**20195202　赵浩天　人工智能1902**

**本次实验实验包括基础部分和提升部分，代码已经放到GitHub上面了**

**LSTM结构示意图：**

**单层示意图：**由于手绘图片打印效果不太好，故照片已经放到GitHub上面，文件名：单层示意图.png

**双层示意图：**同理文件名，双层示意图

**模型设计：**

**单层模型设计：**

1. **Hidden state 和memory cell 的初始值设置**

首先我们要知道RNN在每一步传递的是 hidden state ，而LSTM在每一步不仅需要传递hidden state 还需要传递memory cell，这也就是循环单元要传递的内容。但是第一个单元并没有别的单元给他传递这俩个所需要的参数，所以我们需要在将这个俩个初始的参数设置好，传到模型里面进来，也就是参数H，C的由来。

1. **三个门的作用以及相应实现**

Forget gate和上一个单元传输的memory cell进行计算，选出哪些信息是可以留下的，哪些信息是可以需要忘记的 ⊙

Input gate 和candidate gate 进行预算选出这次的输入信息哪些是重要的

⊙C\_tilda

Output gate 和我们当前状态的cell进行运算选出哪些信息是我们想输出的

⊙ tanh()

**3 三个门以及candidate state以及输出的相应实现**

这三个门以及candidate state以及输出都是由简单的linear层实现的

= σ( + + )

= σ( + + )

= σ( + + )

C\_tilda= tanh( + +

OUT = H +

**双层模型设计：**

双层LSTM就是将俩个单层的LSTM叠加起来，其中每个时间点的单元相互对应，并且下层相应单元的隐藏层输出和上层本层前一个时间单元的隐藏层输出按照公式进行运算，作为该单元的隐藏层输入。

所以说双层与单层的设计区别就主要在于如何传递隐藏层的输出结果。

我采用的第二层的首个单元的隐藏层状态和candidate输入与第一层的一样，H1=H，C1=C在第一层的隐藏层计算完毕后，将其与第二层前一个神经单元的隐藏层做运算H1 = torch.sigmoid(self.W\_xh(H) + self.W\_hh(H1) + self.b\_h)，作为该单元的输入，并用该神经单元的输出更新H1作为下一个神经单元输入的一部分。

**运行结果：**

**单层结果：**

Valid 5504 samples after epoch: 0004 loss = 5.745343 ppl = 312.731

Epoch: 0005 Batch: 100 /603 loss = 5.380189 ppl = 217.063

Epoch: 0005 Batch: 200 /603 loss = 4.905917 ppl = 135.087

Epoch: 0005 Batch: 300 /603 loss = 5.309510 ppl = 202.251

Epoch: 0005 Batch: 400 /603 loss = 5.555227 ppl = 258.586

Epoch: 0005 Batch: 500 /603 loss = 5.392020 ppl = 219.647

Epoch: 0005 Batch: 600 /603 loss = 5.350584 ppl = 210.731

Epoch: 0005 Batch: 604 /603 loss = 4.844369 ppl = 127.023

Valid 5504 samples after epoch: 0005 loss = 5.701885 ppl = 299.431

Test the LSTMLM……………………

Test 6528 samples with models/LSTMlm\_model\_epoch5.ckpt……………………

loss = 5.662318 ppl = 287.815

**双层结果：**

Valid 5504 samples after epoch: 0004 loss = 5.789288 ppl = 326.78

Epoch: 0005 Batch: 100 /603 loss = 5.253248 ppl = 191.186

Epoch: 0005 Batch: 200 /603 loss = 4.933904 ppl = 138.921

Epoch: 0005 Batch: 300 /603 loss = 5.300286 ppl = 200.394

Epoch: 0005 Batch: 400 /603 loss = 5.451818 ppl = 233.182

Epoch: 0005 Batch: 500 /603 loss = 5.402319 ppl = 221.92

Epoch: 0005 Batch: 600 /603 loss = 5.369112 ppl = 214.672

Epoch: 0005 Batch: 604 /603 loss = 4.864778 ppl = 129.642

Valid 5504 samples after epoch: 0005 loss = 5.763897 ppl = 318.587

Test the LSTMLM……………………

Test 6528 samples with models/LSTMlm\_model\_epoch5.ckpt……………………

loss = 5.706549 ppl = 300.831

**遇到的问题以及解决方法：**

1. 与之前学长发的手撸RNN模型相比，实现LSTM的过程中并没有例如(np.eye(n\_class)[input]这样的代码，也就是说没有词的one-hot表示，也没有词嵌入的实现。查了资料后发现nn.embedding()其实是NLP中常用的词嵌入层，已经封装好了，而我需要做的就是了解它的输入输出以及参数的意义。

nn.embedding:这是一个矩阵类，该开始时里面初始化了一个随机矩阵，矩阵的长是字典的大小，宽是用来表示字典中每个元素的属性向量，向量的维度根据你想要表示的元素的复杂度而定。类实例化之后可以根据字典中元素的下标来查找元素对应的向量。

1. 在修改代码的时候解释器下面显示这样：RuntimeError: Expected all tensors to be on the same device, but found at least two devices, cpu经排查之后发现是因为H和C

H = torch.zeros(batch\_size, n\_hidden)

C = torch.zeros(batch\_size, n\_hidden)

这俩个初始化的张量没有加入到我电脑的GPU里面，导致程序在计算时，数据一部分在CPU而一部分在GPU中，因此将代码更改为就行了，这样数据都在GPU上面。

H = torch.zeros(batch\_size, n\_hidden, device=device)

C = torch.zeros(batch\_size, n\_hidden, device=device)

1. 在调成参数batch\_size的时候，有时候会出现这样报错：

RuntimeError: The size of tensor a (128) must match the size of tensor b (32) at non-singleton dimen。

百度之后得知这是因为数据集有时候并不能被batch\_size整除，从而导致最后一个batch的数据大小和batch\_size不相等，从而不能够实现计算。所以最后更改为学长的参数就没再出现这样的问题了。

**实验结果分析：**

首先在运行过程中，我发现双层的比单层的慢了太多了，这也难怪，毕竟双层的笔单层的多了一倍多的计算量。

另外，我发现双层的结果好像还没有单层的好，具体原因我还没有搞明白。然后百度了一下，了解到LSTM层数不能过多，层数的增加会带来时间开销和内存开销的指数级增长，接着，你会体验到层与层之间的梯度消失——这就是最致命的一点。当层数超过了三层，层与层之间的梯度消失情况变得非常明显，再加上是时序模型，导致靠近输入层的LSTM层或者更新迭代放缓，收敛效果和效率急剧下降，甚至非常容易进入局部最小的困境。

**自然语言这门课的心得体会：**

首先在这门课中，我们不仅了解一些非常经典的NLP语言模型，而且还有机会去动手实现他们。而且学会知识的同时我们也学会了如何做人，如何去发展自己。比如，如何在当下大家都在相互卷的过程中，如何去一步步实现自己的价值，这个价值可能是当下看不到的，但是却对你的将来至关重要的。