hiddens, flatten=True)
        self.linear = nn.Linear(num_hiddens, num_outputs)

    def forward(self, V_A, V_B):
        # 对两组比较向量分别求和
        V_A = V_A.sum(dim=1)
        V_B = V_B.sum(dim=1)
        # 将两个求和结果的连结送到多层感知机中
        Y_hat = self.linear(self.h(torch.cat([V_A, V_B], dim=1)))
        return Y_hat
```

训练和评估模型

训练并评估注意力模型，并绘制结果图。


epoch
train loss
train acc
test acc