对于依赖解析以及集合演算模型的认识

21307174 刘俊杰 21307155 冯浩

May 2024

1 报告介绍

自然语言处理（NLP）是人工智能和语言学领域的一个核心方向，它旨在使计算机能够理解、解释和生成人类语言的内容。自然语言解析，作为 NLP 的一个关键组成部分，关注的是如何将句子转换成计算机可以处理的结构化形式，如语法树或依赖图。尽管已有多种解析方法，但大多数依赖于统计或机器学习方法，这些方法往往缺乏生物学上的可信度。近年来，随着神经科学和认知科学的进展，研究者开始探索更符合大脑处理语言方式的计算模型。集合演算（AC）作为一种新兴的计算框架，为模拟大脑处理语言的机制提供了一种可能。

论文《A Biologically Plausible Parser》提出了一个新颖的英语句子解析器，其核心特点是由生物学上可信的神经元和突触实现的英语解析器，并通过集合演算（Assembly Calculus，AC）的用于认知功能的计算框架来实现。这个解析器能够模拟人类大脑处理语言的方式，正确地解析具有一定复杂性的句子。实验表明，该设备能够正确解析相当非平凡的句子。尽管实验涉及的英语句子相对简单，但结果表明，解析器可以扩展到我们实现之外的多个方向，包括语言的大部分内容。

本报告将围绕论文《A Biologically Plausible Parser》中的依赖解析和集合演算模型进行探索学习。

2 集合演算模型

2.1 集合演算概述

集合演算（Assembly Calculus，简称 AC）是由 Papadimitriou 等人于 2020 年提出的一种计算模型，旨在通过模拟大脑中神经元和突触的活动来执行认知功能。AC 的核心思想是“集合”（Assembly），它是由一组高度连接的兴奋性神经元组成的集合，这些神经元通过突触相互连接，并在同一大脑区域内共同工作。

AC 描述了一个动态系统，论文中提到其涉及到神经科学文献中广泛证实的以下部分和属性：

a）脑区域间存在随机连接的神经元；

b）神经元输入的简单线性模型；

c）每个区域内的抑制，使得前 k 个激活最强的神经元被激发；

d）一种简单的 Hebbian 可塑性模型，即随着神经元的激发，突触的强度增加。

2.2 集合演算模型框架

论文中模拟器和实验中使用的集合演算（AC）模型：

2.2.1 大脑区域与神经元

1. 有一个有限数量的大脑区域，每个区域包含 n 个兴奋性神经元，每个区域中的 n 个神经元通过一个随机加权有向 $G_{n,p}$ 图连接。

2. 每个突触 (i,j) 具有一个突触权重 $w_{ij} > 0$，初始值设为 1，这个权重会动态变化。

3. 对于某些无序的区域对 (A,B)，其中 $A \neq B$，存在一个随机有向二分图连接区域 A 中的神经元和区域 B 中的神经元，并且反向连接，每个可能的突触概率也为 p。这些区域之间的连接称为纤维（fiber）。

2.3 动态系统

动态系统是一个具有 a 个节点和随机有向加权边的大型动态加权有向图 $G = (N, E)$。

动态系统的更新是通过一系列精确定义的步骤完成的，每个步骤都基于前一个时间步的状态来计算下一个时间步的状态。

2.3.1 状态表示

在每个时间步 t，动态系统的状态由以下两部分组成：

1. 对于每个神经元 i，一个位 $f_i^t \in \{0,1\}$ 表示神经元 i 在时间 t 是否发射。

2. 所有突触在集合 E 中的突触权重 w_{ij}^t。

2.3.2 状态更新规则

状态从时间步 t 更新到时间步 $t+1$ 遵循以下步骤：

2.3.2.1 突触输入计算

对于每个神经元 i，计算其突触输入 S_i^t，这是时间 t 时所有前突触神经元 j 的权重 w_{ji}^t 的总和，其中 j 是发射的（即 $f_j^t = 1$）：

$$S_i^t = \sum_{(j,i)\in E, f_j^t=1} w_{ji}^t$$

2.3.2.2 神经元激活状态更新

对于每个神经元 i，确定其在时间 $t+1$ 的激活状态 f_i^{t+1}。如果神经元 i 在其区域中突触输入 S_i^t 最高的 k 个神经元之一，则 $f_i^{t+1} = 1$，表示它将在时间 $t+1$ 发射；否则 $f_i^{t+1} = 0$：

$$f_i^{t+1} = \begin{cases} 1 & \texttt{if } i \texttt{ is among the top } k \texttt{ neurons with the highest } S_i^t \\ 0 & \texttt{otherwise} \end{cases}$$

2.3.2.3 **突触权重调整**

对于每个突触 $(i,j) \in E$，根据以下规则更新突触权重 w_{ij}^{t+1}。如果突触后神经元 j 在时间 $t+1$ 发射且突触前神经元 i 在时间 t 发射，则突触权重增加一个因子 $1+\beta$：

$$w_{ij}^{t+1} = w_{ij}^{t}(1 + f_i^t f_j^{t+1}\beta)$$

2.3.3 **高级命令：抑制与解除抑制**

在动态系统中，除了神经元的自然激活和突触权重的调整外，还可以通过高级命令来控制特定神经元或纤维的活动。

2.3.3.1 **抑制操作**（inhibit）

抑制操作用于阻止特定神经元或纤维的活动。这在模拟大脑中某些区域的临时沉默或控制信息流时非常有用。

区域抑制通过命令 $\texttt{inhibit}(A,i)$ 实现，其中 A 是要抑制的大脑区域，i 是执行抑制操作的神经元群体的标识符。

> 抑制区域 A 后，该区域内的所有神经元在随后的时间步中都不会发射，无论它们的突触输入如何。
>
> 这种抑制可以模拟大脑中某些区域的临时性功能关闭。

纤维抑制通过命令 $\texttt{inhibit}((A,B),i)$ 实现，其中 (A,B) 是连接两个区域的纤维，i 是执行抑制操作的神经元群体的标识符。

> 抑制纤维 (A,B) 后，从区域 A 到区域 B 的所有突触连接被阻断，阻止信号传递。
>
> 这可以模拟大脑中不同区域间通信的临时中断。

2.3.3.2 **解除抑制操作**（disinhibit）

解除抑制操作用于取消之前施加的抑制，允许神经元或纤维恢复正常活动。

区域解除抑制通过命令 $\texttt{disinhibit}(A,i)$ 实现，其中 A 是之前被抑制的大脑区域，i 是对应的神经元群体标识符。

> 解除抑制后，区域 A 中的神经元可以再次根据它们的突触输入发射。

这种操作可以模拟大脑中某些区域的功能恢复。

纤维解除抑制通过命令 $\texttt{disinhibit}((A,B),i)$ 实现，其中 (A,B) 是之前被抑制的纤维。

解除抑制后，纤维 (A,B) 再次允许信号从区域 A 传递到区域 B。

这可以模拟大脑中不同区域间通信的恢复。

2.4 集合

集合是动态系统中的一个关键新兴属性，它由一组特殊的 k 个神经元组成，这些神经元位于同一大脑区域内，并且以密集的方式相互连接。

2.4.1 集合的特性

这些神经元之间拥有比随机连接更多的突触连接。

这些突触具有非常高的权重。

集合在大脑中用于表示对象、单词、想法等。

2.4.2 集合的动态行为

在时间 $t=0$，当没有其他神经元发射时，对区域 A 中的一个固定子集 k 个神经元 x 执行 $\texttt{fire}(x)$ 操作。如果存在一个相邻区域 B（通过纤维连接到 A），则集合 x 将在时间 0，1，2，，并影响区域 B 中逐渐演变的 k 个神经元的发射。

2.5 投影操作

投影操作允许集合在不同大脑区域之间传递其活动。

2.5.1 投影的基本概念

集合 x 在区域 A 中的活动可以投影到相邻区域 B。

这个过程导致区域 B 中的神经元集合 y 的形成，该集合是集合 x 的投影。

随着时间的推移，序列 $\{y_t\}$ 将收敛到一个稳定的集合 y。

2.5.2 投影的数学描述

在时间 $t = 1$，集合 y_1 包含区域 B 中接收到来自 x 的最大突触输入的 k 个神经元。在时间 $t = 2$，集合 y_2 包含接收到来自 x 和 y_1 的最高突触输入的神经元集合。

2.5.3 强投影操作

强投影是投影操作的增强，它涉及一系列投影操作。

2.5.3.1 强投影的定义

考虑动态系统中所有未被抑制的区域和纤维。

这些区域和纤维形成一个无向图，通常是一棵树。

所有活动区域中的集合同时发射，激活相邻区域中的神经元。

这个过程持续进行，直到整个系统达到稳定状态。

2.5.3.2 强投影的实现

project* 操作是一种系统范围的操作，它简化了一系列投影操作的复杂性。它允许活动区域的概念，这是对 AC 模型的一个小补充。

2.5.4 project 操作 (Algorithm 1) 伪代码详解

Algorithm 1 是一个迭代的过程，它通过模拟神经元的激活、突触输入的计算、以及突触权重的调整，来实现在多个大脑区域之间传递和处理信息的 ‘project* ‘ 操作。这个算法是实现生物学上可信的自然语言解析器的关键部分，它展示了如何将大脑处理信息的机制转化为计算过程。

2.5.5 初始化活跃集合

```
foreach area A do
    if there is active assembly x in A then
        foreach i   x do
            f1_i = 1
```

Algorithm 1: AC code for project*

foreach area A **do**
  if there is active assembly x in A **then**
    foreach $i \in x$ **do**
      $f_i^1 = 1$;

for $i = 1, \ldots, 20$ **do**
  all A, all $i \in A$, initialize $SI_i^t = 0$;
  foreach uninhibited areas A, B **do**
    if fiber (A, B) inhibited **then**
      skip ;
    $x = \{i \in A : f_i^t = 1\}$;
    foreach $j \in B$ **do**
      SI_j^t += $\sum_{i \in x, (i,j) \in E} w_{ij}$;
  foreach uninhibited area A **do**
    foreach $i \in A$ **do**
      if SI_i^t in top$-k_{j \in A}(SI_j^t)$ **then**
        f_i^{t+1} = 1 ;
      else
        f_i^{t+1} = 0 ;
  foreach uninhibited areas A,B **do**
    if fiber (A, B) inhibited **then**
      skip ;
    foreach $(i, j) \in (A \times B) \cap E$ **do**
      if $f_i^t = 1$ and $f_j^{t+1} = 1$ **then**
        w_{ij} = $w_{ij} \times (1 + \beta)$

– 对于模拟器中的每个大脑区域 A，进行以下操作。– 如果区域 A 中存在一个活跃的集合 x（即之前已经激活的神经元集合），则继续执行。– 对于集合 x 中的每个神经元 i，执行以下操作。– 设置该神经元在时间步 $t = 1$ 时的激活状态为 1，表示它将在下一个时间步被激活。

2.5.6 重置突触输入

```
for i = 1, ..., 20 do
    all A, all i   A, initialize SIt_i = 0
```

– 进行 20 次迭代，这里的 20 是一个设定的迭代次数，用于确保系统有足够的时间达到稳定状态。– 对于所有区域 A 中的所有神经元 i，初始化其在时间步 t 的突触输入 SIt_i 为 0。

2.5.7 计算新的突触输入

```
foreach uninhibited areas A, B do
    if fiber (A, B) inhibited then skip
    x = {i   A : ft_i = 1}
    foreach j   B do SIt_j +=   i x,(i,j) E wij
```

– 对于所有未被抑制的区域对 A 和 B，执行以下操作。– 如果连接区域 A 和 B 的纤维被抑制，则跳过当前的迭代。– 创建一个集合 x，包含区域 A 中所有在时间步 t 激活的神经元 i。– 对于区域 B 中的每个神经元 j，计算所有来自集合 x 的神经元 i 通过纤维 (A, B) 发送过来的突触输入总和，并将其累加到 SIt_j。

2.5.8 更新神经元激活状态

```
foreach uninhibited area A do
    foreach i   A do
        if SIt_i in top-kj A(SIt_j) then
            ft+1_i = 1
        else
            ft+1_i = 0
```

– 对于所有未被抑制的区域 A，执行以下操作。– 对于区域 A 中的每个神经元 i，执行以下操作。– 如果神经元 i 的突触输入 SIt_i 在其所在区域 A 中的 k 个最高突触输入之列，则执行以下操作。– 设置神经元 i 在下一个时间步 $t+1$ 的激活状态为 1。– 否则，设置神经元 i 在下一个时间步 $t+1$ 的激活状态为 0。

2.5.9 调整突触权重

```
foreach uninhibited areas A,B do
    if fiber (A, B) inhibited then skip
    foreach (i, j)  (A × B)  E do
        if ft_i = 1 and ft+1_j = 1 then
            wij = wij × (1 +  )
```

– 对于所有未被抑制的区域对 A 和 B，执行以下操作。– 如果连接区域 A 和 B 的纤维被抑制，则跳过当前的迭代。– 对于所有在区域 A 和 B 之间存在的突触连接 (A,B) 中的神经元对 (i,j)，执行以下操作。– 如果神经元 i 在时间步 t 被激活，并且神经元 j 在时间步 $t+1$ 也将被激活，则执行以下操作。– 增加神经元 i 和 j 之间突触连接的权重，增加的量是原来的 $(1+\beta)$ 倍。

3 依赖解析

3.1 解析器的数据结构

解析器的数据结构是无向图 $G=(A,F)$。A 可以比作是一组大脑区域，F 是一组纤维，是无序区域对。一个重要的领域是 LEX，用于词典，包含单词表示。LEX 在大脑中被认为位于左侧 MTL 中，通过纤维与所有其他区域连接。A 的其余区域可能对应于左侧 STG 中 Wernicke 区域的子区域，单词从左侧 MTL 投射到该子区域以进行句法角色分配。在我们的实验中，这些区域包括 VERB，SUBJ，OBJ，DET，ADJ，ADV，PREP，PREPP 等。除了 LEX，这些区域的几个还通过光纤相互连接。这些区域中的每一个都被假设了，因为它似乎需要解析器在某些类型的简单句子上正确工作。事实证明，他们粗略地、明确无误地贴上了依赖标签。A 能被扩展到其他的一些领域，例如 CONJ 和 CLAUSE。

除了词汇区（LEX）外，所有这些区域都是 AC（大脑皮层的一部分）的标准区域，包含 n 个随机连接的神经元，其中最多 k 个在任何时候都会放电。相比之下，词汇区包含每个词汇在语言中的固定集合 x_w。这些集合是特殊的，因为它们的放电不仅会对系统产生普通的影响，还会执行一个特定于词汇 w 的短程序 α_w，称为词汇 w 的动作。直观地看，词汇区所有词汇集合的动作总和构成了设备的语法。

一个词汇 w 的动作 α_w 由两组抑制和去抑制命令组成，针对特定的区域或纤维。第一组在系统的投射操作之前执行，第二组在之后执行。

3.2 依赖解析

依赖解析是自然语言处理中的重要技术，用于理解句子中的语法结构及词汇之间的关系。以下内容基于所提供的依赖解析模型及其伪代码，探讨了依赖解析的核心概念、工作机制以及个人对其有效性和实现方法的理解。

3.2.1 核心概念

依赖解析通过解析句子中的每个词，确定词汇之间的依赖关系，从而构建出一个依赖树。每个词作为一个节点，依赖关系则是节点之间的有向边。这种解析方法对于理解句子的语法结构、语义分析及信息提取有着重要意义。

3.2.2 模型介绍

在提供的依赖解析模型中，词汇区（LEX）和其他大脑区域（如主语区、动词区）是解析过程的核心。每个词汇在词汇区有一个固定的神经元集合，称为 x_w。这些集合的放电会执行特定的短程序，称为词汇的动作（α_w），其通过去抑制和抑制命令影响其他区域的神经元活动。解析过程中，系统依次处理每个词汇，通过激活其对应的集合并执行其预指令和后指令，构建词汇之间的依赖关系。

3.2.3 解析过程

解析过程分为以下几个步骤：

1. 初始化：激活词汇区（LEX）、主语区（SUBJ）和动词区（VERB）。

2．逐词处理：遍历句子中的每个词汇，激活其在词汇区的集合，并执行其预指令。

3．投射操作：在预指令执行完毕后，进行投射操作，将当前激活的集合投射到其他区域。

4．后指令执行：投射操作后，执行该词汇的后指令，进一步调整神经元活动。

5．依赖关系提取：在解析完成后，通过读取解析树，提取词汇之间的依赖关系。

3.2.4 具体实现

论文中解析模型通过两个主要算法实现：

Parser 主循环 (Algorithm 2)：逐词激活集合并执行指令，构建依赖关系。解析器按顺序处理句子中的每个词。对于每个词，我们在词汇区 (LEX) 激活其集合，并应用其词汇项的预指令。然后，我们进行投射，在去抑制的纤维之间进行投射。之后，应用所有的后指令。输入：一个句子 s 输出：s 的依赖关系解析表示，根节点为动词 (VERB) 每个词的处理包括激活其在词汇区的集合，应用预指令，进行投射，并应用后指令。这个过程中的区域和纤维通过（去）抑制操作来控制信息传递。对于句子 s 中的每个词 w：

1．在词汇区 (LEX) 激活其集合 x_w。

2．遍历预规则 $\alpha_w \rightarrow$ Pre-Commands 中的每个（去）抑制指令，应用这些预指令。

3．进行投射 (project*)，在去抑制的区域之间进行连接。

4．遍历后规则 $\alpha_w \rightarrow$ Post-Commands 中的每个（去）抑制指令，应用这些后指令。

解析树读取 (Algorithm 3)：在解析完成后，读取解析树，生成依赖关系。

1．初始化：初始化栈 s 为 $\{(v, Verb)\}$，其中 v 是动词区 (VERB) 的根集合。

Algorithm 2: Parser, main loop

input : a sentence s
output: representation of dependency parse of s, rooted in VERB

disinhibit(LEX, 0) ;
disinhibit(SUBJ, 0) ;
disinhibit(VERB, 0) ;
foreach word w in s **do**
    activate assembly x_w in LEX ;
    foreach pre-rule (Dis)inhibit($\square, i$) in $\alpha_w \rightarrow$ Pre-Commands **do**
        (Dis)inhibit($\square, i$) ;
*project** ;
foreach post-rule (Dis)inhibit($\square, i$) in $\alpha_w \rightarrow$ Post-Commands **do**
    (Dis)inhibit($\square, i$)

2. 初始化依存关系 D 为一个空集合。

3. 处理栈中的元素：当栈 s 不为空时，循环执行以下操作：

   (a) 从栈 s 中弹出一个元素 (x, A)。

   (b) 将集合 x 投射到词汇区 (LEX)。

   (c) 使用 getWord() 获取在词汇区中活跃的集合 x 对应的词 w_A。

   (d) 尝试投射到其他区域：对于每个与 A 不同的区域 B，且 $(A, B) \in F$ (即 A 和 B 之间存在连接)：尝试将 x 投射到区域 B 中。

      i. 如果 try-project(x, B) 成功返回非 None 的集合 y：

         A. 将 y 投射到词汇区 (LEX)。

         B. 使用 getWord() 获取在词汇区中活跃的集合 y 对应的词 w_B。

         C. 将依存关系 $w_A \rightarrow w_B$ 添加到依存关系集合 D 中。

         D. 将 (y, B) 插入栈 s。

4. 返回依存关系：返回依存关系集合 D。

Algorithm 3: Read out of parse tree in G

input : G after parsing, with active root assembly v in VERB

output: the parse tree stored implicitly in G

initialize stack s as $\{(v, \mathrm{V}erb)\}$;
initialize dependencies $\mathcal{D} = \{\}$;
while s not empty **do**
    (x,A) = s.pop() ;
    project(x, LEX) ;
    w_A = *getWord*() ;
    foreach area $B \neq A$ s.t. $(A, B) \in \mathcal{F}$ **do**
        y = *try-project*(x, B) ;
        if y *not None* **then**
            project(y, LEX) ;
            w_B = *getWord*() ;
            add dependency $w_A \xrightarrow{B} w_B$ to $\mathcal{D}$;
            s.insert((y, B)) ;
return $\mathcal{D}$;

该算法的主要目的是证明解析器的工作原理，它展示了解析器图 G（即其突触权重）的状态和依存关系列表之间的映射关系。

3.2.5 个人理解

通过对该依赖解析模型的研究，我们其工作机制及实现方法有了更深入的理解。具体来说：

1. 生物启发的模型：该模型模拟了大脑的语言处理机制，通过神经元集合的激活和投射，构建词汇之间的依赖关系。这种生物启发的模型具有很强的直观性和合理性。

2. 灵活的解析过程：解析过程分为预指令、投射操作和后指令三个阶段，使得解析过程具有高度的灵活性和可扩展性。

3. 依赖关系的提取：通过读取解析树，提取词汇之间的依赖关系，这种方法直观且高效，有助于进一步的语义分析和信息提取。

3.2.6 总结

依赖解析是自然语言处理中的关键技术，通过模拟大脑的语言处理机制，构建词汇之间的依赖关系。基于提供的依赖解析模型，我们探讨了其核心概念、工作机制及个人理解。这种生物启发的依赖解析方法具有很强的直观性和合理性，对于理解句子的语法结构及实现高级自然语言处理任务具有重要参考价值。

4 心得体会

通过深入研究论文中的依赖解析模型及其背后的集合演算（Assembly Calculus，AC），我们对自然语言处理的复杂性和挑战性有了更深刻的认识。依赖解析不仅仅是理解句子语法结构的技术，更是深入探索语言语义和认知过程的桥梁。论文中提出的模型以一种生物启发的方式，模拟了大脑如何处理语言，通过激活神经元集合和信息投射来构建词汇间的依赖关系，这不仅直观合理，而且具有很强的可扩展性。我特别被这种模型的直观性和合理性所吸引，因为它基于我们对大脑如何工作的理解，并且能够灵活地适应不同的语言结构。

此外，集合演算框架为自然语言解析提供了一种新颖的视角，通过模拟大脑中神经元和突触的活动，AC 不仅理论上可行，而且实验证明其有效性。这种模型的提出，让我们意识到了模仿大脑工作方式来开发计算模型的巨大潜力，尤其是在处理语言的复杂性方面。论文还讨论了模型的扩展性，包括递归、多义性和歧义性处理，这显示了模型的灵活性和强大的适应能力。

读完论文后，我们深受启发，认识到了自然语言处理领域中理论和实践相结合的重要性。我期待将所学知识运用于实践中，通过编写代码来实现依赖解析的基本版本，并进一步探索 AC 在自然语言处理领域的应用。我相信，随着未来研究的深入，集合演算将为我们理解人类语言处理机制提供更多的线索，并推动自然语言理解领域的发展。