23. 问题回答

对知识的追求是深切的人类需求，因此毫不奇怪，自从有了计算机，我们就开始向它们提出问题。

到 1960 年代初，系统使用了两种主要的回答问题范式--基于信息检索和基于知识--来回答有关棒球统计或科学事实的问题。甚至虚构的计算机也参与其中。道格拉斯·亚当斯(Douglas Adams)在《银河系漫游指南》(The Hitchhiker's Guide to the Galaxy)中发明的计算机"深思"(Deep Thought)成功回答了"生命、宇宙和一切的终极问题"[1]。2011 年，IBM 的沃森问题回答系统赢得了电视游戏节目 Jeopardy!，在回答问题方面超越人类，比如：

WILLIAM WILKINSON'S "AN ACCOUNT OF THE
PRINCIPALITIES OF WALLACHIA AND MOLDOVIA"
INSPIRED THIS AUTHOR'S MOST FAMOUS NOVEL[2]

问答系统主要用于满足人类信息需求。人们在许多情况下会提出问题：与虚拟助手交谈时，与搜索引擎进行交互时，在查询数据库时。大多数问答系统都将重点放在这些信息需求的特定子集上：事实类问题，可以用短文本表示的简单事实来回答的问题，例如：

(23.1) Where is the Louvre Museum located?

(23.2) What is the average age of the onset of autism?

在这一章中，我们描述了事实类问题的两个主要范式。

第一种范式，基于信息检索(IR)的 QA，有时称为开放域问题 QA，依赖于 Web 上或 PubMed 等科学论文集中的大量文本。给定用户问题，信息检索可用于查找相关段落。然后，神经阅读理解算法读取这些检索到的段落，并直接从文本跨度中得出答案。

第二种范式，基于知识的 QA，系统改为构建查询的语义表示，例如,把 What states border Texas?(德克萨斯州与哪些州接壤？) 映射到逻辑表示形式：λx.state(x)∧borders(x，texas)，或者把 When was Ada Lovelace born?映射到间隙关系(gapped relation)：birth-year(Ada Lovelace，？X)。然后，这些含义表示用于查询事实数据库。

我们还将简要讨论用于回答问题的其他两个范例。一个事实是，我们在整个 NLP 中使用的庞大的预训练语言模型已经编码了许多事实。我们将介绍如何直接查询语言模型以回答问题。我们还将提及经典的预神经混合问答算法，该算法结合了来自基于 IR 和基于知识的来源的信息。

我们将探讨所有这些方法的可能性和局限性，同时还将介绍两种对问题解答至关重要但在整个 NLP 中也很重要的技术：信息检索(基于 IR 的 QA 的关键组件)和实体连接(类似于基于知识的 QA 的关键组件)。我们将在下一部分开始，介绍信息检索的任务。

最后注意：在本章中，我们仅关注事实类问题的解答，但感兴趣的读者可能还想跟进许多其他重要的 QA 任务。这些任务包括长格式的问题回答(回答为什么问题或其他需要生成长答案的问题)，社区问题回答（在其中我们利用了社区创建的问题-答案配对的数据集诸如 Quora 或 Stack Overflow。最后，一般而言，问答是 NLP 进步的重要基准，因此研究人员已经构建了可以在纽约 Regents 科学考试等考试中成功回答问题的系统，以此作为基准 NLP 和 AI 的一种方式(Clark 等人，2019)。

23.1. 信息检索(IR)

信息检索(Information retrieval，IR)是该领域的名称，涵盖了根据用户信息需求来检索所有形式的媒体。最终的 IR 系统通常称为搜索引擎。我们在本节中的目标是对 IR 进行足够的概述，以了解 IR 在问答中的应用。对信息检索更感兴趣的读者应参阅本章末尾的"历史说明"以及 Manning 等人(2008)的教科书。

我们考虑的 IR 任务称为**即席检索(ad hoc retrieval)**，其中用户向检索系统提出查询，然后检索系统

[1] 答案是 42，但不幸的是，问题的细节一直没有透露。

[2] 答案当然是"谁是布拉姆·斯托克"，小说是《德古拉》。

从某些集合中返回一组有序的文档。文档(document)指的是系统索引和检索的任何文本单位(网页，科学论文，新闻报道，甚至是短文诸如段落)。集合(collection)是指用于满足用户请求的一组文档。术语(term)是指集合中的单词，但也可以包括短语。最后，查询(query)代表用一组术语表示的用户信息需求。图 23.1 显示了即席检索引擎的高级体系结构。

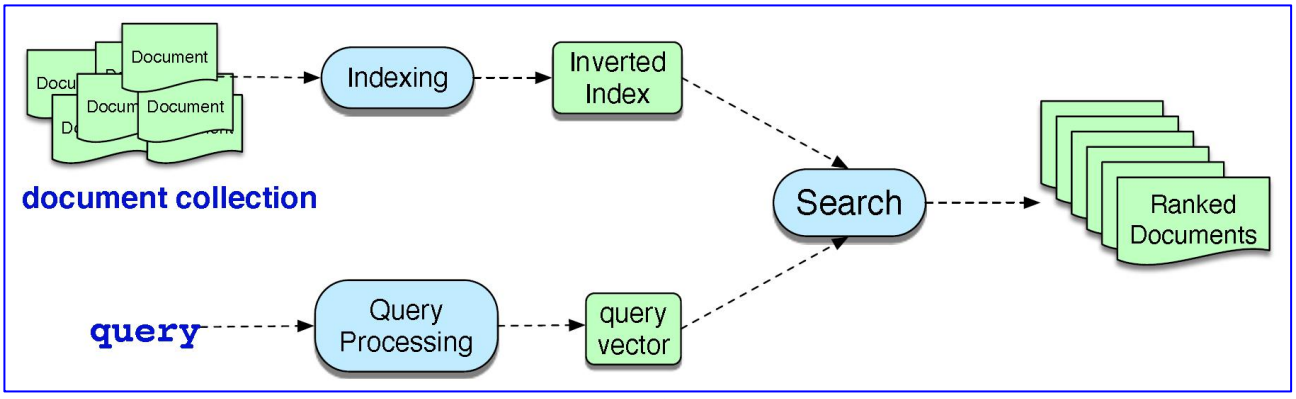


图 23-1：即席 IR 的系统架构

基本的 IR 体系结构使用我们在第 6 章中介绍的向量空间模型，在该模型中，我们基于 unigram 单词计数将查询和文档映射到向量，并使用向量之间的余弦相似度对潜在的文档进行排序(Salton, 1971)。因此，这是第 4 章中介绍的词袋模型的一个例子，因为单词是独立于它们的位置进行考虑的。

23.1.1. 术语加权和文档评分

让我们看看如何对文档和查询之间的匹配进行评分的细节。在 IR 中我们不使用原始的单词计数，而是为每个文档单词计算**术语权重(term weight)**。两个术语加权方案是常见的:在第 6 章介绍的 tf-idf 加权，和一个稍微更强大的变种称为 **BM25**。我们将在这里重新介绍 tf-idf，这样读者就不需要回头看第 6 章了。tf-idf(这里的“-”是一个连字符，而不是减号)是个术语频率 tf 和间接文档频率 idf 的乘积。

频率这个词告诉我们这个词有多频繁;在文档中经常出现的单词可能提供有关文档内容的信息。我们通常使用词频的 $\log_{10}$，而不是原始计数。直觉是，一个单词在文档中出现 100 次并不会使这个单词与文档含义的相关性增加 100 倍。因为我们不能取 0 的对数，所以我们通常在计数中加上 1:

$$\text{tf}_{t,d} = \log_{10}(\text{count}(t,d)+1) \tag{23.3}$$

如果我们使用对数加权，在一个文档中出现 0 次的项将有 tf=log10(0+1)=0，在一个文档中出现 10 次的项将有 $\text{tf}=\log_{10}(10+1)=1.4$, 100 次的 $\text{tf}=\log_{10}(100+1)=2.004$, 1000 次的 tf=3.00044，以此类推。术语 t 的文档频率 dft 是它出现在文档中的数量。仅出现在少数文档中的术语有助于将这些文档从集合的其余部分中区分出来;出现在整个集合中的术语就没有那么有用了。逆文档频率或 idf 词权重(Sparck Jones, 1972)定义为:

$$\text{idf}_t = \log_{10}\frac{N}{\text{df}_t} \tag{23.4}$$

其中，N 为集合中的文档总数，dft 为出现术语 t 的文档数。一个术语出现的文献越少，其权重越高;最小的 0 权重被分配给出现在每个文档中的术语。其中 N 是集合中文档的总数，而 dft 是出现术语 t 的文档数。出现术语的文档越少，此权重就越高；最低权重 0 分配给每个文档中出现的术语。

以下是莎士比亚戏剧语料库中某些单词的一些 idf 值，从仅在像《罗密欧》这样的一部戏剧中出现的非常有启发性的单词，到像《色拉》或《福斯塔夫》中很少出现的那些单词，再到像《傻瓜》，或如此普遍以至于完全没有区别，因为它们出现在所有 37 部《好》或《甜》的戏剧中。

Word	Romeo	salad	Falstaff	forest	battle	wit	fool	good	sweet
df	1	2	4	12	21	34	36	37	37
idf	1.57	1.27	0.967	0.489	0.246	0.037	0.012	0	0

则文档 d 中单词 t 的 tf-idf 值为词频 $\text{tf}_{t,d}$ 和 idf_t 的乘积:

$$\text{tf-idf}(t,d) = \text{tf}_{t,d} \bullet \text{idf}_t \tag{23.5}$$

23.1.2. 文档评分

我们用查询向量 q 对文档 d 的向量 d 的余弦值进行评分：

$$\text{score}(q,d) = \cos(\mathbf{q},\mathbf{d}) = \frac{\mathbf{q}\cdot\mathbf{d}}{|\mathbf{q}||\mathbf{d}|} \tag{23.6}$$

另一种考虑余弦计算的方法是单位向量的点积;我们首先将查询向量和文档向量标准化为单位向量，除以它们的长度，然后取点积:

$$\text{score}(q,d) = \cos(\mathbf{q},\mathbf{d}) = \frac{\mathbf{q}}{|\mathbf{q}|}\cdot\frac{\mathbf{d}}{|\mathbf{d}|} \tag{23.7}$$

我们可以用 tf-idf 值拼出(spell out)公式(23.7)，并把点积拼出积的和:

$$\text{score}(q,d) = \sum_{t\in q}\frac{\text{tf}-\text{idf}(t,q)}{\sqrt{\sum_{q_i\in q}\text{tf}-\text{idf}^2(q_i,q)}}\cdot\frac{\text{tf}-\text{idf}(t,d)}{\sqrt{\sum_{d_i\in d}\text{tf}-\text{idf}^2(d_i,d)}} \tag{23.8}$$

在实践中，通常通过简化查询处理来近似公式(23.8)。查询通常很短，所以每个查询词的计数很可能是 1。并且查询的余弦归一化(除以|q|)对所有的文档都是一样的，所以不会改变任意两个文档 D_i 和 D_j 之间的排名，所以我们一般使用以下简单的分数对一个文档 d 给出一个查询 q:

$$\text{score}(q,d) = \sum_{t\in q}\frac{\text{tf}-\text{idf}(t,d)}{|d|} \tag{23.9}$$

让我们浏览一个针对 4 个 nano 文档集合的小查询示例，计算 tf-idf 值并查看文档的排序。我们假设以下查询和文档中的所有单词都被降格为小写，并且删除了标点符号:

Query: sweet love
Doc 1: Sweet sweet nurse! Love?
Doc 2: Sweet sorrow
Doc 3: How sweet is love?
Doc 4: Nurse!

图 23.2 显示了前两个文档使用公式(23.3)、公式(23.4)和公式(23.5)计算 tf-idf 值和文档向量长度|d|(文档 3 和文档 4 的计算留给读者作为练习)。

	Document 1					Document 2				
word	count	tf	df	idf	tf-idf	count	tf	df	idf	tf-idf
love	1	0.301	2	0.301	0.091	0	0	2	0.301	0
sweet	2	0.477	3	0.125	0.060	1	0.301	3	0.125	0.038
sorrow	0	0	1	0.602	0	1	0.301	1	0.602	0.181
how	0	0	1	0.602	0	0	0	1	0.602	0
nurse	1	0.301	2	0.301	0.091	0	0	2	0.301	0
is	0	0	1	0.602	0	0	0	1	0.602	0
$\lvert d_1\rvert = \sqrt{.091^2 + .060^2 + .901^2} = .141$						$\lvert d_2\rvert = \sqrt{.038^2 + .181^2} = .185$				

图 23-2：计算 nano 文档 1 和文档 2 的 tf-idf 值

Doc	|d|	tf-idf(sweet)	tf-idf(love)	score
1	.141	.060	.091	1.07
3	.274	.038	.091	0.471
2	.185	.038	0	0.205
4	.090	0	0	0

图 23-3：根据公式(23.9)对文档进行排序

图 23.3 显示了 4 篇文献的得分，根据公式(23.9)重新排序。这一排名从向量空间模型上得到了直观的体现。文档 1 是排名最高的，它有两个术语(sweet 和 love，其中 sweet 有两个实例)；在文档 3 之上，文

档 3 的分母长度|d|较大，sweet 的 tf 也较小。文档 3 缺少了其中一个术语(love)，文档 4 两个术语都缺少。

tf-idf 家族中一个稍微复杂一点的变体是 BM25 加权方案(有时在引入该方案的 Okapi IR 系统后被称为 Okapi BM25 (Robertson 等人 1995))。BM25 增加了两个参数:k(调节术语频率和 IDF 之间平衡的旋钮)和 b(控制文档长度规范化的重要性)。给定查询 q，文档 d 的 BM25 得分为:

$$\sum_{t \in q} \overbrace{\log\left(\frac{N}{df_t}\right)}^{\text{IDF}} \overbrace{\frac{tf_{t,d}}{k\left(1-b+b\left(\frac{|d|}{|d_{\text{avg}}|}\right)\right)+tf_{t,d}}}^{\text{weighted tf}} \tag{23.10}$$

其中$|d_{avg}|$是平均文档的长度。当 k 为 0 时，BM25 恢复到不使用术语频率，只使用查询中的术语的二进制选择(加上 idf)。一个大的 k 会导致原始的术语更加频繁(加上 idf)。b 的范围是从 1(按文档长度缩放)到 0(无长度缩放)。Manning 等人(2008)认为合理的值是 k = [1.2,2]， b = 0.75。Kamphuis 等人(2020)对 BM25 的许多次要改型进行了有益的总结。

停用词 过去，通常在表示查询之前从查询和文档中都删除高频单词。将要去除的这种高频单词的列表称为停用列表(stop list)。直觉是，高频术语(通常是诸如 the、a、to 等功能词)的语义权重很小，可能对检索没有帮助，并且还可以帮助缩小我们在下面描述的倒排索引文件。使用停用列表的缺点是，如果一个搜索短语包含停用列表中的词，则这个短语很难被搜索到。例如，通常停用列表会将短语 to be or not to be 减少为 not。在现代的 IR 系统中，停用列表的使用要少得多，部分原因是提高了效率，部分原因是 IDF 加权已经处理了其大部分功能，从而减轻了每个文档中出现的功能词的权重。尽管如此，删除停用词在各种 NLP 任务中有时还是有用的，因此值得留意。

23.1.3. 倒排索引

为了计算分数，我们需要有效地查找在查询中包含单词的文档。(正如我们在图 23.3 中看到的，任何不包含查询项的文档的得分都为 0，可以忽略。)因此，IR 中的基本搜索问题是找到所有的 d∈C 的文档，这些文档包含一个术语 q∈Q。

这个任务的数据结构是**倒排索引(inverted index)**，我们使用它来提高搜索效率，还可以方便地存储有用的信息，如文档频率和每个文档中每个术语的计数。

给定查询词的倒排索引会给出包含该词的文档的列表。它由两部分组成：字典和帖子。词典是一个术语列表(旨在有效访问)，每个术语都指向该术语的帖子列表。帖子列表是与每个术语相关联的文档 ID 的列表，该列表还可以包含诸如术语频率或术语在文档中的确切位置之类的信息。字典也可以为每个术语开始文档频率。例如，上面四个示例文档的简单倒排索引[3]：每个单词的文档频率都在{ }中，一个指向帖子列表的指针→，该列表包含文档 ID 号，术语计数都在[ ]中。可能类似于以下内容：

how {1} → 3 [1]
is {1} → 3 [1]
love {2} → 1 [1] → 3 [1]
nurse {2} → 1 [1] → 4 [1]
sorry {1} → 2 [1]
sweet {3} → 1 [2] → 2 [1] → 3 [1]

给定查询中的术语列表，我们可以非常有效地获取所有候选文档的列表，以及计算 tf-idf 分数所需的必要信息。倒排索引还有其他选择。对于寻找维基百科页面以匹配用户查询的问题域，Chen 等人(2017)表明，基于 bigrams 的索引比 unigrams 更好，并且使用有效的哈希算法而不是倒排索引来提高搜索效率。

23.1.4. 信息检索系统的评估

我们使用相同的精度和召回率指标来衡量排名检索系统的性能。我们假设 IR 系统返回的每个文档都

[3] 译者注：至少包含四项内容：{}，→，ID，[ ].

与我们的目的相关或不相关。精度是所返回的相关文档中所占比例的一部分，召回率是所返回的所有相关文档中所占比例的一部分。更正式地，让我们假设系统响应信息请求返回了 T 个排序的文档，其中的一个子集 R 是相关的，不相交的子集 N 是其余不相关的文档，并且集合中的 U 个文档总体上与 这个请求相关。然后将精度和召回率定义为：

$$\text{Precision} = \frac{|R|}{|T|} \qquad \text{Recall} = \frac{|R|}{|U|} \tag{23.11}$$

不幸的是，这些度量标准不能充分地测量对返回的文档进行排名的系统的性能。如果我们比较两个排名的检索系统的性能，我们需要一个更倾向于相关文档排名更高的指标。我们需要调整精度和召回率，以捕获系统在将相关文档置于更高排名方面做得有多好。

让我们来看一个例子。假设图 23.4 中的表格给出了特定于排名的精度和召回率的值，这些值在我们向下浏览特定查询的一组排序文档时被计算;精度是在给定级别上看到的相关文档的比例，而召回率是在相同级别上找到的相关文档的比例。本例中的召回率度量是基于这样一个查询，该查询有 9 个相关文档，这些文档作为一个整体的存在于集合中。

Rank	Judgment	$\text{Precision}_{\text{Rank}}$	$\text{Recall}_{\text{Rank}}$
1	R	1.0	.11
2	N	.50	.11
3	R	.66	.22
4	N	.50	.22
5	R	.60	.33
6	R	.66	.44
7	N	.57	.44
8	R	.63	.55
9	N	.55	.55
10	N	.50	.55
11	R	.55	.66
12	N	.50	.66
13	N	.46	.66
14	N	.43	.66
15	R	.47	.77
16	N	.44	.77
17	N	.44	.77
18	R	.44	.88
19	N	.42	.88
20	N	.40	.88
21	N	.38	.88
22	N	.36	.88
23	N	.35	.88
24	N	.33	.88
25	R	.36	1.0

图 23-4：针对排名的精度和召回率的计算

图注：当我们向下浏览一组已排名的文档时(假设集合有 9 个相关文档)，将计算特定于排名的精度和召回率值。

注意，召回率不是递减的;当遇到相关文件时，召回率增加；当发现非相关文件时，召回率保持不变。另一方面，精度会上下波动，当找到相关的文档时精度会增加，反之则会减少。可视化精度和召回率最常用的方法是将精度与召回率的关系绘制成**精度-召回率曲线(precision-recall curve)**，表 23.4 中的数据如图 23.5 所示。

图 23.5 显示了单个查询的值。但是，我们需要合并所有查询的值，并以一种可以将一个系统与另一个系统进行比较的方式进行合并。一种方法是绘制 11 个固定召回率(0 到 100，以 10 为步长)的平均精度值。由于我们不太可能拥有这些确切级别的数据点，因此我们对来自我们已有数据点的 11 个召回值使用**插值精度(interpolated precision)**值。为此，我们可以选择在任何召回水平上达到或高于我们计算出的最大召回率时所能达到的最大精度值。换一种说法，

$$\text{IntPrecision}(r) = \max_{i>=r} \text{Precision}(i) \tag{23.12}$$

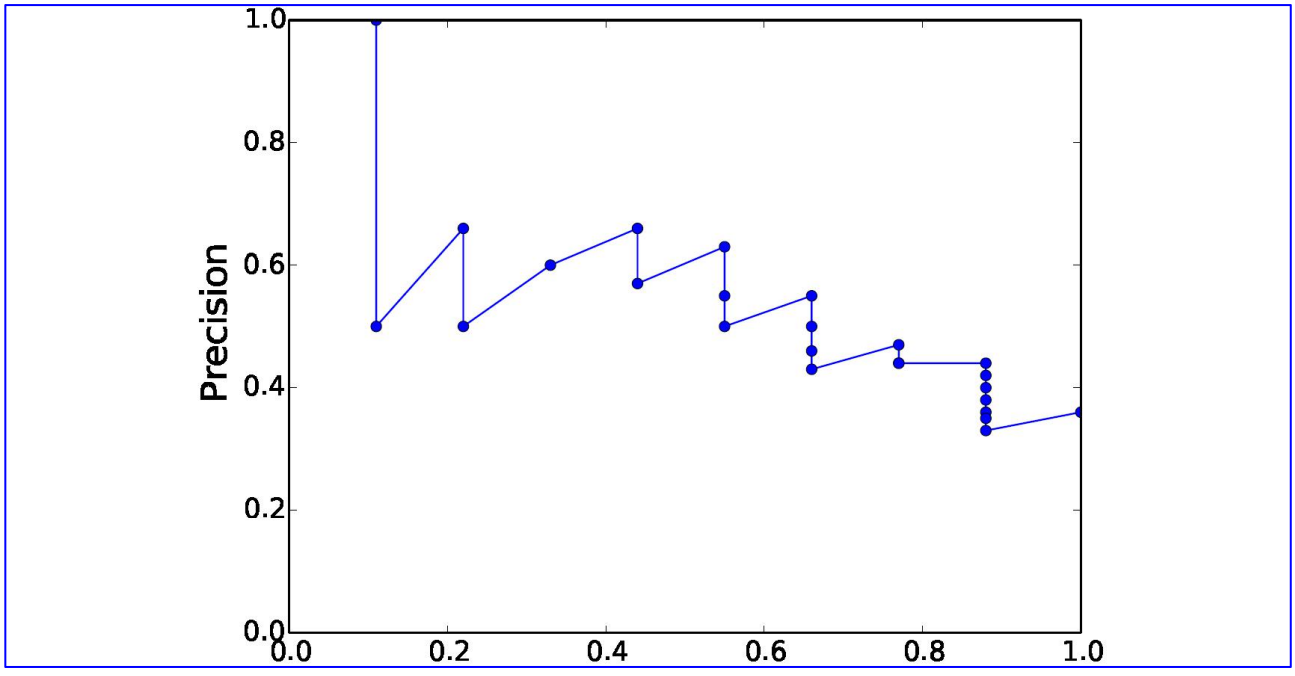


图 23-5：表 23.4 数据的精度和召回率的曲线

这种插值方案不仅可以让我们在一组查询上平均性能，而且还可以平滑原始数据中不规则的精度值。它旨在通过分配被测对象在较高的召回率水平下分配的最大精度值，来为系统带来疑惑的好处。图 23.6 和图 23.7 显示了从我们的示例中得到的内插数据点。

Interpolated Precision	Recall
1.0	0.0
1.0	.10
.66	.20
.66	.30
.66	.40
.63	.50
.55	.60
.47	.70
.44	.80
.36	.90
.36	1.0

图 23-6：从图 23.4 中插入数据点

给定曲线，如图 23.7 所示，我们可以通过比较两种系统或方法的曲线来比较它们。显然，在所有召回值中，更高精度的曲线是首选。然而，这些曲线也可以提供对系统整体行为的洞察。左边的精度较高的系统可能更倾向于精度而不是召回率，而更适合召回的系统将在较高的召回级别上更高(如右图所示)。

评估排名检索的第二种方法是**平均精度的均值(mean average precision，MAP)**，它提供了一个可用于比较竞争系统或方法的度量。通过这种方法，我们再次向下浏览术语的已排序列表，但现在我们仅在遇到相关术语的那些点上注意精度(例如，在图 23.4 中的排名 1、3、5、6 而不是 2 或 4 处)。对于单个查询，我们将这些单独的精度测量值平均化为返回值集合(直至某个固定的临界值)。更正式地说，如果我们假设 R_r 是等于或大于 r 的相关文档的集合，则单个查询的平均精度(AP)为：

$$\text{AP} = \frac{1}{|R_r|}\sum_{d\in R_r}\text{Precision}_i(d) \tag{23.13}$$

其中 $\text{Precision}_r(d)$是在文档 d 的位置上测量的精度。对于一系列的问题 Q，我们对这些平均值进行平均，得到我们最终的 MAP 测量结果:

$$\text{MAP} = \frac{1}{|Q|}\sum_{q\in Q}\text{AP}(q) \tag{23.14}$$

在图 23.4 中，单个查询(因此=AP)的映射是 0.6。

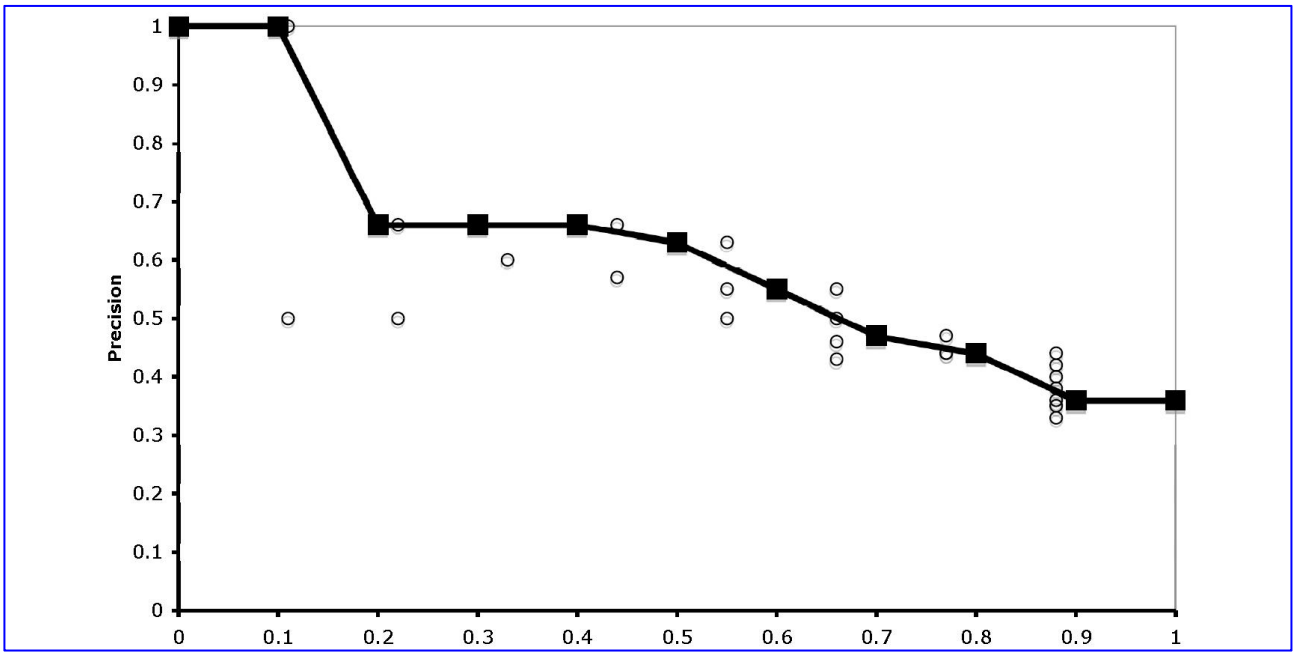


图 23-7：插值的精度-召回曲线

图注：11 个点的插值精度-召回曲线。对每一个查询，11 个标准召回级别的精度从任何更高级别召回的最大值内插。同时给出了原始测量精度召回点。

23.1.5. 稠密向量 IR

人们早就知道，用于 IR 的经典 tf-idf 或 BM25 算法有一个概念上的缺陷:它们只有在查询和文档之间有准确的单词重叠时才能工作。换句话说，提出查询(或询问问题)的用户需要准确猜测答案的作者可能使用了哪些词来讨论这个问题。正如 Lin 等人(2020)所说，用户可能会决定搜索一个悲剧的爱情故事(a tragic love story)，但莎士比亚却写了一个命运多舛的情侣(star-crossed lovers)。这被称为词汇不匹配问题(Furnas 等人，1987)。

这个问题的解决方案是使用一种可以处理同义词的方法:使用(密集的)嵌入，而不是(稀疏的)单词计数向量。这个想法很早就在 LSI 方法中被提出(Deerwester 等人，1990)，但是现代方法都使用像 BERT 这样的编码器。在所谓的双编码器(biencoder)中，我们使用了两个独立的编码器模型，一个编码查询，一个编码文档，并使用这两个向量之间的点积作为分数(图 23.8)。例如，如果我们使用 BERT，我们将有两个编码器 BERT Q 和 BERT D，可以将查询和文档表示为各自编码器的[CLS]符记(Karpukhin 等人，2020):

$$h_q = \text{BERT}_Q(q)[\text{CLS}]$$
$$h_d = \text{BERT}_D(d)[\text{CLS}]$$
$$score(d,q) = h_q \cdot h_d \quad (23.15)$$

更复杂的版本可以使用其他方式来表示编码的文本，例如对所有符记的 BERT 输出使用平均池化而不是使用 CLS 符记，或者可以在编码或点积步骤之后添加额外的权重矩阵(Liu 等人 2016，Lee 等人 2019)。

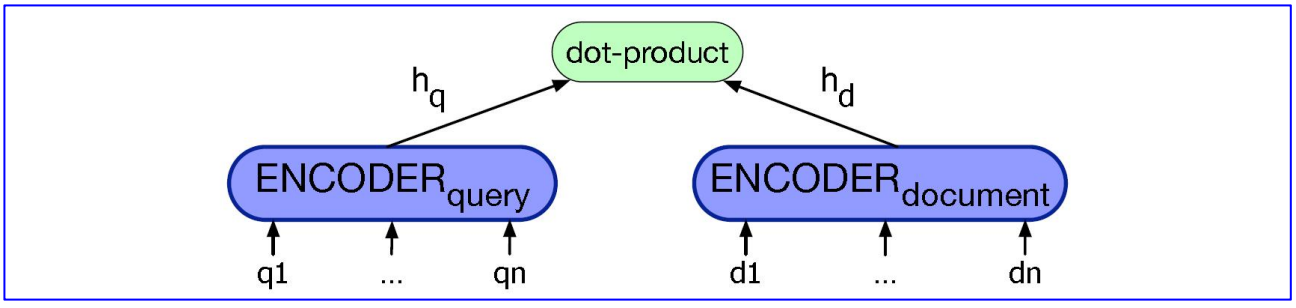


图 23-8：BERT 双编码器用于计算文档与查询的相关性

将密集的矢量用于 IR 或 QA 的检索组件，仍然是一个开放的研究领域。在活跃研究的众多领域中，如何对 IR 任务的编码器模块进行微调(通常通过对查询文档组合进行微调，并通过各种巧妙的方法来获得否定示例)，以及如何处理这一事实：文件通常比 BERT 等编码器可处理的更长(通常通过将文件分成段落)。

效率也是一个问题。每个 IR 引擎的核心都需要对每个可能的文档与查询的相似性进行排序。对于稀疏的单词计数向量，倒排索引可以非常有效地实现这一点。对于基于 BERT 或其他 Transformer 编码器的密集向量算法，寻找与密集查询向量有最高点积的密集文档向量集是最近邻搜索的一个例子。因此，现代系统使用近似最近邻向量搜索算法，如 **Faiss** (Johnson 等人，2017)。

23.2. 基于 IR 的事实性问答

基于 IR 的 QA(有时称为开放域 QA)的目标是通过从 web 或其他大型文档集合中找到短文本片段来回答用户的问题。图 23.9 显示了一些示例事实性问题及其答案。

Question	Answer
Where is the Louvre Museum located?	in Paris, France
What are the names of Odin's ravens?	Huginn and Muninn
What kind of nuts are used in marzipan?	almonds
What instrument did Max Roach play?	drums
What's the official language of Algeria?	Arabic

图 23-9：一些事实性问题及其答案

基于 IR 的 QA 的主要范式是图 23.10 所示的**检索和阅读(retrieve and read)**模型。在这个两阶段模型的第一阶段中，我们从一个文本集合中检索相关的段落，通常使用我们在前一节中看到的类型的搜索引擎。在第二阶段，一个神经阅读理解算法会遍历每一篇文章，找出可能回答问题的跨度。

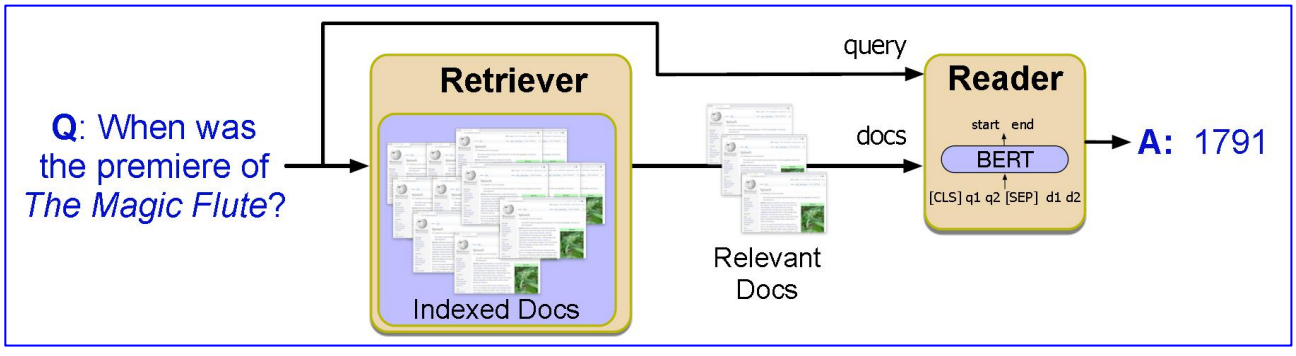


图 23-10：基于 IR 的检索和阅读模型

图注：基于 IR 的事实性 QA 有两个阶段:检索，从集合中返回相关的文档；阅读，其中一个神经阅读理解系统提取答案跨度。

一些问答系统仅专注于第二项任务，即**阅读理解(reading comprehension)**任务。阅读理解系统会得到一个事实性问题 q 和一个可能包含答案的段落 p，然后返回答案 s(或者声明段落中没有答案，或者在某些设置中从一组可能的答案中进行选择)。当然，此设置与满足他们需要回答的问题的用户的信息需求不符(毕竟，如果用户知道哪个段落包含了答案，他们就可以自己阅读)。取而代之的是，该任务最初是根据儿童的阅读理解测试(一种使儿童获得阅读能力并必须回答有关问题的教学手段)而建立的，以此作为评估自然语言理解能力的一种方式(Hirschman 等人，1999)。阅读理解系统仍以这种方式使用，但也已发展成为现代检索和阅读模型的第二阶段。

其他的问答系统解决整个检索和阅读任务;给他们一个事实性的问题和一个大型文档集合(如 Wikipedia 或 web 检索)，并返回一个答案，通常是从文档中提取的一段文本。这个任务通常被称为开放域 QA。在接下来的几节中，我们将介绍基于 IR 的 QA 的各个部分，首先介绍一些用于阅读理解和完整 QA 任务的常用数据集。

23.2.1. 基于 IR 的 QA：数据集

基于 IR 的 QA 的数据集通常是通过首先开发包含(段落，问题，答案)元组的阅读理解数据集来创建的。阅读理解系统可以使用数据集来训练被授予段落和问题的读者，并预测该段落的一个跨度作为答案。包括要从中提取答案的段落，就无需使用阅读理解系统来处理 IR。

例如，**斯坦福问答数据集(Stanford Question Answering Dataset，SQuAD)**由来自维基百科的文章和相关问题组成，这些问题的答案是来自文章的跨度(Rajpurkar 等人，2016)。SQuAD 2.0 还增加了一些设计成无法回答的问题(Rajpurkar 等人，2018)，总共有超过 15 万个问题。图 23.11 显示了 SQuAD 2.0 的一段节选，包含 3 个问题和黄金答案。

SQuAD 是通过让人类阅读给定的维基百科文章，写下关于这篇文章的问题，并选择一个特定的答案来建立的。

其他数据集是通过类似的技术创建的，但是尝试使问题变得更加复杂。**HotpotQA** 数据集(Yang 等人，2018)是通过向众包工作者显示多个上下文文档而创建的，并要求提出需要对所有文档进行推理的问题。

像 SQuAD 或 HotpotQA 这样的数据集中的问题是由首先阅读段落的评注者创建的，这可能使他们的问题更容易回答，因为评注者可能(下意识地)使用了答案文本中的单词。

自然问题(Natural Questions)数据集(Kwiatkowski 等人，2019)将真正的匿名查询合并到 Google 搜索引擎中。一个查询以及来自前 5 个搜索结果的 Wikipedia 页面被呈送到评注者面前，然后评注者注释一个段落篇幅的长答案和一个短跨度的答案。如果文本不包含该段落，则标记为 null。例如，问题“何时将啤酒花添加到酿造过程中？(When are hops added to the brewing process?)”有一个短答案“沸腾过程(the boiling process)”和一个长答案：Brewing 维基百科页面上的整个段落。在使用此数据集时，阅读理解模型会得到一个问题和一个 Wikipedia 页面，并且必须返回长答案、短答案或“无答案”回应。

解决这种可能偏见的一种常见解决方案是从某些问题中建立数据集，这些问题不是特意针对某一篇文章而被写下来的。TriviaQA 数据集(Joshi 等人 2017)包含由琐事爱好者编写的 94K 个问题，以及来自 Wikipedia 和网络的辅助文档，从而产生了 65 万个问题-答案-证据的三元组。

Beyonc´e Giselle Knowles-Carter (born September 4, 1981) is an American singer, songwriter, record producer and actress. Born and raised in Houston, Texas, she performed in various singing and dancing competitions as a child, and rose to fame in the late 1990s as lead singer of R&B girl-group Destiny's Child. Managed by her father, Mathew Knowles, the group became one of the world's best-selling girl groups of all time. Their hiatus saw the release of Beyonc´e's debut album, Dangerously in Love (2003), which established her as a solo artist worldwide, earned five Grammy Awards and featured the Billboard Hot 100 number-one singles "Crazy in Love" and "Baby Boy".
Q: "In what city and state did Beyonc´e grow up?" A: "Houston, Texas"
Q: "What areas did Beyonc´e compete in when she was growing up?" A: "singing and dancing"
Q: "When did Beyonc´e release Dangerously in Love?" A: "2003"

图 23-11：一篇包含问题和答案的文章

图注：一篇(维基百科)文章来自 SQuAD 2.0 数据集(Rajpurkar 等人 2018)，包含 3 个样本问题和标注答案的跨度。

以上数据集均为英文。TyDi QA 数据集包含来自 11 种类型多样语言的 204K 问答配对，包括阿拉伯语、孟加拉语、斯瓦希里语、俄语和泰语(Clark 等人 2020)。在 TyDi QA 任务中，一个问题和一些来自维基百科的文章被给到系统，系统必须(a)选择包含答案的文章(如果没有文章包含答案，则为 null)，(b)标记答案的最小跨度(或为空)。很多问题都没有答案。数据集中的各种语言给 QA 系统带来了挑战，比如问题和答案之间的形态变化，或者单词分割或多种字母的复杂问题。

在阅读理解任务中，系统会给出一个问题，并在其中找到答案。然而，在完整的两阶段 QA 任务中，系统并没有给出一篇文章，而是被要求自己从一些文档集合中进行检索。创建开放领域 QA 数据集的常见方法是修改阅读理解数据集。出于研究目的，这通常是通过使用注释 Wikipedia 的 QA 数据集(如 SQuAD

或 HotpotQA)来实现的。对于训练，整个(问题，短文，答案)三元组用来训练读者。但在推理时，这些段落被删除，系统只给出问题，并访问整个维基百科语料库。然后系统必须进行 IR，以找到一组页面，然后读取它们。

23.2.2. 基于 IR 的 QA：阅读器(答案跨度提取)

基于 IR 的 QA 的第一个阶段是检索器，例如我们在 23.1 节中看到的类型。基于 IR 的 QA 的第二阶段是阅读器。阅读器的工作是把一篇文章作为输入并给出答案。在我们这里讨论的**提取的 QA(extractive QA)**中，答案是文章中的文本的一个跨度[4]。例如，给出“How tall is Mt. Everest?”这样一个问题，以及一个包含 Reaching 29,029 feet at its summit 从句的段落，则阅读器输出 29,029 feet。

答案抽取任务通常采用跨度标注模型:在文章中标识构成答案的**跨度(span)**(连续的文本字符串)。阅读理解的神经算法，给出了一个包含 n 个符记 q_1，...，q_n 的问题 q 和一个包含 m 个符记 p_1，...，p_m 的段落 p。因此，它们的目标是计算每个可能的跨度 a 是答案的概率 P(a|q, p)。

如果每个跨度 a 从 a_s 的位置开始并在 a_e 的位置结束，则我们作出简化的假设，即可以将该概率估计为 $P(a|q, p)= P_{start}(a_s|q, p)P_{end}(a_e|q, p)$。因此，对于段落中的每个符记 p_i，我们将计算两个概率：$p_{start}(i)$ 表示 p_i 是答案跨度的开始，而 $p_{end}(i)$表示 p_i 是答案跨度的结束。用于阅读理解的标准基线算法是将问题和段落传递给像 BERT 这样的编码器(图 23.12)，当作用[SEP]符记分隔的字符串，导致每个段落符记都嵌入一个编码符记。

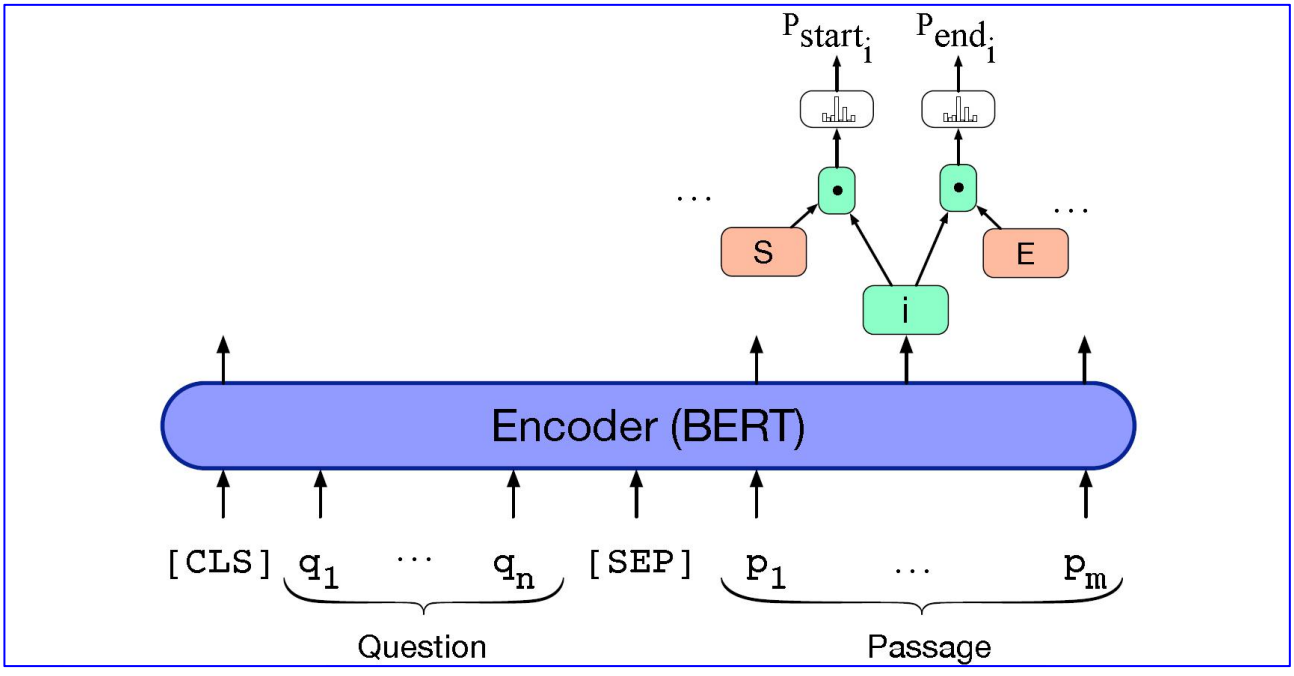


图 23-12：基于跨度的 QA 编码器模型(使用 BERT)

图注：一个编码器模型(使用 BERT)，用于基于跨度的 QA，来自基于阅读理解的 QA 任务。

对于基于跨度的问答，我们将问题表示为第一个序列，将文章表示为第二个序列。我们还需要添加一个线性层，该层将在微调阶段接受训练，以预测跨度的起始和结束位置。我们将添加两个新的特殊向量:一个跨度开始嵌入 S，一个跨度结束嵌入 E，它们将在微调中学到。为了得到每个输出符记 p_i'的跨度开始概率，我们计算 S 和 p_i'之间的点积，然后使用 softmax 对文中所有符记 p_i'进行归一化:

$$P_{\text{start}_i} = \frac{\exp(S \cdot p_i')}{\sum_j \exp(S \cdot p_j')} \tag{23.16}$$

我们用类似的方法来计算跨度结束的概率:

[4] 在这里，我们跳过更困难的抽象化的 QA 任务，在这个任务中，系统可以写出一个不是完全从文章中提取的答案。

$$P_{\text{end}_i} = \frac{\exp(E \cdot p'_i)}{\sum_j \exp(E \cdot p'_j)} \tag{23.17}$$

从位置 i 到位置 j 的候选跨度的得分为 $S \cdot p_i' + E \cdot p_j'$，选择 j≥i 的候选跨度的得分最高的就是模型预测。微调的训练损失是每个实例正确的开始和结束位置的对数概率的总和的负数:

$$L = -\log P_{\text{start}_i} - \log P_{\text{end}_i} \tag{23.18}$$

许多数据集(如 SQuAD 2.0 和 Natural Questions)也包含成对的(question, passage)，其中的 passage 不包括答案。因此，我们还需要一种方法来估计问题的答案不在文档中的可能性。标准做法是，将没有答案的问题视为以[CLS]符记作为答案，因此答案跨度的起始和结束索引将指向[CLS] (Devlin 等人，2019)。

对于许多数据集，我们还需要处理带注释的文档/段落长于 BERT 允许的最大 512 个输入符记的情况。考虑例如自然问题之类的案例，其中带有金色标签的段落是完整的 Wikipedia 页面。在这种情况下，我们可以从标签的 Wikipedia 页面创建多个伪段落观察。每个观察结果是通过将[CLS]、问题、[SEP]和文档中的符记拼接在一起形成的。我们遍历文档，滑动一个大小为 512(或 512 减去问题长度 n、再减去特殊符记)的窗口，并将符记的窗口包装到每个下一个伪通道中。观测值的答案跨度，要么标签为[CLS](在此特定窗口中为没有答案)，要么标记为黄金标签的跨度。Alberti 等人(2019)建议通过使用 128 个符记的跨度来允许窗口重叠。可以使用相同的过程进行推理，将每个检索到的文档分解为单独的观察段落并标签每个观察结果。可以将答案选择为概率最高的跨度(如果没有比[CLS]更大的跨度，则选择 nil)。或者，Alberti 等人(2019)建议通过 nil 分数将开始 s 和结束 e 的跨度的每个分数 g(s，e)进行归一化：

$$g(s,e) = \text{start}_i + \log P_{\text{end}_i} \text{start}_{\text{CLS}} + \log P_{\text{end}_{\text{CLS}}} \tag{23.19}$$

23.3. 实体连接

我们现在看到了第一个主要的问答范式，基于 IR 的 QA。在讨论第二个主要的问答模式——基于知识的问答之前，我们先介绍实体连接这一重要的核心技术，因为任何基于知识的问答算法都需要实体连接。

实体连接(entity linking)是将文本中的提及与本体中真实实体的表示相关联的任务(Ji 和 Grishman, 2011)。

最常见的事实性问答本体是维基百科，因为维基百科通常是回答问题的文本的来源。在这种用法中，每个惟一的 Wikipedia 页面充当特定实体的惟一 ID。这里有一个任务：决定哪一个与个体相对应的 Wikipedia 页面被一个文本提及所指代。这个决定的任务有自己的名字：**维基化(wikification)**(Mihalcea 和 Csomai，2007)。

自最早的系统(Mihalcea 和 Csomai 2007，Cucerzan 2007，Milne 和 Witten 2008)以来，实体连接(大致)分两个阶段完成：提及检测和提及消歧。我们将提供两种算法：一种是使用锚定词典和 Wikipedia 图结构中的信息的简单经典基线(Ferragina 和 Scaiella，2011)；另一种是现代神经算法(Li 等人，2020)。在这里，我们主要关注实体连接到问题而不是其他体裁的应用。

23.3.1. 基于锚字典和 Web 图的连接

作为简单的基准，我们介绍了 Wikipedia 的 TAGME 连接器(Ferragina 和 Scaiella，2011)，该连接器本身借鉴了较早的算法(Mihalcea 和 Csomai 2007，Cucerzan 2007，Milne 和 Witten 2008)。维基化算法将实体集定义为 Wikipedia 页面集，因此我们将每个 Wikipedia 页面称为唯一实体 e。TAGME 首先创建所有实体的目录(即所有 Wikipedia 页面，删除一些歧义和其他元页面)，然后在像 Lucene 这样的标准 IR 引擎中为它们建立索引。对于每个 e 页面，该算法都会计算一个 in-link 的计数 in(e)：来自其他 Wikipedia 页面的指向 e 的 in-link 总数。这些计数可以从 Wikipedia 转储中得出。

最后，该算法需要一个锚字典。一个锚字典列出了每个维基百科页面，它的锚文本(anchor texts):在其他页面上指向它的超连接跨度。例如，网页为斯坦福大学，http://www.stanford.edu，可能指向从另一个

页面使用锚文本诸如 Stanford 或者 Stanford University:

Stanford University

对每个 Wikipedia 页面 e，通过包含 e 的标题以及所有 Wikipedia 页面上指向 e 的所有锚文本的方式，我们计算 Wikipedia 锚字典。对于每个锚定字符串 a，我们还将在 Wikipedia 中计算其总频率 freq(a)(包括非锚定用法)，a 作为连接出现的次数(我们将其称为 link(a))及其 连接概率 linkprob(a)= link(a)/ freq(a)。需要对最终的锚字典进行一些清理，例如，删除仅由数字或单个字符组成的锚字符串，这些字符串非常罕见，或者由于连接概率很低而不太可能成为有用的实体。

提及检测

给定一个问题(或我们尝试连接的其他文本)，TAGME 通过查询锚定词典中最多 6 个单词的每个符记序列来检测提及。可以使用一些简单的启发式方法(例如，如果子字符串具有较小的连接概率，则对子字符串进行修剪)对这套大序列的修剪进行修剪。问题：

When was Ada Lovelace born?

可能会产生锚节点 Ada Lovelace，也可能是 Ada，但是像 Lovelace 这样的子字符串跨度可能会因为连接概率太低而被修剪，但是像 born 这样的跨度的连接概率太低，它们根本不会出现在锚节点字典中。

提及消歧

如果提及跨度是没有歧义的(仅指向一个实体/维基百科页面)，则完成实体连接！ 但是，许多跨度是有歧义的，可以匹配多个 Wikipedia 实体/页面的锚点。TAGME 算法使用两个因素消除跨度的歧义，这被称为先验概率和相关性/连贯性。第一个因子是 p(e | a)，即跨度指代特定实体的概率。对于每个页面 e∈E(a)，将 a 锚定到 e 的概率 p(e | a)是具有锚定文本 a 的 e 中连接的数量与 a 作为锚点出现的总数的比率：

$$\text{prior}(a \to e) \;=\; p(e|a) = \frac{\text{count}(a \to e)}{\text{link}(a)} \tag{23.20}$$

让我们看看这个因素是如何在以下问题中连接实体的:

What Chinese Dynasty came before the Yuan?

在锚字典中，跨度 Yuan 最常见的联想就是中国货币的名称，即概率 p(Yuan_currency| yuan)非常高。比较少见的与 Yuan 相关的维基百科词条包括常见的中国姓氏(泰国人说的一种语言)，以及正确的实体——中国朝代的名称。所以如果我们只根据 p(e|a)来选择，就会造成错误的消歧，错过正确的环节:元朝。

为了在这种情况下提供帮助，TAGME 使用了第二个因素，即该实体与输入问题中其他实体的相关性。在我们的示例中，问题还包含跨度 Chinese Dynasty 这一事实，该事实与 Dynasties_in_Chinese_history 页面有很高的关联性，应该有助于与 Yuan_dynasty 相匹配。

让我们看看它是如何工作的。给定一个问题 q，对于在 q 中检测到的每个候选锚点，我们为 a 的每个可能实体 e∈Ɛ (a)分配一个相关性得分。连接 a→e 的相关性得分是 e 与 q 中所有其他实体之间的加权平均相关性。两个实体在其 Wikipedia 页面共享许多连接的范围内被认为是相关的。更正式地说，两个实体 A 和 B 之间的相关性计算为：

$$\text{rel}(A,B) \;=\; \frac{\log(\max(|\text{in}(A)|,|\text{in}(B)|)) - \log(|\text{in}(A)| \cap |\text{in}(B)|)}{\log(|W|) - \log(\min(|\text{in}(A)|,|\text{in}(B)|))} \tag{23.21}$$

其中 in(x)是指向 x 的 Wikipedia 页面的集合，W 是汇集(collection)中所有 Wikipedia 页面的集合。锚点 b 对候选注释 a→X 的投票是 b 的所有可能实体与 X 的关联度的平均值，并按其先验概率加权：

$$\text{vote}(b,X) = \frac{1}{|\mathcal{E}(b)|} \sum_{Y \in \mathcal{E}(b)} \text{rel}(X,Y)\, p(Y|b) \tag{23.22}$$

a→X 的总关联度得分是 q 中检测到的所有其他锚点的投票总和:

$$\text{relatedness}(a \to X) = \sum_{b \in \mathcal{X}_q \setminus a} \text{vote} b, X \tag{23.23}$$

为了给 a→X 评分，我们将关联度和先验结合起来，选择关联度最高的实体 X (a→X)，在这个值的小ϵ范围内寻找其他实体，形成这个集合，并从这个集合中选择先验 P(X|a)最高的实体。此步骤的结果是单个实体被分配给 q 中的每个跨度。

TAGME 算法还有另外一个步骤：修剪虚假的锚点/实体配对，取连接概率与连贯性的平均值，再把这个平均值分配给分数：

$$\text{coherence}(a \to X) = \frac{1}{|S| - 1} \sum_{B \in \mathcal{S} \setminus X} \text{rel}(B, X)$$

$$\text{score}(a \to X) = \frac{\text{coherence}(a \to X) + \text{linkprob}(a)}{2} \tag{23.24}$$

最后，如果 score(a→X)<λ，则修剪配对，其中阈值λ设置在保留集上。

23.3.2. 基于神经图的连接

最近的实体连接模型基于双编码器，对候选提及跨度进行编码，对实体进行编码，并计算编码之间的点积。这允许对知识库中所有实体的嵌入进行预先计算和缓存(Wu 等人，2019)。让我们来概述一下 Li 等人(2020)的 ELQ 连接算法。给它一个问题 q 和一组来自 Wikipedia 的候选实体以及相关的 Wikipedia 文本，并输出实体 ID 的元组(e，ms，me)、提及开始和提及结束。如图 23.13 所示，它是通过使用来自 Wikipedia 的文本对每个 Wikipedia 实体进行编码，使用来自问题的文本对每个提及跨度进行编码，以及计算它们的相似度来实现的，如下所述。

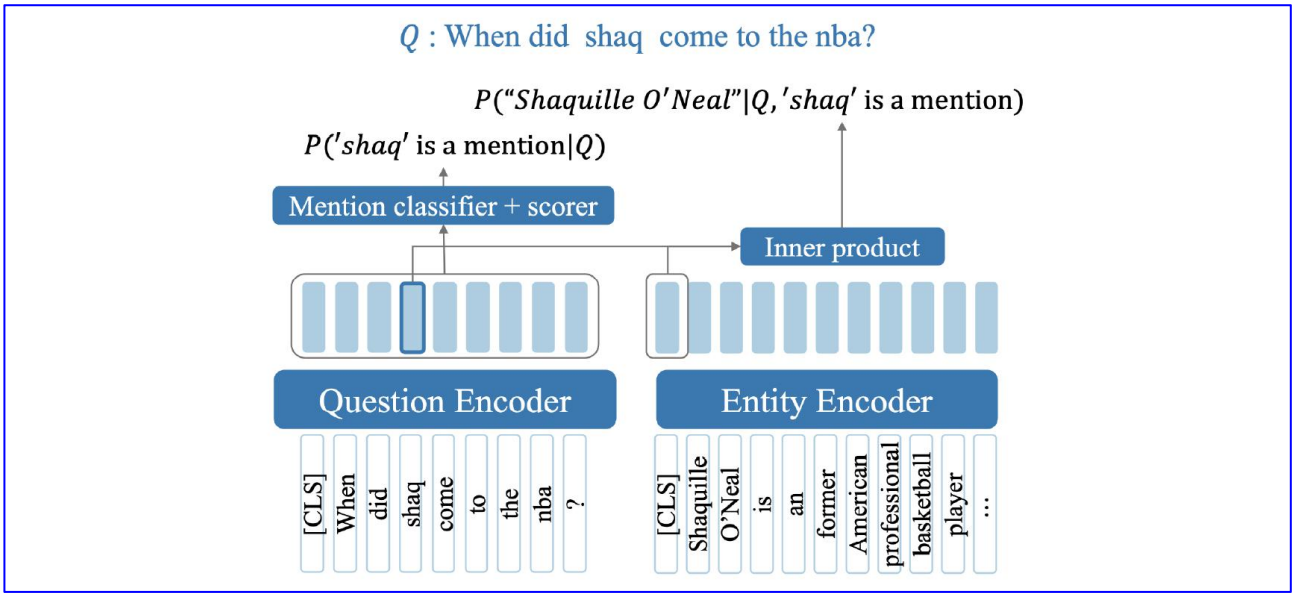


图 23-13：ELQ 算法推理过程示意图

图注：ELQ 算法中用于问题实体连接的推理过程示意图(Li 等人，2020 年)。每个候选问题提及跨度和候选实体分别编码，然后由实体/跨度点积评分。

实体提及检测

为了获得每个问题符记的 h 维嵌入，该算法以常规方式通过 BERT 运行问题：

$$[\mathbf{q}_1 \ldots \mathbf{q}_n] = \text{BERT}([\text{CLS}]q_1 \ldots q_n[\text{SEP}]). \tag{23.25}$$

它计算出 g 中每个跨度[i.j]成为实体提及的似然度，以类似于上面为读者所见的基于跨度的算法的方式。首先，我们计算 i/j 作为提及开始/结束的分数：

$$\mathbf{S}_{\text{start}}(i) = \mathbf{W}_{\text{start}} \cdot \mathbf{q}_i \,, \quad \mathbf{S}_{\text{end}}(j) = \mathbf{W}_{\text{end}} \cdot \mathbf{q}_j \tag{23.26}$$

其中 w_{start} 和 w_{end} 是训练过程中学习的向量。接下来，另一个可训练的嵌入，$w_{mention}$ 被用来计算每一个被提及的符记的得分:

$$\mathbf{S}_{\text{mention}}(t) = \mathbf{W}_{\text{mention}} \cdot \mathbf{q}_t \tag{23.27}$$

然后，通过结合以下三个得分来计算提及概率：

$$p([i,j]) = \sigma\left(s_{\text{start}}(i) + s_{\text{end}}(j) + \sum_{t=i}^{j} s_{\text{mention}}(t)\right) \tag{23.28}$$

实体连接

为了将提及连接到实体，我们接下来为所有 Wikipedia 实体的集合 E = e_1，…，e_i，…，e_w 中的每个实体计算嵌入。对于每个实体 e_i，我们将从实体的 Wikipedia 页面中获取文本，标题为 $t(e_i)$，Wikipedia 页面的前 128 个符记称为描述 $d(ei)$。再通过 BERT 运行，将 CLS 符记 BERT [CLS]的输出作为实体表示：

$$\mathbf{x}_e = \text{BERT}_{[\text{CLS}]}([\text{CLS}]t(e_i)[\text{ENT}]d(e_i)[\text{SEP}]) \tag{23.29}$$

提及跨度可以通过计算连接到实体，其计算方式是：对于每个实体 e 和跨度[i，j]，在跨度编码(符记嵌入的平均值)和实体编码之间，计算点积的相似度。

$$\mathbf{y}_{i,j} = \frac{1}{(j-i+1)} \sum_{t-i}^{j} \mathbf{q}_t$$

$$s(e,[i,j]) = \mathbf{x}_e^{\cdot} \mathbf{y}_{i,j} \tag{23.30}$$

最后，我们使用 softmax 来获得每个跨度的实体分布:

$$p(e|[i,j]) = \frac{\exp(s(e,[i,j]))}{\sum_{e' \in \mathcal{E}} \exp(s(e',[i,j]))} \tag{23.31}$$

训练 ELQ 提及检测和实体连接算法是全监督算法。这意味着，与第 23.3.1 节中的锚定字典算法不同，它需要标记和连接实体边界的数据集。两个这样的标签数据集是 WebQuestionsSP (Yih 等人，2016)，它是来自谷歌搜索问题的 WebQuestions (Berant 等人，2013)数据集的扩展，和 GraphQuestions (Su 等人 2016)。两者在标记和连接的问题中都有实体跨度(Sorokin 和 Gurevych 2018, Li 等人 2020)，导致实体标签版本 WebQSPEL 和 GraphQEL (Li 等人，2020)。在给定训练集的基础上，对 ELQ 提及检测和实体连接阶段进行联合训练，使它们的损失之和达到最优。提及检测损失是一种二元交叉熵损失：

$$\mathcal{L}_{\text{MD}} = -\frac{1}{N} \sum_{1 \le i \le j \le min(i+L-1,n)} \left(y - [i,j] \log p([i,j]) + (1 = y_{[i,j]}) \log(1 - p([i,j]))\right) \tag{23.32}$$

如果[i, j]是一个黄金提及跨度，则 $y_{[i,j]} = 1$，否则为 0。实体连接损失为:

$$\mathcal{L}_{\text{ED}} = -logp(e_g \,|\, [i,j]) \tag{23.33}$$

其中 e_g 为提及的黄金实体[i,j]。

关于 QA 之外的实体连接的其他应用的更多讨论见本章末尾。

23.4. 基于知识的 QA

虽然网络上大量的文本编码了大量的信息，但信息显然也以更结构化的形式存在。我们使用“基于知识的 QA”这个术语是为了通过将自然语言问题映射到结构化数据库上的查询来回答自然语言问题。与基于文本的问题回答范式一样，这种方法可以追溯到自然语言处理的早期，比如像 BASEBALL(Green 等人，1961)这样的系统可以从棒球比赛和统计数据的结构化数据库中回答问题。

基于知识的 QA 使用了两种常见的范式：第一种是基于图的 QA，将知识库建模为图，通常将实体作为节点，将关系或命题作为节点之间的边；第二种是语义解析的 QA，使用我们在第 16 章中看到的语义解析方法。这两种方法都需要某种我们在前一节中描述的实体连接。

23.4.1. 来自 RDF 三元组存储的基于知识的 QA

在完成实体连接之后，让我们介绍一个简单的基于知识的 QA 系统的组件。我们将重点关注基于图的 QA 的最简单的例子，其中数据集是 RDF 三元组形式的一组事实，任务是回答关于其中一个缺失论元的问题。回想一下第 17 章，RDF 是三元组：一个谓词和两个论元，表示一些简单的关系或命题。流行的这种本体通常来自维基百科;DBpedia(Bizer 等人，2009)有超过 20 亿 RDF 三元组，或 Freebase (Bollacker 等人，2008)，现在是 Wikidata 的一部分(Vrandečić和 Krötzsch, 2014)。考虑如下的 RDE 三元组:

主语	谓语	宾语
Ada Lovelace	birth-year	1815

这个三元组可以用来回答诸如“When was Ada Lovelace born?”或“When was Ada Lovelace born?”

存在许多这样的问题数据集。SimpleQuestions(Bordes 等人，2015)包含注释者基于 Freebase 的三元组编写的 10 万个问题。例如，问题“What American cartoonist is the creator of Andy Lippincott?”是根据三元组 (andy lippincott, character created by, garry trudeau)编写的。FreebaseQA(Jiang 等人，2019)将 TriviaQA(Joshi 等人，2017)和其他来源的琐事问题与 Freebase 中的三元组对齐，例如将琐事问题“Which 18th century author wrote Clarissa (or The Character History of a Young Lady), said to be the longest novel in the English language? ”与三元组 (Clarissa, book.written-work.author, Samuel Richardson)对齐。另一个这样的数据集家族来自 WEBQUESTIONS(Berant 等人，2013)，其中包含 5,810 个由网络用户提出的问题，每个问题都以 wh-字开头，仅包含一个实体，并与从 Freebase 页面的手写答案配对问题的实体。WEBQUESTIONSSP(Yih 等人 2016)通过人为创建的语义解析(SPARQL 查询)增强了 WEBQUESTIONS 的使用 Freebase 可回答的那些问题。COMPLEXWEB-QUESTIONS 通过组合和其他类型的复杂问题来增强数据集，从而产生 34,689 个问题以及答案、网络摘要和 SPARQL 查询(Talmor 和 Berant，2018)。

假设我们已经完成了上一节介绍的实体连接阶段。因此，我们已经将 Ada Lovelace 之类的文字说明映射到知识库中的规范实体 ID。对于简单的三元组关系问题回答，下一步是确定要询问的关系，从诸如“When was ... born”的字符串映射到知识库中的规范关系诸如 birth-year。可以将合并后的任务勾画为：

“When was Ada Lovelace born?” → birth-year (Ada Lovelace, ?x)

“What is the capital of England?” → capital-city(?x, England)

下一步是关系检测和连接。对于简单的问题，我们假设问题只有一个关系，关系检测和连接可以以类似神经实体连接模型的方式进行:计算问题文本的编码和每个可能关系的编码之间的相似度(通常通过点积)。例如，在(Lukovnikov 等人，2019)的算法中，为了关系检测的目的，使用 BERT 模型的 CLS 输出来表示问题跨度，并为每个关系 r_i 训练一个单独的向量。然后通过 softmax 对点积计算特定关系 r_i 的概率:

$$\mathbf{m}_r = \text{BERT}_{\text{CLS}}([\text{CLS}]q_1 \cdots q_n[\text{SEP}])$$

$$s(\mathbf{m_r}, r_i) = \mathbf{m_r} \cdot \mathbf{w_{r_i}}$$

$$p(r_i|q_1, \cdots, q_n) = \frac{\exp(s(\mathbf{m_r}, r_i))}{\sum_{k=1} N_R \exp(s(\mathbf{m_r}, r_k))} \tag{23.34}$$

答案的排名

大多数算法都有最后一个阶段，该阶段取实体和关系推理步骤返回的最前面的 j 个实体和最前面的 k 个关系，在知识库中搜索包含这些实体和关系的三元组，然后对这些三元组进行排名。这个排名可以是启发式的，例如，根据提及跨度和实体文本别名之间的字符串相似性来为每个实体/关系配对评分，或者优先考虑具有高的 in-degree(由许多关系连接)的实体。或者，可以通过训练分类器以采用拼接的实体/关系编码并预测概率来完成排名。

23.4.2. 语义解析的 QA

第二类基于知识的 QA 使用语义解析器将问题映射到结构化程序以生成答案。这些逻辑形式可用谓词演算的某些版本，一种查询语言如 SQL 或 SPARQL，或其他一些可执行程序的形式，如图 23.14 所示。

Question	Logical form
What states border Texas?	λx:state(x)∧borders(x; texas)
What is the largest state?	argmax(λx:state(x);λx:size(x))
I'd like to book a flight from San Diego to Toronto	SELECT DISTINCT f1.flight id FROM flight f1, airport service a1, city c1, airport service a2, city c2 WHERE f1.from airport=a1.airport code AND a1.city code=c1.city code AND c1.city name= 'san diego' AND f1.to airport=a2.airport code AND a2.city code=c2.city code AND c2.city name= 'toronto'
How many people survived the sinking of the Titanic?	(count (!fb:event.disaster.survivors fb:en.sinking of the titanic))
How many yards longer was Johnson's longest touchdown compared to his shortest touchdown of the first quarter?	ARITHMETIC diff(SELECT num(ARGMAX(SELECT)) SELECT num(ARGMIN(FILTER(SELECT))))

图 23-14：语义解析器为回答问题而生成的示例逻辑形式

图注：其中包括来自 GeoQuery 美国地理问题数据库中的两个问题(Zelle 和 Mooney，1996)带有谓词演算表示形式，一个 ATIS 问题带有 SQL(Iyer 等人，2017)，一个 Freebase 关系的程序，一个 QDMR 程序（Wolfson 等人，2020)。

因此，问题的逻辑形式要么是查询形式，要么很容易转换为查询形式(例如，谓词演算可以转换为 SQL)。数据库可以是一个完整的关系数据库，也可以是其他一些结构化的知识库。

正如我们在第 16 章中所看到的那样，语义解析算法可以由与手工构建的逻辑形式配对的问题进行完全监督，或者由与答案(表示法)配对的问题进行弱监督，其中逻辑形式仅被建模为潜在变量。

对于完全受监管的情况，我们可以从诸如以下三种数据集中获得一组问题及其正确的逻辑形式，第一是有关美国地理问题的 GEOQUERY 数据集(Zelle 和 Mooney，1996)，第二是需要推理的复杂问题(在历史和足球比赛方面)的 DROP 数据集(Dua 等人，2019)，第三是航班查询的 ATIS 数据集。所有版本的数据集都有 SQL 或其他逻辑形式的版本(Iyer 等人 2017, Wolfson 等人 2020, Oren 等人 2020)。

接下来的任务就是利用这些成对的训练元组，生成一个系统，将新问题映射到它们的逻辑形式。常见的基线算法是一个简单的序列到序列模型，例如使用 BERT 来表示问题符记，将它们传递给第 11 章的

biLSTM 编码器解码器，如图 23.15 所示。

第 16 章中描述的任何其他语义解析算法也都是合适的。

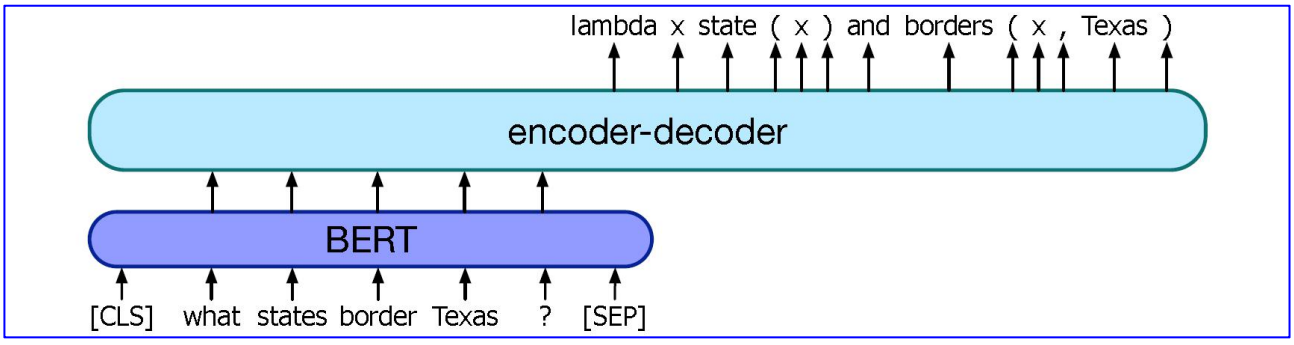


图 23-15：编码器-解码器语义解析器

图注：编码器-解码器语义解析器，用于将问题转换为逻辑形式，带有一个 BERT 预编码器和一个编码器-解码器(biLSTM 或 Transformer)。

23.5. QA 中的语言模型(T5)

进行 QA 的另一种方法是查询预训练的语言模型，迫使模型仅从其参数中存储的信息来回答问题。例如 Roberts 等人(2020 年)使用 T5 语言模型，这是一种经过预编码以填充掩码的任务跨度的编码器-解码器体系结构。图 23.16 示出了该体系结构；删除的跨度用<M>标记，并且训练系统使解码器生成丢失的跨度(用<M>分隔)。

Roberts 等人(2020)然后通过给 T5 系统一个问题，并训练它在解码器中输出答案文本，来对 T5 系统进行微调。使用最大 110 亿参数的 T5 模型具有竞争优势，尽管还不如专门为 QA 而设计的系统那么好。

语言建模还不是一个完整的 QA 解决方案。例如，除了不能很好地工作之外，它们还具有较差的解释性(例如，与标准 QA 系统不同，它们目前无法通过告诉用户答案来自何处而为用户提供更多背景信息)。尽管如此，从语言模型中提取答案的研究仍是未来问题研究的一个有趣领域。

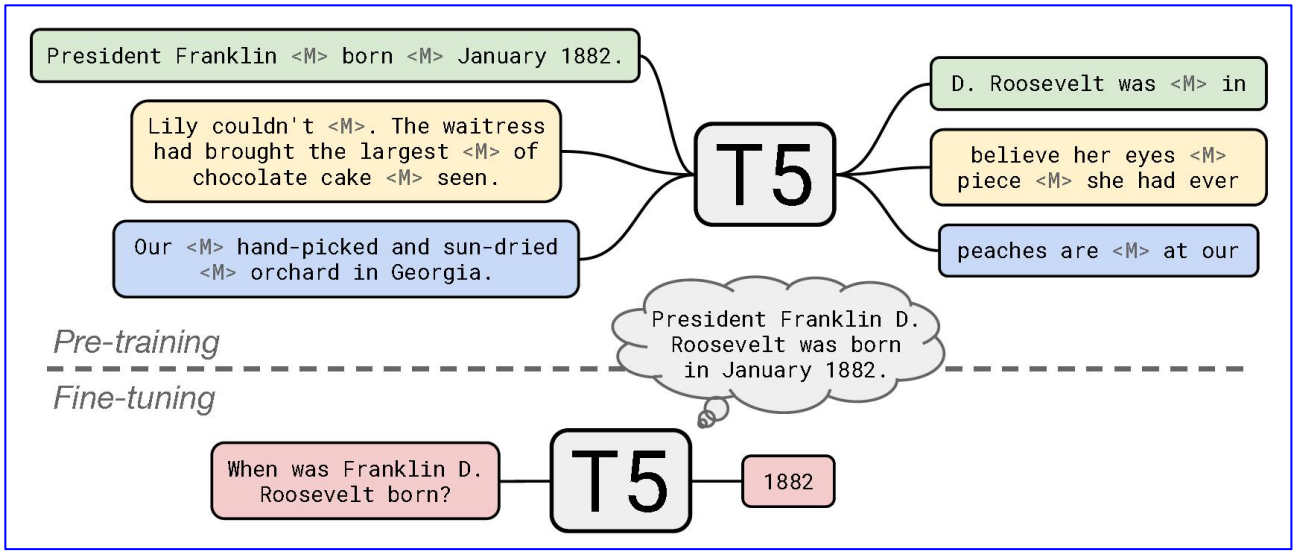


图 23-16：T5 系统的编码器-解码器架构

图注：在预训练中，它通过在解码器中生成缺失的跨度(以<M>分隔)来学习填补任务的掩码跨度(以<M>标记)。然后在 QA 数据集上进行微调，给出问题，不添加任何额外的上下文或段落。数据来自 Roberts 等人(2020)。

23.6. 分类 QA 模型

虽然神经体系结构是用于 QA 的最先进技术，但预神经体系结构使用规则和基于特征的分类器的混合，有时可以获得更高的性能。这里我们总结一个有影响力的分类系统，来自 IBM 的沃森 DeepQA 系统，赢得了《危险边缘》2011 年的挑战(图 23.17)。

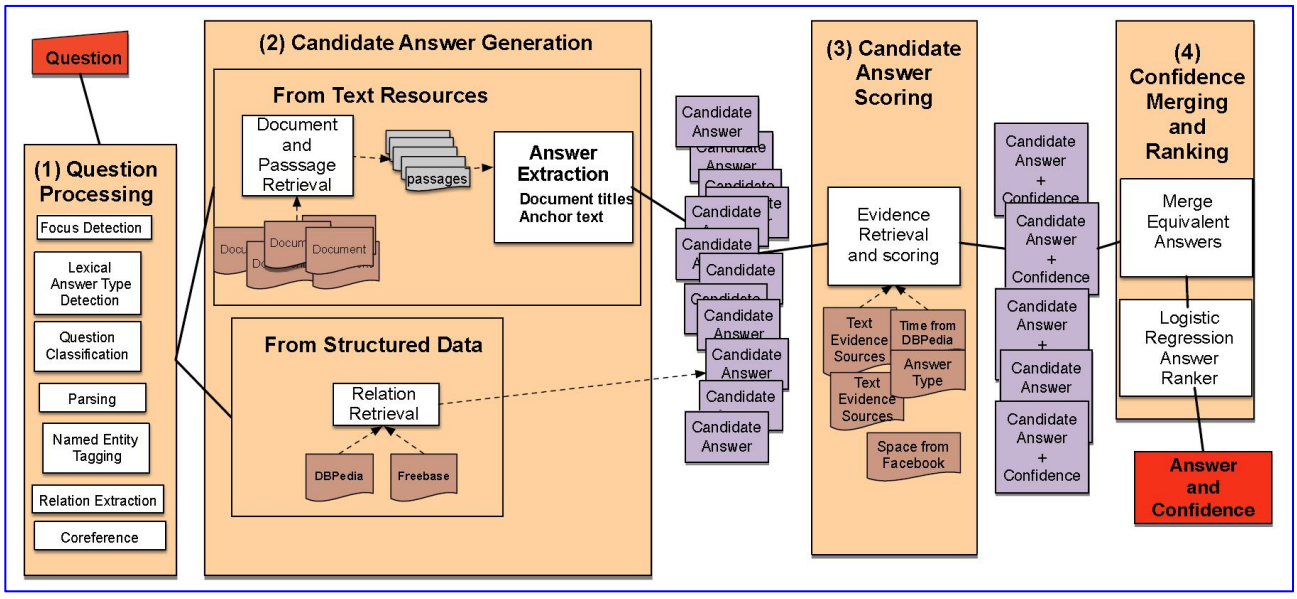


图 23-17：Watson QA 的 4 个主要阶段

图注：Watson QA 的 4 个主要阶段：(1) 问题处理，(2) 候选答案生成，(3) 候选答案评分，(4) 答案合并和置信度评分。

让我们考虑一下它是如何处理这些《危险边缘》的例子，每个都有一个类别，后面跟着一个问题：

Poets and Poetry: **He** was a bank clerk in the Yukon before he published “Songs of a Sourdough” in 1907.

THEATRE: A new play based on **this Sir Arthur Conan Doyle canine** **classic** opened on the London stage in 2007.

问题处理

在此阶段，问题被解析，命名实体被提取(Sir Arthur Conan Doyle 被确定为 PERSON，Yukon 被确定为 GEOPOLITICAL ENTITY，“ Sourdough of Sourdough”被确定为 COMPOSITION)，共指被执行(he 与 clerk 联系在一起)。

两个例子中以粗体显示的问题焦点(focus)被提取出来。焦点是问题中与答案相关联的单词串。它很可能被任何找到的答案字符串中的答案所取代，因此可以用来与支持的段落对齐。在 DeepQA 中，焦点是通过手写规则提取的——这得益于《危险边缘》中相对风格化的句法。问题——比如 Conan Doyle 的例子中提取含有限定词“this”的名词短语的规则，以及 Poet 的例子中提取代词 she, he, hers, him 的规则。

词汇答案类型(lexical answer type)(上面以蓝色显示)是一个或多个单词，它们告诉我们有关答案语义类型的信息。由于《危险边缘》中的问题种类繁多，因此 DeepQA 会选择各种各样的单词作为答案类型，而不是一小组命名的实体。这些词汇答案类型再次由规则提取：默认规则是选择焦点的句法中心词。其他规则改进了此默认选择。例如，附加的词汇答案类型可以是问题中与焦点相关或具有特定句法关系的单词，例如焦点的代词或谓词主格的中心词。在某些情况下，甚至是《危险边缘》类别是指代与其他词汇答案类型兼容的实体类型，则类别可以充当词汇答案类型。因此，在上面的第一种情况下，he、poet 和 clerk 都是词汇答案类型。除了将规则直接用作分类器外，它们还可以用作逻辑回归分类器中的特征，该逻辑分类器可以返回概率以及词汇答案类型。这些答案类型将在以后的“候选答案评分”阶段用作每个候选的证据来源。还提取了类似以下的关系：

authorof(focus,“Songs of a sourdough”)
publish (e1, he, “Songs of a sourdough”)
in (e2, e1, 1907)
temporallink(publish(...), 1907)

最后将题目按类型分类(定义题、多项选择题、益智题、填空题)。这通常是通过在单词或解析树上编写模式匹配正则表达式来实现的。

候选答案生成

接下来，我们将处理后的问题与外部文档和其他知识源相结合，以从文本文档和结构化知识库中建议许多候选答案。我们可以通过关系和已知实体查询诸如 DBpedia 或 IMDB 之类的结构化资源，就像我们在 23.4 节中看到的那样。因此，如果我们提取了关系 authorof(focus，“Songs of a sourdough”)，则我们可以查询一个带有 authorof(？x，“Songs of a sourdough”)的三元组以返回作者。

为了从文本中提取答案，DeepQA 使用了检索和阅读的简单版本。例如，对于 IR 阶段，DeepQA 通过消除停同词生成一个问题，然后提高与焦点相关的任何术语的权重。例如，从这个查询:

MOVIE-“ING”: Robert Redford and Paul Newman starred in this depression-era grifter flick. (Answer: “The Sting”)

下面的加权查询可能会被传递到一个标准的 IR 系统:

(2.0 Robert Redford) (2.0 Paul Newman) star depression era grifter (1.5 flick)

DeepQA 还利用了方便的事实，即绝大多数《危险边缘》的答案是 Wikipedia 文档的标题。要找到这些标题，我们可以专门针对 Wikipedia 文档进行第二次文本检索。然后，我们不是从检索的 Wikipedia 文档中提取段落，而是直接返回排名较高的检索文档的标题作为可能的答案。

一旦有了一组段落，就需要提取候选答案。如果文档恰好是 Wikipedia 页面，我们可以仅使用标题，但是对于其他文本(例如新闻文档)，我们需要其他方法。两种常见的方法是提取文档中的所有锚文本(anchor texts)(锚文本是<a>和之间的文本，用于指向 HTML 页面中的 URL)，或者提取作为维基百科文档标题的段落中的所有名词短语。

候选答案评分

接下来，DeepQA 使用许多证据来对每个候选进行评分。这包括一个分类器，用于对候选答案是否可以解释为潜在答案类型的子类或实例进行评分。考虑候选“难吞咽(difficulty swallowing)”和词汇答案类型“表现(manifestation)”。DeepQA 首先将这些单词中的每个单词与诸如 DBpedia 和 WordNet 之类的本体中的可能实体匹配。因此，候选“困难吞咽(difficulty swallowing)”与 DBpedia 实体“吞咽困难症(Dysphagia)”匹配，然后将该实例映射到 WordNet 类型“症状(Symptom)”。答案类型“表现(manifestation)”映射到 WordNet 类型“条件(Condition)”。系统会寻找一个下位词或同义词连接，在这种情况下，将找到“症状(Symptom)”和“条件(Condition)”之间的下位词。

其他得分是基于使用时间和空间关系提取的 DBpedia 或其他结构化数据库。例如，我们可以提取实体的时间属性(一个人何时出生，何时死亡)，然后与问题中的时间表达式进行比较。如果问题中的时间表达出现在一个人出生之前，那就证明这个人不是问题的答案。

最后，我们可以使用文本检索来帮助检索支持候选答案的证据。我们可以检索与问题匹配的文章，然后用候选答案替换问题中的焦点，用修改后的问题测量重叠的单词或文章的顺序。

这个阶段的输出是一组候选答案，每个答案都有一个评分的特征向量。

答案合并和计分

DeepQA 最终合并了等效的候选答案。因此，如果我们提取了两个候选答案 J.F.K.和 John F. Kennedy，此阶段会将两者合并为一个候选者，例如，使用上面描述的实体连接锚定字典，该字典将列出 Wikipedia 标题的许多同义词(例如，JFK, John F. Kennedy, Senator John F. Kennedy, President Kennedy, Jack Kennedy)。然后，我们合并每个变体的证据，将合并候选者的评分特征向量合并到单个向量中。

现在我们有了一组候选项，每个候选项都有一个特征向量。分类器采用每个特征向量，并为该候选答案分配一个置信度值。对分类器进行了数千个候选答案的训练，每个答案都标明了正确与否以及它们的特征向量，并学会预测成为正确答案的可能性。由于在训练中，不正确的答案比正确的答案要多得多，因此我们需要使用一种标准技术来处理非常不平衡的数据。DeepQA 使用实例权重，为训练中的每个错误答案示例分配 0.5 实例权重。然后，根据此置信度值对候选答案进行排序，从而得出一个最佳答案。

因此，DeepQA 的基本直觉是从基于文本和基于知识的来源中提出大量候选答案，然后使用丰富的证据功能为这些候选者评分。有关更多详细信息，请参见本章末尾提到的论文。

23.7. 事实性答案的评估

在 1999 年的 TREC Q / A 跟踪中，针对事实性问题回答的通用评估指标是**平均倒数排名(mean reciprocal rank，MRR)**。MRR 假设测试题集已被人为标签为正确答案。MRR 还假设系统正在返回简短的答案列表或包含答案的段落。然后，根据第一个正确答案的等级的倒数对每个问题打分。例如，如果系统返回了五个答案，但前三个答案是错误的，因此排名最高的正确答案排名第四，则该问题的倒数得分将为 1/4。带有不包含任何正确答案的返回集的问题被分配为零。那么，系统的分数就是该集合中每个问题的分数的平均值。更正式地说，对于评估系统返回由 N 个问题组成的测试集的一组排名答案的评估，MRR 定义为：

$$\text{MRR}=\frac{1}{N}\sum_{i=1}^{N}{}_{s.t.rank_i\neq 0}\frac{1}{rank_i} \tag{23.35}$$

像 SQuAD 这样的数据集上的阅读理解系统通常使用两个指标进行评估，这两个指标都忽略了标点符号和冠词(a, an, the) (Rajpurkar 等人，2016):

•精确匹配:预测答案与黄金答案完全匹配的百分比。

•F1 得分:预测答案和黄金答案之间的平均重叠。将预测和黄金答案视为一袋符记，并计算 F1，对所有问题的 F1 进行平均。

有许多测试集可用于回答问题。早期系统使用 TREC QA 数据集;从 1999 年到 2004 年，TREC 竞赛的问题和手写的答案是公开的。最近的比赛使用第 23.2.1 节中描述的各种数据集。

除了第 23.2.1 节讨论的数据集之外，还有很多不同的数据集用于训练和测试阅读理解/答案抽取。有些数据集结构是基于这样一个事实:为儿童设计的阅读理解任务往往是多项选择题，即从给出的答案中进行选择。MCTest 数据集使用了这种结构，其中有 500 个虚构的短篇故事，由众包工作者用问题和多项选择答案创建(Richardson 等人，2013)。AI2 推理挑战(ARC) (Clark 等人，2018)的问题设计得很难通过简单的词汇方法回答:

> Which property of a mineral can be determined just by looking at it?
>
> (A) luster [correct] (B) mass (C) weight (D) hardness

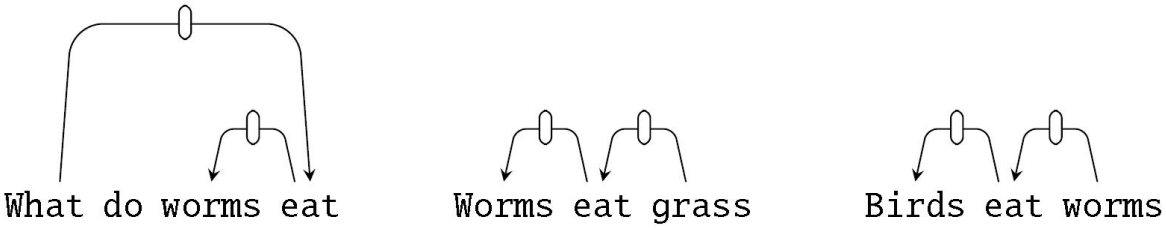
这个 ARC 示例很困难，因为正确的答案 luster(色泽)不太可能在网络上频繁出现诸如 looking at it(看它)这样的短语，而 mineral(矿物)一词与错误的答案 hardness(硬度)高度相关。

23.8. 总结

(无)

23.9. 文献与历史说明

问题回答是最早的自然语言处理任务之一，基于文本和基于知识的范式的早期版本是在 20 世纪 60 年代早期发展起来的。基于文本的算法通常依赖于对问题和文档中的句子进行简单的解析，然后寻找匹配。这种方法很早就被使用过(Phillips, 1960)，但 Simmons 等人(1964)的 Protosynthex 系统可能是最完整的早期系统，它惊人地预示了现代基于关系的系统。Protosynthex 首先从问题的内容词中形成一个查询，然后在文档中检索候选回答句，根据它们与问题的词重叠频率加权排序。然后使用依存分析器对查询和检索到的每个句子进行解析，并对其结构与所选问题结构最匹配的句子进行解析。因此，问题 What do worms eat?会匹配 worms eat grass：在当时使用的依存语法版本中，两者都将主语 worms 作为 eat 的依赖项，而 birds eat worms 作为主语:



另一种基于知识的范式在 BASEBALL 系统中得到了应用(Green 等人 1961)。这个系统通过查询一个结构化的比赛信息数据库来回答有关棒球比赛的问题，比如“Where did the Red Sox play on July 7(7 月 7 日红袜队在哪里比赛)”。将数据库存储为一种属性-值矩阵，每个比赛的属性值为:

```
Month = July
    Place = Boston
        Day = 7
        Game Serial No. = 96
        (Team = Red Sox, Score = 5)
        (Team = Yankees, Score = 3)
```

每个问题都是使用 Zellig Harris 在宾夕法尼亚大学的 TDAP 项目的算法进行成分解析，本质上是一个串联的有限状态转录机(参见 Joshi 和 Hopely 1999，Karttunen 1999 的历史讨论)。然后，在内容分析阶段，每个单词或短语都与计算其部分意义的程序相关联。因此，短语“Where”有代码来指定语义 Place = ?，带来的结果是“Where did the Red Sox play on July 7”这个问题被赋予了意义：

```
Place = ?
Team = Red Sox
Month = July
Day = 7
```

然后将问题与数据库进行匹配以返回答案。Simmons(1965)总结了其他早期的 QA 系统。以知识为基础的问答范式的另一个重要先驱是使用谓词演算作为含义表示语言的工作。LUNAR 系统(Woods 等人 1972, Woods 1978)被设计为一个月球地理事实数据库的自然语言接口。它可以通过将样本解析为逻辑形式来回答诸如“Do any samples have greater than 13 percent aluminum”之类的问题。

```
(TEST (FOR SOME X16 / (SEQ SAMPLES) : T ; (CONTAIN' X16
(NPR* X17 / (QUOTE AL203)) (GREATERTHAN 13 PCT))))
```

网络的兴起将问答的信息检索范式推到了前台。美国政府赞助的 TREC(文本检索会议)评估，自 1992 年以来每年进行一次，为评估信息检索任务和技术提供了一个测试平台。TREC 提供了训练和测试的大型文档集，以及统一的评分系统(Voorhees 和 Harman, 2005)。所有会议的细节可以在国家标准和技术协会网站上的 TREC 页面找到。1999 年，TREC 增加了一个有影响力的 QA 跟踪，导致各种各样的实况和非实况系统在年度评估中竞争。

与此同时，Hirschman 等人(1999)引入了利用儿童阅读理解测试来评估机器文本理解算法的想法。他们获得了一个由 120 篇文章组成的语料库，每篇文章有 5 个问题，每个问题都是为 3-6 年级的孩子设计的，他们建立了一个答案抽取系统，并测量了他们的系统给出的答案与考试发布者给出的答案的对应程度。他们的算法将单词重叠作为一个特征;后来的算法在问题和答案之间增加了命名实体特征和更复杂的相似性(Riloff 和 Thelen 2000, Ng 等人 2000)。

神经阅读理解系统借鉴了这些早期系统的见解，这些早期系统的答案应该关注问题和段落的相似性。许多现代系统的架构轮廓在早期的工作中被提出，如 Hermann 等人(2015a)、Chen 等人(2017)和 Seo 等人(2017)。

待定:更近的 QA 历史。

在 IBM 研发杂志的第 56 卷中的一系列论文中描述了赢得 Jeopardy！挑战的 Watson 系统的 DeepQA 组件。参见例如 Ferrucci(2012)。其他问答任务包括测验碗(Quiz Bowl)，由于可以中断问题，因此需要考虑时间安排(Boyd-Graber 等人，2018)。问题回答也是现代个人助理对话系统的重要功能。有关更多信息，请参见第 24 章。

23.10. 练习

(无)