**研究生学习周报-研二上**

作者：蔡明炜

**目录**

[第一周 9月14日~9月20日 2](#_Toc23122)

[第二周 9月21日~9月27日 6](#_Toc5336)

[第三周 9月28日~10月11日 12](#_Toc6114)

[第四~五周 10月12日~10月25日 14](#_Toc5602)

[第六~七周 10月26日~11月8日 15](#_Toc27061)

### 第一周 9月14日~9月20日

《指导高性能SAT求解器的不满足核预测》

在这个论文中，作者训练了简化的NeuroSAT体系结构，使之可以直接预测实际问题中无法满足的核心。他修改了几个高性能的SAT解算器，用NeuroSAT预测变量在不满意的核心中出现的可能性的周期来定期替换其变量活动分数。改进的MiniSat在5,000秒的标准超时时间内解决了SATCOMP-2018上比原来多10％的问题。修改后的Glucose比原始解决方案解决的问题多11％，而修改后的Z3解决的问题多6％。当训练专门针对特定问题的分配时，收益会更大。以调度域中的难题为基准，经过改进的Glucose在一个小时的超时时间内解决的问题比原始解决方案多20％。

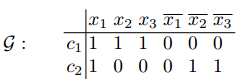
简介

作者主要是利用NeuroSAT来帮助在高性能SAT解算器中为实际问题提供可变分支决策。文中扩展的三个解算器minisat，Glucose，Z3都是用指数变量状态独立衰减和（EVSIDS）启发式算法，这其中包括保持每个变量活动的得分，并在得分最高的自由变量上分支。

神经网络结构：神经网络的基本组成部分是多层感知器（MLP），也称为前馈网络或全连接网络。MLP将固定din的向量作为输入，并为固定dout输出向量。它通过应用一系列（参数化的）仿射变换从x计算y，每一个都紧跟着一个称为激活函数的分量非线性函数。最常见的激活函数（我们在这篇文章中使用的）是整流线性单元（ReLU），它是正数的恒等函数，它将所有负数设置为零。

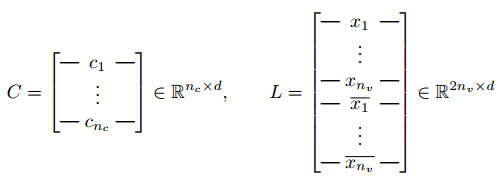
我们使用函数调用表示法来表示MLP的应用，其中MLP的不同参数是隐式连接的。如果是MLP，是向量，则我们用来表示把MLP M应用在x1和x2的连结的结果。出于性能原因，人们几乎从不将MLP应用于单个向量，而是将其应用于连接到矩阵中的一批具有相同维数的向量。如果，我们用来表示首先将X1和X2连接成一个的矩阵的结果，将M分别应用于k行，然后将k个结果连接回矩阵。

神经核的结构。我们用一个nc\*2nv的稀疏矩阵g来表示一个布尔公式CNF（nv是变量，nc是自居），其中第（i，j）个元素为1的唯一可能是第i个子句包含第j个文本。举个例子，我们用一个2\*6的稀疏矩阵来表示这个公式：

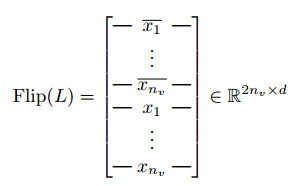


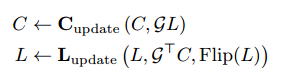
我们的神经网络本身是由三个标准MLPs：

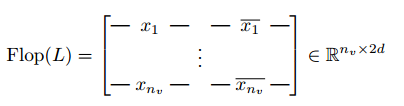


其中d是一个固定的超参数（d=80），网络的前向计算如下：首先，它初始化了两个矩阵，C的每一行对应一个子句，而L的每一行对应于一个文本：

我们将对应于子句c或文本l的行称为该子句或文本的嵌入。请注意，为了便于标注，我们将子句和文字与其嵌入部分合并在一起，因此符号c可以表示实际的子句或嵌入子句的c行。

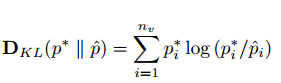
定义操作Flip将矩阵的前半行与后半行交换，以便在Flip（L）中，每个文本的行与其求反的行交换：

在初始化C和L之后，网络执行\消息传递的T次迭代（我们使用T=4），其中一次迭代包括两次更新。首先，每个子句根据其包含的文本的当前嵌入更新其嵌入：接下来，每个文本都会根据它出现在其中的子句的当前嵌入以及它的否定的当前嵌入来更新其嵌入：我们可以简洁地表示这些更新，并使用矩阵G和Flip运算符有效地实现它们：

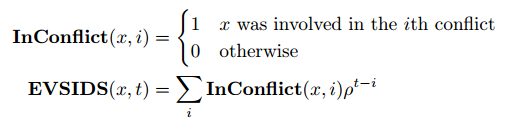
定义运算浮点运算，将矩阵的前半行与下半行沿第二轴连接起来，这样在Flop（L）中，对应于同一变量的两个向量被串联起来：

在T轮迭代以后，网络跳转了L来产生一个矩阵，然后使用了第三个MLP Vproj将V投影到一个nv维向量V帽中：

向量v帽是NeuroCore的输出，由每个变量的数值分数组成，它可以传递给softmax函数，以定义变量的概率分布p帽。在训练过程中，我们将变量上的每个标记位掩码转换为概率分布p∗，方法是为核心中的每个变量分配统一的概率，并为其他变量分配零概率。我们同时优化了三个mlp，以最小化Kullback-Leibler发散：



EVSIDS：变量x在tth冲突后的EVSIDS得分定义如下：



其中p<1是一个超参数，直观的说，EVSIDS对一个变量的评分衡量了这个变脸参与了多少冲突，最近的冲突的权重比之前的冲突更大。我们的方法是根据NeuroCore的输出定期重置这些EVSIDS分数。

NeuroCore在之后返回一个向量v帽，其中某个变量得分越高，说明NeuroCore认为相应的变量更有可能在核心中。我们把v帽变为一个概率分布，这是通过将v帽除以一个标准温度参数τ（值为0.25），并通过softmax。然后我们根据问题中变量的数量以及固定的常数κ（我们使用10^4）来缩放结果向量。 最后，我们立即替换所有EVSIDS分数：

请注意，衰减因子ρ通常很小（MiniSat使用ρ= 0:95），对于一阶近似求解器，每秒平均有10,000个冲突，因此这些分数仅在不到一秒的时间内衰减为0。 但是，通过将EVSIDS重新集中在搜索空间的更重要的部分上，这样的干预仍然可以产生强大的效果。 我们将整合策略称为定期重新关注，以强调我们只是重新关注EVSIDS，而不是试图取代它。 我们基于MiniSat的混合求解器仅每100秒查询NeuroCore一次。

### 第二周 9月21日~9月27日

《Can Neural Networks Understand logical entailment》

简介

本文的重点就是，**对神经网络能否学习逻辑公式的结构特征进行探索**，主要贡献有三个，**引入了一个用于训练和评估模型的捕获蕴含关系的能力的新数据集，并在该数据集上对现有模型进行了全面评估，最后提出了一个树型RNN的变体用以判断蕴含度。**

数据集

我们的数据集\* D由形式为（A，B，A（蕴含于）B）的三元组组成，其中A和B是命题逻辑的公式，若A蕴含于B则A（蕴含于）B则为1，否则为0。 例如，数据点（p∧q，q，1）为正，因为p∧qentailsq，而（q∨r，r，0）为负，因为q∨r不需要。 蕴涵主要是一个语义概念： 蕴涵主要是一个语义概念：如果每个模型中A为true的模型也都是B为true的模型，则A|=B。

我们对数据集提出了各种要求，以排除D +和D-之间的表层结构差异，这些差异可以通过“琐碎的”基线†轻松地利用。 我们在生成过程中对数据施加了以下高级约束，附录A中对此进行了详细说明：我们的类必须保持平衡，正例和负例中的公式必须具有相同的分布长度。 此外，我们试图确保在正例和反例之间在词汇或句法特征的分布上没有可识别的差异。 例如，如果正向蕴含（A，B，1）中的典型B表达式比负向蕴含（A'，B'，0）中的aB'公式具有更多的分离，那将是不可接受的。

如果仅对公式A和B进行抽样，评估是否为A∈ B，则正例和负例的公式分布之间存在显着差异，无需了解问题的结构即可学习利用哪些模型。 为了避免这些问题，我们使用了满足上述要求的另一种方法。 我们对公式（A1，B1，A2，B2）的4元组进行采样，使得：

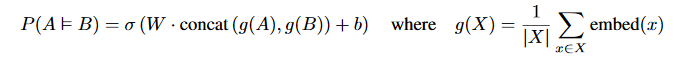


在这里，四个公式中的每一个都有一个正负和一个负负，因此，我们将正负示例之间的粗略结构差异减到最小

各基准和基线

3.1基线（比这个还差是不能接受的）

第一个是线性单词袋（Linear BoW）模型，该模型将每个符号嵌入到向量中并进行平均，以生成后续序列每一面的表示。 这些表示然后通过线性层传递：（σ是否是sigmoid有待商榷，应该是代指激活函数）



第二个是一个类似的结构其中最后的线性层变为一个多层感知机，称为MLP BOW：

在这两种情况下，由于它们只能通过对符号的贡献进行单独建模而不是通过对结构进行建模来捕获捕获，因此预期它们的性能有限，因为求和会破坏所有结构信息（包括单词顺序）。

3.2基准（作为标准方法评判其他方法的好坏）

我们在这里提出了一系列基准模型，不仅作为比较此数据集的新模型的基础，而且还比较和对比了在这个纯粹的句法问题上相当普遍的模型体系结构的性能。

我们将模型分为两类：编码模型和关系模型。 编码模型（以下指定除外）共同学习编码函数和MLP，使得给定结果A|=B，该模型表示：



从这个意义上讲，f生成了结果的每一面的表示，其中包含MLP决定蕴含度所需的所有信息。 相反，关系模型将观察表达式对，并可能通过遍历两个表达式或将一个表达式的子结构与另一个表达式的子结构相关联来做出决定。 这些模型表达了更一般的表述



3.2.1编码器基准

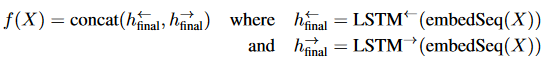
实施的第一个编码器基准是深度卷积网络编码器（ConvNet En-coders），类似于文本文献的卷积网络中描述的架构（Kalchbren-ner等人，2014; Zhang等人，2015; Kim等人。 ，2016）。



第二个和第三个编码器基准是LSTM（Hochreiter和Schmidhuber，1997年）编码器网络（LSTM编码器）及其双向LSTM变体（BiDirLSTM编码器）。 对于LSTM编码器，我们嵌入序列符号，并在其上运行LSTM RNN，忽略输出直到最终状态（只取最终状态的输出）



对于双向变量LSTM，两个独立的LSTM RNN LSTM←和LSTM→在序列上以相反的方向运行。 连接它们各自的最终状态以形成表达式的表示形式



到目前为止，所描述的基准测试并没有明确地以结构为条件，即使它们是已知的，因为它们被设计为从左到右遍历一个序列并隐式地对数据中的依赖关系进行建模。

第四和第五个编码基准是（树）递归神经网络（Tai等人，2015； Le＆Zuidema，2015； Zhu等人，2015； Allamanis等人，2016），也称为TreeRNN。 这些使用解析结构对逻辑表达式进行递归编码，其中树的叶节点（命题变量）作为可学习的矢量嵌入，然后每个逻辑运算符将这些嵌入值中的一个或多个组合在一起以产生新的嵌入。 例如，将表达式（¬a）∨b解析为具有叶子a和b的树，一元节点¬（输入了a的嵌入）和二元节点∨（输入了¬a和b的嵌入） ）。 继阿拉玛尼斯等。 （2016年），第四个编码基准是一个简单的TreeRNN（TreeNet编码器），其中每个运算符“ op”将其输入连接到向量x上，并产生输出



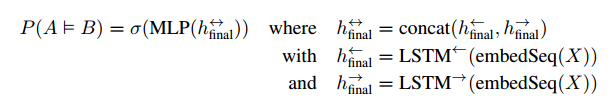
第五个也是最后一个编码基准（TreeLSTM编码器）是TreeRNN的变体，可适应LSTM单元更新。 这有助于捕获远程依赖关系并在树内传播梯度。 我们的实现遵循Tai等。 （2015），修改为具有TreeRNN的每操作参数（另请参见Le＆Zuidema（2015）和Zhu等（2015）的工作）

3.2.2关系基准

除了这些编码基准外，我们还遵循Rocktaschel等人的定义了一对关系基准。 （2015）。 我们将使用LSTM RNN或双向¨LSTM RNN遍历整个序列，但将左侧和右侧序列连接成一个单个序列，并用一个伸出的符号将其分隔开（有效代表 |=）。 对于LSTM变体（LSTM遍历），模型为：



对于双向情况（BiDirLSTM遍历），扩展为



3.2.3 变压器基准

我们还对Transformer模型进行了基准测试，也被称为Attention Is All You Need（Vaswani et al。，2017），这是一种序列到序列模型，可实现机器翻译的最新结果。 就像在关系LSTM模型中一样，我们连接并嵌入了序列，但不是通过一个突出的符号来分隔序列，而是在该嵌入中为右序列添加了可学习的bias。 这增强了Transformer添加计时信号以区分不同位置的符号的方法。 然后，我们解码长度为1的序列，并应用线性变换以获得最终的蕴藏量预测对数。

3.3 PossibleWorldNet

在本节中，我们介绍我们的新模型。 受蕴含的语义（模型理论）定义的启发，我们在TreeNets上提出了一个变体，用于评估不同“可能世界”中的一对公式。

蕴含首先是一个语义概念。给定一个有关世界的集合W



在这里SAT：World ×Formula → Bool ，意指某个公式是否在特定环境world中被满足。

我们首先定义一个产生证书的sat变量，然后定义另一个作用于实际值的变量。首先，定义sat2：World ×Formula → {0，1}：

运用sat2，我们可以重新定义 蕴含 为：

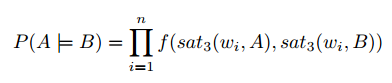


假设我们有一个世界的有限集W={w1,...,wn}，则我们可以定义P（A|=B）为：

这意味着，对于W中的元素都要有A蕴含于B，则P（A|=B）为1，否则为0，即不蕴含

我们将通过用连续函数替换sat2和≤号来产生命题1的松弛。假设我们有一个sat2的变体，他产生实值得向量：



假设我们有一个方程f：Rd×Rd → [0,1]将≤推广到实值向量，于是我们可以重写蕴含为：

在我们得神经模型中，f是由一个简单得线性层使用可学习权值Wf和bf实现的。

我们使用一组随机向量来表示我们的世界{w1,...,wn}，其中wi ∈ Rk是长度为k的向量，该向量的长度均匀地随机绘制。 我们使用简化的TreeNN（请参阅第3.2节）实现sat3，如下所述。 由于sat3取决于我们当前正在评估的特定世界wi，因此我们向TreeNN添加了一个附加参数，以便嵌入程序可以访问当前世界wi。 我们添加了一个额外的权重矩阵W4op，以便命题变量可以了解当前世界的重点。 如果公式的格式为op（l,r），其中op为零元（命题变量），一元（例如，否定）或二元（例如，合取），并且l和r是表达式的组成部分的嵌入，然后



其中x=concat(l,r)

(这里不必多想，我的理解是，将公式A写作一个依存树，叶节点是变量，一元结点是非，2元结点是合取或者析取。当节点是叶节点时，用权重W4op学习输入wi，更新叶节点嵌入；其他结点（1元或2元）则用下层传来的表示，连接表示并用权重矩阵和bias来产生新的表示，依次递归直到达到最高结点得到结果)

（然后分别得到关于wi的A的sat3与B的sat3，用线性层f得到关于这个wi下A是否蕴含于B的一个判断，得到所有的判断的乘积，作为网络判断在整个环境W下A是否蕴含于B的结果）

为了评估A|=B是否成立，可能世界网络生成了一组想象中的“世界”，然后在每个这些世界中评估A和B。这是“在可能世界上卷积”的一种形式。正如我们将在第5节中看到的那样，随着我们增加虚拟世界的数量，模型的质量将稳定增长。

该体系结构受语义（模型理论）方法检测蕴含的启发，但是它并未对命题逻辑或一般形式的逻辑进行任何约束编码。在多个世界中对句子进行评估并将这些评估组合到一个产品中的过程，正是“蕴含”的含义；因此我们推测，这样的架构原则上应同样适用于其他逻辑（例如，直觉逻辑，模态逻辑，一阶逻辑）以及自然语言句子中的非形式蕴涵。

将这些向量抽象为“世界”的特殊解释，此方法生成具有共享权重的模型的n个副本，每个向量wi一份。每个零元运算符在wi上学习不同的投影。它通过组合两个表示的线性层进行预测，然后将预测的乘积作为整体预测。

### 第三周 9月28日~10月11日

写了一下data的生成方式。

首先获得一个cnf文件集合，用shell将这些文件复制到目录/dimacs，然后调用newline\_cnf.py为这些文件添加一行随机单元子句，具体做法如下：shell中：

cp -r cnfformula/$ty/\* dimacs/$ty

python3 python/newline\_cnf.py /dimacs/$ty

在newline\_cnf.py中是这样为cnf文件添加随机单元子句的：

mk\_out\_filename:创建输出文件名，由于文件操作是在当前目录执行的，就算你打开了其他目录的文件，直接关闭文件也只会在当前目录创建一个同名文件并保存你修改的内容（不包含原内容），故要创建输出文件名为读取的文件名+其所在的目录组成，这样才能精准修改文件。

prefix = "%s/%s" % \

(opts.dimacs\_dir, filename)

parse\_dimacs：读取cnf文件，跳过开头为c的子句，直到开头为p的文件开始是正式的cnf子句内容，而生成随机子句所需的变量数就在剔除空格后的第三个位置，故：

header = lines[i].strip().split(" ")

assert(header[0] == "p")

n\_vars = int(header[2])

读取了变量数n\_vars后，就开始为文件加入一个随机子句。

用random生成0到n\_vars中的一个数：

i = random.randint(1,n\_vars)

然后打开文件，并添加子句：text= str(i) + " " + "0"

生成的子句满足dimacs要求，单元素i以及dimacs格式以0结尾。

关闭文件完成修改。

然后使用d4分别重定向原版cnf和添加了子句的新cnf的模型计数，左右输出都保存在文件中：

./d4 cnfformula/$ty/CBS\_k3\_n100\_m403\_b10\_$i.cnf > modelcount/cnf/$ty/CBS\_k3\_n100\_m403\_b10\_$i.txt

./d4 dimacs/$ty/CBS\_k3\_n100\_m403\_b10\_$i.cnf > modelcount/dimacs/$ty/CBS\_k3\_n100\_m403\_b10\_$i.txt

只需要取首字符为s的那一行，即为模型计数。

### 第四~五周 10月12日~10月25日

写了一些代码：

首先写了如何将一个cnf文件根据一个随机生成的目标变量将文件根据目标变量的赋值生成两个子文件的程序 resolve\_cnf.py:

输入一个目录，对该目录下的所有cnf文件进行名字读取，然后根据名字生成存放分化文件的路径，分别为new\_formu\_t和new\_formu\_f，他们分别存放了令目标变量i=t和f的子文件。

然后读cnf文件的行，对所有头文字为c的行进行跳过，直到头文字为p为止，读变量数n\_vars。

根据n\_vars生成一个1到n\_vars之间的随机变量i，将读的文件的所有行，变量数和目标变量i输入给分化cnf公式的函数change\_dimacs。

change\_dimacs用lines.split(“ ”)根据空格对每一行的文字进行提取，并修改p开头那行中的变量数为n\_vars-1，然后定义两个clause\_deleter\_T和clause\_deleter\_F。现在开始对文件进行操作。

对于有目标变量的行，只会面临两个选择：若变量i的赋值及其符号相同，则删除改行，否则删去目标变量，保留其他变量。故只需对所有行进行一次操作，然后用两个表clause\_deleter\_T和clause\_deleter\_F保存公式i=t和公式i=f分别要删除的行，然后对处理后的表分别进行copy，并根据两个删除行表的倒序索引进行行的删除（正序删除会导致删除行出错）。

对表的处理包括：判断每一行中的每一个除行末的0的元素是否大于目标变量。若大于则正元素减一，负元素加一；若元素等于目标变量，则保存这个文字在这行中的位置，并判断其是否大于0或小于0来在考虑是否在两个删除表中加入该行的行号。

最后，用list.copy（）复制处理过的lines为lines\_T和lines\_F，分别从后往前删除行，并分别计算删除的行数，用读取的子句数减去删除的行数作为新文件的子句数，将两种文件分别写进文件夹里即可。（修改一行中的某个元素值用split（“ ”）将某行依据空格分隔提取到list中，根据位置改变list中目标元素的值，再用” ”.join（list）生成一行str）。

然后定义了一个scripts重定向了d4计算模型计数的结果到modelc\_t与modelc\_f文件夹中。

### 第六~七周 10月26日~11月15日

L\_init:一个128维元组

C\_init\_t&f：相同

MLP:ws有4个[128,128]的元组，wb有4个[128]元组，而L\_predict有3个[128,128]和一个[128,1]的ws和3个[128],一个[1]的wb

LSTM的神经元数为128

1. unpack（邻接矩阵）：dense-shape=[文字数，子句数]，indices指出哪里有值，np.ones指出这些值是1，其他位置为0.

tensorflow中的tile()函数是用来对张量(Tensor)进行扩展的，其特点是对当前张量内的数据进行一定规则的复制。最终的输出张量维度不变

tf.cast：数据格式转换。tf.div

tf.till:mutilp矩阵[shape1,..,shapen]中shape1代表第一个维度上扩展shape1倍

L\_output的维数：[文字数，128]

C\_output:[子句数，128]

all-pred:[文字数，1]

all-pred-join:[batch数，每个batch的变量数，2，10]

Logits[batch数，分类数（10）]

Tf.reshape:将输入的张量变为shape的情况