**研究生学习周报**

作者：蔡明炜

**目录**

[第一周 9月16日~9月22日 3](#_Toc41852100)

[第二周 9月23日~9月29日 8](#_Toc41852101)

[第三周 10月7日~10月13日 12](#_Toc41852102)

[第四周 10月14日~10月20日 16](#_Toc41852103)

[第五周 10月21日~10月27日 21](#_Toc41852104)

[第六周 10月28日~11月3日 24](#_Toc41852105)

[第七周 11月4日~11月10日 29](#_Toc41852106)

[第八周 11月11日~11月17日 33](#_Toc41852107)

[第九周 11月18日~11月24日 36](#_Toc41852108)

[第十周 11月25日~12月1日 37](#_Toc41852109)

[第十一周 12月2日~12月8日 39](#_Toc41852110)

[第十二周 12月9日~12月15日 41](#_Toc41852111)

[第十三周～第十七周 12月16日~1月19日 42](#_Toc41852112)

[第十八周～第二十一周1月20日~2月16日 44](#_Toc41852113)

[第二十二周～第二十五周3月23日~3月29日 47](#_Toc41852114)

[第二十六周 3月30日~4月5日 49](#_Toc41852115)

[第一周 4月6日~4月12日 50](#_Toc41852116)

[第二～五周 4月13日~5月10日 54](#_Toc41852117)

[第六周 5月11日~5月17日 57](#_Toc41852118)

[第七周 5月18日~5月24日 59](#_Toc41852119)

[第八周 5月25日~5月31日 63](#_Toc41852120)

### 第一周 9月16日~9月22日

**李宏毅老师的深度学习视频部分：**

***part1：线性回归***

线性回归的主要思路据视频中描述如下：

第一步，找一个函数集合（模型）；第二步，定义一个损失函数L（f）来判断函数（模型）的好坏（寻找最小值）；第三步，运用梯度下降优化模型。

首先建立一个模型，在初始时可以设为y=bi+wi\*xi (i为下标),损失函数定义为：

L(f)=∑ (yⁿ- (b+w\*xiⁿ) ) ²

(n为上标，i为下标，其中yⁿ与xiⁿ为给定训练数据集中的n个样本）

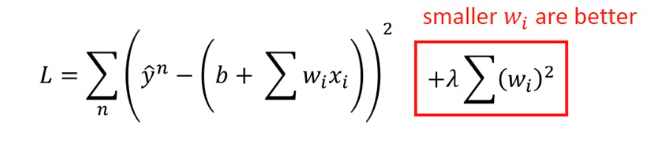
任取L（f）中的一点（w0，b0），在该点分别取偏导，根据偏导的值来移动w与b的值。若w0、b0的偏导为正，说明函数正字上升，为了取最小值，则应该减少w与b；否则就要增加。

ŋ是学习率（learning rate），需要我们自己调整。有关更多梯度下降的内容在下一部分会有阐释。

若是想要模型更加符合实际，我们可以提升模型的容量，加入跟高的X次方，x的次数越多，模型的表示越广。但是这种情况可能会出现过拟合的问题，这一种情况会在第三部分阐释。所以若是单单提升容量无法找到合适的模型，可以考虑往模型中添加更多的参数来设立模型。

还有一种方法是正则化，重新设计损失参数。



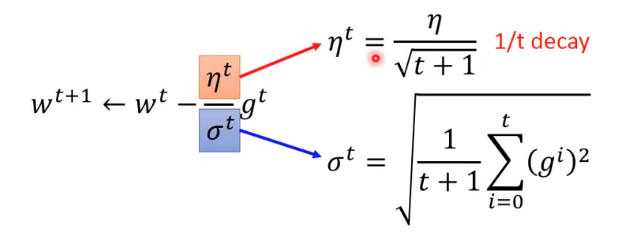
当wi越小，输入的变化对结果输出的影响越小，损失图像越平滑。所以要找到一个大小适合的，使图像尽量平滑。

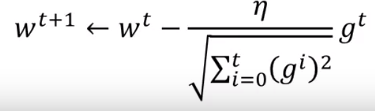
***part2：梯度下降***

Tip1:小心调整learning rate

在进行梯度下降时，如果learning rate较大，可能会发生直接跳过局部最小值，导致错误判断最小值的位置产生误差，所以倾向于随着参数更新减小learning rate。

一种办法是Adagrad算法：

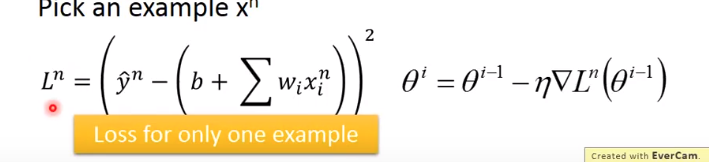




其中σt是所有微分值的均方根。

Tip2：随机梯度下降（Stochastic grradient descent，SGD）

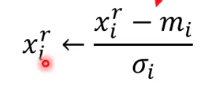
能让训练速度更快。每次只取1个样本，在计算损失时只计算该样本的损失，并且更新参数时只根据该损失来更新参数。



Tip3：特征缩放(Feature Scaling)

让不同的特征的分布相同，更新参数会更加容易。

常见的做法：对所有的样本的第i个特征计算平均值mi与标准差σi，更新所有样本的第i个特征，公式为：



其中r是样本的序号，i是该样本的第i个特征。更新完后，所有样本的第i个特征的平均值是0，方差是1。

Tip4：梯度下降的缺点

在极小值或者微分值是0的地方就会停止，不一定会到达全局最小值。

***part3：误差的主要来源***

我们在做机器学习时，误差主要来源于偏差（bias）与方差（varience）。偏差描述的是算法的预测的平均值和真实值的关系，而方差描述的是同一个算法在不同数据集上的预测值和所有数据集上的平均预测值之间的关系。方差和偏差是有冲突的，称之为变差方差窘境（bias-variance dilemma）。一般来说，当模型容量小的时候，bias较大而varience较小；当容量较大时，bias较小而varience较大。所以我们在寻找目标模型时，要寻找一个平衡点，使偏差方差的总体的泛化误差最小（最理想状态是低bias与低varience，但是一般不存在）。

一般来说，当你建立的模型与训练样本不能拟合时，bias较大；当模型能与训练样本拟合但是不与测试样本拟合时，varience较大。也就是说当欠拟合（underfitting）时，bias较大；过拟合（overfitting）时，varience较大。在判断完当前误差的主要因素后，就可以使用一系列操作来降低误差。

当bias较大时，我们可以重新设计模型，加入更多的特征并增大容量。当varience过大时，我们可以通过增加采样数量和正则化来减小varience。

需要注意的是，我们往往使用公共的测试样本来测试模型的误差，但是我们在根据公共测试样本来调整我们的模型时，其实是在对公共的测试样本来进行拟合，这有可能导致当我们的模型应用于其他的测试样本时出现误差增大的现象。为了防止这种情况出现，我们可以使用交叉验证(Cross-validation)的方法，将训练集分成两部分，一部分用于训练模型，另一部分用于验证误差。这样得出来的模型才有可能通过公共测试样本得出对于这类问题所有可能样本的误差。

一种交叉验证方法时K-foldcross-validation：我们将训练集分为K份，每次只取1份用于验证误差，另外k-1份作为训练样本。对每一个模型都重复k次，直到所有子样本都成为了一次验证样本，得出平均k次的结果。最后选择平均误差最小的那一个模型。

**论文学习部分：**

***《RGCN:用于目标相关情绪分析的循环图卷积网络》***

本文首先阐释了情感分析（意见挖掘）的重要价值，并举了一个例子来解释了一个用户在同一段评论中可能既有正面的情绪，也有负面的情绪，即使整体情绪可能是正面或是负面中的一个。

依赖于目标的情绪分析，也称为基于情景的情绪分析在情绪分析领域是一项细粒度的任务，传统的方法主要是使用基于特征的支持向量机（SVM），然而这些方法需要手动设计特征。而神经网络具有学习连续文本特征的能力，能够通过多个隐藏层生成新的表示，近年来，越来越多的深度学习方法在这项任务中被探索出来，这些方法比基于特征的方法具有更多的可拓展的方式。

基于递归的神经网络的框架使用了发散合成函数，使他们能够处理情感分析中的复杂合成。尽管这些方法利用了句法结构，但他们关注的是与目标词相关联的结构，而忽略了整个文本的结构，而且仅考虑与目标词相关的词，很难处理长句，虽然目标词在任务中起着重要作用，但对整个句子的表征也是必不可少的。

基于递归的神经网络（RNN）、记忆网络和卷积神经网络（CNN）的流行方法不不使用句法信息，而是通过句子序列或注意机制隐式的保持文本结构。但是必须考虑到整个文本结构以及与目标词相关的特定区域结构的重要性。作者举了一个例子：“我吃了一顿丰盛的饭，我们的服务员引导我们去买一种价格范围内非常好的葡萄酒，而不是让我们去买一种价格相近但质量不好的葡萄酒。”目标词“葡萄酒”出现了两次，在句子的不同部分有不同的情感极性。总的来说，用户对“葡萄酒”持有积极的极性。然而，如果我们在句子中交换两个方面，情感的极性就会改变。结果表明，目标词不仅与句子中的词有关，而且与整个文本结构有关。

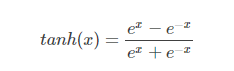
为了解决这个问题，对文本的图形结构执行操作是至关重要的。图卷积网络可以在图结构上运行，其中节点状态通过合并相邻节点的信息来表示。因此随着层数的增加，同一子图中的节点的特征难以区分

（到目前为止论文已经提到了三种神经网络，所以我决定先了解这几种神经网络的大致结构再回来继续学习该篇论文）

***《从多层感知器到卷积网络（一）》***

在多层感知器（MLP）中，曾经科学家使用阶跃函数作为神经元来模拟神经元传递信号，但是直接输出-1或1的方式对于以后的优化工作及其困难，所以有人想到了使用正切函数tanh。

正切函数：

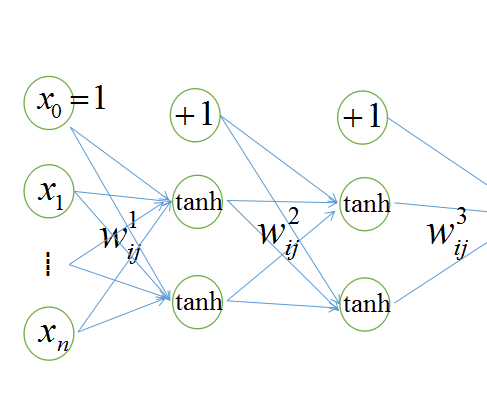


其导数形式为：



tanh函数具有明显的S曲线，在x远大于0时输出1，x远小于0时输出-1，而在其他时候输出一个-1到1之间的数，这样一个平滑的输出范围更利于后期的优化工作。

MLP共3层，分为输入层，隐藏层与输出层。对于输入向量x，在输入第1层前，进行转换变成一个加权分数：。W1指的是第0层与第1层的权重矩阵，b0就是bias项。添加bias项的原因是为了让我们的神经网络表达能力更强。算出加权分数后，就用tanh函数处理：。处理的结果就作为下一层的输入，经过层层处理，最后一层输出我们想要的结果。



为了控制网络的意图，使用一个cost函数（用以衡量神经网络的输出好坏）。运用梯度下降等方法来寻找最优的权重，这个部分在上一部分有所总结，不多加赘述。

**下周学习计划：**

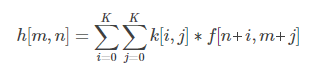
继续跟随深度学习视频课程学习深度学习理论方法，并大致了解完卷积神经网络以及图神经网络的大致结构，推进《RGCN》论文的理解程度。

### 第二周 9月23日~9月29日

***《RGCN:用于目标相关情绪分析的循环图卷积网络》***

为了更好的理解论文，我简要的了解了一下卷积神经网络的概念，具体如下：

卷积是两个变量在某范围内相乘后求和的结果。在wiki中卷积定义为：



f是原始图片，k是核函数，h是卷积后的特征。我们的目的是利用神经网络，学习得出类似k这样的核函数来获取有利于图像识别的特征。

卷积网络是一种特殊的前馈网络，只是将前馈网络中的某些层换成了卷积层。而卷积网络中的卷积层就是一个多核卷积的过程的抽象。

接续上周，本文提出的框架RGCN像递归神经网络一样引入了一个隐状态，节点状态是隐状态和相邻节点信息的表示。使用依存句法分析的结果构造一个文本图，而且目标被视为整个图中的一个特殊区域。用RGCN对文本和目标进行编码并通过这些表示得到情绪分类。

递归神经网络简要概念如下：

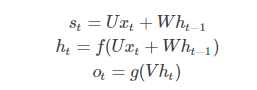
如果一系列输入有明显的上下文特征，也就是说这样的输出必须依赖以前的输入，那么网络需要有一定的“记忆能力”，而递归神经网络（Recurrent Neural Network）便能够赋予网络这样的记忆力。



左侧是递归神经网络的原始结构。与普通三层结构稍有不同的是，隐藏层添加了一个闭环。上图右侧是展开之后的结构。

xt是t时刻的输入，是输入序列的一部分，对于语言模型，每一个xt将代表一个词向量，一整个序列就代表一句话。ht是时刻t的隐藏状态，ot代表时刻t的输出。输入层到隐藏层直接的权重由U表示，它将原始输入进行抽象作为隐藏层的输入。隐藏层到隐藏层的权重W是网络记忆的控制者，负责调度记忆。隐藏层到输出层的权重V，从隐藏层学习到的表示将通过它再一次抽象，并作为最终输出。

在t=0时刻，UVW都被随机初始化好，h0通常初始化为0。然后随着时间推进进行如下计算：



每一次上一轮的隐藏状态作为ht-1将作为时刻t-1的记忆状态参与下一次预测活动，f是激活函数，g通常为softmax函数。所以可以这样理解隐藏状态：

h=f（现在的输入+过去记忆总结）

论文中对图卷积神经网络（GCN）进行了介绍。

在GCN中，图形结构由相邻矩阵表示，对图形结构进行卷积运算。其描述为：



在中，A是图的邻接矩阵，定义为=A+I,其中I是单位矩阵，是其中元素可以被表示为的对角矩阵。表示训练期间要学习的加权矩阵，f（·）是一个激活函数。

将作为隐层向量，单层GCN的隐表示只能捕获直接邻域的信息。因此，图中节点的表示是根据其相邻节点的特征向量递归计算出来的。此外，对于k层gcn，节点的表示可以捕获其k-hop网络邻居中的结构信息。

将图结构与卷积中的节点特征相结合的GCN模型使得同一集群中节点的输出特征更加相似。虽然GCN模型在特征提取方面很强大，但并不完全适合目标依赖情绪分析任务。

作者认为，当前场景的情感极性不仅与目标词有关，还与上下文（全文）有关。他们提出的模型充分考虑了给定文本中词与词的关系，基于以文本为图的图形，而目标词被认为是图的特定区域。框架如下图所示：

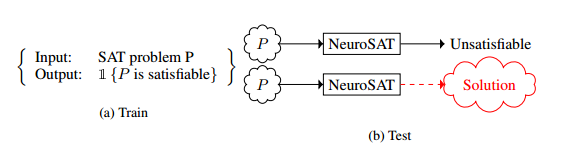


框架分为2块，一部分是文本图形的表示，另一部分是目标词的表示。通过RGCN获取节点的特征，然后通过所有的平均池层获得文本的表示，通过情景词的最大池层获得目标词的表示。两个隐藏状态被发送到完全连接层（fc）后，将完全连接层的输出传递到SoftMax层以获取类似标签。

***《从单位监督学习SAT解算器》***

作者提出的NeuroSAT是一种信息传递神经网络，它只经过训练成为一个分类器来预测可满足性，然后学习解决SAT问题。NeuroSAT通过简单的多次迭代来解决更复杂困难的问题。

命题可满足性问题（SATISFIABLITY problem，简称SAT）是计算机科学中最基本的问题之一。为了让神经网络学会解决SAT问题，作者开发了一个新型消息传递神经网络（novel message passing neroMPNN）NeuroSAT然后训练它作为分类器以预测随机SAT问题集的可满足性。对于每个SAT问题，我们只向NeuroSAT提供单位监督，以表明该问题是否可以得到满足。当对一个新的SAT问题进行预测时，NeuroSAT在找到一个解之前，都是用低置信度进行不满意的猜测，然后收敛到非常高的置信度进行猜测。解决方案本身几乎总是可以从网络的激活中自动解码，从而使NeuroSAT成为一个端到端的SAT求解器。有关训练和测试状态的说明参考下图：



也可以用同样的神经网络结构来帮助构造不满意问题的证明。当作者在一个每个不满意的问题都包含一个小的矛盾的不同的数据集上训练它时（NeuroUNSAT），它不会去寻找令人满意的任务，而是学习检测这些矛盾。正如我们可以从NeuroSAT的激活中提取解一样，我们也可以从神经SAT的激活中提取矛盾的变量。当矛盾中涉及的变量数量相对于变量总数较小时，了解矛盾中涉及的变量可以更有效地构造一个解的证明。

问题背景：命题逻辑的公式是一个使用常量true（1）和false（0）、变量、否定、连词和析取构建的布尔表达式。若将布尔值赋给公示的变量使公式的计算结果为1，则公式是可满足的。例如，公式(x1 ∨x2 ∨x3) ∧┐(x1 ∧ x2 ∧ x3) 是可满足的，因为它将在每个未将x1、x2和x3映射到相同值的赋值下计算为1。对于每一个公式，都存在一个合取范式（cnf）的等价公式，表示为变量（可能为负）析取的合取。在cnf中，公式的每一个连词都称为子句，而子句中的每一个（可能是负数）变量都称为literal（字？）。上面的公式相当于cnf公式（x1∨x2∨x3）∧（┐x1∨┐x2∨┐x3），我们可以更简洁地表示为{1|2|3,‾1|‾2|‾3}。cnf中的公式有一个可满足的赋值条件是且仅当它有一个赋值时，每个子句至少有一个literal映射到1。SAT问题是CNF中的一个公式，其目标是确定公式是否可满足，如果可以，则生成一个令人满意的变量真值分配。我们用n表示SAT问题中变量的数目，用m表示从句的数目。

分类任务。对于SAT问题p，当且仅当P为可满足时我们定义φ（P）为真。第一个目标是学习一个近似于φ的分类器。给定SAT问题的分布ψ，我们可以用形式（P，φ（P））的例子来构造数据集D*train*和D*test*，方法是用现有的SAT解算器对问题p～ψ进行抽样并计算φ（p）。在测试时，我们只得到问题p，目标是预测φ（p），即确定p是否满足要求。最后，我们关心的是解算的方法，其中也包括找到解决问题的方法。

**下周学习计划：**

### 第三周 10月7日~10月13日

***《RGCN:用于目标相关情绪分析的循环图卷积网络》***



框架分为2块，一部分是文本图形的表示，另一部分是目标词的表示。通过RGCN获取节点的特征，然后通过所有的平均池层获得文本的表示，通过情景词的最大池层获得目标词的表示。两个隐藏状态被发送到完全连接层（fc）后，将完全连接层的输出传递到SoftMax层以获取类似标签。

首先，单词序列进入嵌入层。一般来说，嵌入层的作用是将文本映射到一个向量空间中。这样做的目的时方便机器进行计算，将单词转变为词向量，机器便可以通过计算不同词向量之间夹角余弦值而得出单词之间的相似性。再Embedding Layer中，一般方法是对单词进行one hot编码，向量空间的大小或维度被指定为模型的一部分，向量以小的随机数进行初始化。被编码过的词映射成词向量，如果使用循环神经网络则每个单词都可以作为序列中的一个输入。

**图形构造**：设W = {w 1 ，w 2 ，... w N }（| W | = N）表示在语料库中的单词集合。 W s是给定文本的单词集合，W s = {w s1 ，w s2 ，... w sM}（| W s | = M，W s ∈W）和W t ={w t1 ，w t2 ，... w tK}（W ti∈W s ），如图一所示，给定这样一个句子”the food is so good and popular that waiting can really be a nightmare”,W t={food}且Ws = {the; food; is; so; good; and; popular; that;waiting; can; really; be; a; nightmare}。

更正式地说，无向文本图定义为G=（V：E），其中V和E分别是节点集和边集。文本图的构造如下：首先，每个顶点对应于Ws中的一个词，然后建立了从属于一组特定对话的两个词的句法依赖关系的连接。。最后，将自循环添加到文本图中。从无向文本图中得到对称邻接矩阵A（Aij ∈ {0，1}），L代表A的归一化拉普拉斯矩阵。（拉普拉斯矩阵定义为：D-A。D是该图G度的矩阵。归一化算式在上一周已经给出）

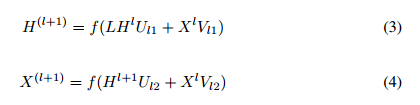
**RGCN层：**

在图形构造后给定的文本将转换为图形。图中的节点由嵌入层处理后的向量xi表示。一般来说语料库中的集合W的长度N大于文本集合Ws的长度M。由于不需要为每个给定的文本建立邻接矩阵A∈，因此将我们将节点序列转换成当前文本的单词序列，然后邻接矩阵A变成

嵌入层的输出表示为Xembed。引入了节点的隐藏状态，将节点的初始隐藏表示定义为Xembed∈，F为特征尺寸。图中节点的初始状态定义为：



其中U 0是权重矩阵，并且∈ 。 对于RGCN层，传播规则定义为：



其中Ul1，Ul2，Vl1和Vl2是可训练参数。由方程可以看出，在RGCN中节点的隐藏状态由节点的上一个隐藏状态和相邻节点的状态更新。它不仅像GCN一样包含了图形结构的信息，还像RNN一样使用了上一次的隐藏状态来更新当前状态。

聚合函数设计为池化操作，这是对特征向量的元素操作。在池化后，用来表示文本图的节点的隐藏特征向量表示为。在这个项目中，作者对给定的文本应用平均池化层，对目标词应用最大池化层。

池化层：使用某一位置的相邻输出的总体统计特征来代替网络在该位置的输出。对输入的特征图进行压缩，一方面使特征图变小，简化网络计算复杂度，一方面进行特征压缩，提取主要特征。平均池化层对特征图的某一个区域内的所有特征值取一个平均值作为该区域的特征，而最大池化层寻找区域内的最大值作为该区域的特征。使用池化可以看作增加了一个无限强的先验。

该模型经过训练，以监督的方式最小化交叉熵误差，损失函数定义为：



其中，pti是模型给出的第i个训练示例的概率。我们使用反向传播来计算所有参数的梯度，使用学习率为0.001的Adam优化算法来训练模型。dropout（丢弃法）和早期停止技术是用来缓解过度拟合。使用GloVe预训练的300维文字嵌入，训练期间不调整。此外，批大小设置为128，最大历元为50。

Adam优化算法：Adam是一种可以替代传统随机梯度下降过程的一阶优化算法，它能基于训练数据迭代的更新神经网络权重。

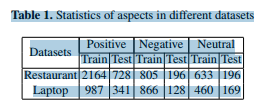
dropout：dropout是指在深度学习网络的训练过程中，对于神经网络单元，按照一定的概率将其暂时从网络中丢弃。

Early stop：早期停止法是指在验证误差达到最小时停止训练。

G1oVe：GloVe是通过整个语料库的所有句子取分析词与词之间的关系的模型。

**实验和结果**

**实验设置：**作者在Semeval2014的公共数据集上测试模型，包括用户生成的有关笔记本电脑和餐厅域的评论。他们删除了一些带有冲突标签的样本。数据集的统计数据如下



作者只对数据集进行了基本的文本预处理，包括字母转换为小写、删除空白、使用python nltk工具进行标记化以及使用斯坦福解析器解析语法依赖项。

在这个的模型中，操作是在图结构上执行的，我们要求基于语法依赖关系的边连接到属于特定词性集的词上。词性集的一部分格式以[“jj”、“nn”、“r”、“cc”、“in”]开头，对于给定的文本，如果句法相关性中成对词的词性部分都是词性集的一部分，则它们之间建立了链接。在实验中，我们开发了具有RELU激活功能的五层RGCN。模型的隐藏尺寸设置为200。

RELU函数：在CNN卷积神经网络中，习惯用RELU函数代替sigmoid, tahh等目标激活函数，这应该是因为，RELU函数在大于0的时候，导数为恒定值，而sigmoid和tahh函数则不是恒定值，相反，sigmoid和tahh的导数，类似于高斯函数的曲线形状，在两端接近目标时，导数变小。

导数小的话，在训练神经网络的时候，会BP反向传播误差，导致收敛减慢；而relu函数则避免了这点。

作者将模型与以下基线方法进行比较：

AdaRNN用于基于递归神经网络的目标相关依赖树上的特征学习，根据输入自适应地选择合成函数。

TD-LSTM是基于lstm网络的模型，其中两个lstm模型用于模拟围绕情绪分类目标字符串的前后上下文。

ATAE-LSTM在lstm网络中使用注意机制，其中不同的方面参与句子的不同部分。

MemNet是一个外部记忆的神经注意模型，由多个计算层组成。

RAM是一个在递归神经网络上采用多注意机制的框架。

IAN是一个交互式注意力网络模型。它使用两个注意力网络以交互方式对目标和上下文进行建模。

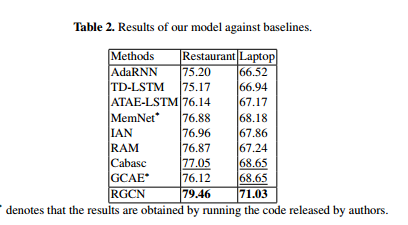
Cabasc基于记忆模型，通过句子级内容注意机制和上下文注意机制两种注意机制，解决语义失配问题。

GCAE基于卷积神经网络和门控机制。

我们注意到，在不同的文献中，对同一数据集上执行的同一模型报告了不同的结果。我们认为基线方法的结果受文本预处理和单词嵌入的影响，如[3]所述。

为了揭示模型的能力，所有模型都使用了相同的词向量，在pytorch中重新实现了作者没有提供实现的模型。按照原始文件中报告的超参数设置，将未报告的参数设置为与此相同。

所有方法均以精度评价为度量，结果如表2所示。最好的分数用粗体突出显示，下划线表示第二好的表现。



结果表明，RGCN在这些数据集上始终优于所有比较方法。这可能是由于句法依赖关系图在两个长距离词之间建立了联系，缩短了情景（方面）和相关词之间的距离。

在基于LSTM的模型中，TD-LSTM、ATAE-LSTM和IAN比其他模型有更好的效果，因为IAN使用上下文和目标注意机制，更好地将句子的重要部分用于情景（方面）词。

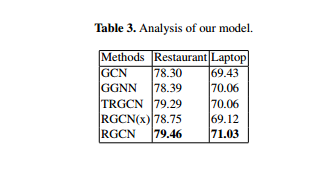
MemNet基于存储网络，包含多个关注层，优于基于LSTM的笔记本电脑模型。与RAM和MemNet相比，Cabasc通过句子级内容注意机制和上下文注意机制增强了从全局角度捕捉某一方面重要信息的能力，从而在笔记本电脑和餐厅的所有基线中都具有最佳的性能。GCAE利用卷积神经网络和门控机制，在笔记本电脑上获得最佳的CABASC效果。

尽管AdaRNN利用了了依赖性分析的结果，但它的性能比其他人差，特别是在笔记本电脑上。与AdaRNN相比，我们的方法将整个文本信息保存在RGCN中，目标集中在特定的方面区域，以避免其他图形部分的噪声。

### 第四周 10月14日~10月20日

***《RGCN:用于目标相关情绪分析的循环图卷积网络》***

为了说明该模型的有效性，作者设计了更多的相关的模型来比较结果。其中包括将RGCN层换成GCN层和门控图神经网络层（GGNN）的模型，模型分别表示为GCN和GGNN。与RNN不同的是，RGCN根据最后节点状态和隐藏状态来更新节点状态。TRGCN指的是节点的状态实现由隐藏状态更新的RGCN图。RGCN（x）是一个将节点状态传递到池层而不是隐藏状态的模型。这些模型的最佳性能如下表所示：



结果表明随着层数的增加，GCN比其他模型的性能更差，这是由于GCN很难区分不同的节点特征。RGCN 的精度高于所有模型，表明该模型在图形结构中具有更好的节点表示能力。实验结果表明，将隐状态引入GCN中可以提高系统性能。

**层的影响：**

如下图2所示，单层的隐藏表示只能捕获直接领域的信息，而具有K层的模型可以包含最多为k帧信息的节点。图2显示，当层设置为2时，GCN优于RGCN。随着层数增加，GCN的精度低于RGCN，且GCN线比RGCN线更平，这与之前的分析结果一致。同时还可以从图中观察到，精度不会随着层数增加而提高。

**嵌入调优的效果：**

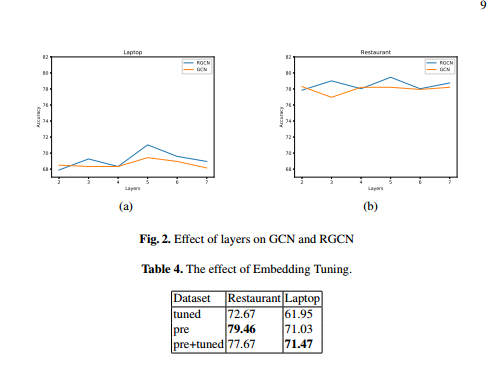
作者推测该神经网络可以捕获没有在预先训练的单词嵌入中表示的机构信息。为此，他比较了以下嵌入调优策略：

—预处理：使用预先培训过的单词嵌入，并不会在培训期间进行调优。

—调优：单词向量由预先训练的单词嵌入初始化，这些参数在监督训练中更新。

—预处理+调优：单词嵌入由两部分组成，一部分是预先训练的单词向量，另一部分是随机初始化的单词嵌入。前者不进行调整，后者在培训中更新。该实验使用50维随机初始化字嵌入并在RGCN模型上进行。

下表4中的结果表明,预处理+调优能在“笔记本电脑”上达到最佳性能，而“预处理”在“餐厅”上得到最佳效果。作者不确定结果的原因。



**结论与个人感受：**

本文提出了一种新的目标依赖情绪分析任务框架。通过与各种模型进行数据对比，可以发现该模型优于大多数模型。同时分析了层数以及嵌入调优的影响，为整合句法信息提供了新的方法。

我认为，将循环神经网络的隐状态引入图形神经网络从而让神经网络拥有记忆能力这一点是有效的，这种方法能够考虑到目标的情感与整个文本结构的关系，相比于传统的模型应该能更容易获取目标节点的特征。

***《从单位监督学习SAT解算器》***

**模型**

SAT问题具有简单的句法结构，因此可以使用标准方法（如RNN）将其编码到向量空间中。然而，命题逻辑的语义学却产生了这种句法方法所忽略的丰富的不变性，如排列不变性和否定不变性。具体地说，一个公式的可满足性不受 重新排序变量的影响（如在整个公式中交换x1和x2），不受重新排列子句的影响(例如，用第二个子句替换第一个子句)，不受重新排列子句中字的影响(例如，将替换为。)。同时公式的可满足性也不受一个给定变量对应的每一个字的求反影响（如在中对所有的x1求反得到）。

NeuroSAT强调排列不变性和否定不变性。作者将SAT问题编码为一个无向图，每一个字、子句都有一个节点，每个文字和子句之间有一条边，而每一对互补文字之间有不同类型的边（如xi与xi求反）。NeuroSAT通过沿着图的边缘来回传递“信息”，迭代地细化每个节点的向量空间嵌入。在每个时间步骤，作者都对每个文字和每个子句进行了嵌入。迭代包括两个阶段。首先，每个子句从其邻近的字接受消息并相应的更新其嵌入。然后每个字从其相邻的子句及其补语（即本身取反）接受消息，然后相应的更新其嵌入。下图2提供了架构的高级说明。



这个图代表了的神经系统高级图解。两个图的顶部分别是4个文字的节点，底部是两个子句节点。

更正式地说，该模型由两个向量、三个多层感知器(**L**msg, **C**msg, **L**vote)和两层layer-norm（层标准化）的 LSTMs参数化。在每一个时间步骤t，我们有一个矩阵，它的第i行包含对字的嵌入，和一个矩阵它的第j行包含对子句的嵌入，我们分别通过（平铺）（什么是平铺）来初始化它。

层标准化：Layer Normalization和普通的标准化类似，是将网络中的层进行归一化的操作。标准化是将数据按比例缩放，使之落入一个小的特定区间。其中最典型的就是数据的归一化，即将数据统一映射到[0,1]区间上。

**LSTMs**：Long Short Term Memory，即长短期记忆。这是一种为了避免常规RNN的梯度消失而被广泛应用的RNN的特例。（为什么？）

关于为什么LSTM可以解决这个问题我看不懂，只能直接附上讲解人的原文：

LSTM的结构可以如下表示：RNN总是以“覆写”的方式计算状态，根据求导的链式法则，这种形式直接导致梯度被表示成连乘积的形式，以致于造成梯度消失—粗略地说，很多个小于1的项的连乘很快的逼近0。现代的RNN（包括但不限于使用LSTM单元的RNN）使用“累加”的形式计算状态，这种累加形式导致导数也是累加形式，因此避免了梯度消失。



RNN在t时刻的隐藏状态h(t)是由h(t-1)和当前输入Xt得到的，而LSTM可以从图中看出与原始的RNN结构有些许不同，也更为复杂。从上图中可以看出，在每个序列索引位置t时刻向前传播的除了和RNN一样的隐藏状态h(t)，还多了另一个隐藏状态，如图中上面的长横线。这个隐藏状态我们一般称为**细胞状态**(Cell State)，记为C(t)。细胞状态就是长记忆，而隐藏状态就是短期记忆。如下图所示:



除了细胞状态，LSTM图中还多了一种结构，称为门控结构（Gate）。一般包括遗忘门，输入门和输出门三种。

**遗忘门（forget gate）**在LSTM中以一定的概率控制是否遗忘上一层的隐藏细胞状态。其结构如下图所示：



使用上一序列的隐藏状态h(t-1)和本序列数据x(t)通过一个激活函数（一般是sigmoid）得到遗忘门的输出f(t)。sigmoid的输出在[0,1]之间，因此这里的输出代表了遗忘上一层隐藏细胞状态的概率。

**输入门（input gate）：**从下图可以看出第一部分使用了sigmoid激活函数得到结果i(t)，第二部分使用了tanh激活函数得到a(t)，两个函数的结果相乘再去更新细胞状态。

****

细胞状态C(t)由两部分组成，如下图，第一部分是C(t-1)和遗忘门输出ft的乘积，第二部分是输入门的i(t)和a(t)的乘积，C(t)=C(t−1)⊙f(t)+i(t)⊙a(t)，⊙为Hadamard积。



**输出门（output gate）**：h(t)的更新由h(t-1)和x(t)通过sigmoid激活函数得到的结果o(t)与细胞状态C(t)和tanh激活函数组成。

h(t)=o(t)⊙tanh(C(t))

⊙为Hadamard积。

### 第五周 10月21日~10月27日

***《SATNet：使用一个可微的可满足性求解器桥接深度学习与逻辑推理》***

**摘要**

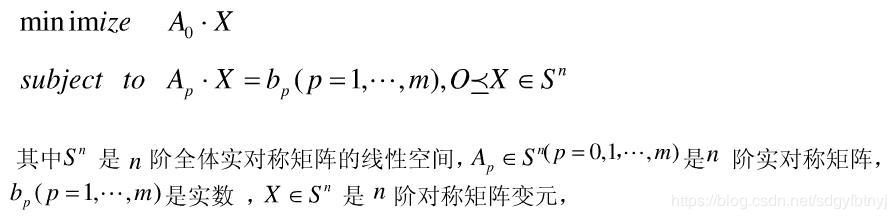
在深度学习架构中集成逻辑推理一直是现代人工智能系统的主要目标。在这篇论文中，我们提出了一个新的方向，通过引入一个可微的最大可满足性**(MAXSAT)**求解器，可集成到更大的深度学习系统的循环中。我们的近似解算器是基于快速坐标下降法来求解与MAXSAT问题相关的半定规划(SDP)。我们通过这个SDP的解决方案来展示如何分析区分和有效地解决相关的反向（传播？）过程。我们证明，通过将这个求解器集成到端到端的学习系统中，我们可以以最少监督的方式学习具有挑战性的问题的逻辑结构。特别地，我们展示了我们可以使用单比特监督来学习奇偶函数(对于深度网络来说，这是一个传统的困难任务)，并且仅仅通过例子来学习如何玩9×9数独游戏。我们还解决了一个“可视化数独”问题，该问题通过将MAXSAT求解程序与传统的卷积架构相结合，将数独谜题的图像映射到相关的逻辑解决方案。因此，我们的方法在深度学习中集成逻辑结构方面显示出良好的前景。

**MAX-SAT**：最大可满足性问题是SAT问题的一个自然的拓展，对于给定的CNF公式，MAX-SAT问题的目标是找到一个赋值使其满足最多的子句。

**坐标下降法**：对一个高维函数，找一个初始点，迭代地通过将大多数自变量固定，只针对剩余自变量（一般为1个）求极值的问题。则通过这个这个方法把一个高维优化问题分解成多个一维问题。

找一个初始点，只对其中一个方向xi作为变量，其他方向的参数全部当作固定值来求xi的导数，找一个xi使f为最小值的点。找到xi以后，另找一个方向xi+1作为新一轮的变量，上一轮得到的xi作为这一轮的固定值代入。不停迭代直到某一次迭代中函数得不到优化，即达到某一住驻点后停止。要求是可微的凸函数。不可微不行。

**半定规划（SDP）**：半定规划是线性规划的一种推广，它是在满足约束“对称矩阵的仿射组合半正定”（？？？？？？）的条件下使线性函数极大（极小）化的问题。这个约束是非线性、非光滑并且是凸的，因而半定规划是一个非光滑凸优化问题。



**端到端**：端到端，其实就是从模型的输入端到输出端，不需要经过预处理或者后处理手段，原始数据进入网络输出结果。

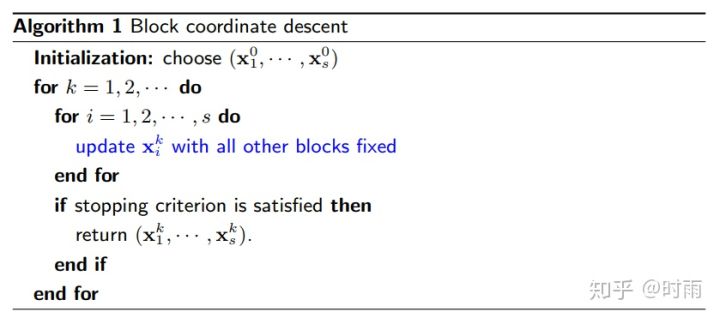
**1、介绍**

尽管现代深度学习已经在许多领域产生了突破性的改进，但最先进的方法仍然难以捕获由离散逻辑关系产生的“硬”和“全局”约束。由于这一缺陷，近来人们对将逻辑或符号推理集成到神经网络体系结构中产生了浓厚的兴趣。然而，除了少数例外，以前的工作主要集中于通过可调的连续参数将现有的关系集成到更大的可微分系统中，而不是发现以真正的端到端方式产生一组观察结果的离散关系。举个例子，考虑一下流行的基于逻辑的益智游戏《数独》，在这款游戏中，玩家必须填写一个9×9的数字网格，以满足特定的约束条件。如果没有给出数独的规则（如问题变量之间的关系），那么最好是连带地学习游戏规则和学习如何以端到端的方式解答数独游戏。

我们认为学习逻辑结构的问题具体表现为可满足性问题，具体地说，是能被很好地建模为SAT或MAXSAT（SAT的优化类似物）实例的问题。这是一个包含大量符号人工智能的丰富领域，由于神经网络依赖于连续可微的参数化，传统上很难将其整合到神经网络架构中。我们的主要贡献是开发并推导出一个可嵌入到更复杂的深层体系结构中的可微分的平滑MAXSAT求解器，并证明该求解器能够从示例中有效地进行端到端的逻辑关系学习(不需要对这些关系进行硬编码)。更具体地说，我们基于求解SDPs的快速块坐标下降法为smoothed SDP relaxation of MAXSAT建立了一个可微分的求解器。我们提供了一种有效的机制，可以通过使用与前向传递中使用的类似的块坐标下降求解器来区分此SDP的最佳解决方案。我们的模块支持GPU加速，大大提高了训练的可扩展性。

使用这个框架，我们能够解决几个问题，尽管它们很简单，但事实证明，传统的深度学习方法和现有的逻辑学习方法根本不可能在没有任何先验知识的情况下可靠地学习。特别的，我们展示了我们可以只用单比特监督来学习奇偶函数，众所周知这对于深度分类器来说是一个挑战。我们还表明我们可以学会玩9×9数独，而这是一个对现代神经网络架构都具有挑战性的问题。我们证明了我们的模块可以快速地恢复一个描述可行的数独解决方案的约束，学习在测试时正确地解决98.3%的谜题，而不需要任何关于问题结构的手工编码知识。最后，我们证明了我们可以将这个可微分的求解器嵌入到更大的架构中，来求解一个“视觉数独”的问题，其输入是一个数独游戏的图像而不是二进制表示。我们证明，在完全端到端的设置下，我们的方法能够将经典的卷积网络(用于数字识别)与可微分的MAXSAT求解器(用于学习逻辑部分)集成在一起。综上所述，这是朝着现代人工智能的主要目标——将逻辑推理集成到深度学习架构——的重大进展。

**块坐标下降**：块坐标下降是坐标下降的更一般化，它通过对变量的**子集**进行同时优化，把原问题分解为多个子问题。在下降的过程中更新的次序**可以是确定或随机的**，如果是确定的次序，我们可以设计某种方式，或是周期或是贪心的方法选择更新子集。下图是一个通用算法。



**松弛（relaxation）**：一般数学意义上所说的松弛，是指减少对变量的约束。求一个点v到源点的最短距离。v到源点的估计距离dist（v）是否满足三角不等式：

dist（v）<=dist(u)+w(u,v)

如果不满足这个约束，那么松弛就是把dist（v）更新为更短的dist（u）+w（u，v），即所谓更“紧”的路径，新的dist（v）也就不再需要检查是否满足这个不等式约束了，这个原来的约束对于dist（v）来说就不再是能将dist（v）收的最紧的约束了，相当于这个原约束被“松弛”了。

**2、相关工作**

近来，深度学习社区越来越重视复合嵌入的概念，一个深度网络中的非传统层以便对系统进行端到端的培训。主要的例子包括逻辑推理模块和优化层。我们的工作结合了这两个领域的研究，利用基于优化的逻辑推理结构的松弛，即MAXSAT的SDP松弛。我们将在下面更详细地探讨这些相关的研究领域。

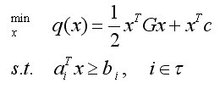
**深度网络中的逻辑推理：**我们的工作与最近将逻辑推理集成到深度学习架构中的兴趣密切相关。以前的大多数系统都侧重于从现有的已知关系集创建可区分（微分）的模块，以便深度网络能够了解这些关系的参数。例如，Palm等人(2017)引入了一个网络，该网络使用关于被允许相互作用的变量的手工编码信息进行关系推理，并在9×9数独上测试了这个网络。类似地，Evans和Grefenstette(2018)通过为特定的“规则模板”构建可微的基于SAT的表示，将归纳逻辑编程集成到神经网络中。当这些网络被输入关于变量之间关系的先验信息时，我们的方法端到端的学习这些关系和它们相关的参数。虽然最近的其他工作也尝试同时学习规则和参数，但是这些体系结构捕获的问题类是有限的。例如，Cingillioglu & Russo(2018)训练神经网络应用特定类别的逻辑程序，即给定命题集是否包含特定结论的二分类问题。虽然这种方法不依赖于预先手工编码的结构，但是我们的方法适用于更广泛的领域，包括可以简化为MAXSAT的任何问题。

### 第六周 10月28日~11月3日

***《SATNet：使用一个可微的可满足性求解器桥接深度学习与逻辑推理》***

**补充：**

二次规划一般形式：



τ是有限指标集。设一集合为I，若对于每个a∈I，都对应了一个集合Aa，则由这些Aa的全体构成的集合A称之为集合族，I就是该集合族的指标集。c，x和{ai}，都是R中的向量。如果Hessian矩阵是半正定的，则我们说(1.1)是一个凸二次规划，在这种情况下该问题的困难程度类似于线性规划(如果=0，二次规划问题就变成线性规划问题了)。如果有至少一个向量满足约束并且在[可行域](https://baike.so.com/doc/6481746-6695451.html)有下界，则凸二次规划问题就有一个全局最小值。如果是正定的，则这类二次规划为严格的凸二次规划，那么全局最小值就是唯一的。

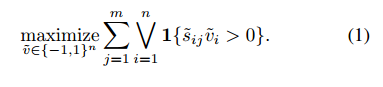
**论文部分**

**深度网络中的逻辑推理：**我们的工作与最近将逻辑推理集成到深度学习架构中的兴趣密切相关。以前的大多数系统都侧重于从现有的已知关系集创建可区分（微分）的模块，以便深度网络能够了解这些关系的参数。例如，Palm等人(2017)引入了一个网络，该网络使用关于被允许相互作用的变量的手工编码信息进行关系推理，并在9×9数独上测试了这个网络。类似地，Evans和Grefenstette(2018)通过为特定的“规则模板”构建可微的基于SAT的表示，将归纳逻辑编程集成到神经网络中。当这些网络被输入关于变量之间关系的先验信息时，我们的方法端到端的学习这些关系和它们相关的参数。虽然最近的其他工作也尝试同时学习规则和参数，但是这些体系结构捕获的问题类是有限的。例如，Cingillioglu & Russo(2018)训练神经网络应用特定类别的逻辑程序，即给定命题集是否包含特定结论的二分类问题。虽然这种方法不依赖于预先手工编码的结构，但是我们的方法适用于更广泛的领域，包括可以简化为MAXSAT的任何问题。

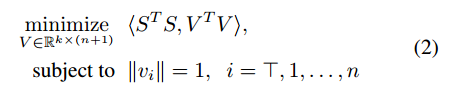
**3、一个可微的可满足性求解器**

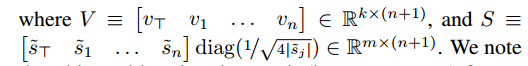
MAXSAT问题是众所周知的可满足性(satisfiability, SAT)问题的优化模拟，其目标是最大化满足的子句数量。我们提出了一个可微分的、平滑的近似MAXSAT求解器，它可以集成到现代的深度网络结构中。该求解器采用快速坐标下降法求解MAXSAT的SDP松弛问题。我们描述了我们的MAXSAT SDP松弛以及我们的MAXSAT深层网络层的前向传递(它使用了这种松弛)。然后，我们展示如何通过MAXSAT SDP解析微分并有效地解决相关的反向传播。

**3.1**求解可满足的SDP公式  
 考虑一个包含n个变量和m个子句的MAXSAT实例。设表示问题变量的（二元指派）二进制分配（作业？），为真值，变量然后定义，，代表着在子句中的符号。然后我们就能把MAXSAT问题表示成：



为了形成(1)的半定松弛首先将离散变量松弛为与某些“真理的方向”有关的连续变量（||x||即对向量x取模，为x的模长，点x到原点的距离）。特别的，我们将连续的Vi与离散的通过基于随机舍入的关联起来。我们还定义了一个与相关的系数向量。然后是MAXSAT的SDP松弛

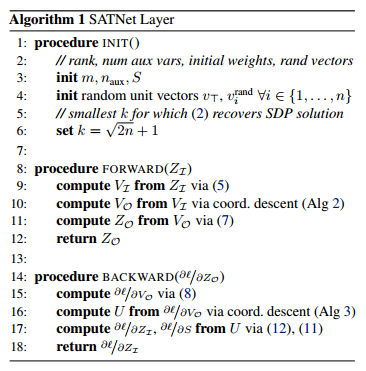


我们注意到这个问题是MIN-UNSAT的一个低秩(但非凸)公式，即等同于MAXSAT。该公式可以重写为SDP，并已证明在的情况下可以恢复最佳SDP解决方案。尽管不具有凸性，但对于所有，可以通过坐标下降来最优地解决问题（2）。

特别的，给出了与有关的目标项，其中为S的第i列向量。在的约束下最小化上的这个数量，就会得到坐标下降更新：



这些更新可证明收敛到SDP（2）的全局最佳固定点。这个更新的更详细的推导可以在附录A中找到。



3.2 SATNet：可满足性解答层

通过使用我们的MAXSAT SDP松弛和相关的坐标下降更新，我们创建了一个的可满足性解答(SATNet)深层网络层。定义为具有已知分配的MAXSAT变量的指数，令为对应于赋值未知的变量的指标。我们允许概率输入或二进制输入,然后输出未知变量的分配是类似的概率性的或(在测试时可选)二进制（二元）的。我们让分别参考所有输入和输出的分配。输出是通过SDP（2）从输入生成的，并且我们层的权重对应于SDP的低秩系数矩阵S。这个前向传递过程如图1所示。我们将在算法1中描述层初始化和前向传递的步骤，并在下面详细介绍。

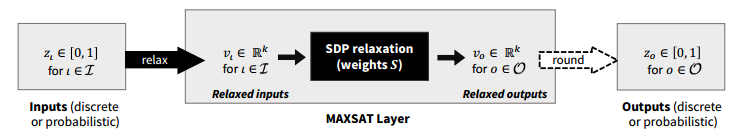


图1.MAXSAT层的前向传递。该层将已知MAXSAT变量的离散分配或概率分配作为输入，通过权值为S的MAXSAT SDP松弛来输出未知变量分配的猜测。

3.2.1.层初始化  
 在初始化SATNet时，用户必须指定此层可以表示的子句m的最大数量。把m设得低一些通常是可取的;特别是低秩结构可以防止过拟合，从而提高泛化能力。

考虑到这种低秩结构，用户可能希望通过辅助变量在一定程度上增加层的表示能力。这里的高级直觉来自布尔满足问题的合取（连接）范式(CNF)表示;向一个问题添加额外的变量可以极大地减少描述该问题所需的CNF子句的数量，因为这些变量所起的作用类似于对推断有用的注册内存。最后，我们设置，其中n表示实际问题变量的数量加上辅助变量（除辅助变量之外的实际问题变量的数量？）。这是我们的MAXSAT松弛（2）恢复其相关SDP的最优解所需的k的最小值。

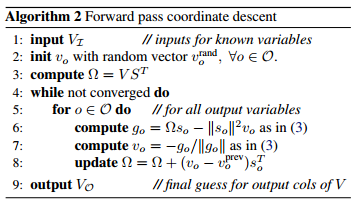
**3.2.2 步骤1:松弛层输入**

我们的层首先将其输入松弛为连续的向量，用于SDP公式(2)。也就是说，我们将每一层的输入 松弛（放宽）到相关联的随机单位向量，。（该等式由3.1节描述的离散变量与其连续松弛之间的概率关系推导而来。）约束（4）可以由满足，其中是随机单位向量。为简单起见，我们使用符号(即V的I索引列子集)来统称通过式(5)得到的所有松弛层输入。

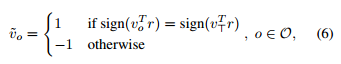
**3.2.3 步骤2：通过SDP生成输出的连续松弛**

给定连续的输入松弛量V，我们的层使用坐标下降更新（3）来计算连续输出的松弛量（我们统称为）的值。 值得注意的是，坐标下降更新仅针对输出变量进行计算，即不针对其赋值作为该层的输入给出的变量进行计算。

算法2中详细介绍了我们用于向前传播的坐标下降算法。该算法保留计算所需的项，然后在每次内部迭代期间通过秩一更新对其进行修改。 因此，每次迭代运行时间为O（nmk）（实际上，只需要进行少量迭代即可收敛）。



给定来自坐标下降的松弛输出VO，我们的层通过（二值化）阈值化或随机舍入（我们将在此处描述）将这些输出转换为离散或概率变量分配ZO。

随机取整的主要思想是，对于每个，我们都可以从单位球面中提取一个随机超平面r并分配，其中是的布尔输出。直观地，当且仅当vo和真向量在随机超平面r的同一侧时，此方案才将设置为“ true”。 给定正确的权重S，此随机舍入过程可确保对某些NP难问题的最佳预期近似比。

在训练期间，我们不会明确执行随机舍入。 相反，我们注意到vo和在任何给定r的同一侧的概率为，因此设置来等同于该概率。

在测试过程中，我们可以以相同的方式输出概率输出，也可以通过阈值（二值化）或随机舍入输出离散的分配。 如果使用随机舍入，我们将舍入多次，然后将zo设置为最大化方程式（1）中的MAXSAT目标的布尔解。 通过先前的工作我们已经观察到，这种重复的舍入在实践中会提高近似率，特别是对于MAXSAT问题更加明显。

### 第七周 11月4日~11月10日

**INTRODUCTION：**

#P-hard:　Sharp-P （#P）的定义主要指NP问题中对应的满足条件的实例或路径的个数的一类问题。简单来说就是比NP还难。

**WMC定义：**

给定命题变量的（有限）集合S，其中命题变量的形式为v或者非v。合取从句是文字的并集，而析取从句是文字的析取。 如果子句恰好具有k个文字，则其宽度为k。赋值ν：S→{0; 1}将每个S中的变量变量映射为0（假）或1（真）。 定命题公式φ，其模型计数＃φ是满足φ的赋值ν的数量。**（读作Omega）**是φ的加权模型计数，其中是权值函数，而是所有可能赋值的集合（德国的某种A的字体）。

如果公式φ如果是合取从句的析取，则为析取范式（DNF）。 我们说，如果DNF（包含CNF）包含最多k个宽度的子句，则其宽度为k。在通常意义上，赋值ν*（****读作纽****）*满足命题公式φ（**读作fai**），记作νφ。 在这项工作中，我们将，使得每个赋值都映射到一个合理的概率，并且。综上所述就是将每个命题变量视为一个独立的伯努利随机变量，并给它赋值一个概率。

伯努利随机变量：伯努利分布[[1]](https://baike.so.com/doc/6390221-6603876.html#refff_6390221-6603876-1) 是一种离散分布,有两种可能的结果。1表示成功，出现的概率为p(其中0<p<1)。0表示失败，出现的概率为q=1-p。

**KLM 算法：**

然后是KLM算法。KLM算法[16]是完全多项式随机逼近方案（FPRAS），并为加权#DNF提供了概率保证。给定误差（**读作epsilon**）且置信度值0 <δ<1**（读作delta）**，KLM能在多项式时间内计算（真实加权模型计数µ的近似值），并使得，即使μ尖落在在乘性约束范围内的概率能达到你的置信的值。

对于具有n个变量和m个子句的析取公式φ，KLM计算出许多采样试验（向上取整），将试验计数器N初始化为0。然后，在每次试验时 ，KLM会执行以下步骤：

1. 如果当前的样本赋值不存在，则以概率随机选一个子句Ci，然后利用变概率分布随机生成一个可满足的Ci分配。
2. 检查当前赋值是否满足随机选择的子句Ck。 如果满足，N增加并生成一个新的样本赋值。 否则，什么都不做。

KLM返回作为加权DNF计数的估计值。 由于分配检查的时间复杂度是O（n），因此KLM的复杂度为。

（s,i）,指s∈Di.

**GNN：**

图神经网络（GNN）是专门设计用于处理结构化图形数据的神经网络。 在GNN中，每个图节点x都由一个不断更新的向量表示，。 节点x从其邻域N（x）接收信息，它是由一条边连接到x的节点集合。 令表示迭代t时的值。节点更新如下所示：



其中Combine和Aggregation是函数，Aggregate是置换不变的。使用层归一化的LSTM [1]作为我们的combine函数，将sum作为我们的Aggregate函数。 这有点类似于门控图神经网络[21]，不过使用了层归一化的LSTM代替了门控递归单元（GRU）[7]。所有迭代终止后，使用最终的节点表示来计算目标输出。

将DNF公式编码为具有文字层，合取层和析取层的3层图形。在文字层中，每个DNF变量都由与其正负文字相对应的2个节点表示，这两个节点由（虚线）边连接以突出显示它们是互补的。在合取词层中，每个节点都代表一个合取词，并连接到出现在该合取词中的文字节点。最后，析取层包含单个析取节点，该节点与合取层中的所有节点连接。为了估算#DNF，我们使用消息传递GNN模型，该模型在相应的DNF图上进行迭代并返回高斯分布。首先，网络使用多层感知器（MLP）在给定它们的概率的情况下计算给定所有文字节点的向量表示形式。将具有概率pi的文字xi的k维表示计算为。合取和析取层中的节点分别初始化为两个表示向量vc和vd，并在训练过程中学习这些向量的值。初始化之后，在T次消息传递迭代中更新节点表示形式

**消息传递迭代包含一下4个步骤：**

a：文字层节点使用MLP 计算消息，并将消息传递到其相邻的合取层节点。 然后，这些合取节点使用sum函数汇总这些消息，并使用归一化的 层更新它们的表示。 更新的合取节点表示形式表示为，

b）连接层节点通过MLP Mc计算并向分离节点发送消息。分离节点使用层标准LSTM Ld聚合这些消息并进行更新

c）析取节点使用MLP Md计算消息并将其发送到合取节点，合取节点使用不同的LSTM单元Lc2更新其表示

d）合区节点使用它们的最新表示，将消息发送到文字层中的相邻节点。文字层节点聚集这些消息，并将它们与其对应的否定文字的消息连接起来（用∣∣表示）。 然后，他们使用此消息使用层标准LSTM  更新其表示形式。

图2提供了一个简单公式的4个消息传递步骤的直观表示。在此协议中，我们使用2个不同的LSTM单元Lc1和Lc2来更新步骤（a）和（c）的合取节点的表示，因此 网络分别为基于文字和基于析取的更新学习了更新过程。 在消息传递结束时，最终的析取节点表示通过传递。 该MLP的最后一层由两个神经元和组成，这两个神经元分别返回预测的高斯分布的均值和标准差。

**损失函数：**

给定ε和δ，KLM会在相对于ε的乘法边界内返回真实模型计数μ的估计μ^，并且该边界以1-δ的概率成立。但是乘性边界区间使其很难适应标准分布。因此，对该边界使用对数得到logμ上的加性边界：

通过设置，我们可以将高斯分布拟合到该边界，其中表示标准高斯分布的逆累积分布函数。累积分布函数是通过分位点得到累积概率，逆累积分布函数是求得累积概率得分位点。因此，实际上是训练GNN以预测logμ，一个负的值。为了将nμ的输出限制为负，nσ的输出限制为正，使用调整过的ELU激活函数：

****应用于nμ与nσ。

对nμ使用-ELU+1(x),nσ使用ELU+1(x)。

为了比较预测的高斯和KLM结果，我们使用Kullback-Leibler（KL）散度.对于两个高斯分布N1（µ1;σ1）和N2（µ2;σ2），其散度为：



这是一个一阶正态分布KL散度的公式，这个散度描述了将N1拟合到N2的损失。

我们将N1设置为网络返回的预测，将N2设置为KLM近似值。此选择至关重要，这是为了避免系统通过学习产生非常大的σ2值来使训练损失最小化。

### 第八周 11月11日~11月17日

本周准备进化计算考试占用较多时间。

**DNF计数实验部分：**实验部分论文中记录了结构泛化能力、规模泛化能力、迭代次数消融实验、运行时间以及验证数据生成过程的实验。在我们的实验中，我们将网络的预测μ尖与KLM的预测进行比较，并检查它们的绝对差是否在预定的加性阈值之内。（消融实验：因为作者提出了一种方案，同时改变了多个条件/参数，他在接下去的消融实验中，会一一控制一个条件/参数不变，来看看结果，到底是哪个条件/参数对结果的影响更大。）

**初始化：**我们在P100 GPU使用KL发散损失训练4次系统,使用学习速率为λ =10的-5次方的Adam优化器[19],梯度剪切率为0.5（梯度剪切这个方案主要是针对梯度爆炸提出的，其思想是设置一个梯度剪切阈值，然后更新梯度的时候，如果梯度超过这个阈值，那么就将其强制限制在这个范围之内。这可以防止梯度爆炸。）,T = 8消息传递迭代次数。

使用整流线性单位（ReLU）作为MLP隐藏层的激活函数，并使用输出层对除fout以外的所有MLP进行线性激活。

**泛化能力：**

在结构泛化能力测试中，网络预测与KLM的预测非常接近，如图3所示。该模型在KLM WMC估测阙值为0.02时，测试集的准确率超过87.37％，阈值为0.1时该比例上升到99.95％。该模型还可以在不同的n上始终保持一致的性能，对于所有四个测试阈值，在任意两个不同的n值之间，精度的差异最大为4.5％。

在所有阈值下，培训和测试准确性之间的接近程度表明该网络尚未拟合或记住其训练公式，而是学习了常规的WMC过程。表3中提供了参数为n的结果。这些结果表明，网络在所有n个值上均保持较高的准确性（例如，阈值0.05的95：5％），因此不依赖于特定的n来实现其较高的整体表现。值得注意的是，该模型还可以抵抗ω的变化。如表4所示，对于阈值0。05和0。1，模型在所有宽度上的得分分别高于96％和99％。有趣的是，它在加权模型计数接近零的较大宽度13、21和34上具有近乎完美的性能，并且在w = 3（计数几乎为1）时，在阈值0.02处具有相对较高的精度。 在这两种极端情况下同时表现良好，再加上中间宽度的高精度，进一步突出了我们模型的稳健性。

**规模泛化：**

在尺寸泛化测试上，该模型在10K和15K变量公式上的精度分别为97：13％和94：83％，阈值0.1，尽管变量的数量是训练中的三倍。表5给出了尺寸泛化的全部结果。表6还给出了由宽度w作为参数的相同结果。这些结果表明，网络在宽度3、5和8上的性能一致，但在w = 13时性能较差 这是由于w = 13的公式在n和m的这个比例下显示出“相变（相转移）”。的确，在这种情况下，模型计数会发生剧烈波动，因为m与p的-1次方处于相同的数量级，其中p表示子句的预期满意度。在训练集中，这种现象发生在较小的宽度上，但对于w = 13则永远不会出现，因此对于较大规模的模型，这是全新的情况。但是，对于阈值0.1，它达到了令人满意的82.5％的准确度。

这些结果表明，即使训练仅限于较小的公式，也可以在大型公式上实现可靠的近似模型计数。从实际的角度来看，这提供了进一步的证据，可以使用较小的公式来训练大规模求解器，这些较小的公式用现有求解器轻松地标记。在不同生成的数据集上进行的其他实验中（请参见附录B），我们的系统也保持了很高的性能，实际上，在完全随机的公式[22]上比在我们的协议生成的公式上表现更好。这进一步突出了我们的GNN的鲁棒性，并验证了我们的数据生成过程的质量。

**迭代次数消融实验：**

在我们的消融实验中，仅通过2条消息传递迭代来运行时，GNN在所有实验中的表现都明显较差。但是，这并不意味着随着消息传递的增加，性能总是会提高。实际上，运行太多的消息传递迭代会使系统易于过度拟合：当运行32个迭代时，系统在结构泛化方面实现类似的性能，但是在大小泛化方面其性能会大大下降。这表明在我们的实例中，必须选择消息传递迭代的折衷值，以实现足够的通信而又不导致过度拟合。

**时间分析：**在一般情况下，我们的GNN在O（m-ω）中运行，其中-ω表示平均公式子句宽度。相比之下，KLM在O（nm）中运行，因此在实际情况下KLM要慢的多，这是因为-w《n。

我们在表7中展示了GNN与KLM（= 0：1，δ= 0:05）在公式设置为（w = 3，34, m = 0.75n）的运行时间。实际上，KLM需要w：34的7.62秒；n = 1K，对于w = 34，它迅速上升到305.61s；n = 15K，而GNN分别仅需要0.02和0.223秒。这是因为GNN利用有限的宽度来提供线性可伸缩性，而KLM使用n和m二次缩放

GNN在较小的宽度上不会像KLM一样慢，因为它不依赖于采样。使用KLM，当随机分配满足子句时便会被替换，这意味着随着子句宽度的减小，随着子句满足的可能性更大，将进行更多替换，这会导致大量的计算开销。例如，KLM需要2375秒（约40分钟）来运行n = 15K且w = 3且使用ε= 0.1和δ= 0.05的公式，而当w = 34时仅需要306s。相比之下，GNN分别仅需要0.104和0.223秒。

**分析模型：**为了检查网络如何进行预测，我们选择了21个从结构测试集中获得的公式fi：i∈[1,21]，其加权KLM模型计数约为21 20-i。然后，我们根据这些公式运行网络，并在每次消息传递迭代结束时计算出预测概率。结果如图4所示。最初，网络以较低的估算值开始。然后，在前3次迭代中，网络累积概率并达到一个峰值，并且可以将其映射到来自文字节点的消息，最终到达析取节点。此后，网络将其估算值降低，最后在最终迭代中对其进行优化。

这个网络的估计值剧烈波动。初始进行较大的估算，然后进行缩减和优化。这样做时，网络似乎一开始是估算合取概率的简单的和，然后迭代使它可以更好地捕获合取之间的交集。这与我们的观察结果一致，因为对交集的任何了解只能从第三次迭代开始进行，这时析取节点和合取节点将彼此传递更多的全局信息。

这也解释了在我们的消融研究中观察到的有限性能：仅进行2次迭代，系统就无法捕获合取析取，因此只能进行幼稚的估计。

**总结：这个GNN模型做出了以下贡献：**

1 它以为单位，在有限宽度的线性时间内，生成高效、高精度的加权#DNF近似值

2 它可以可靠地扩展到包含多达15K变量的#DNF实例，据我们所知，这是神经符号方法的首例

3它是健壮的，因为它可以在训练之后生成给定域上的任何问题实例的近似。

这项研究展示了GNN能够学习解决计算上的难题，前提是这些问题可以在多项式时间内近似。

### 第九周 11月18日~11月24日

**《DNF》**

使用KL散度设计两个正态分布结构的损失函数。

DNF计数这篇论文借鉴了他人关于DNF计数问题的较新的优秀算法，并搭建图神经网络来拟合、优化该算法。这或许为我的论文了思路，去理解问题的理论要点并将数据建模为一种容易表示的结构，并用适合处理这种结构的神经网络来进行训练，以拟合较新的对应的算法，利用神经网络的特殊结构以更少的时间得到目标结果。

设计了各种泛化实验，对于优化问题应该是一个思路。

利用消融实验来验证论文的创新点。本文根据思想对关键变量设计了消融实验，验证了关键部分到底起了多大作用（迭代次数T），并保存了不同样本的每一轮迭代得到的结果来分析神经网络的运行原理，以图示表示出来，非常直观。

本实验的数据集是作者自己开发的随机产生DNF公式的数据集。其中变量的概率是均匀随机选择的。对训练集和评估集中的每个公式使用4个分布，这样一个分布是随机生成的，而其他3个分布是随机分布模1的四分之一增量。这保证了生成的公式中模型计数尽可能均匀地覆盖整个[0,1]范围，从而获得更多有代表性的训练和测试数据。可以作为建立训练集的一种思路

### 第十周 11月25日~12月1日

**《NeuroSAT》**

NeuroSAT主要是对CNF公式进行求解，给定CNF问题P，预测P是否为可满足（可满足时φ（P）为真），然后从中解算出一个解。

从模型结构来看，和之前看的DNF结构十分相似。这个神经网络结构在这里称作信息传递神经网络（MPNN），使用了消息传递机制，每个子句（或文本）从其相连的文本（或子句及补码）接受消息来更新嵌入，与#DNF的三层GNN结构有些相似，那个三层GNN也是子句、文本、文本补码之间的相互消息传递来更新。还有一点相似的是，其传入消息的形成也是将所有邻居的传出消息求和（SUM），应该是一种常见做法？

****

对一次迭代的工作作个简要总结：L(t)第i行包含文字li的嵌入，C(t)第j行包含子句cj的嵌入。Lh(t)、Ch(t)分别为t时刻归一化LSTM Lu、Cu的隐藏状态。M为邻接矩阵（文字li在子句cj中存在，则M（i，j）为1），定义操作符Flip将矩阵L的每一行与文字的否定对应的行交换（应该是生成L的补码矩阵）。更新如下：



归一化LSTM：Cu接受t时刻的其隐藏状态、经过多层感知机Lmsg处理的L（t）文字分配矩阵与邻接矩阵M的转置的乘积两者来更新Cu在t+1时刻的隐藏状态及子句嵌入矩阵C（t+1）。

然后层Lu接收其t时刻的隐藏状态Lh（t）、接受L（t）补码矩阵、经过Cmsg处理的t+1时刻子句嵌入矩阵C（t+1）及邻接矩阵M的积来更新t+1时刻的L（t+1）及隐藏状态Lh（t+1）。

（还没搞懂这里M的作用，暂且当作一种方法）

对于第T次迭代以后，用层Lvote对得到的L（t）矩阵处理得到L\*（T）包含每个文本的投票（对是否可满足的置信值），然后计算文本投票的平均值 y（T）。训练神经网络最小化y（T）与φ（P）之间的sigmoid交叉熵损失（即损失函数的设定）。（经过查询，sigmoid函数常用于二元分类以及激活函数，而softmax常用于多元分类，而softmax在二元分配时退化为sigmoid）。

其在图中展示一开始投票都以低置信度投出不可满足票，然后开始出现一些分散的文本头可满足票。在一个突然出现的阶段转换，所有文字突然开始以高置信度投可满足票，然后投票收敛，网络停止演化。

其训练数据的产生是通过一个分布SR（n）产生的。SR（n）的特性是：产生一对元素，其中一个元素可满足、另一个不可满足，两个元素的区别只在于其中某一个文字的正反。SR（n）随机抽取一个平均值略大于4的整数k，然后随机均匀不替代的抽取k个变量以50%的概率取反。不停以这种方法生成子句ci并添加进SAT问题中，直到添加子句cm使问题不可满足为止。这时，对cm取反又能使问题{c1....c’m}可满足，而{c1.....cm}不可满足。这两个都是SR（n）的样本。

上述的产生数据集的方法也是一个例子，与#DNF中生成数据集不同，这个方法同时生成了可满足与不可满足的样本，因为目的是训练MPNN使之学会预测可满足(应该)。

### 第十一周 12月2日~12月8日

上周M的作用：

M作为邻接矩阵，在第一行中M的转置从L（t）中分别筛选出与文字i相连的所有文字及其嵌入。第二行中，M分别筛选与文字i有关的子句的嵌入。

回到实验部分，实验使用ADAM优化器以的学习率训练我们的模型，以0.65的剪裁比按全局范数剪裁梯度。我们一起对多个问题进行批处理，每个批处理最多包含12000个节点（即文本加子句）。Adam优化器经常出现，且学习率量级基本在10^-5左右。Adam实现简单计算高效，对内存需求少，参数更新不受梯度伸缩变换影响等，是综合来说默认工作性能比较优秀的优化器。梯度剪裁也多有使用，根据导数结果设置剪裁比将导数限制在一个范围内防止梯度爆炸。

有关解码得到可满足分配结果的方法，我还不是很理解。只知道是利用KMeans聚类算法。随机确定2个初始点作为簇中心; 为每个数据分配簇[计算每条数据和簇中心的相似度,分配到最相似的簇上];根据簇中的数据点对每个簇中心进行更新.反复重复直到收敛为止。此部分完全没懂。

在实验部分它同样考虑了规模更大的问题。对于规模更大的问题，想要得到相对较好的结果，需要成倍的增加迭代次数，而且提高的成功率有限，但同样的在SR（40）上运行更多的迭代也能解决SR（200）中25%的问题，在自己设计实验时，应当考虑这种泛化能力。

对于不可满足的问题，NeuroSAT会无限非系统地寻找可满足的解。为了解决这个问题，NeuroSAT需要学会检测一个unsat核，是构建解析证明更有效。

Tensorflow的一些 操作笔记：

TensorFlow将中间节点及节点间的运算关系（OPS）定义在自己内部的一个“图”上，全通过一个“会话”（session）进行图中OPS的具体运算。

1.定义输入节点（即网络入口），用于训练的模型参数以及模型中的节点。

TensorFlow一般通过占位符定义输入结点,使用X=tf.placeholder(“float”)

或者字典类型定义输入节点,一般用于输入较多的情况：

Inputdict={

‘x’:tf.placeholder(“float”),

‘y’:tf.placeholder(“float”)

}

还有直接定义（最少用）。

2.定义学习参数变量

直接定义：W = tf.Variable(tf.random\_normal([1]), name="weight")

b = tf.Variable(tf.zeros([1]), name="bias")

直接定义较为不常用

字典法：paradict={

‘w’:tf.Variable(tf.random\_normal([1])),

‘b’:tf.Variable(tf.zeros([1]))

}

### 第十二周 12月9日~12月15日

本周花了很久装tensorflow，还有各种小问题出现，windows超多warning但可以运行（但sh文件好像只能在linux运行），linux上总显示缺少tensorflow.contrib.rnn，推测是NeuroSAT的python版本更高导致的，还待验证。

加之，准备英语考试，开始复习期末两门课，本周基本没有进展，并且今后应该也会越来越慢。

NeuroSAT的框架是tensorflow，下面对所有文件功能做个记录，防止忘记。下面是看NeuroSAT文件的内容，一边查询一边理解python语法的记录过程。

util.py:一个定义方法的py。五个方法：

repaeat\_end:输入val，n，k，由n次循环产生一个有n个val的列表，并在最后加上k。

lambda：定义一个匿名函数。

Reduce\_with:不知道要干嘛，看不懂。

Decode\_final\_reducer运用降维函数tf.reduce\_\*函数根据输入对张量进行1,2维度x上的求最小最大、求和以及平均值的操作。

Decode\_msg\_reducer则是对消息的削减函数，目标是一个在维度0的通过tf.concat拼接的列表，求值的维度也是0。

Decode\_transfer\_fn定义了激活函数，根据输入返回tensorflow中不同激活函数的位置。

NeuroSAT.py 记录：

\_Init\_方法：NeuroSAT类的初始化方法。第一参数是self（类实例本身）初始化设置了opts。调用了bduild\_network(),通过了build\_network调用的一大堆方法初始化了网络（大概）。

Build\_network:能对以后编程有用的语法tf.set\_random\_seed设计图级随机数，np.random.seed设置相同随机数。声明参数声明占位符（tf.get\_variable,tf.placeholder）.

### 第十三周～第十七周 12月16日~1月19日

**Tensorflow**环境安装出问题是CPU版本太旧不支持AXV编码格式，导致tensorflow1.6.0及之后版本无法运行。通过翻墙在外网找到了热心网友魔改的tensorflow1.6.0版本（使用了某种编码），解决了这个问题。

**中间程序调试：**

该程序本该是在linux上运行的，程序自带了setup等sh文件来简单运行，我根据其中的代码尽量还原了原来的使用环境。

出现了同级目录无法相互导入的情况。解决方式：将整个文件夹设成了source root，部分文件夹正常。

Git了别人的API PyMiniSolvers,已安装。Setup里用了linux中的make指令来编译makefile文件，windows需要下载MingGW后，改名为make来使用，但不明原因导致make出错(缺少zlib？系统文件路径不明？安装并编译zlib失败等等)。而导入了PyMiniSolvers项目后程序报错尽管已经消失，没有makefile程序不能运行，所以决定回学校用linux系统再跑项目。

总结：windows还原linux环境真难。

**NeuroSAT的执行顺序：**gen\_sr\_dimacs.py与dimacs\_to\_data.py ->train.py->test.py->solve.py

**gen\_sr\_dimacs.py:**

通过python命令通过这个py文件分别创建train数据集和test数据集合。

1.使用：if \_name\_==”\_main\_”来使执行该文件时，执行if的语句。如果是其他程序调用该文件，则if的语句不执行。

2.在if中 使用argparse模块创建了一个ArgumentParser对象（命令行解析器），通过add\_argument添加参数并通过parse\_args解析参数。random.seed(输入T)（返回0到T之间的一个随机值）。根据命令行输入的值进行创建子句对。通过gen\_iclause\_pair，首先生一个随机的整数n，通过minisat生成n个变量，然后通过调用generate\_k\_iclause来这n个变量中随机抽取一些变量通过minisolver通过add\_clause添加子句，并通过solve方法判断是否可满足，如果可满足则把该子句加入子句集中;若出现了不可满足的子句，则将该子句加上一个已经出现的否变量变成一个可满足的子句加入子句集中，而不可满足的子句放在一个不可满足的子句集合iclause\_unsat中随着子句集一起返回。

3.最后通过mk\_out\_filenames根据n的大小等变量创建文件，并将子句集中的子句写进文件中。

**边缘概率分布：**

**边缘分布**（Marginal Distribution）指在概率论和统计学的多维随机变量中，只包含其中部分变量的概率分布。也就是说给定一组变量的联合概率分布，定义在其子集上的概率分布被称为边缘概率分布。

一般对于离散型随机变量，我们一般采用求和法则来计算边缘概率。假设有一个和两个变量相关的概率分布：

https://bkimg.cdn.bcebos.com/pic/a6efce1b9d16fdfab41a9a68bf8f8c5495ee7bc5

关于其中一个特定变量的边缘分布则为给定其他变量的条件概率分布：

https://bkimg.cdn.bcebos.com/pic/42166d224f4a20a480f860419b529822730ed056

而对于连续型概率变量，我们通常使用积分代替求和：

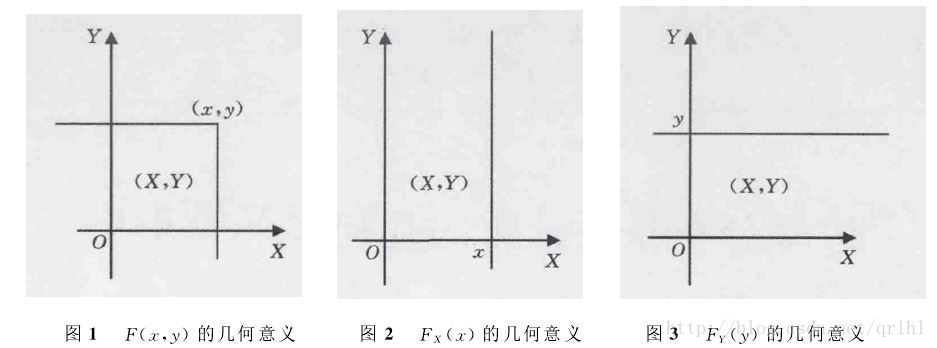


在这个边缘分布中，我们得到只关于一个变量的概率分布，而不再考虑另一变量的影响，实际上进行了降维操作。在实际应用中，例如人工神经网络的神经元互相关联，在计算它们各自的参数的时候，就会使用边缘分布计算得到某一特定神经元（变量）的值。

二维随机变量(X,Y)可视为平面上的随机点（为方便，随机点与随机变量可视为等同），二维随机变量(X,Y)的（联合）分布函数F(X,Y)=P{X≤x,Y≤y}的几何意义显然是随机点(X,Y)落在平面区域{(x,y)|X≤x,Y≤y}内发生的概率，见图１。

二维随机变量 (X,Y)关于X的边缘分布函数FX(x)表示平面上的随机点(X,Y)落在平面直线X=x 左边平面区域内（含直线）{(x,y)|X≤x,Y<∞}发生的概率，见图２。

(X,Y)关于Y的边缘分布函数FY(y)FY(y)表示平面上的随机(X,Y)(X,Y)落在直线Y=y下边平面区域内（含直线）{(x,y)|X<∞,Y≤y}发生的的概率，见图３。



### 第十八周～第二十一周1月20日~2月16日

python刷题：1.找出数组中和为目标值的两个下标返回：len(Array Name),

for <variable> in <sequence>:<statements>

else:<statements>，

range(a,b)or range(a) or range(a,b,c)//c为a、b之间的增量，

使用hashmap快速定位数组中寻找的值：enumerate()将可遍历的数据对象组合成一个索引序列，同时列出值和下标

2.两个逆序链表返回加和：ListNode和and or

3.无重复字符的最长字串：遍历字符串，生成空数组、字符串[]’’,向其中添加。使用list和头尾记号模拟队列

4.整数反转：取整除符号//，数组初始化a=[0 for x in range(x)],乘方函数pow

5.O(log2(m+n))的时间复杂度内寻找两个有序数组的中位数：1python中的else if：elif,格式化输出：print(“%d...”%(i,..)),看见时间复杂度带log的一般用二分法

6.寻找最大回文字符串：切片方法，中心扩散法(确定回文中心，然后左右扩散,根据回文中心左右的必定相等)

7.给定字符串按照从上到下，从左到右的顺序进行Z字排序，排序完按照从左到右，从上到下的顺序进行遍历：多维数组初始化

8.判断给定整数是否是回文数：切片方法，整数转字符串

9.给定一个字符串满足以下条件：排除空格后的第一个字符是+-以及数字以外的情况，并无视满足形式的数字串后面的无关串，在指定最大范围内将字符串转化为数字。使用正则表达式：^匹配字符串开头，[\+\-]一个+或者一个-，?表示前一个字符可有可无，\d一个数字，+表示前一个字符的一个或多个，\*表示前一个字符的0个或多个。

10.盛最多水的容器：给定n个坐标（i，ai），求min（ai,aj）\*(j-i)的最大值。

双指针法：在最左和最右初始化各一个指针，由于水的面积与两边中短的那一边有关，则每次将较短的那条边向内移动，直到两指针相遇。这样两指针必定同时经过面积最大的边。（暴力法超出时间限制）

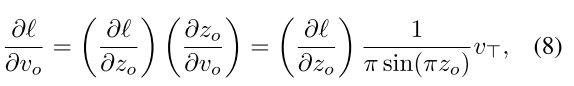
**SATNET（接第六周）**

**3.3 计算反向传播**

作者通过SATNET层推导反向传播更新使其集成到一个神经网络。给定关于层输出的损失函数l的梯度计算关于层输入的梯度和层权重的梯度**。**由于显式展开前向遍历计算并存储中间的雅可比矩阵在时间和内存方面效率低下，作者转而使用了一种有效的坐标下降算法，推导出解析表达式来直接计算所需的梯度。

3.3.1从概率输出到它们的连续松弛

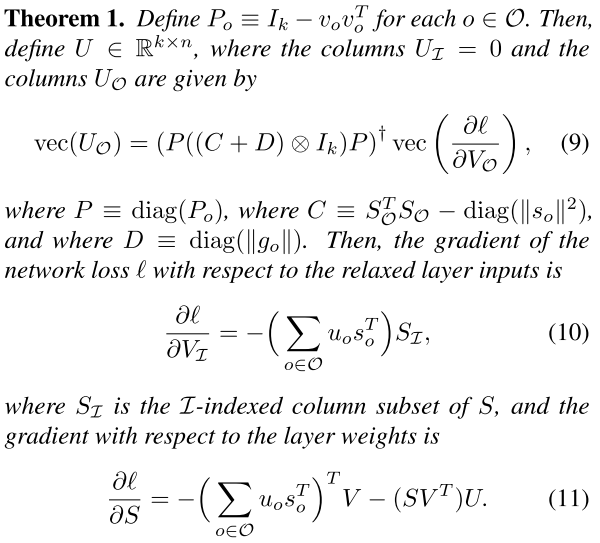
给定关于层输出的梯度，对每个o∈O，有



其中通过等式（7）求微分（或更容易地，通过其重排隐式求微）来获得。

3.3.2 通过SDP反向传播（以下看不懂）

得到关于输出松弛的梯度的解析形式，求导出关于输出松弛的梯度和关于权的松弛。

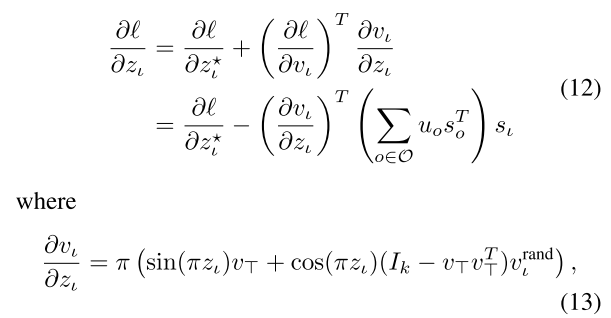
他们将此梯度结果的解析形式归纳为一个定理：(???????)

推导过于复杂看不懂，高层概念为：根据问题的参数和输入来区分SDP问题的解决方案（第3.1节），这需要计算（相对大的）矩阵向量解。（如式9所示）

而式9的求解使用了坐标下降法。

3.3.3 从松弛的输入到原始的输入

使用上一节得到的关于输出松弛的梯度来获得源输入梯度。



且其中A捕获l对B的任何直接依赖关系（与通过v的依赖关系相反）。

总的来说这一章讲解了从将离散输入松弛为连续输入并通过SDP松弛得到松弛输出然后转化为离散输出的过程，再将上述过程反向来推导出反向传播过程，以解决MAXSAT问题。

### 第二十二周～第二十五周3月23日~3月29日

没提就是不懂

Neurosat：

Confusion.py:

定义了混淆矩阵,用于评价模型的优劣。分别定义了True Positive,True Negative,False Positive和False Negative的矩阵情况。并定义了根据所有4种情况的出现的次数获得的准确率百分比来评价模型的正确率。

准确率所需要的情况是预测true事实true以及预测false和事实false，使用这两种情况（True Positive与False Negative）出现的次数除去所有情况出现的总数来获得准确率。

Solver.py:

给定文字和子句，调用github上的PyMiniSolvers包的Minisat求解器来判断是否可满足，返回求解器的的统计信息字典。

Mk\_problem.py:

Assert函数：检查条件，不满足就终止程序

Np.zeros函数：返回一个给定形式的用0填充完的数组

Enumerate（）：将给定的数据转换成索引形式

List.extend():在列表的末尾加入一个序列

List.append（）：在列表末尾加入一个对象

这个类定义了problem这种数据块里的内容，包括vars的个数，子句，是否可满足等内容。

Util.py:

Lambda：定义一个匿名函数

Tensorflow的循环体while\_loop中，循环包括（终止条件，循环体，循环参数），而终止条件和循环体都是函数，故常用lambda来定义。

应该是定义了一个对数组做reduce的工具。根据需求选择调用Tensorflow的reduce\_min,reduce\_max等函数，计算张量的各维度上元素最大最小，平均总和等；并根据激活函数的需求调用relu、tanh、sigmoid等激活函数。

Mlp.py:

多层感知机的定义

Tensorflow variable\_scope:用于定义创建变量（层）的操作的上下文管理器,验证values是否来自同一图形，确保图形是默认的图形，并推送名称范围和变量范围。

Tensorflow get\_variable:创建变量或张量，name:新变量或现有的变量名称；shape:新变量或现有变量的形状；intializer:创建变量的初始化器

Tf.contrib.layers.xavier\_initializer:作为初始化器保持每一层的梯度大小都差不多相同。

Tf.zeros\_initializer:生成张量初始化为0的初始化器。

Tf.matmul：将输入的两个矩阵相乘，返回其乘积

这个文件定义了多层感知机中w值和b值的更新，并定义了前向传播过程，让传入的x值与W值相乘后加上b值，并根据需要为x用激活函数处理后再进入下一层循环。

### 第二十六周 3月30日~4月5日

Gen\_sr\_dimacs.py:

Write\_dimacs\_to:根据输入 的文件名打开文件，并将cnf公式的变量数和子句的数量输入，然后将所有生成的子句写入文件，最后换行

Mk\_out\_filenames:根据opts来生成要打开的文件名

Generate\_k\_iclause：

np.random.choice(n,size=min(n,k),replace=false):从0到n中以一致分布的概率随机不放回的抽取min（n，k）个数字组成数组，然后以一半的几率返回其中的一个数字v+1或者-（v+1），以此来生成k个子句

gen\_iclause\_pair:抽取一个随机数n，然后利用minisolvers生成n个变量，生成随机数k，调用上面的generate\_k\_iclause返回一个数组，利用该数组调用solver根据上面生成的变量生成子句，并用solver.sovle()判断是否可满足，若可满足则添加进可满足的子句列表，否则结束。最后根据可满足的子句列表，为其子句的反来得到一个不可满足的子句列表，并添加一个变量来生成与该不可满足子句列表对应的可满足子句列表，以生成子句互补对。

该文件生成了子句及子句互补对，创建了输出文件并将子句的各种信息写进了文件中。

Dimacs\_to\_data.py:

Parse\_dimacs:打开文件，每次读一行，对该行以空格切片，提取每行保存的变量个数，以及子句集，并返回

Mk\_dataset\_filename:根据opts生成数据集的名称

主要生成数据集，后面有加强批处理的程序没太懂

Problems\_loader.py:

用来调转加载文件。根据现在运行的文件的名字，判断有没有下一个文件，有则获得下一个文件的文件名并返回以供访问

NeuroSat.py：

Declare\_parameters:声明参数

参数更新选择LSTM神经网络

Declare\_placeholder:声明占位符

tf.placeholder作为一种占位符用于定义过程，在具体使用时再赋具体值

Tf.shape返回一个一维整数Tensor用以表示参数的shape

Build\_network:

选择随机核，创建tf.session,声明参数和占位符，定义了传递消息的方式，计算logit和cost，创建最优控制器，初始化变量等

### 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。

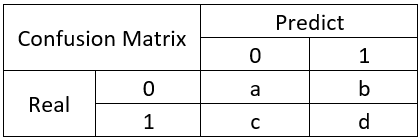
最后计算损失。通过tensorflow的这个函数计算给定的logits与标签之间的sigmoid交叉熵损失，下面这一块我也没有太明白是哪里的损失，可能是训练过程中训练每个变量的损失？总损失就是这两个损失的合。

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。

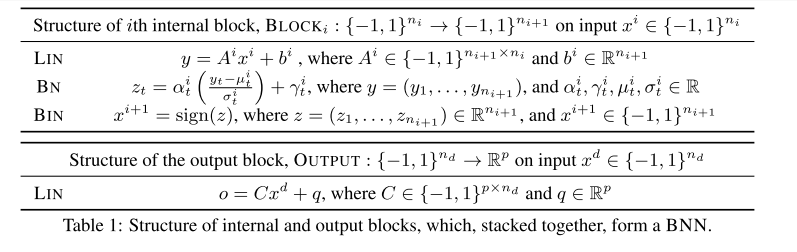
### 第二～五周 4月13日~5月10日

**《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。

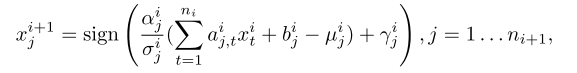


一个BNN由内含多个层的块组成，每个块将二进制向量映射到二进制向量。中间的每个块用BLOCKi表示，而最后一个块用OUTPUT表示，这个块将二进制向量映射到输出。每个BLOCKi都应用了三个变换：线性变换LIN，批量归一化BN，二值化BIN。

**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。

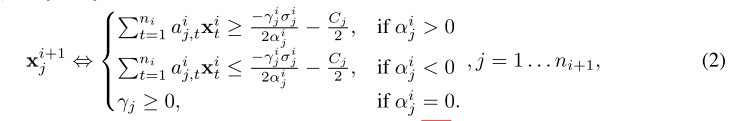
对于上述的LIN，BN，BIN三层，可以直接将一个块编码为其组成：



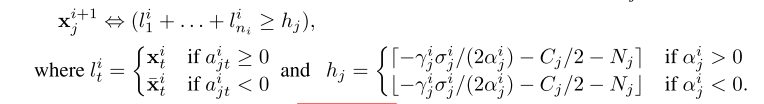
即将前一层的输出代入到下一层中。其中Xi是块输入，Xi+1是块的输出。为了获得SAT编码，我们需要消除二进制变量。用一个布尔变量套用公式来替代二进制变量，则原公式可转化为



用Cj表示，则最终可以得到布尔表示的公式：



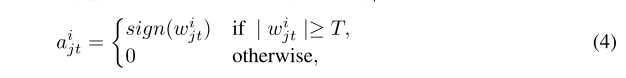
使用一个重言式（恒真式）将上述公式变为类似SAT的基数约束的形式得到：



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**

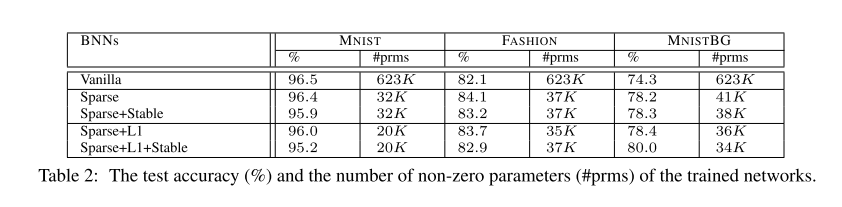
**神经元水平**：从单个神经元来看，A是一个LIN层的系数a的稠密矩阵，所有条目都必须是1或-1。A的高密度使方程3的基数约束中含有大量的变量，这导致了较大的SAT表示。同时神经元的值和基数约束之间有一个等价关系，这个等价关系可能会影响SAT解算器的推理过程。因此我们通过将A三元量化，来使A的系数变得稀疏，以减小SAT的编码大小，具体表示如下



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。

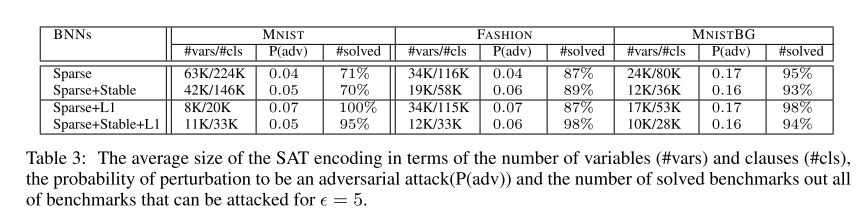
第二个问题是等价关系，解决方法是通过网络进行输入边界的边界传播。我们根据方程式估算每个线性约束的上下界，若两个边界都为正则xi为1，否则为-1，以此来稳定神经元。这样我们就获得一个易于SAT求解的BNN网络。

**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：

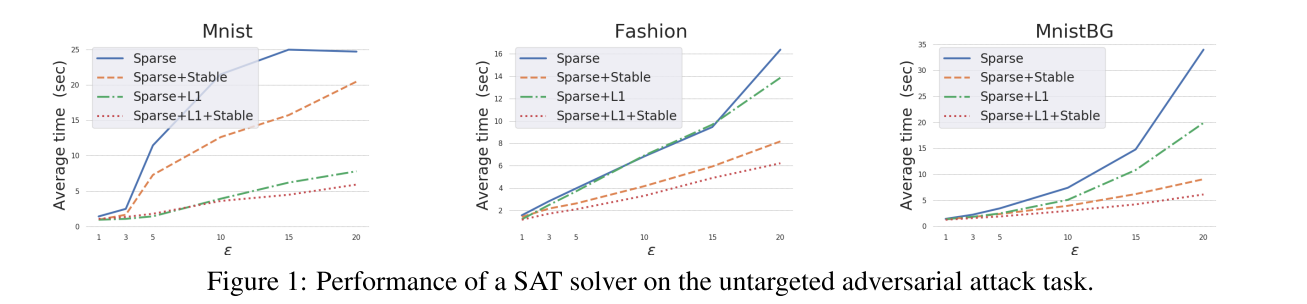


图1 展示了不同模型在不同epsilon下的平均求解时间。可以看出进行稳定化后的模型相比不应用的稀疏模型的求解时间显著变短，速度增加到3倍至6倍。没有原始BNN的结果是因为原始BNN使PYSAT编码时占用的内存过大，时间太长。

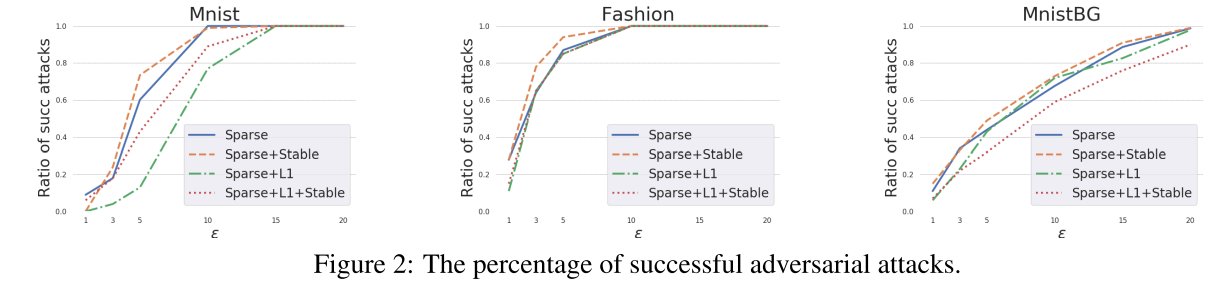


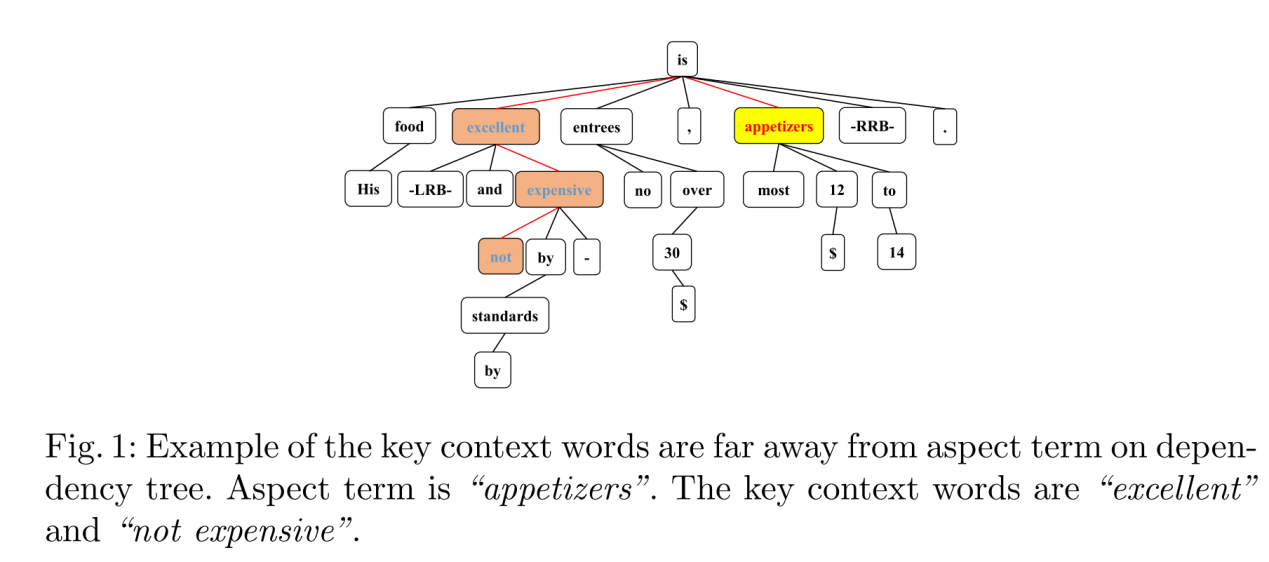
图2用成功进行对抗攻击的比率来表示模型面对对抗攻击的鲁棒性。可以看出随着epsilon增大这个比率稳定到1。在FASHION和MNISTBG下使用L1正则化器并不会显著改变网络的漏洞，但L1对MNIST有帮助.而在比较进行稳定化的网络和不进行稳定化的网络是，稳定化会导致漏洞增加。比如在MNIST数据集下，epsilon等于5时，只有13%的攻击样本成功对稀疏+L1网络进行了攻击，而又43%的样本成功对稀疏+L1+稳定化的网络进行攻击。

**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

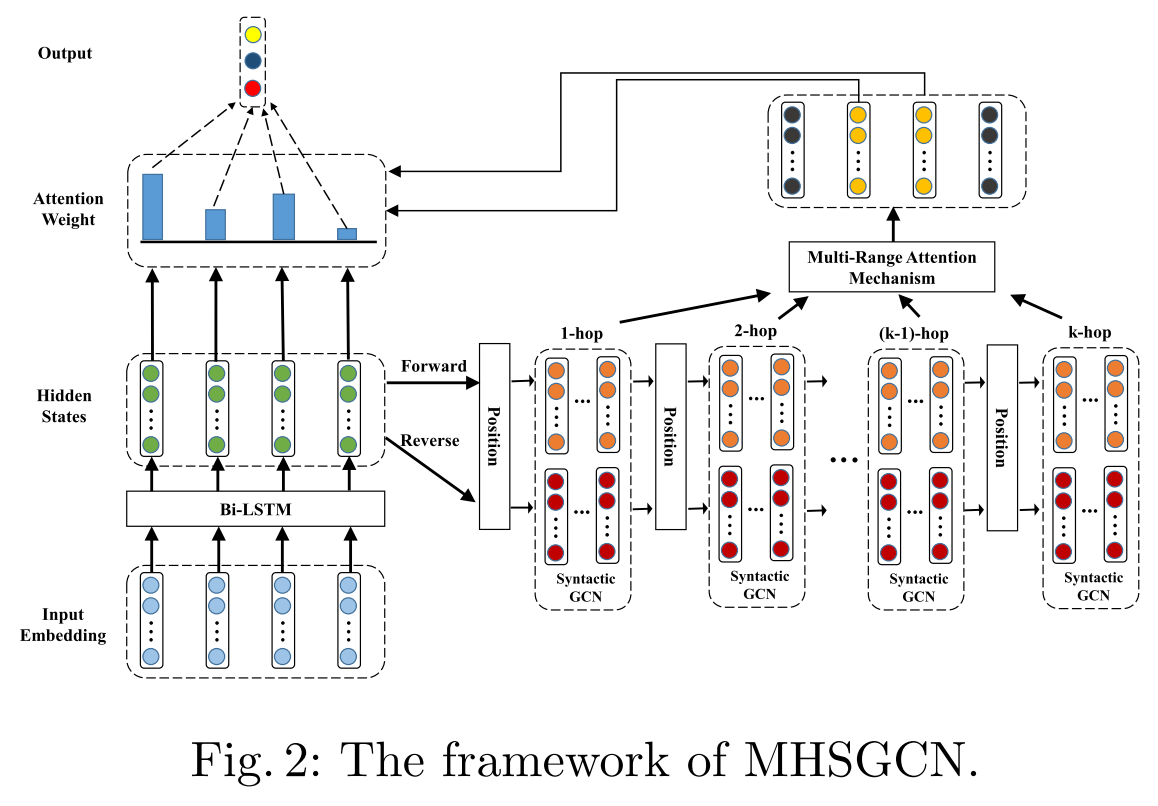
《Multi-hop Syntactic Graph Convolutional Networks for Aspect-based Sentiment Classification》

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**

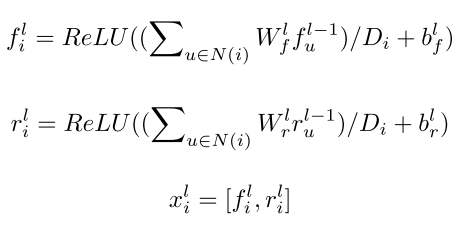
基于方面的情感分类的任务定义如下：给定n个单词的句子，以识别特定方面项的情感极性。 m个单词的方面项是从句子中的第（v +1）个单词开始的句子的子字符串。 多跳句法图卷积网络（MHSGCN）的总体框架如图2所示。



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。

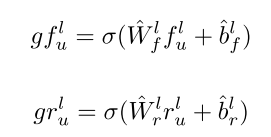
**句法GCN**：

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：

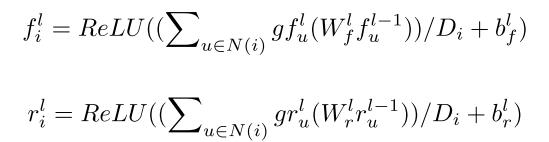


f是前向的结果，r是后向的结果，x是两者的聚合。D i代表第i个节点的度，N（i）代表节点i的邻居节点集。 W是可训练的权重矩阵，而b是可训练的偏差项（d n是隐藏状态的维）。

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：

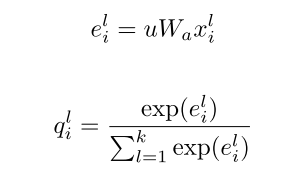


其中W是两个权重矩阵，b是bias项，西格玛是sigmoid函数。所以最开始的方程可以写成如下形式。

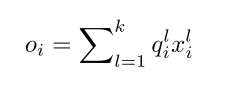


**多级注意机制：**

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：

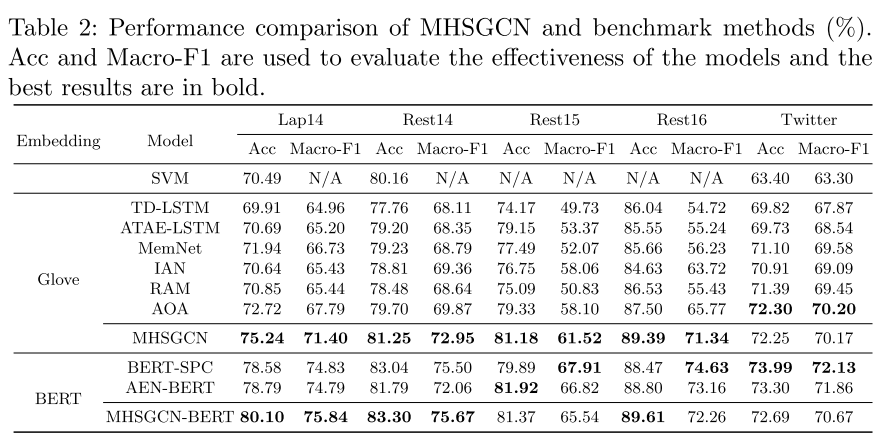


每一层的所有结点的线性组合如下所示：

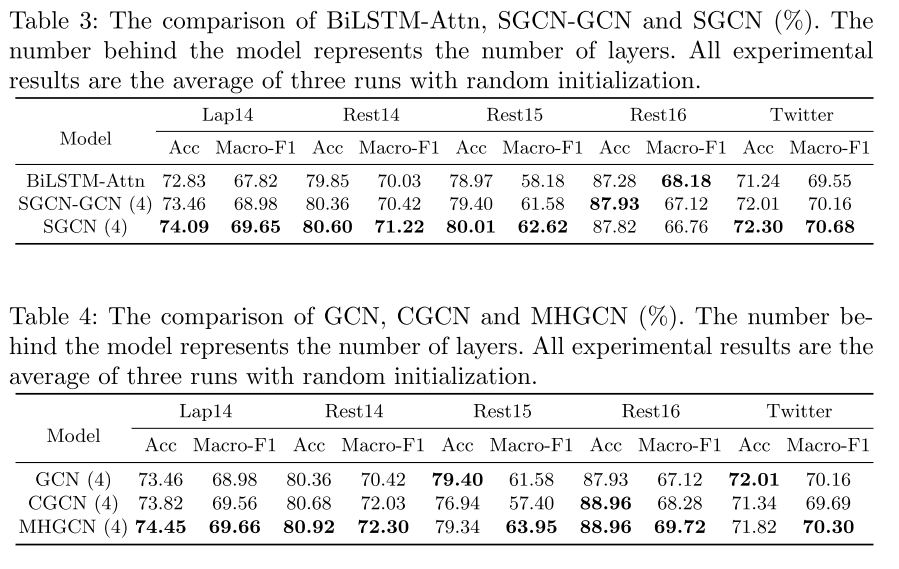


在捕获了多跳的信息之后，我们就能得到上下文的表示。

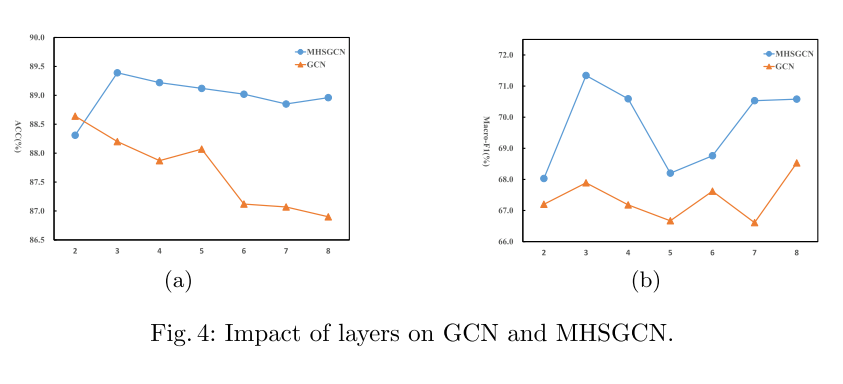
实验部分中，作者与其他的几种基准在一些数据集上进行了对比，还进行了消融实验分别证明了句法GCN，多跳注意机制的有效性，并分析了层数对网络学习能力的影响。



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

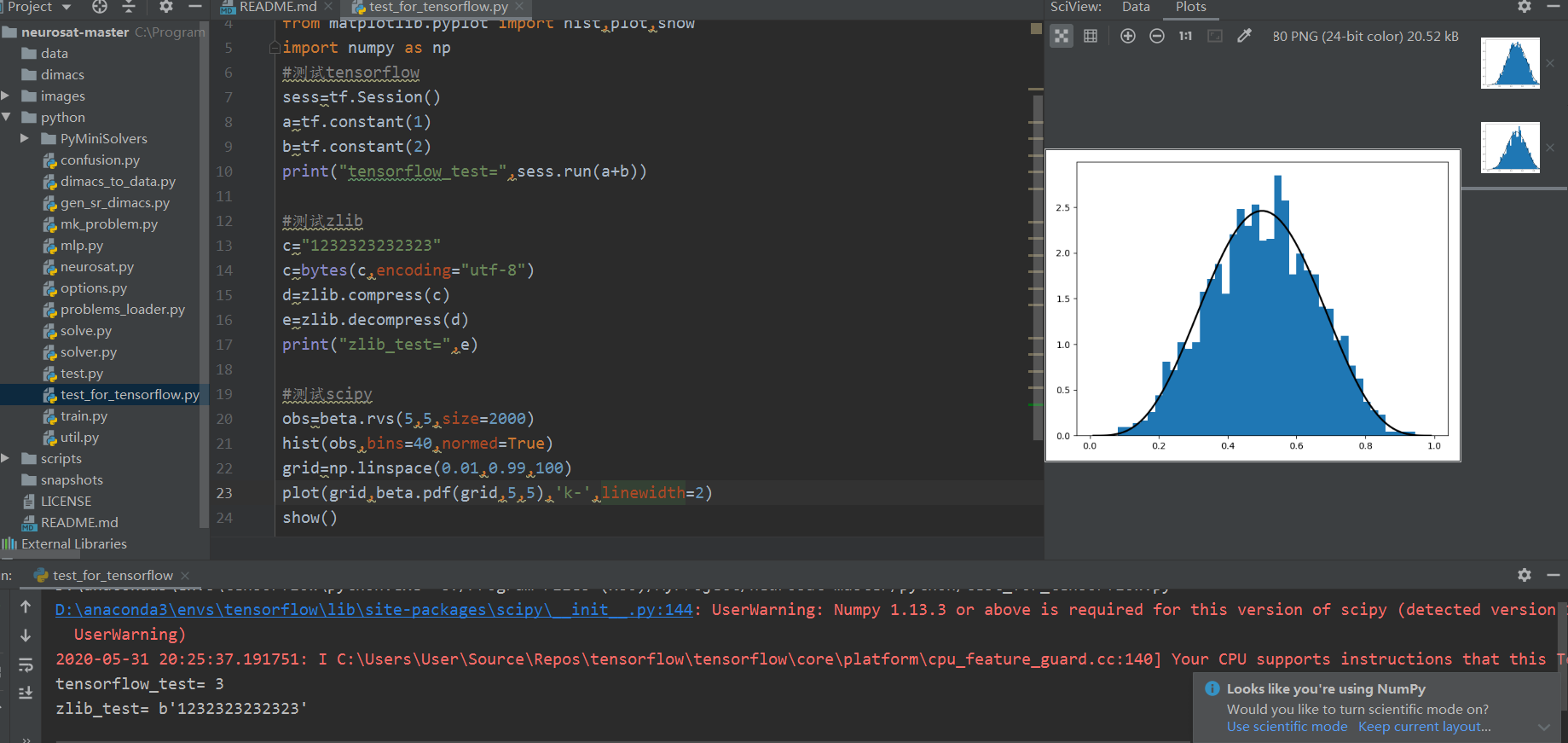
make

cd ../..

mkdir data dimacs snapshots

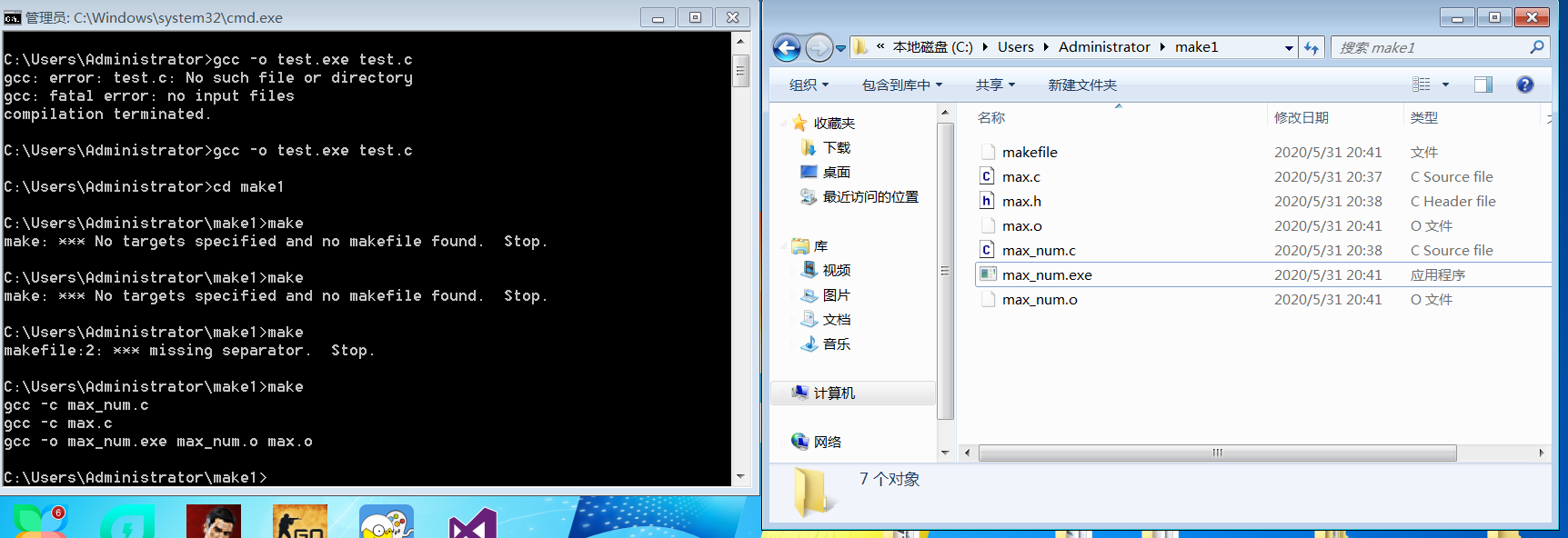
出现问题如下：

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）

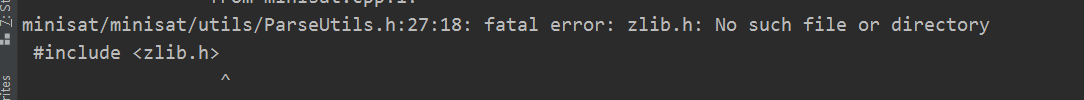


1. 去对应的文件夹clone PyMiniSolvers并make，但是make中出现了错误。

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。