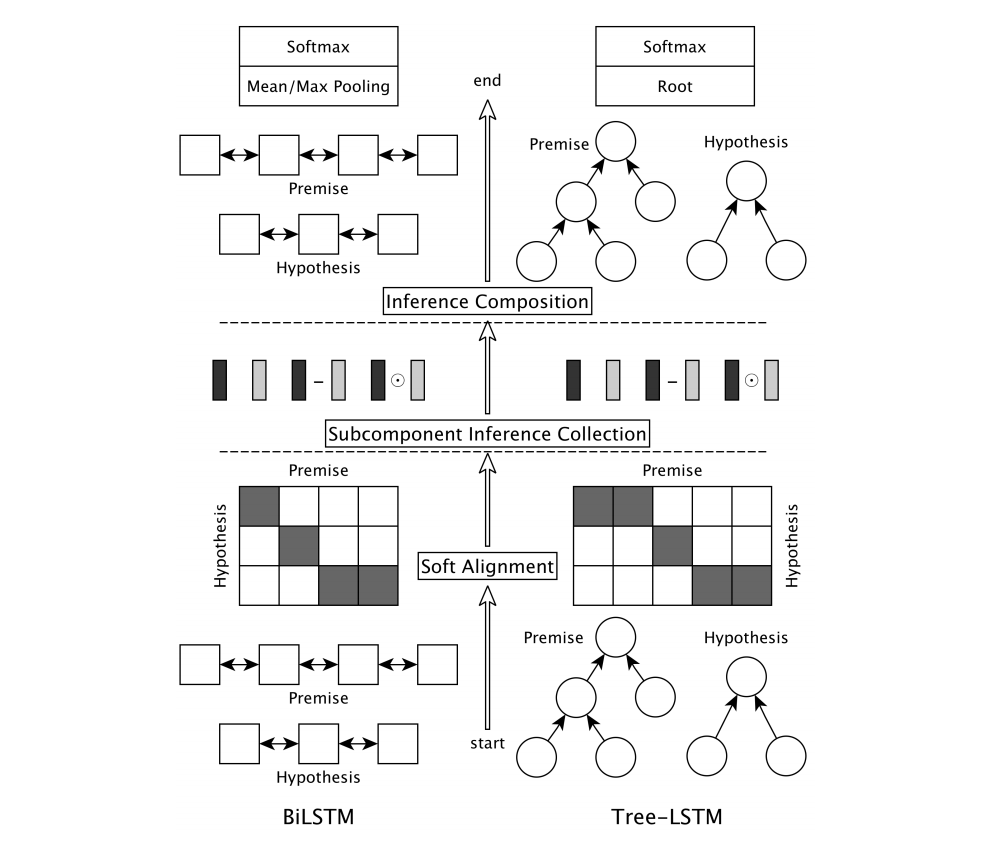
**Tree-Lstm论文调研**

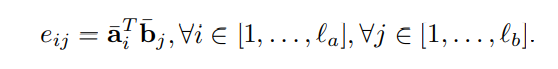
1. **Enhancing and Combining Sequential and Tree LSTM for Natural Language Inference**

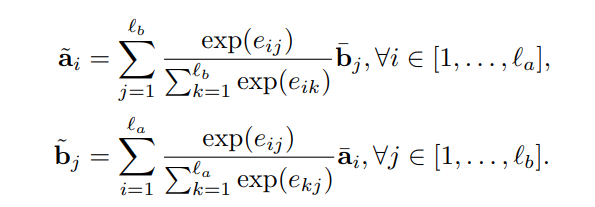
研究问题：NLI（自然语言推理），给定一个句子对，推理两个句子的关系

模型：

第一步：对每个句子过一遍Bilstm，得到每个句子的表示ai和bi（ai代表前提，bi代表推理后的结果）

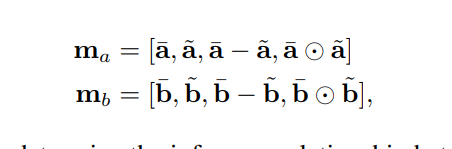
第二部（soft alignment）：计算attention值



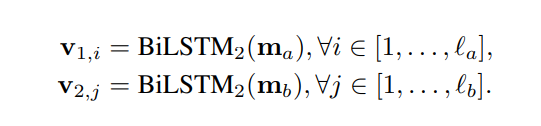


相当于每个句子a通过注意力的得分加权求和所有句子b的向量，最终得到a的表示，对b也一样

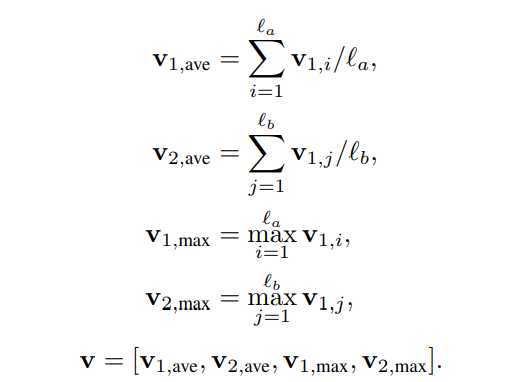
第三步（subcomponent inference collection）：作者自身设定的三个启发式规则



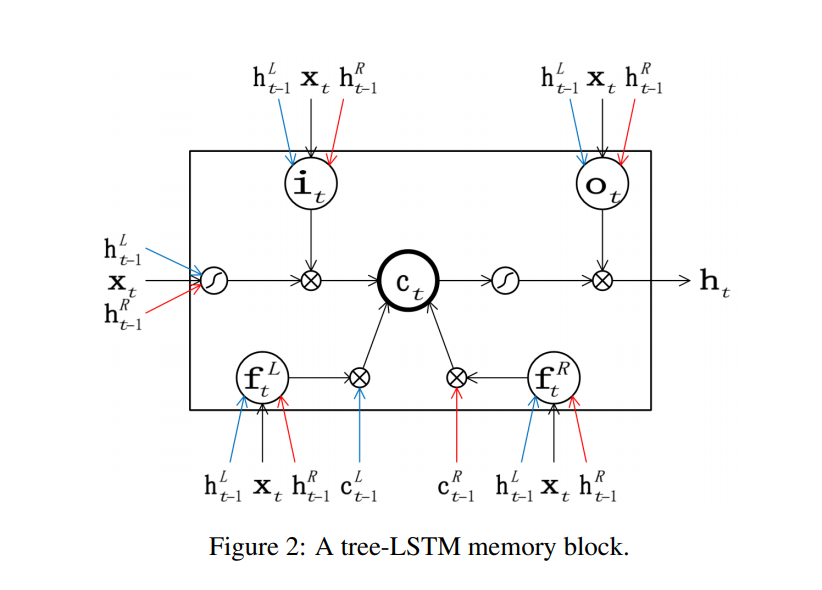
第四步：再对每个m过一个Bilstm，得到最终表示

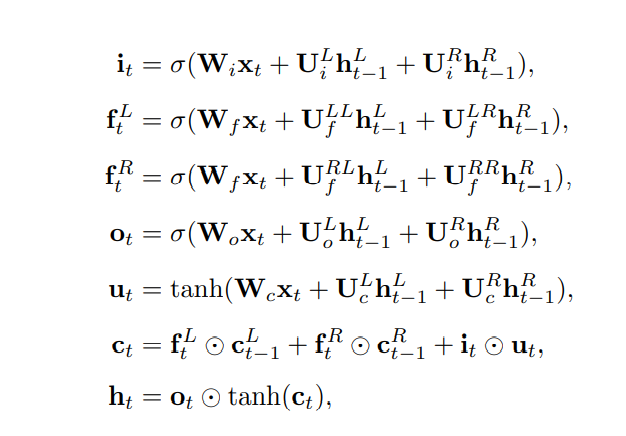


第五步：使用max pooling和average pooling将两个句子合在一起，放入分类器中



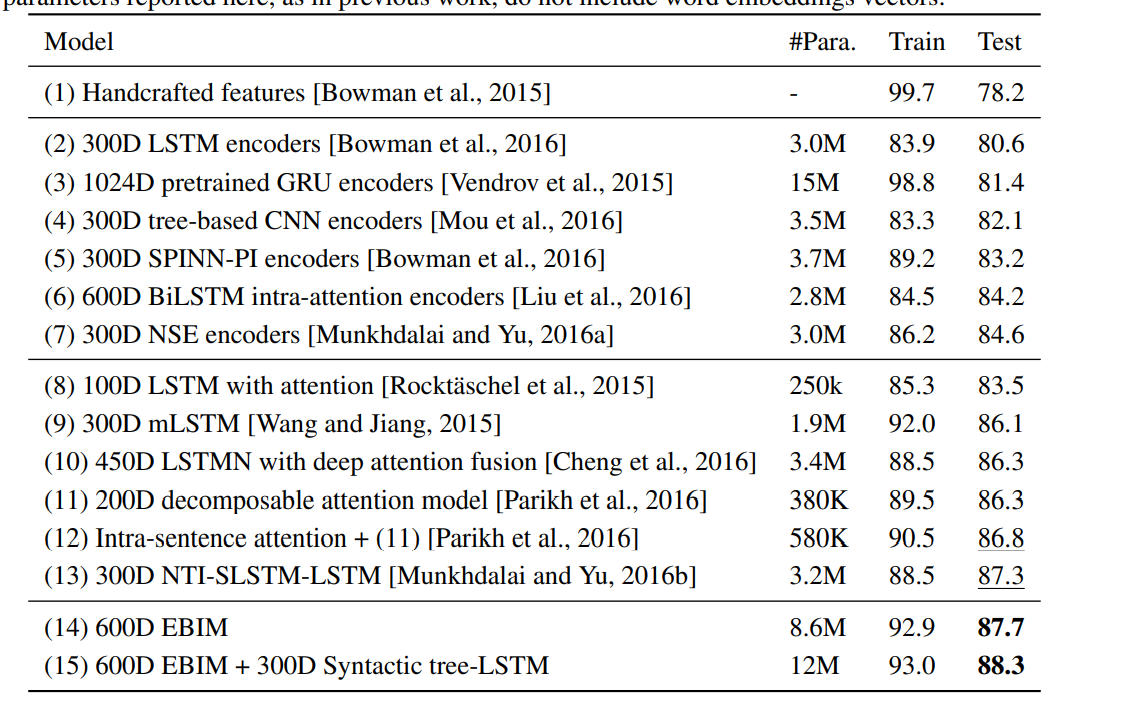
第六步：将上述的Bilstm换成二叉tree-lstm，用根节点的h表示每句话，其他不变





第七步：将tree-lstm得到的结果放入分类器，得到一个概率分布与刚才Bilstm得到的概率分布相加，得到最终的概率分布。

实验结果：



（14）（15）代表作者的模型

作者认为基本模型的潜力并没有被完全挖掘，还有上升空间

使用constituency parser使句子构建为一棵树

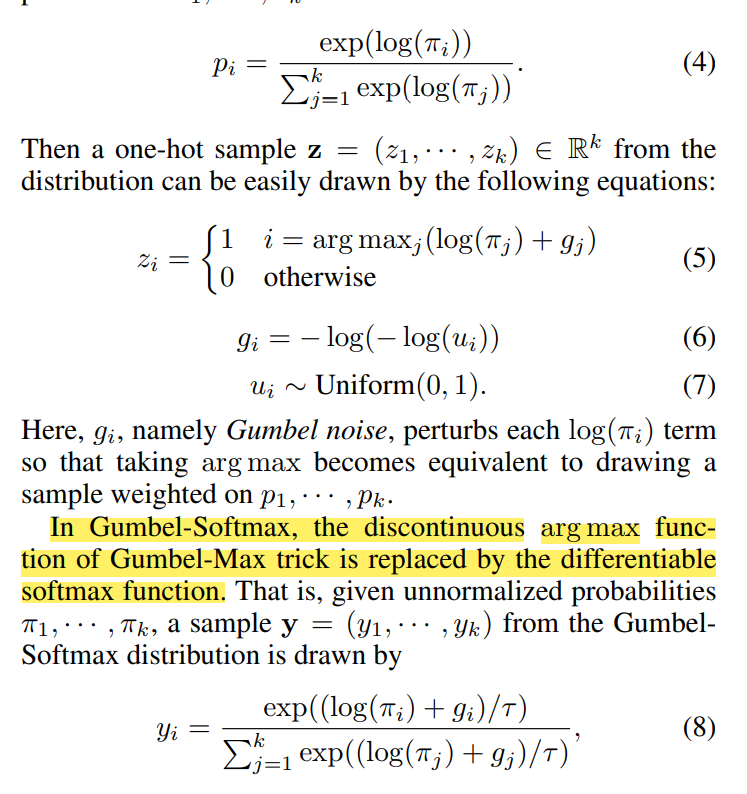
1. **Learning to Compose Task-Specific Tree Structures**

本文主要研究任务：NLI（自然语言推理），推理两个句子的关系 semantic analysis情感分析

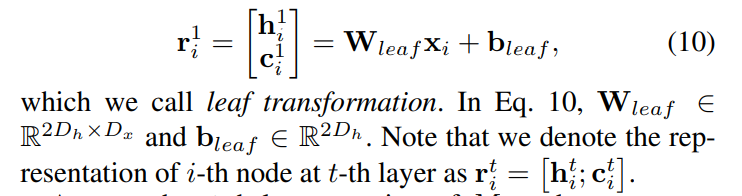
本文没有一个总的模型图，提出了一种将数据构建为树结构的方法，并使用Gumbel Tree-LSTM来编码句子信息（因为作者在论文里也提到符合特定任务需要特定的构建树的方法，代价高昂）

只介绍NLI中的做法：

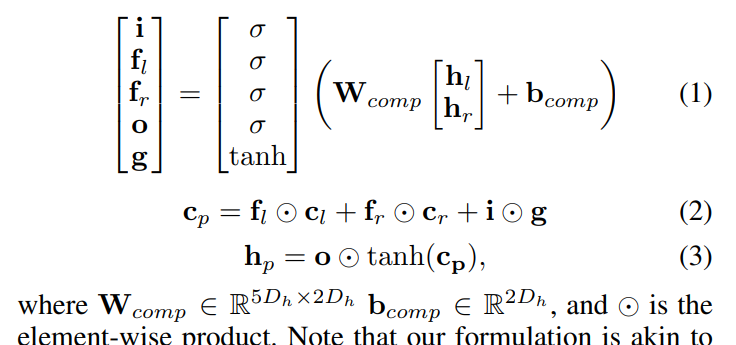
使用分类器为Gumbel-Softmax



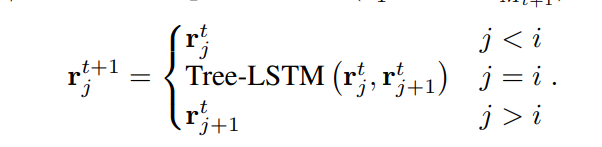
第一步：将每个句子中的词用一个词向量表示，使用一个线性变化获得初始的h和c



第二步：假设在第t层，当两个相邻的点被选为做融合成一个父节点，使用tree-lstm的公式



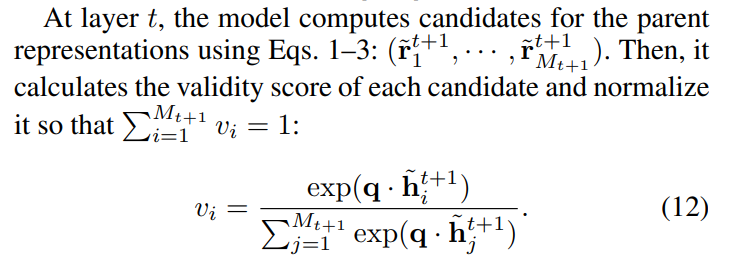
得到两个节点融合后的节点（t+1层）的表示，融合的规则如下：

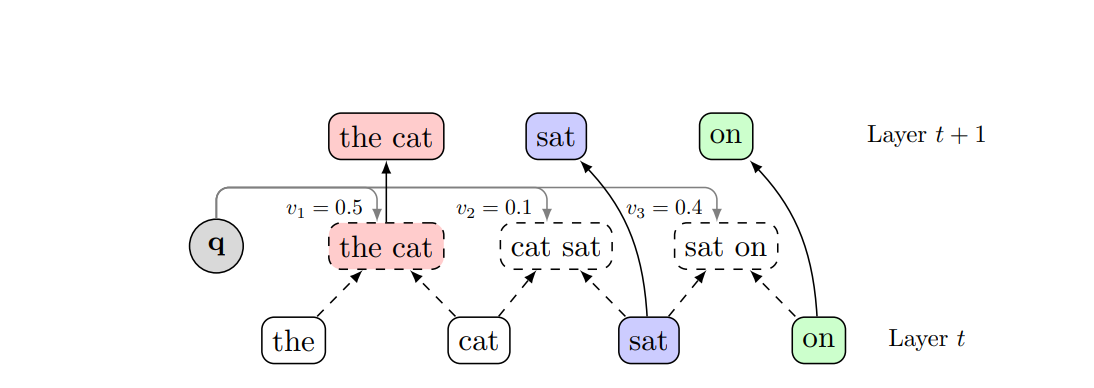


一直重复，两个点之间融合，到最后融合到只有一个点的表示

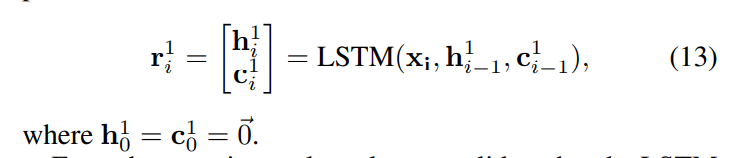
如何知道哪两个点要融合为一个点呢？作者使用一个得分机制来评判，并且选择的两个点必须是相邻点（例如：1，2，3，4，必须为1，2或者2，3或者3，4）

第三步：

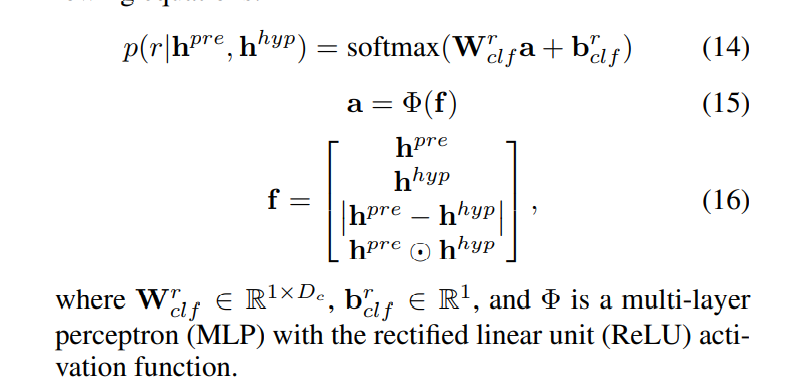




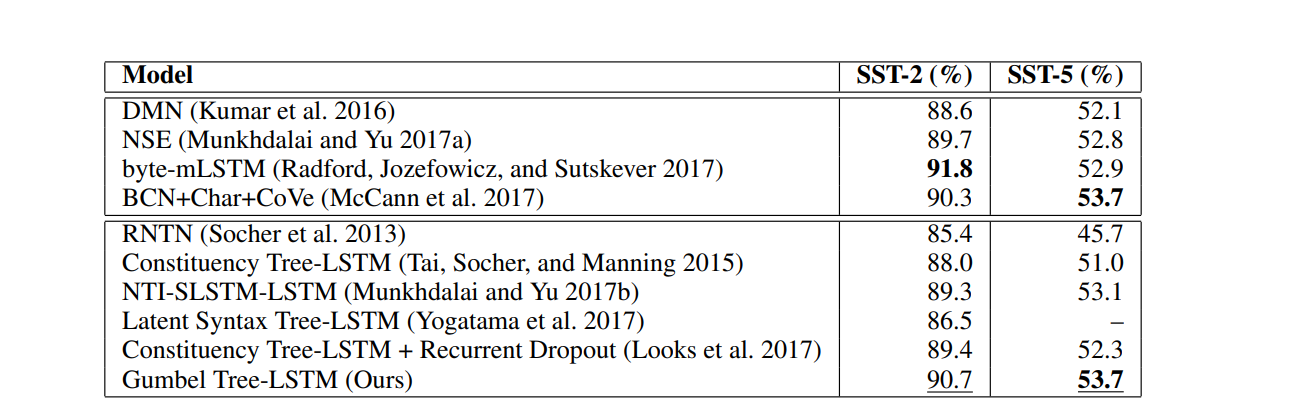
第四步：除了第一步线性变化外，还要对每层的叶节点过一个lstm，得到h和c



第五步：将两个句子丢入提出的架构中，得到句子a和b的表示，然后在使用下面公式计算最终概率



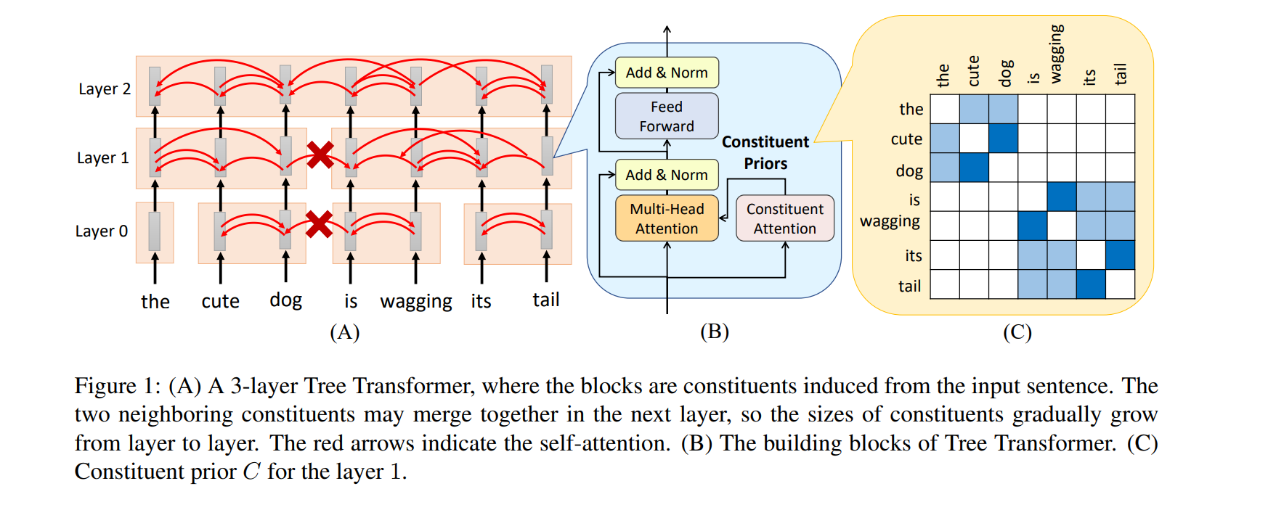
实验结果：



**3、Tree Transformer: Integrating Tree Structures into Self-Attention**

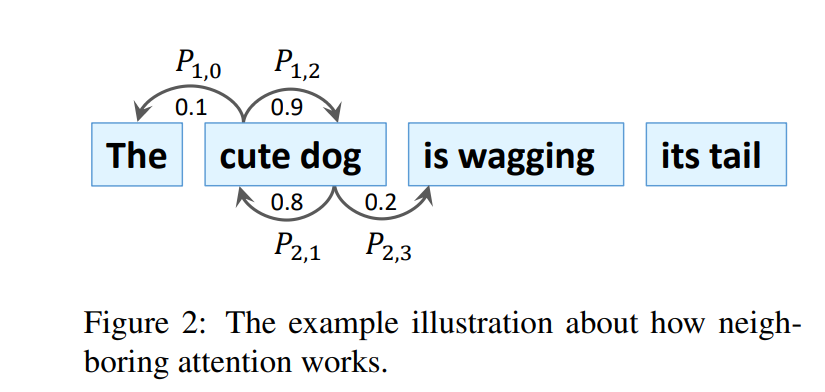
作者认为传统的transformer的attention机制并没有学到符合人们预期的结构性的信息，因此提出树状的transformer来更好的获得结构性信息

大体上结构与传统transformer结构并没有太大区别，只是提出了一种构造树的方法，加了一个constituent attention模块

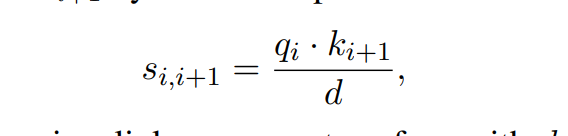


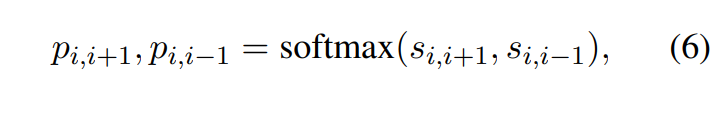
Constituent attention模块：

第一步：有一个强约束，只在词的前和后计算两个词该合并的概率（例如：1，2，3，4，只是计算1，2或者2，3或者3，4）

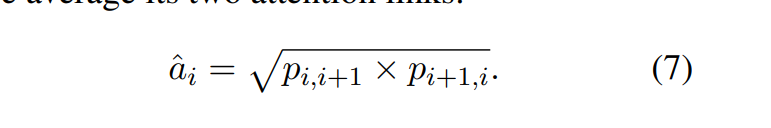


计算公式为：

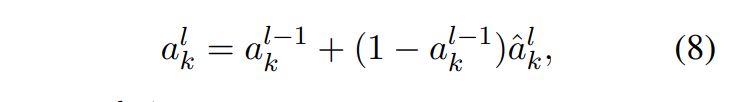




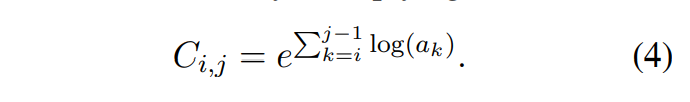
因为两个得分可能不一样，因此取一个平均值，例如2，3和3，2计算的得分不一样，但本质上是同一件事，两个是不是该合并



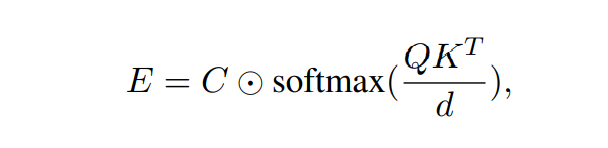
第二步：因为本文结构不止一层，因此上一步计算的a也不止一层，最终由下面公式计算得到一个a的序列集合（例如：a1，a2，a3………）



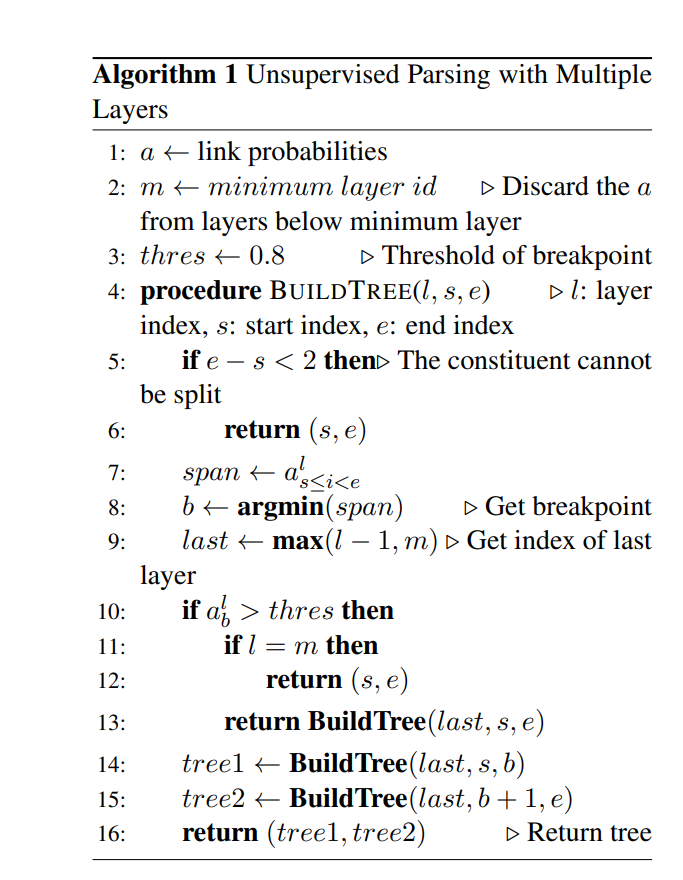
第三步：计算C这个矩阵，计算结果为第一幅图的（c），对称矩阵



第四步：计算注意力的时候有限制，因为必须满足连在一块的才能有注意力的值，不在一块的没有注意力值

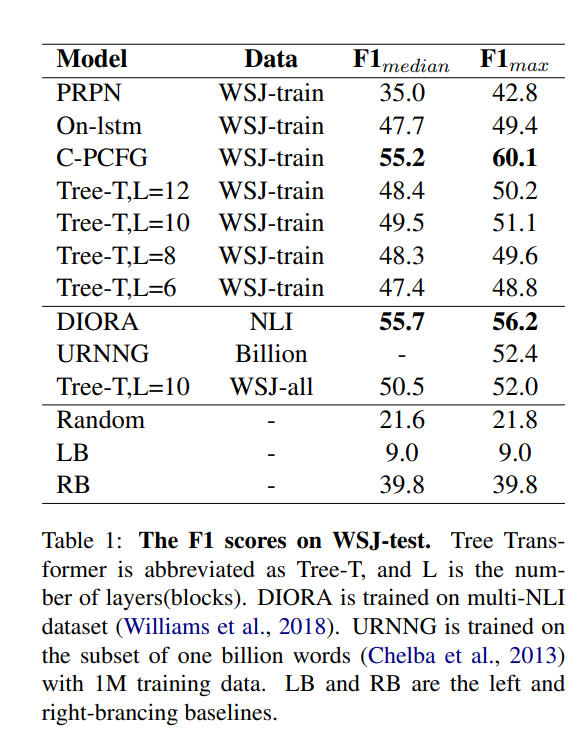


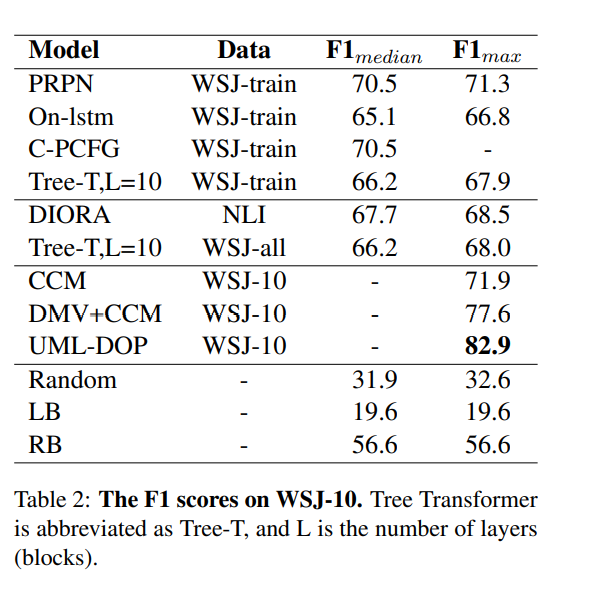
第五步：Unsupervised Parsing from Tree Transformer



从高层往底层计算，有个阈值，小于第几层后的值不再考虑

实验：Grammar Induction





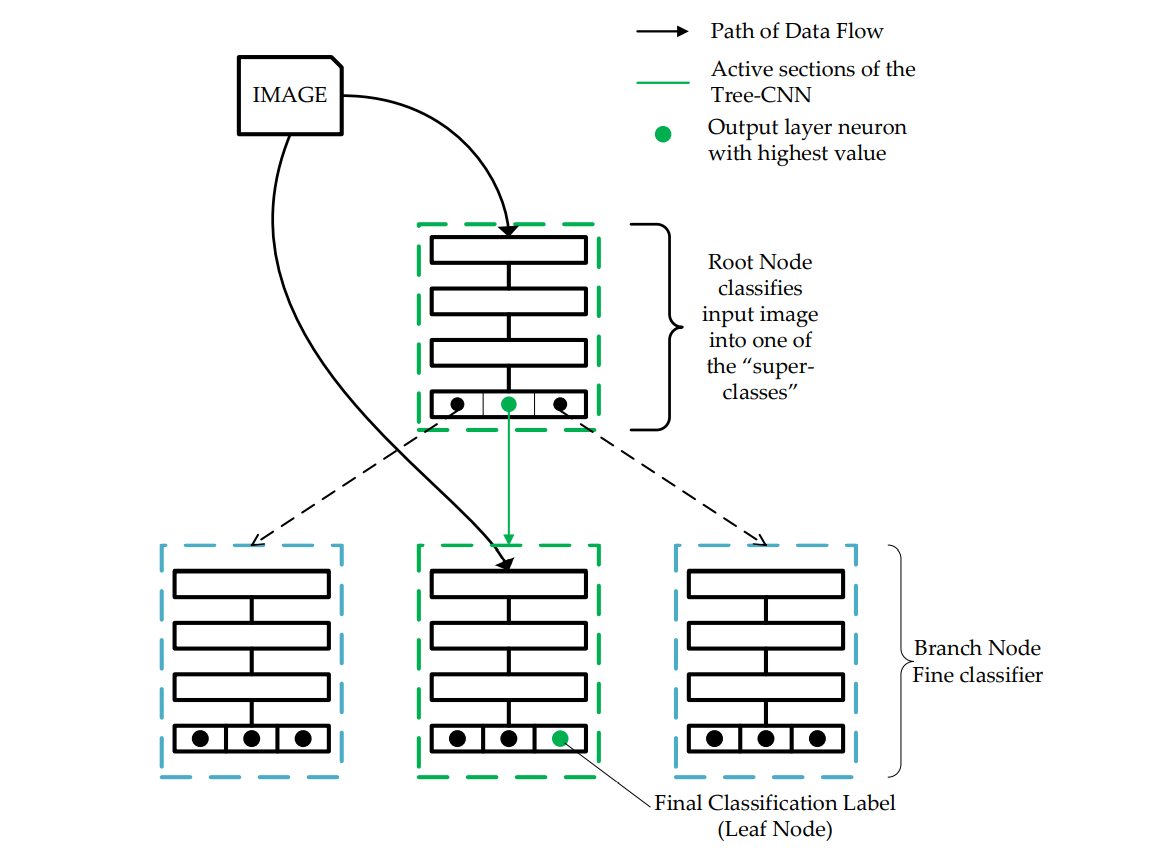
从图上可知 效果并不是最优

1. **Tree-CNN: A Hierarchical Deep Convolutional Neural Network for Incremental Learning**

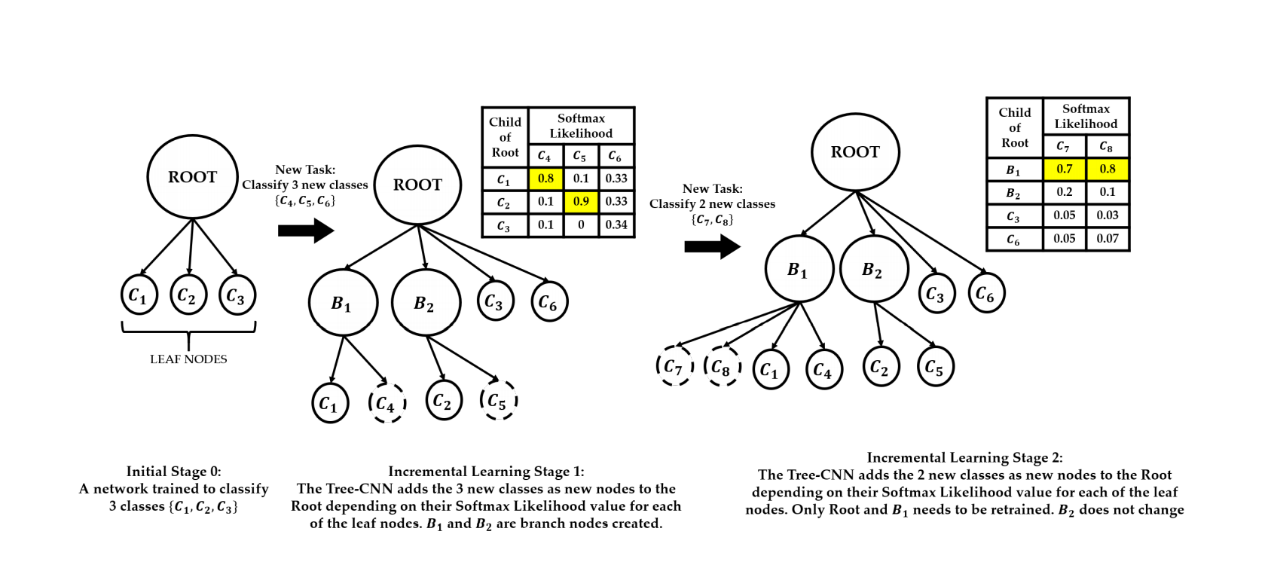
这篇论文总共30页，太长，因此我也没有太细看，而且是增量学习领域，本人也不是太了解

作者提出的问题就是，传统的DCNN（深层的CNN）虽然在特定数据集上表现的很好，但是现实生活中数据总是在不断增加，那已经训练好的CNN如果又给了一个新的数据集去训练，那就会造成catastrophic forgetting的问题，也就是会更新掉以前的参数（类似于忘记以前的参数），因此为了解决这个问题，作者提出了基于增量学习的方法来解决，构造一棵Tree-CNN的架构

上层节点代表超类，也就是一大类（比如是猫，是狗），下层节点开始进行细分（比如猫的性别，猫的颜色），叶节点就代表最终的分类，模型如下：



但是因为数据在不断增加，因此作者需要增加叶节点的数量，来增加新的分类



上图为增加方式，通过计算一个概率来确定哪个节点需要加深