张量逻辑：人工智能的语言

Pedro Domingos Paul G. Allen 计算机科学与工程学院 华盛顿大学 西雅图, WA 98195-2350, USA PEDROD@CS.WASHINGTON.EDU

摘要

人工智能的进步因缺乏一种具备所有必需功能的编程语言而受到阻碍。像 PyTorch 和 TensorFlow 这样的库提供了自动微分和高效的 GPU 实现，但它们只是 Python 的附加组件，而 Python 从未为人工智能设计。它们对自动推理和知识获取的支持不足，导致了一系列长期且代价高昂的、试图将这些功能强行附加的权宜之计。另一方面，像 LISP 和 Prolog 这样的人工智能语言缺乏可扩展性和对学习的支持。本文提出了张量逻辑，这是一种通过在基础层面统一神经网络和符号人工智能来解决这些问题的语言。张量逻辑中唯一的构造是张量方程，它基于一个观察：逻辑规则和爱因斯坦求和本质上是相同的操作，而其他一切都可以归结为它们。我将展示如何在张量逻辑中优雅地实现各种关键的人工智能形式，包括变换器、形式推理、核机器和图模型。最重要的是，张量逻辑为新的方向提供了可能，例如在嵌入空间中进行可靠的推理。这结合了神经网络的可扩展性和可学习性，以及符号推理的可靠性和透明性，并可能成为人工智能更广泛应用的基础。

关键词: 深度学习、自动推理、知识表示、逻辑编程、爱因斯坦求和、嵌入、核机器、概率图模型

解读

这篇摘要为我们即将展开的智力探险设定了宏伟的舞台。作者佩德罗·多明戈斯（Pedro Domingos）是人工智能领域的思想领袖，他在这里提出了一个大胆的主张：当前人工智能领域之所以遇到瓶颈，比如我们常听说的模型“幻觉”（胡说八道）或不可解释的“黑箱”问题，其根本原因在于我们缺少一种“正确的语言”。

想象一下，你想建造一架先进的喷气式飞机，但手头只有制造马车的工具和图纸。你可以用这些工具强行拼凑，甚至给马车装上一个喷气引擎（就像 Python 加上 PyTorch/TensorFlow），它或许能跑得飞快，但它本质上仍然是一辆结构脆弱、难以控制的马车，而不是一架真正的飞机。作者认为，这就是今天人工智能的现状。我们用一种通用语言（Python）和一些强大的“引擎”（深度学习库）来解决问题，但这种组合在处理逻辑、知识和推理这些更高级的认知任务时显得力不从心。

这篇论文的“英雄”——张量逻辑（Tensor Logic）——应运而生。它的核心思想极具颠覆性：逻辑推理（比如“所有人都会死，苏格拉底是人，所以苏格拉底会死”）和神经网络中的数学运算（爱因斯坦求和，一种处理高维数组的强大数学工具）实际上是同一枚硬币的两面。如果这个观点成立，那就意味着我们可以创造一种单一、优雅的语言，既能像神经网络一样从数据中学习，又能像逻辑学家一样进行严谨的推理。

摘要中提到的“在嵌入空间中进行可靠的推理”是这篇论文最激动人心的承诺。所谓“嵌入空间”，你可以想象成一个高维的“意义地图”，在这个地图里，“国王”和“皇帝”的坐标很近，“香蕉”和“水果”的坐标也很近。现代人工智能模型（如 GPT）就是在这个空间里进行模糊的、基于相似性的“思考”，这导致它们有时会犯一些常识性错误。张量逻辑的目标是在这个强大的“意义地图”上建立一套严格的交通规则（逻辑），使得模型的思考既能利用数据学习到的直觉，又能遵循逻辑的严谨性，从而变得既强大又可靠。这不仅是一项技术突破，更可能是一场范式革命。

1. 引言

当一个领域找到了它的语言，它就会腾飞。牛顿发明了微积分，物理学才得以起飞，没有微积分就不可能实现。没有亥维赛的矢量微积分符号，麦克斯韦方程组将无法使用。正如数学家和物理学家喜欢说的那样，一个好的符号就成功了一半。没有复数，大部分电气工程将无法实现；没有布尔逻辑，数字电路也无从谈起。现代芯片设计得益于硬件描述语言，数据库得益于关系代数，互联网得益于互联网协议，而万维网则得益于 HTML。更普遍地说，没有高级编程语言，计算机科学就不会有长足的发展。定性领域也同样关键地依赖于它们的术语。即使是艺术家，也依赖于其流派的习语和风格惯例进行创作。

一个领域的语言为其从业者节省时间，集中他们的注意力，并改变他们的思维方式。它将领域团结在共同的方向上，减少熵。它使关键事物变得显而易见，并避免了反复从头开始拼凑解决方案。

人工智能找到它的语言了吗？LISP 是最早的高级编程语言之一，它使符号人工智能成为可能。在 80 年代，Prolog 也变得流行起来。然而，这两种语言都存在可扩展性差和缺乏学习支持的问题，并最终被像 Java 和 C++ 这样的通用语言所取代，即使在人工智能领域内也是如此。图模型为概率人工智能提供了一种通用语言，但它们的适用性受到推理成本的限制。

解读

这一部分通过一系列精彩的历史类比，强调了一个核心观点：工具塑造思想。作者告诉我们，科学和工程的每一次重大飞跃，都伴随着一种新“语言”或“符号系统”的诞生。微积分不仅仅是计算曲线下面积的工具，它提供了一种全新的方式来思考变化和运动，从而解锁了整个经典物理学。矢量符号不仅仅是简化了麦克斯韦方程，它让物理学家能够直观地“看到”电磁场的行为。

这个观点对我们理解科学的本质至关重要。一个好的符号系统就像一副特制的眼镜，戴上它，原本混乱复杂的世界突然变得清晰有序。它能将我们从繁琐的细节中解放出来，让我们专注于问题的核心。作者在这里暗示，人工智能领域目前正戴着一副“不合适的眼镜”，这副眼镜让我们看东西很费劲，限制了我们的想象力。

接着，作者对人工智能历史上曾尝试过的“语言”进行了回顾，就像一位历史学家在审视过去的失败尝试，以找出未来的方向。

- **LISP 和 Prolog**：它们是“符号人工智能”的语言，擅长处理逻辑和规则，就像一位严谨的哲学家。你可以用它们轻松地编写“如果 A 且 B，则 C”这样的规则。但它们的问题在于，面对现实世界海量的、模糊的数据时，它们显得非常脆弱和低效，就像一个哲学家无法处理股票市场的实时数据一样。它们缺乏从数据中“学习”的能力。

- **图模型**：这是“概率人工智能”的语言，擅长处理不确定性。它用一种优雅的图形方式来表示变量之间的概率关系，比如“下雨”这个事件会增加“地面湿”的概率。但它的问题是，当关系网络变得非常复杂时，进行推理计算（比如计算某个事件发生的精确概率）的成本会呈指数级增长，变得不切实际。

- **Java 和 C++**：这些是强大的通用语言，但它们对于人工智能来说太“底层”了。用它们来写 AI 模型，就像用最基本的砖块和水泥来建造一座摩天大楼，虽然可行，但极其繁琐，你大部分时间都在和语言本身搏斗，而不是在思考人工智能的核心问题。

通过这番回顾，作者为我们描绘出了一幅“语言真空”的图景。人工智能领域充满了各种强大的思想，但缺少一种能够将这些思想优雅、高效地表达出来的统一语言。这正是张量逻辑试图填补的空白。

像马尔可夫逻辑这样的形式主义无缝地结合了符号和概率人工智能，但也受到推理成本的阻碍。Python 目前是人工智能事实上的语言，但它从未为此而设计，这一点显而易见。像 PyTorch 和 TensorFlow 这样的库提供了自动微分和 GPU 实现等重要功能，但对于自动推理和知识获取等关键任务毫无帮助。神经符号人工智能试图通过将深度学习模块与符号人工智能模块相结合来改善这一点，但最终往往兼具两者的缺点。总而言之，人工智能显然还没有找到它的语言。

这样一种语言有明确的期望特性。与 Python 不同，它应该隐藏所有与人工智能无关的东西，让 AI 程序员能够专注于重要的事情。它应该便于将先验知识整合到 AI 系统中，并对其进行自动推理。它还应该便于自动学习，并且最终得到的模型应该是透明和可靠的。它应该能够毫不费力地扩展。符号 AI 具有其中一些特性，深度学习具有另一些，但两者都不具备全部。因此，我们需要将它们融合。

张量逻辑通过统一它们的数学基础来做到这一点。它基于这样一个观察：基本上所有的神经网络都可以用张量代数构建，所有的符号 AI 都可以用逻辑编程构建，而这两者在根本上是等价的，唯一的区别在于所使用的原子数据类型。我将首先简要回顾逻辑编程和张量代数。论文的核心部分定义了张量逻辑，并描述了其推理和学习引擎。然后，我将展示如何在其上优雅地实现神经网络、符号 AI、核机器和图模型。我将展示张量逻辑如何实现嵌入空间中可靠和透明的推理。我提出了两种扩展它的方法。论文最后讨论了张量逻辑的其他潜在用途、其广泛应用的前景以及实现这一目标的后续步骤。

解读

在这一段，作者将矛头直指当前人工智能领域的“王者”——Python。他毫不客气地指出，Python 成为 AI 语言纯属历史偶然，它本身并不是为此而生的。这就像让一位优秀的诗人去写工程蓝图，他或许能完成任务，但过程必然是笨拙和低效的。PyTorch 和 TensorFlow 这些库，就像是给这位诗人配备的强大计算器和绘图工具。它们能解决计算问题（如自动微分和 GPU 加速），但无法解决语言固有的结构性问题——即如何自然地表达逻辑和知识。

“神经符号人工智能”是当前解决这个问题的一种流行尝试。它的思路很直接：既然神经网络擅长处理数据，符号 AI 擅长处理逻辑，那我们把它们俩“粘”在一起不就行了？作者对此持怀疑态度，认为这种方法常常是“两害相权取其重”，最终得到一个既笨重又脆弱的系统。这就像把法拉利的引擎装在一辆拖拉机上，你既得不到法拉利的速度，也失去了拖拉机的可靠性。

于是，作者清晰地列出了他心目中理想 AI 语言的“愿望清单”（desiderata）：

1. **专注性**：隐藏所有无关细节，让开发者只关心 AI 问题本身。
2. **知识与推理**：能轻松地把人类已有的知识（比如物理定律或语法规则）写进模型，并让模型能基于这些知识进行推理。
3. **学习能力**：能高效地从数据中学习。
4. **透明与可靠**：模型不应是“黑箱”，它的决策过程应该是可以理解和验证的。
5. **可扩展性**：能够轻松处理大规模的数据和模型。

这五点精确地指出了当前深度学习（擅长 3 和 5）和符号 AI（擅长 2 和 4）的各自优缺点。作者的雄心壮志是要创造一种语言，同时满足所有这五个条件。

最后，作者揭示了张量逻辑的“惊天秘密”：神经网络的数学基础（张量代数）和符号 AI 的基础（逻辑编程）在本质上是等价的！它们的区别仅仅在于处理的数据类型不同：逻辑编程处理的是离散的符号（比如“是”或“否”），而张量代数处理的是连续的数值。如果能打通这层窗户纸，就能创造出一个真正统一的框架。这正是本文的核心论点，也是接下来所有精彩内容的理论基石。

2. 背景

2.1 逻辑编程

符号人工智能中最广泛使用的形式是逻辑编程（Lloyd, 1987）。最简单的逻辑编程语言，也足以满足我们的目的，是 Datalog（Greco and Molinaro, 2016）。一个 Datalog 程序是一组规则和事实。事实是形如 r(o1,...,on) 的陈述，其中 r 是一个关系名，o 是对象名。例如，`Parent(Bob, Charlie)` 陈述了鲍勃是查理的父母，而 `Ancestor(Alice, Bob)` 陈述了爱丽丝是鲍勃的祖先。规则是形如 A0←A1,...,Am 的陈述，其中箭头表示“如果”，逗号表示合取（AND），每个 A 的形式为 r(x1,...,xn)，其中 r 是关系名，x 是变量或对象名。例如，规则 `Ancestor(x, y) ← Parent(x, y)` 表示父母是祖先，而规则 `Ancestor(x, z) ← Ancestor(x, y), Parent(y, z)` 表示如果 x 是 y 的祖先，并且 y 是 z 的父母，那么 x 是 z 的祖先。非正式地，一条规则表示，如果存在已知事实使得其右侧（或称为“体”）中的所有关系同时为真，那么其左侧（或称为“头”）也为真。

解读

这一节为我们进入张量逻辑的世界铺设了第一块基石：逻辑编程。作者选择了 Datalog 作为代表，因为它足够简单又能体现核心思想。我们可以把 Datalog 理解为一种极其严谨的“知识表达与推理系统”。

这个系统由两种基本元素构成：

1. **事实 (Facts)**：这是我们知识库里的基本信息，是无需证明的、确凿的陈述。比如 `Parent(Bob, Charlie)`（鲍勃是查理的父亲），这就是一条事实。你可以把它们想象成数据库里的一行行记录，或者是一个家族图谱上已知的、直接的亲子连线。

2. **规则 (Rules)**：这是我们用来从已知事实中推导出新知识的“推理引擎”。一条规则的结构是“结论 ← 条件1, 条件2,...”。这里的 `←` 符号读作“如果”，逗号 `,` 读作“并且 (AND)”。

    - `Ancestor(x, y) ← Parent(x, y)` 这条规则说：“如果 x 是 y 的父母，那么 x 也是 y 的祖先。” 这是一个基本定义。

    - `Ancestor(x, z) ← Ancestor(x, y), Parent(y, z)` 这条规则是关键，它定义了“祖先”这个概念的递归性。它说：“如果 x 是 y 的祖先，并且 y 是 z 的父母，那么 x 就是 z 的祖先。” 这条规则就像一个多米诺骨牌效应。如果我们知道爱丽丝是鲍勃的祖先（事实1），鲍勃是查理的父亲（事实2），那么通过这条规则，我们就能推导出“爱丽丝是查理的祖先”这个新知识。这里的变量 `y` (鲍勃) 就像一个连接点，把两条信息串联了起来。

这个系统之所以强大，是因为它能从有限的、明确的事实出发，通过几条简单的、通用的规则，推导出无穷无尽的、隐含的知识。这正是人类逻辑思维的一种简化模型。理解 Datalog 的核心——事实和规则——是理解张量逻辑如何将这种符号推理能力融入数学计算的关键第一步。

例如，上述规则和事实意味着 `Ancestor(Alice, Charlie)` 为真。

在数据库术语中，一个 Datalog 规则是一系列连接（join）后跟一个投影（projection）。两个关系 R 和 S 的（自然）连接是所有可以由 R 和 S 中具有相同参数值的元组形成的元组集合。当两个关系没有共同参数时，它们的连接简化为它们的笛卡尔积。一个关系 R 在其参数子集 G 上的投影是通过从 R 的元组中丢弃所有不在 G 中的参数而得到的关系。例如，规则 `Ancestor(x, z) ← Ancestor(x, y), Parent(y, z)` 在 y 上连接了关系 `Ancestor(x, y)` 和 `Parent(y, z)`，并将结果投影到 x,z 上；元组 `Ancestor(Alice, Bob)` 和 `Parent(Bob, Charlie)` 产生了元组 `Ancestor(Alice, Charlie)`。

逻辑编程中两种常见的推理算法是前向链和后向链。在前向链中，规则被反复应用于已知事实以推导新事实，直到无法推导出更多事实为止。结果被称为程序的演绎闭包或不动点，所有感兴趣的问题都可以通过简单地检查它来回答。例如，对于查询 `Ancestor(Alice, x)`（“爱丽丝是哪些人的祖先？”），根据上述规则和事实，答案是 {Bob, Charlie}。

后向链试图通过寻找匹配查询的事实，或头部匹配查询且体部匹配事实的规则来回答问题，如此递归进行。例如，查询 `Ancestor(Alice, Charlie)` 不匹配任何事实，但它匹配规则 `Ancestor(x, z) ← Ancestor(x, y), Parent(y, z)`，并且该规则的体部匹配事实 `Ancestor(Alice, Bob)` 和 `Parent(Bob, Charlie)`，因此答案为真。

Datalog 中的前向链和后向链是可靠的推理过程，意味着它们给出的答案保证在逻辑上遵循程序中的规则和事实。逻辑程序既有声明性语义，也有过程性语义，意味着一条规则既可以被解释为关于世界的陈述，也可以被解释为通过调用体部中的过程并组合结果来计算其头部的过程。

归纳逻辑编程（ILP）领域关注于从数据中学习逻辑程序（Lavrač and Džeroski, 1994）。例如，一个 ILP 系统可能会从一个小的父母和祖先关系数据库中归纳出上述规则。一旦归纳出来，这些规则就可以回答任何长度、涉及任何人的祖先链问题。一些 ILP 系统还可以进行谓词发明，即发现数据中未明确出现的关系，类似于神经网络中的隐藏变量。

解读

这一部分用更专业的数据库和算法术语重新解释了 Datalog 的工作原理，这对于理解它与张量代数的对应关系至关重要。

- **连接 (Join) 和 投影 (Projection)**：这是理解规则的关键。
    - **连接**：想象你有两张表格，一张是“祖先-后代”表，另一张是“父母-子女”表。规则 `Ancestor(x, z) ← Ancestor(x, y), Parent(y, z)` 的核心操作就是“连接”这两张表。连接的条件是第一张表的“后代”列 (`y`) 必须等于第二张表的“父母”列 (`y`)。例如，`Ancestor(Alice, Bob)` 和 `Parent(Bob, Charlie)` 这两条记录可以通过 `Bob` 这个共同点连接起来，形成一个临时的、更长的记录 `(Alice, Bob, Charlie)`。这个操作找到了推理链中的“中间人”。
    - **投影**：在连接之后，我们得到的记录 `(Alice, Bob, Charlie)` 包含了中间人 `Bob` 的信息。但我们最终的结论 `Ancestor(Alice, Charlie)` 并不需要他。所以，我们进行“投影”操作，丢掉中间的 `y` 列，只保留 `x` 和 `z` 列，得到最终结果 `(Alice, Charlie)`。这个操作提取了推理的最终结论。

- **前向链 (Forward Chaining) vs. 后向链 (Backward Chaining)**：这是两种不同的推理策略。
    - **前向链**：是一种“数据驱动”的方法。它从所有已知事实开始，像一部不知疲倦的机器，应用所有规则，生成所有可能的新事实，再用这些新事实继续生成更多事实，直到知识库不再增长为止。这就像把所有原料都倒进一个大锅里，让它们充分反应，最后得到所有可能的产物。回答问题时，只需在最终的产物库里查找即可。
    - **后向链**：是一种“目标驱动”的方法。它从一个具体的问题（查询）开始，比如“爱丽丝是查理的祖先吗？”。然后它会反向查找：要证明这个，我需要什么？哦，根据规则，我需要证明存在一个 y，使得“爱丽丝是 y 的祖先”并且“y 是查理的父母”。然后它会继续递归地去证明这两个子目标。这就像一个侦探，从结论出发，一步步反向追溯证据链。
- **归纳逻辑编程 (Inductive Logic Programming, ILP)**：如果说 Datalog 是从规则推导事实（演绎），那么 ILP 就是反过来，从大量事实中“猜”出规则（归纳）。这正是“学习”的本质。比如给机器看大量的家族图谱，它能自动学习到“祖先”的递归定义。更神奇的是“谓词发明”（predicate invention），这相当于机器在学习过程中自己创造了一个新的、有用的概念，比如它可能会发现“祖母”这个概念很有用，即使数据里从未明确标注过。这与神经网络中的“隐藏层”学习到有意义的特征有着深刻的相似之处，为后文的统一埋下了伏笔。

2.2 张量代数

一个张量由两个属性定义：它的类型（实数、整数、布尔等）和它的形状（Rabanser et al., 2017）。张量的形状包括它的阶（rank，即索引的数量）和每个索引的维度大小（size）。例如，一个视频可以表示为一个形状为 (t, x, y, c) 的整数张量，其中 t 是帧数，x 和 y 是一帧的像素宽度和高度，c 是颜色通道数（通常为 3）。矩阵是二阶张量，向量是一阶张量，标量是零阶张量。一个阶为 r、第 i 维大小为 ni 的张量总共包含 $\prod_{i=1}^{r} n_i$ 个元素。张量 A 在维度 1 的位置 i1、维度 d 的位置 id 等处的元素表示为 $A_{i1,\ldots,id,\ldots,ir}$。张量的这个通用元素常被用来代表张量本身。

两个形状相同的张量 A 和 B 的和是一个张量 C，使得 $C_{i1,\ldots,id,\ldots,ir}=A_{i1,\ldots,id,\ldots,ir}+B_{i1,\ldots,id,\ldots,ir}$。

两个阶分别为 r 和 r′ 的张量 A 和 B 的张量积是一个阶为 r+r′ 的张量 C，使得 $C_{i1,\ldots,id,\ldots,ir,j1,\ldots jd',\ldots,jr'}=A_{i1,\ldots,id,\ldots,ir}B_{j1,\ldots,jd',\ldots,jr'}$。

爱因斯坦标记法通过省略所有求和符号来简化张量方程，并隐式地对所有重复的索引进行求和。例如，$A_{ij}B_{jk}$ 表示矩阵 A 和 B 的乘积，对 j 进行求和，得到一个索引为 i 和 k 的矩阵： $C_{ik}=A_{ij}B_{jk}=\sum_j A_{ij}B_{jk}$

更一般地，两个具有共同索引 β 的张量 A 和 B 的爱因斯坦求和（简称 einsum）是一个张量 C，使得 $C_{\alpha\gamma}=\sum_\beta A_{\alpha\beta}B_{\beta\gamma}$, 其中 α、β 和 γ 是索引集，α 是 A 的索引中未出现在 B 中的子集，α 和 β 的元素可以以任何顺序交错，B 和 γ 也类似。神经网络中几乎所有的线性和多线性运算都可以简洁地表示为 einsum（Rocktäschel, 2018; Rogozhnikov, 2022）。

与矩阵一样，张量可以分解为更小张量的乘积。特别是，塔克分解（Tucker decomposition）将一个张量分解为一个更紧凑的、同阶的核心张量和 k 个因子矩阵，每个因子矩阵将核心张量的一个索引扩展为原始张量的一个索引。例如，如果 A 是一个三阶张量，用爱因斯坦标记法，其塔克分解为 Aijk=MipMjq′Mkr″Cpqr, 其中 C 是核心张量，M 是因子矩阵。

解读

现在，我们转向第二块基石：张量代数。如果说逻辑编程是符号 AI 的语言，那么张量代数就是神经网络的语言。这一节的目标是让读者熟悉这种语言的基本词汇和语法。

- **张量 (Tensor)**：初学者常常对“张量”这个词感到困惑，但它的概念其实非常直观，是“数组”的推广。
    - **0阶张量**：一个单独的数字，比如 5。也叫**标量 (scalar)**。
    - **1阶张量**：一列数字，比如 ``。也叫**向量 (vector)**。
    - **2阶张量**：一个数字网格，比如一个 3x3 的矩阵。也叫**矩阵 (matrix)**。
    - **3阶张量**：一个数字立方体。一张彩色图片就是一个很好的例子，它的三个维度分别是：图片高度、图片宽度、颜色通道（红、绿、蓝）。
    - 更高阶的张量就是更高维度的数组，虽然难以可视化，但在数学上是同样的概念。
- **爱因斯坦求和 (Einstein Summation / einsum)**：这可以说是现代深度学习库中最强大、最优雅的工具之一。它的核心思想是一个简单的约定：“**如果一个索引在一个公式的右边出现了两次，就意味着要对这个索引的所有可能值进行求和，并从最终结果中消掉这个索引。**”
    - 以矩阵乘法 Cik=∑jAijBjk 为例。在爱因斯坦标记法中，我们直接写成 Cik=Aij Bjk。因为索引 `j` 在右边出现了两次（一次在 A 中，一次在 B 中），求和符号 ∑j 就被“心照不宣”地省略了。而 `i` 和 `k` 各只出现一次，所以它们保留在结果 C 中。
    - 这个看似简单的省略，实际上是一种极其强大的编程范式。它能用一行代码表达各种复杂的多维数组运算，如转置、点积、外积、张量收缩等，极大地简化了代码，使其更接近数学公式本身。这正是作者所说的“一个好的符号就成功了一半”。

- **塔克分解 (Tucker Decomposition)**：这是张量的一种“压缩”方法。想象一个巨大的、复杂的三阶张量 A（比如一段视频数据）。塔克分解告诉我们，这个大张量可能可以被一个更小的“核心张量” C 和三个“因子矩阵” M, M', M'' 近似地表示出来。
    - 你可以把核心张量 C 看作是数据中“最本质的特征”或“基本概念”。
    - 因子矩阵 M, M', M'' 则像是三本“密码本”或“转换指南”，告诉我们如何将这些基本概念组合、变换，以重构出原始的、高维的数据。
    - 这个概念非常重要，因为它提供了一种从复杂数据中提取简洁结构的方法。在后文中，作者会把它和逻辑编程中的“谓词发明”联系起来，暗示着这种数学上的分解操作，可能对应着认知上的“概念发现”过程。

至此，作者已经分别介绍了符号 AI 和神经网络的“语言”。下一章，他将揭示这两种看似截然不同的语言之间惊人的内在联系。

3. 张量逻辑

3.1 表示

张量逻辑基于对两个关键问题的回答：张量和关系之间有什么关系？以及 Datalog 规则和 einsum 之间有什么关系？

第一个问题的答案是，一个关系是一个稀疏布尔张量的紧凑表示。例如，一个社交网络可以由邻接矩阵 Mij 表示，其中 i 和 j 遍历所有个体，Mij=1 表示 i 和 j 是邻居，否则为 0。但对于大型网络，这是一种低效的表示，因为几乎所有元素都将是 0。该网络可以更紧凑地表示为一个关系，为每对邻居提供一个元组；不在关系中的对被假定为非邻居。更一般地，一个 n 阶稀疏布尔张量可以由一个 n 元关系紧凑地表示，每个非零元素对应一个元组，并且效率增益通常随 n 呈指数增长。

第二个问题的答案是，一个 Datalog 规则是布尔张量上的一个 einsum，并对结果逐元素应用一个阶跃函数。（具体来说，是亥维赛阶跃函数，H(x)=1 如果 x>0，否则为 0。）例如，考虑规则 Aunt(x,z)←Sister(x,y),Parent(y,z). 将关系 Aunt(x,z)、Sister(x,y) 和 Parent(y,z) 视为布尔矩阵 Axz、Sxy 和 Pyz，则 Axz=H(SxyPyz)=H(∑ySxyPyz) 当且仅当对于至少一个 y，Sxy 和 Pyz 都为 1 时，结果将为 1。换句话说，einsum SxyPyz 实现了 Sister(x,y) 和 Parent(y,z) 的连接。如果 x 是 z 的姑姑/姨妈，y 就是 x 的那个同时也是 z 父母的兄弟姐妹。阶跃函数是必要的，因为通常对于给定的 (x, z) 对，可能存在多个 y 使得 Sxy=Pyz=1，导致结果大于 1。阶跃函数随后将其简化为 1。

解读

这是整篇论文的“文眼”，是所有思想汇合的核心。作者在这里揭示了他惊人的发现：逻辑和代数在底层是相通的。

第一个洞见：关系就是稀疏张量。

- 想象一个庞大的社交网络，有十亿用户。如果我们要用一个矩阵（2阶张量）来表示谁和谁是朋友，这个矩阵将有 109×109=1018 个元素。其中，如果用户 `i` 和用户 `j` 是朋友，矩阵的 `(i, j)` 位置就是 1，否则是 0。这个矩阵绝大部分元素都会是 0，因为每个人只认识极少数人。存储这样一个“稀疏”的矩阵是巨大的浪费。
- 我们通常如何存储社交网络？我们只记录那些实际存在的朋友关系，比如一个列表：`(张三, 李四)`, `(李四, 王五)`... 这就是逻辑编程中的“关系”。
- 所以，作者的第一个论断是：逻辑编程中的“关系”表示法，只不过是数学上“稀疏布尔张量”的一种高效、紧凑的存储方式。它们描述的是同一个东西。

第二个洞见：Datalog 规则就是带阶跃函数的 einsum。 这是最关键的一步。让我们再次分析 `Aunt` 这条规则：`Aunt(x,z) ← Sister(x,y), Parent(y,z)`。

- **逻辑层面**：要判断 `x`是不是`z`的阿姨，我们需要找到一个“中间人”`y`，这个`y`必须同时满足两个条件：1. `x`是`y`的姐妹 (`Sister(x,y)`)；2. `y`是`z`的父母 (`Parent(y,z)`)。只要能找到**至少一个**这样的`y`，结论就成立。
- **代数层面**：现在我们把这三个关系看作布尔矩阵 A, S, P。
    1. `S_{xy}P_{yz}`：这是爱因斯坦求和，也就是矩阵乘法。`S` 和 `P` 的元素都是 0 或 1。乘积 `S_{xy}P_{yz}` 只有在 `S_{xy}=1` **并且** `P_{yz}=1` 时才等于 1，否则为 0。这完美地对应了逻辑上的“**AND**”操作。
    2. ∑ySxyPyz：这个求和操作遍历了所有可能的“中间人”`y`。如果存在一个 `y` 使得 `S_{xy}P_{yz}=1`，那么求和结果就至少是 1。如果存在多个这样的 `y`（比如你有两个姐妹都嫁给了同一个人的兄弟），求和结果就会大于 1。这个求和操作完美地对应了逻辑上的“**存在至少一个 (EXISTS)**”操作。
    3. H(...)：亥维赛阶跃函数的作用是“取有无”。不管求和结果是 1, 2, 还是 100，只要它大于 0（意味着至少存在一个满足条件的 `y`），`H`函数就把它变成 1 (True)。如果求和结果是 0（一个满足条件的 `y` 都没有），`H`函数就让它保持 0 (False)。这确保了最终结果是一个干净的布尔值。

通过这个例子，作者雄辩地证明了，一条逻辑规则的计算过程，可以被一个张量方程完美地、一步不差地复刻出来。逻辑编程中的“连接”操作对应于张量乘法，而“投影”（消去中间变量 `y`）则对应于求和。这不再是类比，而是一种数学上的等价。

设 U 和 V 为任意张量，α、β 和 γ 为索引集。那么 Tαγ=H(UαβVβγ) 是一个布尔张量，其索引为 αγ 的元素在存在某个 β 使得 UαβVβγ=1 时为 1。换句话说，T 表示与 U 和 V 对应的关系的连接。

由于张量和关系之间、einsum 和 Datalog 规则之间存在直接的对应关系，因此也应该有直接对应于数据库连接和投影的张量操作。因此，我们定义张量投影和张量连接如下。

一个张量 T 在其索引子集 α 上的投影是 πα(T)=∑βTαβ其中 β 是 T 中不属于 α 的索引集。（β 的元素可以与 α 的元素以任何顺序交错。）换句话说，T 在 α 上的投影是对于 α 的每个值，T 中具有该 α 值的所有元素的总和。例如，一个向量可以通过求和其所有元素投影

为一个标量，一个矩阵可以通过将每行求和为一个向量的元素来投影为一个列向量，一个立方体张量可以投影到它的任何一个面上，然后那个面投影到它的一条边上，再然后投影到一个角上，等等。如果张量是布尔的，并且投影后跟一个阶跃函数，张量投影就简化为数据库投影。

两个张量 U 和 V 沿一个共同索引集 β 的连接是 $(U\times V)_{\alpha\beta\gamma}=U_{\alpha\beta}V_{\beta\gamma}$其中 α 是 U 中不属于 V 的维度子集，γ 和 V 也类似。（同样，α、β 和 γ 可以以任何顺序交错。）换句话说，两个张量在一个共同索引子集 β 上的连接，对于每个具有相同 β 值的元素对，都有一个元素，该元素是它们的乘积。如果 U 的阶为 r，V 的阶为 r′，且 $|\beta|=q$，则 U x V 的阶为 $r+r'-q$。当两个张量没有共同索引时，它们的连接简化为它们的张量积（对于矩阵是克罗内克积）。当它们所有维度都相同时，它简化为它们的逐元素乘积（对于矩阵是哈达玛积）。如果张量是布尔的，张量连接简化为数据库连接。

解读

在建立了核心的对应关系后，作者开始正式定义张量逻辑的“基本操作”。他从数据库理论中借用了“投影”和“连接”这两个术语，并赋予了它们在张量世界中的精确数学定义。

- **张量投影 (Tensor Projection)**：这个操作的本质是“**降维**”或“**信息汇总**”。它的数学形式是 $\sum_{\beta} T_{\alpha\beta}$，即对某些维度（索引 β）进行求和，从而消掉这些维度。
    - **直观例子**：
        - 你有一个向量（1阶张量），代表一个班级所有学生的身高。对这个向量进行投影（即把所有元素加起来），就得到了全班身高的总和，这是一个标量（0阶张量）。你“消掉”了“学生个体”这个维度。
        - 你有一个矩阵（2阶张量），行代表学生，列代表各科成绩。如果你对每一行进行投影（把该行的所有科目成绩加起来），你会得到一个向量，每个元素是每个学生的总分。你“消掉”了“科目”这个维度。
    - **与逻辑的联系**：在逻辑编程中，投影是丢弃中间变量。在张量逻辑中，它是通过求和来消掉对应的索引。当张量是布尔的时，对一个维度求和后跟一个阶跃函数，就等价于逻辑上的“存在”量词（OR 操作），这与数据库投影的语义完全一致。

- **张量连接 (Tensor Join)**：这个操作的本质是“**信息组合**”或“**关系建立**”。它的数学形式是 UαβVβγ，即基于共同的维度（索引 β）将两个张量的信息结合起来。
    - **直观例子**：
        - **矩阵乘法**是张量连接最经典的例子。AijBjk 就是在共同索引 `j` 上进行连接。
        - **逐元素乘积 (Hadamard product)**：如果两个矩阵 A 和 B 形状完全相同，它们的连接 (A×B)ij=AijBij 就是将对应位置的元素相乘。这可以看作是在所有索引 `(i, j)` 上进行连接。
        - **外积 (Outer product)**：如果两个向量 u 和 v 没有共同索引，它们的连接 (u×v)ij=uivj 会产生一个矩阵。
    - **与逻辑的联系**：在逻辑编程中，连接是基于共同变量将事实组合起来。在张量逻辑中，它是基于共同索引将张量相乘。当张量是布尔的时，乘法操作等价于逻辑“AND”，这与数据库连接的语义完全一致。

通过这两个定义，作者完成了一次漂亮的“概念移植”。他将逻辑推理中最核心的两个操作——连接和投影——用张量代数中最基本的两个操作——乘法和求和——进行了重新定义。这使得整个逻辑推理的过程，变成了一系列定义清晰的数学运算。

一个张量逻辑程序是一组张量方程。张量方程的左侧（LHS）是正在计算的张量。右侧（RHS）是一系列张量连接，后跟一个张量投影，以及一个可选的、逐元素应用于结果的单变量非线性函数。张量由其名称后跟一个方括号括起来的、逗号分隔的索引列表表示。连接符号被隐式省略，投影是投影到 LHS 上的索引。例如，一个单层感知器由张量方程实现 Y=step(W[i]X[i]), 其中在 i 上连接并将其投影出去，实现了 W 和 X 的点积。张量也可以通过列出其元素来指定，例如 W=[0.2,1.9,−0.7,3] 和 X=。然后输入 `Y?` 会导致 Y 被求值。

请注意，与 NumPy、PyTorch 等中的 einsum 实现一样，张量方程比原始的爱因斯坦标记法更通用：被求和的索引是那些没有出现在 LHS 中的索引，因此一个重复的索引可能被求和，也可能不被求和。例如，在 Y[i]=step(W[i]X[i]) 中的索引 i 没有被求和。下面实现多层感知器时利用了这一点。

张量元素默认为 0，具有相同 LHS 的方程被隐式求和。这既保留了与逻辑编程的对应关系，又使张量逻辑程序更短。张量类型可以声明或推断。将张量设置为等于一个文件会将文件读入张量。读取文本文件会得到一个布尔矩阵，其 ij-th 元素为 1 表示文本中的第 i 个位置包含词汇表中的第 j 个词。（当然，该矩阵不是以这种低效形式存储的；稍后会详细介绍。）例如，如果文件是字符串 "Alice loves Bob" 并读入矩阵 M，结果是 M[0,Alice]=M[1,loves]=M=1，而所有其他 i, j 的 M[i,j]=0。（请注意，任意常量，而不仅仅是整数，都可以用作索引。）反之，将文件设置为等于一个张量会将张量写入文件。

这就是张量逻辑的全部定义。没有关键词、其他构造等。但是，允许一些语法糖是方便的，虽然不增加语言的表达能力，但使编写常用程序更方便。例如，我们可以允许：一个方程中有多个项（例如，Y=step(W[i]X[i]+C)）；索引函数（例如，x[i,t+1]=W[i,j]x[j,t]）；归一化（例如，Y[i]=softmax(X[i])）；其他张量函数（例如，Y[k]=concat(X[i,j])）；备选投影算

子（例如，`max=` 或 `avg` 而不是 `+=`，后者是默认值）；切片（例如，X[4:8]); 和过程附件（预定义或外部定义的函数）。张量逻辑接受 Datalog 语法；用圆括号而不是方括号表示张量意味着它是布尔的。特别是，一个稀疏布尔张量可以更紧凑地写为一组事实。例如，向量 X= 也可以写为 X(1), X(2)，其中 X(0) 和 X(3) 隐式为 0。类似地，将字符串 "Alice loves Bob" 读入矩阵 M 会产生事实 M(0,Alice), M(1,loves) 和 M(2,Bob)。

作为另一个简单的例子，一个多层感知器可以通过方程实现 x[i,j]=sig(W[i,j,k]x[i−1,k]), 其中 i 遍历层，j 和 k 遍历单元，`sig()` 是 sigmoid 函数。不同层的大小可以不同（相应的权重矩阵会隐式地用零填充以构成完整的张量）。或者，我们可以为每一层使用不同的方程。

一个基本的递归神经网络（RNN）可以实现为 x[i,∗t+1]=sig(W[i,j]x[j,∗t]+V[i,j]U[j,t]) 其中 X 是状态，U 是输入，i 和 j 遍历单元，t 遍历时间步。`*t` 符号表示 t 是一个虚拟索引：没有为它分配内存，`x[i]` 向量的连续值被写入相同的位置。由于 RNN 是图灵完备的（Siegelmann and Sontag, 1995），上述实现意味着张量逻辑也是图灵完备的。

解读

在定义了基本操作之后，作者现在向我们展示了张量逻辑作为一种编程语言的实际面貌。这一部分的核心是展示其**极简主义**和**强大的表达能力**。

- **唯一的构造：张量方程**。这是张量逻辑最引人注目的特点。没有 `if-else`，没有 `for` 循环，没有函数定义，只有方程。整个程序就是一系列的数学声明。
    - `Y = step(W[i]X[i])`：这是一个完整的程序，用于计算一个感知器。这里的语法非常巧妙。`W[i]X[i]` 意味着在索引 `i` 上进行连接（乘法）。因为 `i` 没有出现在左侧的 `Y` 中，所以它被自动投影（求和）。这行代码同时表达了点积和非线性激活，极其简洁。
    - `Y[i] = step(W[i]X[i])`：这个例子展示了语法的灵活性。因为 `i` 现在出现在了左侧，所以它**不会**被求和。这个方程计算的是两个向量的逐元素乘积。这种“由左侧决定右侧如何计算”的规则，是 `einsum` 思想的自然延伸，优雅且强大。
- **默认规则和语法糖**：
    - **默认为0**：这直接继承自稀疏张量的思想。你只需要定义那些“存在”或“有值”的元素，其他一切都默认为不存在。这让程序非常干净。
    - **隐式求和**：具有相同左侧的方程会自动相加。这对应于逻辑编程中，多条规则可以推导出同一个结论。例如，你可以有两条规则来定义“祖先”，一条是基于“父母”，另一条是基于“祖父母”。在张量逻辑中，你只需写两个具有相同左侧的方程，它们的结果会自动合并。
    - **Datalog 语法兼容**：用圆括号 `()` 表示布尔张量（即逻辑关系），这使得任何合法的 Datalog 程序也都是一个合法的张量逻辑程序。这是一个绝妙的设计，确保了对传统逻辑 AI 的向后兼容性。

- **表达能力展示**：
    - **多层感知器 (MLP)**：`x[i,j] = sig(W[i,j,k]x[i-1,k])` 这一行代码就定义了一个完整的、任意深度的神经网络。`i` 代表层数，`j` 和 `k` 代表神经元。`W[i,j,k]` 是一个三阶权重张量，连接了第 `i-1` 层的 `k` 神经元和第 `i` 层的 `j` 神经元。这个方程递归地定义了每一层的输出是如何由前一层的输出计算得来的。
    - **递归神经网络 (RNN)**：`x[i,*t+1] =...` 这个方程展示了如何处理序列数据。`*t` 这个“虚拟索引”的设计非常聪明，它表示“时间”这个维度并不真的在内存中展开，而是通过迭代计算来处理。这表明张量逻辑不仅能定义静态结构，还能定义动态的计算过程。
    - **图灵完备性**：通过证明能实现 RNN，作者顺便证明了张量逻辑是图灵完备的。这意味着，从理论上讲，任何可计算的问题都可以用张量逻辑来解决。它不仅仅是一个 AI 专用工具，而是一种通用的计算语言。

总而言之，这一节展示了张量逻辑如何通过一个极其简单的核心（张量方程），构建出一个既能描述逻辑规则又能描述复杂神经网络的、图灵完备的编程语言。它的设计哲学是“声明式”而非“过程式”的——你只需“声明”变量之间的数学关系，而不用关心具体的计算步骤。这种高度的抽象正是其力量的源泉。

3.2 推理

张量逻辑中的推理是通过前向链和后向链的张量泛化来进行的。

在前向链中，一个张量逻辑程序被视为线性代码。张量方程依次执行，每个方程计算那些必要输入已可用的张量元素；这个过程重复进行，直到没有新的元素可以计算或满足停止标准为止。

在后向链中，每个张量方程被视为一个函数。查询是顶层调用，每个方程调用其 RHS 上的张量的方程，直到所有相关元素在数据中可用，或者没有用于子查询的方程。在后一种情况下，（子）查询元素默认赋值为 0。

选择使用前向链还是后向链取决于应用。

解读

这一节将逻辑编程中的推理算法——前向链和后向链——推广到了张量逻辑的领域。这两种推理方式现在可以应用于任何张量逻辑程序，无论是纯逻辑的、纯神经网络的，还是两者的混合体。

- **前向链 (Forward Chaining)**：可以理解为“**模拟**”或“**正向计算**”。
    - **工作方式**：它就像一个数据流系统。从输入数据（相当于“事实”）开始，程序会检查哪些方程的右侧（输入）已经准备就绪。一旦某个方程的输入都齐了，就计算它的左侧（输出）。这个新计算出的输出又可能成为其他方程的输入，如此循环往复，直到整个系统达到一个稳定状态（没有新的东西可以计算了）。
    - **应用场景**：这非常适合于神经网络的计算。输入一张图片，第一层的计算准备就绪，计算出第一层的激活值；然后第二层的输入准备就绪，计算第二层的激活值……一层层地向前传播，直到得到最终的输出。对于逻辑系统，它对应于从已知事实出发，推导出所有可能的结论。
- **后向链 (Backward Chaining)**：可以理解为“**求解**”或“**逆向推导**”。
    - **工作方式**：它从一个目标（查询）开始。比如，你想知道 `Y` 的值。程序会找到定义 `Y` 的方程，比如 `Y = f(A, B)`。然后，它会把计算 `A` 和 `B` 的值作为新的子目标。它会递归地去寻找定义 `A` 和 `B` 的方程，直到最终追溯到已知的输入数据，或者发现某个子目标无法计算（此时默认为 0）。
    - **应用场景**：这非常适合于逻辑查询。比如问“苏格拉底会死吗？”。系统会反向查找规则，直到找到基本事实。在张量逻辑的框架下，这意味着可以进行更复杂的、目标驱动的计算。例如，在一个大型模型中，你可能只关心某个特定输出神经元的值，后向链可以只计算与这个输出相关的路径，而无需计算整个网络，从而可能节省大量计算。

通过提供这两种推理模式，张量逻辑获得了一种在传统深度学习框架中不常见的灵活性。开发者可以根据任务的性质（是需要从输入推导出所有结果，还是需要回答一个特定的问题）来选择最高效的计算策略。这进一步模糊了“执行一个神经网络”和“进行一次逻辑查询”之间的界限。

3.3 学习

因为张量逻辑中只有一种类型的语句——张量方程——所以自动微分一个张量逻辑程序变得特别简单。撇开单变量非线性不谈，一个张量方程的 LHS 对其 RHS 上的一个张量的导数，仅仅是 RHS 上其他张量的乘积。更准确地说，如果 Y[...]=T[...]x1[...]...xn[...] 那么 ∂T[...]∂Y[...]=x1[...]...xn[...] 这个的特例包括：如果 Y=AX，那么 ∂Y/∂X=A；如果 Y=W[i]X[i]，那么 ∂Y/∂W[i]=X[i]；以及如果 Y[i,j]=M[i,k]X[k,j]，那么 ∂Y[i,j]/∂M[i,k]=X[k,j]。

因此，一个张量逻辑程序的梯度也是一个张量逻辑程序，每个方程及其 RHS 上的每个张量对应一个方程。为简洁起见省略索引，损失 L 对张量 T 的导数为 ∂T∂L=∑EdYdLdUdY ∏Y\TX, 其中 E 是 T 出现在其 RHS 中的方程，Y 是方程的 LHS，U 是其非线性函数的参数，X 是 U 中的张量。

学习一个张量逻辑程序需要通过一个或多个张量方程来指定损失函数及其应用的张量。例如，为了通过最小化最后一层输出的平方损失来学习一个 MLP，我们可以使用方程 `Loss` = (Y[e]−X[∗e,N,j])2 其中 e 遍历训练样本，j 遍历单元，Y 包含目标值，X 是如上定义的

MLP，并扩展了一个用于样本的虚拟索引，N 是层数。默认情况下，所有未作为训练数据提供的张量都将被学习，但用户可以指定某些张量保持不变（例如，超参数）。优化器本身可以用张量逻辑编码，但通常会使用预先提供的一个。

虽然传统神经网络中的反向传播对所有训练样本都应用于相同的架构，但在张量逻辑中，结构可能会因样本而异，因为不同的方程可能适用于不同的样本，而通过样本所有可能推导的并集进行反向传播将是浪费的。幸运的是，这个问题已经有了一个解决方案，即通过结构进行反向传播（backpropagation through structure），它对每个样本，在其推导中每出现一次方程，就更新一次该方程的参数（Goller and Küchler, 1996）。将此应用于 RNN 产生了反向传播通过时间（backpropagation through time）的特例（Werbos, 1990）。

学习一个由固定方程集组成的张量逻辑程序是相当灵活的，因为一个方程可以代表任何具有相同连接结构的规则集。（例如，一个 MLP 可以代表任何命题规则集。）此外，张量逻辑中的张量分解实际上是谓词发明的泛化。例如，如果要学习的程序是方程 A[i,j,k]=M[i,p]M′[j,q]M″[k,r]C[p,q,r] 并且 A 是唯一的数据张量，那么学习到的 M, M′, M″ 和 C 构成了 A 的塔克分解；将它们阈值化为布尔值就将它们变成了发明的谓词。

解读

这一节是张量逻辑真正展现其魔力的地方，它揭示了学习（即神经网络的训练）在这个框架下是多么自然和优雅。

核心洞见：求导法则的惊人简洁性。

- 在微积分中，我们知道乘法法则 `(uv)' = u'v + uv'`。在张量逻辑中，由于其纯粹的乘法结构，求导法则变得极其简单。如果一个输出 `Y` 是由多个张量 `T, X1, X2,...` 相乘得到的，那么 `Y` 对其中任何一个张量 `T` 的偏导数，就等于所有其他张量 `X1, X2,...` 的乘积。
- 这个性质是革命性的。它意味着，计算梯度（学习过程的核心）的规则与进行前向推理的规则具有完全相同的形式！**一个张量逻辑程序的梯度本身就是另一个张量逻辑程序。**

对比传统框架：

- 在 PyTorch 或 TensorFlow 中，开发者定义一个“前向传播”（forward pass）的计算过程。框架会在后台自动构建一个计算图，然后根据链式法则来执行“反向传播”（backward pass）以计算梯度。这个反向传播的过程对于开发者来说是隐藏的、自动的。
- 在张量逻辑中，“反向传播”不再是一个神秘的后台过程。它就是另一个可以用张量逻辑语言明确写出来的程序。这意味着“推理”和“学习”不再是两个分离的概念，而是同一个计算框架下的两种不同操作，它们共享同一种语言和结构。这种深刻的统一性是前所未有的。

学习过程的定义：

- 学习一个模型，只需要再增加一个定义“损失函数”（Loss）的张量方程。例如，`Loss = (Y_true - Y_pred)^2`。
- 然后，系统可以自动地、符号化地计算出 `Loss` 对模型中所有可学习参数（比如权重张量 `W`）的导数。这些导数本身也是张量方程，可以被张量逻辑的推理引擎执行，从而更新参数。

更深层次的联系：

- **通过结构反向传播 (Backpropagation through structure)**：这个概念解决了当计算路径（即使用的规则）因输入而异时如何学习的问题。这在纯逻辑或混合系统中非常常见。它的思想很简单：一个参数的更新幅度，应该正比于它在“导致”当前输出的推理路径中被使用的次数。这使得张量逻辑能够学习那些结构不固定、更像推理过程的模型，而不仅仅是像传统神经网络那样结构固定的模型。
- **张量分解即谓词发明 (Tensor decomposition as predicate invention)**：这是符号 AI 和连接主义 AI 融合的又一个绝佳例证。
  - **谓词发明**是符号 AI 的一个“圣杯”：让机器自动发现新的、有用的概念。
  - **张量分解**是数学上将一个大张量分解为几个小张量和核心张量的过程。
  - 作者指出，学习一个张量分解模型的过程，实际上就是在进行谓词发明。数据张量 `A` 是观察到的事实，而学习到的因子矩阵 `M` 和核心张量 `C` 就是机器“发明”出来的、能够很好地解释这些事实的“潜在概念”或“隐藏谓词”。如果将这些学习到的数值张量进行二值化（大于某个阈值就为1，否则为0），你就得到了新的、离散的逻辑谓词。这为实现真正意义上的知识发现提供了一条具体的、可计算的路径。

4. 实现 AI 范式

除非另有说明，以下实现均使用前向链。

4.1 神经网络

一个卷积神经网络是一个带有卷积层和池化层的 MLP（LeCun et al., 1998）。卷积层在图像的每个位置应用一个滤波器，可以通过一个形如 `Features[x, y] = relu(Filter[dx, dy, ch] * Image[x+dx, y+dy, ch])` 的张量方程来实现，其中 x 和 y 是像素坐标，dx 和 dy 是滤波器坐标，ch 是 RGB 通道。

池化层将一块邻近的滤波器组合成一个，可以通过 `Pooled = Features[x, y]` 来实现，其中 / 是整数除法，S 是步长。这导致每个维度上 S 个连续位置的滤波器输出被求和成一个。这实现的是和池化；最大池化会将 `=` 替换为 `max=`，等等。一个卷积层和池化层可以合并为一个方程 `Pooled = relu(...)`。

图神经网络（GNNs）将深度学习应用于图结构数据（例如，社交网络、分子、代谢网络、万维网）（Zhou, 2022）。表 1 显示了一个简单 GNN 的实现。网络的图结构由 `Neig(x, y)` 关系定义，每个相邻的 (x, y) 对都有一个事实；或者等价地，由布尔张量 `Neig[x, y] = 1`（如果 x 和 y 相邻）和 0（否则）定义。主张量是 `Emb[n, l, d]`，包含每个节点 n 在每一层 l 的 d 维嵌入。初始化将每个节点的第 0 层嵌入设置为其特征（外部定义或学习的）。网络然后执行 L 轮消息传递，每层一轮。每次迭代开始时，对每个节点应用一个或多个感知器层。（表 1 显示了一个。为了保持排列不变性，权重 W 不依赖于节点。虽然张量逻辑中没有上/下标，但我在这里为了简洁而使用它们。）GNN 然后通过连接张量 `Neig(n, n')` 和 `Z[n', l, d]` 来聚合每个节点的邻居的新特征 Z。对于每个节点，这会将所有非邻居的贡献清零；结果是邻居特征的总和。（在内部，这可以通过迭代节点的邻居或其他方法高效地完成；见第 6 节。）聚合的特征然后可以传递给另一个 MLP（未显示），之后它们与节点的特征使用权重 WAgg 和 WSelf 组合，以产生下一层的嵌入。

GNN 最常见的应用是节点分类、边预测和图分类。对于二分类问题，每个节点通过其最终嵌入与一个权重向量的点积进行分类，并将结果通过 sigmoid 函数得到类别概率。对于多分类问题（未显示），每个节点的最终嵌入与每个类别 c 的权重向量 WOut[c,d] 进行点积，得到一个 logits 向量，然后通过 softmax 函数得到类别概率 Y[n,c]。边预测通过对节点的嵌入进行点积并将结果通过 sigmoid 函数来预测每对节点之间是否存在边。图分类为整个图产生一个类别预测，与节点分类相同，只是结果是一个标量 Y 而不是一个向量 Y[n]。

解读

这一节是张量逻辑的“实力展示”。作者通过一系列简短而优雅的代码，证明了这种新语言能够轻松地实现当今人工智能领域最重要、最复杂的模型。这就像一位语言学家发明了一种新语言后，用它写出了媲美莎士比亚的十四行诗和媲美荷马的史诗，以证明其表达能力。

- **卷积神经网络 (CNN)**：
    - `Features[x, y] = relu(Filter[dx, dy, ch] * Image[x+dx, y+dy, ch])`
    - 这行代码完美地捕捉了卷积的本质。`Filter` 是一个小的权重张量（卷积核），`Image` 是输入图像。`Image[x+dx, y+dy, ch]` 表示以 `(x, y)` 为中心取一个与滤波器大小相同的“图像块”。`Filter` 和这个“图像块”的乘积再求和（因为 `dx, dy, ch` 没有出现在左侧，所以被自动投影），就完成了一次卷积操作。整个方程意味着在图像的每个 `(x, y)` 位置都执行这个操作，生成一个特征图 `Features`。
    - 池化操作 `Pooled = Features[x, y]` 也同样巧妙。`x/S` (整数除法) 的意思是，`x` 坐标在 `0` 到 `S-1` 范围内的所有 `Features` 元素，都会被累加到 `Pooled` 的第 `0` 个位置。这自然地实现了将一个 `S x S` 的区域“池化”成一个值的操作。

- **图神经网络 (GNN)**：
    - GNN 的核心思想是，一个节点（比如社交网络中的你）的特征应该由其邻居（你的朋友们）的特征来决定。张量逻辑的实现清晰地展示了这个过程。
    - `Neig(x, y)` 是一个布尔张量（邻接矩阵），定义了图的结构。
    - **聚合 (Aggregation)** 步骤：`Agg[n, l, d] = Neig(n, n') * Z[n', l, d]`。这行代码是 GNN 的精髓。对于一个特定的节点 `n`，`Neig(n, n')` 只有在 `n'` 是 `n` 的邻居时才为 1。所以，这个乘法操作的效果是“过滤”出 `n` 的所有邻居的特征 `Z[n',...]`，然后将它们相加（因为 `n'` 被投影掉了）。这就完成了从邻居那里“收集信息”的过程。
    - **更新 (Update)** 步骤：`Emb[n, l+1, d] =...` 这一步将聚合来的邻居信息 (`Agg`) 和节点自身上一层的信息 (`Emb[n, l, d]`) 结合起来，生成该节点在新一层 `l+1` 的表示（嵌入）。
    - 整个 GNN 的计算过程，被分解为一系列清晰的、基于图结构的张量运算，展示了张量逻辑处理非欧几里得结构数据的能力。

这些例子表明，张量逻辑不仅仅是一种理论构造，更是一种强大的实践工具。它能用极其接近数学直觉的方式，简洁地表达复杂的模型架构。

表 1：张量逻辑中的图神经网络

下表展示了图神经网络在张量逻辑中的实现：

组件	方程
图结构	Neig(x,y)
初始化	Emb[n,0,d]=X[n,d]
MLP	Z[n,l,d′]=relu(WP[l,d′,d]Emb[n,l,d]), etc.
聚合	Agg[n,l,d]=Neig(n,n′)Z[n′,l,d]
更新	Emb[n,l+1,d]=relu(WAggAgg[n,l,d]+WSelfEmb[n,l,d])
节点分类	Y[n]=sig(WOut[d]Emb[n,L,d])
边预测	Y[n,n′]=sig(Emb[n,L,d]Emb[n′,L,d])
图分类	Y=sig(WOut[d]Emb[n,L,d])

解读

这张表格是理解 GNN 在张量逻辑中如何工作的“速查手册”。它将一个完整的 GNN 模型分解为几个核心组件，并为每个组件提供了对应的张量逻辑方程。

- **图结构 (Graph structure)**：`Neig(x,y)` 是一切的基础。它是一个“事实”集合，告诉我们图中哪些节点是相连的。这可以是朋友关系、分子键或者网页链接。
- **初始化 (Initialization)**：`Emb[n,0,d] = X[n,d]`。`Emb` 代表“嵌入(Embedding)”，是每个节点在模型中的向量表示。在第 0 层（即开始时），每个节点的嵌入就是它的初始特征 `X`。这可能是一个用户的年龄、性别，或者一个原子在元素周期表中的属性。
- **MLP**：`Z[n,l,d'] =...`。在聚合邻居信息之前，通常会先对每个节点自身的嵌入进行一次非线性变换（比如通过一个小的神经网络），以提取更高级的特征。`W_P` 是这个变换的权重。
- **聚合 (Aggregation)**：`Agg[n,l,d] = Neig(n,n') * Z[n',l,d]`。这是 GNN 的核心。如前所述，它计算了节点 `n` 的所有邻居 `n'` 的特征 `Z` 的总和。
- **更新 (Update)**：`Emb[n,l+1,d] =...`。这一步决定了节点 `n` 在下一层的嵌入。它通常是节点自身旧嵌入和邻居聚合信息的一个加权组合。`W_Agg` 和 `W_Self` 是学习到的权重，决定了节点应该更“相信”邻居的意见，还是更“坚持”自己的看法。这个过程会重复 L 次（L 是 GNN 的层数），让信息在图上充分传播。
- **下游任务 (Node/Edge/Graph classification)**：在 L 层的信息传播之后，每个节点都有了一个富含其邻域结构信息的最终嵌入 `Emb[n,L,d]`。
    - **节点分类**：用这个最终嵌入来预测节点的标签（比如一个用户是否是机器人）。
    - **边预测**：将两个节点的最终嵌入结合起来，预测它们之间是否存在一条边（比如预测两个人是否可能成为朋友）。
    - **图分类**：将图中所有节点的最终嵌入聚合起来（这里是通过求和），来预测整个图的属性（比如一个分子是否具有毒性）。

这张表清晰地展示了张量逻辑如何将一个复杂的算法流程，表达为一组声明式的、可组合的数学方程。

注意力，作为大型语言模型的基础，在张量逻辑中也很容易实现（Vaswani et al., 2017）。给定一个嵌入矩阵 X[p,d]，其中 p 遍历项目（例如，文本中的位置），d 遍历嵌入维度，查询、键和值矩阵通过将 X 乘以相应的权重矩阵获得： Query[p,dk]=WQ[dk ,d]X[p,d] Key[p,dk]=WK[dk,d]X[p,d] Val[p,dv]=WV[dv,d]X[p,d]

然后，注意力可以分两步计算，第一步将每个位置的查询与每个键进行比较： Comp[p,p′.]=softmax(Query[p,dk]Key[p′,dk]/sqrt(Dk)), 其中 `sqrt(D_k)` 用键维度的平方根来缩放点积。符号 p′. 表示 p′ 是要被归一化的索引（即，对于每个 p，softmax 应用于由 p′ 索引的向量）。然后，注意力头返回由相应比较加权的值向量的总和： Attn[p,dv]=Comp[p,p′]Val[p′,dv].

我们现在可以用仅仅十几个张量方程实现一个完整的变换器（表 2）。正如我们在 3.1 小节中看到的，一段文本可以由关系 X(p,t) 表示，该关系陈述了文本的第 p 个位置包含第 t 个词元。（词元化规则很容易用 Datalog 表达，此处未显示。）然后，通过将 X(p,t) 与嵌入

矩阵 Emb[t,d] 相乘，获得文本的嵌入 EmbX[p,d]。下一个方程实现了原始论文中的位置编码（Vaswani et al., 2017）；其他选项也是可能的。（顺便说一句，这个方程也显示了条件和 case 语句如何在张量逻辑中实现：通过将每个表达式与相应的条件连接。）然后，残差流被初始化为文本嵌入和位置编码的总和。

注意力如上所述实现，每个张量有两个额外的索引：b 用于注意力块，h 用于注意力头。然后，注意力头的输出被连接起来，加到残差流中，并进行层归一化。MLP 层的实现与之前一样，带有用于块和位置的额外索引，它们的输出也被归一化并加到流中（未显示）。最后，通过将流与每个词元的输出权重向量进行点积，并通过 softmax，获得输出（词元概率）。

解读

这是本节乃至整篇论文的“高光时刻”。作者展示了如何用张量逻辑来实现 Transformer，这个驱动着 ChatGPT 等大型语言模型的革命性架构。令人震惊的是，这个极其复杂的模型可以用区区十几个方程就清晰地定义出来。

让我们来解读这个过程的核心——**自注意力机制 (Self-Attention)**。

- **基本思想**：一句话中的每个词，为了更好地理解自身的含义，需要“关注”句子中的其他词。例如，在句子“河边的银行倒闭了”中，“银行”这个词需要关注“河边”来确定它指的是地理概念，而不是金融机构。
- **Query, Key, Value (Q, K, V)**：这是注意力的核心比喻。对于句子中的每个词，我们都通过乘以不同的权重矩阵，生成三个不同的向量：
  - **Query (查询)**：代表“我正在寻找什么样的信息？”。
  - **Key (键)**：代表“我能提供什么样的信息？”。
  - **Value (值)**：代表“如果我的信息被认为相关，我将提供这个具体内容。”
- **计算过程**：
  1. **计算相关性分数**：`Comp[p, p'] = softmax(Query[p] * Key[p'])`。为了确定第 `p` 个词应该多大程度上关注第 `p'` 个词，我们将第 `p` 个词的 `Query` 向量与第 `p'` 个词的 `Key` 向量进行点积。这个点积结果就衡量了它们之间的“相关性”。然后用 `softmax` 函数将一个词对所有其他词的相关性分数转换成一个总和为 1 的权重分布（注意力权重）。
  2. **加权求和**：`Attn[p] = Comp[p, p'] * Val[p']`。第 `p` 个词的最终输出，是句子中所有词的 `Value` 向量的加权平均。而这个权重，正是上一步计算出的注意力权重 `Comp[p, p']`。这意味着，与 `p` 相关性越高的词，其 `Value` 对 `p` 的最终输出贡献越大。

张量逻辑的实现完美地反映了这个过程：

- 生成 Q, K, V 的三个方程就是简单的矩阵乘法。

- `Comp` 方程用一个 `einsum` (`Query[p,dk]Key[p',dk]`) 计算了所有词对之间的点积。`p'.` 符号优雅地表示 `softmax` 是沿着 `p'` 维度进行的。
- `Attn` 方程则执行了最后的加权求和。

整个 Transformer 的其余部分，如多头注意力（通过增加 `h` 索引）、多层堆叠（通过增加 `b` 索引）、残差连接和层归一化，也都被简洁地表达为张量方程。这雄辩地证明了张量逻辑的表达能力，它能将一个原本需要数百行 Python 代码和复杂类定义的架构，提炼为一组清晰的、声明式的数学关系。

表 2：张量逻辑中的变换器

下表展示了变换器在张量逻辑中的实现：

组件	方程
输入	X(p,t)
嵌入	EmbX[p,d]=X(p,t)Emb[t,d]
位置编码	PosEnc[p,d]=Even(d)sin(p/Ld/De)+Odd(d)cos(p/L(d−1)/De)
残差流	Stream[0,p,d]=EmbX[p,d]+PosEnc[p,d]
注意力	Query[b,h,p,dk]=WQ[b,h,dk,d]Stream[b,p,d], etc.
	Comp[b,h,p,p′.]=softmax(Query[b,h,p,dk]Key[b,h,p′,dk]/sqrt(Dk))
	Attn[b,h,p,dv]=Comp[b,h,p,p′]Val[b,h,p′,dv]
合并与层归一化	Merge[b,p,dm]=concat(Attn[b,h,p,dv])
	Stream[b,p,d.]=lnorm(WS[b,d,dm]Merge[b,p,dm]+Stream[b,p,d])
MLP	MLP[b,p]=relu(WP[p,d]Stream[b,p,d]), etc.
输出	Y[p,t.]=softmax(WO[t,d]Stream)

解读

这张表是 Transformer 架构的“蓝图”，以张量逻辑的语言绘制。它将这个复杂的模型分解为一系列逻辑清晰的步骤。

- **输入 (Input) & 嵌入 (Embedding)**：输入 `X(p, t)` 是一个稀疏的布尔关系，表示“在位置 `p` 的词是 `t`”。`EmbX` 方程通过查找词嵌入矩阵 `Emb`，将这些离散的词转换成连续的向量表示。

- **位置编码 (Pos. encoding)**：Transformer 的一个问题是它本身无法感知词的顺序。位置编码通过给每个位置 `p` 的嵌入向量加上一个独特的、基于 `sin` 和 `cos` 函数的“位置信号” `PosEnc` 来解决这个问题。这个方程展示了张量逻辑如何通过与布尔条件 (`Even(d)`, `Odd(d)`) 相乘来实现 `if-else` 逻辑。

- **残差流 (Residual stream)**：`Stream` 是贯穿整个模型的核心数据流。它初始化为词嵌入和位置编码的和。后续的每个模块（注意力和 MLP）的输出都会被“加”回这个流中，这被称为残差连接，是深度学习中的一个关键技巧，有助于训练更深的模型。

- **注意力 (Attention)**：这是模型的核心计算，如前所述，分为计算 Q, K, V，计算注意力权重 `Comp`，以及加权求和得到 `Attn`。注意这里的额外索引 `b` (block/层) 和 `h` (head/头)，表明这是一个多层、多头的注意力机制。

- **合并与层归一化 (Merge and layer norm)**：多头注意力的输出 `Attn` 被连接 (`concat`) 起来，并通过一个线性变换 (`W_S`)，然后加回到残差流 `Stream` 中。`lnorm` 代表层归一化，是另一个稳定训练的关键技术。

- **MLP**：每个注意力层之后通常会跟一个简单的前馈神经网络（MLP），对 `Stream` 中的每个位置的表示进行进一步的非线性变换。

- **输出 (Output)**：经过 B 个块的处理后，最终的 `Stream` 包含了每个位置丰富的上下文信息。最后一步，通过乘以一个输出权重矩阵 `W_O` 并应用 `softmax`，将这些表示转换成词汇表中每个词的概率分布 `Y`，从而可以预测下一个词。

这张表以惊人的简洁性，描绘了现代最强大 AI 模型之一的完整计算流程，有力地证明了张量逻辑作为一种统一 AI 语言的潜力。

4.2 符号 AI

一个 Datalog 程序是一个有效的张量逻辑程序。因此，任何可以用 Datalog 完成的事情都可以在张量逻辑中完成。这足以实现许多符号系统，包括在无函数领域中的推理和规划。要容纳函数（如 Prolog 中），需要在张量逻辑中实现合一（unification）（Lloyd, 1987）。

解读

这一节的内容虽然简短，但意义重大。它正式确认了张量逻辑对传统符号 AI 的**完全兼容性**。

- **Datalog 的直接嵌入**：作者在前文已经论证，任何 Datalog 程序（由事实和规则组成）都可以直接被看作是一个操作稀疏布尔张量的张量逻辑程序。这意味着，所有基于 Datalog 的经典 AI 应用，例如专家系统、演绎数据库、以及某些类型的规划问题，都可以“开箱即用”地在张量逻辑框架下运行，无需任何修改。这为张量逻辑提供了一个坚实的“基本盘”，确保了它至少和传统逻辑编程一样强大。

- **超越 Datalog：合一 (Unification)**：作者也指出了 Datalog 的局限性。Datalog 主要处理常量和变量，但不支持复杂的、带有函数的结构化术语，而这是 Prolog 等更强大逻辑语言的核心。例如，在 Prolog 中，你可以处理像 `father(John)` 这样的术语。
    - **合一 (Unification)** 是使 Prolog 如此强大的核心算法。它是一个模式匹配过程，可以找到两个包含变量的复杂术语是否可以被“等同”。例如，`father(X)` 和 `father(John)` 可以通过将变量 `X` 绑定到 `John` 来合一。
    - 作者在这里指出，为了让张量逻辑能够支持 Prolog 的全部功能，需要在其框架内实现“合一”算法。这虽然是一个挑战，但作者暗示这是可行的。一旦实现，张量逻辑将不仅能覆盖 Datalog，还能覆盖更广泛的、图灵完备的逻辑编程范式。

简而言之，这一节宣告了张量逻辑在符号 AI 领域的“血统纯正”。它不仅继承了 Datalog 的全部能力，还指明了通往更强大逻辑推理（如 Prolog）的道路，确保了符号 AI 的丰富遗产能够在这个新框架中得到保留和发展。

4.3 核机器

一个核机器可以通过方程实现 Y[Q]=f(A[i]Y[i]K[Q,i]+B), 其中 Q 是查询样本，i 遍历支持向量，f() 是输出非线性函数（例如，sigmoid）（Schölkopf and Smola, 2002）。核 K 则由其自己的方程实现。例如，一个多项式核是 K[i,i′]=(X[i,j]x[i′,j])n 其中 i 和 i′ 遍历样本，j 遍历特征，n 是多项式的次数。一个高斯核是 K[i,i′]=exp(−(X[i,j]−X[i′,j])2/Var)（更准确地说，K 是核关于样本的格拉姆矩阵。）结构化预测，即输出由多个相互关联的元素组成（Bakr et al., 2007），可以通过一个输出向量 Y[Q,k] 和陈述输出之间以及输出与输入之间相互作用的方程来实现。

解读

在展示了对神经网络和符号 AI 的支持后，作者现在将目光投向了机器学习的另一个重要分支——**核机器 (Kernel Machines)**，其中最著名的代表是支持向量机 (SVM)。

- **核方法的核心思想**：有些数据在原始特征空间中是线性不可分的（比如一个环形分布的数据），但如果将它们映射到一个更高维度的空间，它们可能就变得线性可分了。核技巧的巧妙之处在于，它允许我们在**不显式进行这种高维映射**的情况下，直接计算数据点在高维空间中的点积。这个点积的计算函数就是**核函数 (Kernel Function) K**。

- **张量逻辑的实现**：
    - `Y[Q] = f(A[i]Y[i]K[Q,i] + B)`：这个方程完美地捕捉了 SVM 预测阶段的数学形式。`Q` 是我们要预测的新样本，`i` 遍历的是训练后留下的“支持向量”（最重要的训练样本）。`K[Q,i]` 计算了新样本 `Q` 和每个支持向量 `i` 之间的“相似度”（由核函数定义）。`A[i]` 和 `Y[i]` 是训练得到的权重和标签。整个求和过程就是根据新样本与支持向量的相似度，对支持向量的“影响力”进行加权求和，最后通过一个非线性函数 `f` 得到预测结果。
    - `K[i, i'] =...`：张量逻辑的优雅之处在于，核函数本身也可以用一个张量方程来定义。
        - **多项式核**：`(X[i,j]X[i',j])^n`。这里的 `X[i,j]X[i',j]` 是一个点积，计算了样本 `i` 和 `i'` 在原始特征空间中的相似度。然后取 n 次方，就得到了它们在某个高维多项式特征空间中的点积。
        - **高斯核**：`exp(...)`。这个方程计算了两个样本 `i` 和 `i'` 之间欧氏距离的平方，然后通过一个指数函数将其转换为一个范围在 0 到 1 之间的相似度度量。

通过这几个方程，作者证明了张量逻辑不仅能处理神经网络的“层级结构”和逻辑编程的“规则结构”，还能优雅地表达核机器的“基于相似度的结构”。这进一步展示了其作为一种统一机器学习语言的广泛适用性。

4.4 概率图模型

一个图模型将一组随机变量的联合概率分布表示为因子的归一化乘积， P(X=x)=Z1∏kϕk(x{k}); , 其中每个因子 ϕk 是变量子集 x{k} 的一个非负函数，而 Z=∑x∏kϕk(x{k}) (Koller and Friedman, 2009)。如果每个因子是变量给定其父节点（在某个偏序中的前驱）的条件概率，则该模型是贝叶斯网络，且 Z=1。

表 3 显示了离散图模型中的构造和操作如何直接映射到张量逻辑中的构造和操作。一个因子是一个非负实数值的张量，每个变量一个索引，索引的每个值对应变量的一个值。一个状态 x 的未归一化概率是每个张量中对应于 x 的元素的乘积。因此，一个贝叶斯网络可以用张量逻辑编码，每个变量一个方程，根据其条件概率表（CPT）和父节点的分布来陈述该变量的分布： PX[x]=CPTX[x,par1,...,parn]P1[par1]...Pn[parn].

图模型中的推理是计算边际概率和条件概率，由两种操作的组合构成：边缘化和逐点乘积。在一个因子中边缘化变量子集 Y 会将它们求和消去，留下一个关于剩余变量 X 的因子： ϕ′(X)=∑Yϕ(X,Y). 边缘化就是张量投影。两个关于变量子集 X 和 Y 的势的逐点乘积将它们组合成一个关于 X U Y 的单一势，是相应张量的连接。

每个图模型都可以表示为一个连接树，这是一个因子的树，其中每个因子是原始模型中因子的连接。所有边际和条件查询都可以在树的大小上线性时间内回答，通过相继边缘化因子并将它们与父节点的因子逐点相乘。一个连接树是一个树状的张量逻辑程序，即其中没有张量出现在多于一个 RHS 中。因此，线性时间推理可以通过在该程序上进行后向链来执行。具体来说：可以通过向程序中添加方程 Z=T[...] 来计算配分函数 Z，其中 T[...] 是根因

子方程的 LHS，并查询 Z；可以通过将 E 作为一组事实添加到程序中，查询 Z，然后除以原始的 Z，来计算证据的边际概率 P(E)；而给定证据的查询的条件概率可以计算为 P(Q|E)=P(Q,E)/P(E)。

然而，连接树可能比原始模型大指数倍，需要近似推理。两种最流行的方法是循环置信传播和蒙特卡洛采样。循环置信传播是在代表模型的张量逻辑程序上进行前向链。采样可以通过带有选择性投影的后向链来实现（即，用其项的随机子集替换一个投影）。

解读

最后，作者展示了张量逻辑如何囊括人工智能的另一大支柱——**概率图模型 (Probabilistic Graphical Models, PGM)**。PGM，如贝叶斯网络，是用图来表示一组变量之间概率依赖关系的强大工具。

这一节的核心是一个惊人的“翻译词典”，它将 PGM 的所有核心概念直接翻译成了张量逻辑的语言（见表 3）。

- **因子 (Factor) ↔ 张量 (Tensor)**：在 PGM 中，一个因子 ϕ_k 是一个表格，存储了某些变量组合的“分数”或“势”（非负数）。这在数学上就是一个张量，张量的每个维度对应一个变量。
- **因子乘积 (Product of factors) ↔ 张量连接 (Tensor Join)**：PGM 的联合概率是通过将所有因子相乘得到的。这正对应于张量逻辑中的连接操作（基于共同变量/索引的乘法）。
- **边缘化 (Marginalization) ↔ 张量投影 (Tensor Projection)**：在 PGM 中，如果我们想知道某个变量的概率，需要将联合概率分布中的所有其他变量“求和消去”（边缘化）。这在数学上完全等价于张量投影（对某些索引求和）。

这个完美的对应关系意味着：

1. **任何一个图模型都可以直接写成一个张量逻辑程序。** 一个贝叶斯网络可以被写成一系列张量方程，每个方程定义一个变量的条件概率分布 (CPT) 如何由其父节点的概率分布计算而来。
2. **图模型中的推理算法在张量逻辑中有天然的对应物。**
    - **精确推理 (Exact Inference)**，如在连接树上的推理，可以被看作是在一个树状结构的张量逻辑程序上进行高效的后向链计算。计算 P(Q|E) 这样的条件概率，变成了执行几个张量逻辑查询。
    - **近似推理 (Approximate Inference)**，如置信传播 (Belief Propagation)，被揭示为不过是在代表模型的张量逻辑程序上进行**前向链**计算。而蒙特卡洛采样，则可以被实现为一种特殊的、带随机性的**后向链**。

至此，作者完成了一次“大一统”的壮举。他已经证明，神经网络、符号逻辑、核机器和概率图模型这四大主流 AI 范式，都可以被视为张量逻辑这一种更基础、更通用语言的“特例”。它们看似不同的外表下，共享着同样的核心计算结构：张量上的连接与投影。

表 3：张量逻辑中的图模型

下表展示了图模型在张量逻辑中的实现：

组件	实现
因子	张量
边缘化	投影
逐点乘积	连接
连接树	树状程序
P(Query|Evidence)	Prog(Q,E)/Prog(E)
置信传播	前向链
采样	选择性投影

解读

这张表格是理解概率图模型（PGM）与张量逻辑之间对应关系的“罗塞塔石碑”。它清晰地揭示了两种形式体系在概念和操作上的直接等价性。

- **因子 (Factor) ↔ 张量 (Tensor)**：PGM 中的一个因子，比如一个条件概率表 (CPT)，本质上就是一个多维数组，其维度对应于相关的随机变量。这正是张量的定义。
- **边缘化 (Marginalization) ↔ 投影 (Projection)**：边缘化是在概率论中通过对无关变量求和来计算边缘概率的过程。这在操作上与张量投影（对某些索引求和）完全相同。
- **逐点乘积 (Pointwise product) ↔ 连接 (Join)**：组合多个因子以获得更大范围变量的联合分布，是通过逐点乘积实现的。这对应于张量逻辑中的连接操作。
- **连接树 (Join tree) ↔ 树状程序 (Tree-like program)**：连接树是一种用于在 PGM 中进行高效精确推理的数据结构。它本身可以被看作是一个没有环路的、树状的张量逻辑程序，其中每个节点的计算都依赖于其子节点的计算结果。
- **P(Query|Evidence)**：计算条件概率是 PGM 推理的核心任务。在张量逻辑中，这可以通过运行程序两次来完成：一次是同时包含查询 `Q` 和证据 `E` 的事实（得到联合概率 P(Q,E)），另一次是只包含证据 `E` 的事实（得到证据的概率 P(E)），然后将两者相除。
- **置信传播 (Belief propagation) ↔ 前向链 (Forward chaining)**：置信传播是一种流行的近似推理算法，它通过在图的节点间迭代地传递“消息”来工作。作者指出，这个迭代过程可以被看作是在代表 PGM 的张量逻辑程序上执行前向链，直到系统收敛到一个不动点。

- **采样 (Sampling) ↔ 选择性投影 (Selective projection)**：像蒙特卡洛这样的采样方法，是通过从概率分布中抽取样本来近似推理。这可以在张量逻辑中通过一种修改版的投影来实现：不进行完全的求和，而是根据概率随机选择求和项中的一个或几个子集。

这张表有力地证明了张量逻辑不仅仅是“能够实现”PGM，而是 PGM 的一种“原生语言”。PGM 中的所有核心概念和算法，都在张量逻辑中找到了简单、直接的对应物。

5. 在嵌入空间中进行推理

张量逻辑最有趣的特性是它所启发的新模型。在本节中，我将展示如何在嵌入空间中进行知识表示和推理，并指出这种方法的可靠性和透明性。

首先考虑一个物体的嵌入是一个随机单位向量的情况。这些嵌入可以存储在一个矩阵 Emb[x,d] 中，其中 x 遍历物体，d 遍历嵌入维度。将 Emb[x,d] 乘以一个独热向量 V[x]，就可以检索出相应物体的嵌入。如果 V[x] 是一个代表集合的多热向量， S[d]=V[x]Emb[x,d] 就是集合中物体嵌入的叠加。对于某个物体 A，点积 D[A]=S[d]Emb[A,d] 如果 A 在集合中，结果近似为 1，否则近似为 0（标准差为 N/D，其中 N 是集合的基数，D 是嵌入维度）。然后以 1/2 为阈值进行判断，就可以知道 A 是否在集合中，其错误概率随嵌入维度的增加而降低。这类似于布隆过滤器（Bloom, 1970）。

同样的方案可以扩展到嵌入一个关系。为简单起见，考虑一个二元关系 R(x,y)。那么 EmbR[i,j]=R(x,y)Emb[x,i]Emb[y,j] 是关系中元组嵌入的叠加，其中一个元组的嵌入是其参数嵌入的张量积。这是一种张量积表示（Smolensky, 1990）。它可以通过迭代元组，将相应的张量积加到结果中，以线性于 R 的时间计算出来。方程 D=EmbR[i,j]Emb[A,i]Emb 检索 R(A,B)，即如果元组 (A, B) 在关系中，D 近似为 1，否则为 0，因为 D=EmbR[i,j]Emb[A,i]Emb =(R(x,y)Emb[x,i]Emb[y,j])Emb[A,i]Emb =R(x,y) (Emb[x,i]Emb[A,i])(Emb[y,j]Emb) ≃R(A,B). 倒数第二步是有效的，因为 einsum 是可交换和可结合的。（特别是，结果不依赖于张量出现的顺序，只依赖于它们的索引结构。）最后一步是有效的，因为两个随机单位向量的点积近似为 0。

通过同样的推理，方程 D[A,y]=EmbR[i,j]Emb[A,i]Emb[y,j] 返回与 A 有关系 R 的所有物体的嵌入的叠加，而 D[x,y]=EmbR[i,j]Emb[x,i]Emb[y,j] 返回整个关系 R(x,y)。EmbR[i,j]、Emb[x,i] 和 Emb[y,j] 构成了数据张量 D[x,y] 的塔克分解，其中 EmbR[i,j] 是核心张量，Emb[x,i] 和 Emb[y,j] 是因子矩阵。

关系符号本身也可以被嵌入。（例如，R(A,B) 中的 R、A 和 B 都可以被嵌入。）这会产生一个三阶张量。任意元数的关系可以简化为（关系，参数，值）三元组的集合。因此，整个数据库可以被嵌入为一个单一的三阶张量。

下一步是嵌入规则。我们可以通过将其前件和后件替换为它们的嵌入来嵌入一个 Datalog 规则：如果规则是 Cons(...)←Ant1(...),...,Antn(...), 其嵌入是 EmbCons[...]=EmbAnt1[...]...EmbAntn[...], 其中 EmbAnt1[...]=Ant1(...)Emb[...]...Emb[...], 等等。现在，可以通过在嵌入的规则上进行前向或后向链来在嵌入空间中进行推理。查询的答案可以通过将其张量与其参数的嵌入连接来提取，如上所示对任何关系都适用。这给出了近似正确的结果，因为每个推断出的张量都可以表示为嵌入关系连接的投影之和，并

且其每个参数的乘积 Emb[x,i]Emb[x′,i] 近似为单位矩阵。错误概率随嵌入维度的增加而降低，如前所述。为了进一步降低它，我们可以定期提取、阈值化和重新嵌入推断出的张量（在极限情况下，在每次规则应用之后）。

解读

如果说前面的章节证明了张量逻辑是一个强大的“统一者”，那么这一章则展示了它作为一个“创新者”的巨大潜力。这里介绍的思想是全新的、开创性的，它试图解决当前深度学习最核心的两个问题：**缺乏严谨的逻辑推理能力**和**结果的不可靠性（幻觉）**。

核心思想：将整个知识库“溶解”到几何空间中。

- **嵌入 (Embedding)**：首先，为世界上的每一个对象（人、物、概念）分配一个高维空间中的向量（坐标）。这些向量不是随机的，而是通过学习得到的，使得语义上相似的对象在空间中的位置也相近。这就是所谓的“嵌入空间”或“意义空间”。
- **嵌入关系**：一个逻辑关系，比如 `Loves(Romeo, Juliet)`，不再是一个离散的符号陈述。它的嵌入 `EmbLoves` 是通过将 `Romeo` 的嵌入向量和 `Juliet` 的嵌入向量进行**张量积**，然后再与关系 `Loves` 本身的嵌入向量结合而成的。一个知识库中所有的 `Loves` 关系，都被叠加（求和）到同一个表示关系 `Loves` 的张量 `EmbLoves` 中。这样，整个知识库就被“压缩”成了一系列代表不同关系的张量。
- **嵌入规则**：推理规则，如 `Grandparent(x,z) ← Parent(x,y), Parent(y,z)`，也被转换成嵌入空间中的张量方程：`EmbGrandparent = EmbParent * EmbParent`。逻辑上的“连接”操作，现在变成了嵌入张量之间的乘法和求和。

在嵌入空间中推理：

- **查询**：当我们想查询 `Loves(Romeo, Juliet)` 是否为真时，我们不再进行符号匹配。取而代之的是，我们将 `EmbLoves` 张量与 `Romeo` 的嵌入和 `Juliet` 的嵌入进行点积。由于嵌入向量（在理想情况下）是近似正交的，这个计算结果会近似于 1（如果关系存在）或 0（如果关系不存在）。
- **推理链**：在嵌入规则上执行前向链，就相当于在嵌入空间中进行一系列的几何变换。`EmbParent` 乘以自身，生成了 `EmbGrandparent` 张量。这意味着，逻辑推理的过程，被转化为了在“意义空间”中进行张量运算。

为什么这很重要？

- **模糊性与精确性的结合**：传统的符号逻辑是“脆”的，它要求精确匹配。`King` 和 `Emperor` 在它看来是完全不同的两个符号。在嵌入空间中，`King` 和 `Emperor` 的向量非常相似，因此在一个对象上成立的推理，可以在另一个相似的对象上“泛化”或“类比”过去。这赋予了逻辑系统处理模糊性和相似性的能力，这是纯符号系统所不具备的。

- **可解释的结构**：这种方法将一个庞大的、离散的知识图谱，转化为了一个结构化的、连续的数学对象（张量）。作者在这里还指出了一个深刻的联系：对关系数据张量 `D[x,y]` 进行的塔克分解，其核心张量和因子矩阵，正对应于关系 `R` 的嵌入和实体 `x`, `y` 的嵌入。这为我们理解和分析知识的内在结构提供了强大的数学工具。

这一整套方法，旨在构建一个既能利用神经网络强大的表示和泛化能力，又能进行结构化、多步推理的系统。这是通往更高级、更鲁棒的人工智能的关键一步。

然而，最有趣的情况是当物体的嵌入是学习得到的时候。嵌入矩阵与其转置的乘积，Sim[x,x′]=Emb[x,d]Emb[x′,d], 现在是格拉姆矩阵，通过其嵌入的点积来衡量每对物体的相似性。相似的物体相互“借用”推论，权重与其相似性成正比。这导致了一种强大的类比推理形式，它在一个深度架构中明确地结合了相似性和组合性。

如果我们对每个方程应用一个 sigmoid 函数， σ(x,T)=(1+e−x/T)1, 将其温度参数 T 设置为 0，实际上将格拉姆矩阵简化为单位矩阵，使程序的推理变为纯粹的演绎。这与 LLM 形成对比，后者即使在 T=0 时也可能产生幻觉。它也比检索增强生成（Jiang et al., 2025）强大指数倍，因为它有效地检索了事实在规则下的演绎闭包，而不仅仅是事实本身。

增加温度使推理越来越具有类比性，越来越不相似的例子也会相互借用推论。最优的 T 将取决于应用，并且对于不同的规则可以不同（例如，一些规则可能是数学真理，T=0，而其他规则可能用于积累弱证据，具有较高的 T）。

推断出的张量可以在推理过程中的任何点被提取出来。这使得推理高度透明，与基于 LLM 的推理模型形成对比。在足够低的温度下，它也高度可靠且不受幻觉影响，再次与基于 LLM 的模型形成对比。同时，它具有在嵌入空间中推理的泛化和类比能力。这可能使其成为广泛应用的理想选择。

解读

这是本文思想最深刻、最激动人心的部分。作者在这里揭示了张量逻辑如何可能解决当前大型语言模型（LLM）最棘手的两个问题：**幻觉（hallucination）和不透明性（opacity）**，同时保留其强大的泛化能力。

从随机嵌入到学习嵌入：

- 前面的讨论假设嵌入向量是随机的。现在，我们让这些嵌入向量通过数据来**学习**。这意味着模型会自动调整每个对象在“意义空间”中的位置，使得经常在一起出现的、或行为相似的对象，其坐标也变得相近。

- `Sim[x, x'] = Emb[x, d]Emb[x', d]` 这个方程变得至关重要。它计算了任意两个对象 `x` 和 `x'` 嵌入向量的点积。因为嵌入是学习到的，这个点积现在不再是随机的 0 或 1，而是成了一个衡量两者“**语义相似度**”的连续值。`King` 和 `Emperor` 的点积会接近 1，而 `King` 和 `Cabbage` 的点积会接近 0。

温度 T：在逻辑与诗歌之间调节的旋钮 作者引入了一个带“温度”参数 `T` 的 sigmoid 函数，这是一个点睛之笔。这个 `T` 就像一个控制系统思维模式的旋钮：

- **当 T → 0 (低温，逻辑学家模式)**：
    - 此时，sigmoid 函数变成一个完美的阶跃函数。相似度矩阵 `Sim[x, x']` 实际上退化为了一个单位矩阵：只有当 `x` 和 `x'` 完全相同时，结果才为 1，否则为 0。
    - 在这种模式下，推理是**纯粹演绎的**。系统只接受严格的、符号化的匹配。`Grandparent(x,z) ← Parent(x,y), Parent(y,z)` 这条规则只会精确地应用于已知的 `Parent` 关系。
    - **关键优势**：这种推理是**可靠的、无幻觉的**。它的结论是基于其知识库和规则逻辑推导出来的，有据可查。这与 LLM 形成了鲜明对比，LLM 即使在 `T=0`（即采用最可能的输出）时，其输出本质上仍是概率性的猜测，可能与事实相悖。
- **当 T > 0 (高温，诗人/类比模式)**：
    - 此时，sigmoid 函数变得平滑。相似度矩阵 `Sim[x, x']` 会将 `King` 和 `Emperor` 视为部分等同。
    - 推理变成了**类比推理**。如果系统知道“国王有王冠”，它可能会推断出“皇帝（因为与国王相似）也可能有王冠”。相似的对象可以相互“借用”推理结论，借用的强度取决于它们的相似度。
    - 这赋予了系统强大的**泛化能力**，能够处理未见过的情况，这正是深度学习的优势所在。

透明性与超越 RAG：

- **透明性**：在推理过程的任何一步，我们都可以“检查”中间推导出的张量。这就像在解数学题时可以查看每一步的草稿，使得整个推理过程不再是一个黑箱。
- **超越 RAG**：检索增强生成（RAG）是目前减轻 LLM 幻觉的一种流行技术，它通过检索相关事实来辅助生成。作者指出，张量逻辑在 `T=0` 模式下比 RAG 强大得多，因为它检索的不仅仅是原始事实，而是这些事实在所有规则下的“**演绎闭包**”——即所有可以被逻辑推导出来的隐含知识。

这一节描绘了一幅未来 AI 的蓝图：一个可以在严格的逻辑推理和强大的类比泛化之间平滑过渡的系统。它既有符号 AI 的可靠性和透明性，又有神经网络的灵活性和学习能力。这可能是通往真正可信赖、通用人工智能的一条非常有前途的道路。

6. 扩展

对于大规模的学习和推理，涉及密集张量的方程可以直接在 GPU 上实现。稀疏和混合张量的操作可以使用（至少）两种方法之一来实现。

第一种是关注点分离：密集（子）张量的操作在 GPU 上实现，而稀疏（子）张量的操作则通过将（子）张量视为关系，使用数据库查询引擎来实现。然后，查询优化的全部技术都可以应用于组合这些稀疏（子）张量。整个密集子张量可以被数据库引擎视为单个元组，其中一个参数指向该子张量。然后使用 GPU 对密集子张量进行连接和投影。

第二种也是更有趣的方法是，在 GPU 上执行所有操作，首先通过塔克分解将稀疏张量转换为密集张量。这比直接在稀疏张量上操作要高效得多（呈指数级），并且正如我们在上一节中看到的，即使是随机分解也足够了。代价是会有一个小的错误概率，但这可以通过适当设置嵌入维度和通过阶跃函数对结果进行去噪来控制。通过塔克分解进行扩展具有一个显著的优势，即它与前几节中描述的学习和推理算法无缝结合。

解读

在描绘了宏伟的理论蓝图之后，作者转向了一个至关重要且非常实际的问题：**如何让张量逻辑处理真实世界的、海量的数据？** 这一节提出了两种可行的工程路径。

方法一：混合动力系统 (Separation of Concerns)

- **思想**：这是一种“专业分工”的策略。我们知道，现实世界的知识（如社交网络、语言知识图谱）通常是巨大的但**稀疏的**（大部分连接不存在）。而神经网络的权重和激活值则是**密集的**。
- **实现**：
    - 对于稀疏的部分（比如逻辑关系），我们把它交给一个**数据库查询引擎**。数据库经过几十年的发展，已经拥有了极其成熟和高效的稀疏数据处理和查询优化技术。
    - 对于密集的部分（比如嵌入向量、权重矩阵），我们把它交给 **GPU**。GPU 是为大规模并行浮点运算而生的，是处理密集张量的最佳工具。
    - 整个系统就像一个混合动力汽车，CPU（运行数据库）和 GPU 协同工作，各自处理自己最擅长的任务。密集子张量在数据库层面被看作一个“黑箱”指针，当需要对它们进行运算时，再调用 GPU 来执行。

方法二：纯电系统，但有近似 (Tucker Decomposition)

- **思想**：这是一种更激进、也更优雅的策略。它的核心思想是：**将所有稀疏问题都转化为密集问题，然后全部交给 GPU 处理。**

- **实现**：
    - 如何转化？答案是**塔克分解**。正如前面提到的，塔克分解可以将一个巨大的、稀疏的高阶张量，近似地表示为一个小的、密集的**核心张量**和几个**因子矩阵**（它们也是密集的）。
    - 这个过程相当于为原始的稀疏知识学习一个紧凑的、连续的“**嵌入表示**”。
    - **优点**：一旦所有东西都变成了密集张量，我们就可以利用 GPU 的全部威力来进行极其高效的并行计算。更重要的是，这种方法与第五节中“在嵌入空间中推理”的思想完美契合。学习、推理和扩展都统一在同一个“嵌入+密集计算”的框架下。
    - **代价**：这种分解通常是近似的，因此计算结果会引入微小的误差。但作者指出，这个误差是可控的：我们可以通过增加嵌入的维度（即因子矩阵的大小）来降低误差。此外，可以在计算的某些步骤使用阶跃函数来“净化”结果，将接近 1 的值强制设为 1，接近 0 的值设为 0，从而消除累积的噪声。

这两种扩展策略表明，作者不仅考虑了张量逻辑的理论优美性，也为其在现实世界中的大规模应用规划了清晰的路线图。特别是第二种方法，它进一步强化了论文的核心论点：将离散的符号问题转化为连续的几何问题，可能是实现可扩展、可学习的推理系统的关键。

7. 讨论

张量逻辑可能在人工智能之外也很有用。科学计算本质上是将方程翻译成代码，而使用张量逻辑，这种翻译比以前的语言更直接，纸上的符号和代码中的符号常常存在一一对应的关系。在科学计算中，相关方程随后被包装在控制其执行的逻辑语句中。张量逻辑通过将相应的布尔张量放宽为数值张量，并可选择地将结果阈值化回逻辑，使得这种控制结构可以自动学习。同样的方法原则上适用于使任何程序变得可学习。

任何新的编程语言在广泛应用方面都面临着陡峭的攀升。张量逻辑成功的机会有多大？人工智能编程不再是一个小众领域；张量逻辑可以像 Java 驾驭互联网浪潮一样，驾驭人工智能浪潮走向广泛应用。与 Python 的向后兼容性是关键，而张量逻辑很适合这一点：它最初可以作为 einsum 的更优雅实现和 Python 对推理任务的扩展来使用，随着它的发展，它可以吸收越来越多 NumPy、PyTorch 等的特性，直到取代它们。

最重要的是，新语言的采用是由它们治愈的巨大痛点和它们支持的杀手级应用驱动的，而张量逻辑恰恰拥有这些：例如，它可能治愈 LLM 的幻觉和不透明性，并且是推理、数学和编码模型的理想语言。

围绕张量逻辑培养一个开源社区将是重中之重。张量逻辑适合于紧密集成了编码、数据整理、建模和评估的 IDE，如果它能起飞，供应商们将竞相支持它。它也特别适合于人工智能的教学和学习，这是它可以传播的另一个途径。

下一步包括直接在 CUDA 中实现张量逻辑，在广泛的应用中使用它，开发库和扩展，并追求它所带来的新研究方向。

有关张量逻辑的更多信息，请访问 tensor-logic.org。

解读

在论文的最后，作者从更高的视角审视了张量逻辑的潜力和未来。这部分更像是一份宣言和一份战略规划。

超越 AI 的潜力

- 作者认为张量逻辑的价值不仅限于 AI。在**科学计算**领域（如物理模拟、气候建模），研究人员的核心工作就是将复杂的数学方程翻译成代码。张量逻辑的语法与数学方程本身高度一致，可以大大简化这个过程。
- 更具革命性的是，传统科学计算程序中充满了 `if-else` 这样的硬编码逻辑来控制模拟流程。张量逻辑允许将这些布尔逻辑“软化”为连续的数值张量，从而使整个程序的**控制流本身也变得可学习、可优化**。这为“可微分编程”这一前沿领域提供了一个强大的新范式，理论上可以让任何计算机程序都具备从数据中学习和调整自身行为的能力。

成功的战略规划 作者清醒地认识到，推广一门新语言极其困难。他为此制定了一个清晰的“市场进入策略”：

1. **搭上时代的快车**：AI 是当今科技界最大的浪潮，就像 90 年代的互联网。一门为 AI 而生的语言，可以乘着这股浪潮获得巨大的关注度和用户基础。
2. **拥抱现有生态**：与 Python 的兼容性至关重要。张量逻辑不会要求开发者一夜之间抛弃所有熟悉的工具。它可以作为一个 Python 库起步，提供比现有 `einsum` 更强大、更优雅的功能，并逐步扩展，最终实现无缝替代。
3. **解决核心痛点**：一门新语言必须能解决现有工具无法解决的、令人头疼的问题。张量逻辑瞄准的正是当前 AI 领域最大的痛点：LLM 的**幻觉**和**不透明性**。如果它能提供一个可靠的解决方案，那么开发者将有极强的动力去采纳它。
4. **打造杀手级应用**：除了解决痛点，还需要展示其在创造新事物方面的独特优势。作者认为，张量逻辑是构建下一代**推理模型、数学模型和代码生成模型**的理想语言，这些领域都要求极高的逻辑严谨性，是纯粹的 LLM 难以胜任的。
5. **建设社区**：开源是现代软件成功的关键。通过建立一个活跃的社区，开发工具（如 IDE）、库和教程，可以极大地降低入门门槛，加速语言的传播和发展。教育是另一个重要的传播途径，张量逻辑的简洁和直观性使其非常适合作为教授 AI 核心概念的教学语言。

最后，作者给出了明确的行动计划：开发底层的 CUDA 实现以获得极致性能，在各种真实应用中验证其价值，并探索由这一新范式开启的广阔研究前景。这篇论文不仅提出了一个深刻的理论，也为将其变为现实描绘了一幅切实可行的路线图。

致谢

本研究部分由 ONR 拨款 N00014-18-1-2826 资助。

解读

致谢部分虽然简短，但提供了重要的背景信息。ONR 指的是美国海军研究办公室（Office of Naval Research），是美国军方一个重要的基础科学研究资助机构。获得此类机构的资助，通常意味着该研究方向被认为具有长期的、战略性的重要价值，可能在国防、情报分析、自主系统等领域有潜在应用。这从一个侧面印证了这项研究的前沿性和重要性。

参考文献

G. Bakr, T. Hofmann, B. Schölkopf, A. Smola, B. Taskar, and S. Vishwanathan, editors. *Predicting Structured Data*. MIT Press, Cambridge, MA, 2007.

B. Bloom. Space/time tradeoffs in hash coding with allowable errors. *Comm. ACM*, 13: 422-426, 1970.

C. Goller and A. Küchler. Learning task-dependent distributed representations by backpropagation through structure. In *Proc. Int. Conf. Neural Networks*, pp. 347-352, 1996.

S. Greco and C. Molinaro. *Datalog and Logic Databases*. Morgan & Claypool, San Rafael, CA, 2016.

P. Jiang, S. Ouyang, Y. Jiao, M. Zhong, R. Tian, and J. Han. Retrieval and structuring augmented generation with large language models. In *Proc. Int. Conf. Knowl. Disc. & Data Mining*, pp. 6032-6042, 2025.

D. Koller and N. Friedman. *Probabilistic Graphical Models: Principles and Techniques*. MIT Press, Cambridge, MA, 2009.

N. Lavrač and S. Džeroski. *Inductive Logic Programming: Techniques and Applications*. Ellis Horwood, Chichester, UK, 1994.

Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proc. IEEE*, 86:2278-2324, 1998.

J. W. Lloyd. *Foundations of Logic Programming (2nd ed.)*. Springer, Berlin, Germany, 1987.

S. Rabanser, O. Shchur, and S. Günnemann. Introduction to tensor decompositions and their applications in machine learning. arXiv:1711.1078, 2017.

T. Rocktäschel. Einsum is all you need - Einstein summation in deep learning. https://rockt.ai/2018/04/30/einsum, 2018.

A. Rogozhnikov. Einops: Clear and reliable tensor manipulations with Einstein-like notation. In *Proc. Int. Conf. Learn. Repr.*, 2022.

B. Schölkopf and A. J. Smola. *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. MIT Press, Cambridge, MA, 2002.

H. Siegelmann and E. Sontag. On the computational power of neural nets. *J. Comp. & Sys. Sci.*, 50:132-150, 1995.

P. Smolensky. Tensor product variable binding and the representation of symbolic structures in connectionist systems. *Artif. Intel.*, 46:159-216, 1990.

A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. Gomez, L. Kaiser, and I. Polosukhin. Attention is all you need. *Adv. Neural Inf. Proc. Sys.*, 30:5998-6008, 2017.

P. Werbos. Backpropagation through time: What it does and how to do it. *Proc. IEEE*, 78:1550-1560, 1990.

Z. Liu & J. Zhou. *Introduction to Graph Neural Networks*. Morgan & Claypool, San Rafael, CA, 2022.

解读

参考文献列表是一篇学术论文的“知识地图”，它揭示了作者思想的来源和所站立的“巨人肩膀”。对于高阶学生来说，这是一个极好的资源，可以顺藤摸瓜，深入探索相关领域。

这个列表可以分为几个主要类别：

- ** foundational AI Paradigms**:
    - **逻辑编程**: `Lloyd (1987)` 提供了逻辑编程的基础，而 `Greco and Molinaro (2016)` 聚焦于 Datalog。`Lavrač and Džeroski (1994)` 则是关于从数据中学习规则的归纳逻辑编程。
    - **神经网络**: `LeCun et al. (1998)` 是关于卷积神经网络的开创性论文。`Vaswani et al. (2017)` 是著名的 "Attention is All You Need"，Transformer 架构的来源。`Goller and Küchler (1996)` 和 `Werbos (1990)` 探讨了更复杂的反向传播技术，这对于学习结构化和时序模型至关重要。
    - **概率图模型**: `Koller and Friedman (2009)` 是该领域的权威教科书。
    - **核机器**: `Schölkopf and Smola (2002)` 是关于支持向量机和核方法的经典著作。
- **核心数学工具**:

    张量: `Rabanser et al. (2017)` 提供了张量分解的介绍。`Rocktäschel (2018)` 和 `Rogozhnikov (2022)` 的文章强调了 `einsum` 在深度学习中的核心地位。

- **神经符号思想的先驱**:

  `Smolensky (1990)` 的论文是关于使用张量积来表示符号结构的早期重要工作，是本文第五节思想的重要前身。

- **对比的现代技术**:

  `Jiang et al. (2025)` (注意年份是未来的，这是学术论文中的常见做法，表示已接收但未正式出版的会议论文) 描述了检索增强生成（RAG），作者在文中将其作为对比，以凸显张量逻辑的优势。

通过浏览这个列表，我们可以清晰地看到，作者的工作建立在人工智能各个主要分支的坚实基础之上。他并非凭空创造，而是将这些看似不相关的领域的思想和工具融会贯通，最终提炼出了张量逻辑这个统一的框架。对于希望深入研究的学生，从这些经典文献入手，将是一条极佳的学习路径。