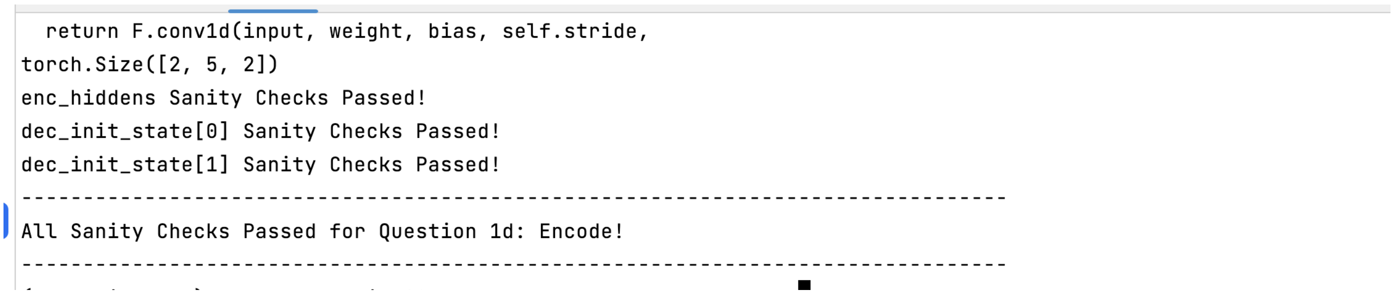
a4

(a)-(d)



```
  return F.conv1d(input, weight, bias, self.stride,
torch.Size([2, 5, 2])
enc_hiddens Sanity Checks Passed!
dec_init_state[0] Sanity Checks Passed!
dec_init_state[1] Sanity Checks Passed!
--------------------------------------------------------------------------------
All Sanity Checks Passed for Question 1d: Encode!
--------------------------------------------------------------------------------
```

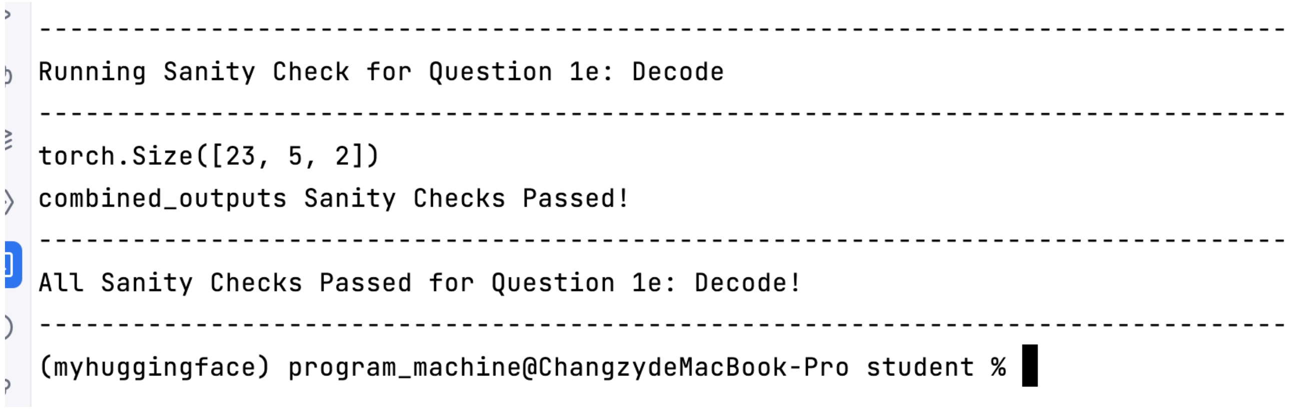
还是stanford好,有指导有参考,CMU的简直了,给你一大堆文件你自己写去吧.花了几个小时写这几题,受益匪浅. LSTM先让你用了,然后你再去了解原理,这样反而是更好的,我相信我会在接下来更好的理解LSTM的机制.

(f)

这个确实不容易,尤其是有很多陌生知识的时候,收获很大,不过每走一步可能回顾过去写的代码,因为变量和函数以及文档太多了.

(e)

e被step的返回值卡住卡了半天.



```
--------------------------------------------------------------------------------
Running Sanity Check for Question 1e: Decode
--------------------------------------------------------------------------------
torch.Size([23, 5, 2])
combined_outputs Sanity Checks Passed!
--------------------------------------------------------------------------------
All Sanity Checks Passed for Question 1e: Decode!
--------------------------------------------------------------------------------
(myhuggingface) program_machine@ChangzydeMacBook-Pro student %
```

(g)

First explain (in around three sentences) what effect the masks have on the entire attention com- putation. Then explain (in one or two sentences) why it is necessary to use the masks in this way.
使用mask的原因就是因为输入需要使用padding吧,但是padding的部分不参与计算,所以具体计算的时候就要mask掉.

> GPT的回答:掩码在注意力计算中的作用是选择性地遮蔽序列的某些元素，以防止其影响网络输出。这有助于防止过度拟合和提高泛化性能。使用这种方式使用掩码是必要的，以确保模型不会过于关注无关信息，导致在未见数据上表现不佳。通过选择性地遮蔽元素，模型可以将注意力集中在输入序列的最重要部分，从而更有效地学习.

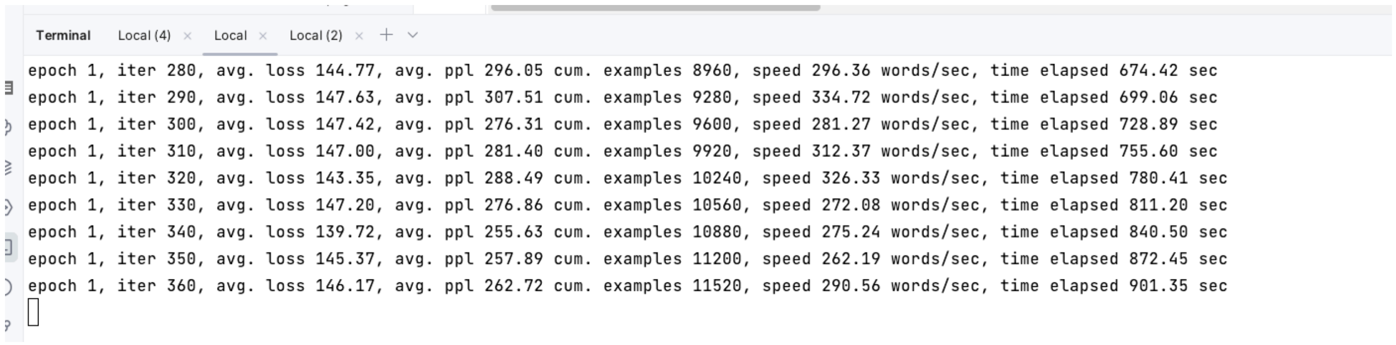
说明我们的理解是偏差不大的.

上面就是编码工作的全部,训练的话就再说吧.

(i) (4 points) (written) In class, we learned about dot product attention, multiplicative attention, and additive attention.
As a reminder, dot product attention is et,i = sTt hi, multiplicative attention is et,i = sTt Whi, and additive attention is et,i = vT tanh(W1hi + W2st).
i. (2 points) Explain one advantage and one disadvantage of dot product attention compared to multiplicative attention.
意思是点乘注意力和矩阵乘注意力相比,好坏处之一.

> 相对于乘法注意力，点积注意力的一个优点是计算速度更快，因为其计算方法较为简单。同时，点积注意力能够有效地捕捉到输入序列中元素之间的相关性，对于一些任务可能会有更好的表现。缺点在于，点积操作不具备缩放性，当向量维度较高时，点积的值通常很大，可能会导致模型训练不稳定或梯度消失问题。此外，点积操作还需要将输入向量标准化，这也会增加计算代价。
> ii. (2 points) Explain one advantage and one disadvantage of additive attention compared to multiplicative attention.
> 下面是加性注意力和矩阵乘注意力相比的好坏处之一.
> 相对于乘法注意力，加法注意力的一个优点是计算速度快，因为其计算过程更简单。与此同时，乘法注意力往往能够更好地表现输入序列元素之间的相关性，对于一些计算任务来说可能会有更好的表现。缺点在于，在处理长序列时，由于加法过程不能捕捉到序列中元素之间的交互作用，因此它的表示能力有限，可能无法捕捉到整个序列的重要信息。

之前由于怕苦怕难和时间长等等原因不敢迈出这训练的第一步,但还是怎么说呢,这一步都不敢迈出去,怎么谈去做别的实验什么的.这个NMT整体是个很好的学习工具,把它的思想学到了比你读好几本书,看多少门课都有用!加油!



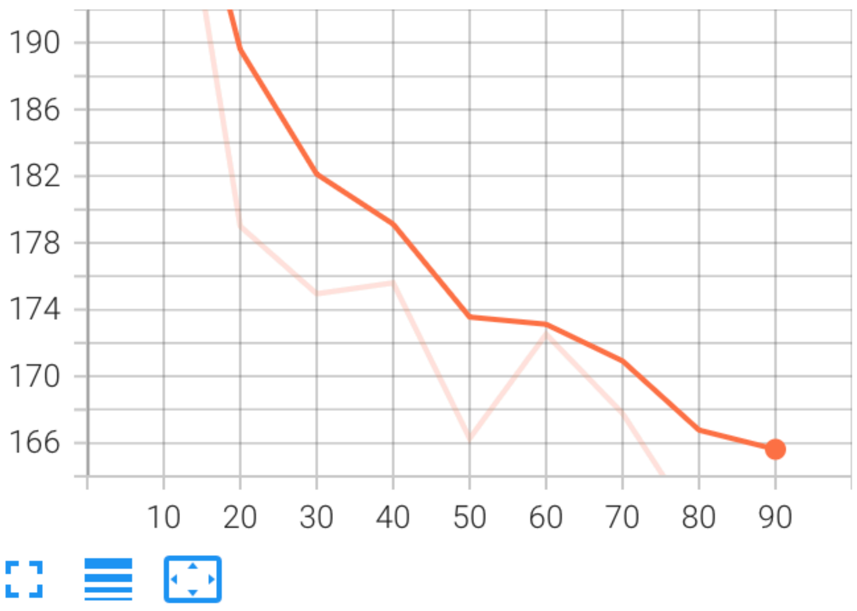
```
epoch 1, iter 280, avg. loss 144.77, avg. ppl 296.05 cum. examples 8960, speed 296.36 words/sec, time elapsed 674.42 sec
epoch 1, iter 290, avg. loss 147.63, avg. ppl 307.51 cum. examples 9280, speed 334.72 words/sec, time elapsed 699.06 sec
epoch 1, iter 300, avg. loss 147.42, avg. ppl 276.31 cum. examples 9600, speed 281.27 words/sec, time elapsed 728.89 sec
epoch 1, iter 310, avg. loss 147.00, avg. ppl 281.40 cum. examples 9920, speed 312.37 words/sec, time elapsed 755.60 sec
epoch 1, iter 320, avg. loss 143.35, avg. ppl 288.49 cum. examples 10240, speed 326.33 words/sec, time elapsed 780.41 sec
epoch 1, iter 330, avg. loss 147.20, avg. ppl 276.86 cum. examples 10560, speed 272.08 words/sec, time elapsed 811.20 sec
epoch 1, iter 340, avg. loss 139.72, avg. ppl 255.63 cum. examples 10880, speed 275.24 words/sec, time elapsed 840.50 sec
epoch 1, iter 350, avg. loss 145.37, avg. ppl 257.89 cum. examples 11200, speed 262.19 words/sec, time elapsed 872.45 sec
epoch 1, iter 360, avg. loss 146.17, avg. ppl 262.72 cum. examples 11520, speed 290.56 words/sec, time elapsed 901.35 sec
```

说是要跑4个小时,没法在本地跑,电脑烫的吓人,而且没有GPU,代码是配置好了,colab也不能这么长时间跑,试试别的途径.

参照文档的做法
下面两条指令应该就够了
sh run.sh train_local
sh run.sh test

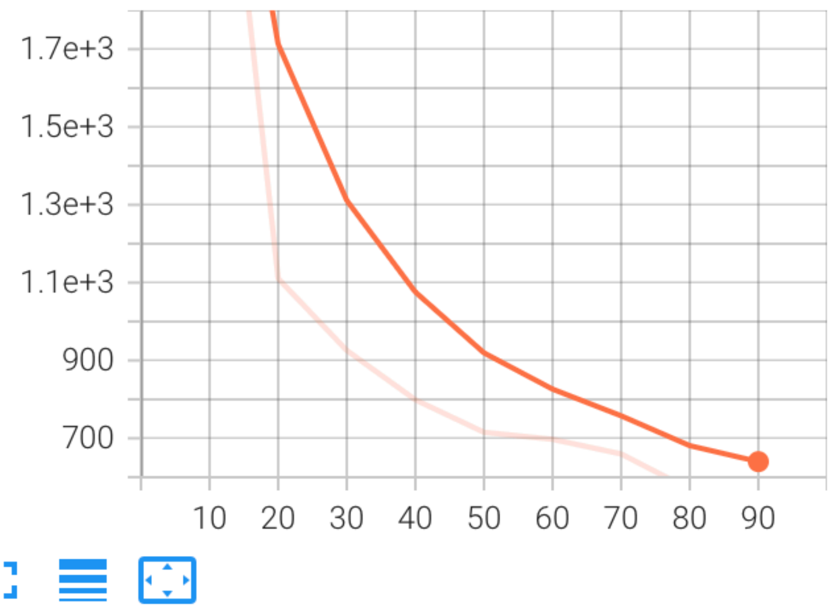
tensorboard serve --logdir=runs可以查看一些日志信息.
正如文档说的,可以看到loss和perplxity在显著的下降,当然,我的电脑也显著的发烫.

loss/train
tag: loss/train



perplexity

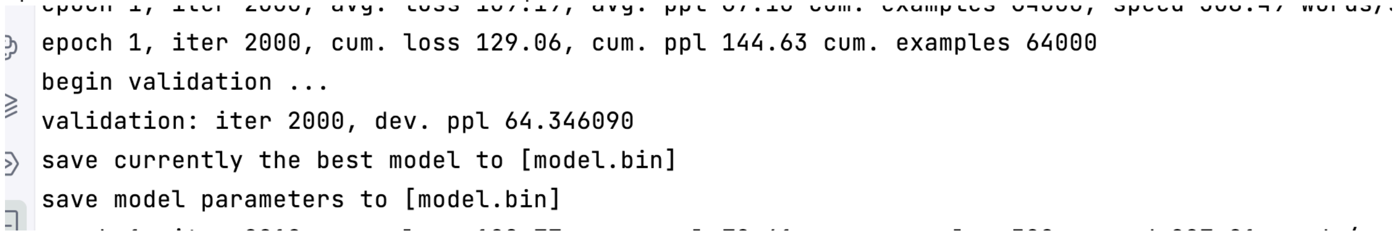
perplexity/train
tag: perplexity/train



文档里提出了一个问题:If it's not obvious, think about why we regard h_1^{enc} 和 h_m^{enc} as the 'final hidden state' of the Encoder.
我认为这主要是由于这是一个双向的lstm,1和m都充当了首尾的角色,所以采用了这两个值去做final state.

> epoch 1, iter 1950, avg. loss 114.67, avg. ppl 73.41 cum. examples 62400, speed 332.54 words/sec, time elapsed 4841.99 sec

cpu跑真的好慢啊,再等会跑俩小时了,怎么epoch1都没跑完啊.



```
epoch 1, iter 2000, cum. loss 129.06, cum. ppl 144.63 cum. examples 64000
begin validation ...
validation: iter 2000, dev. ppl 64.346090
save currently the best model to [model.bin]
save model parameters to [model.bin]
```

> begin validation ...
> validation: iter 4000, dev. ppl 34.291627
> save currently the best model to [model.bin]
> save model parameters to [model.bin]

已经跑了3小时了,才跑到iter4000.

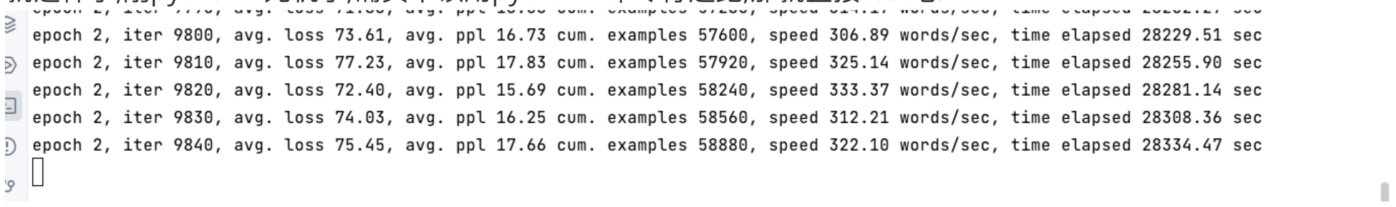
> epoch 1, iter 4660, avg. loss 87.13, avg. ppl 30.06 cum. examples 21120, speed 250.24 words/sec, time elapsed 13586.30 sec

loss和ppl相比于上午已经降低了很多了,我可以训练到晚上,我倒看看你能要多久.

已经跑了6个小时了,再跑3小时就到晚上六点了.

> epoch 2, iter 7260, avg. loss 74.86, avg. ppl 18.33 cum. examples 40320, speed 309.01 words/sec, time elapsed 21706.43 sec

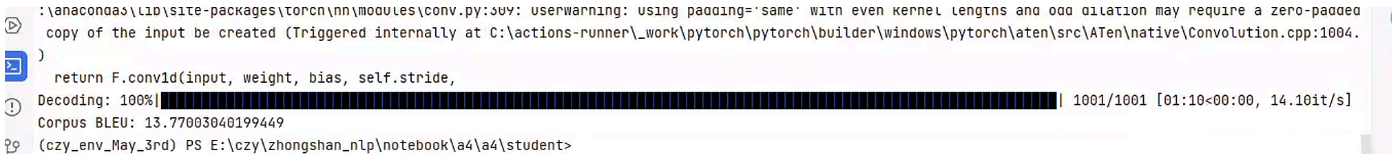
就这样了,刚pycharm死机了,确实不该用pycharm命令行过跑,那就直接test吧.



```
epoch 2, iter 9800, avg. loss 73.61, avg. ppl 16.73 cum. examples 57600, speed 306.89 words/sec, time elapsed 28229.51 sec
epoch 2, iter 9810, avg. loss 77.23, avg. ppl 17.83 cum. examples 57920, speed 325.14 words/sec, time elapsed 28255.90 sec
epoch 2, iter 9820, avg. loss 72.40, avg. ppl 15.69 cum. examples 58240, speed 333.37 words/sec, time elapsed 28281.14 sec
epoch 2, iter 9830, avg. loss 74.03, avg. ppl 16.25 cum. examples 58560, speed 312.21 words/sec, time elapsed 28308.36 sec
epoch 2, iter 9840, avg. loss 75.45, avg. ppl 17.66 cum. examples 58880, speed 322.10 words/sec, time elapsed 28334.47 sec
```

测试的话他要有cuda环境,那就在实验室电脑上跑.

文档里说要跑到18以上,唉,我这cpu真的是无语,13也还行吧,也是完成了,收获颇丰,后续还可以继续研究它的代码,拿来diy.



```
:\anaconda3\lib\site-packages\torch\nn\modules\conv.py:309: UserWarning: Using padding='same' with even kernel lengths and odd dilation may require a zero-padded
 copy of the input be created (Triggered internally at C:\actions-runner\_work\pytorch\pytorch\builder\windows\pytorch\aten\src\ATen\native\Convolution.cpp:1004.
)
  return F.conv1d(input, weight, bias, self.stride,
Decoding: 100%|██████████████████████████████████████████████████████████████████████████████| 1001/1001 [01:10<00:00, 14.10it/s]
Corpus BLEU: 13.77003040199449
(czy_env_May_3rd) PS E:\czy\zhongshan_nlp\notebook\a4\a4\student>
```