实验环境：torchtext=0.6.0，torch=2.0.1，transformers=4.30.2

文本处理(in txthd.py)

读取评论与情感评级

```python
path=["../../aclImdb/train/neg","../../aclImdb/train/pos","../../aclImdb/test/neg","../../aclImdb/test/pos"]

txts=[[],[],[],[]]
score=[[],[],[],[]]

for i in range(4):
# 遍历指定目录下所有文件
    files=os.listdir(path[i])
    for file in files:
        file_path=path[i]+'\\'+file
        # print(file_path)
        with open(file_path,'r',encoding='utf-8') as f:
            data=f.read()
            txts[i].append(data)
            score[i].append(int(file[-5]))
```

对评论去分词(+->单词列表)

- 英文比中文好去多了，中文去完标点，还要用jieba分词，

```python
txt_list=[]
str_list=[]
for line in total_txts:
#可能还有
    latstr=re.sub("[\s+\.\!\/_,$%^*()+\"\']+|[+——！，。？、~@#￥%……&*（）]+", " ",line)
    word_list=latstr.split(' ')
    txt_list.append(word_list)
    str_list.append(' '.join(word_list))
```

- 这里txt_list的每一项是单词列表，对应于一个评论，用于lstm
- str_list的每一项对应去掉分词的评论，用于bert

字符串修剪

找出按正态分布涵盖95%的评论的长度

```python
len_list=[]
for word_list in txt_list:
    len_list.append(len(word_list))
len_list=np.array(len_list)
max_len=np.mean(len_list)+2*np.std(len_list)
print(int(max_len))
```

修剪+前面填充

- 忘记哪里说，在较短评论的前面填充space比在后面填充space更合理

```
pruned_txt_list=[]
for word_list in txt_list:
    if(len(word_list)<max_len):
        word_list=[' ']*(int(max_len)-len(word_list))+word_list
        pruned_txt_list.append(word_list)
    elif len(word_list)>max_len:
        pruned_txt_list.append(word_list[:int(max_len)])

print(len(pruned_txt_list))
print(pruned_txt_list[0])
print(pruned_txt_list[1])
```

单词向量化嵌入(by glove)

- 利用下载的glove.6B.100d.txt，构建完成预训练的单词-向量表，
- 并导出字典：单词->索引，索引->单词，单词->向量，

```
##### 字符列表-->字符向量
from torchtext.vocab import GloVe,Vectors
from torchtext import data


TEXT = data.Field(sequential=True,use_vocab=True)
vectors=Vectors(name="../../src/glove.6B.100d.txt")
TEXT.build_vocab(pruned_txt_list, vectors=vectors)

##### 导出
with open("../pkl/stoi.pkl","wb") as f:
    pickle.dump(TEXT.vocab.stoi,f)
    print(type(TEXT.vocab.stoi))
with open("../pkl/itos.pkl","wb") as f:
    pickle.dump(TEXT.vocab.itos,f)
    print(type(TEXT.vocab.itos))
with open("../pkl/vectors.pkl","wb") as f:
    pickle.dump(TEXT.vocab.vectors,f)
    print(type(TEXT.vocab.vectors))
```

LSTM(in lstm.py)

lstm原理

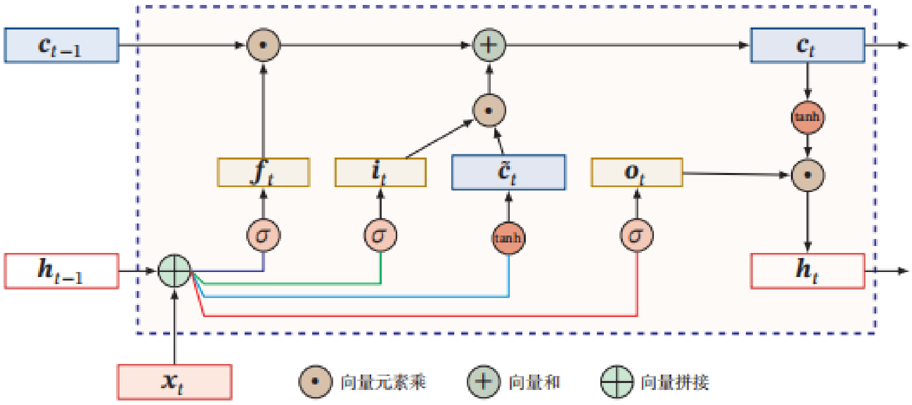


图 6.7　LSTM 网络的循环单元结构

$$
\begin{aligned}
&W_i \in R^{d\times v}, x_t \in R^v, b_t \in R^d; v = \text{向量长度，} d = \text{隐藏层长度} \\
&h \in R^d, W_h \in R^{d\times d} \\
&g_t = tanh(W_{ig}x_t + b_{ig} + W_{hg}h_{t-1} + b_{hg}) \in R^{d\times 1} \\
&i_t = \sigma(W_{ii}x_t + b_{ii} + W_{hi}h_{t-1} + b_{hi}) \in R^{d\times 1} \\
&f_t = \sigma(W_{if}x_t + b_{if} + W_{hf}h_{t-1} + b_{hf}) \in R^{d\times 1} \\
&o_t = \sigma(W_{io}x_t + b_{io} + W_{ho}h_{t-1} + b_{ho}) \in R^{d\times 1} \\
&c_t = f_t \odot c_{t-1} + i_t \odot g_t \in R^{d\times 1} \\
&h_t = o_t \odot tanh(c_t)
\end{aligned}
\tag{1}
$$

在对文本的处理过程中，以评论长度(固定为max_len=592)为总时间，

第i周期，self.hidd，self.cell，input[i]分别作为ht-1，ct-1(历史单元)，xt，

其中ht-1和xt产生内门gt，it，ft，ot；ct-1+gt再利用遗忘门ft和输入门it产生ct，

最后ct和输出门产生外门ht.

手写多层LSTM

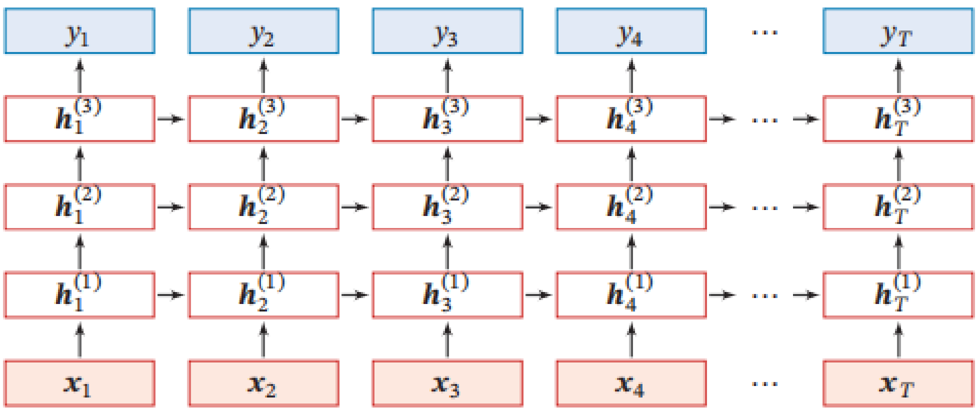


图 6.9　按时间展开的堆叠循环神经网络

LSTM传递过程代码：

```
        output=torch.zeros(input.size(0),input.size(1),self.hidden_size).to(device)
        for i in range(len(input)):
            x_t=input[i]
            x_t=x_t.unsqueeze(-1)       # x=(batch_size,input_size,1)
            # print(f'device2:{self.w_ig.device,x_t.device}')
            g_t=self.tanh(torch.bmm(self.w_ig,x_t)+torch.bmm(self.w_hg,self.hidd)+self.b_g)      # output (batch_size,hidden_size,1)
            i_t=self.sigmoid(torch.bmm(self.w_ii,x_t)+torch.bmm(self.w_hi,self.hidd)+self.b_i)
            f_t=self.sigmoid(torch.bmm(self.w_if,x_t)+torch.bmm(self.w_hf,self.hidd)+self.b_f)
            o_t=self.sigmoid(torch.bmm(self.w_io,x_t)+torch.bmm(self.w_ho,self.hidd)+self.b_o)
            self.cell=torch.mul(f_t,self.cell)+torch.mul(i_t,g_t)    # output (batch_size,hidden_size,1)
            self.hidd=torch.mul(o_t,self.tanh(self.cell))
            # print(f'hidd_size={self.hidd.shape}')
            output[i]=self.hidd.squeeze(-1)  #(batch_size,output_size)
```

如果考虑是多层LSTM，每一层的输出还要作为下一次的输入再进行一次LSTM传播；

第一层：（seq_len，batch_size，input_size)->(seq_len，batch_size，hidden_size)

第2-n_layer-1层：(seq_len，batch_size，hidden_size)->(seq_len，batch_size，hidden_size)

第n_layer层：(seq_len，batch_size，hidden_size)->(seq_len，batch_size，output_size)

- 注意，每一层开头的隐藏层输入h0都要重置

```
        if(self.n_layer>=3):
            for i in range(self.n_layer-2):
                output=self.hidd_forward(output)
        self.cell=self.init_cell().to(device)   #重置内外门
        self.hidd=self.init_hidd().to(device)
```

- 在最后一层，得到h1~hn，通过分类层得到y1~yn，
- 我们只取最后一层作为情感的表示，y1~yn-1没必要计算
- 由hn得到yn的代码如下：

```
            if i==len(input)-1:
                target=self.classfier(self.hidd.squeeze(-1))    # (batch_size,output_size)
                target=self.logsoftmax(target)
```

PS1：双向LSTM实现类似，在尾部设置hn+1,cn+1即可，

PS2：数据集定义、加载，对结果作损失计算正确性，与前2次类似

双向LSTM(单层)调库实现

- 对单向LSTM，作如下更改即可：

  nn.LSTM的bidirectional=False，init_hidd和init_cell返回参数第一个维度均为self.n_layers，self.layer的第一个线性层第一维=self.hidden_size

```
class LSTM_last_ele(nn.Module):
    def __init__(self,input_size,hidden_size,n_layers,batch_size):
        super(LSTM_last_ele, self).__init__()
        self.n_layers = n_layers
        self.hidden_size = hidden_size
        self.batch_size=batch_size
        self.lstm = nn.LSTM(input_size, hidden_size, n_layers, bidirectional=True, batch_first=False)
        self.layer = nn.Sequential(nn.Linear(2*self.hidden_size, 32),
                                   nn.Tanh(),nn.Linear(32,output_size),
                                   nn.Softmax(dim=-1))
        self.hidd=self.init_hidd()
        self.cell=self.init_cell()

    def init_hidd(self):
        return torch.zeros(self.n_layers*2,self.batch_size,self.hidden_size).to(device)

    def init_cell(self):
        return torch.zeros(self.n_layers*2,self.batch_size,self.hidden_size).to(device)


    def forward(self,input):
        output, (self.hidd,self.cell) = self.lstm(input, (self.hidd, self.cell))    # 592,64,100->592,64,128
        output = self.layer(output[-1, :, :])          #592,64,128->64,128, 只取最后一个元素,64,128->164,10
        return output
```

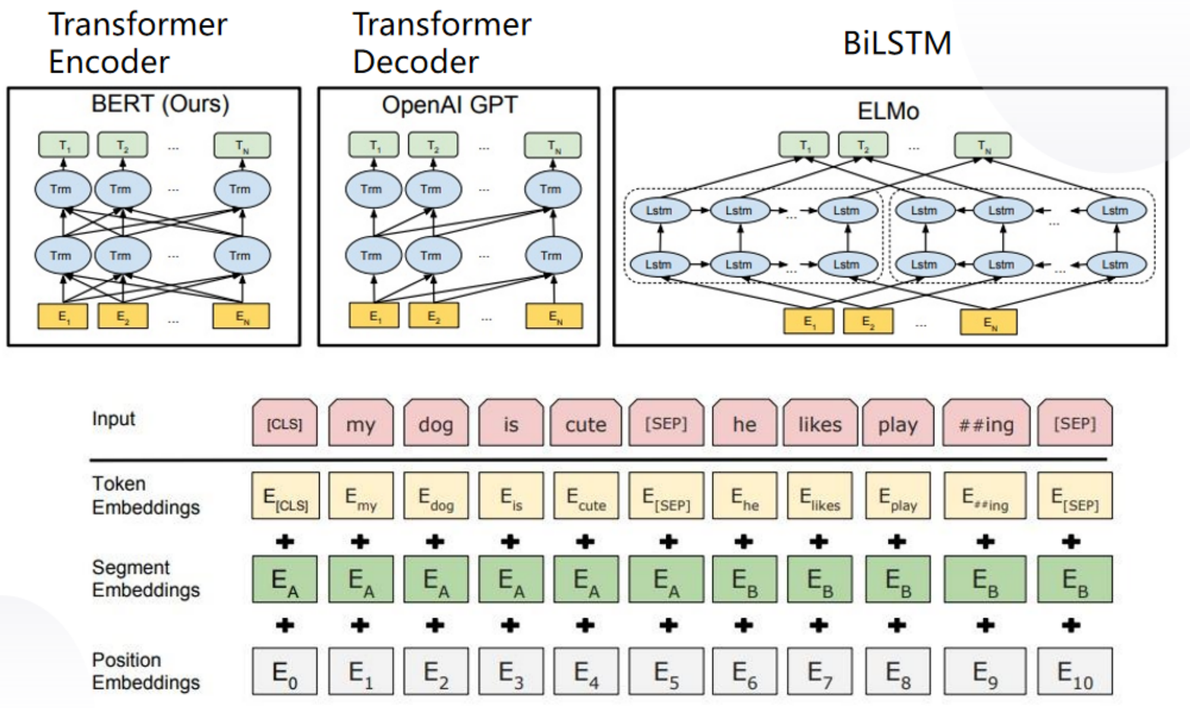
Bert(in batch_bert.py)

原理

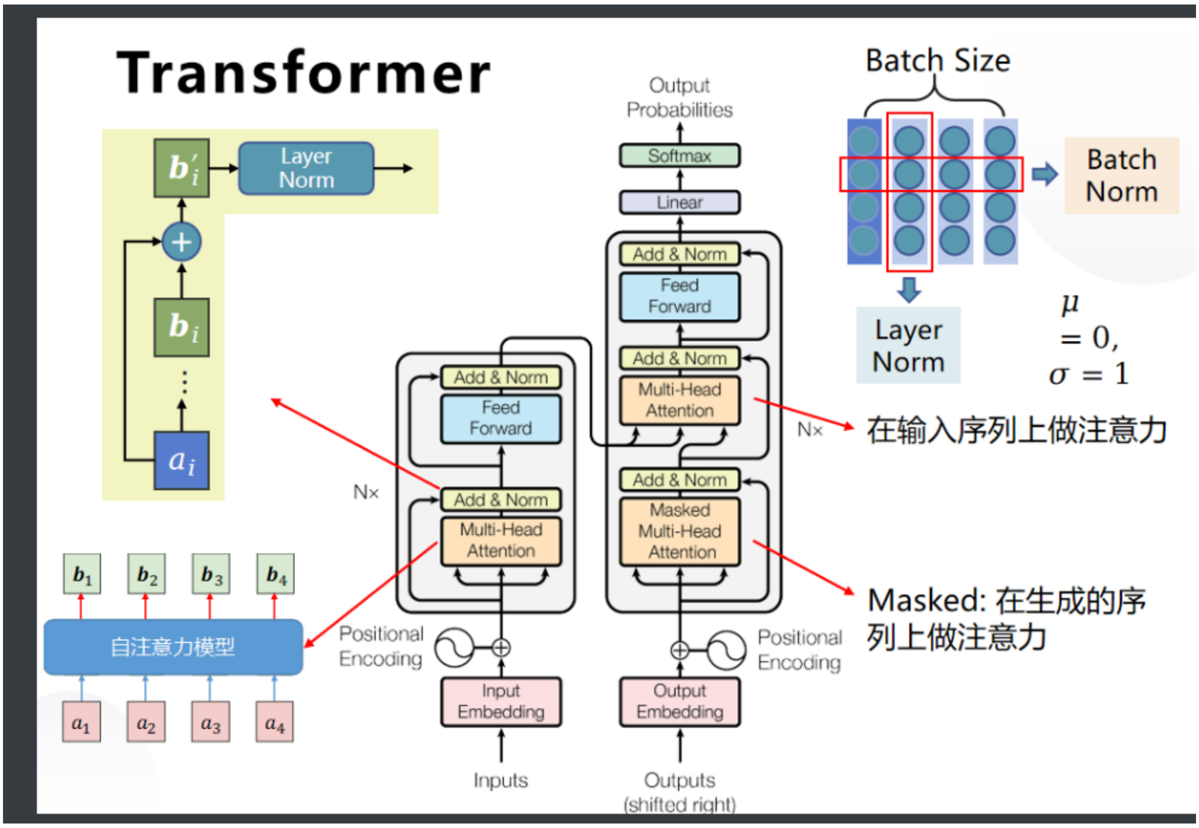
Bidirectional Encoder Representations from Transformers (BERT)=双向编码器-转换器表示，先把单词向量化(token embedding)，再用position embedding+masked embedding，

本实验的实现中没用到segment embedding（从属于那个句子，分段信息向量）

预训练模型

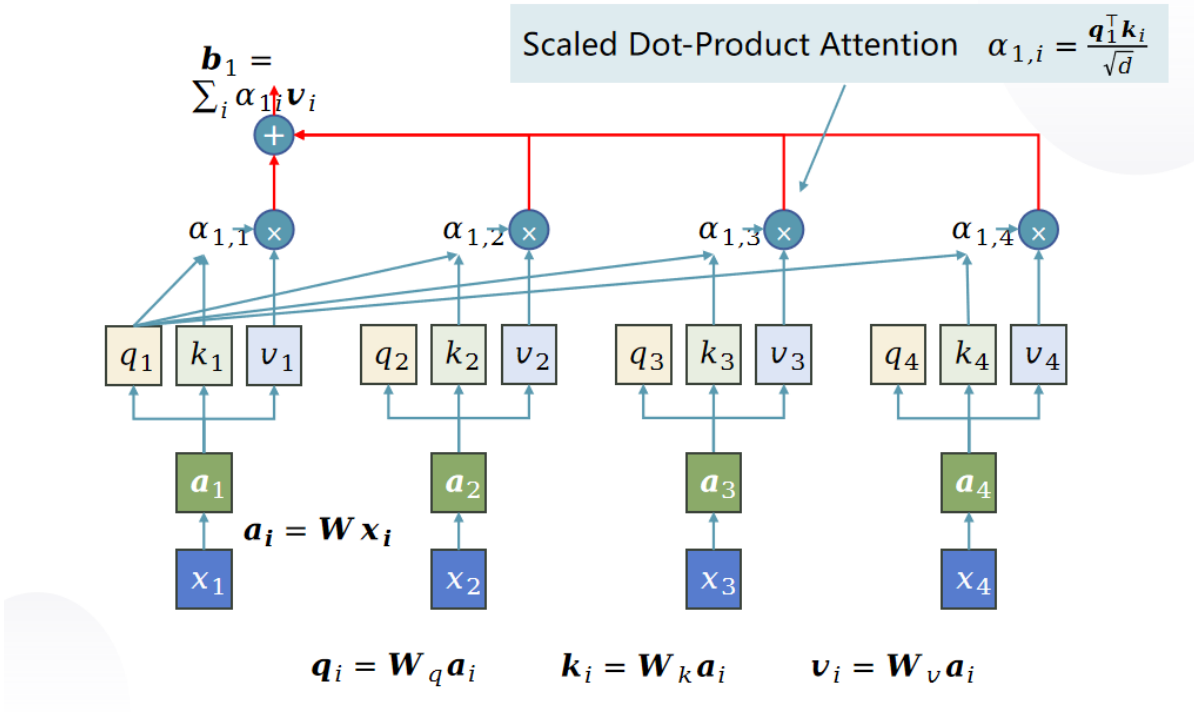


从E1~EN得到T1~TN用编码器实现，



而编码器的基本单元是注意力机制

自注意力模型



实现

bert文本处理(in txthd.py)

bert的输入不是单词列表，而是字符串列表，因此需要利用去标点的原单词列表重新建一个bert_txt_list

```
max_len2=257
bert_txt_list=[]
for word_list in txt_list:
    if(len(word_list)<max_len2):
        bert_txt_list.append(word_list)
    elif len(word_list)>max_len2:
        bert_txt_list.append(word_list[:int(max_len2)])
pdb.set_trace()
```

max_len=257是因为后面的bert输入长度不能超过512，虽然后续好像可以用tokenizer自带的裁剪保证~~

构造数据集中tokenizer.encode_plus对评论编码

```
class BertDataSet(data.Dataset):
    def __init__(self,texts,tokenizer,max_len):
        self.tokenizer = tokenizer
        self.comment_text = texts
        self.targets = train_score_label
        self.max_len=max_len

    def __len__(self):
        return len(self.comment_text)

    def __getitem__(self, index):
        comment_text =' '.join(self.comment_text[index]).lower()     #小写
        # print(comment_text)
```

```
        inputs = self.tokenizer.encode_plus(
            comment_text,
            None,
            add_special_tokens=True,
            max_length=self.max_len,
            padding='max_length',
            return_token_type_ids=True,
            truncation=True
        )
        ids = inputs['input_ids']
        mask = inputs['attention_mask']
        token_type_ids = inputs["token_type_ids"]


        return {
            'ids': torch.tensor(ids, dtype=torch.long),
            'mask': torch.tensor(mask, dtype=torch.long),
            'token_type_ids': torch.tensor(token_type_ids,
dtype=torch.long),
            'targets': torch.tensor(self.targets[index], dtype=torch.float)
        }
```

Model=bert层+分类器

```
class Net(nn.Module):
    def __init__(self,input_size,hidden_size,output_size):
        super(Net,self).__init__()

        self.bert=BertModel.from_pretrained('bert-base-uncased')

        self.classifier=nn.Sequential(
            nn.Dropout(0.5),
            nn.Linear(input_size,hidden_size),
            nn.ReLU(),
            nn.Linear(hidden_size,output_size)
        )

    def forward(self,input_ids,token_type_ids,attention_mask):
        out=self.bert(input_ids,token_type_ids,attention_mask)
        out=out.last_hidden_state[:,-1,:]
        out=self.classifier(out)        #(batch_size,output_size)=(8,10)
        return out
```

拙劣的调参过程

LSTM-2分类

With Adam optim—1层nn.LSTM(双向)

吹爆torch.optim.adam()在情感分类中的优越性!

```
epoch:1;     loss:0.010703;   accu:0.534778
epoch:2;     loss:0.010666;   accu:0.572581
epoch:3;     loss:0.010928;   accu:0.528730
epoch:4;     loss:0.010943;   accu:0.506048
epoch:5;     loss:0.010783;   accu:0.516129
total=1984.0
Accuracy of the network on the test dataset: 51 %
epoch:6;     loss:0.010717;   accu:0.530746
epoch:7;     loss:0.010666;   accu:0.522177
epoch:8;     loss:0.010640;   accu:0.534778
epoch:9;     loss:0.010665;   accu:0.543347
epoch:10;     loss:0.010243;   accu:0.606351
total=1984.0
Accuracy of the network on the test dataset: 69 %
epoch:11;     loss:0.009084;   accu:0.725806
epoch:12;     loss:0.008301;   accu:0.775706
epoch:13;     loss:0.008408;   accu:0.767641
epoch:14;     loss:0.007722;   accu:0.817540
epoch:15;     loss:0.007906;   accu:0.798891
total=1984.0
Accuracy of the network on the test dataset: 81 %
epoch:16;     loss:0.007806;   accu:0.808468
epoch:17;     loss:0.007289;   accu:0.843750
epoch:18;     loss:0.006993;   accu:0.860887
epoch:19;     loss:0.007009;   accu:0.860887
epoch:20;     loss:0.007039;   accu:0.857359
total=1984.0
Accuracy of the network on the test dataset: 80 %
epoch:21;     loss:0.006731;   accu:0.879032
epoch:22;     loss:0.006616;   accu:0.887097
epoch:23;     loss:0.006508;   accu:0.894153
epoch:24;     loss:0.006602;   accu:0.888105
epoch:25;     loss:0.006432;   accu:0.900202
total=1984.0
Accuracy of the network on the test dataset: 80 %
epoch:26;     loss:0.006274;   accu:0.912298
epoch:27;     loss:0.006221;   accu:0.912802
epoch:28;     loss:0.006163;   accu:0.917843
epoch:29;     loss:0.006027;   accu:0.926915
epoch:30;     loss:0.006250;   accu:0.911794
total=1984.0
Accuracy of the network on the test dataset: 80 %
epoch:31;     loss:0.006089;   accu:0.923387
```

handle overfitting

过拟合，加个dropout(0.5)层

```
total=1984.0
Accuracy of the network on the test dataset: 74 %
epoch:16;    loss:0.006818;  accu:0.874496
epoch:17;    loss:0.007321;  accu:0.839214
epoch:18;    loss:0.006457;  accu:0.898185
epoch:19;    loss:0.006217;  accu:0.914315
epoch:20;    loss:0.006125;  accu:0.919859
total=1984.0
Accuracy of the network on the test dataset: 79 %
epoch:21;    loss:0.006120;  accu:0.920867
epoch:22;    loss:0.006076;  accu:0.922883
epoch:23;    loss:0.005861;  accu:0.936996
epoch:24;    loss:0.006058;  accu:0.925403
epoch:25;    loss:0.005986;  accu:0.930444
total=1984.0
Accuracy of the network on the test dataset: 77 %
epoch:26;    loss:0.006086;  accu:0.923891
epoch:27;    loss:0.006357;  accu:0.904234
epoch:28;    loss:0.006117;  accu:0.921371
epoch:29;    loss:0.006023;  accu:0.927419
epoch:30;    loss:0.006164;  accu:0.917843
total=1984.0
Accuracy of the network on the test dataset: 76 %
epoch:31;    loss:0.006090;  accu:0.921875
epoch:32;    loss:0.006111;  accu:0.919859
epoch:33;    loss:0.005978;  accu:0.929940
epoch:34;    loss:0.006096;  accu:0.921875
epoch:35;    loss:0.006574;  accu:0.889617
total=1984.0
Accuracy of the network on the test dataset: 75 %
epoch:36;    loss:0.006234;  accu:0.911794
epoch:37;    loss:0.006085;  accu:0.921371
epoch:38;    loss:0.005829;  accu:0.939012
epoch:39;    loss:0.005754;  accu:0.944052
epoch:40;    loss:0.005630;  accu:0.952621
total=1984.0
Accuracy of the network on the test dataset: 75 %
epoch:41;    loss:0.005617;  accu:0.953125
epoch:42;    loss:0.005614;  accu:0.953629
epoch:43;    loss:0.005673;  accu:0.950101
epoch:44;    loss:0.005713;  accu:0.947077
epoch:45;    loss:0.005627;  accu:0.952621
total=1984.0
Accuracy of the network on the test dataset: 78 %
epoch:46;    loss:0.005612;  accu:0.954133
epoch:47;    loss:0.005682;  accu:0.949597
epoch:48;    loss:0.005616;  accu:0.953125
epoch:49;    loss:0.005534;  accu:0.958165
epoch:50;    loss:0.005562;  accu:0.955645
```

```
total=1984.0
Accuracy of the network on the test dataset: 75 %
```

还是过拟合，添加l2正则化项，

```python
l2_regularization = sum(torch.sum(torch.pow(param, 2)) for param in net.parameters())
...
running_loss+=(criterion(out,label)+l2_lambda*l2_regularization).item()
loss=criterion(out,label)+l2_lambda*l2_regularization
```

l2_lambda=0.01和0.001没有效果

```
tf-docker ~/work/zode/src > python lstm.py
score's lenth=12500
2000
2000
Could not load symbol cublasGetSmCountTarget from libcub
epoch:1;    loss:0.023152;  accu:0.501008
epoch:2;    loss:0.011231;  accu:0.503528
epoch:3;    loss:0.010850;  accu:0.498488
epoch:4;    loss:0.010833;  accu:0.498992
epoch:5;    loss:0.010832;  accu:0.486895
total=1984.0
Accuracy of the network on the test dataset: 50 %
epoch:6;    loss:0.010833;  accu:0.464214
```

```
tf-docker ~/work/zode/src > python lstm.py
score's lenth=12500
2000
2000
Could not load symbol cublasGetSmCountTarget from libcu
epoch:1;    loss:0.012203;  accu:0.488407
epoch:2;    loss:0.010956;  accu:0.517137
epoch:3;    loss:0.010897;  accu:0.505544
epoch:4;    loss:0.010858;  accu:0.489415
epoch:5;    loss:0.010850;  accu:0.491431
total=1984.0
Accuracy of the network on the test dataset: 50 %
```

遂改成0.0001，貌似效果好点了，

```
Accuracy of the network on the test dataset: 82 %
epoch:31;     loss:0.006608;   accu:0.932964
epoch:32;     loss:0.006546;   accu:0.933972
epoch:33;     loss:0.006723;   accu:0.924395
epoch:34;     loss:0.006523;   accu:0.938004
epoch:35;     loss:0.006635;   accu:0.932964
total=1984.0
Accuracy of the network on the test dataset: 82 %
epoch:36;     loss:0.007044;   accu:0.910786
epoch:37;     loss:0.006747;   accu:0.931956
epoch:38;     loss:0.006551;   accu:0.940524
epoch:39;     loss:0.006594;   accu:0.939516
epoch:40;     loss:0.006711;   accu:0.933972
total=1984.0
Accuracy of the network on the test dataset: 82 %
epoch:41;     loss:0.006588;   accu:0.938004
epoch:42;     loss:0.006682;   accu:0.932460
epoch:43;     loss:0.006753;   accu:0.929435
epoch:44;     loss:0.006548;   accu:0.944052
epoch:45;     loss:0.006386;   accu:0.951109
total=1984.0
Accuracy of the network on the test dataset: 82 %
epoch:46;     loss:0.006312;   accu:0.951109
epoch:47;     loss:0.006273;   accu:0.952621
epoch:48;     loss:0.006508;   accu:0.939012
epoch:49;     loss:0.007048;   accu:0.910786
epoch:50;     loss:0.006824;   accu:0.927923
total=1984.0
Accuracy of the network on the test dataset: 81 %
tf-docker ~/work/zode/src >
```

再改改，l2_lambda=2e-4，好一点点，就到这了

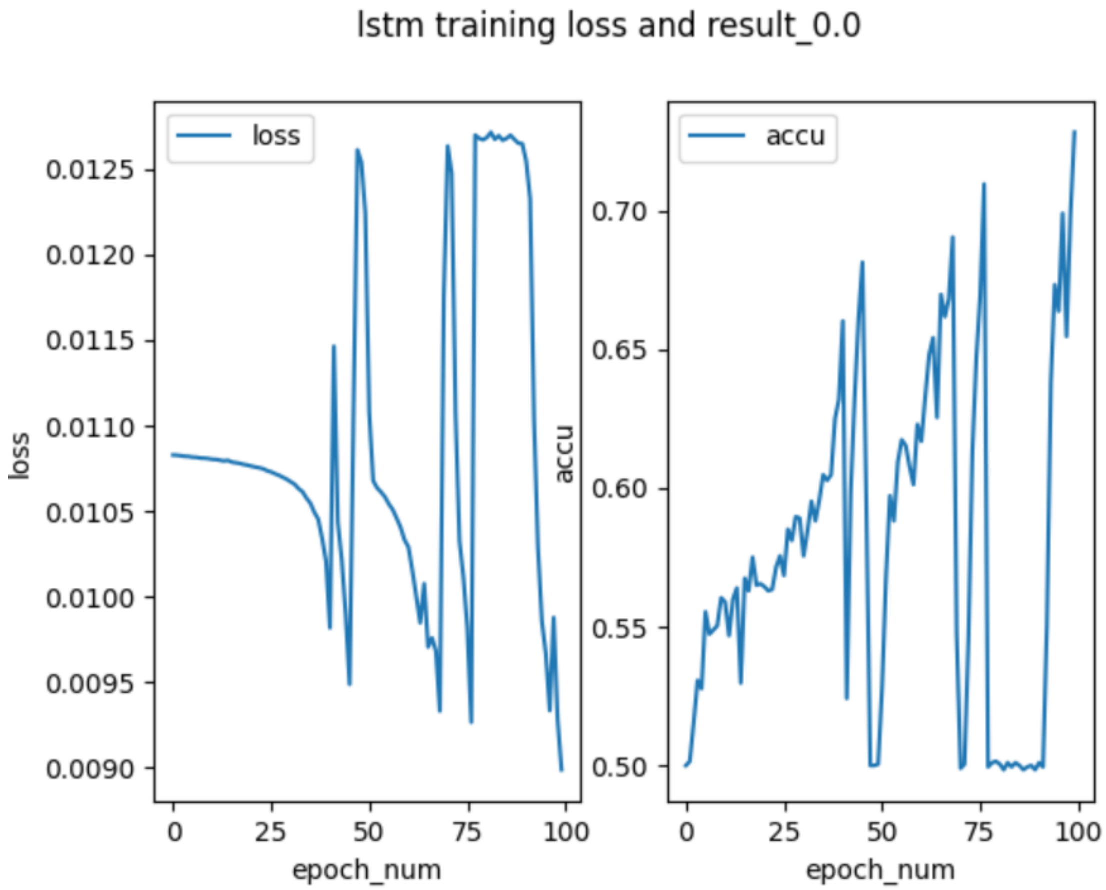
```
Accuracy of the network on the test dataset: 83 %
epoch:26;     loss:0.007384;  accu:0.886089
epoch:27;     loss:0.007471;  accu:0.879032
epoch:28;     loss:0.007591;  accu:0.876008
epoch:29;     loss:0.007428;  accu:0.883569
epoch:30;     loss:0.007299;  accu:0.893145
total=1984.0
Accuracy of the network on the test dataset: 82 %
epoch:31;     loss:0.007188;  accu:0.899698
epoch:32;     loss:0.007193;  accu:0.894657
epoch:33;     loss:0.007541;  accu:0.873992
epoch:34;     loss:0.007401;  accu:0.892641
epoch:35;     loss:0.007242;  accu:0.898185
total=1984.0
Accuracy of the network on the test dataset: 81 %
epoch:36;     loss:0.007406;  accu:0.883569
epoch:37;     loss:0.007167;  accu:0.901714
epoch:38;     loss:0.007244;  accu:0.894153
epoch:39;     loss:0.007413;  accu:0.886593
epoch:40;     loss:0.007301;  accu:0.899698
total=1984.0
Accuracy of the network on the test dataset: 83 %
epoch:41;     loss:0.007075;  accu:0.908770
epoch:42;     loss:0.006842;  accu:0.917843
epoch:43;     loss:0.006926;  accu:0.912298
epoch:44;     loss:0.007453;  accu:0.884577
epoch:45;     loss:0.007253;  accu:0.909274
total=1984.0
Accuracy of the network on the test dataset: 83 %
epoch:46;     loss:0.007060;  accu:0.908770
epoch:47;     loss:0.006972;  accu:0.909778
epoch:48;     loss:0.006837;  accu:0.916331
epoch:49;     loss:0.006789;  accu:0.922379
epoch:50;     loss:0.006774;  accu:0.924395
total=1984.0
Accuracy of the network on the test dataset: 82 %
tf-docker ~/work/zode/src >
```

With SGD——

下面是使用SGD优化器的辛酸历程，可不看，因为最后的test是用前面的双向nn.LSTM的

momentum=0.9，且采用学习率减半==

经测试，使用SGD作优化器，很容易出现下面的景象——炼着炼着陡然回到解放前，



虽然随着时间的推移，从解放前收敛的速度更快了；

---

lstm调库，单向，1层

- 学习率=0.1，20轮*0.5，然后18轮回到解放前，所以学习率不能设置太大！！！

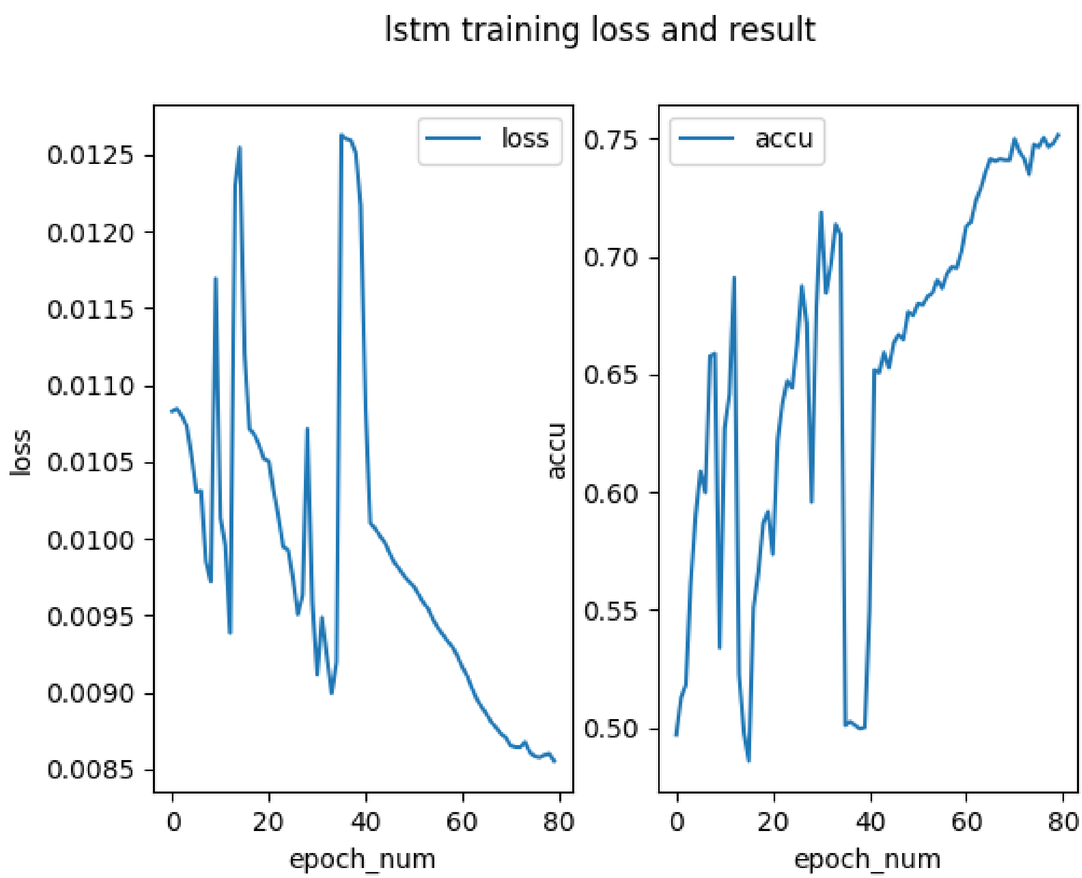
```
tf-docker ~/zode/src > python lstm_pkg.py
score's lenth=12500
2000
2000
Could not load symbol cublasGetSmCountTarget from libcublas.so.
cublasGetSmCountTarget
epoch:1;     loss:0.010865;   accu:0.481
epoch:2;     loss:0.010813;   accu:0.532
epoch:3;     loss:0.010783;   accu:0.536
epoch:4;     loss:0.010755;   accu:0.528
epoch:5;     loss:0.010639;   accu:0.580
total=1984.0
Accuracy of the network on the test dataset: 54 %
epoch:6;     loss:0.010751;   accu:0.543
epoch:7;     loss:0.010481;   accu:0.591
epoch:8;     loss:0.010165;   accu:0.635
epoch:9;     loss:0.010880;   accu:0.589
epoch:10;     loss:0.010520;   accu:0.579
total=1984.0
Accuracy of the network on the test dataset: 53 %
epoch:11;     loss:0.010271;   accu:0.608
epoch:12;     loss:0.009841;   accu:0.649
epoch:13;     loss:0.009940;   accu:0.642
epoch:14;     loss:0.009823;   accu:0.654
epoch:15;     loss:0.009498;   accu:0.682
total=1984.0
Accuracy of the network on the test dataset: 63 %
epoch:16;     loss:0.009460;   accu:0.682
epoch:17;     loss:0.009311;   accu:0.699
epoch:18;     loss:0.012617;   accu:0.505
epoch:19;     loss:0.012691;   accu:0.499
epoch:20;     loss:0.012452;   accu:0.500
```

- 学习率=0.1，5轮*0.5，

```
AttributeError: 'LSTM_last_ele' object has no attribute 'hidden_siz
tf-docker ~/zode/src > python lstm_pkg.py
score's lenth=12500
2000
2000
Could not load symbol cublasGetSmCountTarget from libcublas.so.11. 
cublasGetSmCountTarget
epoch:1;     loss:0.010836;  accu:0.492
epoch:2;     loss:0.010822;  accu:0.523
epoch:3;     loss:0.010788;  accu:0.543
epoch:4;     loss:0.010747;  accu:0.553
epoch:5;     loss:0.010641;  accu:0.574
total=1984.0
Accuracy of the network on the test dataset: 56 %
epoch:6;     loss:0.010521;  accu:0.593
epoch:7;     loss:0.010193;  accu:0.636
epoch:8;     loss:0.010222;  accu:0.609
epoch:9;     loss:0.010329;  accu:0.596
epoch:10;     loss:0.009969;  accu:0.656
total=1984.0
Accuracy of the network on the test dataset: 62 %
epoch:11;     loss:0.009740;  accu:0.668
epoch:12;     loss:0.009858;  accu:0.653
epoch:13;     loss:0.009807;  accu:0.650
epoch:14;     loss:0.009800;  accu:0.656
epoch:15;     loss:0.010059;  accu:0.635
total=1984.0
Accuracy of the network on the test dataset: 65 %
epoch:16;     loss:0.009362;  accu:0.701
epoch:17;     loss:0.009188;  accu:0.705
epoch:18;     loss:0.009695;  accu:0.675
epoch:19;     loss:0.009328;  accu:0.695
epoch:20;     loss:0.008861;  accu:0.729
total=1984.0
Accuracy of the network on the test dataset: 72 %
epoch:21;     loss:0.009374;  accu:0.688
epoch:22;     loss:0.009225;  accu:0.699
epoch:23;     loss:0.008889;  accu:0.727
epoch:24;     loss:0.008758;  accu:0.735
epoch:25;     loss:0.008940;  accu:0.720
total=1984.0
Accuracy of the network on the test dataset: 75 %
epoch:26;     loss:0.008589;  accu:0.747
epoch:27;     loss:0.008772;  accu:0.738
epoch:28;     loss:0.009125;  accu:0.703
epoch:29;     loss:0.008617;  accu:0.747
epoch:30;     loss:0.008462;  accu:0.759
total=1984.0
Accuracy of the network on the test dataset: 75 %
```

- 学习率=0.1,8轮减半，训80轮

```
Accuracy of the network on the test dataset: 63 %
epoch:51;     loss:0.009690;   accu:0.680
epoch:52;     loss:0.009637;   accu:0.679
epoch:53;     loss:0.009582;   accu:0.683
epoch:54;     loss:0.009543;   accu:0.684
epoch:55;     loss:0.009474;   accu:0.690
total=1984.0
Accuracy of the network on the test dataset: 65 %
epoch:56;     loss:0.009418;   accu:0.686
epoch:57;     loss:0.009376;   accu:0.693
epoch:58;     loss:0.009333;   accu:0.696
epoch:59;     loss:0.009295;   accu:0.695
epoch:60;     loss:0.009238;   accu:0.702
total=1984.0
Accuracy of the network on the test dataset: 67 %
epoch:61;     loss:0.009166;   accu:0.713
epoch:62;     loss:0.009109;   accu:0.715
epoch:63;     loss:0.009030;   accu:0.724
epoch:64;     loss:0.008958;   accu:0.729
epoch:65;     loss:0.008906;   accu:0.736
total=1984.0
Accuracy of the network on the test dataset: 70 %
epoch:66;     loss:0.008860;   accu:0.741
epoch:67;     loss:0.008808;   accu:0.740
epoch:68;     loss:0.008772;   accu:0.741
epoch:69;     loss:0.008731;   accu:0.741
epoch:70;     loss:0.008706;   accu:0.741
total=1984.0
Accuracy of the network on the test dataset: 72 %
epoch:71;     loss:0.008655;   accu:0.750
epoch:72;     loss:0.008644;   accu:0.744
epoch:73;     loss:0.008643;   accu:0.741
epoch:74;     loss:0.008675;   accu:0.735
epoch:75;     loss:0.008609;   accu:0.747
total=1984.0
Accuracy of the network on the test dataset: 73 %
epoch:76;     loss:0.008584;   accu:0.746
epoch:77;     loss:0.008576;   accu:0.751
epoch:78;     loss:0.008591;   accu:0.746
epoch:79;     loss:0.008601;   accu:0.748
epoch:80;     loss:0.008554;   accu:0.752
total=1984.0
Accuracy of the network on the test dataset: 74 %
tf-docker ~/zode/src >
```



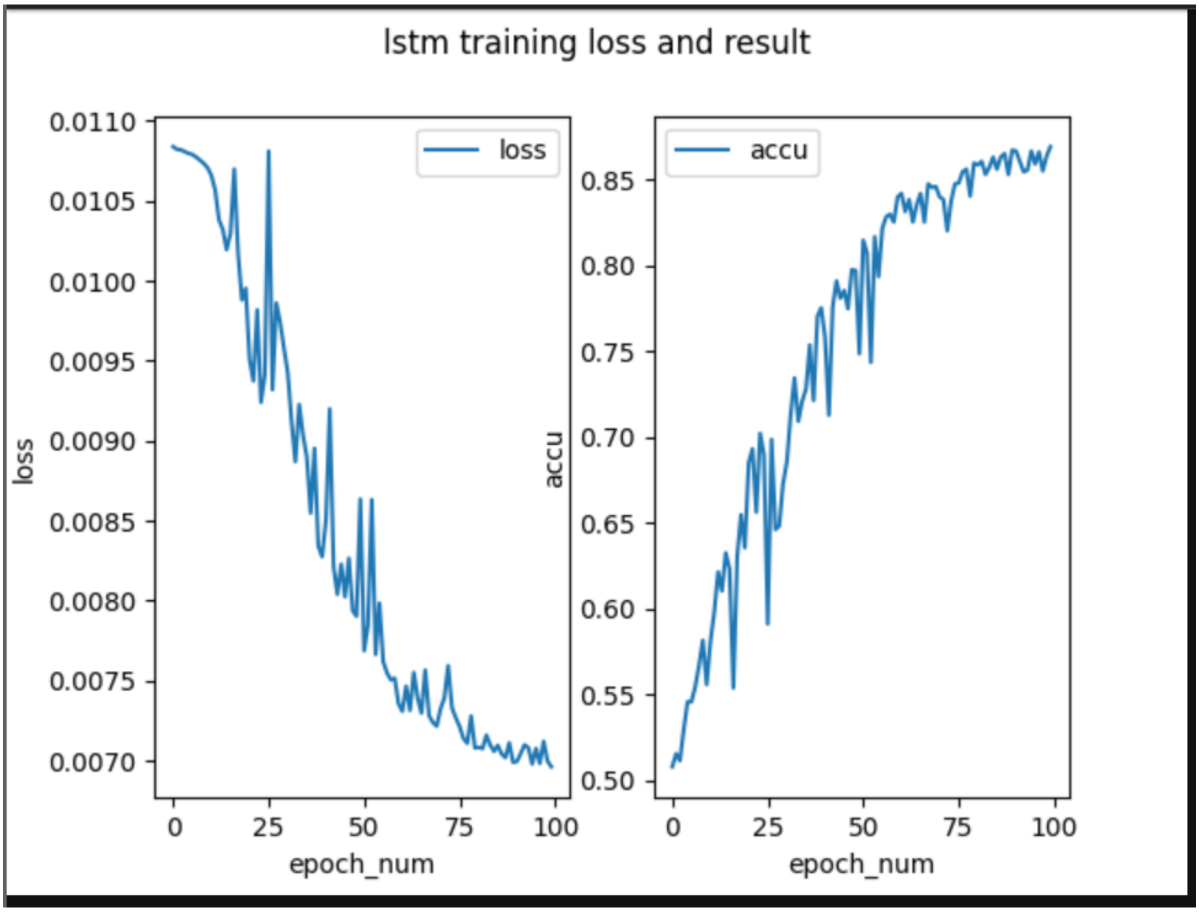
- 模型应该还没有完全收敛，主要是10,20,40附近的几次震荡太伤了，
- 测试准确率=74%
- 可见初始学习率太大，
- 初始lr=0.03，每10轮*减半

最好是在一个学习率下，极值点两边跳跃的时候减半

lstm调库，双向，1层

- lr初始=0.03，20轮减半，训100轮
- 参数向正确的方向拟合，loss也可能会发生震荡

```
Accuracy of the network on the test dataset: 83 %
epoch:81;     loss:0.007084;   accu:0.858
epoch:82;     loss:0.007077;   accu:0.860
epoch:83;     loss:0.007160;   accu:0.853
epoch:84;     loss:0.007099;   accu:0.857
epoch:85;     loss:0.007060;   accu:0.863
total=1984.0
Accuracy of the network on the test dataset: 83 %
epoch:86;     loss:0.007097;   accu:0.856
epoch:87;     loss:0.007043;   accu:0.863
epoch:88;     loss:0.007020;   accu:0.865
epoch:89;     loss:0.007113;   accu:0.853
epoch:90;     loss:0.006991;   accu:0.867
total=1984.0
Accuracy of the network on the test dataset: 83 %
epoch:91;     loss:0.006996;   accu:0.866
epoch:92;     loss:0.007046;   accu:0.860
epoch:93;     loss:0.007100;   accu:0.854
epoch:94;     loss:0.007082;   accu:0.855
epoch:95;     loss:0.006980;   accu:0.866
total=1984.0
Accuracy of the network on the test dataset: 82 %
epoch:96;     loss:0.007078;   accu:0.859
epoch:97;     loss:0.006983;   accu:0.866
epoch:98;     loss:0.007122;   accu:0.855
epoch:99;     loss:0.006999;   accu:0.863
epoch:100;     loss:0.006963;   accu:0.869
total=1984.0
Accuracy of the network on the test dataset: 83 %
tf-docker ~/zode/src >
```



训练100轮，模型差不多收敛了，测试准确率=83%

- 可见双向LSTM要优于LSTM，这是因为还结合下文
- 训练高于测试较多，说明过拟合，需要dropout
- 自己写的lstm再2分类上效果很烂，就不展示了

my lstm，双层

初始lr从1e-1到1e-6，loss没有任何收敛的意思，下图为1e-6的截图

```
tf-docker ~/work/zode/src > python lstm_pkg.py
score's lenth=12500
1000
1000
epoch:1;     loss:0.010840;   accu:0.501042
epoch:2;     loss:0.010840;   accu:0.501042
epoch:3;     loss:0.010843;   accu:0.498958
epoch:4;     loss:0.010843;   accu:0.498958
epoch:5;     loss:0.010839;   accu:0.502083
total=960.0
Accuracy of the network on the test dataset: 50 %
epoch:6;     loss:0.010842;   accu:0.500000
epoch:7;     loss:0.010838;   accu:0.503125
epoch:8;     loss:0.010841;   accu:0.500000
epoch:9;     loss:0.010839;   accu:0.502083
epoch:10;     loss:0.010844;   accu:0.497917
total=960.0
Accuracy of the network on the test dataset: 50 %
epoch:11;     loss:0.010836;   accu:0.505208
epoch:12;     loss:0.010841;   accu:0.501042
epoch:13;     loss:0.010840;   accu:0.501042
epoch:14;     loss:0.010840;   accu:0.501042
epoch:15;     loss:0.010847;   accu:0.495833
total=960.0
Accuracy of the network on the test dataset: 50 %
epoch:16;     loss:0.010849;   accu:0.493750
epoch:17;     loss:0.010839;   accu:0.502083
epoch:18;     loss:0.010839;   accu:0.502083
epoch:19;     loss:0.010843;   accu:0.498958
epoch:20;     loss:0.010840;   accu:0.501042
total=960.0
Accuracy of the network on the test dataset: 49 %
epoch:21;     loss:0.010850;   accu:0.492708
epoch:22;     loss:0.010843;   accu:0.498958
epoch:23;     loss:0.010843;   accu:0.498958
epoch:24;     loss:0.010845;   accu:0.496875
epoch:25;     loss:0.010844;   accu:0.497917
total=960.0
Accuracy of the network on the test dataset: 50 %
epoch:26;     loss:0.010842;   accu:0.500000
epoch:27;     loss:0.010843;   accu:0.498958
epoch:28;     loss:0.010845;   accu:0.496875
epoch:29;     loss:0.010834;   accu:0.506250
epoch:30;     loss:0.010839;   accu:0.502083
total=960.0
Accuracy of the network on the test dataset: 49 %
```

拯救with梯度裁剪

```
nn.utils.clip_grad_norm_(net.parameters(), 1) #gradient clip
```

- 理论上可以避免SGD累积过大，导致的loss陡然上升，未测试

Bert

实验记录

	netmodel	其他	在15周期
1	3层ReLU+200样本	学习率=1e-5,1e-3,5e-5	一直0.5
2		学习率=1e-6	0.848
3	Drop0.3+3层ReLU		0.805
4	换成2000样本		0.836
5	dp0.5	lr*=0.5/5T	

序号2结果：

```
tf-docker ~/work/zode/src > python batch_bert.py
Some weights of the model checkpoint at bert-base-uncased were not used when initializing BertModel: ['cls.predictions.transform.LayerNorm.bias', 'cls.seq_relationship.weight', 'cls.predictions.bias', 'cls.predictions.transform.LayerNorm.weight', 'cls.predictions.decoder.weight', 'cls.seq_relationship.bias', 'cls.predictions.transform.dense.weight', 'cls.predictions.transform.dense.bias']
- This IS expected if you are initializing BertModel from the checkpoint of a model trained on another task or with another architecture (e.g. initializing a BertForSequenceClassification model from a BertForPreTraining model).
- This IS NOT expected if you are initializing BertModel from the checkpoint of a model that you expect to be exactly identical (initializing a BertForSequenceClassification model from a BertForSequenceClassification model).
epoch:1;     loss:0.086;  accu:0.527
epoch:2;     loss:0.084;  accu:0.570
epoch:3;     loss:0.078;  accu:0.805
epoch:4;     loss:0.063;  accu:0.860
epoch:5;     loss:0.063;  accu:0.767
test    loss:0.246;  accu:0.568
epoch:6;     loss:0.058;  accu:0.785
epoch:7;     loss:0.069;  accu:0.728
epoch:8;     loss:0.062;  accu:0.745
epoch:9;     loss:0.066;  accu:0.680
epoch:10;     loss:0.037;  accu:0.915
test    loss:0.094;  accu:0.815
epoch:11;     loss:0.038;  accu:0.875
epoch:12;     loss:0.027;  accu:0.920
epoch:13;     loss:0.014;  accu:0.980
epoch:14;     loss:0.016;  accu:0.953
epoch:15;     loss:0.008;  accu:0.980
test    loss:0.114;  accu:0.848
tf-docker ~/work/zode/src >
```

有点过拟合了，用dropout

序号3结果

```
test    loss:0.114;  accu:0.848
tf-docker ~/work/zode/src > python batch_bert.py
Some weights of the model checkpoint at bert-base-uncased we
ng BertModel: ['cls.predictions.bias', 'cls.predictions.tran
redictions.decoder.weight', 'cls.predictions.transform.Layer
tionship.weight', 'cls.seq_relationship.bias', 'cls.predicti
'cls.predictions.transform.LayerNorm.bias']
- This IS expected if you are initializing BertModel from th
ined on another task or with another architecture (e.g. init
lassification model from a BertForPreTraining model).
- This IS NOT expected if you are initializing BertModel fro
 that you expect to be exactly identical (initializing a Ber
model from a BertForSequenceClassification model).
epoch:1;     loss:0.087;  accu:0.495
epoch:2;     loss:0.085;  accu:0.562
epoch:3;     loss:0.084;  accu:0.537
epoch:4;     loss:0.078;  accu:0.710
epoch:5;     loss:0.069;  accu:0.762
test    loss:0.146;  accu:0.637
epoch:6;     loss:0.065;  accu:0.738
epoch:7;     loss:0.070;  accu:0.718
epoch:8;     loss:0.048;  accu:0.835
epoch:9;     loss:0.070;  accu:0.677
epoch:10;     loss:0.042;  accu:0.905
test    loss:0.145;  accu:0.637
epoch:11;     loss:0.055;  accu:0.777
epoch:12;     loss:0.024;  accu:0.943
epoch:13;     loss:0.022;  accu:0.938
epoch:14;     loss:0.016;  accu:0.960
epoch:15;     loss:0.015;  accu:0.953
test    loss:0.122;  accu:0.805
```

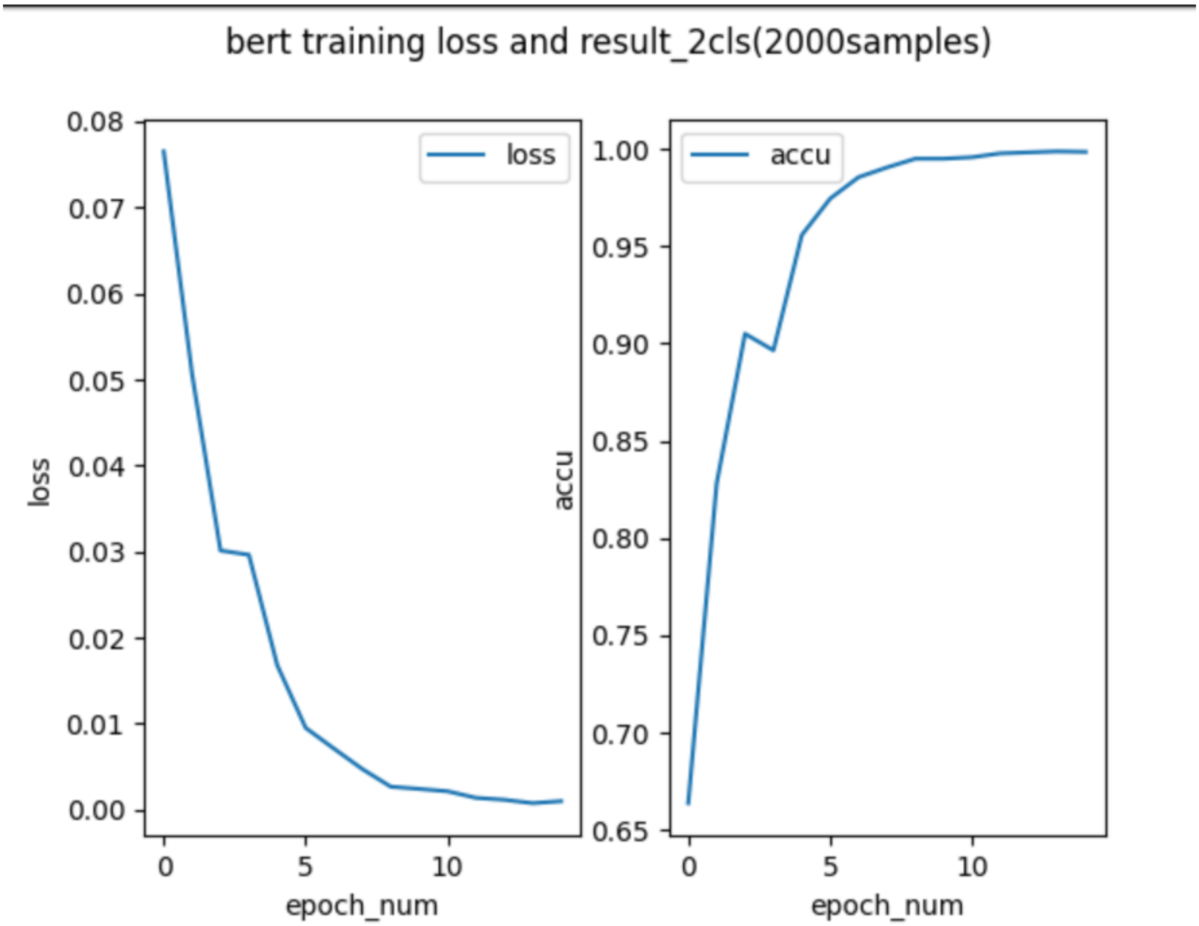
4号结果

```
SyntaxError: unexpected character after line continuation cha
tf-docker ~/work/zode/src > python batch_bert.py
Some weights of the model checkpoint at bert-base-uncased wer
ctions.bias', 'cls.seq_relationship.weight', 'cls.predictions
- This IS expected if you are initializing BertModel from the
- This IS NOT expected if you are initializing BertModel from
epoch:1;     loss:0.078;   accu:0.628
epoch:2;     loss:0.048;   accu:0.829
epoch:3;     loss:0.033;   accu:0.892
epoch:4;     loss:0.021;   accu:0.937
epoch:5;     loss:0.013;   accu:0.964
test    loss:0.081;   accu:0.885
epoch:6;     loss:0.011;   accu:0.971
epoch:7;     loss:0.007;   accu:0.983
epoch:8;     loss:0.005;   accu:0.990
epoch:9;     loss:0.005;   accu:0.988
epoch:10;     loss:0.004;   accu:0.989
test    loss:0.133;   accu:0.885
epoch:11;     loss:0.003;   accu:0.993
epoch:12;     loss:0.003;   accu:0.996
epoch:13;     loss:0.004;   accu:0.990
epoch:14;     loss:0.006;   accu:0.985
epoch:15;     loss:0.006;   accu:0.987
test    loss:0.197;   accu:0.836
```

过拟合，且因为后续学习率衰减得不够，在极值点附近震荡，导致了15周期的波动，因此有改进5

5号结果

```
tf-docker ~/work/zode/src > python batch_bert.py
'HTTPSConnectionPool(host='huggingface.co', port=443): Max retries exceeded with url: /bert-base-uncased/r
used by ConnectTimeoutError(<urllib3.connection.VerifiedHTTPSConnection object at 0x7fe14d6ad9d0>, 'Connec
med out. (connect timeout=10)'))' thrown while requesting HEAD https://huggingface.co/bert-base-uncased/re
'HTTPSConnectionPool(host='huggingface.co', port=443): Max retries exceeded with url: /bert-base-uncased/r
Caused by ConnectTimeoutError(<urllib3.connection.VerifiedHTTPSConnection object at 0x7fe126a65730>, 'Conn
timed out. (connect timeout=10)'))' thrown while requesting HEAD https://huggingface.co/bert-base-uncased/
Some weights of the model checkpoint at bert-base-uncased were not used when initializing BertModel: ['cls
seq_relationship.weight', 'cls.seq_relationship.bias', 'cls.predictions.transform.LayerNorm.weight', 'cls.
se.bias', 'cls.predictions.transform.LayerNorm.bias', 'cls.predictions.decoder.weight', 'cls.predictions.t
- This IS expected if you are initializing BertModel from the checkpoint of a model trained on another tas
cture (e.g. initializing a BertForSequenceClassification model from a BertForPreTraining model).
- This IS NOT expected if you are initializing BertModel from the checkpoint of a model that you expect to
itializing a BertForSequenceClassification model from a BertForSequenceClassification model).
epoch:1;    loss:0.077;  accu:0.664
epoch:2;    loss:0.051;  accu:0.828
epoch:3;    loss:0.030;  accu:0.905
epoch:4;    loss:0.030;  accu:0.896
epoch:5;    loss:0.017;  accu:0.956
test   loss:0.095;  accu:0.870
epoch:6;    loss:0.010;  accu:0.975
epoch:7;    loss:0.007;  accu:0.986
epoch:8;    loss:0.005;  accu:0.991
epoch:9;    loss:0.003;  accu:0.995
epoch:10;    loss:0.002;  accu:0.995
test   loss:0.101;  accu:0.900
epoch:11;    loss:0.002;  accu:0.996
epoch:12;    loss:0.001;  accu:0.998
epoch:13;    loss:0.001;  accu:0.998
epoch:14;    loss:0.001;  accu:0.999
epoch:15;    loss:0.001;  accu:0.999
test   loss:0.133;  accu:0.901
tf-docker ~/work/zode/src > python test.py
```



比原先好点，但还是有点过拟合，可以继续增大训练样本数or增加dropout，限于时间(不做进一步尝试)

最后的real测试(in test.py)

- 训练好的BiLSTM和Bert的参数，已放在src下，可利用test.py中定义的LSTM_last_ele类和Bert类加载这两个参数，进行测试(直接运行test.py即可)

数据集：test_neg[5000:6000]+test_pos[2500:3500]，均未在前面的训练和验证中出现，

```
tf-docker ~/work/zode/src > python test.py
Could not load symbol cublasGetSmCountTarget from libcublas.so.11. Error: /usr/local/cuda/lib64/libcublas.so.11: undefined symbol: cublasGetSmCountTarget
total=1984.0
Accuracy of the BiLSTM on the real test dataset: 79.083 %
Some weights of the model checkpoint at bert-base-uncased were not used when initializing BertModel: ['cls.predictions.transform.dense.weight', 'cls.predictions.transform.LayerNorm.weight', 'cls.predictions.decoder.weight', 'cls.predictions.bias', 'cls.predictions.transform.dense.bias', 'cls.seq_relationship.weight', 'cls.seq_relationship.bias', 'cls.predictions.transform.LayerNorm.bias']
- This IS expected if you are initializing BertModel from the checkpoint of a model trained on another task or with another architecture (e.g. initializing a BertForSequenceClassification model from a BertForPreTraining model).
- This IS NOT expected if you are initializing BertModel from the checkpoint of a model that you expect to be exactly identical (initializing a BertForSequenceClassification model from a BertForSequenceClassification model).
Accuracy of Bert on testDataset is 90.550 %
tf-docker ~/work/zode/src >
```

BiLSTM效果达到79%，而Bert则高达90.5%，

这是由于Bert除了LSTM，还结合了Encoder+Decoder，并且经过了大量训练，

并且这两个模型的参数量也不一样，

脑 › Backup (D:) › Note › DLNote › lab4_RNN › zode › params

名称	修改日期	类型	大小
bert.params	2023/6/20 9:55	PARAMS 文件	428,122 KB
lr_adma_l2_2e-4.params	2023/6/20 9:56	PARAMS 文件	955 KB

膜拜大模型的威力！