Lecture Notes: Dependency Parsing

Keyphrases: Dependency Parsing

1. Dependency Grammar and Dependency Structure

与编译器中的解析树类似， **NLP** 中的解析树是用于分析句子的句法结构。使用的结构主要有两种类型——短语结构和依存结构。

短语结构文法使用短语结构语法将词组织成嵌套成分。以后章节将对此进行更详细的说明。我们现在关注依存语法。

句子的依存结构展示了单词依赖于另外一个单词（修饰或者是参数）。词与词之间的二元非对称关系称为依存关系，描述为从 **head**（被修饰的主题）用箭头指向 **dependent** （修饰语）。一般这些依存关系形成树结构。他们通常用语法关系的名称（主体，介词宾语，同位语等）。一个依存树的例子如下图所示，有时候也会在树的头部加一个假的ROOT节点，这样每个单词都依存于唯一的节点。

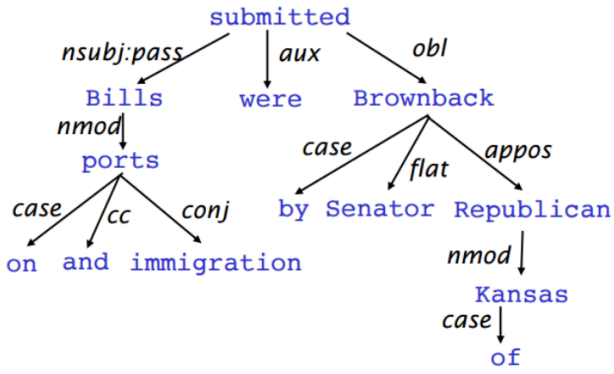


Figure 1: Dependency tree for the sentence "Bills on ports and immigration were submitted by Senator Brownback, Republican of Kansas"

1.1 Dependency Parsing

依存分析是对输入的句子S分析其语法依存结构的任务。依存解析器的输出是一棵依存树，其中输入句子的单词是通过依存关系的方式连接。正式的，依存解析问题要求生成一个从输入句子$S = w_0 w_1 \dots w_n$到依存树图G的映射。许多基于依存的变种方法已经在近几年被开发出来，包括基于神经网络的方法，这个后面我们会详细描述。

准确来说，依存解析有两个子问题：

1. 学习：给定一个用依赖语法图标注的句子训练集D，创建一个可以用于解析新句子的解析模型M。
2. 解析：给定一个解析模型M和一个句子S，根据M生成对于S最优的依存关系图D。

1.2 Transition-Based Dependency Parsing

基于转换的依存解析依赖于一个定义了所有可能转换的状态机器，从而能够生成从输入句子到依存树的映射。在这里，学习问题是生成一个模型，使得它能够基于历史转换信息预测状态机器中的下一个转换。解析问题是在被提供之前创建的模型的前提下，针对输入的句子构建最优的转换序列。大多数基于转换的系统不会使用正式的语法。

1.3 Greedy Deterministic Transition-Based Parsing

这个系统由Nivre在2003年提出，并与当时其他方法有极大的不同。

这个转换系统是一个由“状态”和“转换”组成的状态机器。模型提供了一系列从一些初始状态到多个终止状态中的一个的转换。

状态：

对于句子$S = w_0 w_1 \dots w_n$，一个状态可以被描述成一个三元组$c = (\sigma, \beta, A)$

1. 一个堆栈σ，存储从S中来的w_i
2. 一个缓冲区β，存储从S中来的w_i
3. 一组依存弧A，其形式为(w_i, r, w_j)，$w_i, w_j \in S$，r表示依存关系

因此，对于任意句子 $S = w_0 w_1 \dots w_n$

1. 一个形式为$([w_0]_\sigma, [w_1, \dots, w_n]_\beta, \emptyset)$ 的初始状态 c_0（现在只有 ROOT 在堆 σ 中，没有被选择的单词都在缓冲区 β 中）。
2. 一个形式为 $(\sigma, []_\beta, A)$ 的终点状态。

转换：

在两种状态间有三种转换：

1. Shift：将缓冲区中的第一个单词转移到栈顶。（前提条件：缓冲区不空）
2. Left-Arc：向A中增加一个依存弧(w_j, r, w_i)，其中w_i是次栈顶元素（第二个），w_j是栈顶元素。将w_i从栈中删除。（前提条件：栈中最少有两个元素并且w_i不为ROOT）
3. Right-Arc：向A中增加一个依存弧(w_i, r, w_j)，其中w_i是次栈顶元素（第二个），w_j是栈顶元素。将w_j从栈中删除。（前提条件：栈中最少有两个元素）

一个更加正式的定义在图二中展示：

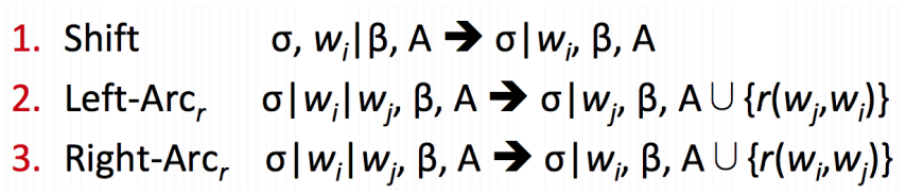


Figure 2: Transitions for Dependency Parsing.

1.4 Neural Dependency Parsing

虽然依赖项解析有很多深层模型，这部分特别侧重于贪心，基于转移的神经网络依存语法解析器。与传统的基于特征的判别依存语法解析器相比，神经网络依存语法解析器性能和效果更好。与以前模型的主要区别在于这类模型依赖稠密而不是稀疏的特征表示。

我们将要描述的模型采用上一部分中讲述的标准依存弧转换系统。最终，模型的目标是预测从一些初始状态 c 到一个终点状态的转换序列，对模型中的依存语法树进行编码的。由于模型是贪心的，它基于从当前的状态 $c = (\sigma, \beta, A)$提取特征，然后尝试一次正确地预测一次转移$T \in SHIFT, LeftArc_r, RightArc_r$。

特征选择

因为模型复杂度的要求不同，所以神经网络输入的定义是灵活的。一个给定句子S的特征一般包括下面几个子集：

1. S_{word}：句子中那些在栈σ顶和缓冲区β顶部的单词的词向量表示。
2. S_{tag}：句子S中某些词的词性（Part-of-Speech POS）标签。POS标签组成一个小的离散集：$P = \{NN, NNP, NNS, DT, JJ, ..\}$
3. S_{label}：句子S中某些词的弧标签。弧标签组成一个小的离散集，描述了依存关系：$L = \{amod, tmod, nsubj, csubj, dobj, \dots\}$

对于每一个特征类型，我们会有一个对应的嵌入矩阵，将特征的one-hot表示映射到一个d-维的稠密向量表示。完整的S_{word}嵌入矩阵是$E^w \in \mathbb{R}^{d \times N_w}$，其中$N_w$是词汇表的大小。对应的，POS和label的嵌入矩阵是$E^t \in \mathbb{R}^{d \times N_t}$ 和 $E^l \in \mathbb{R}^{d \times N_l}$，其中$N_t$和$N_l$分别是不同POS的数量和不同弧标签的数量。

最后，令每一个特征集中被选去出来的元素数量为$n_{word}, n_{tag}, n_{label}$。

特征选择例子

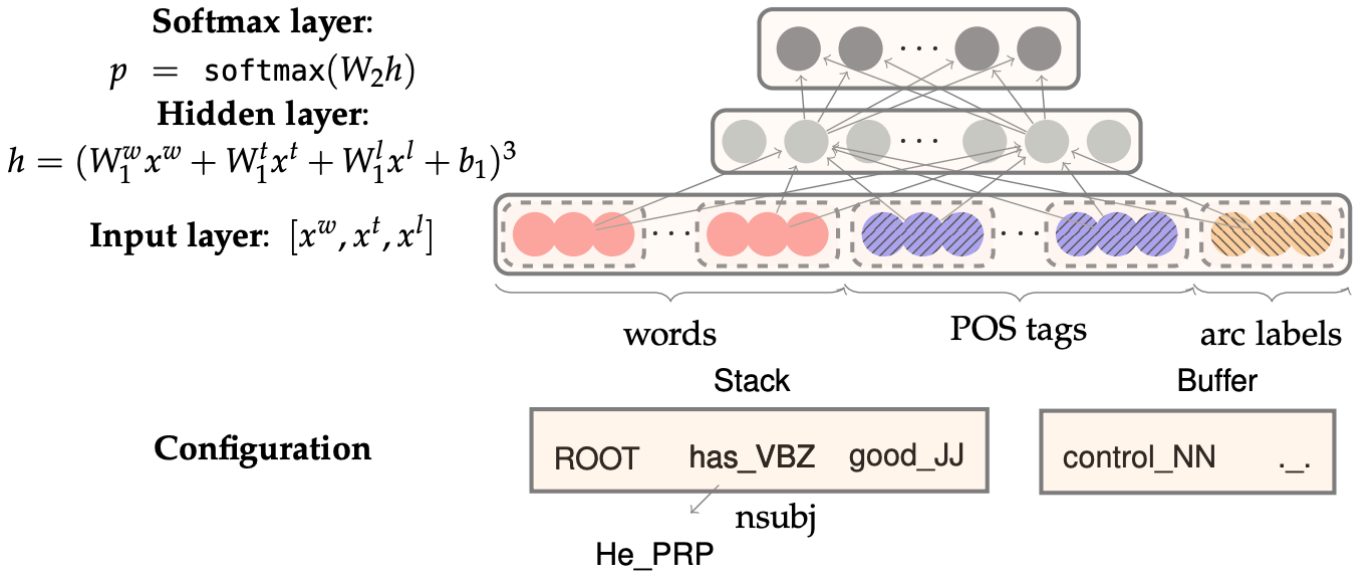
S_{word}， S_{tag}， S_{label}设置如下：

1. S_{word}：取栈和缓冲区的前三个单词：$s_1, s_2, s_3, b_1, b_2, b_3$。栈顶和次栈顶单词的第一个和第二个最左最右的孩子：$lc_1(s_i), rc_1(s_i), lc_2(s_i), rc_2(s_i), i = 1, 2$。栈顶和次栈顶单词的最左孩子的最左孩子，最右孩子的最右孩子：$lc_1(lc_1(s_i)), rc_1(rc_1(s_i)), i = 1, 2$。总共$S_{word}$包含了$n_w = 18$个元素。
2. S_{tag}：对应的POS标签构成了$S_{tag}(n_t = 18)$。
3. S_{label}：对应的弧标签，排除在栈和缓冲区中的6个词（$n_l = 12$）。

注意我们使用一个特殊的 NULL 表示不存在的元素：当堆和缓冲区为空或者还没有指定依存关系时。对一个给定句子例子，我们按照上述的方法选择单词，词性标注和依存标签，从嵌入矩阵 E^w, E^t, E^l 中提取它们对应的稠密的特征的表示，然后将这些向量连接起来作为输入$[x^w, x^t, x^l]$。在训练时间，我们反向传播到稠密的向量表示，以及后面各层的参数。

前向传播神经网络模型

网络包含一个输入层：$[x^w, x^t, x^l]$，一个隐藏层和一个最终的softmax输出层，cross-entropy作为损失函数。我们可以定义一个单独权重矩阵在隐藏层，从而能够在拼接后的输入层$[x^w, x^t, x^l]$直接操作，也可以使用三个权重矩阵$[w_1^w, w_1^t, w_1^l]$，如图三所示。然后我们使用非线性函数并使用一个额外的仿射层$[W_2]$，是的能够输出层的神经元数与真实输出相匹配。



现在在图三中，使用的非线性函数是$f(x) = x^3$。

有关 greedy transition-based 神经网络依存语法解析器的更完整的解释，请参考论文："**A Fast and Accurate Dependency Parser using Neural Network**"