Entity Structure Within and Throughout: Modeling Mention Dependencies for Document-Level Relation Extraction

Xu, B.; Wang, Q.; Lyu, Y.; Zhu, Y.; and Mao, Z.

摘要

实体作为关系抽取任务中的基本元素，具有一定的结构。在这项工作中，将这种结构表述为提及对之间的独特依赖关系。本文提出了 SSAN，它将这些结构依赖性纳入标准的自我注意机制，并贯穿整个编码阶段。具体来说，在每个自注意构建块内设计了两个可选的转换模块，以产生注意偏差，从而自适应地调整其注意力流。实验证明了所提出的实体结构的有用性和 SSAN 的有效性，它显著优于竞争基准，在文档级关系提取数据集上取得了最新的成果。

关键词：实体；关系抽取；实体结构

1 引言

1.1 关系抽取

关系提取是指从文本中提取语义关系，这种语义关系通常发生在两个或多个实体之间。这些关系可以是不同类型的。” Paris is in France ” 表示巴黎与法国之间的” is in ” 关系。这可以用三元组 (Paris, is in, France) 来表示。

关系提取旨在从原始文本中发现关系事实，作为结构化知识。它对于知识库构建、问题解答和生物医学文本分析等许多实际应用都非常重要。早期的研究主要将这一问题限制在句内和单个实体对设置中，许多最近的工作已经努力将其扩展到文档级文本中，使其成为一项更实际但也更具挑战性的任务。文档级文本包含大量的实体，这些实体在多次提及中被定义，它们之间自然表现出有意义的依赖关系

1.2 文档级关系抽取

图 1给出了最近提出的文档级关系提取数据集 DocRED（Yao et al.2019）的一个示例，该数据集说明了几个提到的依赖关系：（1）Coming Down Again 和第一句中的 the Rolling Stones 是密切相关的，因此我们可以根据他们的本地上下文来识别 R1：表演者（2）第一句的 Coming Down Again、第二句的 It、第三句的 The song 指代的是同一个实体，因此需要将他们一同考虑和推理（3）第一句中的 the Rolling Stones 和第二句中的 Mick Jagger 虽然没有显示出直接的联系，但可以通过两个相关的提及来联系起来：Coming Down Again 和 It，这对于预测两个实体之间的目标关系 R2：成员有很大影响。the Rolling Stones 和 Nicky Hopkins 之间也存在类似的依赖关系，这有助于识别 R3：成员。

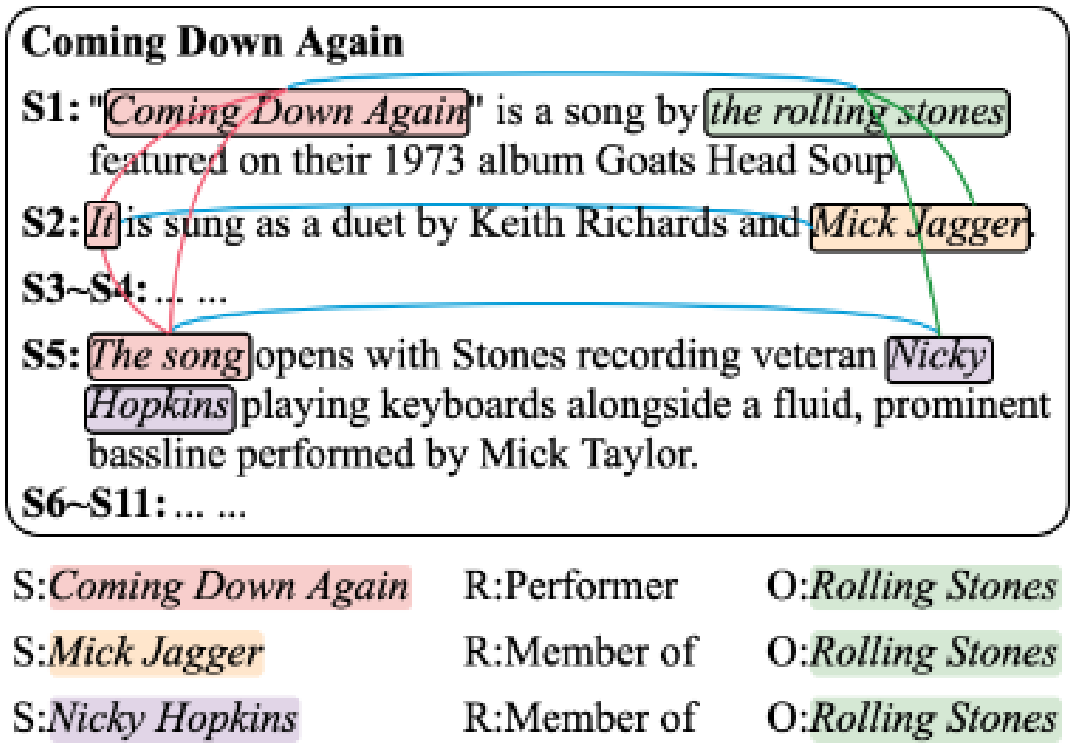

图 1: DocRED 数据集示例

2 相关工作

2.1 基于共引用依赖关系

以前的许多工作都试图利用这种实体结构，特别是共引用依赖关系。通常使用的技巧是将核心信息简单地编码为额外特征，并将其集成到初始输入词嵌入中。Verga、Strubell 和 McCallum（2018）提出了一个多实例学习的改编版本，以聚合来自核心引用的预测。其他人也直接将平均池应用于共参考提及的表示（Yao 等人，2019）。总之，这些启发式技术仅仅使用在预或后处理阶段，因此建模能力有限。此外，除了共引用之外，它们中的大多数都没有包含其他有意义的依赖项。

2.2 基于图的方法

最近，基于图的方法在建模实体结构方面显示出了巨大的优势（Sahu 等人，2019；Christopoulou、Miwa 和 Ananiadou 2019；Nan 等人，2020）。通常，这些方法依赖于通用编码器（通常是 LSTM）首先获得输入文档的上下文表示。然后，他们通过构建一个精心设计的图来引入实体结构，其中实体表示通过传播进行相应更新。然而，由于编码网络和图形网络之间的异质性，这种方法将文本推理阶段和结构推理阶段隔离开来，这意味着上下文表示首先不能从结构指导中受益。

3 本文方法

3.1 本文方法概述

本文认为结构依赖性应该被纳入编码网络和整个系统。为此，我们首先在一个统一的框架下制定上述实体结构，其中我们定义了涵盖其间交互的各种提及依赖关系。然后，我们提出 SSAN（结构化自注意力网络），该网络配备了一种新颖的自我关注机制（Vaswani 等人，2017），以在其构建块内并通过自下而上的所有网络层对这些依赖性进行有效建模。注意，尽管本文只关注文档级关系提取的实

体结构，但本文开发的方法很容易适用于所有基于 Transformer 的预训练语言模型，以包含任何结构依赖性。

3.2 实体结构

实体结构描述实体在文档中的分布以及它们之间的依赖关系，考虑以下两种结构：

（1）共现（Co-occurence）结构：两个提及是否在同一句子中

（2）共指（Coreference）结构：两个提及是否指向同一实体

这两种结构都可以“TRUE”和“FALSE”来描述：

（1）对于共现结构，将文档分割成句子，并将它们作为显示提及交互的最小单元。“intra”和“inter”分别表示两个提及是内部的和句子间的。

（2）在共指结构中，“True”表示两种提及指的是同一个实体，因此需要一起进行研究和推理；“False”表示在某些谓词下可能相互关联的一对不同的实体。将它们分别表示为“coref”和“relate”。

		Coreference	
		True	False
Co-occurence	True	*intra+coref*	*intra+relate*
	False	*inter+coref*	*inter+relate*

图 2: 实体结构

除了实体提及之间的依赖性，我们还进一步考虑了实体提及与其句内非实体（NE）词之间的另一种依赖性。我们将其称为 intraNE。对于其他句子间的非实体词，我们假设不存在关键依赖性，并将其视为 NA。因此，整体结构被公式化为以实体为中心的邻接矩阵，其所有元素来自有限依赖集。整个结构形成了以实体为中心的邻接矩阵，元素包含：

$$sij \in \{intra + coref, inter + coref, intra + relate, inter + relate, intraNE, NA\}$$

3.3 SSAN

SSAN 继承了 Transformer（Vaswani 等人，2017）编码器的架构，它是一个由相同的构建块组成的堆栈，由前馈网络、残余连接和层标准化组成。作为其核心组件，我们提出了结构化的自我关注机制，包括两个可选的 Transformation 模块。给定输入令牌序列 x=（x1，x2，…，xn），遵循上述公式，我们引入 $S = s_{ij}$ 来表示其结构，其中 $i, j \in \{1, 2, \ldots, n\}$ 和 $s_{ij} \in \{intra + coref, inter + coref, intra + relate, inter + relate, intraNE, NA\}$ 是一个离散变量，表示从 xi 到 xj 的依赖关系。注意，为了实际实现，这里我们将依赖关系从概念级扩展到标记级。

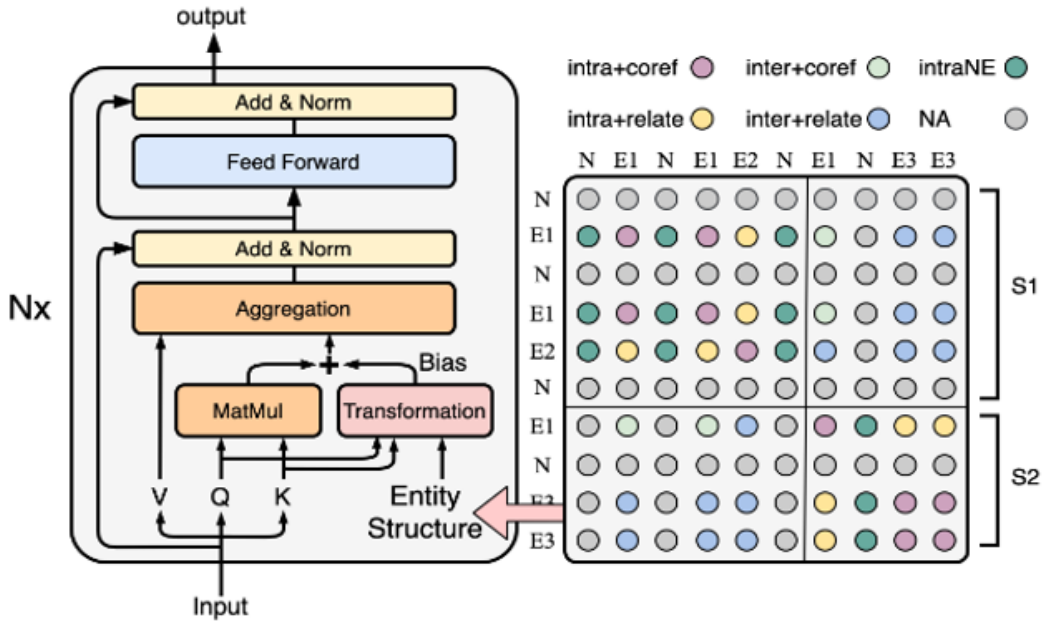

图 3: SSAN 的总体架构。左图将结构化的自我注意力作为其基本组成部分。右图解释了实体结构公式。这个例子包括两个句子：S1、S2 和三个实体：E1、E2 和 E3。N 表示非实体。第 i 行和第 j 列中的元素表示从 x_i 到密钥标记 x_j 的依赖关系，我们使用不同的颜色区分依赖关系。

在每一层 l 中，将输入 x_i^l 分别映射到 **Q,K,V** 向量上：

$$\mathbf{q}_i^l = x_i^l \mathbf{W}_l^Q, \mathbf{k}_i^l = x_i^l \mathbf{W}_l^K, \mathbf{v}_i^l = x_i^l \mathbf{W}_l^V$$

基于这些输入和实体结构 S，我们计算非结构化注意力值和结构化注意力偏差，然后将它们聚合在一起，以指导最终的自我注意力流。非结构化注意力值由标准自我注意力中的 **q,k** 计算得到：

$$e_{ij}^l = \frac{\mathbf{q}_i^l {\mathbf{k}_j^l}^T}{\sqrt{d}}$$

同时，使用一个额外的模块来对结构性依赖进行建模。将实体结构参数化，使得该模型受益于结构性依赖的指导：

$$\tilde{e}_{ij}^l = e_{ij}^l + \frac{transformation(\mathbf{q}_i^l, \mathbf{k}_i^l, s_{ij})}{\sqrt{d}}$$

在获得调节后的注意力得分后，应用 softmax，与 **v** 聚合起来：

$$z_i^{l+1} = \sum_{j=1}^{n} \frac{exp\tilde{e}_{ij}^l}{\sum_{k=1}^{n} exp\tilde{e}_{ik}^l} \mathbf{v}_j^l$$

3.4 Transformation 模块

在这里，有两个可供选择的 Transformation 模块：Biaffine 和 Decomposed Linear

$$bias_{ij}^l = Biaffine(s_{ij}, \mathbf{q}_i^l, \mathbf{k}_j^l) \quad \textbf{或} \quad Decomp(s_{ij}, \mathbf{q}_i^l, \mathbf{k}_j^l)$$

Biaffine Transformation: 同时并有方向地处理 **q** 向量和 **k** 向量, 直接为每个依赖项建立先验偏差模型，而不依赖于其上下文。

$$bias_{ij}^l = \mathbf{q}_i^l \mathbf{A}_{l,s_{ij}} {\mathbf{k}_j^l}^T + b_{l,s_{ij}}$$

Decomposed Linear Transformation: 在 **q** 向量和 **k** 向量上分别引入偏差。

$$bias_{ij}^{l} = \mathbf{q}_i^l \mathbf{K}_{l,s_{ij}}^T + \mathbf{Q}_{l,s_{ij}} \mathbf{k}_j^{l\,T} + b_{l,s_{ij}}$$

结构化自注意力的整体计算公式为：

$$\tilde{e}_{ij}^{l} = e_{ij}^{l} + \frac{transformation(\mathbf{q}_i^l, \mathbf{k}_i^l, s_{ij})}{\sqrt{d}} = \frac{\mathbf{q}_i^l \mathbf{k}_j^{l\,T} + \mathbf{q}_i^l \mathbf{A}_{l,s_{ij}} \mathbf{k}_j^{l\,T} + b_{l,s_{ij}}}{\sqrt{d}} \text{或} \frac{\mathbf{q}_i^l \mathbf{k}_j^{l\,T} + \mathbf{q}_i^l \mathbf{K}_{l,s_{ij}}^T + \mathbf{Q}_{l,s_{ij}} \mathbf{k}_j^{l\,T} + b_{l,s_{ij}}}{\sqrt{d}}$$

3.5 SSAN 应用于关系抽取

所提出的 SSAN 模型将文档文本作为输入，并在整个编码阶段内在实体结构的指导下构建其上下文表示。在这项工作中，简单地将其用于最小设计的牵引关系。在编码阶段之后，通过平均池为每个目标实体 e_i 构建固定维表示。然后，对于每个实体对，根据预先指定的关系模式计算相关性 r 的概率为：

$$P_r(e_s, e_o) = sigmoid(e_s \mathbf{W}_r e_o)$$

该模型使用交叉熵损失进行训练:

$$L = \sum_{<s,o>} \sum_{r} CrossEntropy(P_r(e_s, e_o), \overline{y}_r(e_s, e_o))$$

4 复现细节

4.1 与已有开源代码对比

参考相关源代码:https://github.com/BenfengXu/SSAN

原论文中最主要的贡献就在于提出结构化的自我关注机制，并通过两个可供选择的 Transformation 模块实现：Biaffine 和 Decomposed Linear。在复现过程中，将这两个 Transformation 模块结合在了一起，使其各自占有一定的权重。下面的代码实现了将两种实体结构先验知识结合起来的创新点。

```
self.entity_structure = entity_structure
if entity_structure != 'none':
    num_structural_dependencies = 5  # 5 distinct dependencies of entity structure, please refer to our paper.
    self.bias_layer_k = nn.ParameterList(
        [nn.Parameter(nn.init.xavier_uniform_(torch.empty(self.num_attention_heads, self.attention_head_size)))
         for _ in range(num_structural_dependencies)])
    self.bias_layer_q = nn.ParameterList(
        [nn.Parameter(nn.init.xavier_uniform_(torch.empty(self.num_attention_heads, self.attention_head_size)))
         for _ in range(num_structural_dependencies)])
    self.bili = nn.ParameterList(
        [nn.Parameter(nn.init.xavier_uniform_(torch.empty(self.num_attention_heads, self.attention_head_size, self.attention_head_size)))
         for _ in range(num_structural_dependencies)])
    self.abs_bias = nn.ParameterList(
        [nn.Parameter(torch.zeros(self.num_attention_heads)) for _ in range(num_structural_dependencies)])
```

图 4: 不只是简单地复现了论文的内容，并且结合了论文中提出的两种实体结构先验知识：Decomp（分解）和 Biaffine（双线性）。首先，初始化了 num_structural_dependencies 个参数，其中 num_structural_dependencies 的数量为 5，这表示在实体结构中存在 5 种不同的依赖关系。然后，使用 nn.ParameterList 函数初始化了 bias_layer_k、bias_layer_q、bili、abs_bias 四组参数。bias_layer_k 和 bias_layer_q 是 Decomp 的参数，它们的形状分别为（num_attention_heads，attention_head_size），并且使用 xavier_uniform 随机初始化。bili 是 Biaffine 的参数，其形状为（num_attention_heads，attention_head_size，attention_head_size），并且也使用 xavier_uniform 随机初始化。abs_bias 是绝对偏差参数，其形状为（num_attention_heads），初始值全部设置为 0。

```
if self.entity_structure == 'dec_bia':
  attention_bias_q = torch.einsum("bnid,nd->bni", query_layer, self.bias_layer_k[i]).unsqueeze(-1).repeat(1, 1, 1, query_layer.size(2))
  attention_bias_k = torch.einsum("nd,bnjd->bnj", self.bias_layer_q[i], key_layer).unsqueeze(-2).repeat(1, 1, key_layer.size(2), 1)
  attention_scores += alpha * (attention_bias_q + attention_bias_k + self.abs_bias[i][None, :, None, None]) * structure_mask[i]

  attention_bias = torch.einsum("bnip,npq,bnjq->bnij", query_layer, self.bili[i], key_layer)
  attention_scores += (1 - alpha) * (attention_bias + self.abs_bias[i][None, :, None, None]) * structure_mask[i]
```

图 5: 两种实体结构先验知识是不能直接结合的，如果需要同时使用两种先验知识，需要构造一个新的先验知识方法来结合两种先验知识。在这里 alpha 是一个比例系数，可以根据需要调整，若 alpha 为 0，则为 Biaffine；若 alpha 为 1，则为 Decomposed Linear；若 alpha 介于 0 和 1 之间，则是综合考虑了 Biaffine 和 Decomposed Linear 两种实体结构先验知识。具体而言，首先，将 decomp 结构与 biaffine 结构的先验知识结合在一起，然后使用一个系数 alpha 决定它们在结合结果中的相对重要性。其中，attention_bia_q 和 attention_bias_k 分别是 decomp 结构对于当前头的查询层和键层的影响。attention_bias 则是 biaffine 结构对于当前头的影响。最后，在计算注意力分数时，添加每个实体结构先验知识的贡献，并乘以结构掩码，以确定它是否对每个头产生影响。

4.2 创新点

在复现过程中，将原文提出的两个 Transformation 模块即两种实体结构先验知识结合在了一起，各自占有一定的权重。结合两种实体结构先验知识（'decomp' 和'biaffine'）作为创新点的意义在于可以得到更精确的实体关系描述。相比仅使用其中一种先验知识，结合两种先验知识的方法可以充分利用每种先验知识的优势，并通过协同工作来实现更精确的实体关系描述。

5 实验结果分析

Model	Test Ign F1 / F1
ContexAware (2019)	48.40 / 50.70
EoG*(2019)	49.48 / 51.82
BERT Two-Phase (2019a)	- / 53.92
GloVe+LSR (2020)	52.15 / 54.18
HINBERT (2020)	53.70 / 55.60
BERT+LSR (2020)	56.97 / 59.05
CorefRoBERTa (2020)	57.68 / 59.91
RoBERTa Base Baseline	57.27 / 59.48
$SSAN_{Decomp}$	57.72 / 59.75
$SSAN_{Biaffine}$	57.71 / 59.94
$SSAN_{dec\_bia}$	57.72 / 59.89

图 6: 在 DocRED 数据集上的结果

SSAN 以明显的优势超过其基线。将模型与以前的工作进行了比较，这些工作没有考虑实体结构，也没有在编码器内部和整个编码器中对它们进行显式建模。具体而言，ContexAware（Yao et al.2019）、BERT Two Phase（Wang et al.2019a）和 HINBERT（Tang et al.2020）没有考虑实体之间的结构依赖性。EOG（Christopoulou、Miwa 和 Ananiadou 2019）和 LSR（Nan 等人 2020）使用图形方法来执行结构推理，但仅在 BiLSTM 或 BERT 编码器之后。CorefBERT 和 CorefRoBERTa（Ye 等人，2020）进一步使用共指预处理任务预处理 BERT 和 RoBERTa，以实现共指结构的隐式推理。总的来说，这些结果证明了实体结构的有用性和 SSAN 的有效性。

6 总结与展望

这项工作中形式化了用于文档级关系抽取的实体结构。在此基础上，提出 SSAN 来有效地结合这些结构先验，它们同时和交互地形成实体的上下文推理和结构推理。在数据集上得到的性能证明了实体结构的有用性和 SSAN 模型的有效性。

对于未来的工作，有两个有希望的方向：1）将 SSAN 应用于更多任务，如阅读理解，其中实体结构或语法是有用的先验信息。2）扩展实体结构公式，以包括更多有意义的依赖，例如基于话语结构的更复杂的交互。