利用逻辑归纳偏置学习语言表示

Jianshu Chen腾讯人工智能实验室，美国华盛顿州
贝尔维尤 98004,
jianshuchen@global.tencent.com

arXiv:2302.09458v1 [cs.CL] 19 Feb 2023

摘要

Transformer 架构在解决自然语言任务方面取得了巨大成功，它们从大规模无标注文本中学习强大的语言表示。在本文中，我们进一步探索一种新的逻辑归纳偏置，以改进语言表示学习。逻辑推理是一种形式化方法，能够根据给定的知识和事实得出答案。受这一观点启发，我们开发了一种名为 FOLNet（一**阶逻辑网**络）的新型神经网络架构，以编码这种新的归纳偏置。我们构建了一组作为可学习霍恩子句的神经逻辑算子，并进一步将其前向链接 为全可微神经网络架构（FOLNet）。有趣的是，我们发现 Transformer 中的自注意力模块可以由两个神经逻辑算子组合而成，这或许解释了它们强大的推理能力。我们提出的 FOLNet 与其他预训练模型具有相同的输入和输出接口，因此可以使用类似的损失函数进行预训练或微调。此外，FOLNet 还可以通过即插即用的方式替换其他预训练模型。借助逻辑归纳偏置，我们期望通过预训练学到的“逻辑演绎技能”同一集合能够同样胜任解决各种下游任务。因此，FOLNet 学到的语言表示具有强得多的迁移能力。在若干语言理解任务上的实验结果表明，我们的预训练 FOLNet 模型优于现有的强大 Transformer 架构方法。[1]

1 引言

预训练 Transformer 模型在解决自然语言任务方面取得了巨大成功，它们从大规模无标注文本中学习强大的语言表示。通过在有限数量的标注数据上进行微调，学习到的表示可以轻松迁移到不同的下游任务（Radford 等，2018；Devlin 等人，2018；Lan 等人，2019；Liu 等人，2019；Yang 等人，2019）。随着模型规模进一步扩大，即使不进行微调，它们也展现出强大的零样本或少样本泛化能力（Radford 等，2019；Brown 等，2020；Chowdhery 等，2022）。除了大规模模型和训练数据之外，成功的一个重要原因是 Transformer 架构中编码的强大关系归纳偏置（Vaswani 等，2017）；它能够有效地建模词元之间的成对关系，并利用这些关系计算语言表示。

本文旨在超越 Transformer 模型中的归纳偏置，探索一种新的逻辑归纳偏置，以实现更好的语言表示学习。核心思想是将语言表示的计算视为一个逻辑推理过程；也就是说，通过逻辑推理，从原始离散词元序列逐步演绎出语言表示。具体而言，我们将输入序列中的词元视为逻辑编程中的项，并将其属性和关系视为具有不同元数的谓词。随后，最终的语言表示由基本输入属性和关系（例如词元 ID 和相对距离）推导为高级属性和关系。最重要的是，我们要求此类演绎过程的构建遵循一阶逻辑的原则，以便编码这种逻辑归纳偏置。

遵循上述逻辑归纳偏置，我们推导出一种基于原理构建的神经网络架构，称为 FOLNet（一阶**逻辑网络**），用于学习语言表示。具体而言，我们构建了一组

[1]代码以及预训练模型检查点将公开发布。

神经逻辑算子，并将其作为可学习霍恩子句，再通过前向链式推理 将其整合为全可微神经网络架构。具体而言，FOLNet架构由两个相互作用的分支组成，分别负责一元和二元关系推理。有趣的是，我们发现自注意力机制可以由我们开发的两个神经逻辑算子构建，而整个Transformer架构可以理解为FOLNet的单分支版本。这一新发现的联系或许可以部分解释Transformer架构出人意料的强大推理表现（Wei等，2022；Lewkowycz等，2022）。正如我们将在实验中展示的那样，这种双分支架构具有若干显著优势，对于学习更好的语言表示至关重要。此外，我们还基于逻辑归纳偏置，对不同的位置编码策略建立了新的统一理解。例如，我们发现，现有流行的相对位置编码可以由我们两个神经逻辑算子的退化版本构建。更重要的是，这也使我们能够提出一种新的、基于原理构建的相对位置编码，它简单而且在实践中相当有效。值得注意的是，我们提出的FOLNet与其他预训练Transformer模型（例如BERT）具有相同的输入和输出接口，因此可以使用类似的损失函数进行训练。在解决下游任务时，它还支持以即插即用方式替换其他预训练模型。我们的逻辑归纳偏置假设，所有自然语言任务都共享“逻辑演绎技能”；也就是说，预训练过程中学到的这些技能应同样适用于解决各种下游任务。因此，FOLNet能够学习到迁移泛化能力强得多的语言表示。在若干语言理解任务（GLUE、SQuAD 2.0和FOLIO）上的实验结果表明，采用类似损失函数进行预训练时，我们的FOLNet模型以明显优势超越了Transformer架构。结果清楚地表明，使用逻辑归纳偏置来学习语言表示具有优势。

2 语言表示的逻辑归纳偏置

自然语言文本可以被视为离散符号序列，而语言表示学习研究的是如何将这些离散符号映射为某些更易于计算的形式这一问题。一种广泛使用的方法是分布式表示，它将离散的词元 ID 映射为稠密向量（Mikolov 等，2013；Pennington 等，2014；Peters 等，2018）。许多不同的函数形式，例如 LSTM（Hochreiter与Schmidhuber，1997），以及较新的 Transformer 模型（Vaswani 等，2017），都被用于实现此类映射。它们通常为自然语言建模编码不同类型的归纳偏置。例如，循环神经网络使用同一组模型参数随时间更新隐藏状态，从而编码了时间上的平移不变性（Battaglia等，2018）。这些归纳偏置形式持续推动自然语言任务求解水平达到最先进水平。在本节中，我们介绍一种名为逻辑归纳偏置的新型归纳偏置，它将与分布式表示协同工作，以设计更有效的表示映射。我们的主要思想是将语言表示映射视为一个逻辑推理过程；也就是说，语言表示是从原始离散词元序列中逐步演绎得到的。具体而言，我们将输入序列中的词元视为逻辑编程中的项（或对象），并将它们的属性和关系视为具有不同元数的谓词。根据逻辑归纳偏置，我们所要计算的语言表示，是可以从这些输入的（基本）属性和关系中演绎出的（高级）属性和关系。最重要的是，我们要求此类演绎过程的构建遵循一阶逻辑的原则，以便将逻辑归纳偏置编码到表示学习过程中。现在，我们采用与 Evans与Grefenstette（2018）中所使用的类似（逻辑编程）记法，将语言表示学习形式化为一个逻辑编程问题。

- **项：** 我们考虑一个不含函数符号的一阶逻辑系统，因此项只能是变量或常量。它们分别用于表示一般对象或特定的关注对象。在语言表示学习的语境中，我们将每个文本序列实例 x（长度为 T）建模为一个常量集合 $x = \{x_1, \ldots, x_T\}$，其中每个词元 x_t 都是一个常量（$t = 1$，……，T）。我们像逻辑编程中那样，用小写字母表示常量，用大写字母表示变量。例如， X 是一个用于表示一般对象（如词元）的变量。
- **原子：** 对于每个项，我们将其属性和关系定义为一个 r元谓词$p(X_1, \ldots, X_r)$；根据关于（$X_1, \ldots, X_r$）的某个属性/关系是否成立，该谓词取 T（真）或 F（假）。例如，词元 a是否取词汇表中的第 v 个 id，是一个一元谓词 TokenID$_v(a)$，其中 $v = 1, \ldots, V$，且 V

语言表示	逻辑编程	FOLNet
文本序列中的词元 x_t	常量： x_t	论证： x_t 张量$\mathbf{u}_l(x_t)$
词元 ID、相对距离等	输入（基本）原子： $\mathcal{C}_0 = \mathcal{B}$	输入张量： $\{\mathbf{u}_0(x)$、$\mathbf{u}_0(x,\ y)\}$
最终语言表示	推导出的（高级）原子： $\mathcal{C}_L$	输出张量： $\{\mathbf{u}_L(x)$，$\mathbf{u}_L(x,\ y)\}$
表示映射（部分）	使用通用子句（8）的肯定前件	神经逻辑算子：参见表 2
表示映射（部分）	1 步演绎： $\mathcal{C}_l = \mathrm{con}_{\mathcal{R}}(\mathcal{C}_{l-1})$	前向传播：1 层
表示映射（完整）	L 步演绎：前向链式推理（3）	前向传播： L 层

表1：将语言表示学习确定为逻辑编程问题 .

表示词汇表大小；两个词元 a 与 b 之间的距离是否等于 d，是一个二元谓词 $\mathrm{Dist}_d(a,b)$，其中 $|d| < T$。如果一个原子不含变量，则称其为基项，例如，上述 $\mathrm{TokenID}_v(a)$ 和 $\mathrm{Dist}_d(a,b)$ 都是基原子。

- **子句：**推理过程由一组“如果—那么”形式的子句构成：

$$q \leftarrow p_1 \wedge \cdots \wedge p_m, \tag{1}$$

其中，$p_1, \ldots, p_m$ 和 q 分别表示体原子和头原子，而 $\wedge$ 表示合取（即逻辑与）。这些原子分别充当前提和结论：如果 $p_1, \ldots, p_m$ 为真，那么 q 也为真。上述形式的子句称为确定霍恩子句（Horn，1951年）。如果一个子句中的所有变量都被常量替换，我们就称该子句为基事实规则。例如，对子句 $q(X,Y) \leftarrow p(X,Y)$应用替换 $\theta \triangleq \{a/X, b/Y\}$ 后，会得到一个基事实规则：$q(a,b) \leftarrow p(a,b)$。这可以看作是将一般子句应用于一个特定实例。

我们的目标是学习一组子句，并将它们组合成一个从输入谓词（例如，$\mathrm{TokenID}_v(x_t)$ 和 $\mathrm{RelDist}_d(x_t, x_\tau)$）到语言表示的映射。具体而言，设 $\mathcal{R}$ 为一个子句集合，$\mathtt{ground}(\mathcal{R})$ 为相应的基事实规则集合。我们将把 $\mathtt{ground}(\mathcal{R})$ 中的基事实规则应用于基原子集合 $\mathcal{X}$ 所产生的直接后果定义为

$$\mathtt{con}_{\mathcal{R}}(\mathcal{X}) = \mathcal{X} \cup \left\{ q \middle| q \leftarrow p_1 \wedge \cdots \wedge p_m \in \mathtt{ground}(\mathcal{R}), \bigwedge_{i=1}^{m} p_i \in \mathcal{X} \right\}. \tag{2}$$

它可以理解为一组基原子，这些原子可由 $\mathcal{X}$ 与其自身 $\mathcal{X}$ 推导得出。给定一组输入基原子 $\mathcal{B}$，我们可以重复应用 $\mathcal{R}$ 中的基事实规则，进行 L 步：

$$\mathcal{C}_l = \mathtt{con}_{\mathcal{R}}(\mathcal{C}_{l-1}), \qquad \mathcal{C}_0 = \mathcal{B} \text{ and } l = 1, \ldots, L. \tag{3}$$

然后，$\mathcal{C}_L$ 是在 L 步内可由 $\mathcal{B}$（包括其自身 $\mathcal{B}$）推导出的所有可能基原子（谓词）。上述过程称为 **前向链式推理**：通过重复应用（即链式连接）子句，它可以从输入前提（即 $\mathcal{B}$）推导出所有可能的结论。如果我们想验证谓词 q' 是否成立（即是否可以被蕴含），只需检查 q' 是否在 $\mathcal{C}_L$ 中即可。在语言表示学习中，例如，我们从以下输入（基本）原子开始：

$$\mathcal{B} = \{\mathtt{TokenID}_v(x_t), \mathtt{RelDist}_d(x_t, x_\tau), \ldots | t, \tau = 1, \ldots, T\}, \tag{4}$$

并通过对（学习得到的）子句进行前向链式推理，推导出 $\mathcal{C}_L$ 作为最终表示。例如，在解决一个（抽取式）问答问题时，某个词元是否为答案片段的开头（或结尾），被建模为该词元的一个高级推导属性，即

$$\mathcal{C}_L = \{\mathtt{AnswerStartsAt}(x_t), \mathtt{AnswerEndsAt}(x_t) | t = 1, \ldots, T\}. \tag{5}$$

在自回归语言建模中，推导出的性质会成为下一个词元 ID。表 1 总结了语言表示与逻辑编程之间的上述对应关系。接下来，我们将构建一种神经网络架构来编码这种逻辑归纳偏置。本文中，我们将用粗体字母表示向量（小写）和矩阵（大写）。

3 FOLNET：一种具有逻辑归纳偏置的神经网络架构

在本节中，我们开发一种名为 **F**irst-**O**rder **L**ogic **Net**work（FOLNet）的新型神经网络架构，用于编码第2节所介绍的逻辑归纳偏置。具体而言，我们重点关注：（i）如何将原子表示为连续向量（第 3.1 节）；（ii）如何设计近似式（2）的神经推理步骤（第 3.2 节）；以及（iii）如何将它们前向链接为基于式（3）的全可微架构（第 3.3 节）。总体神经网络架构及其与逻辑归纳偏置的对应关系如图1所示。接下来，我们将逐步讨论如何设计该架构、它的优势，以及它与现有方法之间的一些重要联系。

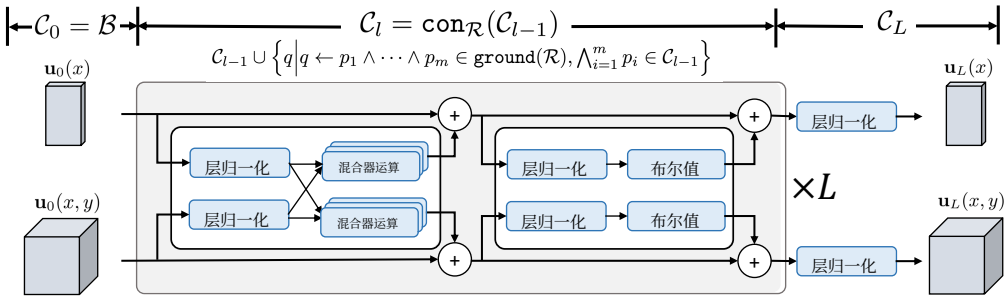


图1：FOLNet 架构概览及其对逻辑归纳偏置的编码方式。神经逻辑算子对各个子句进行建模，并通过前向链接将其构建为可微模型。“混合器运算”指表 2 中的运算符 c、j、m 和 t，因为它们会沿对象维度进行归约。

3.1 原子的向量表示

回顾一下，我们使用一个 r 元基原子 $p_d(x_1, \ldots, x_r)$ 来刻画对于词元元组（$x_1, \ldots, x_r$），第 d 个属性/关系是否成立，其中 $d = 1, \ldots, D_r$。为了克服学习这些取值于 {T, F} 的离散值原子的困难，我们引入$u_d(x_1, \ldots, x_r) \in \mathbb{R}$ 作为其连续表示，并刻画该原子p_d 为真的程度。例如，在 ProbLog（De Raedt 等，2007；De Raedt 与 Kimmig，2015；Fierens 等，2012；Manhaeve 等，2018）中，$u_d(\cdot)$ 给出了原子 $p_d(\cdot)$ 为真的概率。在本文中，我们将 $u_d(\cdot)$ 视为相应原子的 logit，即 $\Pr\{p_d(\cdot) = \mathrm{T}\} = 1/(1 + e^{-u_d(\cdot)})$；$u_d(\cdot)$ 的值越大，原子 $p_d(\cdot)$ 为真的可能性就越高。我们还可以很容易地验证，$p_d(\cdot)$ 的否定原子的 logit 简单地为 $-u_d(\cdot)$。令 $\mathbf{u}(x_1, \ldots, x_r) \in \mathcal{U} \subset \mathbb{R}^{D_r}$为一个 D_r 维向量，其中收集了 $u_d(x_1, \ldots, x_r)$ 作为其第 d 个元素。那么，$\mathbf{u}(x_1, \ldots, x_r)$ 将成为具有 r 元数的 D_r 原子的连续（logit）向量表示。例如，对于输入文本序列 $x = (x_1, \ldots, x_T)$，我们可以使用 $\mathbf{u}(x_t)$ 表示每个词元x_t的 D_1（一元）属性，并使用 $\mathbf{u}(x_t, x_\tau)$ 刻画任意词元对之间的 D_2（二元）关系。输入的（基本）属性/关系和演绎得到的（高级）属性/关系都将在同一个 logit 向量空间中表示，并在其中执行演绎过程。为方便起见，我们也可以直接通过其 logit 向量表示 $\mathbf{u}(\cdot)$ 来指代一组原子。

3.2 神经肯定前件推理

我们现在开发一组神经算子，用于实现式（2）所刻画的演绎步骤。首先，我们先介绍一阶逻辑中的肯定前件（MP）规则（Andrews，2013），该规则指出，若子句 $B \leftarrow A$ 和陈述 A 成立，则 B 为真：

$$B \Leftarrow \{B \leftarrow A \text{ and } A\}. \tag{6}$$

在式（2）的情境中，选择 $A = p_1 \wedge \cdots \wedge p_m$和 $B = q$时，式（2）中的演绎可视为利用 $\mathtt{ground}(\mathcal{R})$ 中所有基子句进行的一次 MP 推理。本文将重点限定在所有原子的元数为一或二的情形。也就是说，我们只考虑形式为$\mathbf{u}(x_t)$ 和$\mathbf{u}(x_t, x_\tau)$ 的原子（分别以其向量形式表示）。[2]接下来，我们需要实现从原子集合 $\{\mathbf{v}(a), \mathbf{v}(a, b)\}$ 到另一原子集合 $\{\mathbf{u}(x), \mathbf{u}(x, y)\}$的 MP 推理，这可分为以下四类：

$$\mathbf{u}(x) \Leftarrow \mathcal{R}_{\mathtt{UU}}, \mathbf{v}(a); \quad \mathbf{u}(x) \Leftarrow \mathcal{R}_{\mathtt{UB}}, \mathbf{v}(a, b); \quad \mathbf{u}(x, y) \Leftarrow \mathcal{R}_{\mathtt{BU}}, \mathbf{v}(a); \quad \mathbf{u}(x, y) \Leftarrow \mathcal{R}_{\mathtt{BB}}, \mathbf{v}(a, b) \tag{7}$$

其中，$\mathcal{R}_{\mathtt{UU}}$、$\mathcal{R}_{\mathtt{UB}}$、$\mathcal{R}_{\mathtt{BU}}$ 和 $\mathcal{R}_{\mathtt{BB}}$ 表示从某一元数的原子推导出另一元数原子的规则集合。例如，$\mathcal{R}_{\mathtt{BU}}$ 定义了一组子句，用于从一元（U）原子推导出二元（B）原子。现在，我们继续在 logit 空间中对这些子句及其推理进行建模。给定一组前提原子 P_1，……，P_M，我们考虑具有以下一般形式的 N子句：

$$Q_n \leftarrow \left(\bigwedge_{m \in \mathcal{M}_{n,+}} P_m \right) \bigwedge \left(\bigwedge_{m \in \mathcal{M}_{n,-}} \neg P_m \right), \quad n = 1, \ldots, N, \tag{8}$$

[2]原则上，将我们的工作推广到更高元数相对简单，但在实践中会导致较高的计算复杂度。我们将此类扩展留待未来工作。

Sym.	Ops.	类型标注	批次维度	秩维度	logit空间中的神经算子	核激活	备注
b	bool	$\mathrm{U} \leftarrow \mathrm{U} \times \mathrm{U}$	x	w	$\mathbf{u}_{hs}(x) = \sum_w \mathbf{K}_{hw}(x)\mathbf{v}_{ws}(x)$	恒等映射	
c	c拼接	$\mathrm{U} \leftarrow \mathrm{U} \times \mathrm{B}$	h	a	$\mathbf{u}_{hs}(x) = \sum_a \mathbf{K}_{hs}(a)\mathbf{v}_h(x, a)$	$\mathrm{Softmax}_a$	
j	join	$\mathrm{U} \leftarrow \mathrm{B} \times \mathrm{U}$	h	a	$\mathbf{u}_{hs}(x) = \sum_a \mathbf{K}_h(x,\ a)\mathbf{v}_{hs}(a)$	$\mathrm{Softmax}_a$	自注意力 通用相对位置编码
m	mu	$\mathrm{U} \leftarrow \mathrm{B} \times \mathrm{B}$	x	a	$\mathbf{u}_{hs}(x) = \sum_a \mathbf{K}_h(x,\ a)\mathbf{v}_s(x,\ a)$	$\mathrm{Softmax}_a$	
a	结合律	$\mathrm{B} \leftarrow \mathrm{U} \times \mathrm{U}$	h	w	$\mathbf{u}_h(x, y) = \sum_w \mathbf{K}_{hw}(x)\mathbf{v}_{hw}(y)$	恒等映射	自注意力 通用相对位置编码
p	prod	$\mathrm{B} \leftarrow \mathrm{U} \times \mathrm{B}$	x	w	$\mathbf{u}_h(x, y) = \sum_w \mathbf{K}_{hw}(x)\mathbf{v}_w(x, y)$	恒等映射	
t	转置	$\mathrm{B} \leftarrow \mathrm{B} \times \mathrm{B}$	h	a	$\mathbf{u}_h(x, y) = \sum_a \mathbf{K}_h(x,\ a)\mathbf{v}_h(a,\ y)$	$\mathrm{Softmax}_a$	

表 2：我们所有采用受限核的神经逻辑算子列表（命名协议见附录 A.1）。请注意，每个算子都有唯一的批处理维度（B-dim）和唯一的归约维度（R-dim）。算子的类型标注定义了核、前提和输出原子的元数。例如，$\mathrm{U} \leftarrow \mathrm{B} \times \mathrm{U}$ 表示核、输入原子和输出原子的元数分别为 2、1 和 1。我们还列出了用于计算相应核的激活函数，其中 Softmax 的归一化维度标注在其下标中。

其中 Q_n 是头原子，$\neg$ 表示逻辑否定（即 NOT），而 $\mathcal{M}_{n,+}$ 和 $\mathcal{M}_{n,-}$ 是 $\mathcal{M} = \{1, \ldots, M\}$ 的两个子集，且 $\mathcal{M}_{n,+} \cap \mathcal{M}_{n,-} = \emptyset$。然后，头原子$\{Q_n\}$ 的 logit 向量 $\mathbf{u}$ 可以通过对前提 $\{P_m\}$ 的 logit 向量 $\mathbf{v}$ 进行矩阵乘法，并接着应用（可选的）逐元素非线性激活函数来近似推断（附录 D）：

$$\mathbf{u} = \sigma(\mathbf{K}\mathbf{v}), \tag{9}$$

其中 K 是一个 $N \times M$ 核矩阵，用于表示（8）中的子句，而 $\sigma(\mathrm{z}) = \ln(1 + 2e^{\mathrm{z}})$ 是激活函数。值得注意的是，K 的每一行都刻画了（8）右侧的合取模式：对于第 n 个子句，其第 (n, m) 个元素 K_{nm} 的值为正（负），意味着 m 在$\mathcal{M}_{n,+}$ 中更（不）可能。由此可知，（7）中 $\mathcal{R}_{\mathrm{UU}}, \mathcal{R}_{\mathrm{UB}}, \mathcal{R}_{\mathrm{BU}}$ 和 $\mathcal{R}_{\mathrm{BB}}$ 对应的核分别具有 $\mathrm{K}_{\mathrm{UU}}(x, a)$、$\mathrm{K}_{\mathrm{UB}}(x, a, b)$、$\mathrm{K}_{\mathrm{BU}}(x, y, a)$ 和 $\mathrm{K}_{\mathrm{BB}}(x, y, a, b)$ 的形式，它们分别是$D_1 \times D_1$, $D_1 \times D_2, D_2 \times D_1$ 和 $D_2 \times D_2$ 矩阵。（9）则可表示为这些矩阵与各自对应的前提（即 $\mathrm{v}(a)$、$\mathrm{v}(a, b)$、$\mathrm{v}(a)$ 或 $\mathrm{v}(a, b)$）之间的矩阵乘法，随后对其中出现的 a 和 b 求和（参见附录 D 中的（30）–（33））。最后，当用“≡”替换“←”时，可以省略激活函数 $\sigma(\cdot)$；其中 $A \equiv B$ 当且仅当 $A \leftarrow B$ 和 $B \leftarrow A$。

直接实现（9）来处理（7）中的推理规则，一大限制是内存与计算成本高。例如，核 $\mathrm{K}_{\mathrm{BB}}(x, y, a, b)$ 需要 $O(D_2^2 T^4)$ 来存储其值。而 MP 推理（9）现在变为 $\mathrm{u}(x, y) = \sum_{a,b} \mathrm{K}(x, y, a, b)\mathrm{v}(a, b)$，其计算复杂度也为 $O(D_2^2 T^4)$。因此，我们必须限制核的规模，并采用不同的方法降低整体复杂度，例如在不同维度之间共享 $\mathrm{K}_{\mathrm{BB}}(x, y, a, b)$ 的值。现在，我们基于以下原则提出一种系统化的方法：

1. 我们将所有核限制为以下形式：$\{\mathbf{K}(\omega), \mathbf{K}(\omega,\ \nu)\}$，即元数 $r = 1,\ 2$。2. 在矩阵乘法中，我们选取一个归约维度和一个批处理维度。

基于上述假设，我们对一元核函数和一元原子的谓词维度进行因式分解，使得 $\mathrm{K}_d(\omega) = \mathrm{K}_{hs}(\omega)$ 和 $\mathrm{u}_d(x) = \mathrm{u}_{hs}(x)$，其中 $d = (h-1)S + s$，且 $h = 1$、.....、H 和 $s = 1$、.....、S。这一做法受到多头注意力机制（Vaswani 等，2017）的启发，其中 h类似于头索引，H 表示头的数量，而 S 表示头的大小。然后，我们通过恰当选择不同的归约维度和批处理维度，枚举所有能够将来自 $\{\mathrm{K}(\omega)、\mathrm{K}(\omega, \nu)\}$ 的核与来自 $\{\mathrm{v}(a)、\mathrm{v}(a, b)\}$ 的前提相乘的兼容神经逻辑算子。基于此，我们在表 2 中列出每种类型标注对应的神经逻辑算子，并在附录 A.1 中进一步讨论它们在（受限）肯定前件推理中的不同作用。

与 Transformer 的联系 有趣的是，我们发现 j 算子和 a 算子的形式与 Transformer 中的自注意力机制相似，其中 a 算子计算自注意力分数，而 j 算子执行自注意力操作。此外，m-算子和 p-算子确实推广了 (Shaw 等，2018) 提出的现有相对位置编码，这些编码广泛用于不同的 Transformer 变体，例如 T5 模型 (Raffel 等，2020)。具体而言，我们在附录 E 中表明，将 $\mathbf{v}_w(x, y)$ 设为与实例无关后，p-算子便会计算 Shaw 等人（2018）公式 (5) 中的第二项，其中 $\mathbf{v}_w(x, y)$ 扮演 a_{ij}^K 的角色。而将 $\mathbf{v}_s(x, a)$ 设为与实例无关后，m-算子便会计算第二

项，即 Shaw 等人（2018）公式 (3) 中的第二项，其中 $\mathbf{v}_s(x, a)$ 扮演 a_{ij}^V 的角色。也就是说，在这种退化 设置下，这两个算子将构成该文提出的相对位置编码。请注意，他们的 a_{ij}^K 和 a_{ij}^V 是静态可学习嵌入，而我们的 $\mathbf{v}_w(x, y)$ 和 $\mathbf{v}_s(x, a)$ 则针对每个实例动态计算（我们将在第 3.3 节讨论）。因此，我们的 m-op 和 p-op 也可以视为一种更具适应性的相对位置编码，其优势将在实验中进一步展示（第 4 节）。

3.3 前向链式推理与可微学习

请注意，一般而言，逻辑算子接收一个核 K 和一个前提 v，以推断出一个结果 u（表 2）。具体来说，它针对特定的类型标注（例如 U $\leftarrow$ B $\times$ U），对式（2）中的逻辑演绎进行“神经化”。应用所有神经逻辑算子，相当于完整执行式（2），这就是式（3）中的一个递归步骤，它将一组 $\{\mathrm{u}_{l-1}(x), \mathrm{u}_{l-1}(x, y)\}$ 映射为 $\{\mathrm{u}_l(x), \mathrm{u}_l(x, y)\}$。因此，我们可以自然地将它们的 L 个阶段进行前向链式组合，构建一个全可微架构，对式（3）中的推理链进行建模。图1展示了这样的前向链式推理过程，以及它如何编码第2节所述的逻辑归纳偏置。剩下的一个问题是，如何从图1中的 FOLNet 架构获取核 K(·) 和前提 v(·)。我们只需对前一层的输出 $\{\mathrm{u}_l(x), \mathrm{u}_l(x, y)\}$应用两个线性投影（分别用于前提和核）即可。对于核，我们还可以在线性投影之后应用激活函数来计算核（各算子所用核激活函数的列表见表 2）。换言之，我们通过 FOLNet 本身（的中间演绎结果）对核 K(·) 和前提 v(·)进行参数化。这基于这样一个观察：子句本身就是谓词，因为 $A \leftarrow B$ 被定义为 $A \vee \neg B$ (Andrews, 2013)，其中 $\vee$ 表示析取（逻辑或）。现在我们来描述图1中 FOLNet 的输入和输出。在输入端，我们可以通过标准的嵌入查找，将词元的离散词元 ID x_t 编码为 $\mathrm{u}_0(x_t)$形式的向量。同样，我们也将两个词元x_t和 x_τ之间的（离散）相对距离转换为 $\mathrm{u}_0(x_t, x_\tau)$形式的向量。 $\{\mathrm{u}_0(x_t), \mathrm{u}_0(x_t, x_\tau)\}_{t,\tau}$将作为基本原子的向量表示，并输入 FOLNet 模型（图1）。经过 L 层（即 L 步演绎）后，输出 $\{\mathrm{u}_L(x_t), \mathrm{u}_L(x_t, x_\tau)\}_{t,\tau}$成为式（3）中 $\mathcal{C}_L$ 的向量表示，并用作最终的语言表示。因此，我们的 FOLNet 模型与其他 Transformer 模型具有相同的输入输出接口，可以即插即用地用于解决下游任务。因此，我们的模型也可以使用与其他仅编码器模型相同的损失函数（例如 MLM、NSP、SOP 等），在大规模文本上进行预训练——关于如何根据 $\{\mathrm{u}_L(x_t), \mathrm{u}_L(x_t, x_\tau)\}_{t,\tau}$计算这些损失函数，请参见附录 A.4。通过稍微修改神经逻辑算子（见附录 A.5），我们的模型还可以扩展为仅解码器版本和编码器-解码器版本，进而能够以自回归方式预训练来预测下一个词。最后，我们将讨论 FOLNet 架构的若干重要属性。

双分支架构 请注意，图1中的 FOLNet 模型有两个分支：(i) 用于对$\mathbf{u}_l(x)$进行推理的一元谓词分支，以及 (ii) 用于对$\mathbf{u}_l(x,\ y)$进行推理的二元谓词分支。这与 Transformer 模型的单分支架构（附录A.2中的图3）形成鲜明对比。我们还注意到，当 FOLNet 仅加载 j 算子和 a 算子时，它会退化为 Transformer架构的双分支变体。在我们的实验中，我们将展示，即使在这种退化设置下，FOLNet 的表现仍优于 Transformer。这可能是因为二元谓词分支在整个推理过程中显式维护成对关系$\mathbf{u}_l(x,\ y)$。此外，显式的二元谓词分支还使我们能够将相对距离知识直接输入模型，而无需像现有相对位置编码那样执行不那么直观的操作（Shaw 等，2018）。在第 4 节的实验中，我们将展示这种简单而有效的相对位置信息使用方法的优势，以及 FOLNet 双分支架构的其他一些优势。

4 实验

4.1 实验设置

现在，我们在不同设置下评估 FOLNet 模型，并试图回答以下问题：编码逻辑归纳偏置的神经网络架构（*FOLNet*）能否比 *Transformer* 模型学到更好的语言表示？为此，我们需要排除其他

	模型	参数量	PE	D_2	Loss	MNLI-m/mm	QQP	QNLI	SST-2	CoLA	STS-B	MRPC	RTE	平均值
1	BERT	110M	APE	-	掩码语言模型与下一句预测	84.5/-	91.3	91.7	93.2	58.9	89.5	87.3	68.6	83.1
2	BERT（我们的）	110M	APE	-	掩码语言模型与下一句预测	83.9/84.1	90.9	88.2	92.6	61.5	89.2	88.2	66.8	82.7
3	FOLNet：j.a	110M	APE	12	掩码语言模型与下一句预测	84.9/84.5	91.1	91.6	92.2	61.1	89.6	89.5	72.2	84.0
4	FOLNet：j.a	109M	RPE	12	掩码语言模型与下一句预测	85.0/84.9	91.4	91.4	93.5	63.8	90.1	89.9	72.9	84.7
5	FOLNet：j.a	110M	相对位置编码*	12	掩码语言模型与下一句预测	84.7/84.9	91.4	91.5	93.8	63.4	89.8	90.6	72.6	84.7
6	FOLNet：jm.ap	123M	RPE	12	掩码语言模型与下一句预测	**85.7/85.3**	**91.6**	**91.8**	**93.8**	**65.7**	**90.4**	**91.0**	**73.5**	**85.4**
7	FOLNet：j.a	109M	RPE	16	掩码语言模型与下一句预测	85.2/85.2	91.3	91.8	93.7	64.1	89.9	89.3	71.8	84.6
8	FOLNet：j.a	109M	RPE	32	掩码语言模型与下一句预测	**85.8/85.7**	91.4	**92.0**	**93.7**	**64.2**	90.0	**90.5**	71.8	84.9
9	FOLNet：j.a	110M	RPE	64	掩码语言模型与下一句预测	85.7/85.5	**91.4**	91.8	93.2	63.5	**90.1**	90.2	**74.7**	**85.1**
10	FOLNet：j.at	110M	RPE	64	掩码语言模型与下一句预测	86.7/86.6	91.6	92.8	93.1	63.6	91.1	89.9	80.9	86.2
11	FOLNet：j.atp	117M	RPE	64	掩码语言模型与下一句预测	87.4/87.4	91.9	93.3	94.0	62.9	91.3	91.4	81.6	86.7
12	FOLNet：jm.atp	124M	RPE	64	掩码语言模型与下一句预测	88.1/87.6	91.7	93.9	94.2	64.7	91.2	91.4	83.2	87.3
13	FOLNet：jmc.atp	138M	RPE	64	掩码语言模型与下一句预测	**88.2/87.9**	**91.9**	**94.1**	**94.5**	**66.9**	**91.6**	**91.5**	**83.5**	**87.7**
14	FOLNet：jmc.atp	137M	RPE	12	掩码语言模型与下一句预测	85.9/86.3	91.6	92.7	93.6	63.4	90.5	91.0	75.8	85.6
15	FOLNet：jmc.atp	138M	RPE	64	掩码语言模型，句子顺序预测	88.3/87.9	91.8	94.2	94.7	65.6	91.1	91.1	83.2	87.5

表3：在 GLUE基准开发集上对 FOLNet 的分析。所有结果均为五个随机种子的中位数。从上到下，第一个区块展示了双分支架构的优势，并将我们新提出的位置编码与其他方法进行比较；第二个区块分析二元谓词维度的影响；第三个区块对所有逻辑算子进行消融研究；最后一个区块展示了 FOLNet 使用其他预训练损失的结果。APE 表示绝对位置编码，RPE* 表示 T5 使用的相对位置编码（Shaw 等，2018；Raffel 等，2020），RPE 则表示我们提出的相对位置编码。为进行公平比较，我们还采用与 FOLNet 相同的设置预训练了一个 BERT（基础版）模型（第 #2 行）。

混杂因素，并确保唯一的差异在于模型架构本身。首先，我们选择使用广泛采用的掩码语言模型（MLM）损失函数（Devlin 等人，2018年）对 FOLNet 模型进行预训练，并额外加入下一句预测（NSP）（Devlin 等人，2018年）或句序预测（SOP）（Lan 等人，2019年）损失函数。许多广泛采用的仅编码器 Transformer 模型变体，例如 BERT、RoBERTa、ALBERT、DeBERTa 和 Megatron-LM，都是使用这些损失函数进行预训练的。因此，我们也将这些模型作为主要基线。虽然可能存在其他效率更高的预训练损失函数，例如（Bao 等人，2020；Clark 等人，2020；Yang 等人，2019；Meng 等人，2021）中的方法，但我们认为，开发具有更优归纳偏置的新模型架构，是一条独立的研究路线。因此，我们将探索 FOLNet 的其他预训练损失函数留待未来工作。此外，我们考虑两种预训练数据集设置：（i）维基百科 + BookCorpus（Zhu 等人，2015）（文本共 16GB）；（ii）包含 160GB 文本的较大数据集，其中包括维基百科、BookCorpus2、OpenWebText2 和 Pile-CC（从 Pile数据集中提取（Gao 等人，2020））。在所有实验中，我们都使用带有 32,768 个不区分大小写的 BPE 词汇表的 BERT分词器（Sennrich 等人，2016）。[3] 我们考虑两种不同规模的 FOLNet 模型：$\text{FOLNet}_{\text{Base}}$ 和 $\text{FOLNet}^{\text{Large}}$，其规模与文献中的基础版（例如 $\text{BERT}_{\text{Base}}$）和大型模型（例如 $\text{BERT}^{\text{Large}}$）相当。最后，FOLNet 模型始终使用 128 个词元的序列长度进行预训练，但在评估时，会针对序列长度更长的不同下游任务进行测试（例如 384 或 512）。评估时，我们采用三个基准：GLUE（Wang 等人，2019年）、SQuAD 2.0（Rajpurkar 等人，2016b）和 FOLIO（Han等人，2022年），并针对每项任务分别对预训练模型进行微调。有关这些下游任务和超参数的更多详细信息，请参阅附录B。

4.2 FOLNet架构分析

我们首先对 FOLNet 进行深入分析，以展示其相较于 Transformer架构 的优势。为此，我们首先在维基百科和 BookCorpus 数据集上预训练 $\text{FOLNet}_{\text{Base}}$ 模型，该数据集与 BERT 所使用的数据集相同。随后，我们进一步采用掩码语言模型和下一句预测作为预训练损失，使其与 BERT 保持一致。我们在 GLUE 基准上从不同方面分析 FOLNet，并将结果报告于表 3 中。下面我们将继续讨论这些结果。

[3]尽管 Liu 等（2019）指出，对规模大得多的 160GB 文本进行预处理时，使用字节级BPE会更加理想，但为简化流程，我们仍使用相同的 BERT分词器（基于字符级BPE）处理这些 160GB 文本，而它已经展现出我们模型的强大性能。采用（Liu 等，2019；Radford 等，2019；Brown 等，2020）中那样、词汇表更大的 50K 字节级BPE分词器，或许能进一步提升在 160GB 文本上预训练的 FOLNet 模型。

模型	参数量	GLUE									SQuAD 2.0		FOLIO
		MNLI-m/mm	QQP	QNLI	SST-2	CoLA	STS-B	MRPC	RTE	平均值	EM	F1	Acc
$\text{BERT}_{\text{Base}}$	110M	84.5/-	91.3	91.7	93.2	58.9	89.5	87.3	68.6	83.1	73.7	76.3	57.8
$\text{RoBERTa}_{\text{Base}}$	110M	85.8/85.5	91.3	92.0	93.7	60.1	88.5	87.3	68.2	83.3	77.7	80.5	-
$\text{DeBERTa}_{\text{Base}}$	134M	86.3/86.2	-	-	-	-	-	-	-	-	79.3	82.5	-
FOLNet Base	138M	**88.2/87.9**	**91.9**	**94.1**	**94.5**	**66.9**	**91.6**	**91.5**	**83.5**	**87.7**	**84.7**	**87.9**	**64.2**
$\text{BERT}^{\text{Large}}$	340M	86.6/-	91.3	92.3	93.2	60.6	90.0	88.0	70.4	84.1	79.0	81.8	62.3
$\text{RoBERTa}_{\text{Base}}$	125M	87.6/-	91.9	92.8	94.8	63.6	91.2	90.2	78.7	86.4	80.5	83.7	**64.7**
DeBERTa Base	134M	88.8/88.5	-	-	-	-	-	-	-	-	83.1	86.2	-
FOLNet Base	138M	**89.4/89.7**	**92.2**	**94.4**	**95.6**	**69.9**	**92.5**	**92.0**	**87.0**	**89.2**	**85.5**	**88.6**	64.2
$\text{RoBERTa}^{\text{Large}}$	356M	90.2/90.2	92.2	94.7	96.4	68.0	92.4	90.9	86.6	88.9	86.5	89.4	67.7
$\text{DeBERTa}^{\text{Large}}$	384M	91.1/91.1	92.3	95.3	96.8	70.5	-	-	-	-	88.0	90.7	-
$\text{ALBERT}_{\text{XXL}}$	235M	90.4/-	92.0	95.2	96.8	68.7	92.7	90.2	88.1	89.3	87.2	89.9	-
$\text{ALBERT}_{\text{XXL+}}$	235M	90.8/-	92.2	95.3	**96.9**	71.4	**93.0**	90.9	89.2	89.9	87.4	90.2	-
$\text{FOLNet}^{\text{Large}}$	437M	**91.2/91.3**	**92.5**	**95.8**	96.8	**71.5**	92.2	**93.5**	**91.1**	**90.6**	**88.5**	**91.5**	**70.6**
$\text{Megatron}_{\text{1.3B}}$	1.3B	90.9/91.0	92.6	-	-	-	-	-	-	-	87.1	90.2	-
$\text{Megatron}_{\text{3.9B}}$	3.9B	91.4/91.4	92.7	-	-	-	-	-	-	-	88.5	91.2	-

表4：GLUE、SQuAD 2.0 和 FOLIO 开发集上的总体结果。表格的上半部分（由实线分隔）展示了在维基百科+ BookCorpus（16GB）上预训练的模型结果，下半部分则展示了在扩展数据（160GB）上预训练的模型结果。我们使用虚线分隔每个部分中不同规模的模型。所有结果均为五个随机种子的中位数。FOLIO 的基线结果由 Han 等人（2022）提供。这里我们使用 $\text{ALBERT}_{\text{XXL}}$ 指代经过 100 万步预训练的 ALBERT 模型，并使用 $\text{ALBERT}_{\text{XXL+}}$ 指代经过 150 万步预训练的 $\text{ALBERT}_{\text{XXL}}$ 模型。

双分支架构的优势 请回想一下，当 FOLNet 仅包含 join 和 assoc 算子时，可以将其视为Transformer架构的双分支版本。为进行公平比较，我们使用与 BERT 相同的绝对位置编码（APE）对 FOLNet 模型进行预训练（表3 第 3 行）。值得注意的是，即使在这种过度退化的情况下，FOLNet 的平均表现仍明显优于 BERT。为 FOLNet 配备我们新提出的相对位置编码（RPE）（表3 第 4 行）后，其平均表现将比 BERT 高出 2 个百分点。值得注意的是，与采用 T5 相对位置编码的模型相比（第 5 行中的 RPE*），其平均表现不相上下（甚至略好）。正如我们在前文讨论的，T5 的 RPE 是我们的 mu 和 prod 算子的退化版本。表3 第 6 行表明，将 mu 和 prod 算子添加到 FOLNet 中，可以进一步显著提升性能。

更大维度的优势 D_2 我们可以看到（表3中的第7至第9行），增大维度 D_2会持续提升性能。正如我们很快将要揭示的，当 FOLNet 配备全部逻辑算子后，拥有更大的 D_2 对于释放它们的全部能力至关重要；也就是说，对于配备全部逻辑算子的 FOLNet，增大 D_2 所带来的性能提升会更为显著。

逻辑算子的贡献 现在，我们通过逐一将逻辑算子加入 FOLNet，直至其配备全部逻辑算子，来分析这些逻辑算子的贡献。从表3中的第9行到第13行可以看出，这会显著提升平均性能。在第14行中，我们评估了一个配备全部逻辑算子的 FOLNet，并将其 D_2 从64降至12，结果显示性能显著下降。这证实了拥有相对较大的 D_2 对于存储推导关系的重要性。

4.3 总体性能

在仔细考察了 FOLNet 架构的各个方面后，我们现在继续在三个基准（GLUE、SQuAD 2.0 和 FOLIO）上对其进行全面评估。我们还将考察进一步扩大预训练数据量和模型规模后其性能的变化。为了在更大的（160GB）数据集上预训练模型，我们发现使用 SOP 损失函数生成预训练数据更加高效。这是因为 NSP 损失函数要求我们从另一篇文档中采样负句子。首先，我们通过在维基百科和 BookCorpus 上使用 SOP 损失函数预训练 $\text{FOLNet}_{\text{Base}}$，验证其性能。结果（表3中的第15行）显示，与使用 NSP 的模型（第13行）相比，其性能略有下降。不过，考虑到在大型语料库上进行预训练时的便利性，这种下降是相对可以接受的。因此，在进行预训练时，我们将用 SOP 替代 NSP

在 160GB 数据集上预训练 $\text{FOLNet}_{\text{Base}}$ 和 $\text{FOLNet}^{\text{Large}}$。我们在表4中展示了 GLUE基准上的结果，并将其与其他基线方法进行比较。可以观察到，在 16GB 数据和 160GB 数据上预训练的 $\text{FOLNet}_{\text{Base}}$ 模型，在三个基准上都显著优于其他基于 Transformer 的模型。在 16GB 数据上预训练的 $\text{FOLNet}_{\text{Base}}$，其性能优于模型规模更大的 $\text{BERT}^{\text{Large}}$ 模型3×。此外，它甚至优于在 160GB 数据上预训练的 $\text{RoBERTa}_{\text{Base}}$。在 160GB 数据上预训练 $\text{FOLNet}_{\text{Base}}$ 模型时，我们甚至在 GLUE基准上超过了 $\text{RoBERTa}^{\text{Large}}$ 模型（ 89.3 对 88.9 ）。同样，我们的 $\text{FOLNet}^{\text{Large}}$ 模型也显著优于所有其他基线方法。它甚至优于经过 150 万步预训练的 $\text{ALBERT}_{\text{XXL+}}$（即预训练期间使用的词元多 50%）。值得注意的是，我们的 $\text{FOLNet}^{\text{Large}}$ 模型在 GLUE 和 SQuAD 2.0 基准上的性能与 $\text{Megatron}_{\text{3.9B}}$ 模型相当，而后者的规模是我们模型的 10 倍。此外，尽管我们的 FOLNet 模型并非为解决推理问题而设计（而是在词元级别将推理作为归纳偏置融入其中），但我们的模型在 FOLIO 任务上仍持续展现出更强的一阶逻辑推理能力（例如，$+3.9$优于 $\text{RoBERTa}^{\text{Large}}$）。最后，我们想强调的是，所有 $\text{FOLNet}_{\text{Base}}$ 和 $\text{FOLNet}^{\text{Large}}$模型都使用 128 的序列长度进行预训练，而其他基线方法使用的是 512。不过，它们在 GLUE、SQuAD 2.0 和 FOLIO 上评估时，使用的序列长度分别为 128、384 和 512。特别是，仅通过使用 1,004 个训练样例在 FOLIO 上进行微调，它就能够泛化到长得多的序列（512），而这些序列在预训练期间从未出现过。

5 相关工作

Transformer语言模型自 Bengio 等（2000）以来，神经语言模型领域已有大量研究。近年来，通过探索各种预训练 Transformer 模型（Vaswani 等，2017）的变体来解决下游语言任务，相关研究取得了巨大成功，例如通过微调（Radford 等，2018；Devlin 等人，2018；Lan 等人，2019；Liu 等人，2019；Yang 等人，2019），或使用大语言模型进行零样本/少样本学习（Radford 等，2019；Brown 等，2020；Chowdhery 等，2022）。另一条活跃的研究方向着眼于开发更有效的预训练损失函数（Yang 等人，2019；Clark 等人，2020；Bao 等人，2020；Tay 等人，2022），以超越广泛使用的自回归或掩码语言建模目标。开发新的神经网络架构以学习更好的语言表示的研究则相对有限。本文旨在沿着这一方向推进，并开发一种基于逻辑归纳偏置的新型神经网络架构。

逻辑编程与神经推理 我们的逻辑归纳偏置受到逻辑编程（Horn，1951年；De Raedt 等，2007年；De Raedt 和 Kimmig，2015年；Fierens 等，2012年；Manhaeve 等，2018年）以及归纳逻辑编程（Evans 和 Grefenstette，2018年；Muggleton，1991年；1996年；Friedman 等，1999年）的启发。不同于这些工作，我们并不直接处理推理问题。相反，我们致力于将逻辑归纳偏置编码到神经模型中，以学习更好的语言表示。另一类相关工作则关注开发能够广义上执行推理的神经模型。例如，研究者提出了不同变体的记忆增强网络（Le 等，2020；Santoro 等，2018年；Graves 等，2014年；2016年；Santoro 等，2016年；Le 等，2019年），通过记忆单元及其配套的神经读写模块增强控制网络。此外，Rocktäschel 和 Riedel（2017年）以及 Minervini 等（2020）通过反向链接构建可微神经网络，研究了证明知识库中结构化查询的问题。Bergen 等（2021）提出了三角注意力机制，用于从其他关系中推导关系；这可以被视为具有单一关系分支的 Transformer。这些方法在解决（相对小规模的）推理任务方面十分有效。然而，目前尚不清楚它们是否能够在大规模文本上进行有效的预训练，以解决多样化的下游自然语言任务。

6 结论

我们提出了一种新颖的逻辑归纳偏置，将语言表示学习视为逻辑编程问题。随后，我们开发了一种完全可微的神经网络架构（FOLNet），通过对一组丰富的神经逻辑算子进行前向链式推理，有效地编码这一归纳偏置。所提出的 FOLNet 架构具有与 Transformer 模型相同的输入输出接口，并且可以在大规模文本数据上进行预训练。实验结果表明，FOLNet 架构显著优于不同变体的 Transformer 模型；此外，由于采用了全新的双分支架构以及一组丰富的神经逻辑算子，它还具有诸多固有优势。

参考文献

Peter B Andrews. 数理逻辑与类型论导论：通过证明探求真理，第 27 卷。Springer Science & Business Media，2013 年。

Hangbo Bao、Li Dong、Furu Wei、Wenhui Wang、Nan Yang、Xiaodong Liu、Yu Wang、Jianfeng Gao、Songhao Piao、Ming Zhou 等。UniLMv2：用于统一语言模型预训练的伪掩码语言模型。载于国际机器学习会议，第 642–652 页。PMLR，2020。

Peter W Battaglia、Jessica B Hamrick、Victor Bapst、Alvaro Sanchez-Gonzalez、Vinicius Zambaldi、Mateusz Malinowski、Andrea Tacchetti、David Raposo、Adam Santoro、Ryan Faulkner 等。关系归纳偏置、深度学习与图网络。*arXiv* 预印本 *arXiv:1806.01261*，2018年。

Yoshua Bengio、Réjean Ducharme 和 Pascal Vincent。神经概率语言模型。载于 T. Leen、T. Dietterich 和 V. Tresp（编），神经信息处理系统进展，第 13 卷。麻省理工学院出版社，2000。网址：https://proceedings.neurips.cc/paper/2000/ file/728f206c2a01bf572b5940d7d9a8fa4c-Paper.pdf。

Luisa Bentivogli、Peter Clark、Ido Dagan 和 Danilo Giampiccolo。第五届 PASCAL文本蕴涵识别挑战赛。载于*TAC*，2009年。

Leon Bergen、Timothy O'Donnell 和 Dzmitry Bahdanau。利用边缘Transformer实现系统泛化。载于神经信息处理系统进展，34：1390–1402，2021年。

Tom Brown、Benjamin Mann、Nick Ryder、Melanie Subbiah、Jared D Kaplan、Prafulla Dhariwal、Arvind Neelakantan、Pranav Shyam、Girish Sastry、Amanda Askell 等。语言模型是少样本学习器。载于神经信息处理系统进展，33：1877–1901，2020年。

Daniel Cer、Mona Diab、Eneko Agirre、Iñigo Lopez-Gazpio 和 Lucia Specia。SemEval-2017任务1：多语言与跨语言聚焦评测中的语义文本相似度。载于第*11*届语义评测国际研讨会（*SemEval-2017*）论文集，第1–14页，温哥华，加拿大，2017年8月。计算语言学协会。doi：10.18653/v1/S17-2001。URL https://aclanthology.org/S17-2001。

Aakanksha Chowdhery、Sharan Narang、Jacob Devlin、Maarten Bosma、Gaurav Mishra、Adam Roberts、Paul Barham、Hyung Won Chung、Charles Sutton、Sebastian Gehrmann 等。PaLM：借助 Pathways 扩展语言建模。*arXiv* 预印本 *arXiv:2204.02311*，2022年。

Kevin Clark、Minh-Thang Luong、Quoc V Le 和 Christopher D Manning。Electra：将文本编码器预训练为判别器而非生成器。*arXiv* 预印本 *arXiv:2003.10555*，2020。

Ido Dagan、Oren Glickman 和 Bernardo Magnini。PASCAL文本蕴涵识别挑战。载于机器学习挑战研讨会，第177–190页。Springer，2005年。

Luc De Raedt 和 Angelika Kimmig。概率（逻辑）编程概念。*MachineLearning*，100(1)：5–47，2015年。

Luc De Raedt、Angelika Kimmig 和 Hannu Toivonen。ProbLog：一种概率Prolog及其在链接发现中的应用。载于*IJCAI*，第 7 卷，第 2462–2467 页。海得拉巴，2007年。

Jacob Devlin、Ming-Wei Chang、Kenton Lee 和 Kristina Toutanova。BERT：用于语言理解的深度双向变换器预训练。载于*NAACL-HLT* 论文集，第 4171–4186 页，2018年。

William B. Dolan 和 Chris Brockett。自动构建句子释义语料库。载于第三届国际释义研讨会（*IWP2005*）论文集，2005年。网址：https://aclanthology.org/I05-5002。

Richard Evans 和 Edward Grefenstette。从噪声数据中学习解释规则。人工智能研究杂志，61：1–64，2018年。

Daan Fierens、Guy Van den Broeck、Maurice Bruynooghe 和 Luc De Raedt。概率逻辑编程的约束。在 *NIPS* 概率编程研讨会论文集，第 1–4 页，2012 年。

Nir Friedman、Lise Getoor、Daphne Koller 和 Avi Pfeffer。概率关系模型的学习。在 *IJCAI* 论文集，1999 年。Leo Gao、Stella Biderman、Sid Black、Laurence Golding、Travis Hoppe、Charles Foster、Jason Phang、Horace He、Anish Thite、Noa Nabeshima 等。The Pile：用于语言建模的多样化文本 800 GB 数据集。*arXiv* 预印本 *arXiv:2101.00027*，2020 年。Tianyu Gao、Adam Fisch 和 Danqi Chen。让预训练语言模型成为更出色的少样本学习器。在 计算语言学协会第 *59* 届年会暨自然语言处理国际联合会议第 *11* 届会议论文集（第 *1* 卷：长篇论文），第 3816–3830 页，2021 年。Danilo Giampiccolo、Bernardo Magnini、Ido Dagan 和 Bill Dolan。第三届 PASCAL 文本蕴含识别挑战赛。在 *ACL-PASCAL* 文本蕴含与释义研讨会论文集，第 1–9 页，布拉格，2007 年 6 月。计算语言学协会。URL https://aclanthology.org/W07-1401。Alex Graves、Greg Wayne 和 Ivo Danihelka。神经图灵机。*arXiv* 预印本 *arXiv:1410.5401*，2014 年。Alex Graves、Greg Wayne、Malcolm Reynolds、Tim Harley、Ivo Danihelka、Agnieszka Grabska- Barwi´nska、Sergio Gómez Colmenarejo、Edward Grefenstette、Tiago Ramalho、John Agapiou 等。利用具有动态外部记忆的神经网络进行混合计算。《自然》，538(7626)：471–476，2016 年。R Bar Haim、Ido Dagan、Bill Dolan、Lisa Ferro、Danilo Giampiccolo、Bernardo Magnini 和 Idan Szpektor。第二届 PASCAL 文本蕴含识别挑战赛。在 第二届 *PASCAL* 文本蕴含识别挑战赛研讨会论文集，第 7 卷，2006 年。

Simeng Han、Hailey Schoelkopf、Yilun Zhao、Zhenting Qi、Martin Riddell、Luke Benson、Lucy Sun、Ekaterina Zubova、Yujie Qiao、Matthew Burtell 等。Folio：基于一阶逻辑的自然语言推理。*arXiv* 预印本 *arXiv:2209.00840*，2022年。

Sepp Hochreiter 和 Jürgen Schmidhuber。长短期记忆。神经计算，9(8): 1735–1780，1997。Alfred Horn。关于在代数直和中为真的句子1。《符号逻辑杂志》，16(1):14–21，1951年。Shankar Iyer、Nikhil Dandekar 和 Kornl Csernai。首个 Quora 数据集发布：问题对。发表于*Quora*，2017年1月。网址：https://www.quora.com/q/quoradata/First-Quora-Dataset-Release-Question-Pairs。Zhenzhong Lan、Mingda Chen、Sebastian Goodman、Kevin Gimpel、Piyush Sharma 和 Radu Soricut。ALBERT：一种用于自监督学习语言表示的轻量级 BERT。*arXiv* 预印本 *arXiv:1909.11942*，2019年。Hung Le、Truyen Tran 和 Svetha Venkatesh。神经存储程序记忆。发表于国际学习表征会议，2019年。Hung Le、Truyen Tran 和 Svetha Venkatesh。自注意力联想记忆。发表于国际机器学习会议，第5682–5691页。PMLR，2020。Hector Levesque、Ernest Davis 和 Leora Morgenstern。Winograd模式挑战。发表于第十三届知识表示与推理原理国际会议，2012年。Aitor Lewkowycz、Anders Andreassen、David Dohan、Ethan Dyer、Henryk Michalewski、Vinay Ramasesh、Ambrose Slone、Cem Anil、Imanol Schlag、Theo Gutman-Solo、Yuhuai Wu、Behnam Neyshabur、Guy Gur-Ari 和 Vedant Misra。利用语言模型解决定量推理问题，2022年。网址：https://arxiv.org/abs/2206.14858。Percy Liang。基于Lambda依存的组合语义学。*CoRR*，abs/1309.4408，2013。网址：http://arxiv.org/abs/1309.4408。

Peter J Liu、Mohammad Saleh、Etienne Pot、Ben Goodrich、Ryan Sepassi、Lukasz Kaiser 和 Noam Shazeer。通过概括长序列生成维基百科。发表于国际学习表征会议，2018年。

Yinhan Liu、Myle Ott、Naman Goyal、Jingfei Du、Mandar Joshi、Danqi Chen、Omer Levy、Mike Lewis、Luke Zettlemoyer 和 Veselin Stoyanov。RoBERTa：一种经过稳健优化的 BERT 预训练方法。*arXiv* 预印本 *arXiv:1907.11692*，2019年。

Robin Manhaeve、Sebastijan Dumancic、Angelika Kimmig、Thomas Demeester 和 Luc De Raedt。DeepProbLog：神经概率逻辑编程。神经信息处理系统进展，31，2018年。

Yu Meng、Chenyan Xiong、Payal Bajaj、Paul Bennett、Jiawei Han、Xia Song 等。COCO-LM：通过纠正和对比文本序列进行语言模型预训练。神经信息处理系统进展，34：23102–23114，2021。

Paulius Micikevicius、Sharan Narang、Jonah Alben、Gregory Diamos、Erich Elsen、David Garcia、Boris Ginsburg、Michael Houston、Oleksii Kuchaiev、Ganesh Venkatesh 和 Hao Wu。混合精度训练。在国际学习表征会议，2018年。网址：https: //openreview.net/forum?id=r1gs9JgRZ。

Tomas Mikolov、IlyaSutskever、Kai Chen、Greg S Corrado 和 Jeff Dean。词语和短语的分布式表征及其组合性。神经信息处理系统进展，26，2013。

Pasquale Minervini、Matko Bošnjak、Tim Rocktäschel、Sebastian Riedel 和 Edward Grefenstette。在*AAAI* 人工智能会议论文集中，pp. 5182–5190，2020。

Stephen Muggleton。归纳逻辑编程。*New Gener.Comput.*，8(4):295–318，1991年。

Stephen Muggleton。随机逻辑程序。归纳逻辑编程进展，32：254–264，1996年。

NVIDIA。PyTorch 扩展：用于在 PyTorch 中轻松进行混合精度和分布式训练的工具。在线获取：*https://https://github.com/NVIDIA/apex*，2019年。

Adam Paszke、Sam Gross、Francisco Massa、Adam Lerer、James Bradbury、Gregory Chanan、Trevor Killeen、Zeming Lin、Natalia Gimelshein、Luca Antiga 等。PyTorch：一种命令式风格的高性能深度学习库。神经信息处理系统进展，32，2019年。

Jeffrey Pennington、Richard Socher 和 Christopher D Manning。GloVe：词表示的全局向量。载于 *2014*年自然语言处理实证方法会议论文集（*EMNLP*），第1532–1543页，2014年。

Matthew E. Peters、Mark Neumann、Mohit Iyyer、Matt Gardner、Christopher Clark、Kenton Lee 和 Luke Zettlemoyer。深度上下文化词表示。载于*2018*年北美计算语言学协会分会人类语言技术会议论文集：第一卷（长篇论文），第2227–2237页，新奥尔良，路易斯安那州，2018年6月。计算语言学协会。doi：10.18653/v1/N18-1202。URL：https: //aclanthology.org/N18-1202。

Alec Radford、Karthik Narasimhan、Tim Salimans、Ilya Sutskever 等。通过生成式预训练提升语言理解。*OpenAI*，2018年。

Alec Radford、Jeffrey Wu、Rewon Child、David Luan、Dario Amodei、Ilya Sutskever 等。语言模型是无监督的多任务学习者。*OpenAI*博客，1(8):9，2019年。

Colin Raffel、Noam Shazeer、Adam Roberts、Katherine Lee、Sharan Narang、Michael Matena、Yanqi Zhou、Wei Li、Peter J Liu 等。使用统一文本到文本转换器探索迁移学习的极限。*J. Mach. Learn. Res.*，21(140)：1–67，2020。

Pranav Rajpurkar、Jian Zhang、Konstantin Lopyrev 和 Percy Liang。SQuAD：用于文本机器理解的 100000+ 个问题。载于*2016*年自然语言处理经验方法会议论文集，第 2383–2392 页，得克萨斯州奥斯汀，2016年11月 a。计算语言学协会。doi：10.18653/v1/D16-1264。URL https://aclanthology. org/D16-1264。

Pranav Rajpurkar、Jian Zhang、Konstantin Lopyrev 和 Percy Liang。SQuAD：用于机器理解文本的 100,000+ 个问题。载于 *2016* 年自然语言处理经验方法会议论文集，第 2383–2392 页，2016b。

Tim Rocktäschel 和 Sebastian Riedel。端到端可微证明。神经信息处理系统进展，30，2017年。

Adam Santoro、Sergey Bartunov、Matthew Botvinick、Daan Wierstra 和 Timothy Lillicrap。利用记忆增强神经网络进行元学习。载于国际机器学习会议，第 1842–1850 页。PMLR，2016年。

Adam Santoro、Ryan Faulkner、David Raposo、Jack Rae、Mike Chrzanowski、Theophane Weber、Daan Wierstra、Oriol Vinyals、Razvan Pascanu 和 Timothy Lillicrap。关系循环神经网络。神经信息处理系统进展，31：7299–7310，2018年。

Rico Sennrich、Barry Haddow 和 Alexandra Birch。使用子词单元进行稀有词的神经机器翻译。载于计算语言学协会第*54*届年会，第1715–1725页。计算语言学协会（ACL），2016年。

Peter Shaw、Jakob Uszkoreit 和 Ashish Vaswani。结合相对位置表示的自注意力。载于*NAACL-HLT*（*2*），第464–468页，2018年。网址：https://aclanthology.info/papers/ N18-2074/n18-2074。

Koustuv Sinha、Shagun Sodhani、Jin Dong、Joelle Pineau 和 William L Hamilton。CLUTRR：一个用于从文本进行归纳推理的诊断基准。载于自然语言处理实证方法会议论文集（*EMNLP*），第4506–4515页，2019年。

Richard Socher、Alex Perelygin、Jean Wu、JasonChuang、Christopher DManning、Andrew Y Ng 和 Christopher Potts。情感树库上的语义组合性的递归深度模型。载于*2013*年自然语言处理实证方法会议论文集，第1631–1642页，2013年。

Yi Tay, Mostafa Dehghani, Vinh Q. Tran, Xavier Garcia, Dara Bahri, Tal Schuster, Huaixiu Steven Zheng, Neil Houlsby 和 Donald Metzler。《统一语言学习范式》。*arXiv* 预印本 *arXiv:2205.05131*，2022年。

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser 和 Illia Polosukhin。《注意力就是你所需要的一切》。神经信息处理系统进展，30，2017年。

Petar Veli ˇckovi ´c, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò 和 Yoshua Bengio。《图注意力网络》。载于国际学习表征会议，2018年。

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy 和 Samuel R Bowman。《GLUE：自然语言理解多任务基准与分析平台》。载于第七届国际学习表征会议，*ICLR 2019*，2019年。

Alex Warstadt、Amanpreet Singh 和 Samuel R Bowman。《神经网络可接受性判断》。计算语言学协会汇刊，7:625–641，2019年。

Jason Wei、Xuezhi Wang、Dale Schuurmans、Maarten Bosma、Ed Chi、Quoc Le 和 Denny Zhou。思维链提示能够激发大语言模型的推理能力。*arXiv* 预印本 *arXiv:2201.11903*，2022年。

Adina Williams、Nikita Nangia 和 Samuel Bowman。通过推理进行句子理解的广覆盖挑战语料库。载于*2018*年北美计算语言学协会分会会议论文集：人类语言技术，第*1*卷（长篇论文），第1112–1122页，新奥尔良，路易斯安那州，2018年6月。计算语言学协会。doi：10.18653/v1/N18-1101。URL https: //aclanthology.org/N18-1101。

Zhilin Yang、Zihang Dai、Yiming Yang、Jaime Carbonell、Russ R Salakhutdinov 和 Quoc V Le。XLNet：面向语言理解的广义自回归预训练。神经信息处理系统进展，32，2019年。

Yang You、Jing Li、Sashank Reddi、Jonathan Hseu、Sanjiv Kumar、Srinadh Bhojanapalli、Xiaodan Song、James Demmel、Kurt Keutzer 和 Cho-Jui Hsieh。深度学习的大批量优化：76 分钟内训练 BERT。发表于 国际学习表征会议，2020。网址：https://openreview.net/forum?id=Syx4wnEtvH。

Yukun Zhu、Ryan Kiros、Rich Zemel、Ruslan Salakhutdinov、Raquel Urtasun、Antonio Torralba 和 Sanja Fidler。《对齐书籍与电影：通过观看电影和阅读书籍，迈向故事式视觉解释》。发表于 *IEEE*国际计算机视觉会议（*ICCV*）论文集，2015年12月。

补充材料

A THEFOLNET 架构的补充细节

A.1 关于神经逻辑算子的更多讨论

使用受限核进行肯定前件推理。首先，不难证明，表 2 中的神经逻辑算子可以通过批量矩阵乘法实现（图2），即并行执行矩阵乘法的多个切片以获得输出。不同算子会为矩阵乘法选择各自的归约维度（R-dim）和批处理维度（B-dim）（见表 2）。例如，join 算子将 a-维度选为归约维度，将 h-维度选为批处理维度，从而在 h 上并行执行矩阵乘法。根据 (9)，矩阵乘法的每个切片都可视为基于其自身一组子句进行的独立神经肯定前件推理。例如，对于 join 算子，令 $\mathrm{K}_{\text{join},h}$ 表示一个矩阵，其中 $\mathrm{K}_h(x,a)$ 是其第 (x,a) 个元素。那么，$\mathrm{K}_{\text{join},h}$ 就是第 h 个核切片，它表示 join 算子的一组特定子句。此外，回想一下，核切片每一行的值都刻画了 (8) 右侧的合取模式，其中正值（或负值）决定是否应将原始前提 P_m（或其否定前提$\neg P_m$）用于合取（见第 3.2 节和附录 D）。因此，矩阵乘法中的归约维度应理解为 (8) 中的合取维度。对于 join 算子，这意味着前提 $\mathrm{v}_{hs}(a)$ 的 a- 维度上执行合取；若要推导 $\mathrm{u}_{hs}(x)$，则合取模式由 $\mathrm{K}_{\text{join},h}$ 的第 x 行决定。相比之下，从一元原子到一元原子的完整肯定前件推理（(7) 中的第一个表达式）应在 (h,s,a) 的联合维度上执行合取，其复杂度要高得多。因此，join 算子通过将合取操作限制在 a-维度上，控制肯定前件推理的复杂度。此外，值得注意的是，join 算子对不同 s 的前提应用相同的核切片；也就是说，它在 s-维度上实现了核共享。这是根据第 3.2 节中的原理采取的另一种复杂度降低策略。同样的结论也适用于所有其他算子，它们各自选择自己的

合取维度和核共享维度（见图2）。

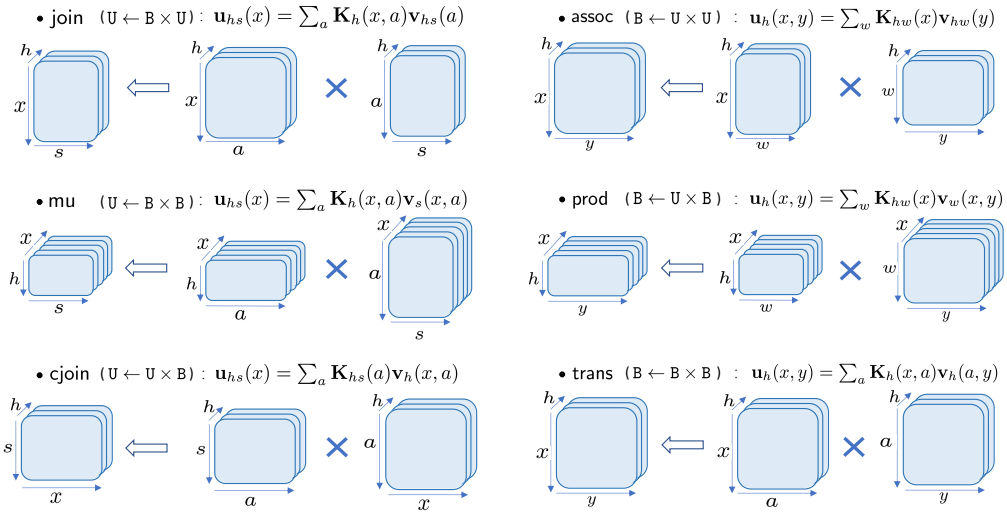


图2：我们的神经逻辑算子可以利用硬件加速器（例如图形处理器和张量处理器）高效地实现为批量矩阵乘法。每个矩阵乘法切片都代表一次独立的神经肯定前件推理，并基于各自的一组子句。

命名协议与示例。 现在我们将介绍逻辑算子的命名协议，并通过具体示例说明每个算子如何在不同方面帮助推理。

• 布尔值：它作用于谓词维度，扮演布尔函数的角色，对一组输入真值进行运算，因此得名布尔运算符。它用于通过某些布尔运算，根据关于同一 x 的其他属性，推导出x 的某个属性。例如，它可以对如下推理进行建模：Edible(x) $\leftarrow$ IsMushroom(x) $\wedge \neg$ IsToxic(x)，其中右侧的合取模式由核决定。

• join：我们将这一算子称为 join 算子，因为它与 λ-DCS（Liang，2013）中的连接操作形式相似。例如，当核表示 IsBornIn(x，y) 时，可用它从前提 CountryIsUSA(y) 推断出 CitizenOfUSA(x)，其中核决定了 CountryIsUSA(y) 在 y 上的合取模式。

• cjoin：由于它只是交换了 join 算子中核和前提的角色，我们将其称为共轭连接运算符。它与连接算子发挥相似的作用。

• mu：我们将其称为 mu 算子，因为它可以被视为 λ-DCS（Liang，2013）中 mu 抽象操作的一种更一般的形式。例如，当核对 ApplyJobAt(x，y) 建模时，可用它从前提 ReceiveOfferFrom(x，y) 推导出 HasAJob(x)，其中核决定了 ReceiveOfferFrom(x，y) 在 y 上的合取模式。

• assoc：我们将其称为关联运算符，因为它可以被视为计算两个向量之间的关联。例如，当核表示 LosAngeles(y) 时，可以用它从 SanFrancisco(x) 推断出 SameStateAs(x, y)，其中核决定了 SanFrancisco(x) 在谓词维度上的合取模式。在这个例子中，谓词维度上只有一个前提原子。一般情况下，特定的合取模式会应用于多个前提原子，以产生输出。

• prod：我们将其称为 prod 算子，因为它类似于在谓词维度上计算乘积。例如，当核对 Graduated(x) 建模时，可以用它从 StudyAt(x,y) 推断出 GraduatedFrom(x, y)，其中核决定了 StudyAt(x,y) 在谓词维度上的合取模式。在这个例子中，谓词维度上只有一个前提原子。一般情况下，特定的合取模式会应用于多个前提原子，以产生输出。

• 转置：我们将其命名为 trans 算子，因为它的形式让人联想到使用传递性质。例如，当核表示 FatherOf(x，z) 时，可以用它根据 ParentOf(z，y) 推出演绎结果 GrandFatherOf(x，y)；其中，核决定了 ParentOf(z，y) 在维度 z 上的合取模式。

请注意，我们没有与类型标注 B $\leftarrow$ B $\times$ U 对应的逻辑算子，因为当核的作用函数选为恒等映射时，它与 prod 算子相同。此外，上述示例进一步表明，核（即子句）本身也是原子，或者可以根据某些原子计算得出。这证明了我们在第 3.3 节中采用的策略是合理的：用 FOLNet 本身的（中间演绎结果）来参数化核。

A.2 与 Transformer架构的比较

图（3）表明，Transformer架构可以理解为 FOLNet 架构的单分支版本，其中仅包含连接算子和关联运算符。

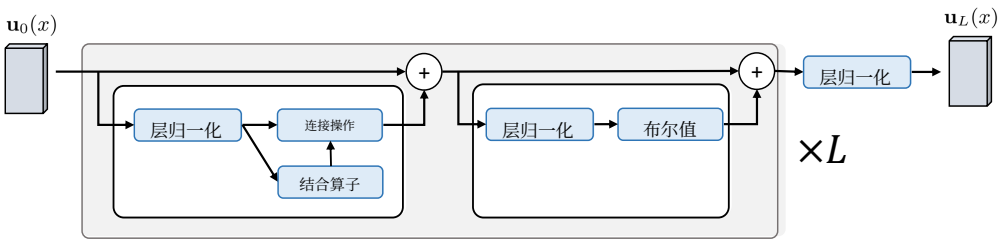


图3：Transformer架构概览。它可以被理解为 FOLNet 架构的单分支版本，仅包含连接算子和关联运算符。

A.3 布尔运算符

在 FOLNet 中，我们不将布尔运算符与其他运算符并行串联，而是将其与其他运算符以级联方式连接。这类似于在 Transformer 中将前馈网络与自注意力模块级联。不过，我们也可以像在 Transformer 中将前馈网络与自注意力并行设置那样，将其与其他运算符并行设置；Chowdhery 等（2022）便采用了这种做法。此外，为了使 FOLNet 能够直接与 Transformer 架构进行比较，我们还为布尔运算符采用了相同的前馈网络架构（即带有 GeLU 单元的多层感知机）。

A.4 输入输出接口

从输出原子计算预训练损失（例如 MLM、NSP 和 SOP） 如前所述，FOLNet 模型将具有与现有 Transformer 模型类似的输入输出接口，因此我们可以通过从 $\{\mathsf{u}_L(x)\}$ 计算预训练损失（例如 MLM、NSP 和 SOP），从而无缝采用现有的预训练损失。回顾一下，$\mathsf{u}_L(x_t)$ 是表示对象（词元） x_t 的派生（高级）属性的向量，其中 $t = 1$、.....、T。对于被掩码词元 x_t，$\mathsf{u}_L(x_t)$ 包含可通过其他词元的输入属性和关系推导出的属性。因此，$\mathsf{u}_L(x_t)$ 中这些推导出的属性可用于预测原始的被掩码词元。例如，我们可以使用线性分类器，再接一个 Softmax 运算符，来计算词汇表上的概率分布以及 MLM 损失。同样地，为了计算 NSP 和 SOP 损失，我们在输入序列开头添加一个特殊的“[CLS]”词元，这样，与“[CLS]”词元对应的 $\mathsf{u}_L(x)$ 就会被输入二分类器，以计算 NSP 或 SOP 损失。在下游任务（例如序列分类、多项选择和序列标注）中，$\mathsf{u}\,\{\mathsf{u}_L(x_t)\}$ 的用法也类似。另一方面，我们没有使用输出二元谓词 $\{\mathsf{u}_L(x_t, y_\tau)\}$ 来计算任何损失。主要原因是，我们希望采用现成的预训练损失，以便针对所提出的模型架构进行一对一比较。由于这些损失大多是为 Transformer架构开发的，而该架构是一种单分支模型，其主路径上只有一元谓词（图 3），因此这些损失主要根据一元属性 $\{\mathsf{u}_L(x_t)\}$ 计算，也就不足为奇了。尽管如此，我们相信，新引入的二元谓词分支可以为利用 $\mathsf{u}_L(x_t, y_\tau)$（即词元关系）开发额外预训练损失开辟新的途径。例如，我们可以随机交换两个词元 x_t 和 x_τ，并使用 $\mathsf{u}_L(x_t, x_\tau)$ 来预测它们是否发生了交换。我们将把为 FOLNet 开发更有效的预训练损失留作未来工作。

中的输入基原子 $\mathcal{B}$ 及其 logit 向量表示 在我们的所有工作中，我们只考虑将纯文本作为输入，这与其他预训练语言模型类似。因此，输入处的基原子应仅编码可直接从纯文本中获取的信息，包括输入序列中的词元 ID（记为 $\mathsf{TokenID}_v(x_t)$）以及任意两个词元之间的相对距离（记为 $\mathsf{RelDist}_d(x_t,\ x_\tau)$）。具体而言，如果词元 x_t 在词汇表中对应第 v 个 ID（$v = 1,\ .....,\ V$），则 $\mathsf{TokenID}_v(x_t) = \mathsf{T}$；否则 $\mathsf{TokenID}_v(x_t) = \mathsf{F}$。当输入序列由一对序列组成时，我们将输入序列构造如下：

“`[CLS]` Sequence #0 `[SEP]` Sequence #1 `[SEP]` `[PAD]` ... `[PAD]`”,

这与 BERT 和其他 Transformer 模型使用的输入格式类似。在这种情况下，我们将引入一个额外的基原子 $\mathsf{SeqID}_s(x_t)$，用于表征词元 x_t 是否属于第 s 个序列（$s = 0,\ 1$）；也就是说，如果词元 x_t 属于第 s 个序列，则 $\mathsf{SeqID}_s(x_t) = \mathsf{T}$，否则为 F。这个原子类似于现有预训练 Transformer 模型中的片段 ID（或 token 类型 ID）。此外，如果 $d = \mathsf{dist}_\Delta(x_t,\ x_\tau)$，则相对位置原子 $\mathsf{RelDist}_d(x_t,\ x_\tau) = \mathsf{T}$；否则为 F。其中，$\mathsf{dist}_\Delta(x_t,\ x_\tau)$ 是一个带截断参数 Δ 的截断距离函数：

$$\mathrm{dist}_\Delta(x_t, x_\tau) = \begin{cases} \max(1-\Delta, \min(\tau - t, \Delta - 1)) & t > 0, \tau > 0 \text{ and } \mathtt{SeqID}_s(x_t) = \mathtt{SeqID}_s(x_\tau) \\ \Delta + 1 & t > 0, \tau > 0 \text{ and } \mathtt{SeqID}_s(x_t) \neq \mathtt{SeqID}_s(x_\tau) \\ \Delta & t = 0, \tau > 0 \\ -\Delta & t > 0, \tau = 0 \\ 0 & t = 0, \tau = 0 \end{cases}.$$

上述截断距离函数会将相对距离限制在 $[-\Delta + 1, \Delta - 1]$ 范围内，前提是两个词元来自同一个序列。此外，每当它们

属于两个不同的序列时，函数也会分配一个特殊距离 ID。换句话说，如果任意两个词元分别属于两个不同的序列，那么它们之间的距离都相同。同样，我们还使用两个特殊距离 ID 来表示 [CLS] 词元（$t = 0$ 或 $\tau = 0$）与其他词元（$\tau > 0$ 或 $t > 0$）之间的距离，从而使所有常规词元与特殊 [CLS] 词元之间的距离（无论方向）都相同。这种相对位置编码（原子）保留了文本序列的平移不变性，并且比 BERT 和 RoBERTa 中使用的绝对位置编码具有更好的泛化能力。值得注意的是，它使 FOLNet 能够泛化到预训练期间未见过的长得多的序列。总之，我们将以下基原子作为 FOLNet 模型的输入：

$$\mathcal{B} = \{\texttt{TokenID}_v(x_t), \texttt{SeqID}_s(x_t), \texttt{RelDist}_d(x_t, x_\tau) | t, \tau = 1, \ldots, T\}. \tag{10}$$

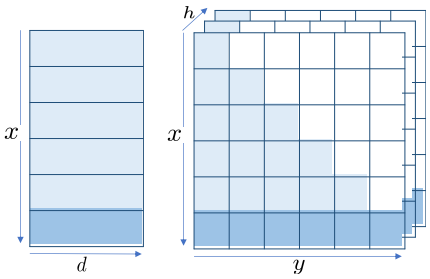
通过在 v 和 s 上叠加 TokenID$_v(x_t)$ 与 SeqID$_s(x_t)$，在 d 上叠加 RelDist$_d(x_t,\ x_\tau)$，并分别使用 1 和 0 表示 T 和 F，我们将得到用于表示 $\mathcal{B}$ 中所有输入原子的 0-1 向量。随后，我们可以通过嵌入查找进一步将它们转换为稠密向量表示 $\{\mathbf{u}_0(x_t),\ \mathbf{u}_0(x_t,\ x_\tau)\}_{t,\ \tau}$。嵌入查找过程也可以理解为寻找一组更细粒度的可学习属性（或关系），用于刻画给定词元 x_t（或词元对 $(x_t,\ x_\tau)$），其中这些属性（或关系）与表 2 中执行肯定前件推理的神经逻辑算子处于相同的 *logit*空间 中。最后，上述基本原子只是我们 FOLNet 模型的一种特定设计，也可以采用其他方式对输入信息进行编码。当除了纯文本之外还有额外信息可用时，我们也可以将其编码为 $\mathcal{B}$ 中新的一元或二元基础原子。

A.5 将 FOLNet 扩展到文本到文本版本

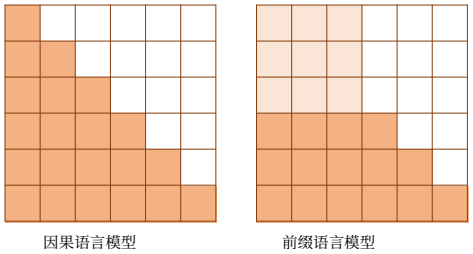
到目前为止，我们主要关注 FOLNet 模型的仅编码器版本。现在我们将展示，如何以相对直接的方式将其扩展为仅解码器版本和编码器-解码器版本。这种扩展对于文本到文本生成任务很有用。

仅解码器 为了开发仅解码器版本的 FOLNet，我们需要让模型以自回归方式生成输出词元。在 FOLNet 中，这要求一元原子 $\{\mathbf{u}(x_t)$、$\mathbf{u}(x_t,\ x_\tau)\}$ 只能根据过去的前提推断：$\{\mathbf{v}(x_\omega)$、$\mathbf{v}(x_\omega,\ x_\nu) : \omega \leq t\}$。此外，我们还进一步限制二元原子，使其具有因果模式，即只要满足 $t < \tau$ 和 $\omega < \nu$，就有 $\mathbf{u}(x_t,\ x_\tau) = 0$ 和 $\mathbf{u}(x_\omega,\ x_\nu) = 0$。图 4(a) 展示了一元原子和二元原子的这些模式。请注意，二元原子的因果结构对应于非零元素的下三角模式。具体而言，深蓝色原子直接负责生成下一个词元（单词），而自回归生成要求模型仅根据浅蓝色原子更新这些原子。我们通过将适当的 0-1 掩码与神经逻辑算子中的二元核和前提相乘，来强制实现这种自回归属性。在表 5 的最后一列中，我们列出了应进行掩码处理的核和前提；图 4(b) 则展示了与仅解码器模型的两种变体对应的掩码：(i) 因果语言模型（CausalLM）（Radford 等，2018；2019；Brown 等，2020；Chowdhery 等，2022），以及 (ii) 前缀语言模型（PrefixLM）（Liu 等，2018；Raffel 等，2020）。CausalLM 对二元原子使用下三角掩码，使信息模式始终保持单向。另一方面，PrefixLM 会对输入片段使用双向掩码（图 4(b) 左上部分），并对输出（目标）片段使用单向掩码（图 4(b) 右下部分）。相应地，PrefixLM 版本的二元原子将采用与图 4(b) 中掩码相似的模式，分别对前缀片段和输出片段之间的关系进行建模。通过这些简单的修改，我们的仅解码器 FOLNet 模型将具有与仅解码器 Transformer 相同的输入输出接口；它可以通过在一元原子 $\mathbf{u}_L(x_t)$ 上使用线性分类器进行预训练，以预测下一个词元。训练完成后，它便可以自回归地生成词元。

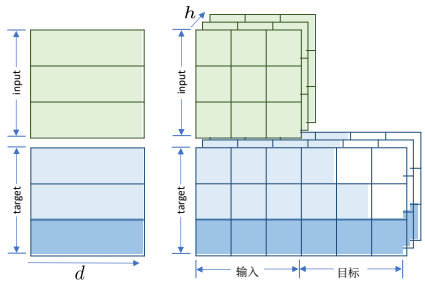
编码器-解码器 对于编码器-解码器变体，我们为编码器和解码器分别使用两组独立的 FOLNet 堆栈，其中每组都具有与图1相同的整体架构，并拥有各自的一组原子（图4(c)中的绿色和蓝色模块）。具体而言，编码器将与我们在正文中前面详细讨论过的仅编码器版本完全相同。同时，解码器部分将类似于仅解码器变体，但还需进行一些额外修改。首先，解码器需要维护一个略有不同版本的二元原子（图4(c)）。具体来说，除了输出词元之间的关系外，解码器还必须对从输入词元指向输出词元的（单向）关系进行建模（即图4(c)中$\mathbf{u}_h(x, y)$ 的左下部分）。这些关系对于根据输入词元推导输出词元至关重要，其作用类似于 Transformer 中的交叉注意力分数。这相应地意味着，我们还需要调整掩码，以分别处理这些关系（见图4(d)）。其次，还应对表 5 中的神经逻辑算子进行略微调整，以纳入来自编码器输出的额外前提原子。例如，join 算子中的前提 **v()** 现在应由编码器输出中的一元原子（经过线性投影）与解码器前提拼接而成。同样，assoc-operator 中的前提 v() 也应由编码器输出（经过线性投影）与解码器前提拼接而成。这两个修改后的算子与 Transformer 解码器中的自注意力和交叉注意力机制具有相似的理念。cjoin 算子类似于 join 算子，只是现在拼接发生在核 K()（与



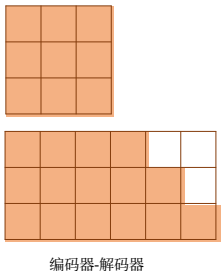
(a) 仅解码器 FOLNet 的原子



(b) 仅解码器 FOLNet 的掩码



(c) 编码器-解码器 FOLNet 的原子



（d）编码器-解码器 FOLNet 的掩码

图 4：将 FOLNet 扩展为文本到文本版本。顶部一行展示了仅解码器版本的原子及掩码策略，底部一行展示了编码器-解码器变体。左侧展示了这些情况下的一元原子和二元原子 $\{\mathbf{u}_d(x),\ \mathbf{u}_h(x,\ y)\}$，其中$d = (h-1)S + s$。绿色和蓝色分别表示编码器和解码器原子的非零模式。绿色原子对应于编码器-解码器变体中的编码器部分。深蓝色表示直接负责在解码器中生成下一个词元的原子。值得注意的是，这些原子仅由浅蓝色原子推导而来，因此整个生成过程保留了自回归特性。右侧的白色和橙色块表示 0-1 掩码位置，其中浅橙色部分对应于 PrefixLM 中的前缀段。同样，编码器-解码器变体分别为编码器和解码器设置了独立的掩码集合，其中编码器部分的掩码被设计为具有双向信息模式。PrefixLM 中的前缀部分也采用了类似的设计。

Sym.	Ops.	类型标注	批次维度	秩维度	logit空间中的神经算子	Mask
b	bool	$\mathrm{U} \leftarrow \mathrm{U} \times \mathrm{U}$	x	w	$\mathbf{u}_{hs}(x) = \sum_w \mathbf{K}_{hw}(x)\mathbf{v}_{ws}(x)$	-
c	c拼接	$\mathrm{U} \leftarrow \mathrm{U} \times \mathrm{B}$	h	a	$\mathbf{u}_{hs}(x) = \sum_a \mathbf{K}_{hs}(a)\mathbf{v}_h(x,\ a)$	$\mathbf{v}_h(x,\ a)$
j	join	$\mathrm{U} \leftarrow \mathrm{B} \times \mathrm{U}$	h	a	$\mathbf{u}_{hs}(x) = \sum_a \mathbf{K}_h(x,\ a)\mathbf{v}_{hs}(a)$	$\mathbf{K}_h(x,\ a)$
m	mu	$\mathrm{U} \leftarrow \mathrm{B} \times \mathrm{B}$	x	a	$\mathbf{u}_{hs}(x) = \sum_a \mathbf{K}_h(x, a)\mathbf{v}_s(x, a)$	$\mathbf{K}_h(x, a)$, $\mathbf{v}_s(x, a)$
a	结合律	$\mathrm{B} \leftarrow \mathrm{U} \times \mathrm{U}$	h	w	$\mathbf{u}_h(x,\ y) = \sum_w \mathbf{K}_{hw}(x)\mathbf{v}_{hw}(y)$	-
p	prod	$\mathrm{B} \leftarrow \mathrm{U} \times \mathrm{B}$	x	w	$\mathbf{u}_h(x,\ y) = \sum_w \mathbf{K}_{hw}(x)\mathbf{v}_w(x, y)$	$\mathbf{v}_w(x,\ y)$
t	转置	$\mathrm{B} \leftarrow \mathrm{B} \times \mathrm{B}$	h	a	$\mathbf{u}_h(x,\ y) = \sum_a \mathbf{K}_h(x, a)\mathbf{v}_h(a, y)$	$\mathbf{K}_h(x, a)$, $\mathbf{v}_h(a, y)$

表 5：FOLNet 仅解码器版本中需要掩码的二元核和前提。在编码器-解码器变体中，其解码器中也会对相似部分的核和前提进行掩码。

输入词元在推导输出词元方面发挥着类似作用，这类似于 Transformer 中的交叉注意力分数。因此，我们还需要调整掩码，以分别处理这些关系（见图4(d)）。其次，还应对表 5 中的神经逻辑算子进行略微调整，以纳入来自编码器输出的额外前提原子。例如，join 算子中的前提 $\mathbf{v}_{hs}(a)$ 现在应由编码器输出中的一元原子（经过线性投影）与解码器前提拼接而成。同样，assoc-operator 中的前提 $\mathbf{v}_{hw}(y)$ 也应由编码器输出（经过线性投影）与解码器前提拼接而成。这两个修改后的算子与 Transformer 解码器中的自注意力和交叉注意力机制具有相似的理念。cjoin 算子类似于 join 算子，只是现在拼接发生在核 $\mathbf{K}_{hs}(a)$（与

线性投影）。mu 算子和 prod 算子与之前相同。同时，解码器的 trans 算子需要按照以下新表达式实现：

$$\mathbf{u}_h(x,y) = \begin{cases} \sum_{a\in\mathcal{T}} \mathbf{K}_h(x,a)\mathbf{v}_h^{\mathrm{D}}(a,y) & y \in \mathcal{T} \\ \sum_{a\in\mathcal{I}} \mathbf{K}_h(x,a)\mathbf{v}_h^{\mathrm{E}}(a,y) + \sum_{a\in\mathcal{T}} \mathbf{K}_h(x,a)\mathbf{v}_h^{\mathrm{D}}(a,y) & y \in \mathcal{I} \end{cases}$$

其中，$\mathcal{T}$和 $\mathcal{I}$分别定义为目标序列和输入序列，核 $\mathbf{K}_h(x,a)$ 是通过对解码器中的二元原子 $\mathbf{u}(x,a)$ 应用线性投影得到的，而前提 $\mathbf{v}_h(a,y)$ 中的上标 E 或 D 表示它们来自编码器还是解码器。值得注意的是，我们观察到，编码器-解码器版本的 FOLNet 在其解码器模块中也保留了双分支架构。这与标准 Transformer 解码器形成鲜明对比，后者采用单分支架构，其主路径上只有一元原子。

B 实验细节

B.1 下游任务概述

GLUE GLUE基准（Wang 等人，2019年）由 9 项任务组成：MNLI（Williams 等人，2018年）、QQP（Iyer 等人，2017年）、QNLI（Rajpurkar 等人，2016a）、SST-2（Socher 等人，2013）、CoLA（Warstadt 等人，2019年）、STS-B（Cer 等人，2017年）、MRPC（Dolan 和 Brockett，2005）、RTE（Dagan 等人，2005；Haim 等人，2006；Giampiccolo 等人，2007；Bentivogli 等人，2009）、WNLI（Levesque 等人，2012）。这些任务涵盖了广泛的自然语言理解任务，例如自然语言推理、释义、语言可接受性和情感分析。我们按照 BERT（Devlin 等人，2018年）采用的相同流程对 FOLNet 模型进行微调。我们不在 WNLI 上评估模型，因为该任务通常需要特殊流程。GLUE 中所有任务的基本介绍（包括其评估指标）见表6。

SQuAD 2.0 SQuAD 2.0（Rajpurkar等人，2016b）是一个基于维基百科构建的抽取式问答数据集。该任务的目标是根据上下文段落预测答案片段。SQuAD 2.0 是一个更新版本，在原始 SQuAD 1.1 版本的基础上新增了 50,000 个无法回答的问题。性能指标为精确匹配（EM）和 F1 分数。

FOLIO FOLIO（Han等人，2022年）是一个包含一阶逻辑推理问题的自然语言推理数据集。给定由一组前提所定义的世界，模型需要判断结论陈述是否正确。该任务被表述为一个三分类问题，标签为“真”“假”和“未知”。我们遵循 Han等人（2022年）采用的相同流程，将问题表述为序列对分类问题。具体而言，我们将所有前提拼接成一个序列（即序列 A），然后再与结论拼接（即序列 B），两个序列之间以 [SEP]分隔。此外，在填充之前，我们在开头添加一个 [CLS] 词元，并在末尾添加一个 [SEP]。这样一来，该任务的格式就与自然语言推理任务（例如 MNLI）相同。表 7（引自 FOLIO论文（Han等人，2022年））展示了 FOLIO数据集中的一个示例，说明解决该问题需要较强的一阶推理能力。该数据集有官方的训练集/验证集/测试集划分，分别包含 1,004/204/227 个示例。提交本文时，测试集尚不可用，因此我们只报告验证集上的性能。

	MNLI	QQP	QNLI	SST-2	CoLA	RTE	MRPC	STS-B
Size	393K	364K	108K	67K	8.5K	2.5K	3.7K	5.7K
Task	推理	相似度	问答/推理	情感	可接受性	推理	释义	相似度
指标	准确率	准确率/F1值	准确率	准确率	马修斯相关系数	准确率	准确率/F1值	皮尔逊相关系数/斯皮尔曼相关系数。
类别数	3	2	2	2	2	2	2	1（回归）

表6：GLUE基准中不同任务的基本信息。

B.2 实现细节和超参数

我们的预训练和微调流程均使用 PyTorch（Paszke 等人，2019年），并基于 Apex 库采用自动混合精度（AMP）学习（Micikevicius 等人，2018年）。

一个基于野生火鸡维基百科页面的 FOLIO 示例	
自然语言前提 1. 野生火鸡有六种：东部野生火鸡、奥塞奥拉野生火鸡、古尔德野生火鸡，梅里亚姆野生火鸡、里奥格兰德野生火鸡和眼斑野生火鸡。 2. Tom 不是一只东部野生火鸡。 3. Tom 不是一只奥塞奥拉野生火鸡。 4. Tom 也不是古尔德野生火鸡、梅里亚姆野生火鸡或里奥格兰德野生火鸡。 5. Tom 是一只野生火鸡。	**自然语言结论 ->标签** A. Tom 是一只眼斑野生火鸡。 ->真 B. Tom 是一只东部野生火鸡。->假 C. Joey 是一只野生火鸡。->未知
一阶逻辑前提 1. $\forall x$(野生火鸡(x) $\rightarrow$ (东部火鸡(x) $\vee$ 奥西奥拉(x) $\vee$ 古尔德火鸡(x) $\vee$ 梅里亚姆火鸡(x) $\vee$ 里奥格兰德火鸡(x) $\vee$ 眼斑火鸡(x))) 2. ¬(野生火鸡(tom) $\wedge$东部火鸡(tom)) 3. ¬(野生火鸡(tom) $\wedge$奥西奥拉(tom)) 4. 野生火鸡(tom) $\rightarrow$ ¬(古尔德火鸡(tom) $\vee$ 梅里亚姆火鸡(tom) $\vee$ 里奥格兰德火鸡(tom)) 5. 野生火鸡(tom)	**一阶逻辑结论 -> 标签** A. 眼斑火鸡(tom) -> 真 B. 东部火鸡(tom) -> 假 C. 野生火鸡($joey$) -> 未知

表 7：我们展示了一个 FOLIO数据集中的示例，该示例直接引自 FOLIO论文（Han等人，2022年）。它表明，解决这个问题需要很强的一阶逻辑推理能力。“NL”代表“自然语言”，“FOL”代表“一阶逻辑”。在实验中，我们仅使用自然语言部分来完成任务。

（NVIDIA，2019年）。预训练时，我们采用大批量训练（批量大小为 131K），并使用 LAMB优化器（You等人，2020）。使用 512 个 V100 GPU，在 Wikipedia + BookCorpus（16GB）上对 FOLNet$_{\text{Base}}$ 进行 8K 步预训练约需 12 小时。在 160GB 数据上对 FOLNet 基础版 进行 128K 步预训练，使用 512 个 V100 GPU 需 7 天。在 160GB 数据上对 FOLNet^{Large} 进行 128K 步预训练，使用 512 个 V100 GPU 需 19 天。对于所有下游任务的微调，我们也采用基于 Apex 的 AMP学习，优化器则使用 Apex 库中的 FusedAdam。

我们在表8中报告了 FOLNet 预训练的超参数。不同下游任务微调所用的超参数列于表9。微调任务的超参数按任务分别搜索，结果取五个随机种子的中位数。

超参数	**FOLNet$_{\text{Base}}$**	**FOLNet$_{\text{Base}}$**	**FOLNet^{Large}**
预训练数据量	16G	160G	160G
层数 (L)	12	12	24
一元隐藏层大小（D_1）	768	768	1024
二元隐藏层大小（D_2）	64	64	64
一元布尔（前馈网络）中间层大小	3072	3072	4096
二进制布尔（前馈网络）中间层大小	256	256	256
注意力头数量（H）	12	12	16
注意力头大小（S）	64	64	64
相对位置编码截断参数（Δ）	64	64	64
丢弃率	0.1	0.1	0.1
注意力丢弃率	0.1	0.1	0.1
预热比例	1%	1%	1%
峰值学习率	1e-2	2e-3	1.6e-3
批次大小	131,072	131,072	131,072
权重衰减	0.01	0.01	0.01
最大步数	8K	128K	128K
学习率衰减	线性	线性	线性
LAMB ϵ	1e-6	1e-6	1e-6
LAMB β_1	0.9	0.9	0.9
LAMB β_2	0.999	0.999	0.999
梯度裁剪	0.0	0.0	0.0
序列长度（T）	128	128	128

表8：不同设置下 FOLNet 预训练的超参数。

C 附加实验

GLUE上的零样本性能我们评估了FOLNet模型在GLUE基准上的零样本性能。具体而言，我们采用Gao等人（2021）提出的相同方法和提示模板进行零样本预测。我们仅考虑FOLNet^{Large} 模型，并将其与

超参数	GLUE-small/big	SQuAD 2.0	FOLIO
最大训练轮数	{5, 10, 20} / {2, 3, 5}	{2, 3}	{60, 80}
基础版（16G）的峰值学习率:	{6e-5, 8e-5 1e-4, 2e-4}	{ 8e-5, 9e-5, 1e-4 }	{ 6e-5, 7e-5, 8e-5}
Base/Large（160G）的峰值学习率:	{1e-5, 2e-5, 3e-5, 4e-5}	{1e-5, 2e-5, 3e-5, 4e-5}	{ 1e-5, 2e-5, 3e-5}
批次大小	{16, 32} / {32}	{16, 32}	{16, 32}
学习率衰减	线性	线性	线性
预热比例	{6%, 25%} / {6%}	6%	{6%, 25%}
序列长度	128	384	512
Adam ϵ	1e-6	1e-6	1e-6
Adam β_1	0.9	0.9	0.9
Adam β_2	0.999	0.999	0.999
梯度裁剪	0.0	0.0	0.0
丢弃率	0.1	0.1	0.1
权重衰减	0.01	0.01	0.01

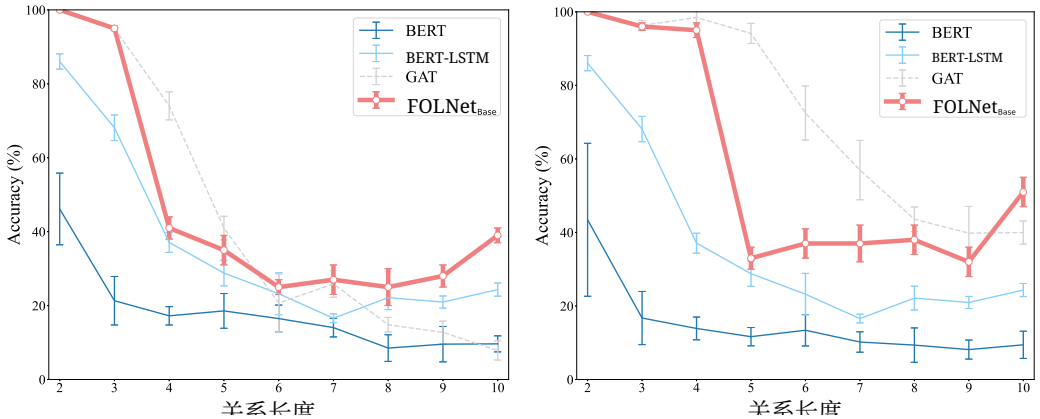
表9：在 GLUE、SQuAD 2.0 和 FOLIO 任务上进行微调的超参数。GLUE-small 指 CoLA、STS-B、MRPC 和 RTE。GLUE-big 指 MNLI、QQP、QNLI 和 SST-2。

模型	方法	数据	MNLI-m/mm Acc	QQP F1	QNLI Acc	SST-2 Acc	CoLA mcc	STS-B 皮尔逊相关系数	MRPC F1	RTE Acc	Avg
多数类猜测	-	-	32.7/33.0	0.0	49.5	50.9	0.0	-	81.2	52.7	33.3
RoBERTa^{Large}	MLM	160G	**50.8**/51.7	49.7	50.8	**83.6**	2.0	-3.2	61.9	51.3	44.3
FOLNet^{Large}	MLM	160G	**50.8/52.6**	**55.4**	**59.7**	79.5	**6.4**	**23.6**	**77.2**	**58.2**	**51.5**

表10：在 GLUE 开发集上的零样本性能。“Acc”表示准确率，“mcc”表示马修斯相关系数，“Pear.”是皮尔逊相关系数的缩写。

RoBERTa^{Large}进行比较，二者在模型规模、预训练数据集和预训练损失方面相近。此外，零样本预测方法也相似：二者都使用各自的掩码语言模型头预测标签词。结果汇总于表10，结果表明，FOLNet^{Large}在8项任务中的7项上优于RoBERTa^{Large}，平均提升7.2分。

CLUTRR 上的性能 为进一步考察推理能力，我们在 CLUTRR（**C**ompositional **L**anguage **U**nderstanding and **T**ext-based **R**elational**R**easoning，组合语言理解与基于文本的关系推理）数据集（Sinha 等人，2019年）上评估 FOL-Net 模型。CLUTRR 是一个半合成诊断基准，旨在评估模型的系统泛化能力。具体而言，对于给定的自然语言故事，模型需要推断两个家庭成员之间的（隐含）关系。为解决这一问题，模型必须从实体中提取关系，并掌握支配这些关系的逻辑规则。CLUTRR 通过在包含逻辑规则的未见组合的故事，以及需要更多推理步骤的故事上测试模型，考察系统泛化能力。遵循 Sinha 等人（2019年）的相同设置，我们首先使用长度为 $k = 2,3$ 和 $k = 2,3,4$ 的子句分别对 FOLNet 模型进行微调，然后在长度为 $k = 2,\ldots,10$ 的子句上评估它们。具体而言，对于每个输入实例，我们将自然语言故事与文本查询拼接起来（由一个 [SEP] 词元分隔），并将问题设为序列对分类任务。此外，我们在开头添加一个 [CLS] 词元，并在末尾追加一个 [SEP] 词元。然而，与 Sinha 等人（2019年）不同，为简单起见，我们不使用特定任务的特殊嵌入替换实体，而是将其视为普通英语单词。实体表示技术将作为未来工作加以研究，这可能会进一步提升我们的性能。我们使用 Adam 优化器对 FOLNet$_{\text{Base}}$ 进行 100 个 epoch 的微调，学习率为 5×10^{-5}，批量大小为 16，预热比例为 6%。在实验中，我们主要与同样使用自然语言输入的基于 Transformer 的基线进行比较。具体而言，我们将结果与 Sinha 等人（2019年）的 BERT$_{\text{Base}}$ 和 BERT-LSTM 进行比较；它们具有相同的模型规模和预训练语料库。此外，还有一系列直接处理 CLUTRR符号输入 的方法，例如图注意力网络（GAT）（Veli$\check{}$ckovi$\acute{}$c 等人，2018年）。由于这些方法直接使用故事背后的结构化逻辑图作为输入，因此避开了解析自然语言故事的困难，通常比基于文本的方法表现得好得多。我们将 GAT 作为参照，以考察 FOLNet 模型能否借助逻辑归纳偏置缩小这一性能差距



(a) 训练子句长度 $k = 2$，3　　(b) 训练子句长度 $k = 2$、3，4

图5：CLUTRR基准上的系统泛化性能。

。借助我们的逻辑归纳偏置。完整结果见图5。首先，我们的 FOLNet$_{\text{Base}}$ 性能显著优于两种基于文本的方法（即 BERT$_{\text{Base}}$ 和 BERT-LSTM），且优势很大。特别是，当测试 k 超出训练集时，我们将 BERT$_{\text{Base}}$ 的性能提升了最高达 8% $\sim$ 41%（绝对提升）。与此同时，当测试 k 在训练期间已出现时，这一优势进一步提升至 54% $\sim$ 81%，而 FOLNet$_{\text{Base}}$ 取得了与 GAT 相当的性能。值得注意的是，当测试 $k = 10$时，FOLNet$_{\text{Base}}$ 的性能甚至显著优于 GAT（高出 10% $\sim$ 20%）。这些结果证实了编码逻辑归纳偏置的益处。

D 神经肯定前件

本节将展示，使用形式为 (8) 的子句进行的一般肯定前件推理，可以表示为矩阵乘法，后接可选的非线性激活函数。

假设 P_1、.....、P_M 是前提原子，Q 是头部（即结论）原子。我们将集合 $\mathcal{M} = \{1, \ldots, M\}$ 划分为三个互不重叠的集合 $\mathcal{M}_+$、$\mathcal{M}_0$ 和 $\mathcal{M}_-$，并定义如下的一般子句（为简化记法，我们省略了式 (8) 中的下标 n）：

$$Q \leftarrow \left(\bigwedge_{m \in \mathcal{M}_+} P_m \right) \bigwedge \left(\bigwedge_{m \in \mathcal{M}_-} \neg P_m \right). \tag{11}$$

注意，表达式 (11) 可用于表示这样一个一般子句：从 $\{P_1, \ldots, P_M, \neg P_1, \ldots, \neg P_M\}$ 中任意子集的前提推导出 Q。其中，集合 $\mathcal{M}_+$ 和 $\mathcal{M}_-$ 分别包含所选原始前提和否定前提的索引，而集合 $\mathcal{M}_0$ 则包含被忽略前提的索引。因此，每一种特定的划分 $\mathcal{M} = \mathcal{M}_+ \cup \mathcal{M}_0 \cup \mathcal{M}_-$ 也对应一个唯一的子句。要使用子句 (11) 进行肯定前件推理，我们可以分两步进行：(i) 根据以下内容计算子句体（即式 (11) 的右侧）

$$P \equiv \left(\bigwedge_{m \in \mathcal{M}_+} P_m \right) \bigwedge \left(\bigwedge_{m \in \mathcal{M}_-} \neg P_m \right), \tag{12}$$

以及（ii）应用通用的肯定前件推理规则：$Q \Leftarrow \{Q \leftarrow P$和 $P\}$。在接下来的两个小节中，我们将分别推导实现这两个步骤的神经运算。具体而言，在 ProbLog 设置下，神经运算将在 logit空间中进行。

D.1 体原子组合的矩阵乘法

为推导逻辑表达式 (12) 的运算，我们将原子$P, P_1, \ldots, P_M$为真的概率定义为以下逻辑形式：

$$\Pr\{P = \mathsf{T}\} = \frac{1}{1 + e^{-z}}, \qquad \Pr\{P_m = \mathsf{T}\} = \frac{1}{1 + e^{-v_m}}, \quad m = 1, \ldots, M, \tag{13}$$

其中， z 和 v_m 分别是与原子 P 和 P_m 相对应的 logit。此外，对于每个 m，我们还引入一个变量 $W_m = +1, 0, -1$，用于分别指示 m 是否属于 $\mathcal{M}_+$、$\mathcal{M}_0$ 和 $\mathcal{M}_-$。然后，(12) 可由以下条件概率刻画：

$$\Pr\{P=\mathsf{T}|\mathcal{W},\ \mathcal{P}\} = \begin{cases} 1 & \bigwedge_{m=1}^{M} \big[\quad W_m=1 \wedge P_m= \quad \vee \ W_m=-1 \wedge P_m= \quad \vee \ W_m=0 \ \big] \\ 0 & \end{cases} \tag{14}$$

T)(F)( ) 否则 ’

其中 $\mathcal{W} \triangleq \{W_1, \ldots, W_M\}$ 和 $\mathcal{P} \triangleq \{P_1, \ldots, P_M\}$。我们进一步假设，变量 $P_1, \ldots, P_M$ 彼此独立，且也独立于 $W_1, \ldots, W_M$。此外，对于给定的 $\mathcal{W} = \{W_1, \ldots, W_M\}$ 集合，我们引入以下关于 $P_1, \ldots, P_M$ 的随机事件：

$$\mathcal{E} \triangleq \left\{ (P_1, \ldots, P_M) \colon \bigwedge_{m=1}^{M} \big[(W_m = 1 \wedge P_m = \mathsf{T}) \vee (W_m = -1 \wedge P_m = \mathsf{F}) \vee (W_m = 0) \big] \right\}, \tag{15}$$

以及它的指示函数 $\mathbb{I}((P_1, \ldots, P_M) \in \mathcal{E})$。指示函数 $\mathbb{I}(\cdot)$ 在其括号内的表达式为真时取 1，否则取 0。然后，我们可以推导出 $\Pr\{P = \mathsf{T}|\mathcal{W}\}$：

$$\begin{aligned}
\Pr\{P = \mathsf{T}|\mathcal{W}\} &= \sum_{\mathcal{P}} \Pr\{P = \mathsf{T}|\mathcal{W}, \mathcal{P}\} \Pr\{P_1, \ldots, P_M|\mathcal{W}\} \\
&\overset{(a)}{=} \sum_{\mathcal{P}} \Pr\{P = \mathsf{T}|\mathcal{W}, \mathcal{P}\} \Pr\{P_1, \ldots, P_M\} \\
&\overset{(b)}{=} \sum_{\mathcal{P}} \mathbb{I}\big((P_1, \ldots, P_M) \in \mathcal{E}\big) \Pr\{P_1, \ldots, P_M\} \\
&\overset{(c)}{=} \sum_{\mathcal{P}} \mathbb{I}\big((P_1, \ldots, P_M) \in \mathcal{E}\big) \prod_{m=1}^{M} \Pr\{P_m\} \\
&\overset{(d)}{=} \sum_{(P_1, \ldots, P_M) \in \mathcal{E}} \prod_{m \in \mathcal{M}_+} \Pr\{P_m\} \prod_{m \in \mathcal{M}_-} \Pr\{P_m\} \prod_{m \in \mathcal{M}_0} \Pr\{P_m\} \\
&\overset{(e)}{=} \sum_{(P_1, \ldots, P_M) \in \mathcal{E}} \prod_{m \in \mathcal{M}_+} \Pr\{P_m = \mathsf{T}\} \prod_{m \in \mathcal{M}_-} \Pr\{P_m = \mathsf{F}\} \prod_{m \in \mathcal{M}_0} \Pr\{P_m\} \\
&\overset{(f)}{=} \prod_{m \in \mathcal{M}_+} \Pr\{P_m = \mathsf{T}\} \prod_{m \in \mathcal{M}_-} \Pr\{P_m = \mathsf{F}\} \sum_{\{P_m \colon m \in \mathcal{M}_0\}} \prod_{m \in \mathcal{M}_0} \Pr\{P_m\} \\
&\overset{(g)}{=} \prod_{m \in \mathcal{M}_+} \Pr\{P_m = \mathsf{T}\} \prod_{m \in \mathcal{M}_-} \Pr\{P_m = \mathsf{F}\} \\
&\overset{(h)}{=} \prod_{m \in \mathcal{M}_+} \frac{1}{1 + e^{-v_m}} \prod_{m \in \mathcal{M}_-} \frac{1}{1 + e^{v_m}} \\
&\overset{(i)}{=} \prod_{m=1}^{M} \frac{1}{(1 + e^{-v_m})^{I_m^+}} \prod_{m=1}^{M} \frac{1}{(1 + e^{v_m})^{I_m^-}} \\
&= \prod_{m=1}^{M} \frac{1}{(1 + e^{-v_m})^{I_m^+}} \cdot \frac{1}{(1 + e^{v_m})^{I_m^-}} \\
&\overset{(j)}{\leq} \frac{1}{1 + e^{-\sum_{m=1}^{M} (I_m^+ - I_m^-) v_m}}
\end{aligned} \tag{16}$$

其中，步骤 (a) 利用了 $P_1, \ldots, P_M$ 与 $W_1, \ldots, W_M$ 之间的独立性；步骤 (b) 代入条件概率的定义 (14)；步骤 (c) 利用了假设，即 $P_1, \ldots, P_M$ 彼此独立；步骤 (d) 代入分解式 $\mathcal{M} = \mathcal{M}_+ \cap \mathcal{M}_0 \cap \mathcal{M}_-$；步骤 (e) 根据事件 $\mathcal{E}$ 的定义 (15) 得出（即在事件 $\mathcal{E}$ 中，对于 $m \in \mathcal{M}_+$，$P_m =$ 为 T；对于 $m \in \mathcal{M}_-$，$P_m =$ 为 F；而 P_m 是 { 中的任意值 T,F}）；步骤 (f) 提出各求和项中的公因子，并保留对 $\{P_m$ 中所有可能取值的求和：$m \in \mathcal{M}_0\}$（根据定义 (15)）；步骤 (g) 利用了事件 $\{P_m :$ $m \in \mathcal{M}_0\}$ 的总概率为 1 这一事实；步骤 (h) 代入 (13) 中的第二个逻辑斯蒂表达式；步骤

(i) 引入变量 $I_m^+ \triangleq \mathbb{I}(m \in \mathcal{M}_+)$ 和 $I_m^- \triangleq \mathbb{I}(m \in \mathcal{M}_-)$；步骤 (j) 中的上界通过展开分母中的乘法并去掉所有交叉项得到，这些交叉项均为非负值。请注意，当（在所有 $I_1^+, \ldots, I_M^+$ 和 $I_1^-, \ldots, I_M^-$ 中）只有一个唯一的 I_m^+ 或 I_m^- 非零时，上界 (16) 是紧的。上述结果 (16) 是以 $W_1, \ldots, W_M$ 的某个特定取值为条件的，而它们是离散变量。因此，很难以可微方式直接学习这些变量。为解决这一问题，我们进一步假设 $W_1, \ldots, W_M$ 是随机变量。然后，对 $W_1, \ldots, W_M$ 取期望，得到：

$$\Pr\{P = \mathsf{T}\} \leq \mathsf{E}\left\{ \frac{1}{1 + e^{-\sum_{m=1}^{M} (I_m^+ - I_m^-) v_m}} \right\}, \tag{17}$$

其中，期望中的随机性来自 I_m^+ 和 I_m^-，而它们又来自 W_1、.....、W_M。现在，我们采用一种简单而有效的策略来近似右侧，以获得可微实现。具体而言，我们使用以下近似：

$$\mathsf{E}\left[\frac{1}{1 + e^{X}} \right] \approx \frac{1}{1 + e^{\mathsf{E}X}}, \tag{18}$$

当 X 的分布更加集中于单个峰值附近时（即变得确定），该近似会更加准确。使用上述近似，我们得到

$$\Pr\{P = \mathsf{T}\} \approx \frac{1}{1 + e^{-\sum_{m=1}^{M} (\kappa_m^+ - \kappa_m^-) v_m}} \tag{19}$$

其中，$\kappa_m^+ \triangleq \mathsf{E}[I_m^+] = \Pr\{W_m = +1\}$，并且 $\kappa_m^- \triangleq \mathsf{E}[I_m^-] = \Pr\{W_m = -1\}$。将公式（13）中的第一个表达式代入公式（19）的左侧，我们可以得出，$\Pr\{P = \mathsf{T}\}$ 的 logit 可近似为：

$$z = \sum_{m=1}^{M} (\kappa_m^+ - \kappa_m^-) v_m = \langle \boldsymbol{\kappa}, \mathbf{v} \rangle \tag{20}$$

其中，$\boldsymbol{\kappa}$ 和 $\mathbf{v}$ 分别表示将 $\kappa_m^+ - \kappa_m^-$ 和 v_m 作为其 m 个元素收集起来的向量。上述表达式表明，原子 P 的 logit z 可以通过在两个向量 $\boldsymbol{\kappa}$ 和 $\mathbf{v}$ 之间进行简单的内积运算来计算。进一步回顾可知，W_1、.....、W_M 的不同取值对应于 $\mathcal{M} = \mathcal{M}_+ \cup \mathcal{M}_0 \cup \mathcal{M}_-$ 的不同划分，而这些划分进一步为公式（12）中的 P 定义了不同的逻辑表达式。由于 $\boldsymbol{\kappa}$ 是一个将 $\Pr\{W_m = +1\} - \Pr\{W_m = -1\}$ 收集为其第 m 个元素的向量，因此也可以将其视为逻辑表达式（12）的特征向量，该逻辑表达式是子句（11）的体。因此，$\boldsymbol{\kappa}$ 也是子句（11）的特征向量，因为它决定了体原子之间的逻辑组合。当我们有多个逻辑表达式 $\boldsymbol{\kappa}_1$、.....、$\boldsymbol{\kappa}_N$，它们由原子 P_1、.....、P_M 组合成 N 个逻辑表达式时，可以通过以下矩阵乘法计算输出逻辑表达式的 logit：

$$\mathbf{z} = \mathbf{K}\mathbf{v}, \tag{21}$$

其中 $\mathbf{z}$ 是表示 N 输出原子的 logit 向量，而 $\mathbf{K}$ 是一个矩阵，其第 n 行由 $\boldsymbol{\kappa}_n$ 组成 .

D.2 蕴含运算的非线性激活

在本小节中，我们继续在 logit 空间中推导通用肯定前件推理的神经算子，$Q \Leftarrow \{Q \leftarrow P$ 和 $P\}$。同样，我们考虑 ProbLog 设置，其中原子 P和 Q 分别被赋予概率 $\Pr\{P = \mathsf{T}\}$ 和 $\Pr\{Q = \mathsf{T}\}$，用以表示它们为真的可能性。令 $C \triangleq (Q \leftarrow P)$ 为该子句。我们的目标是根据 $\Pr\{P = \mathsf{T}\}$，推断结果的概率 $\Pr\{Q = \mathsf{T}\}$，已知事实为 $C = \mathsf{T}$。

为此，我们首先推导条件概率 $\Pr\{Q = \mathsf{T}|P, C = \mathsf{T}\}$。我们首先需要根据逻辑蕴含“$\leftarrow$”的定义（Andrews，2013）建立条件概率 $\Pr\{C = \mathsf{T}|P, Q\}$，然后应用贝叶斯定理。注意，$C = (Q \leftarrow P)$ 定义为$C \triangleq (Q \vee \neg P)$，可表示为以下条件概率：

$$\Pr\{C = \mathsf{T}|P, Q\} = \begin{cases} 0 & \text{if } P = \mathsf{T} \text{ and } Q = \mathsf{F} \\ 1 & \text{otherwise} \end{cases}. \tag{22}$$

也就是说，只有当前提 P 为真且结论 Q 为假时，子句 C 才为假。接下来，我们进一步假设 $\Pr\{Q=\mathsf{T}|P\}=\Pr\{Q=\mathsf{F}|P\}=1/2$。这一假设的直觉是：当我们只得到输入前提 P（不含 C）时，对结果 Q 没有任何先验知识。然后，根据贝叶斯定理，我们有

$$\begin{aligned}\Pr\{Q=\mathsf{T}|P,C=\mathsf{T}\}&=\frac{\Pr\{Q=T,C=\mathsf{T}|P\}}{\Pr\{C=\mathsf{T}|P\}}\\&=\frac{\Pr\{Q=\mathsf{T}|P\}\Pr\{C=\mathsf{T}|P,Q=\mathsf{T}\}}{\sum_{Q\in\{\mathsf{F},\mathsf{T}\}}\Pr\{Q|P\}\Pr\{C=\mathsf{T}|P,Q\}}\\&\overset{(a)}{=}\frac{\Pr\{C=\mathsf{T}|P,Q=\mathsf{T}\}}{\Pr\{C=\mathsf{T}|P,Q=\mathsf{T}\}+\Pr\{C=\mathsf{T}|P,Q=\mathsf{F}\}}\\&\overset{(b)}{=}\begin{cases}1 & \text{if } P=\mathsf{T}\\0.5 & \text{if } P=\mathsf{F}\end{cases},\end{aligned}\tag{23}$$

其中，步骤 (a) 和 (b) 分别代入 $\Pr\{Q=\mathsf{T}|P\}=\Pr\{Q=\mathsf{F}|P\}=1/2$ 和 (22)。

接下来，我们在假设前提 P 与当前肯定前件推理步骤中使用的子句 C 相互独立的条件下，推导 $\Pr\{Q=\mathsf{T}|C=\mathsf{T}\}$。我们有

$$\begin{aligned}\Pr\{Q=\mathsf{T}|C=\mathsf{T}\}&=\sum_{P\in\{\mathsf{T},\mathsf{F}\}}\Pr\{Q=\mathsf{T}|P,C=\mathsf{T}\}\Pr\{P|C=\mathsf{T}\}\\&\overset{(a)}{=}\sum_{P\in\{\mathsf{T},\mathsf{F}\}}\Pr\{Q=\mathsf{T}|P,C=\mathsf{T}\}\Pr\{P\}\\&\overset{(b)}{=}\Pr\{P=\mathsf{T}\}+\frac{1}{2}\Pr\{P=\mathsf{F}\}\\&=\frac{1}{2}+\frac{1}{2}\Pr\{P=\mathsf{T}\},\end{aligned}\tag{24}$$

其中，步骤 (a) 使用了 P 与 C 相互独立这一假设，步骤 (b) 则代入 (23)。

最后，我们在 logit空间中推导条件概率 (24) 的表达式。具体而言，设 q 和 p 分别是 Q 和 P的 logit，它们对相应的概率进行参数化：

$$\Pr\{P=\mathsf{T}\}=\frac{1}{1+e^{-z}},\qquad\Pr\{Q=\mathsf{T}|C=\mathsf{T}\}=\frac{1}{1+e^{-u}}.\tag{25}$$

将 (25) 中的第一个表达式代入 (24)，再经过一些简单的代数运算，我们得到 n

$$\Pr\{Q=\mathsf{T}|C=\mathsf{T}\}=\frac{1}{1+e^{-\ln(1+2e^{z})}}.\tag{26}$$

将式 (26) 的右侧与式 (25) 的右侧进行比较，可得 P 的 logit 为

$$u=\ln(1+2e^{z}).\tag{27}$$

v上述表达式表明，要在 logit 空间中实现肯定前件，我们只需对输入前提 logit p 应用上述简单的非线性激活函数。为了进一步了解上述 logit 空间中的肯定前件推理，我们对上述非线性激活函数进行进一步分析。注意，$\ln(1+e^{x})$ 是非负凸函数。利用詹森不等式和非负性，我们可以证明，上述非线性激活函数的下界为 $\ln(1+2e^{x})\geq\mathrm{ReLU}(x+\ln 2)$，其中右侧确实是左侧的良好近似。因此，我们可以在 logit 空间中应用（平移后的）ReLU 函数，以低得多的计算复杂度实现一般的肯定前件。

D.3 统合一切

给定 N 个形式如（11）的不同子句和 M 个输入前提 P_1、.....、P_M，可以通过以下方式（近似地）实现使用肯定前件规则对 N 个结果原子 Q_1、.....、Q_N 进行演绎：

$$\mathbf{z}=\mathbf{K}\mathbf{v}\tag{28}$$

$$\mathbf{u} = \ln(1 + 2e^{\mathbf{z}}), \tag{29}$$

其中，$\mathbf{K}$ 是一个 $N \times M$ 矩阵，其第 n 行为第 n 个子句的特征向量；$\mathbf{v}$是一个 M 维向量，其中包含输入前提 P_1、.....、P_M的 logit；$\mathbf{u}$是一个 N 维向量，其中包含输出原子 Q_1、.....、Q_N的 logit。请注意，使用 N 个子句进行肯定前件推理时，可以通过并行计算来实现：首先将矩阵 $\mathbf{K}$ 左乘 logit 向量 $\mathbf{v}$，然后经过一种特殊的逐元素非线性激活函数（可用平移后的 ReLU 函数近似）。因此，本文将 子句核（简称 核）称为矩阵 $\mathbf{K}$。此外，我们注意到，当（11）中的蕴含关系“←”被逻辑等价关系“≡”取代时，原子句（11）变为（12），因此可以省去（29）中的激活函数。最后，不难证明，（7）中四类规则的神经肯定前件推理（28）可等价地表示为:

$$\mathbf{u}(x) = \sum_{a} \mathbf{K}_{\mathtt{UU}}(x, a)\mathbf{v}(a) \tag{30}$$

$$\mathbf{u}(x) = \sum_{a,b} \mathbf{K}_{\mathtt{UB}}(x, a, b)\mathbf{v}(a, b) \tag{31}$$

$$\mathbf{u}(x, y) = \sum_{a} \mathbf{K}_{\mathtt{BU}}(x, y, a)\mathbf{v}(a) \tag{32}$$

$$\mathbf{u}(x, y) = \sum_{a,b} \mathbf{K}_{\mathtt{BB}}(x, y, a, b)\mathbf{v}(a, b), \tag{33}$$

其中，$\mathbf{K}_{\mathtt{UU}}(x、\ a)$、$\mathbf{K}_{\mathtt{UB}}(x、\ a、\ b)$、$\mathbf{K}_{\mathtt{BU}}(x、\ y、\ a)$ 和 $\mathbf{K}_{\mathtt{BB}}(x、\ y、\ a、\ b)$ 分别是与 $\mathcal{R}_{\mathtt{UU}}$、$\mathcal{R}_{\mathtt{UB}}$、$\mathcal{R}_{\mathtt{BU}}$ 和 $\mathcal{R}_{\mathtt{BB}}$ 相对应的 $D_1 \times D_1$、$D_1 \times D_2$、$D_2 \times D_1$ 和 $D_2 \times D_2$ 矩阵。

现有相对位置编码的组合

现在我们展示如何利用 m-算子和 p-算子组合出现有的相对位置编码（在本文中记为 RPE⋆）。首先，我们使用 Shaw 等人（2018）的原始记法，写出现有 RPE⋆ 的表达式。具体而言，RPE⋆ 根据以下表达式计算（多头）自注意力输出（为简洁起见，省略自注意力头 h 的记法）：

$$z_i = \sum_{j=1}^{n} \alpha_{ij}(x_j W^V + a_{ij}^V) = \sum_{j=1}^{n} \alpha_{ij} x_j W^V + \sum_{j=1}^{n} \alpha_{ij} a_{ij}^V \tag{34}$$

$$e_{ij} = \frac{x_i W^Q (x_j W^K + a_{ij}^K)^T}{\sqrt{d_z}} = \frac{x_i W^Q (x_j W^K)^T + x_i W^Q (a_{ij}^K)^T}{\sqrt{d_z}}, \tag{35}$$

其中，z_i 是第 i 个词元的自注意力输出，e_{ij} 是未归一化的自注意力分数，α_{ij} 是自注意力概率（通过对 e_{ij} 应用 Softmax 并沿 j 进行归一化得到），x_i 是注意力输入中第 i 个词元的向量，W^Q/W^K/W^V分别是自注意力操作中查询/键/值的权重矩阵，n 是序列长度，d_z 是每个头的维度。值得注意的是，a_{ij}^V 和 a_{ij}^K 是可学习的相对位置嵌入向量，定义如下:

$$a_{ij}^K = w_{\mathrm{clip}(j-i,k)}^K$$
$$a_{ij}^V = w_{\mathrm{clip}(j-i,k)}^V$$
$$\mathrm{clip}(x, k) = \max(-k, \min(k, x)).$$

也就是说，a_{ij}^V 和 a_{ij}^K 是与 i号和 j号词元之间的（经截断的）相对距离相对应的嵌入向量，其中 k 是截断阈值。因此，（34）–（35）中的第二项都是渗入自注意力机制计算过程的 RPE⋆ 偏置项。下面我们将说明，这两项可以看作表 2 中我们的 m-算子和 p-算子的退化形式。为方便起见，我们首先重写这两个算子的表达式:

$$\mathsf{m}: \quad \mathbf{u}_{hs}(x) = \sum_{a} \mathbf{K}_h(x, a)\mathbf{v}_s(x, a) \tag{36}$$

$$\mathsf{p}: \quad \mathbf{u}_h(x,y) = \sum_w \mathbf{K}_{hw}(x)\mathbf{v}_w(x,y). \tag{37}$$

我们首先说明，（34）中的第二项可以由 m-算子（36）构成。为此，请注意，（36）中的词元 x 和 a 分别可以对应为（34）中的 i 和 j。此外，核 $\mathrm{K}_h(x,a)$ 和前提 $\mathrm{v}_s(x,a)$ 分别可以对应为自注意力概率 α_{ij} 和相对位置嵌入向量 a_{ij}^V。因此，除我们的"头"索引 h 外，m-算子与（34）中的第二项具有相同的形式。同样地，我们可以说明，（35）中的第二项可以由 p-算子构成。注意，（35）中的第二项是向量 $x_i W^Q$ 和向量 a_{ij}^K 之间的内积。令 $\mathrm{k}_h(x)$ 为一个向量，其中 $\mathrm{K}_{hw}(x)$ 是其第 w 个元素；令 $\mathrm{v}(x,y)$ 为一个向量，其中 $\mathrm{v}_w(x,y)$ 是其第 w 个元素。那么，p- 算子也可以看作向量 $\mathrm{k}_h(x)$ 和 $\mathrm{v}(x,y)$ 之间的内积。将 $\mathrm{k}_h(x)$ 和 $\mathrm{v}(x,y)$ 分别对应为向量 $x_i W^Q$ 和 a_{ij}^K 后，我们可以得出，（35）中的第二项可以由 p-算子构成。除这些相似之处外，我们的 m-算子和 p-算子在以下方面比原始 RPE⋆ 更加通用且功能更强。首先，回顾第 3.3 节，核 K 和 v 均由 FOLNet 参数化（通过对中间表示 $\{\mathrm{u}_l(x)、\mathrm{u}_l(x,y)\}$ 进行线性投影，随后可能再应用激活函数）。因此，我们的 $\mathrm{v}_s(x,a)$ 和 $\mathrm{v}_w(x,y)$ 取决于实例，并且在不同输入实例之间有所不同。相比之下，（34）–（35）中的 a_{ij}^V 和 a_{ij}^K 是静态嵌入向量，对所有输入实例都相同。此外，我们的 m-算子和 p-算子以逐层方式自适应；也就是说，每一层都会根据自身的中间表示$\{\mathrm{u}_l(x)、\mathrm{u}_l(x,y)\}$自适应地计算各自的 m-算子和 p-算子，而在 RPE⋆ 中，向量 a_{ij}^V 和 a_{ij}^K 通常在各层之间完全相同。因此，现有的 RPE⋆ 操作可以看作能够由我们的 m-算子和 p-算子构成的退化特例。