读书报告

——对于依赖解析以及集合演算模型的认识

第 20 组——林宇浩、熊蔚然、王哲

目录

一、对依赖解析的认识

依赖解析（Dependency Parsing，也可翻译为依赖解析）是自然语言处理（NLP）中的一个重要领域，用于分析句子的句法结构。与传统的短语结构解析不同，依赖解析通过建立单词之间的二元语法关系来描述句子的句法结构。这种方法在处理形态丰富且词序相对自由的语言时具有显著优势。

依赖解析的基本概念

依赖语法是一种描述句子结构的形式体系，也是依赖解析的基础。依赖语法不直接涉及短语成分和短语结构规则，而是通过描述句子中单词之间的依赖关系来定义句法结构。依赖结构由一组有向二元语法关系组成，每个关系连接一个主词和一个从属词。每个句子都有一个根节点，通常是句子的谓语动词，所有其他单词都直接或间接地依赖于这个根节点。

依赖关系

依赖解析中使用的依赖关系可以分为两大类：从句关系（Clausal Argument Relations）和修饰关系（Nominal Modifier Relations）。从句关系描述句子中谓词（通常是动词）与其参数之间的语法角色，而修饰关系则说明了单词修饰其主词的方式。常见的从句关系包括主语（NSUBJ）、直接宾语（DOBJ）和间接宾语（IOBJ），修饰关系则包括名词修饰语（NMOD）、形容词修饰语（AMOD）和限定词（DET）等。

Universal Dependencies 项目

Universal Dependencies（UD）项目提供了一套跨语言适用的依赖关系标准，这些关系在语言学上有实际意义，在计算上可用，并且在跨语言上具有一致性。UD 项目的依赖关系清单有助于实现依赖解析的标准化，使其在多语言处理任务中更具通用性和可比性。这些标准化的依赖关系可以表示常见语言中的所有可能出现的语法关系，为依赖解析的标准化提供了理论基础。图 15.3 展示了这些依赖关系在英语句子中的实际应用。

Relation	Examples with *head* and **dependent**
NSUBJ	**United** *canceled* the flight.
DOBJ	United *diverted* the **flight** to Reno.
	We *booked* her the first **flight** to Miami.
IOBJ	We *booked* **her** the flight to Miami.
NMOD	We took the **morning** *flight*.
AMOD	Book the **cheapest** *flight*.
NUMMOD	Before the storm JetBlue canceled **1000** *flights*.
APPOS	*United*, a **unit** of UAL, matched the fares.
DET	**The** *flight* was canceled.
	Which *flight* was delayed?
CONJ	We *flew* to Denver and **drove** to Steamboat.
CC	We flew to Denver **and** *drove* to Steamboat.
CASE	Book the flight **through** *Houston*.

Figure 15.3 Examples of core Universal Dependency relations.

依赖解析的形式化定义

依赖解析的形式化定义主要包括有向图和生成树的概念。依赖结构通常被表示为有向图，其中顶点表示单词或词根，边表示语法关系。这种表示方法直观地展示了句子中各个单词之间的语法关系，有助于理解句子的语法结构。为了简化解析过程，在满足一定条件的依赖有向图中，可将图转换为无环、有根的依赖树。具体来说，依赖树必须满足以下条件：

1.只有一个根节点，且根节点无入边。

2.除根节点外，每个顶点只有一个入边。

3.从根节点到每个顶点都有唯一的路径。

这种形式化定义确保了每个单词都有一个唯一的主词，依赖结构是连通的，并且每个单词都能通过唯一的路径连接到根节点。

投射性

投射性（Projectivity）是依赖解析中的一个重要概念。如果在句子中主词和从属词之间的每个单词都有一条路径，这个依赖结构就是投射的。投射结构没有交叉边，能很好地表示语序严格的语言。然而，许多自然语言中的句子结构是非投射的，特别是在词序灵活的语言中。非投射结构允许存在交叉边，这使得依赖解析在这种情况下能够更灵活地处理这些语言。

依赖解析的优势

依赖解析的一个主要优势是它在处理形态丰富且词序相对自由的语言时的优越性。课件中举了捷克语的例子，在捷克语中，语法宾语可以出现在位置副词之前或之后，而传统的短语结构语法需要为这种情况制定多个规则。依赖解析通过抽象出词序信息，只表示解析所必需的语法关系，简化了这种处理。此外，依赖解析能够直接表示动词及其参数之间的重要信息，这些信息在更复杂的短语结构解析中通常被隐藏。例如，在依赖结构中，动词"prefer"的参数直接与其相连，而在短语结构树中，它们与动词的连接则更远。

基于转换的依赖解析

基于转换的依赖解析是一种高效的解析方法，使用栈和输入缓冲区来逐步构建依赖树。这种方法包括三个主要操作符：LEFTARC、RIGHTARC 和 SHIFT。LEFTARC 在栈顶单词和栈顶第二个单词之间建立主从关系，并删除第二个单词；RIGHTARC 在栈顶第二个单词和栈顶单词之间建立主从关系并删除栈顶单词；SHIFT 则将输入缓冲区头部的单词移入栈中。

这种操作符集的定义被成为弧标准（arc standard）方法，它有两个显著特点，第一是转换操作符只在栈顶元素之间指定依赖关系，第二是当一个元素被分配了它的主词，它就会从栈中移除，不再用于进一步处理。还有一些其他的基于转换的系统有不同的解析行为，但弧标准方法非常有效且易于实现。

解析算法

基于转换的解析算法通过一系列转换操作来逐步建立句子的依赖结构。每个步骤中，解析器根据当前配置选择正确的转换操作符，从而构建出最终的依赖树。图 15.6 展示了这种算法的基本流程：

```
function DEPENDENCYPARSE(words) returns dependency tree

  state ← {[root], [words], [] }  ; initial configuration
  while state not final
    t ← ORACLE(state)         ; choose a transition operator to apply
    state ← APPLY(t, state)   ; apply it, creating a new state
  return state
```

Figure 15.6 A generic transition-based dependency parser

在每一步，解析器会查询一个 oracle，该 oracle 负责提供在当前配置下应使用的正确转换操作符。然后解析器将该操作符应用于当前配置，生成一个新配置。该过程在所有单词都已被处理并且根节点是栈中唯一剩余元素时结束。

Oracle 的作用

在基于转换的方法中，oracle（即决策函数）通过机器学习模型进行训练，训练好的 oracle 能够在解析的每一步选择最优的转换操作符。训练一个高效的 Oracle 需要构建一个包含配置-转换对的训练集，这些二元对是从已知的参考解析结果中生成的，具体步骤如下：

1.构建初始配置：解析器从一个初始配置开始，该配置包含一个栈、一个输入缓冲区和一个空的依赖关系集。栈中包含根节点（ROOT），输入缓冲区中包含句子的所有单词。

2.模拟解析过程：解析器逐步处理输入缓冲区中的单词，每一步都基于当前配置选择一个转换操作符。为了生成训练数据，每一步的选择由参考解析结果指导，即 Oracle 在每个配置下选择根据参考解析结果能生成正确依赖关系的操作符。

3.记录配置-转换对：在每一步，记录当前的配置和所选的转换操作符。这些记录构成训练集中的实例，用于训练机器学习模型。

4.处理所有单词：继续上述步骤，直到输入缓冲区中的所有单词都被处理完，并生成完整的依赖树。整个过程生成了大量的配置-转换对，涵盖了多种句子结构和依赖关系。

下图举例说明了生成训练数据的具体过程：

Step	Stack	Word List	Predicted Action
0	[root]	[book, the, flight, through, houston]	SHIFT
1	[root, book]	[the, flight, through, houston]	SHIFT
2	[root, book, the]	[flight, through, houston]	SHIFT
3	[root, book, the, flight]	[through, houston]	LEFTARC
4	[root, book, flight]	[through, houston]	SHIFT
5	[root, book, flight, through]	[houston]	SHIFT
6	[root, book, flight, through, houston]	[]	LEFTARC
7	[root, book, flight, houston]	[]	RIGHTARC
8	[root, book, flight]	[]	RIGHTARC
9	[root, book]	[]	RIGHTARC
10	[root]	[]	Done

Figure 15.8 Generating training items consisting of configuration/predicted action pairs by simulating a parse with a given reference parse.

获得了训练数据后就可以开始训练 Oracle。训练过程通常采用监督学习的方法，即利用训练数据中的配置-转换对训练一个分类器。具体步骤如下：

1.特征提取：从每个配置中提取特征，用于表示配置的状态。特征的具体形式包括栈顶和缓冲区头部单词的词形、词性，以及它们之间的依赖关系。例如，栈顶单词的词形特征可以表示为 s1.w，缓冲区前端单词的词性特征可以表示为 b1.t。还可以通过组合这些特征，生成更多的特征模板，获得更好的训练效果。

2.生成训练实例：利用提取的特征和对应的转换操作符，生成训练实例。每个训练实例由特征向量和标签（转

换操作符）组成。

3.选择机器学习算法：选择适当的机器学习算法来训练分类器。常用的算法包括多项逻辑回归和支持向量机（SVM）。这些算法能够处理大量的稀疏特征，适用于依赖解析的任务。

4.训练模型：使用训练实例训练分类器。分类器通过学习特征与转换操作符之间的关系，建立一个模型，该模型能够在新的解析过程中根据配置状态选择最优的转换操作符。

5.模型评估与优化：评估分类器的性能，确保其在训练数据和测试数据上都表现良好。可以通过交叉验证等方法优化模型参数，提高分类器的准确性。

通过上述过程，训练好的 Oracle 能够在实际解析过程中高效地选择每一步的最优转换操作符，从而逐步构建出正确的依赖树。

基于转换的依赖解析的改进方法

基于转换的方法可以通过多种方式进行改进，以提高性能，解决该方法的一些明显缺陷。课件中介绍了两种常用的方法。

第一种方法是弧急切转换系统（Arc Eager）。弧急切方法能够比弧标准方法更早地断言向右的关系。在弧标准方法中，当一个元素被分配了它的主词，这个元素就会被移除，也就意味着后续无法将这个被移除的元素作为其他元素的主词。所以对于一个单词，弧标准方法必须等待以这个单词作为主词的各个依赖关系被分配完，才可能应用 RIGHTARC 操作符将它指定为其他单词的从属词。这会导致单词在栈中等待的平均时间变长，而一般来说，一个单词等待被分配它的主词的时间越长，发生错误的概率就越高。而弧急切系统允许单词尽早被指定其主词，即在单词的后续从属词还未被看到时就可以将单词分配给它自己的主词。课件中通过对 LEFTARC 和 RIGHTARC 操作符进行一些小的更改并添加一个新的 REDUCE 操作符来实现这一点：

LEFTARC：在输入缓冲区头部的单词和栈顶单词之间建立主从关系；弹出栈顶单词。

RIGHTARC：在栈顶单词和输入缓冲区头部的单词之间建立主从关系；将输入缓冲区前端的单词移入栈顶。

SHIFT：从输入缓冲区前端移除单词并将其推入栈。

REDUCE：弹出栈顶单词。

通过这些变化，解析器可以更早地确定依存关系，从而减少解析过程中潜在的错误积累。

第二种方法是束搜索（Beam Search）。基于转换方法的计算效率源于其对句子的单次遍历，贪婪地做出决策而不考虑其他选择。但这种策略也存在潜在问题——一旦做出决策就无法撤销。束搜索通过系统地探索替代的决策序列，解决了这一问题。

束搜索的方法是将广度优先搜索策略与启发式过滤器结合起来，管理大量潜在序列。具体步骤如下：

初始化议程 :解析器从一个默认初始配置开始，其中栈包含根节点，单词列表是句子的单词集合，关系集为空。

扩展议程：对于每个状态，应用所有适用的操作符，生成新的配置，并对这些配置进行评分。

更新议程：将每个新配置添加到边界中，前提是边界中有空间。如果议程的大小达到束宽度限制，只添加比分最低的配置更好的新配置，并移除最低分的元素以保持在限制内。

迭代处理：继续扩展和更新议程，直到所有状态都是最终状态。

这种方法通过允许解析器探索多个可能的解析路径，提高了解析的鲁棒性和准确性。

基于转换的依赖解析的时间复杂度

基于转换的依赖解析方法因其高效性而被广泛应用。基于转换的方法对句子的每个单词进行一次遍历。在每次遍历过程中，只会依次进行移入（SHIFT）、左弧（LEFTARC）或右弧（RIGHTARC）操作。由于每个单词在解析过程中只会被处理一次，所以整个解析过程是线性时间的。

对于长度为 n 的句子，基于转换的方法最多进行 2n 次操作（n 次 SHIFT 操作和 n 次 ARC 操作）。具体来说，每个单词需要一次 SHIFT 操作将其移入栈中，然后通过 LEFTARC 或 RIGHTARC 操作与其他单词建立依赖关系。

基于转换的方法采用贪婪算法，在每一步选择当前最优的操作符。虽然这种方法在某些情况下可能会导致次优解，但其高效性和简单性使得它在实践中表现良好。

束搜索方法虽然增加了一些计算复杂度，但在合理的束宽度下，仍能保持接近线性时间的性能。通过并行处理和优化算法实现，束搜索方法可以在实际应用中高效运行。

基于上述分析，基于转换的依赖解析方法的总体时间复杂度为 O(n)，其中 n 是句子的长度。线性复杂度使得该

方法在处理大规模语料库时具有较高的效率。

基于图的依赖解析

基于图的依赖解析通过在所有可能的依赖树中搜索最大化评分的树来进行解析。这些方法使用有向图和图论中的最大生成树（MST）算法。给定一个句子，首先构建一个完全连接的有向图，顶点表示单词，边表示所有可能的主从关系。权重反映每个可能关系的评分，由训练数据生成的模型提供。

最大生成树算法

最大生成树算法包括贪婪边选择、重新评分边成本和递归清理阶段。贪婪选择最高评分的入边，如果生成的边集合包含环，则通过递归清理阶段消除环，最终生成一个无环的最优生成树。基于图的方法能够生成非投射树，适用于词序相对自由的语言。

以下是最大生成树算法的主要步骤：

1.对每个顶点选择最高评分的入边。

2.如果生成的边集合产生一个生成树，则完成解析。

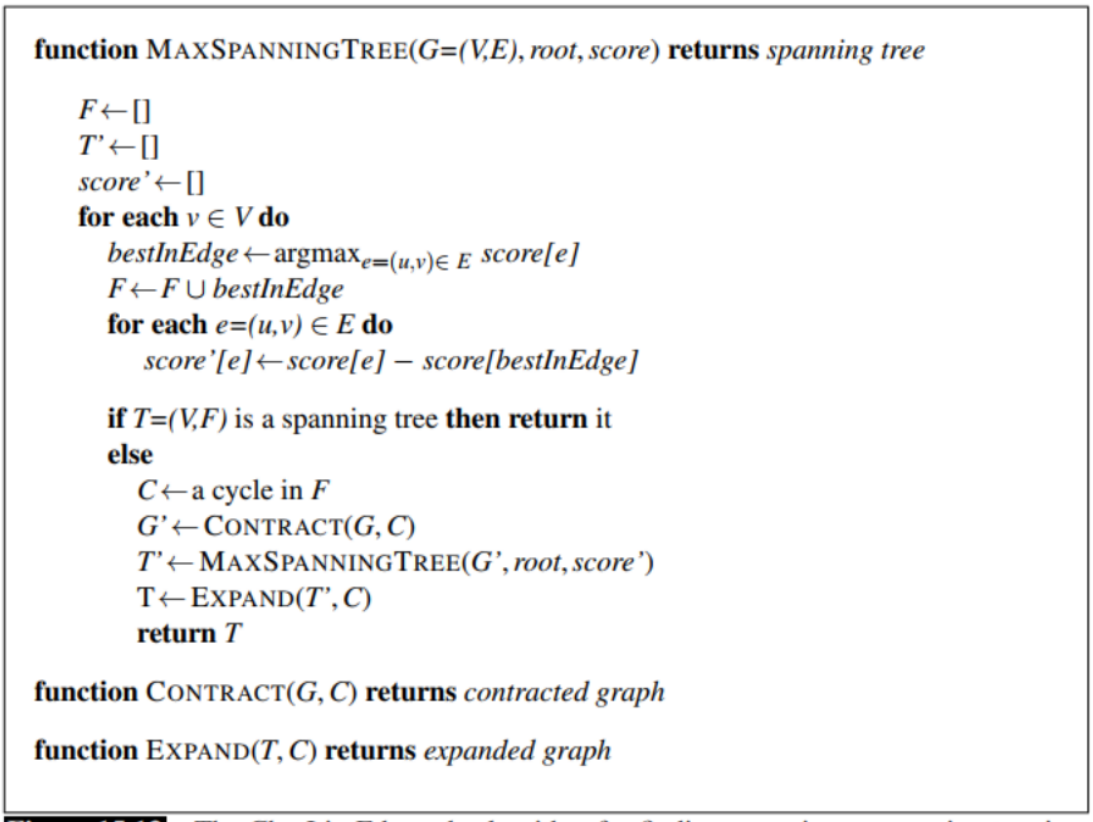
3.如果边集合包含环，调整环中边的评分，通过递归清理阶段消除环。

4.生成无环的最优生成树。

在最大生成树算法中，重新评分边成本和递归清理阶段是处理包含环的边集合的关键步骤。重新评分边成本的目的是调整环中边的权重，使得环中的边不会影响最大生成树的选择。具体来说，对环中每个顶点，找到进入该顶点的最大入边，并将其权重作为基准。然后，从进入该顶点的每条边的权重中减去这个基准权重。这种调整使得环中的最大入边的权重为零，从而不会影响生成树的选择。

递归清理阶段的目的是通过折叠环来消除环，并在调整后的图上递归地应用最大生成树算法。具体步骤如下：在边集合中找到一个环，并将其折叠为一个新的节点，这个新节点代表整个环。对进入和离开环的边进行调整，使得它们现在进入或离开这个新的折叠节点。在新的图上递归地应用最大生成树算法，直到生成无环的最大生成树。在递归结束后，展开折叠的节点，恢复环中的所有顶点和边，除了要删除的那条边。每个环中哪些边被删除是由最后生成的最大生成树决定的，即删除每个环中不在最大生成树中的边。

通过这种方式，最大生成树算法能够有效地处理包含环的边集合，最终生成一个无环的最优生成树。



```
function MAXSPANNINGTREE(G=(V,E), root, score) returns spanning tree

  F ← []
  T' ← []
  score' ← []
  for each v ∈ V do
    bestInEdge ← argmax_{e=(u,v)∈ E} score[e]
    F ← F ∪ bestInEdge
    for each e=(u,v) ∈ E do
      score'[e] ← score[e] − score[bestInEdge]

  if T=(V,F) is a spanning tree then return it
  else
    C ← a cycle in F
    G' ← CONTRACT(G, C)
    T' ← MAXSPANNINGTREE(G', root, score')
    T ← EXPAND(T', C)
    return T

function CONTRACT(G, C) returns contracted graph

function EXPAND(T, C) returns expanded graph
```

Figure 15.13 The Chu-Liu Edmonds algorithm for finding a maximum spanning tree in a weighted directed graph.

最大生成树算法的计算复杂度为 O(n^3)，其中 n 是句子的长度。这是因为算法需要在完全连接的图上操作，该图有 O(n^2)条边，每条边的操作复杂度为 O(n)。

应用和评估

依赖解析在自然语言处理的许多任务中有广泛应用，如信息提取、共指解析、问答和机器翻译等。为了评估依赖解析器的性能，常用的指标包括有标签和无标签的依赖准确率（LAS 和 UAS）。这些指标通过计算系统输出和参考解析之间的匹配关系来衡量解析器的准确性。此外，还可以使用精确率和召回率等指标评估解析器对特定依赖关系的处理能力。

信息提取

在信息提取任务中，依赖解析可以帮助识别实体和关系。例如，通过解析句子的依赖结构，可以准确地提取主语、谓语和宾语之间的关系，从而构建结构化的知识图谱。这对于自动摘要、文本分类和信息检索等任务非常有用。

共指解析

共指解析是指识别文本中指代相同实体的不同表达。依赖解析通过分析句子的语法结构，帮助识别名词短语之间的共指关系。例如，在句子"The president gave a speech. He was very eloquent."中，依赖解析可以帮助识别"the president"和"he"指代相同的实体，从而实现更准确的共指解析。

问答系统

在问答系统中，依赖解析可以帮助理解用户问题的结构，并从文本中提取答案。例如，对于问题"Who is the president of the United States?"，依赖解析可以识别问题的主语、谓语和宾语，并在文本中找到相应的答案。这对于构建智能问答系统和对话系统具有重要意义。

机器翻译

在机器翻译中，依赖解析可以帮助分析源语言和目标语言之间的句法关系，从而提高翻译的准确性和流畅性。例如，通过依赖解析，可以更好地理解源语言句子的结构，确保在翻译过程中保留句子的语义和语法信息。这对于

处理形态复杂和词序灵活的语言尤其重要。

总结

依赖解析作为一种重要的自然语言处理技术，通过建立单词之间的二元语法关系来分析句子的语法结构。依赖解析在处理形态丰富且词序相对自由的语言时具有显著优势。基于转换和基于图的方法是依赖解析的两种主要技术，前者通过栈和缓冲区的转换操作高效地构建依赖树，后者通过最大生成树算法在所有可能的依赖树中搜索最优解。随着机器学习和深度学习方法的发展，依赖解析的准确性和效率不断提高，应用前景广阔。

通过对依赖解析的深入研究，我们可以更好地理解语言的内在结构，为自然语言处理的各种应用提供有力支持。在未来的发展中，依赖解析将继续发挥重要作用，推动自然语言处理技术的不断进步。同时，依赖解析在多语言处理、语义理解和智能问答系统等领域的应用也将不断拓展，成为推动人工智能发展的重要力量。

依赖解析的研究不仅有助于提升自然语言处理系统的性能，还为语言学研究提供了新的工具和方法。通过结合计算语言学和传统语言学的理论，我们可以更深入地理解语言的复杂性和多样性，从而开发出更强大的语言处理技术。此外，依赖解析还可以与其他自然语言处理技术相结合，如词性标注、命名实体识别和语义角色标注，进一步提升系统的整体性能。

二、对于集合演算模型的认识

①文章介绍

本文的通讯作者是 Christos H. Papadimitriou，他是美国科学院院士、工程院院士，他主要从事算法与复杂性方面的研究，他也是理论计算科学（TCS）领域的顶级专家之一。看了一下他的谷歌学术页面，近几年他发表了多篇关于神经科学和脑科学方面的论文。本文是他在 2021 年发表的，刊登在学术期刊 Transactions of the Association for Computational Linguistics（CCF B 类）上。正如冯老师在班级群里分享的文章中所介绍的，本文主要是 Papadimitriou 上一篇文章的进一步工作。上一篇文章为 Brain computation by assemblies of neurons，其于 2020 年发表，刊登于美国国家科学院院刊。上一篇文章中提出了集合体 assemblies 的概念，并提出了称为交互循环网络的大脑数学模型，在此基础上实现了集合演算的操作。为了测试和应用上一篇文章中提出的理论，Papadimitriou 团队构建了一个自然语言处理系统，使用集合演算来解析英语句子。这就是我们现在看到的这篇文章的由来，所以本文和前文关系非常紧密，很多疑问可以在前文中找到答案。

②内容理解

下面我们以概念分点的形式总结论文的核心内容，下面的内容经过了我们阅读完后的总结提炼和结构调整，并对原文中一些公式的表示进行了改进，使得公式更易于阅读和理解。对于系统状态的计算、读出算法等较复杂的内容，我们还用自己理解后的语言重述了原文的表达，并额外增加了细节描述和例子描述，使各概念和算法等内容表示地更加清晰，也易于我们自己的深入理解。

1、神经元（neuron）

神经元在文中以小写字母表示，如 i、j 等等。

2、LEX

LEX 是单词所对应的神经元的集合。我们会将语言中的每个单词编码为大脑区域中不同的神经元集合，这些集合称为 LEX。后文中 LEX 表示一个重要的大脑区域。

3、集合演算 Assembly Calculus（AC）

集合演算是一个计算系统，旨在通过提供对激活的神经元动态系统的高级描述和控制，以一种程式化但生物学上合理的方式，对认知功能进行建模。

4、大脑区域（area）

每个大脑区域包含 n 个兴奋性神经元，大脑区域的数量是有限的。每个区域中的 n 个神经元通过随机加权的有向图$G_{n,p}$连接，意味着每个有序神经元对独立地具有相同的连接概率 p。大脑区域在文中用大写字母表示，如 A、B、……

5、突触（synapse）

对于链接神经元的每个突触(i,j)，都有一个突触权重w_{ij}>0，初始值为 1，其会动态变化。

6、神经纤维（fiber）

A 和 B 为两个大脑区域。对于特定的无序区域对(A,B)，$A \neq B$，存在一个随机有向二部图，对每个可能的突触以概率 p 将 A 中的神经元连接到 B 中的神经元并返回。这些区域之间的连接称为神经纤维。

7、动态系统或动力系统（dynamical system）

动态系统整体表示为一个加权有向图$G=(N,E)$，具有节点和随机有向加权边。其中 N 是节点，即各个神经元。E 是边，即突触和神经纤维。

动态系统中，时间是离散的，每个时间点之间的步长为 20 毫秒。动态系统在时间步 t 时的状态，由以下几部分组成：

（1）每个神经元 i 有一个比特f_i^t，f_i^t取值为 0 或 1，1 表示神经元 i 在时间 t 时处于激活状态。

（2）边集 E 中所有突触的突触权重w_{ij}^t

（3）抑制信息（在下文说明）

已知时间步 t 时的状态可以计算出下一个时间步 t+1 时的状态，计算方法如下所示：

步骤 1：对于每个神经元 i，计算其突触输入$SI_i^t=\sum_{(j,i)\in E,f_j^t=1} w_{ji}^t$。如果突触前连接的是神经元 j，突触后连接的是神经元 i。若神经元 j 在时间步 t 时处于激活状态，则神经元 j 会通过突出对神经元 i “放电”，表现为将突触的权重加到神经元 i 上。

步骤 2：每个神经元的突触输入SI_i^t，即与其相连的激活神经元的突触权重之和，在步骤 1 中已经计算完毕。现在将大脑区域中各个神经元的突触输入值进行比较，如果神经元 i 是其大脑区域内突触输入值最高的 k 个神经元之一，则设置$f_i^{t+1}=1$。也就是突出输入高的神经元会在下一个时间步 t+1 中被激活。

步骤 3：现在更新突触权重。对于动态系统中的每个突触(i,j)，突触前连接的是神经元 i，突触后连接的是神经元 j。如果突触后神经元 j 在时间 t+1 时激活，并且突触前神经元在时间 t 时激活，则这个突触权重会增加 1+β 倍，

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

8、cap

将某个大脑区域中在时间 t 激发的 k 个神经元的集合，称为该区域的 cap。

9、抑制信息

集合演算提供了用于动态系统的高级控制命令——抑制和去抑制（或“消抑制”、“非抑制”）。抑制通过抑制性神经元群体来实现，抑制性神经元的激活会阻止其他神经元激活。

抑制（inhibited）	一个大脑区域中的神经元被阻止激活。inhibit(A, i) 通过激活名为 i 的群体来抑制 A。
去抑制（disinhibited）	取消对一个大脑区域中的神经元的抑制。“去抑制”会抑制一群当前正在抑制 A 的抑制性神经元，操作表示为 disinhibit(A, i)，即抑制群体 i 从而让群体 i 无法抑制大脑区域 A。

神经纤维也可以被抑制：

抑制（inhibited）	阻止将突触输入传送到其他大脑区域，$\mathrm{inhibit}((A,B),i)$表示通过激活名为 i 的群体来抑制从大脑区域 A 通往大脑区域 B 的神经纤维。
去抑制（disinhibited）	取消对神经纤维的抑制，$\mathrm{disinhibit}((A,B),i)$表示抑制上述的群体 i，从而让 i 无法抑制神经纤维(A,B)。

如果大脑区域 A 在时间 t 被抑制，并在时间t'被解除抑制，则整个抑制期间，大脑区域 A 中的神经元是否处于激活维持不变，即对于$i \in A$，令$t* \in \{(t+1),(t+2),\ldots,t'\}$，有$f_i^{t*} = f_i^t$。

10、集合体（assembly）

集合体是一组特殊的 k 个神经元，它们全部位于同一大脑区域，它们紧密互连——也就是说这 k 个神经元之间的突触数量比大脑区域中的随机突触要多得多，并且这些突触具有非常高的权重。已知它们可以表示为大脑中的物体、词语、想法等。

11、投影（projection）

假设在时间 0 时，没有神经元被激活，x 是大脑区域 A 中由 k 个神经元组成的一个特定子集，通常这 k 个神经元将对应一个集合体，我们执行 fire(x)将子集 x 中的神经元激活。假设有一个相邻的大脑区域 B，A 和 B 通过神经纤维相连，初始时 B 中也没有神经元被激活。忽略其他区域的影响，如果大脑区域 A 中的集合体 x 分别在时间 0、1、2、…时激活，则其对大脑区域 B 产生的效果分别在时间 1、2、3、…时生效。由于 A 中集合体 x 激活带来的影响，B 中被激活的突触输入值最高的 k 个神经元的情况会不断演化，即 B 的 cap 会不断变化，将形成一个序列，记作y^1, y^2, y^3 …。在时间 1 时，B 中接收到来自 x 的突触输入，y^1是 B 中突触输入值最高的 k 个神经元。在时间 2 时，B 中接收到来自 x 和y^1的突触输入（A 中 x 激活，效果通过神经纤维传递到 B。B 内部y^1在上一时间步激活，效果通过区域 B 内部的突触传递到下一时间步），y^2是 B 中突触输入值最高的 k 个神经元，注意根据动态系统状态计算的步骤 3，此时从 x 到y^1的突触权重已经增加。

Papadimitriou 在前作中表明了，如果这种相互作用影响下的演化持续下去，序列$\{y^t\}$最终会高概率收敛到 B 中特定的集合体 y，将集合体 y 称为 x 在 B 中的投影。

12、活跃（active）

如果某个去抑制区域包含一个最近激活过的集合体，则称该区域活跃。A 中的活跃集合体 x 表示 A 中最近激活的 k 个神经元集合。

13、强投影（project*或 strong project）

考虑动态系统的所有非抑制区域，以及这些区域所相连的所有非抑制神经纤维，这些区域和神经纤维组成了一个无向图。强投影操作中，我们首先将动态系统中所有的活跃集合体同时激活，则激活效果会通过各个非抑制纤维进入各个相邻的非抑制区域，在相互作用下这些区域会轮流激活，直到最终趋于稳定，即被激活的神经元不再随着时间变化。我们将这个系统范围的操作表示为 project*或 strong project。project*操作的伪代码如下所示。

foreach area A **do** 遍历每个区域
  if there is active assembly x in A **then** 如果此区域存在活跃集合体x
    foreach $i \in x$ **do** 将活跃集合体x中的神经元设为激活状态
      $f_i^1 = 1$;

for $i = 1, \ldots, 20$ **do** 进行20次迭代
  all A, all $i \in A$, initialize $SI_i^t = 0$; 将神经元的突触输入初始化为0
  foreach uninhibited areas A, B **do** 遍历每个去抑制区域对(A, B)
    if fiber (A, B) inhibited **then**
      skip ; 如果A, B之间的神经纤维被抑制了，则跳过
    $x = \{i \in A : f_i^t = 1\}$; x为区域A中激活的神经元
    foreach $j \in B$ **do**
      SI_j^t += $\sum_{i \in x, (i,j) \in E} w_{ij}$; 计算A对B的突触输入

  foreach uninhibited area A **do** 遍历每个去抑制区域A
    foreach $i \in A$ **do**
      if SI_i^t in top$-k_{j \in A}(SI_j^t)$ **then** 如果i是突触输入最高的k个神经元之一
        $f_i^{t+1} = 1$; 则神经元i被激活
      else
        $f_i^{t+1} = 0$;
  foreach uninhibited areas A,B **do** 遍历每个去抑制区域对(A, B)
    if fiber (A, B) inhibited **then**
      skip ; 如果A, B之间的神经纤维被抑制了，则跳过
    foreach $(i, j) \in (A \times B) \cap E$ **do**
      if $f_i^t = 1$ and $f_j^{t+1} = 1$ **then**
        $w_{ij} = w_{ij} \times (1 + \beta)$ 更新突触权重

14、强集合演算（strong Assembly Calculus, sAC）

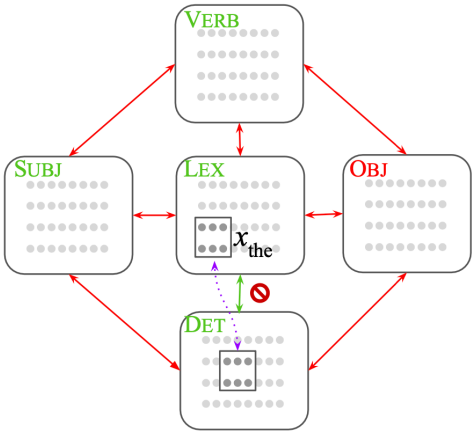
在定义了上述几种操作后，我们可以引入术语强集合演算。强集合演算是一个计算系统，具有这三种操作 :inhibit 和 disinhibit（可以应用到任意大脑区域和神经纤维），以及强投影操作 project*。强集合演算 sAC 是图灵完备的。

15、解析器（Parser）

Parser 是强集合演算中的一个程序，其数据结构是无向图 G=(A, F)，其中 A 是大脑区域的集合。F 是神经纤维集合，即无序的大脑区域对的集合。

16、LEX

LEX 是一个重要的大脑区域，其包含单词表示。在真实的人类大脑中，LEX 被认为位于左内侧颞叶。解析器的大脑区域集合 A 中，除 LEX 以外的其他大脑区域，在真实的人类大脑中可能对应于左侧颞上回中的韦尼克区域的子区域。回到解析器，LEX 通过神经纤维与解析器的大脑区域集合 A 中所有其他大脑区域相连。这些其他区域包括 VERB、SUBJ、OBJ、DET、ADJ、ADV、PREP 和 PREPP 等等。这些其他区域除了与 LEX 相连之外，也有一些通过神经纤维相互连接。如下图所示，可以看到 LEX 与其他区域都有相连，而其他区域某些二者之间有相连。



除了 LEX 之外，所有其他区域都是前面集合演算中提到的标准大脑区域。每个大脑区域包含 n 个随机连接的神经元，在任何时间点最多有 k 个神经元激活。LEX 大脑区域的性质则比较特殊，在下面和动作的概念一同介绍。

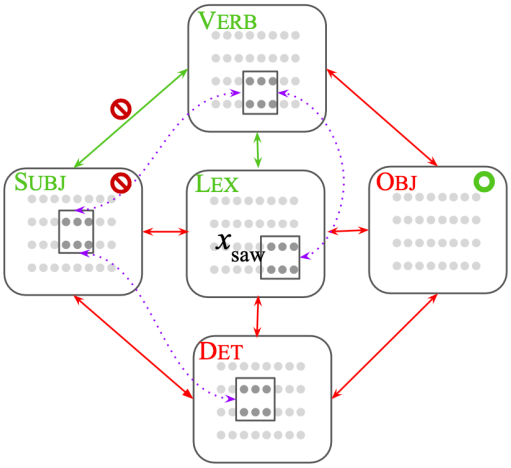
17、动作（action）

LEX 和其他大脑区域有一些不同，LEX 为语言中的每个单词包含了一个固定的集合体x_w。x_w集合体是特殊的。因为x_w的激活，除了集合体激活后对动态系统产生的普通影响之外，还需要在动态系统中执行特定于单词 w 的短程序α_w，这称为 w 动作（action of w）。直观上，LEX 集合体中所有操作的总和构成了类似于设备语法（grammar）的东西。

单词 w 的动作α_w是两组由 inhibit 和 disinhibit 操作组成的命令，其针对特定的大脑区域或神经纤维。第一组命令在动态系统的 project*操作之前执行，第二组命令在动态系统的 project*操作之后执行。比如下图是单词 saw 的动作，可以看到，第一组命令叫做前置命令（pre-commands），下图中写在左侧。第二组命令叫做后置命令（post commands），下图中写在右侧。

$$\alpha_{\text{saw}} = \left\{ \begin{array}{l} \textit{Pre-commands} = \\ \text{disinhibit((LEX, VERB), 0)} \\ \text{disinhibit((VERB, SUBJ), 0)} \end{array} \vdots \begin{array}{l} \textit{Post-commands} = \\ \text{inhibit(SUBJ), 0)} \\ \text{disinhibit(OBJ), 0)} \\ \text{inhibit((LEX, VERB), 0)} \end{array} \right\}$$

前置命令取消了对 LEX 和 VERB 之间神经纤维的抑制，还取消了对 LEX 和 SUBJ 之间神经纤维的抑制。这使得 VERB 中能够形成一个集合体，该集合体是 LEX 中的单词集合体x_{saw}和 SUBJ 中代表主语名词短语的集合体这二者的合并。因为在英语中主语 SUBJ 先于动词 VERB，所以 SUBJ 中代表主语名词短语的集合体必须是在之前已经构建好了的。由于单词 saw 是及物动词，后置规则包括在预期强制宾语时解除对 OBJ 的抑制，所以可以看到在后置命令中存在 disinhibit(OBJ, 0)。



$$\alpha_{\text{saw}} = \left\{ \begin{array}{l} \textit{Pre-commands} = \\ \text{disinhibit((LEX, VERB), 0)} \\ \text{disinhibit((VERB, SUBJ), 0)} \end{array} \vdots \begin{array}{l} \textit{Post-commands} = \\ \text{inhibit(SUBJ), 0)} \\ \text{disinhibit(OBJ), 0)} \\ \text{inhibit((LEX, VERB), 0)} \end{array} \right\}$$

作者认为组成集合体x_w的 k 个已激活神经元，包含某些神经元子群体（subpopulations），这些神经元子群体的激活会引起对某些神经群体适当的抑制和去抑制，这些子群体可以在所有及物动词之间共享，从而像上面强制宾语的例子一样实现某些后置规则。

解析器按顺序处理句子中的每个单词。对于每个单词，我们激活其在 LEX 中对应的集合体，并执行单词对应的前置命令。然后，我们执行 project*操作，沿着去抑制的神经纤维在去抑制的区域之间进行投影。最后，我们执行全部后置命令。对应的伪代码如下所示。

input : a sentence s 输入：一个句子s
output: representation of dependency parse of s, rooted in VERB 输出：句子s的依赖解析的表示

disinhibit(LEX, 0) ; 取消对LEX区域的抑制
disinhibit(SUBJ, 0) ; 取消对SUBJ区域的抑制
disinhibit(VERB, 0) ; 取消对VERB区域的抑制
foreach word w in s **do** 遍历句子s中的每个单词
  activate assembly x_w in LEX ; 激活LEX中单词w对应的集合体x_w
  foreach pre-rule (Dis)inhibit($\square, i$) in
   $\alpha_w \rightarrow$ Pre-Commands **do** 执行单词w对应的动作中的每条前置命令
    (Dis)inhibit($\square, i$) ;
*project** ; 执行project*操作
foreach post-rule (Dis)inhibit($\square, i$) in
 $\alpha_w \rightarrow$ Post-Commands **do** 执行单词w对应的动作中的每条后置命令
  (Dis)inhibit($\square, i$)

18、读出（Readout）

完成上面的算法后，我们能够得到句子的依赖关系树。在解析器无向图 G 的大脑区域之间和内部的突触连接w_{ij}中，依赖关系树被隐式地表示。通过下面介绍的读出（readout）算法，给定 G 的状态（事实上，只需要权重w_{ij}和 VERB 区域中最后激活的和 k 个神经元），能够输出一个依赖项列表。

进入读出算法前，先介绍一下相关定义。操作try − project(x, B)表示将某个区域 A 中的集合体 x 投影到区域 B 中，要求只有当能够在 B 中产生稳定的集合体时才执行投影，比如 B 中的 cap 随着重复激活而变化就是不稳定的情况。如果能够在 B 中产生稳定的集合体，try − project(x, B)将返回 B 中的集合体 y，否则将返回 NONE。另一个定义getWord()是一个函数：在任何给定时间，当集合体x_w在 LEX 中处于活跃状态时，该函数返回相应的单词 w。

读出算法的伪代码如下所示。

input : G after parsing, with active root assembly v in VERB 输入：解析后的无向图G，大脑区域VERB中带有活跃集合体v
output: the parse tree stored implicitly in G 输出：隐式存在在无向图G中的依赖解析树

initialize stack s as $\{(v, Verb)\}$; 将堆栈s初始化为(v, Verb)
initialize dependencies $\mathcal{D} = \{\}$; 依赖项集D初始为空集
while s not empty **do** 堆栈s非空时执行循环
  (x,A) = s.pop() ; 弹出s栈顶元素(x, A)
  project(x, LEX) ; 将集合体x投影到LEX区域中
  w_A = *getWord*() ; w_A是此时LEX中活跃状态的集合体所对应的单词
  foreach area $B \neq A$ s.t. $(A, B) \in \mathcal{F}$ 遍历与A相连的所有其他区域B
   do
    y = *try-project*(x, B) ; 尝试将集合体x投影到区域B中
    if y *not None* **then** 如果可以在区域B中形成稳定的集合体y
      project(y, LEX) ; 将集合体y投影到LEX区域中
      w_B = *getWord*() ; w_B是此时LEX中活跃状态的集合体所对应的单词
      add dependency $w_A \xrightarrow{B} w_B$ to $\mathcal{D}$; $w_A \rightarrow w_B$添加到依赖项集D中
      s.insert((y, B)) ; 将(y, B)压入栈
return $\mathcal{D}$; 返回依赖项集D

读出算法主要是证明解析器工作的一种手段，它演示了解析器无向图 G 的状态（即突触权重）和依赖项列表之间的映射。

③总结和感想

本篇论文是第一篇通过实验模拟神经元和突触来再现人类复杂语言现象的文章，Papadimitriou 教授参与组织了

大量脑科学和神经科学相关的实验，从真实的大脑行为中对人类大脑中的文字表征情况进行建模。本文中基于集合演算的解析器能够通过数学证明和模拟来验证这些操作均符合在大脑中观察到的行为。

本篇文章富有开创性。在人工智能的早期发展阶段，许多研究人员热衷于模仿大脑的结构和功能，希望通过这种方式构建智能系统，并通过神经科学和脑科学领域的知识对新出现的模型进行解释。不过随着人工智能领域的发展，更多种类的模型和算法随之出现，很多模型不再局限于模仿生物神经系统，而是更加注重解决实际问题和提高性能，例如深度学习在多领域取得了巨大成功，尽管神经网络的灵感来源于对生物神经系统的理解，但是如今已很少有人再从这方面对深度模型的效果进行解释，深度学习自身也面临着可解释性差的问题。同时如今模型还融合大量其他领域的知识和技术，例如强化学习受益于马尔可夫决策过程和优化理论。这些导致如今大部分研究人员都较少从神经科学和脑科学的领域对新兴的模型进行探索。正如作者所写道的，七十年前就已提出了集合体的假设，人们对大脑如何处理语言的问题在大量先前工作中已有了许多见解，然而这些进展都未出现关于神经元活动产生语言精确方式的具体叙述。而 Papadimitriou 教授团队从神经科学、脑科学、心理语言学等领域出发，再次从人类大脑中找寻灵感，成功将人类大脑中复杂的语言过程进行建模并完成实验模拟，这项工作具有重要的价值。

本文从脑科学出发的立场能够带来我们很多启发。文中集合演算模型提供了一种新的思路，即通过模拟神经元和突触的活动来实现复杂的语言现象。这一思想可以带给人工智能领域的模型设计很多新的思考，尤其是在自然语言处理领域。同时本文深入探讨了语言处理背后的神经基础，这有助于我们更好地理解人类语言能力的本质，探索更深层次的语言处理机制，甚至有助于解决语言障碍的产生机理问题，并基于此开发建立在神经科学原理上的智能系统。

④文章翻译

虽然本文发表于 2021 年，但是其开创的研究和跨学科的特点使其具有里程碑意义，我们也注意到本文在中文互联网上并没有任何相关的资料，所以我们决定在阅读的同时，顺便将文章翻译成中文，以供低年级学生或其他非专业领域的人员进行阅读。下面是论文主要 5 个章节的翻译，均是人工根据英文翻译和校对。为了保证翻译的正确性，有一些细节还需要检查和完善，在完成后我们将写成博客发布以进行分享。

A Biologically Plausible Parser

abstract

本文实现了一种英语解析器，其由生物学上合理的神经元和突触实现，并且使用了最近提出的认知函数计算框架——集合演算（Assembly Calculus）。论文实验中仅涉及了简单的英语句子，但他们的实验结果表明这个解析器还可以扩展到其他方向，从而能够涵盖很多其他语言。比如，他们提出了一个简单的俄语版本的解析器，并讨论了如何处理递归、嵌入和一词多义。

1 Introduction

语言是一种独特的人类功能，涉及创造、表达、理解和维护有关世界的层次结构信息。毫无疑问，语言是通过神经元和突触的活动来实现的——但是如何实现呢？在认知实验方面，即心理语言学、计算心理语言学和大脑成像方面，已经有了大量先前的工作，这引发了人们对大脑如何处理语言的许多见解。然而，这些进展尚未出现关于单个神经元活动产生语言的精确方式的具体叙述。特别是，我们没意识到有哪个实验通过模拟神经元和突触再现了相当复杂的语言现象。这就是本文追求的方向。

神经科学的现状阻碍了对人脑神经元制造语言的方式进行全面的计算理解，神经科学（a）主要研究人类以外动物的感觉和运动大脑功能；以及（b）关于计算建模，只专注于神经元和神经元回路的水平，缺乏理解高级认知功能如何从神经元活动中产生所需的高级计算模型。

最近，提出了一种大脑功能的计算模型，即集合演算 Assembly Calculus（AC）。AC 描述了一个动态系统，涉及以下部分和属性，这在神经科学文献中得到了充分证明： a) 神经元之间具有随机连接的大脑区域； 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 个最活跃的神经元放电； d) Hebbian 可塑性的简单模型，突触的强度随着神经元的放电而增加。这些特性产生了一个重要的物体：集合体（assembly），一大组高度互连的兴奋性神经元，全部位于同一大脑区域。

Hebb 在七十年前就提出了集合体的假设，并二十年前在实验动物的大脑中被发现。人们越来越认识到，集合体在大脑的工作方式中发挥着核心作用，并且最近

被称为“大脑的字母表”。集合体可以通过几乎同时的激发来表示一个物体、情节、单词或想法。Papadimitriou 等人的工作显示了这一点，无论是在理论还是在模拟中，集合体都是上述条件(a)–(d)下动态系统的一个新兴属性。

在 AC 中，动态系统还使得，通过比如投影、交互投影、关联、模式完成和合并等操作，来创建和操纵集合体成为可能。这些操作在两个正交意义上是现实的：首先，它们对应于在实验中观察到的集合体的行为，或者有助于解释其他实验。其次，它们可证明与单个神经元和突触的活动相对应（它们“编译（compile down）”）；在 Papadimitriou 等人的工作中，这种对应关系通过数学和模拟得到了证明。Papadimitriou 等人提出了假设，认为 AC 可能是高级认知功能的基础。特别是，在讨论部分中，提出了 AC 的一种称为合并（merge）的特定操作可能在句子的生成中发挥作用。

请注意，关于 AC 的健全性，通过数学证明确定的是 AC 操作能够高概率正确工作，其中潜在的概率事件是随机连接组的创建。到目前为止，用 AC 进行的仿真实验已经证明，AC 的单个命令或其短序列可以在模拟器中成功可靠地执行。这样的模拟可以扩展吗？例如，能否在 AC 中实现一项计算要求较高的认知功能，例如句子解析，以及由此产生的动态系统是否稳定可靠？这就是我们下面描述的实验的本质。

在本文中，我们提出了一个由 AC 驱动的解析器。换句话说，我们设计并模拟了一个涉及程式化神经元、突触和大脑区域的生物学真实动态系统。我们首先将语言中的每个单词编码为大脑区域中不同的神经元集合，我们称这些集合为 LEX。接下来，我们向这个动态系统提供一系列单词（作为激发 LEX 中相应的单词编码组件序列的信号）。

重要的问题是：动态系统能否正确解析句子？我们怎么知道？

首先要回答后一个问题，我们的动态系统有一个读出（Readout）程序，在处理序列后，它会重新访问系统的所有区域并恢复链接结构。我们要求这个结构是输入句子的解析。我们的实验表明，在上述意义上，我们的解析器确实可以正确解析相当不简单的句子，事实上，解析速度（即神经元激活的周期数）与语言器官的速度相当。

我们设计的该设备涉及多个大脑区域以及连接这些区域的突触纤维（fibers of synapses），并使用 AC 的操作，并以第 3 节中解释的小方式进行增强。原则上可以使用 Papadimitriou 等人工作中的原始 AC 操作集，但这会使它的运作变得复杂，并需要引入更多的大脑区域。由此产生的设备依赖于强大的单词表示，并且本质上是一个词汇化的解析器，产生类似于依赖图的东西。

虽然我们的实验需要解析相当简单的英语句子，但在第 6 节中，我们认为我们的解析器可以在许多不同的方向上扩展，例如错误检测和恢复、一词多义和歧义、递归和英语以外的语言。我们还构建了一个仅作为玩具的俄语解析器，以及一个通用设备，它以单词表示、语法（syntactic）动作和连接的大脑区域的形式输入语言描述。

目标。本研究旨在探讨上面已经强调的两个问题：

1. 是否可以通过模拟神经元和突触来实现相当复杂的语言现象，例如句子的解析？

2. AC 能否实现计算要求较高的认知功能？由此产生的动态系统是否稳定可靠？

特别是，我们并不是说我们对解析器的实现一定类似于大脑实际实现语言的方式，或者它可以预测实验数据。相反，我们将解析器视为完全由模拟神经元动力学构建的重要语言装置的存在证明。

3 The Assembly Calculus

本节介绍我们的模拟器和实验中使用的集合演算 assembly calculus（AC）的版本。这个版本的 AC 与 Papadimitriou 等人之前的工作几乎相同，但包括此处描述的细微修改。AC 是一个计算系统，旨在通过提供对激活的神经元动态系统的高级描述和控制，以一种程式化但生物学上合理的方式，对认知功能进行建模。大脑区域 A、B……的数量是有限的，每个区域（area）包含 n 个兴奋性神经元。每个区域中的 n 个神经元通过随机加权的有向图$G_{n,p}$连接，这意味着每个有序神经元对独立地具有相同的连接概率 p。每个突触(i,j)都有一个突触（synapse）权重$w_{ij}>0$，初始值为 1，其会动态变化。对于特定的无序区域对(A,B)，$A \neq B$，存在一个随机有向二部图，对每个可能的突触以概率 p 将 A 中的神经元连接到 B 中的神经元并返回。这些区域之间的连接称为神经纤维（fiber）。总而言之，动态系统是一个大型动态加权有向图$G=(N,E)$，具有节点和随机有向加权边。

事件以离散的时间步长发生（将每个步长视为 20 毫秒）。动态系统在每个时间步 t 的状态由几部分组成：(a) 对于每个神经元 i，一个比特$f_i^t \in \{0,1\}$表示 i 是否在时间 t 激活。(b) E 中所有突触的突触权重w_{ij}^t。给定时间 t 的状态，时间 t+1 的状态计算如下：

1. 对于每个神经元 i，计算其突触输入$SI_i^t = \sum_{(j,i)\in E, f_j^t=1} w_{ji}^t$，即在时间 t 时激活的突触前神经元的所有权重总和。

2. 对于每个神经元$f_i^{t+1}=1$——也就是说，i 会在时间 t+1 时触发，如果 i 是其区域内具有最高SI_i^t的 k 个神经元之一（任意打破任何联系）。

3. 对于每个突触$(i,j)\in E$，$w_{ij}^{t+1}=w_{ij}^t\left(1+f_i^t f_j^{t+1}\beta\right)$；也就是说，当且仅当突触后神经元在时间 t+1 激活并且

突触前神经元在时间 t 激活时，突触权重才会增加 1+β 倍。

我们将某个区域中在时间 t 激发的 k 个神经元的集合，称为该区域的 cap。

上面这些是动态系统的方程。AC 还提供用于该系统的高级控制的命令。一个区域可以被“抑制（inhibited）”（即，其中的神经元分别被阻止激活）和“去抑制（disinhibited）”（取消抑制，如果其当前有效）。我们还假设神经纤维可以被抑制（阻止将突触输入传送到其他区域）。在大脑中，抑制是通过抑制性神经元群体来实现的，这些神经元的激活会阻止其他神经元激活。事实上，一个区域或一根光纤上可能有多个群体。我们用 inhibit(A, i)来表示这个命令，它通过激活名为 i 的群体来抑制 A。类似地，“去抑制”会抑制一群当前正在抑制 A 的抑制性神经元（例如上面的 i），这种操作表示为 disinhibit(A, i)。例如，如果我们 inhibit(A, i)并 inhibit(A, j)，然后 disinhibit(A, j)，A 仍然被抑制（因为 i）。最后，我们假设纤维(A, B)可以类似地被抑制或取消抑制，表示为 inhibit((A, B), i)和 disinhibit((A, B), i)。

我们现在定义这个动态系统在时间 t 的状态。状态包含每个神经元的激活状态、边权重w_{ij}和抑制信息。如果区域 A 在时间 t 被抑制，并在时间t'被解除抑制，我们假设对于所有$i \in A,\ j \in (t+1) \dots t', f_i^j = f_i^t$。即，在整个抑制期间神经元是否激活维持不变。

系统的一个重要的新兴属性是集合体的属性。集合体是一组特殊的 k 个神经元，全部位于同一区域，它们紧密互连——也就是说，这 k 个神经元之间的突触数量比随机的要多得多，并且这些突触具有非常高的权重——已知可以表示为大脑中的物体、词语、想法等。

假设在时间 0，当没有其他东西激活时，我们对区域 A 中的 k 个神经元 x 的固定子集（通常这 k 个神经元将对应于一个集合体）执行 fire(x)，并假设有一个相邻区域 B（通过纤维连接到 A），B 中当前没有神经元激活。由于区域 A 中的集合体 x 在时间 0、1、2、…时激活（忽略所有其他区域），它会在时间 1、2、…时生效。B 中一组不断演化的 k 个神经元的激活，是一系列 cap，称这些集合为$y^1, y^2, \dots$。在时间 1，y^1将是 B 中恰好接收到来自 x 的最高突触输入的 k 个神经元。在时间 2，y^2将是 B 中接收来自 x 和y^1组合的最高突触输入的神经元集合，并且回想一下从 x 到y^1的突触权重已经增加。如果这种情况持续下去，Papadimitriou 等人的工作就表明了这一点，序列$\{y^t\}$最终以高概率（其中概率空间是系统的随机连通性）收敛到 B 中的稳定集合体 y，称为 x 在 B 中的投影（the projection of x in B）。AC 中有更多的高级操作（倒数投影、合并、关联等），包括能够执行任意空间有界计算的计算框架。在这里，我们将只关注投影，尽管如下所述进行了增强。

假设区域 A 中的一个集合体 x 如上投影到区域 B，形成一个新的集合体 y，在此过程中，B 中的神经元激活回 A。非常直观的是，在投影之后，y 是密集互连的，并且由于它的构建方式，它与 x 中的神经元具有密集的突触连接。因此，如果 x 再次激活，y 也会效仿。在本文中，我们还假设（并且我们的实验验证了这个假设），由于 A 和 B 之间的纤维是相互的，因此从 y 到 x 也存在很强的突触连接，因此，如果 y 激活，x 接下来也会激活。

接下来我们定义投影操作的增强——相当于一系列投影操作——我们称之为强投影（参见算法 1）。考虑动态系统的所有去抑制区域，以及包含它们的所有去抑制神经纤维。这定义了一个无向图（在我们的使用中它始终是一棵树）。如果某个去抑制区域包含一个集合体——即一个最近激活的集合体在这个区域之中，则将该区域称为活跃（active）。现在，假设所有这些集合体同时激活，通过每个去抑制纤维进入每个其他去抑制相邻区域，并且这些区域轮流激活，可能创建新的集合体并进一步向下激活树，直到过程稳定（即相同的神经元不断从一步到下一步激活）。我们将这个系统范围的操作表示为 strong project 或 project*。请注意，project*几乎是糖衣语法（syntactic sugar），因为它只是缩写了一系列投影（这可以在 AC 模型中完成）；然而，我们使用的活跃（active）区域的概念只是对 AC 的一个小补充。虽然这个修改很小，但它简化了解析器的实现，不过它可能会以牺牲更多的 AC 大脑区域和时间步长为代价而被删除。

我们对解析器的实验表明，操作 project*确实按照描述的方式工作，即过程始终稳定。我们引入术语“强集合演算”（strong Assembly Calculus, sAC）来指代这种计算系统，其运算为：适用于任何区域、任何纤维的 inhibit 和 disinhibit 运算，以及强投影运算 project*。不难看出 sAC 是图灵完备的，就像 Papadimitriou 等人中展示的 AC 一样，但我们不需要这个。

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**
        $w_{ij} = w_{ij} \times (1 + \beta)$

在上面的伪代码中，“A 中的活跃集合体 x（active assembly x in A）”表示 A 中最近激活的 k 个神经元集合（根据系统的动力学，它恰好是一个集合体），或一组被激活的 k 个神经元（设置为在 t=1 时触发）：我们使用它来激活与第 4 节中的固定单词相对应的集合体。

4 The Parser

4.1 Parser Architecture

Parser 是 sAC 中的一个程序，其数据结构是无向图 G=(A, F)。A 是一组大脑区域，F 是一组神经纤维，即无序的区域对。一个重要的区域是 LEX，对于词典（lexicon）来说，其包含单词表示。LEX 在大脑中被认为位于左侧内侧颞叶（MTL），通过神经纤维与所有其他区域相连。A 的其余区域可能对应于左侧颞上回（STG）中韦尼克（Wernicke）区域的子区域，单词从左侧内侧颞叶 MTL 投影到这些子区域，以进行语法（syntactic）角色分配。在我们下面的实验和叙述中，这些区域包括 VERB、SUBJ、OBJ、DET、ADJ、ADV、PREP 和 PREPP。除了 LEX 之外，这些区域中也有一些通过神经纤维相互连接（见图 2）。这些区域中的每一个都是假设的，因为解析器似乎有必要正确地处理某些类型的简单句子。事实证明，它们大致但无误地对应于依赖标签（dependency labels）。A 可以扩展更多区域，例如 CONJ 和 CLAUSE；有关其中一些扩展的讨论，请参阅第 6 节。

除了 LEX 之外，所有这些区域都是 AC 的标准大脑区域，包含 n 个随机连接的神经元，其中在任何时间点最多有 k 个神经元激活。相反，LEX 为语言中的每个单词包含一个固定的集合体$\mathbf{x_w}$。$\mathbf{x_w}$集合体是特殊的，因为$\mathbf{x_w}$的激活，除了激活集合体对系统的普通影响之外，还需要执行特定于单词 w 的短程序α_w，称为 w 动作（action of w）。直观上，LEX 集合体中所有操作的总和构成了类似于设备语法（grammar）的东西。

单词 w 的动作α_w是两组 INHIBIT 和 DISINHIBIT 命令，针对特定的区域或神经纤维。第一组在系统的 project*操作之前执行，第二组在系统的 project*操作之后执行。

单词“chase”的动作如图 1 所示（图 2 中给出了其他词类的更多动作示例及其命令）。事实上，每个标准及物动词都有相同的动作。前置命令（pre-commands）取消（disinhibit）了对从 LEX 和 SUBJ 到 VERB 的神经纤维的抑制，允许在 VERB 中形成一个集合体，该集合体是 LEX 中的单词集合体$\mathbf{x_{chase}}$和 SUBJ 中代表主语名词短语的集合体的合并，因为在英语（我们关于英语的子集）中主语先于动词，所以到目前为止必须已经构建好了此集合体。由于 chase 是及物动词，后置规则（post-rule）包括在预期强制宾语时解除（disinhibition）对 OBJ 的抑制。就神经元和突触而言，我们认为组成集合体$\mathbf{x_w}$的 k 个已激活神经元包含某些神经元子群体（subpopulations），这些神经元亚群的激活会激发适当的抑制和去抑制某些神经群体（populations）（通过一个延迟算子（delay operator）在后置命令（post commands）中的神经元群体）。这些子群体可以在所有及物动词之间共享。

4.2 Operation

如算法 2 所示，解析器按顺序处理句子中的每个单词。对于每个单词，我们激活其在 LEX 中的集合体，应用其词汇项（lexical item）的前置命令（pre-commands）。然后，我们执行 project*，沿着去抑制的神经纤维在去抑制的区域之间进行投影。然后，应用任何后置命令。

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($\square, i$)

在上面的伪代码中，$\square \in A \cup F$ 可以表示区域或神经纤维，具体取决于命令。

4.3 Readout

完成算法 2 后，我们将论证句子的依存树在 G 的大脑区域之间和内部的突触连接w_{ij}中被隐式地表示。为了验证这一点，我们有一个读出算法（readout），给定 G 的状态（事实上，只需要 VERB 中最后激发的w_{ij}和 k 个神经元），输出一个依赖项列表。我们的实验验证了当用算法 1 解析句子后将其应用于 G 的状态时，我们得到了完整的依赖关系（dependencies）集。

为了符号方便，我们定义一个操作 try−project(x, B)，其将某个区域 A 中的集合体 x 投影到区域 B 中，但前提是这会在 B 中产生稳定的集合体。这相当于，在解析器执行期间，仅在 x 被投影到 B 时才执行投影（作为某个步骤中 project*的一部分）。最后，将 getWord()定义为这样的函数：在任何给定时间，当集合体x_w在 LEX 中处于活跃（active）状态时，该函数都返回相应的单词 w。在下面的伪代码中，如果成功，try−project(x, B)将返回 B 中的集合体 y，否则返回 NONE。

我们提出的读出算法主要是作为证明解析器工作的一种手段，因为它演示了解析器图 G 的状态（即它的突触（synaptic）权重）和依赖项列表之间的映射。然而，我们注意到这在生物学上并非不可信。try-project 并不是一个新的原语（primitive），并且可以通过神经元、突触和激活来实现。最简单的是，x 可以激活到 B 中，如果 B 中生成的 k-cap 不稳定（例如，随着重复激活而变化），则它不是一个集合体。或者（在模拟中），可以从 B 投影到 LEX 中，如果 B 中的 cap 不是解析过程中形成的稳定集合体，则 LEX 中的 k-cap 也将不是集合，即不对应于任何有效的x_w（这与第 6 节中讨论的“无意义组件”有关）。此外，算法 3 的堆栈只是为了说明清楚起见；由于投影的集合体始终是上一轮投影的集合体，因此有人认为可以使用 project*来实现该算法。

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 $\{(v, Verb)\}$;
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 $\mathcal{D}$;

5 Experiments

我们在 Python 中提供了解析器的实现。为了实现这一目标，我们显着扩展了现有的 AC 模拟库（Papadimitriou 等人的工作，2020）（例如添加抑制机制，并使像 LEX 这样的大脑区域具有固定的、预定义的集合体）。重要的是，整个算法在最低级别上是通过模拟单个神经元和突触来运行的，并且任何更高层次的概念或结构（例如解析树本身）都是一种抽象，其完全编码在代表大脑的巨大图和动态系统的个体激活和神经元到神经元突触权重中。

我们提供了一组由解析器正确解析的 200 个不同的句子。对于每个句子，我们都有一个目标正确的依存结构；我们验证读出（Readout）操作是否为每个句子生成这种依赖结构。

这些句子是手工设计的，目的是证明我们的解析器（和 AC）可以处理的各种句法现象；这包括介词短语、及物性和不及物性（包括既可以是及物也可以是不及物的动词）、可选的副词放置在动词之前或之后等等。请参阅第 8 节了解这些例句及其所代表的句法（syntactic）模式。我们提醒读者，据我们所知，这是第一个在单个神经元和突触级别上实现的现实解析器。

解析的 200 个句子是从解析器设计用来处理的 10

个句法模式和 100 个单词的词汇表中抽取的。由于底层动态系统原则上会以低概率发展出不稳定性，因此失败仍然是可能的，但在我们的实验中并没有发生。有一些语法结构是 Parser 目前无法处理的，但我们认为在我们的模型中，这些语法结构在两个意义上是可能的：首先，语法（单词动作）可以扩展以处理这些结构，更重要的是，动态系统仍将以高概率正确解析它们。这种结构的一个主要例子是句子嵌入（"the man said that..."）。有关此类扩展的更多讨论，请参阅第 6 节。我们模拟中的解析速度与神经语言处理的已知界限相称。在每个步骤中，参与的集合体必须激活足够多次数以使 project*运行稳定；我们通过仿真发现，对于模型超参数的合理设置，收敛会发生在 10-20 个激活周期内，即使涉及多个大脑区域。假设每秒 50-60 个神经元尖峰（AC 时间步），并允许在单词动作中（并行）执行抑制/去抑制操作，这将需要少量的额外周期，我们达到了 0.2-0.5 秒/字的频率范围。

6 Extensions and Discussion

UParser 和俄语解析器。解析器的底层元算法，我们称之为 UParser，其将一组区域 A、一组神经纤维 F、一组单词 W 和每个$w \in W$的一组动作a_w作为输入（其命令仅操作 A 中的区域），并使用第 3 节的算法 2 通过强集合演算（sAC）进行解析。UParser 与语言无关，可以被视为对语言的通用神经基础进行建模：一个特定语言通过指定 A（大致是其语法成分）、F、W 和a_i来实例化该基础。这绝不限于英语；例如，我们的模型和算法同样能够很好地处理词序相对自由的高度屈曲的（inflectional）语言，其中句法（syntactic）功能以形态表示。我们通过一个解析器来演示这一点，该解析器针对俄语的一个作为玩具的子集。俄语是一种在简单句子中具有自由词序的语言。特别是，我们的解析器处理一组在排列下闭合的句子；同一句子的排列被正确解析，并生成相同的依赖树（如算法 3 所验证）。有关俄语解析器的更多详细信息，请参阅第 8 节。

大的坏问题和递归。第 3 节的解析器是相当概要性的，其需要被扩展到处理递归结构，例如复合和嵌入子句。考虑一下 “The big bad problem is scary” 这句话。按照第 3 节的解析器，在单词 “big” 之后，在 ADJ 中创建了代表 “big” 的集合体 a，系统预计在后续步骤中（当遇到名词时）将其投影到主语 SUBJ 区域中，以在 SUBJ 中形成集合体代表主语 NP。然而，下一个单词不是名词，而是 “bad”，是形容词链的一部分。现在，如果我们从 LEX 投影到 ADJ 中形成代表 “bad” 的程集合体 b，我们就会丢失对 a 的所有引用，并且无法恢复单词 “big” ！对于链接问题有几种简单且合理的解决方案，它们也可用于解析副词链、复合名词和其他复合结构。

一种解决方案是在 A 中使用两个区域COMP_1和COMP_2。当我们遇到第二个形容词时（或者更一般地说，同一词性的第二个单词，否则会触发同一区域，覆盖之前的集合体），我们从 LEX 投影到COMP_1而不是 ADJ，但同时从 ADJ 投影到COMP_1将第一个形容词与第二个形容词联系起来。对于链中的第三个形容词，我们将 LEX 投影到COMP_2，但也将COMP_1单向（unidirectionally）投影到COMP_2；一般来说，对于除 LEX 之外的形容词 i，我们从$\mathrm{COMP}_{\mathrm{parity(i)}}$（包含之前的形容词组合）单向投影$\mathrm{COMP}_{\mathrm{parity(i-1)}}$。我们在模拟中演示了这两个区域的不同长度的解析链。然而，该解决方案的一个限制是它需要单向神经纤维；如果上面从COMP_1到COMP_2的投影是相应的（reciprocal），则无法将COMP_2中的集合体链接到COMP_1中的另一个集合体。

另一种可能更现实并且不需要单向纤维的方法（尽管它不能处理任意长度的链）是添加 2 个以上的区域，$\mathrm{COMP}_2, \ldots, \mathrm{COMP}_m$，对于一些小但合理的 m，可能是 7。解析过程与两区域时的解决方案一样，但通过将COMP_i链接到COMP_{i+1}来进行。区域数量 m 可以模拟处理此类复杂结构的认知极限，其经过充分研究，参见 Grodner 和 Gibson (2005)；卡尔森（2007）的工作。为了在高要求的情况下或通过练习来模拟更长的链，大脑可以招募额外的大脑区域。

这样的操作可以处理形式为$S \rightarrow A^*!A^+$的右递归。中部嵌入递归比较复杂。为了给具有嵌入句子或关系从句的句子构建解析树，对于解析器的内循环的执行，两个句子的主语和/或动词可能需要共存于相应的区域中。这不是问题，因为大脑区域可以处理数以万计的集合体，但必须保留连接结构。当嵌入语句被服务时，必须能够保存嵌入语句的状态。一种可能性是引入 CURRENTVERB 区域，它可以像上面的复合区域一样链接起来，并将充当这种要求更高的递归形式的激活记录堆栈。这个想法是，通过恢复嵌入句子的动词，我们还可以通过链接，并在嵌入句子结束后，恢复嵌入句子开始时嵌入句子的完整状态，并继续解析。这需要实现和试验。

消除二义性（Disambiguation）。在英语中，单词 rock 可以是不及物动词、及物动词或名词；到目前为止，我们都忽略了一词多义和二义性的难题。为了处理一词多义，每个单词 w 可能需要有许多动作集$\alpha_w^1, \alpha_w^2, \ldots$并且解析器必须消除这些动作集之间的歧义。

我们相信解析器可以扩展来处理这种二义性。动作的选择可以通过分类器来计算，将当前句子中几个单词的前瞻（look-ahead）和后瞻（look-behind）（或者可能只是它们的词性）作为输入，并选择一个动作集；分类器可以在以前见过的句子的语料库上进行训练。这也需要实现和实验。

错误检测。语法判断是任何语言的母语者判断格式是否良好的直觉和内在能力；这包括检测语法错误的能力。神经语言学家对这种现象的神经基础越来越感兴趣；例如，最近的实验发现，当一个句子片段突然以语法上非法的方式继续时，会检测到一致的信号。解析器内置了检测一些格式错误的句子的能力。对此至少有两种简单的机制。一种是当一个片段被一个单词继续时，该单词立即使其不符合语法，例如在 “the dogs lived” 后面加上 “cat” 。 解析器在处理了不及物动词 “lived” 之后，并没有取消对区域 OBJ 的抑制，并且所有其他与名词相关的区域都被抑制。遇到 “cat” 时，project*不会激活任何集合体；我们称之为空投影（empty-project）错误。

其他类型的语法违规可以通过无意义集合体（nonsense assembly）错误来检测。当一个区域被投影到 LEX 中时，在读出过程中就可能出现这样的错误，但是 LEX 中的结果集合体对于任何$w \in W$都不对应于x_w；换句话说，当第 3 节的读出算法的函 getWord()失败时，这表明 G 的状态必定是由不可能的句子产生的。我们提供了一个非法句子列表，我们的解析器模拟可以检测到空投影或无意义集合体错误，表明不同类型的语法违规。

我们的解析器当前无法检测到的一种语法错误是数字协议（number agreement:）：我们的模拟器不会抱怨输入 “Cats chases dogs;” 通过使用单独的区域来投影单数和复数形式，这个问题并不难解决。其他类型的协议，例如性别协议，也可以类似地对待。

布罗卡（Broca）区的作用。据观察，语言处理，尤其是短语和句子的形成，会激活左额叶的布罗卡区；到目前为止，解析器仅对被认为位于左侧 STG 和 MTL 中的过程进行建模。我们假设布罗卡区可能涉及构建一个概括句子的简洁语法树，该语法树仅由主语、动词和宾语（如果有的话）组成（但可以通过其他区域访问已解析句子的其余部分）。这将涉及新的区域 S（用于句子）和 VP（用于动词短语），神经纤维从 VERB 和 OBJ 到 VP，以及从 VP 和 SUBJ 到 S。构建基本的三叶语法树可以在后台被动地进行，而其余的语法处理则在我们当前的模型中进行。

封闭式课程和替代架构。诸如 the、of、my 之类的词可能位于 Wernicke 区域中专用于封闭词性的区域中，而不是像我们为简单起见在模型和模拟中假设的那样成为 LEX 的一部分。事实上，在探索这里描述的架构之前，我们考虑了神经科学文献中提出的这个方向的替代方案。

假设左侧 MTL 中的 LEX 仅包含识别单词所需的语音和形态信息，并且所有语法信息（例如动作a_w）都位于 Wernicke 区域中，也许又是在与我们在这里假设的子领域不同的子领域。例如，SUBJ 可以包含每个名词（在俄语中，以其主格形式），作为与 LEX 中相应组件来回永久投影的组件；OBJ（俄语宾格）也是如此。动词都永久投射在 VERB 中，依此类推。在这个模型中，句法动作是特定于区域的，而不是单词。我们计划进一步探索这样的解析器会出现哪些新问题，以及它将解决哪些问题，以及它在性能、复杂性和生物学合理性方面可能有哪些优点和缺点。

我们模型的预测。解析器本质上是一个概念证明，一个 “存在定理（existence theorem）”，它确立了复杂的认知功能，特别是语言功能，可以通过集合演算以生物学上现实的方式实现。为此，我们选择了一种与大部分文献中关于语言处理的已知和信念兼容的架构。可以做出一些可以通过脑电图和功能磁共振成像实验来验证的预测。其中一项预测是，处理具有相似句法结构的句子（例如 “dogs chase cats” 和 “sheep like dogs”）会产生相似的大脑活动特征，而处理具有不同句法结构的句子（ “give them food” ）会生成不同的签名。我们目前正在与纽约市立大学认知神经科学家 Tony Ro 和 Tatiana Emmanouil 合作设计和进行此类实验。

如果能够对人类大量单个神经元进行广泛记录，则可以对 Parser 模型进行具体测试，因为它可以预测与可识别 LEX 区域中的词汇项相对应的长期集合体。还应该可以实时观察与句法结构和解析过程中形成的词性相对应的集合体的形成；还可以识别相应的区域（例如动词、主语等）。在识别区域时，这些区域的精确解剖位置可能因人而异，但在任何个体中都是一致的。我们的最终预测是，长期寻求的语言神经基础由——或者更确切地说，可以有效地抽象为——左 MTL 和 STG 以及大脑其他地方的大脑区域和神经纤维的集合组成，由 project*操作提供支持，并在关键时期适应个人的母语和其他情况。

未来的工作。未来的工作将集中于扩展解析器的范围。这包括上面提到的扩展（特别是嵌入和递归）。另一个重点是将这项工作与计算心理语言学的现有方向相结合。这包括增强解析器以展示第 2 节中描述的标志性心理语言学需求。我们的解析器实际上具有与该文献相同意义上的增量性，但是实现中间结构的连通性会很有趣。另一个未来的方向是考虑如何使用 AC 中实现的解析器来预测或建模实验数据（例如处理时间）。最后，我们强调了未来工作的一个悬而未决的问题，即情境（contextual）行动问题。我们给定带有标签的依赖项集 D 的句子 s，以及一个解析器，其中包含 D 中出现的每个标签的区域，以及 LEX。每个单词$w \in s$是否存在上下文动作a_w^*，使得解析算法与读出相结合，产生 D？我们能构建一个返回上下文动作的 oracle 函数$O(s, w)$吗？如果不能，那么在这种形式化下可以识别哪种标记依赖

树集合？

这些行为如何在神经上表现出来？我们能否基于单词及其在 s 中的直接邻居，在 AC 中合理地实施上下文操作？实现这一目标的一个步骤可能是首先在神经元和突触水平上实现区域和神经纤维的抑制和去抑制，这在本文中被视为原始操作（通过对具有负边缘权重的抑制群体进行建模，并将它们连接到 LEX 中的集合体）。

三、分工

熊蔚然：阅读，负责依赖解析部分的写作
林宇浩：阅读，负责集合演算部分的写作
王哲：阅读，参与讨论和问题探究