语言分析与机器翻译实验报告

学号：2001883 姓名：郑乃公 班级：计硕2003

**一、RNNLM简介**

首先阅读了论文” Recurrent neural network based language model”，论文中提出了一种新的基于循环神经网络的语言模型(RNN LM)，并将其应用于语音识别。语音识别实验显示，与使用相同数据量训练的模型进行比较时，华尔街日报任务的单词错误率降低了约18％，而对难度更大的NIST RT05任务进行了比较，即使backoff模型使用了比RNNLM更多的数据进行训练，其词错误率也降低了约5％。并且提供了充足的经验证据，表明connectionis语言模型除了具有很高的计算（训练）复杂度之外，还优于标准n-gram技术。

在PPT中，说明了FNN与RNN之间的关系以及区别。FNN是前馈神经网络，是n-gram方法的一种，首先将每个词用0-1向量表示，之后词化成分布式表示，之后将n-1个词进行级联，输入到神经网络中，利用Tanh函数和Softmax函数进行计算，得到概率。虽然FNNLM有效，但是和传统的n-gram LM一样，需要依赖有限上下文假设。所以循环神经网络RNN解决了此问题，它假设每个词的生成都依赖已经生成的所有词，对于不同位置的词的生成概率都可以用同一个函数描述。对输入序列(x0,x1,…,xt,…)，其中xt表示序列中的第t个元素，也被称为时刻t输入，对应输出序列(y1,y2,…,yt,…)。再循环圣经网络中，每个时刻的输出都可以用一个循环单元描述，并且t时刻的状态是t-1时刻状态的函数，每个时刻t的循环单元的输出成为下个时刻t+1循环单元输入的一部分。

**二、首先根据已有FNNLM学习程序各部分执行的功能。**

1、LoadArgs(argc, argv, model);

将命令行参数argv中的参数分析，并导入到模型model中。

其中关键参数有model.eSize : embedding size，model.vSize : vocabulary size，model.hDepth :隐藏层深度，model.hSize : 隐藏层输出向量大小，learningRate : 学习率，nEpoch : Epoch总数，model.n : n-gram方法的n。

2、Check(model);

检查关键参数是否输入以及正确。

3、Init(model);

初始化相关张量的大小，包括embeddingW，隐藏层中的参数hiddenW、hiddenB，输出层的参数outpuW和outputB，以及embeddingW,outputW,hiddenW的初始值，这里使用的是-minmax到minmax的均匀分布，outputB和hideenB的初始值为0。

4、Train(trainFN, shuffled, model);

训练使用的函数，trainFN是训练数据的文件名，shuffled表示是否将训练数据随机打乱。

首先使用LoadNGrams()加载一段ngrams，使用MakeWordBatch()形成输入input和gold的张量。根据autoDiff判断是否使用自动微分，这里只讨论使用自动微分的情况。首先使用Clear()使梯度清零，使用ForwardAutoDiff()计算前向传播，使用CrossEntropy计算损失函数为交叉熵时的损失，autoDiffer.Backward实现反向传播和自动微分，Update()更新模型参数。

在计算前向传播度的函数ForwardAudoDiff()中，使用Gather()得到句子的embedding，使用Reshape()将embedding变形成一个向量，使用hidden = HardTanH(MMul(hidden, model.hiddenW[i]) + model.hiddenB[i])计算，之后使用output = Softmax(MMul(hidden, model.outputW) + model.outputB, 1)将输出变成一个概率分布。

5、Dump(modelFN, model);

在训练结束后保存模型

6、Read(modelFN, model);

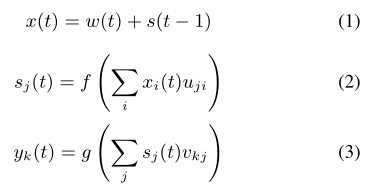
读入模型

7、Test(testFN, outputFN, model);

使用新数据测试模型

**三、根据RNNLM和FNNLM的区别修改相关代码**

首先，关注RNNLM的公式。在RNNLM论文中，输入层、隐藏层和输出层的计算公式如下：



其中，f(z)是sigmoid函数，g(z)是softmax函数。

之后根据代码的流程，逐步修改代码。

首先是参数读入函数LoadArgs()，在此函数中没有需要修改的。但在参数的输入上需要修改和注意的一点是，由于x(t)=w(t)+s(t-1)，所以w(t)和s(t-1)的行数、列数需要相等，而w(t)的列数等于每个词的分布式表示的列数，即model.eSize，而s(t)的列数是隐藏层呢输出的列数，即model.hSize，所以要求在输入时eSize和hSize相等。

之后是初始化函数Init()，发现在公式中没有使用偏移量，所以在初始化函数中不需要初始化hiddenB和outputB，将相关代码删去。

之后是训练函数Train()：

1、在ngrams的读入函数LoadNGram()中（现在修改为LoadOneSentence()），再FNNLM中，起到的作用是从文件中读入一批单词，将其整理成为ngram需要的格式，即形如(i,i+1,…,i+n-1)的n个单词存储在ngrams[i]中。现在在RNNLM中，需要的是每次读入一句话，所以修改相关代码，首先将MAX\_N\_GRAM修改为128，意为每次最多处理128个单词的句子。其次将ngrams修改为XTensor[MAX\_N\_GRAM]，以及结构体NGram中的words改为int[1]，ngrams[i]表示一句话的第i个单词。在每次读入一句话后，判断句子中单词的总数是否大于MAX\_N\_GRAM，如果大于，那么截取前MAX\_N\_GRAM个单词存入ngrams，并且下次从之后的MAX\_N\_GRAM个单词开始读入；如果小于，那么把句子中的单词都存入ngrams，并且下次从一个新的句子开始读入。最后，返回的是ngrams中单词数量-1，因为最后一个单词通常是截止符，而且在判断时，是通过前m个单词判断第m+1个单词，所以做此操作。

2、之后对于MakeWordBatch()函数，在FNNLM中起到的作用是将每个ngrams中的第i个单词（1<=i<=n-1）的0-1表示合并到矩阵input[i]中，以及将第n个单词合并到矩阵gold中。在RNNLM中，由于每个x(t)都需要根据上一个循环单元输出的s(t-1)以及现在的单词才能得到，所以不能将整个x都合并为一个矩阵。在只考虑使用自动微分的情况下，只需要更改得到gold的MakeWordBatch()函数。现在将gold变量修改成为一个长度为ngramNum的XTensor数组，gold[i]代表了ngrams中的第i个单词的0-1表示。

3、在仅考虑自动微分的情况下，考虑每次对一个词进行前向传播、损失计算、反向传播和参数更新。因为每次输入都需要前一个循环单元的输出s(t-1)，所以增加变量XTensor hidden，初始值为0.1f。

4、对于前向传递函数ForwardAutoDiff()，在FNNLM中，首先将输入转化为0-1表示，并利用Gather()函数得到分布式表示，利用Reshape()化成低维向量，之后进行计算。在RNNLM中需要先转化为0-1分布然后使用Gather()得到分布式表示，但不需要Reshape()，之后进行计算，在论文中首先使用的是Sigmoid然后使用Softmax，但在查阅相关资料以及课上的ppt后，发现f(z)这步使用TanH函数的较多，所以在程序中使用了HardTanH函数。·

**四、做过的尝试**

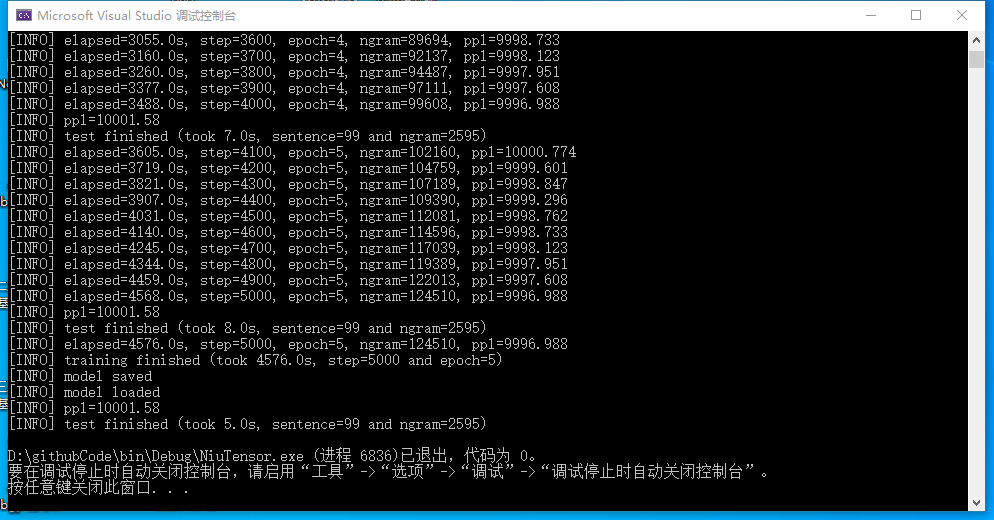
1、在输入方面，曾经尝试每次读入一整句话，不受长度限制，通过将NGram结构体中的words设为int\*类型，之后在LoadOneSentence()函数中，根据读入的单词的个数，动态分配，但是后续出现问题：在没有修改ngrams->words时，其值自己发生变更，而且没有找到解决方法，所以还是采用了现在定长MAX\_N\_GRAM的方式。

2、在输出方面，曾经尝试过利用Concatenate()函数将一次输入的所有output和gold整理成为一整个XTensor，但是这样操作之后在反向传播自动微分的时候出现问题，并且不知道如何修改。现在改为每次输入一个词，对单个词进行前向传播、反向传播、计算损失和更新参数。以及尝试过将ngrams整体输入，得到若干个output后再进行反向传播、计算损失和更新参数，但是发现模型的参数不更新，每一个epoch中得到的结果都是相同的。但是在现在的结构下，此问题仍然没有解决，模型的参数依然没有更新。

**五、结论**

在本项目中，根据论文“Recurrent neural network based language model”以及课上PPT，尝试完成RNNLM，尝试了多种方法，以及各种的组合，虽然程序能够运行，也进行了各个步骤，但是模型的参数不更新，体现在每一个epoch中，相同输入的ppl没有发生改变。认为可能是反向传播自动微分的部分出现了问题，但是由于对代码、基础知识以及相关算法研究的不够深入，不知道错误到底产生在哪，以及如何修改，导致了现在的结果。

并且由于没有服务器，需要使用CPU进行训练，所以使用了上课时的数据集中，训练部分的前1000行做训练，测试集中的前100行做测试。



**六、感受**

首先在代码方面，基本是第一次用C++的相关库写自己的代码，以前都是自己从零开始写或者使用的Python中相关的库，发现自己对于C++代码的熟练度还有很多不足，例如对于符号\*和&的使用方面仍然很不明确，不能清晰地掌握什么时候该用哪个，以及各自起到的功能。而且在使用NiuTensor库以及修改自己代码中的bug时，阅读了库中的代码，发现所有函数都有明确的解释，包括函数的起到的功能以及输入输出的含义，以及代码中大部分都有注释，解释了起到的功能。我认为这一点非常值得我学习，因为在很多时候自己写的代码都没有注释，先不论其他人是否能看懂，即使是隔了一段时间后的自己也经常看不懂，所以在今后的代码中，自己应该加强对于注释的标注，方便他人方便自己。并且我认为，因为NiuTensor库中注释很完整，所以在教学过程中使用NiuTensor库很合适，可以继续的沿用下去，并且在教学或者PPT中也可以增加一些基础函数的示例以及介绍，例如XTensor的输出函数Dump()，这个函数花费了我一段时间才找到以及能够正确地输出内容。

其次，感受到了“造车轮”的不易。很多时候我们都是“用车轮”的人，使用别人开源的已经完成的基础功能，完成自己代码中的其他功能，导致很多时候不知道“造车轮”有多困难，因为需要考虑代码的健壮性，还需要完成足够多的功能，才能造成一个好的车轮。

最后，通过学习这门课以及完成这个代码感受到了语言分析与机器翻译带来的乐趣。在课上肖老师和朱老师讲述的基础知识以及各种人生经验、经历的故事都让我受益匪浅，虽然现在目前的研究方向与自然语言处理并不相关，但在学习了相关知识后，相信在将来的某一时间一定会排上用场，无论是在以后的招聘中还是在碰到某一个可以用相关知识解决的问题时。