**研究生学习周报-研一下**

作者：蔡明炜

**目录**

[第一周 4月6日~4月12日 2](#_Toc44279356)

[第二～五周 4月13日~5月10日 7](#_Toc44279357)

[第六周 5月11日~5月17日 11](#_Toc44279358)

[第七周 5月18日~5月24日 14](#_Toc44279359)

[第八周 5月25日~5月31日 19](#_Toc44279360)

[第九周 6月1日~6月7日 21](#_Toc44279361)

[第十周 6月8日~6月14日 23](#_Toc44279362)

[第十一～二周 6月15日~6月28日 25](#_Toc44279363)

### 第一周 4月6日~4月12日

组会准备: 《NeuroSAT》代码讲解

该程序主要由四大部分组成：生成dimacs格式的数据集过程，将dimacs数据转换成训练数据的过程，训练和测试过程，从结果中求解出可满足的分配的过程。

gen\_sr\_dimacs：首先介绍一下dimacs数据格式。dimacs是用来保存cnf公式的一种热门数据格式。

DIMACS文件用字符c引导注释，紧接着注释之后的一行“p cnf <变量数目> <子句数目>”，然后接下来是各个子句以各不相同介于-nbvar和nbvar的非空数字组成，以0结束。比如：

p cnf 3 4

1 -3 0

2 3 -1 0

-2 -1 0

-2 -3 0

子句分别是 x1^-x3,x2^x3^-x1,-x2^-x1,-x2^-x3

下面函数主体。这个函数调用了一个sat求解器minisat的方法，并自己定义了4个主要方法，其中比较重要的方法是generate\_k\_iclause和gen\_iclause\_pair。

当执行这个python文件时，先通过argparse模块从命令行读取参数，主要的是输出的路径，生成子句对的数量，变量的最大和最小数量。然后根据要生成的子句对数量n\_pairs，循环生成子句对。在循环中，调用gen\_iclause\_pair生成一个子句对，得到变量数量，子句集，不可满足的子句和可满足的子句的返回值。

这里就是论文中写的生成子句对的方式。利用随机函数从max与min中随机挑选一个n，在solver中生成n个变量。然后进入循环。k是一个平均值略大于4的小整数，调用generate\_k\_iclause生成一个包含min(n,k)个不同变量的子句。generate\_k\_iclause的内容是，利用np.random.choice函数从n个数中随机不放回的抽取min(n,k)个数，不放回保证抽取的数不会重复，这些数就代表变量的编号，比如说抽到4就代表变量x4，然后对每一个变量以0.5的概率随机取反，以表示变量的否定。这里加了1是因为抽取从0开始的。这样返回了一个列表，代表了子句中出现的变量及他们的值。

然后gen\_iclause\_pair继续执行，通过minisat的solver判断刚才生成的子句是否可满足，若可满足则将该子句加入子句集并继续循环，直到不可满足为止。保存不可满足的子句为unsat，对该子句的第一位求反则可得到一个可满足的子句sat，一起返回。

根据上述得到的返回值，分别生成保存unsat和sat的子句的文件名，并将两个子句集分别写入文件。这里先将不可满足的子句对加入到子句集中保存，然后用反向索引的方式用sat的子句覆盖掉unsat的子句，然后保存。

然后是将dimacs文件转变为数据集的文件：dimacs\_to\_data。这个函数本身定义了两个方法，一个是读取dimacs文件的方法parse\_dimacs，一个是根据输入创建保存batch的文件的文件名的函数mk\_dataset\_filename。这个函数调用了一个利用minisat判断是否可满足的函数solver，还有一个定义了问题格式并根据输入建立问题的类mk\_problem。输入dimacs文件保存在的目录路径，要输出数据集的路径，和每组最大的节点数。函数通过os.listdir获得dimacs存放的文件夹中的所有文件的名字，按排列好的顺序依次通过parse\_dimacs读取文件，每次读一行，若首个字符为c则为注释，读下一行 ，直到首个字符不为c为止。当首行的第一个字符为p时，开始读取cnf公式，获取变量数n\_vars和所有保存的子句iclauses。计算总结点个数，新加入的总节点个数是否会导致当前batch爆满，判断当前batch是否建立完毕，准备完毕就置batch\_ready为true，调用mk\_batch\_problem生成问题并加入batches列表中，清空problems列表，n\_nodes\_in\_batch清零。完成上述后再调用solver得到将这一轮新加入的数据是否可满足，并放入problems表中，更新batch中的节点数量等，进入下一轮循环。

mk\_batch\_problem就是将n\_vars，子句集，is\_sat等问题加入到一个总表中，构成一整组问题。添加长子句进总子句集时用shift\_iclauses偏移子句，就是将长子句中的子句向右或向左偏移前面总变量数的长度。

在上述循环结束后检查一遍有没有多余的问题，然后就创建文件夹等，生成文件名，将batch写入文件并存储。

test/train都是根据输入的周期数来调用函数neurosat进行测试或训练。Neurosat是整篇代码的主要函数，train/test都先初始化一个Neurosat对象，然后调用Neurosat里面的train\_epoch/test方法来进行训练\测试。

Neurosat自己定义了很多个方法，也调用了很多个方法。自己定义的方法按照功能分成3（4）个部分：build\_network，train\_epoch，和test。

首先是初始化Neurosat时用到的方法build\_network()。这个方法调用了建立神经网络的所有方法，下面我会逐个讲解。

init\_random\_seeds初始化随机种子，construct\_session建立session，tf.session是tensorflow控制和输出文件的执行的语句。

在讲解接下来的代码之前先介绍一下这篇论文的模型组成部分。模型架构如下：



子句从其相邻的文字的消息并更新其嵌入，接下来每个文字从其相邻子句及其补码接收消息，相应地更新其嵌入。

模型由两个向量（Linit，Cinit）、三个多层感知机（Lmsg，Cmsg，Lvote），两个归一化LSTM（Lu，Cu）构成。在每个时间步骤t，都有矩阵L（t），其第i行包含文字li的嵌入，和矩阵C（t），其第j行包含子句cj的嵌入。而对于两个LSTM，他们都两个隐藏状态Lh（t）和Ch(t)都初始化为零矩阵

消息传递机制为：



M是一个邻接矩阵，M转置用于将L(t)聚合到子句，M则把C(t)聚合到L（t）。

Flip是取文字补码的一个操作符。

在T轮迭代后我们得到输出L(T)，然后使用一个多层感知机计算文字的投票，该投票对每个文字认为可满足就投一个正数，不可满足就为负数，最后计算投票的平均值，得到对于整个问题是否可满足的一个预测logits。训练网络以最小化这个logits与真实标签之间的sigmoid交叉熵损失。

declare\_parameters函数初始化了上述的组成部分，用tf.get\_variable随机初始化了两个向量L\_init和C\_init，L和C分别代指文字和子句。然后通过调用MLP函数初始化多层感知机LC\_msg和CL\_msg。MLP随机初始化了权重和偏置。其中还定义了前向传播的方法，使用tf.matmul进行矩阵乘法计算wx+b。

然后用tf.contrib.rnn.LayerNormBasicLSTMCell初始化了两个层归一化的LSTMCell。这里的decode\_transfer\_fn是定义在util中的根据输入选择激活函数的方法，默认是relu。最后初始化文字投票层及其偏置。

declare\_placeholders声明了占位符，占位符就是一个空壳，在需要时再为其进行赋值。

pass\_message定义了消息的传递机制。首先初始化输出矩阵，然后使用LSTMStateTuple方法来得到LSTM的输出。LSTMStateTuple中存储了两个元素c和h，其中c是隐藏状态，h是LSTM的输出。通过while\_loop循环进行消息传递。其中while\_cond是继续循环的条件，while\_body是循环体内执行的操作。while\_cond在为达到给定的迭代次数前，循环继续。while\_body就是上述消息的更新过程，使L(t)通过多层感知机，用矩阵乘法乘上邻接矩阵，然后将结果与C(t)一起输入更新子句的LSTM的到C（t+1），再将C(t+1)输入多层感知机，并乘上邻接矩阵，再通过Flip得到L（t）的补码，一起输入更新文字的LSTM得到L(t+1)，迭代完毕后得到结果final\_lits和final\_clauses。

然后计算logits，与上述相同，先使final\_lits通过多层感知机得到投票的总体，然后分组并使用reduce\_mean计算均值得到logits。这里数据为什么要这么处理不懂。

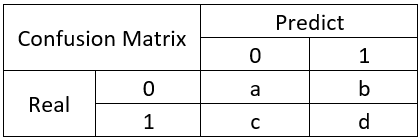
最后计算损失。通过tensorflow的这个函数计算给定的logits与标签之间的sigmoid交叉熵损失，下面这一块我也没有太明白是哪里的损失，可能是训练过程中训练每个变量的损失？总损失就是这两个损失的合。

计算完了损失就要进行优化。建立优化器，根据输入设置学习率的更新策略，用维持常量，多项式内衰减和按指数衰减等。然后使用tf.train.AdamOptimizer初始化Adam优化器，进行梯度下降。包括compute\_gradients计算梯度，使用gradients clipping让权重更新限制在一个合理的范围内，然后将处理后compute\_gradients的返回值作为参数用apply\_gradients对变量进行更新。

最后是init\_saver，将训练的结果保存起来

初始化完了神经网络，接下来就进入训练或者测试了。训练的方法train\_epoch和测试的方法test内容基本相同。

train\_epoch调用了一个problems\_loader从文件夹中读取保存着问题的文件，从中读取问题集。初始化了一个混淆矩阵，这个混淆矩阵用于评估模型的好坏。

ConfusionMatrix的形式如下

其用一个二维数组保存四种情况出现的次数：预测true，事实true，记为tt

预测true，事实false，记为tf

预测false，事实true，记为ft

预测false，事实false，记为ff

很明显我们需要预测与事实相符合，即tt与ff，而ft与tf则代表预测错误。模型的准确率就用tt与ff出现的次数的合除以总此处得到一个概率。

所以每得到一组logits与is\_sat就更新confusion Matrix，积累各种情况的出现的次数，来计算模型的准确率。

回到train\_epoch，通过build\_feed\_dict将读取到的problem中的参数赋值给占位符，使用sess.run运行优化器并读取logits和cost的计算结果，更新混淆矩阵，计算模型准确率。这样一个周期的训练就结束了，返回结果。

test方法基本和train\_epoch完全相同。只是test.py初始化完neurosat对象后，使用了restore获取保存的模型。

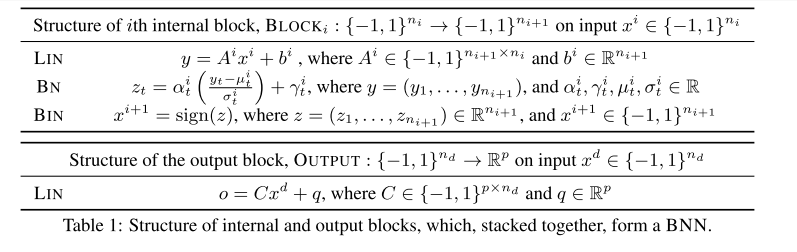
### 第二～五周 4月13日~5月10日

**《in search for a sat friendly BNN artifecture》**

分析神经网络的行为是深度学习最重要的挑战之一，而二值化神经网络BNN是一类重要的、可以用布尔逻辑等效表示的神经网络。这篇文章通过分析BNN的结构设计，讨论了它们如何影响基于逻辑的推理器的性能，并提出了一个对BNN结构和训练程序的修改，为SAT求解器提供了一个更简单的网络，又不牺牲主要任务的准确性。

在这篇文章里，他对SAT进行了以下定义。SAT公式是根据一组布尔变量{x1…xn}定义的，其中一个文字是变量xi或其补码。他的基数约束定义为，其中li是一个文字，k是一个常数。这个基数约束代表了一个等价的公式X’的变量至少有k个为真。

而BNN的结构如下图所示。

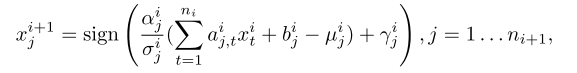


一个BNN由内含多个层的块组成，每个块将二进制向量映射到二进制向量。中间的每个块用BLOCKi表示，而最后一个块用OUTPUT表示，这个块将二进制向量映射到输出。每个BLOCKi都应用了三个变换：线性变换LIN，批量归一化BN，二值化BIN。

**基于逻辑对BNN进行分析**

对于每个BNN，都有一个SAT公式，使得该公式的解正好是BNN的所有有效输入/输出对的集合。这种翻译为基于逻辑的BNN分析打开了许多可能性。首先我们要将BNN用SAT进行编码。

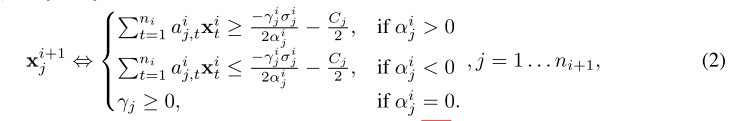
对于上述的LIN，BN，BIN三层，可以直接将一个块编码为其组成：



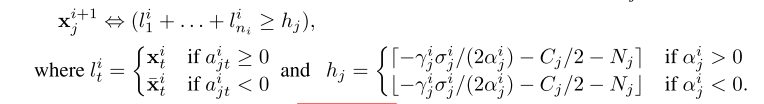
即将前一层的输出代入到下一层中。其中Xi是块输入，Xi+1是块的输出。为了获得SAT编码，我们需要消除二进制变量。用一个布尔变量套用公式来替代二进制变量，则原公式可转化为



用Cj表示，则最终可以得到布尔表示的公式：



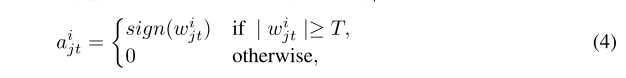
使用一个重言式（恒真式）将上述公式变为类似SAT的基数约束的形式得到：



其中Nj是第系数a是-1的行数。则可以将上述式子作为一个SAT的基数约束，用pysat工具来转化为SAT公式。

**从SAT的角度分析BNN**

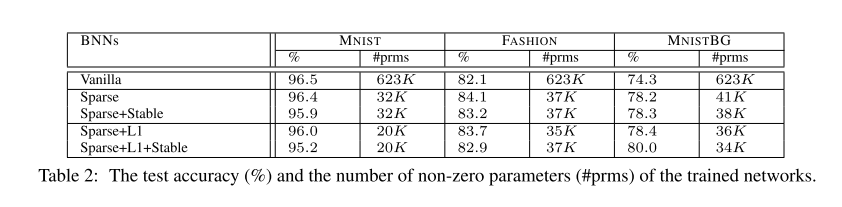
**神经元水平**：从单个神经元来看，A是一个LIN层的系数a的稠密矩阵，所有条目都必须是1或-1。A的高密度使方程3的基数约束中含有大量的变量，这导致了较大的SAT表示。同时神经元的值和基数约束之间有一个等价关系，这个等价关系可能会影响SAT解算器的推理过程。因此我们通过将A三元量化，来使A的系数变得稀疏，以减小SAT的编码大小，具体表示如下



w是BNN的用梯度下降来更新A的权值表，我们通过设置阈值T来使系数a出现0，这个阈值时从一个原始BNN的权重分布中获得的。

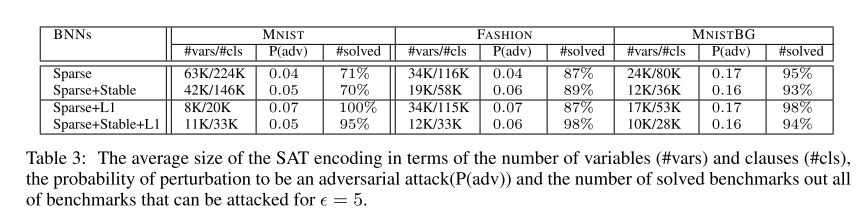
第二个问题是等价关系，解决方法是通过网络进行输入边界的边界传播。我们根据方程式估算每个线性约束的上下界，若两个边界都为正则xi为1，否则为-1，以此来稳定神经元。这样我们就获得一个易于SAT求解的BNN网络。

**实验**

首先，作者先比对了原始BNN以及各种经过修改的BNN的编码大小以及模型准确率的区别。如下图：

可以看出，使用了稀疏（三元量化）的网络相比原始网络，编码大小显著减小而网络准确率相等。而进一步使用L1正则化稀疏化后反而降低了网络的准确率，随着编码大小变得越小准确率变小的幅度更大。最后，稳定化会降低网络精度，但是会显著减小编码大小。

然后是放入了对抗性攻击的扰动的网络。



在扰动为5时，可以看出在存在扰动时稳定化能很大程度的减小编码大小。这些图表明，三元量化极大地提高了三个数据基的性能。而加入了L1正则化则会为MNIST数据基提供很大的帮助，而对后两个数据基只能稍微的改善性能。最后稳定方法用于加速求解速度（加速倍率达到3x至6x）。

**总结**

我从这个文章主要学到了三元量化（稀疏化）矩阵A对于在BNN上应用SAT求解的帮助（包括在保持精度的同时缩减编码大小），以及稳定化对于求解SAT问题的速度的帮助。

### 第六周 5月11日~5月17日

上周论文补充：

神经网络分析方法：1.对神经网络进行认证，训练一个满足给定特性的经过验证的网络，必须保证事先知道一组属性（并非总是可能的），在训练过程中强制执行一组属性会显著影响主要任务上的网络的准确性。

2．神经网络的验证，以经过训练的网络作为输入，验证是否存在某些属性。

从SAT角度分析BNN：

块水平上分析：一个块由一组约束对相同的变量进行变量，对这些变量进行多次计数，这提供了一个识别共享计算并利用它们获得更简洁编码的机会。共享计算的能力取决于矩阵A中出现1和-1的模式，这些系数控制变量在每个约束中是正是负，由于A在行之间出现1和-1时没有强制模式，因此对于BNN的标准体系结构来说共享量相当的小。

整个网络的水平上分析：将整个BNN看作一个区块链。BINBNN的公式的分块结构表明，解算器可以利用公式分解。其次，BINBNN公式有效地模拟了所有可能输入的网络函数。从公式的最后或第一个块开始，随着搜索的进行向第一个（最后一个）块移动，而不是在层之间跳转。这样的模式指导可能有助于更快地发现矛盾。

实验：

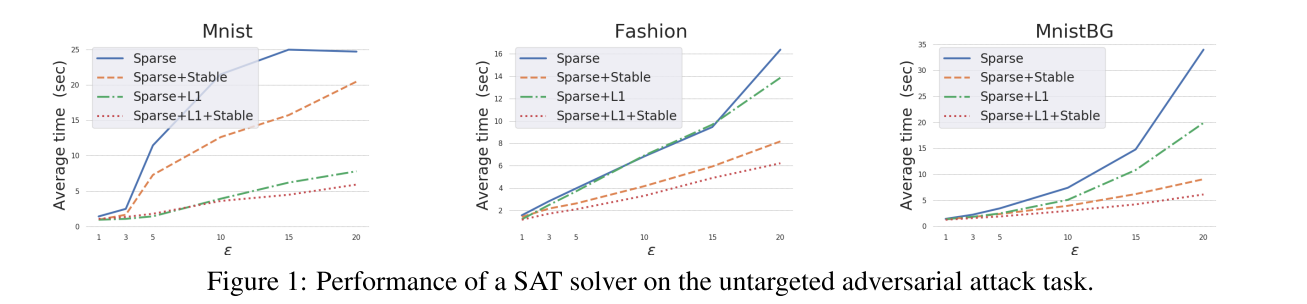


图1 展示了不同模型在不同epsilon下的平均求解时间。可以看出进行稳定化后的模型相比不应用的稀疏模型的求解时间显著变短，速度增加到3倍至6倍。没有原始BNN的结果是因为原始BNN使PYSAT编码时占用的内存过大，时间太长。

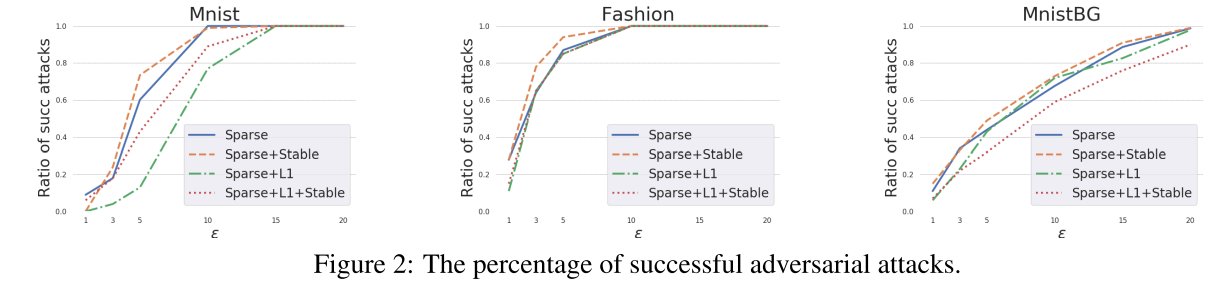


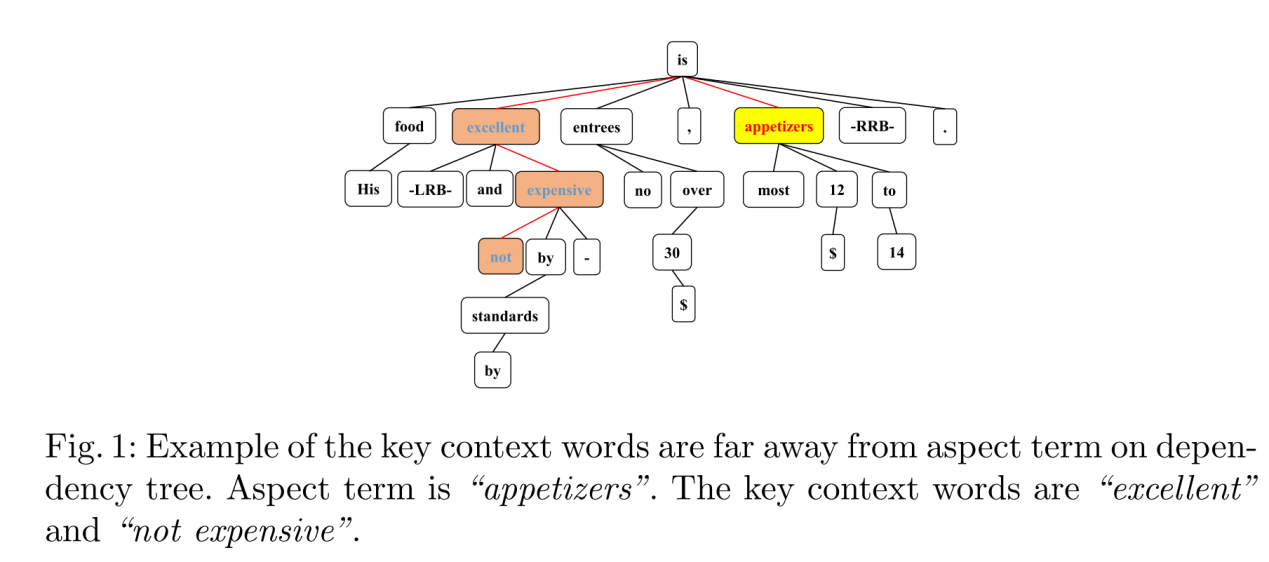
图2用成功进行对抗攻击的比率来表示模型面对对抗攻击的鲁棒性。可以看出随着epsilon增大这个比率稳定到1。在FASHION和MNISTBG下使用L1正则化器并不会显著改变网络的漏洞，但L1对MNIST有帮助.而在比较进行稳定化的网络和不进行稳定化的网络是，稳定化会导致漏洞增加。比如在MNIST数据集下，epsilon等于5时，只有13%的攻击样本成功对稀疏+L1网络进行了攻击，而又43%的样本成功对稀疏+L1+稳定化的网络进行攻击。

**MHSGCN：多跳句法图卷积网络**

基于方面的情感分类是情感分析中的一项基本任务，他的目的是分析给定句子中特定目标的情感极性。如句子“这家餐厅的食物很好吃，但是服务态度不好” 中，关于食物和服务的情感极性分别是正面和负面的。

总的来说，早期的传统方法采用注意机制来发现上下文和方面项之间的关联，获取单词之间的语义信息，但都是通过考虑单词的顺序来获得语义信息，而不考虑句子的句法结构。而有研究表明，从依存关系解析中获得的句法信息可以有效解决长单词的依存关系。为此，使用一种依赖树来通过图卷积网络根据依存关系聚合语法信息并捕获句子的结构信息。但是这也会出现一些问题。GCN捕获信息的距离取决于他有多少卷机层，但随着GCN层数增加，由于过度平滑问题，GCN不会带来其他的信息。而在依存树上，关键上下文单词与方面项的距离总是相距在两跳以上，这可能会导致关键信息无法有效的传递到目的地。

例子如下一个依存关系树：



该树对应的句子为：His food is excellent –LRB- and not expensive by NYC standards – no entrees over $30,most appetizers $ 12 to 14 –RRB-.其中appetizers是方面项，not expensive是关键上下文单词。可以清楚看到两者之间的距离大于2跳。

为了解决这个问题，不货到句子的句法结构，这篇文章的作者提出了一个多跳句法图卷积网络（Multi-hop Syntactic Graph Convolutional Network,简称MHSGCN）。这种GCN根据句子结构中信息传递的不同方式设计的，同时多范围注意机制计算模型中各层输出的注意系数，最后将各层输出线性组合，这种方式能有效加深网络的层数并避免过度平滑问题。

### 第七周 5月18日~5月24日

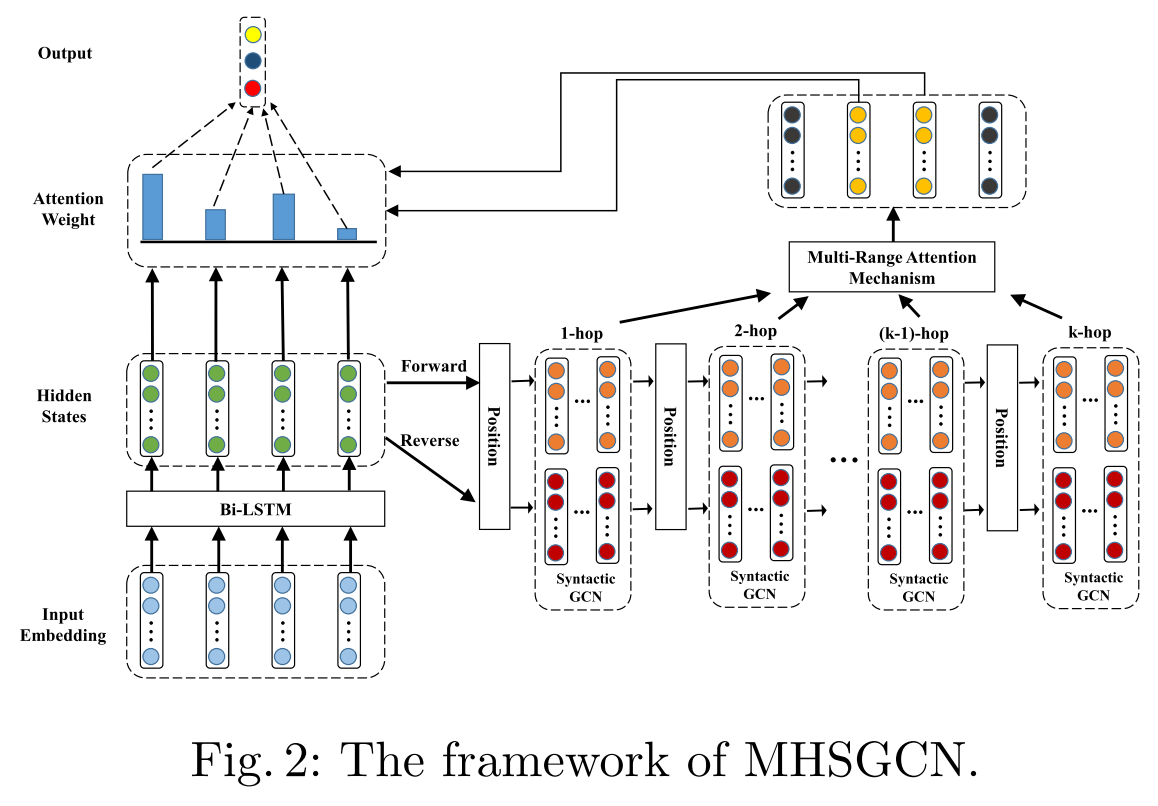
《Multi-hop Syntactic Graph Convolutional Networks for Aspect-based Sentiment Classification》

基于方面的情感分类是情感分析中的一项基本任务。 基于方面的情感分类的目的是分析给定句子中特定目标的情感极性。

传统方法基本都是通过考虑单词的顺序来获得语义信息的，而不考虑句子的句法结构。研究表明，从依存关系解析中获得的句法信息可以有效地解决长篇单词的依存关系。依赖树可以重新定义为图的一种特殊形式。图卷积网络（GCN）可以根据依存关系树聚合语法信息，并捕获句子的结构信息。 GCN可以捕获信息的距离取决于它具有多少个卷积层。但是大多数基于GCN的模型表明，两层GCN可以达到最佳效果。随着GCN层数的增加，由于过度平滑问题，GCN将不会带来其他信息。但是，在基于方面的情感分类的某些情况下，在依存树上关键上下文单词与方面项相距两跳以上。为了解决问题并捕获到句子的句法结构，作者提出了一个多跳句法图卷积网络（MHSGCN）模型。句法GCN是根据句子结构中信息传递的不同方式设计的，同时多范围注意机制计算模型中各层输出的注意系数，最后将各层输出线性组合，这种方式能有效加深网络的层数并避免过度平滑问题。

**方法**

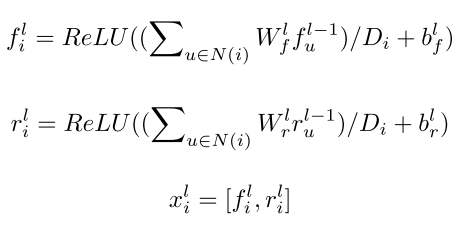
基于方面的情感分类的任务定义如下：给定n个单词的句子，以识别特定方面项的情感极性。 m个单词的方面项是从句子中的第（v +1）个单词开始的句子的子字符串。 多跳句法图卷积网络（MHSGCN）的总体框架如图2所示。



其中比较重要的两个部分是句法GCN的设计和注意机制，其他我个人不做记录。

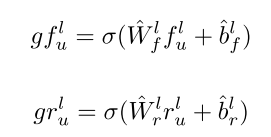
**句法GCN**：

句子结构可以根据依赖树1转换为图形式。 在依赖树上建立父子节点之间的双向连接边，得到邻接矩阵A∈R n×n，然后应用GCN捕获句子的句法信息。 但是，在依存树上，信息不仅可以沿语法相关弧向前传递，还可以沿语法相关弧反向传递。 因此，我们可以利用两个不同的权重矩阵来捕获不同的传递模式。 因此，第i个单词的表示形式通过以下方式更新：

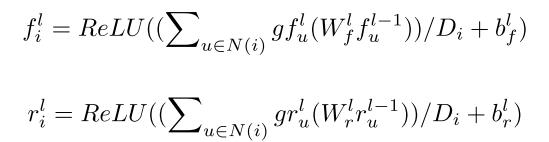


f是前向的结果，r是后向的结果，x是两者的聚合。D i代表第i个节点的度，N（i）代表节点i的邻居节点集。 W是可训练的权重矩阵，而b是可训练的偏差项（d n是隐藏状态的维）。

从所有邻居节点统一接收信息将导致大量无用信息被汇总。 为了解决这个问题，我们介绍门控制机制：

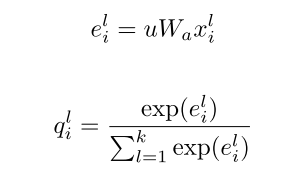


其中W是两个权重矩阵，b是bias项，西格玛是sigmoid函数。所以最开始的方程可以写成如下形式。

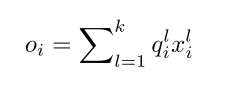


**多级注意机制：**

为了解决关键上下文单词与依存关系树上的方面术语相距太远的问题，以使关键信息无法到达目标单词。 我们介绍了多范围注意力机制。 我们保留语法GCN {X 1，X 2，...，X k}每一层的输出，其中k是层的编号。首先，将共享的权重矩阵W a应用于语法GCN每一层的输出，然后通过u和Waxi计算注意力系数，最后使用softmax归一化注意力系数。 注意，u表示在k跳范围内嵌入的上下文，需要学习。 计算过程可以描述为：

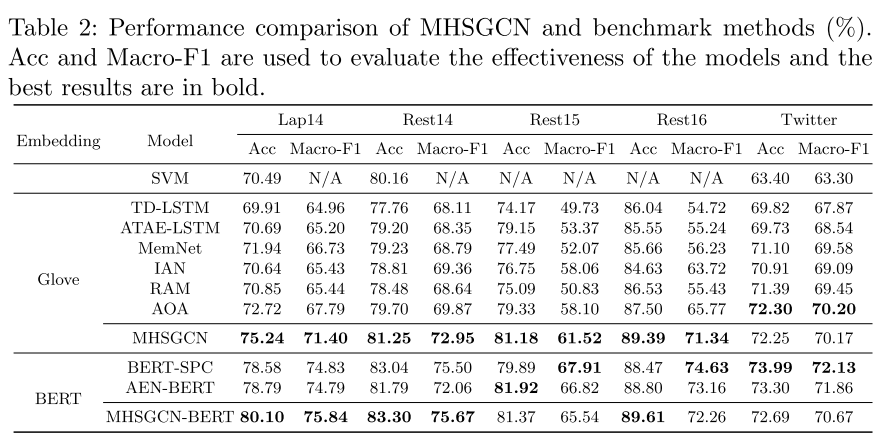


每一层的所有结点的线性组合如下所示：

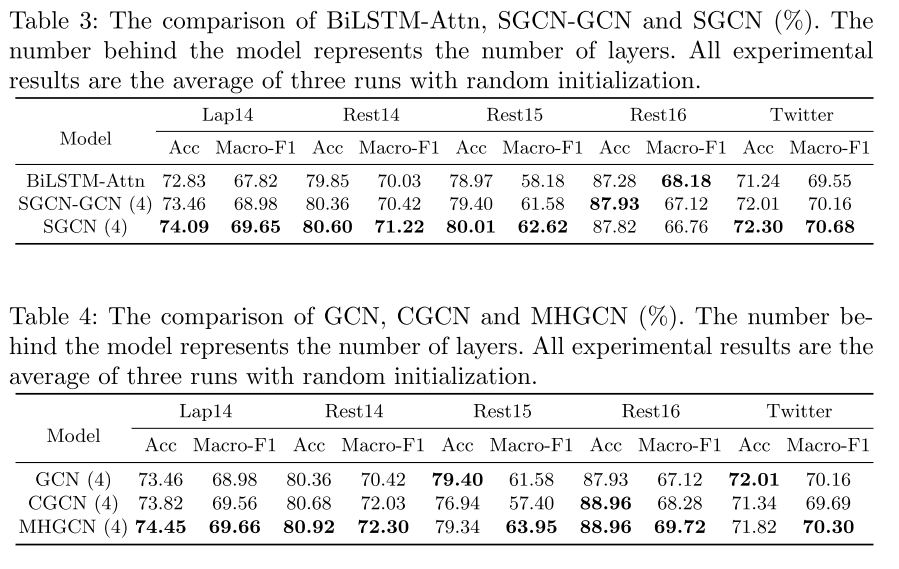


在捕获了多跳的信息之后，我们就能得到上下文的表示。

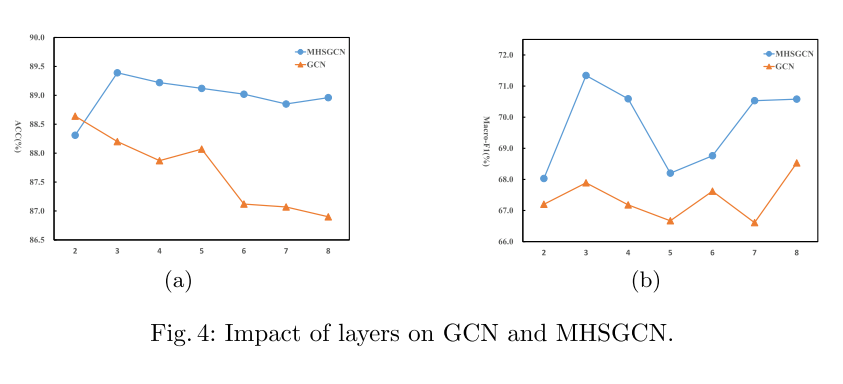
实验部分中，作者与其他的几种基准在一些数据集上进行了对比，还进行了消融实验分别证明了句法GCN，多跳注意机制的有效性，并分析了层数对网络学习能力的影响。



可以看出MHSGCN在绝大部分的情况下结果更好。



上面两张图分别是只有句法的GCN和普通GCN对比，和只有注意机制的GCN和普通GCN进行对比，分别证明了句法GCN和注意机制的作用。



上图证明了GCN的层数对于模型准确率的影响。

### 第八周 5月25日~5月31日

经过测试，由于百度AI Studio的linux系统锁定了sudo操作，导致neurosat项目的环境无法安装，所以决定改换阵营，去google colaboratary（自带免费GPU CPU，tensorflow，附赠15GB免费空间的在线编程平台），但是由于不用VPN无法使用，同时VPN炸了（正在联系寻找原因），故重新尝试在windows下安装环境。发现tensorflow已经不是主要原因，tensorflow经过重新尝试安装已经安装成功。

然后根据setup.sh里面的语句进行环境安装，原语句如下

sudo apt-get install python3-pip zlib1g-dev -y

sudo pip3 install --upgrade pip

sudo pip3 install tensorflow==1.4.0 scipy sklearn

cd python

git clone https://github.com/liffiton/PyMiniSolvers.git

cd PyMiniSolvers

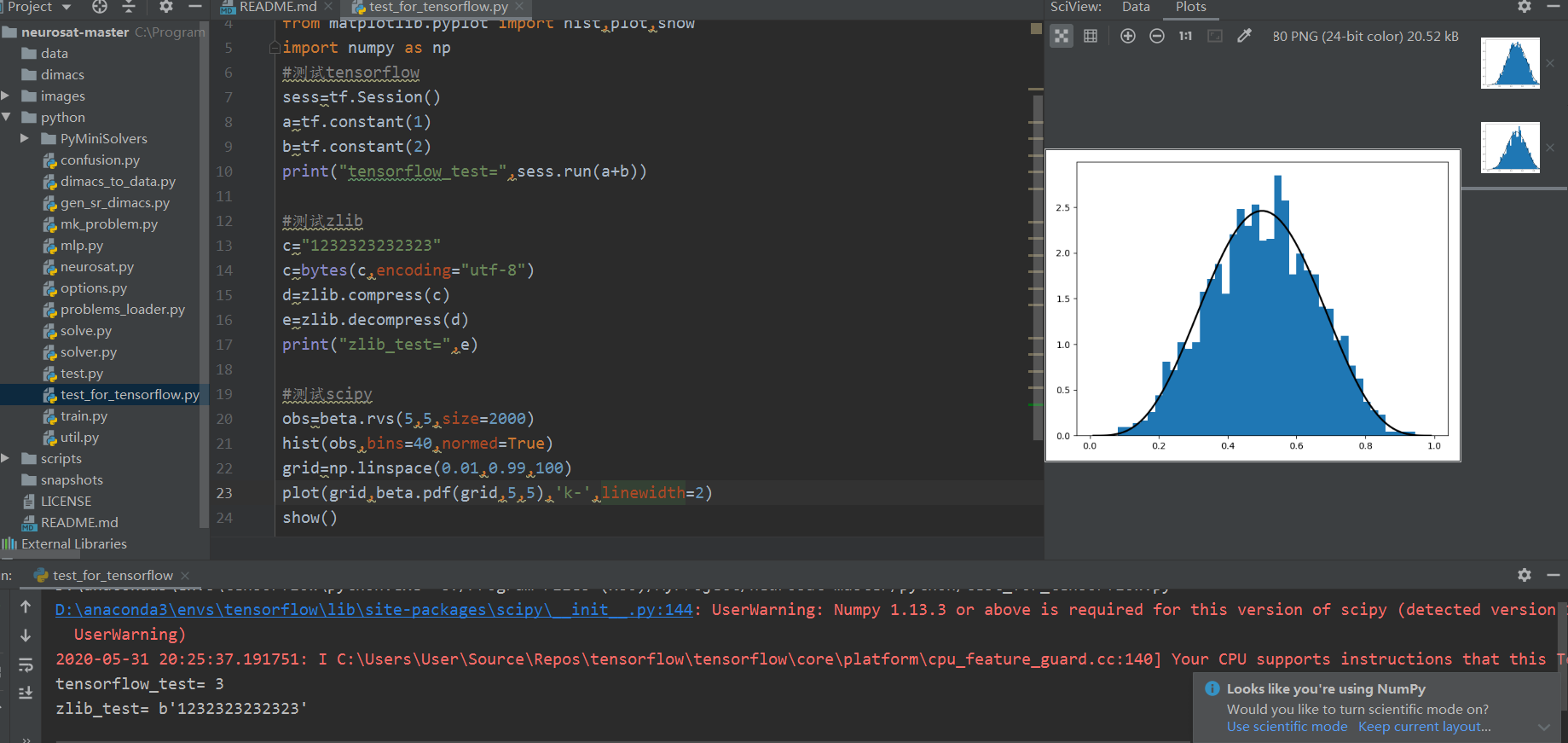
make

cd ../..

mkdir data dimacs snapshots

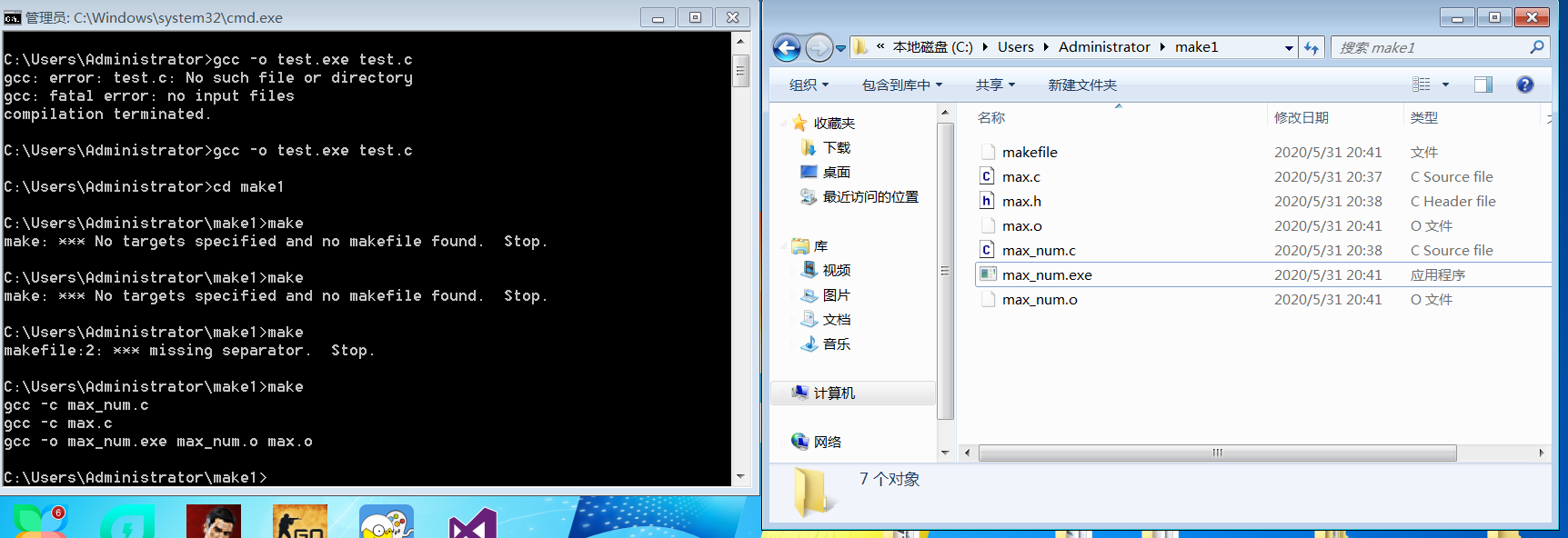
出现问题如下：

1. sudo apt-get install python3-pip zlib1g-dev –y 基本都是sudo的tensorflow语句，要换成windows的安装方式:直接下载zlib的安装包。
2. sklearn，scipy等通过anaconda安装完成并测试，如下（附带zlib及tensorflow）

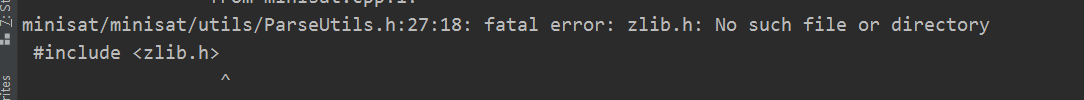


1. 去对应的文件夹clone PyMiniSolvers并make，但是make中出现了错误。

首先确定，gcc即make本身没有错误，测试如下：



在安装时make时出现的问题如下：



可是上面已经测试了zlib安装成功了，那么只能是切换环境以后的差异。

已经尝试了将zlib.h所在的目录加入了环境变量，或者是将zlib.h即相关文件拷到出现问题的文件的所在目录下，依然还是这个找不到zlib.h的问题。现在还在寻找原因并等待VPN恢复。

### 第九周 6月1日~6月7日

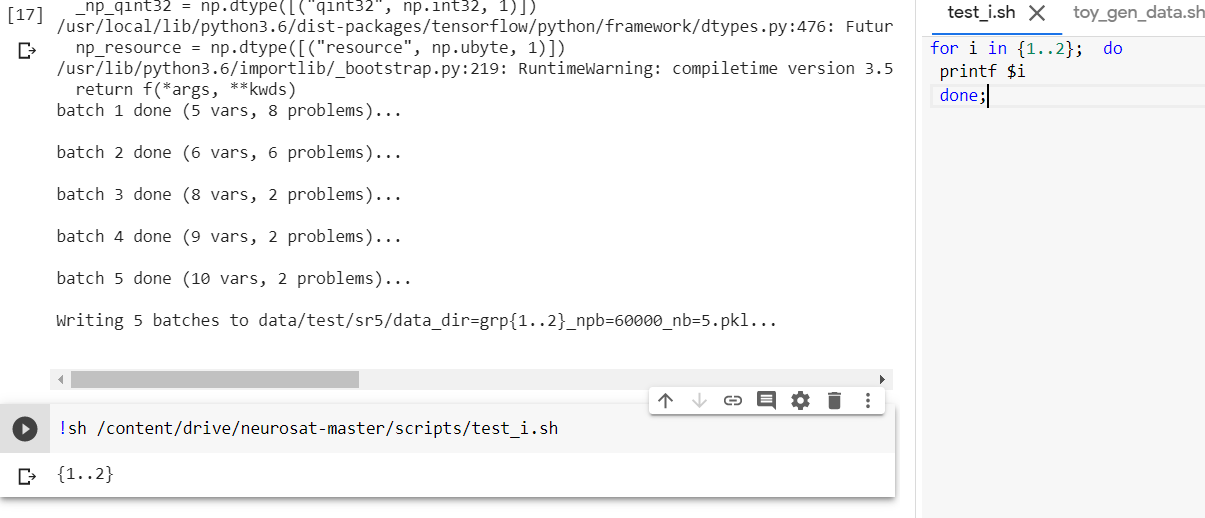
colab出现的问题：



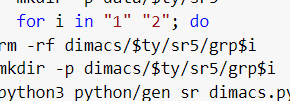
解决方法：protobuf与tensorflow冲突，删除tensorflow所有模块（包括gpu），删了protobuf再装，不行再尝试删掉keras

colab每次关闭以后重新打开需要重新安装环境，比较麻烦。

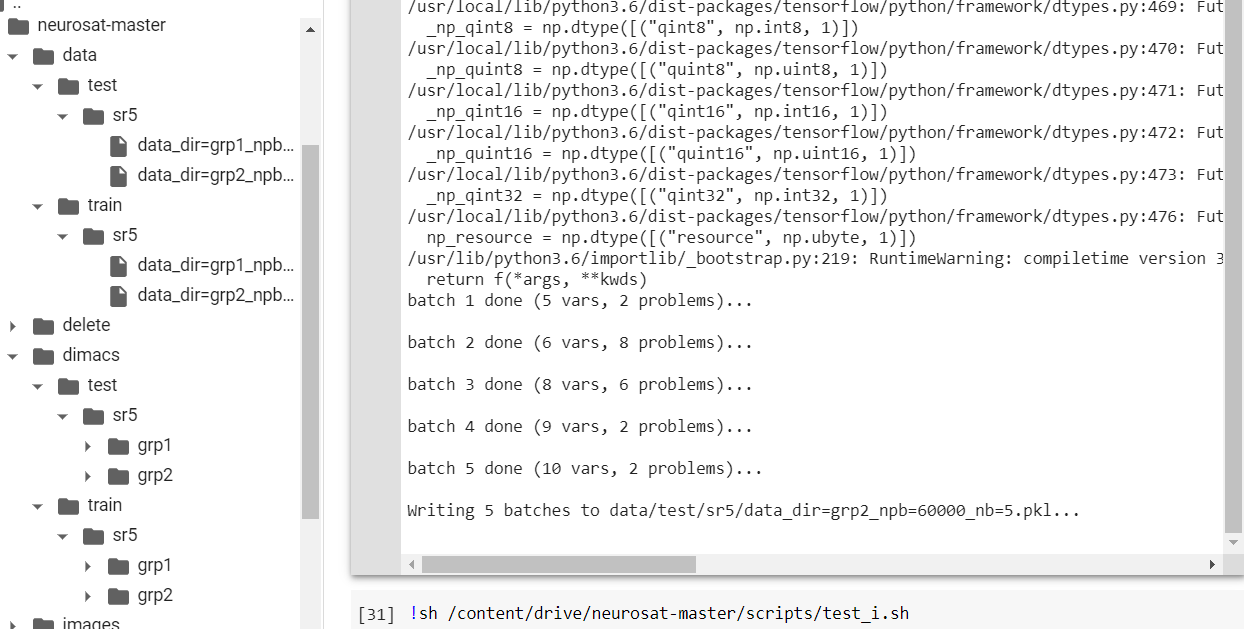
对于shell语句，for i in {1..2};do会使i={1..2}



改成这个形式使用

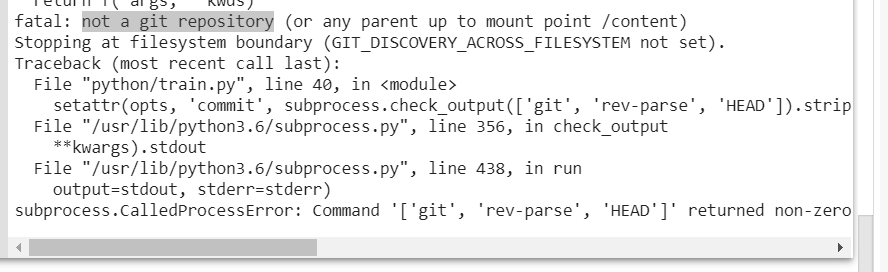


然后成功运行了gen\_sr\_dimacs.py与dimacs\_to\_data.py生成了训练数据与测试数据：



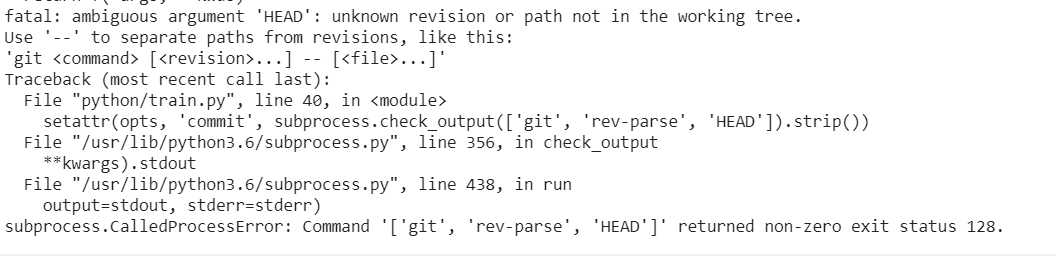
data是pkl文件，这是一种python的存储方式。而dimacs中存储的就是dimacs格式的cnf公式文件。

运行toy\_train.py出现问题：



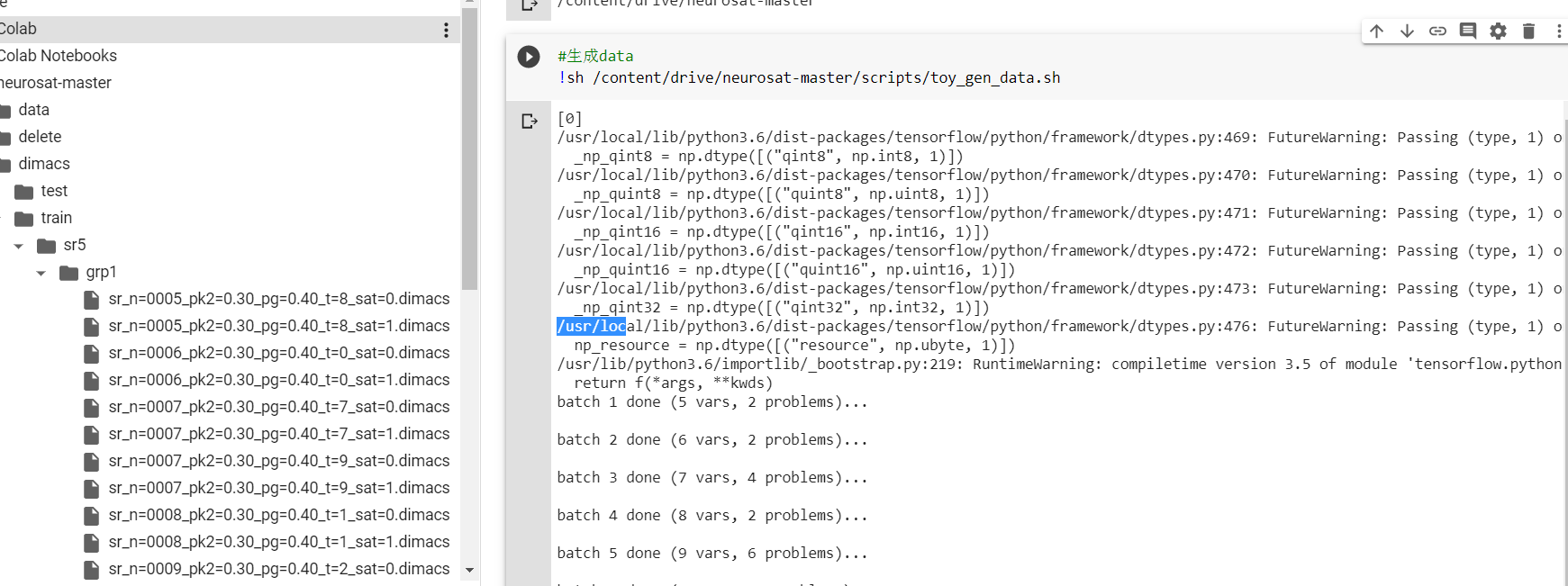
查了解决方法：git init，用完之后还是没解决。

### 第十周 6月8日~6月14日



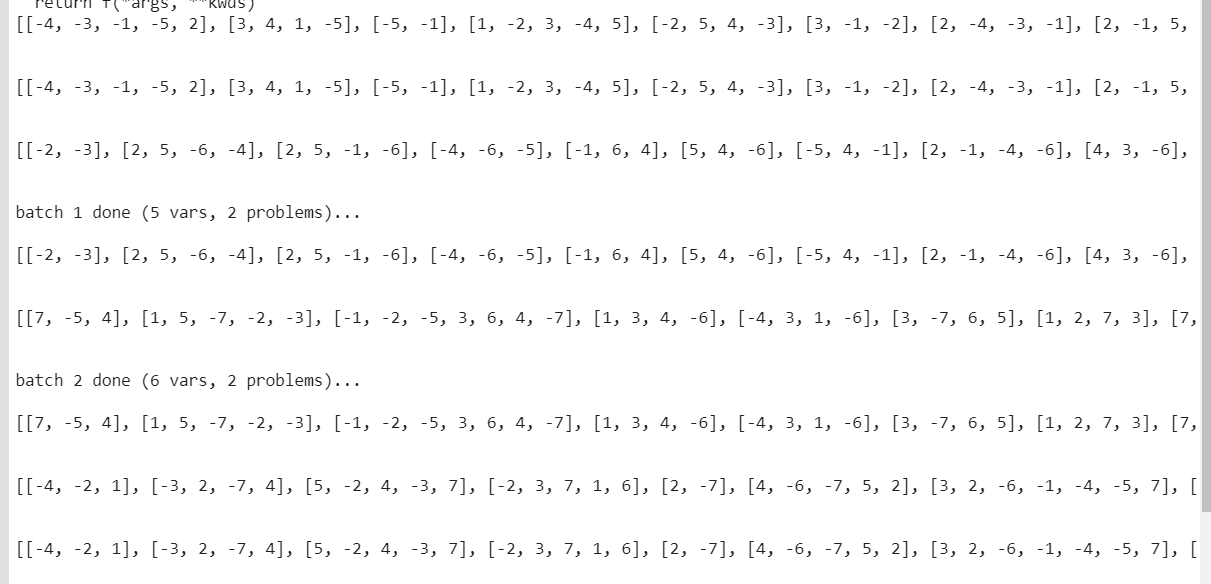
这里是git rev-parse出了错误，现在直接屏蔽掉有关的代码后可以成功进行训练与测试。

下面进行代码分析。



可以看到生成dimacs时，显示出了变量数（sr\_n），两个默认随机值（pk2与pg），以及是第几个生成的子句对（t）,是否可满足用sat=0/1标注，设置了生成10组子句对，n在5到10之间。生成dimacs的过程没有什么不懂的，不必细细研究。

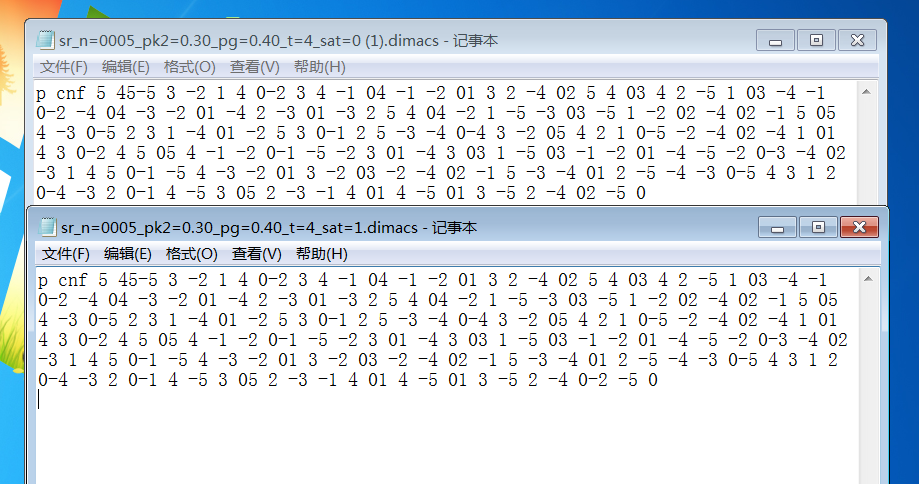
然后是dimacs转化为data的过程。



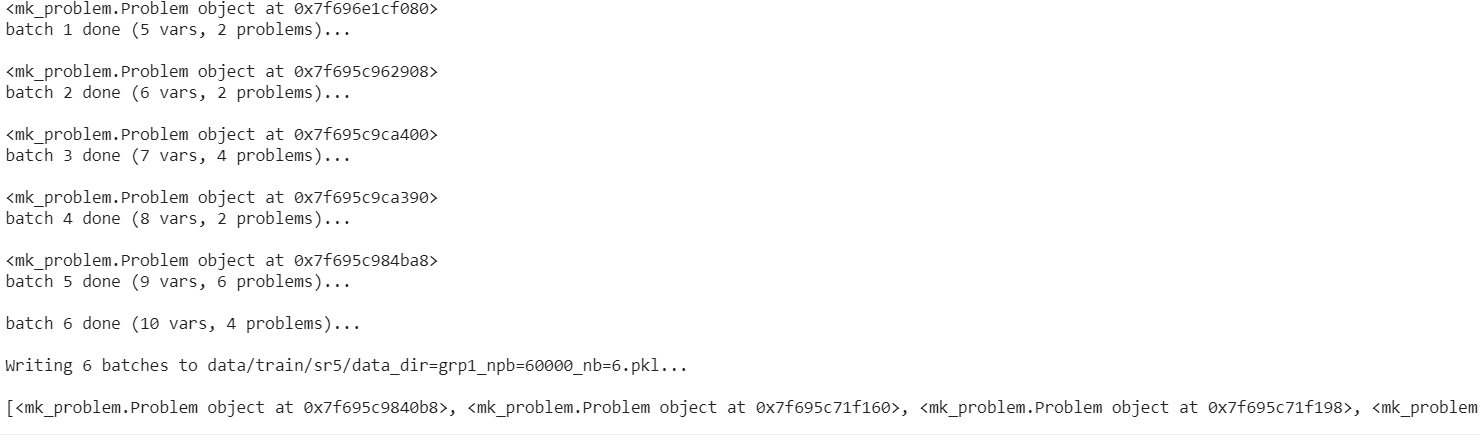
从中可以看到，每一个dimacs文件算作一个problem，而对于每一个batch是在新读取一个文件以后再判断加入这个文件的内容以后batch是不是满了，如果满了就先把batch中的写入到文件中去，如图的上半部分，读取了3个文件（3行，我设置了每读一个文件就换一个行），才把前两个文件中读取的problem写入文件。而读取的方法就是略过dimacs文件的标注，直接读取每一个子句作为列表中的一个项。这样 第一个部分生成dimacs到dimacs转换成data的部分就结束了。

### 第十一～二周 6月15日~6月28日

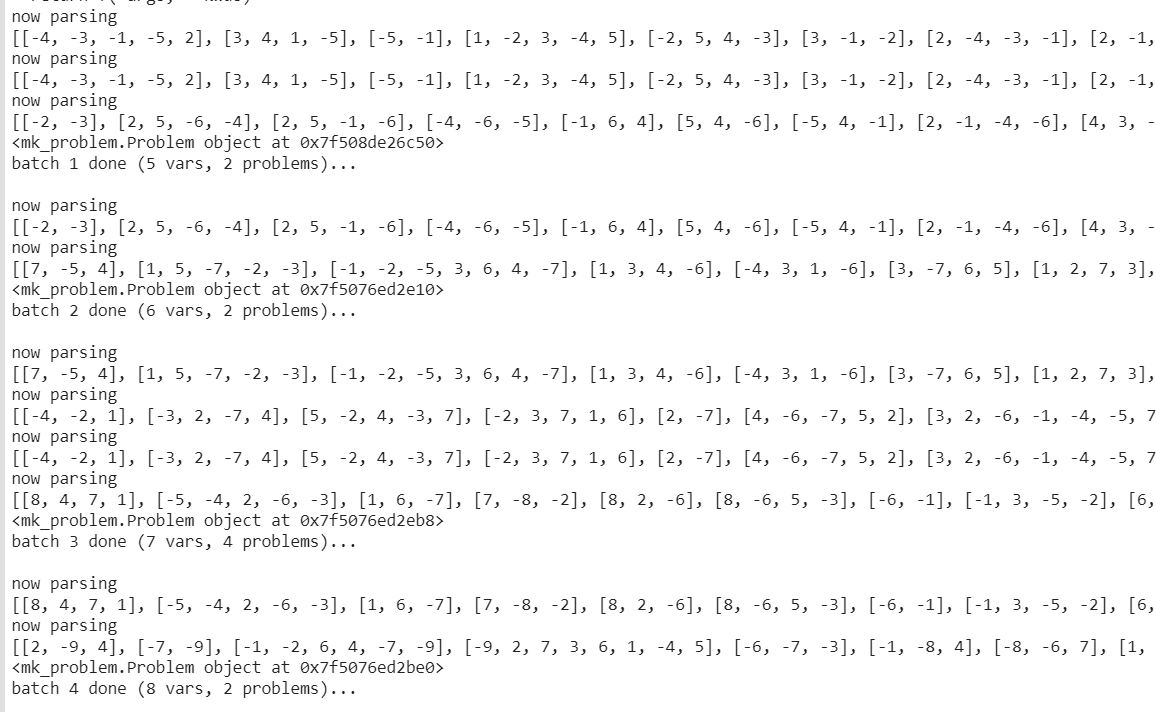
（1）生成的子句对：



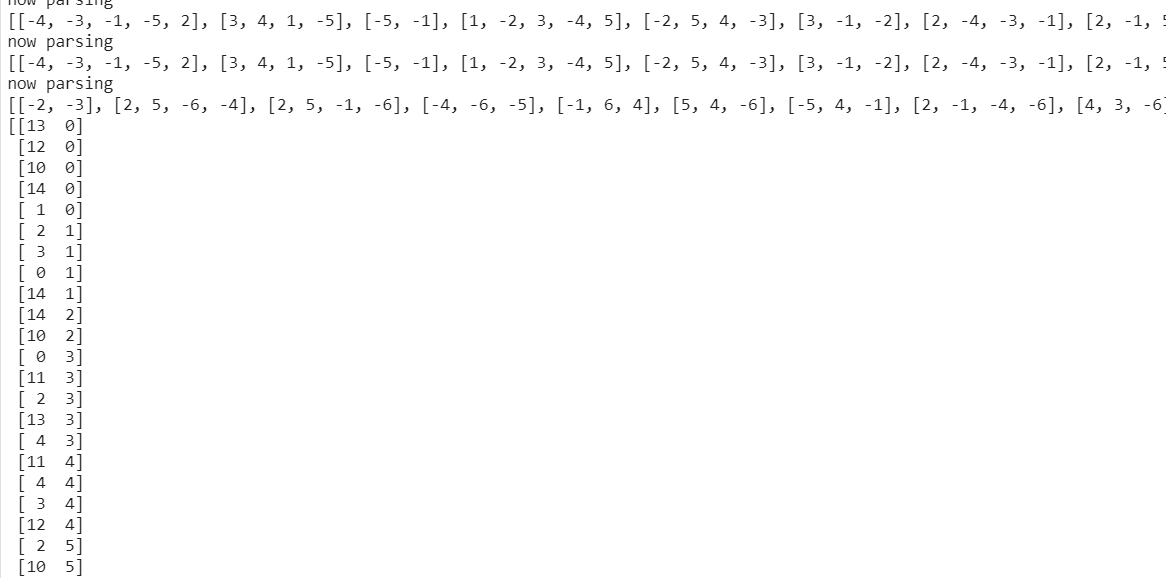
n=5，5个变量，随机系数0.3与0.4，t是第几个生成的子句，sat标注是否可满足。可以看出当读到首位为p时开始读取cnf公式，直接读取header[2]来获得变量数量，然后对之后的行读取。从文件的最后一行可以证明，就是他生成子句对的方式是对最后一条子句取反，来得到一个sat与一个unsat的子句集

（2）

problem是结构体，dimacs\_to\_data将通过mk\_problem得到的object存放的位置保存在了pkl格式文件中。所以直接print problem失败了。



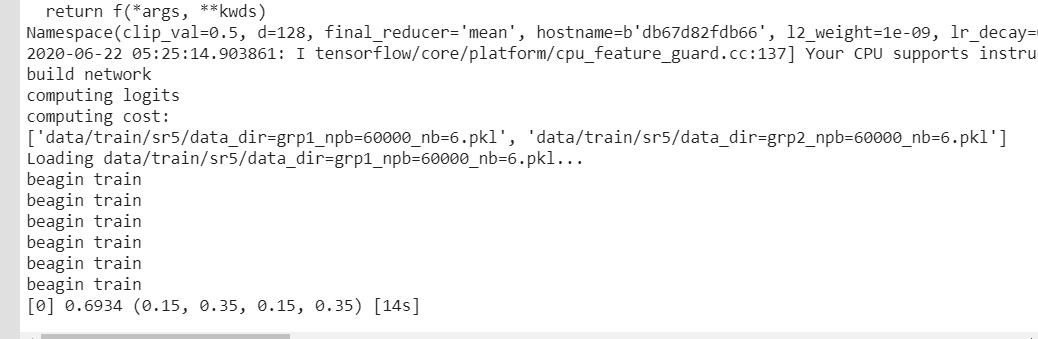
可以看出，一个problem就是一个文件。读取问题加入batch的方式采取了先读取再判断，在读取了当前问题后判断batch是否溢出，溢出把之前的batch写入再把这次读的加入batch。



dimacstodata的一个重要步骤，根据读取的子句来计算一个邻接矩阵。邻接矩阵是子句的表示。第一个数字标注是否为负，第二个数字代表变量（x-1），第三个数字代表第几个子句。

（3）

直接print tensor对象只能输出他的shape，得不到具体的值，所以试用了tf.Print,.Print,以及eval（）或者print sess.run（tensorvalue）来输出tensor的具体值，但是由于占位符以及tensorflow框架的特性，最终失败了。占位符未初始化时，声明可以照常进行，但是用sess.run来让tensorflow来计算具体值时，他会提示你必须feed这些占位符，于是我在传递参数的定义函数、计算logits等函数中，让这个函数运行时print（xxx正在运行），而在训练时也print了begin train，可以看到训练时进行了参数传递，但是那些函数没有被调用，tesorflow直接根据定义了的公式来计算了对应的值，同理更新也是如此，tensorflow直接去往优化器计算gradient的时候设计的所有tf变量出进行了参数更新，而不用自己定义。



首先顺序来看，L\_init首先是随机初始化的一张代表LSTM隐藏状态的一张表，他通过嵌入层进行升维，然后作为最终得到代表每个文字节点状态的表的输出格式存在。通过全连接层，LSTM层与表示子句的邻接的矩阵的输入来更新这个表的状态，代表了对这个问题是否可满足的一个整体判断，然后通过一个全连接层进行投票来表示这个判断，以达到训练网络来判断是否可满足的目的。