第 3 章　大模型理论基础

在探索领域特定的算法架构之前，充分了解目标领域的数据特点和结构是至关重要的。数据特点和结构直接影响选择合适的算法与构建适用的模型。通过深入了解数据的属性、分布、关联性及可能存在的潜在模式，可以更好地把握问题的本质，有针对性地设计和调整算法架构，从而提高模型的性能和效果。因此，在进行算法研究和开发之前，对数据进行全面的分析和理解是必不可少的步骤，这有助于我们更好地把握问题的本质，并为解决方案的设计提供有效的指导。

本章将从数据开始，介绍自然语言处理（Natural Language Processing，NLP）中的时间序列格式数据及数据的预处理技术，包括分词、Token 和 Embedding 等大模型中常见的基础概念。Transformer 架构不是一蹴而就的，本章将从历史发展的角度讲述其来龙去脉，包括从最初的统计语言模型、神经网络模型到如今的大模型。

3.1　自然语言领域中的数据

3.1.1　时间序列数据

在深度学习领域，特定领域的算法和架构的设计是基于该领域内数据的独特特征。举例来说，在处理图像数据时，研究人员广泛采用卷积神经网络，以便有效地捕获图像的空间特征。相比之下，处理受到噪声干扰的数据，如文本数据，则常使用自动编码器结构，这种结构能够有效地处理噪声并生成清晰的输出数据。因此，深入理解特定领域的数据特点和结构对于设计该领域的算法架构至关重要。这一观点同样适用于自然语言处理领域。在探索领域特定的算法架构之前，充分了解目标领域的数据特点和结构尤为关键。

在自然语言处理领域，序列数据是至关重要的。与非序列数据不同，序列数据中的样本之间存在特定的顺序关系，这一顺序关系在很大程度上决定了数据的含义和解释。在处理序列数据时，样本的顺序不仅是一种表现形式，更是数据的本质特征之一。举例来说，在文本数据中，一句话中的词语顺序往往决定了句子的含义和语义逻辑。因此，无论是在序列标注、机器翻译、文本生成还是其他自然语言处理任务中，都必须考虑样本的顺序信息。相比之下，在非序列数据中，

如在图像数据或结构化数据中，样本之间的顺序关系并不具备重要性，因为这些数据中的样本通常是相互独立的，其顺序变化或部分样本缺失不会对数据的含义产生显著影响。然而，在序列数据中，即使是微小的顺序变化或样本的缺失，都可能导致数据的含义发生巨大变化。因此，在处理序列数据时，必须特别注意样本顺序的保持和数据的完整性。

最典型的序列数据有以下几种类型。

（1）文本数据：包括句子、段落、文章等，其中词语的排列顺序对语义有重要影响。正如学生考试时，曾出现的试题（连词成句），若答题连词不当，一整句话都将被评为不得分。

（2）时间序列数据：包括股票价格随时间变化的数据，气象中的温度、湿度、风速等随时间变化的数据，心电图中的心率随时间变化的数据等。

（3）音频数据：音频信号中的波形数据，如语音信号、音乐等，其中音频的采样点是按时间顺序排列的。

（4）视频数据：视频是由一系列帧组成的，每帧都是图像数据，它们按照时间顺序排列以形成连续的视频流。

（5）DNA 序列数据：生物学领域中常见的 DNA 序列数据，包含基因序列、蛋白质序列等，它们的排列顺序对生物功能具有重要意义。

（6）符号序列数据：包括乐谱、密码等，其中符号的排列顺序决定了其含义和功能。

类似的数据还有很多。例如，在金融领域，股票交易数据是一种重要的时间序列数据。股票价格和交易量等变量的时间顺序对于预测股票价格趋势不可或缺。此外，社交媒体数据也是一种序列数据，每个用户发布的帖子、评论或互动都形成了一个个时间序列，理解这些序列数据有助于分析用户的行为模式和趋势。另外，传感器数据也是常见的序列数据类型，例如用于监控环境的温度、湿度和气压数据。在工业领域，机器运行状态的监测数据也是重要的序列数据，对机器的预测性维护至关重要。显而易见，处理序列数据时，我们需要确保算法不仅能够理解单个样本，还能够学习样本之间的关联。现今，这些能够学习样本之间关系的算法已经成为自然语言处理架构中重中之重的一部分。

3.1.2 分词

在自然语言处理中，文本数据的样本之间的联系，即词与词之间、字与字之间的联系是通过语义连接实现的。因此，在文字序列中，每个样本通常表示一个单词或一个字。对于英文数据而言，大多数情况下一个样本是一个单词，偶尔也可能是更小的语言单位，如字母或半词。在中文数据中，一个二维表通常对应一个句子或一段话，而单个样本则表示一个单词或字。

尽管原始文本数据大多是段落形式的，但在深度学习中，将文本数据的样本分割为词或字作为输入。因此，对文本数据通常需要进行分词处理，即将连续的文本划分为具有独立意义的词或词组。良好的分词可以降低算法理解文本的难度，并提升模型的性能。例如，将句子“轻舟已过万重山”分为“轻舟”“已过”“万重山”三个词，可能比将其分为“轻”“舟”“已”“过”“万”“重”“山”七个字更易于理解；且要求每个字都自带完整语义，实际上会带来一些困难。

由于不同语言具有不同的特点，因此每种语言所采用的分词方式也不同。例如，英文等拉丁语系的语言通常通过空格来分割单词，只需按照空格进行分词即可自然得到良好的结果。而中文、日文和韩文等语言没有空格用于分割，因此分词可能会面临挑战。特别是对于中文，甚至需要考虑断句对语义理解的影响，因为不同的分词结果可能导致不同的语义解读。

现在，对于不同语言的分词，我们都有丰富的操作手段。

1．中文分词

中文分词是将连续的中文文本切分成有意义的词或词组的过程，是自然语言处理中的一个基础任务。以下是一些常见的中文分词方法。

（1）基于词典的分词方法：基于词典的分词方法使用预先构建的词典进行分词。文本中的词如果出现在词典中，则被认为是一个词，否则，将文本进一步细分为字或其他子单位。常见的词典包括哈工大、北大等机构提供的大型词典，以及一些开源的词典资源。

（2）基于统计的分词方法：基于统计的分词方法通常利用语料库中的统计信息来确定词语边界。其中，最常见的方法是最大匹配法，即从左到右或从右到左地在文本中寻找最长的匹配词。另一种方法是隐马尔可夫模型（Hidden Markov Model，HMM），它通过学习词语出现的概率和转移概率来确定最可能的词语切分。

（3）基于规则的分词方法：基于规则的分词方法通过定义一系列规则来切分文本。这些规则既可以是基于语言学和语法规则的，也可以是基于模式匹配的。例如，利用中文的语法规则和词性信息来确定词语边界。

（4）基于深度学习的分词方法：近年来，随着深度学习的发展，基于深度学习的中文分词方法也逐渐流行起来。这些方法通常利用神经网络模型，如循环神经网络（Recurrent Neural Network，RNN）、长短期记忆（Long Short-Term Memory，LSTM）网络、卷积神经网络（Convolutional Neural Networks，CNN）或者自注意力（Self-attention）机制（Transformer），通过大规模数据学习中文文本的词语边界。

2．常用工具

以下是一些常用的中文分词工具，它们具有不同的特点和适用场景，可以根

据具体需求选择合适的工具进行中文分词。对于大部分应用，例如简单的文本预处理，一般使用 Jieba（结巴分词）就足够了。但是，如果需要更深入的语言学特性或高准确性的处理，可能需要考虑 THULAC 或 LTP 等更全面的工具。

（1）Jieba：Jieba 是一个常用的开源中文分词工具，具有简单易用、高效稳定的特点。它基于前缀词典实现，支持精确模式、全模式、搜索引擎模式等多种分词方式，并提供自定义词典和关键词提取等功能。

（2）THULAC：THULAC 是由清华大学自然语言处理与社会人文计算实验室开发的中文词法分析工具包，具有较高的分词准确率和效率。它采用基于词语和字符的联合标注模型，支持词性标注和命名实体识别等功能。

（3）哈工大 LTP：哈工大 LTP（Language Technology Platform，语言技术平台）是一个集成了多种自然语言处理工具的平台，其中包括中文分词器。哈工大 LTP 的分词器基于隐马尔可夫模型和条件随机场（Conditional Random Field，CRF）模型，具有较高的分词准确率和速度。

（4）NLPIR：NLPIR（其前身称为 ICTCLAS）是由中国科学院计算技术研究所开发的中文分词系统，具有较好的分词效果和稳定性。它基于词频统计和规则匹配的方法实现，支持多种分词模式和自定义词典。

（5）斯坦福分词器（Stanford Segmenter）：斯坦福分词器是由美国斯坦福大学开发的中文分词工具，它基于条件随机场模型，具有较高的分词准确率。它还支持英文分词和多语言分词等功能。

3. 英文分词

英文分词相对于中文分词来说，相对简单，因为英文单词通常以空格或标点符号分隔，但仍然存在一些特殊情况需要处理。以下是一些常用的英文分词方法。

（1）基于空格分词：最简单的英文分词方法是根据单词之间的空格进行分词。这种方法适用于大多数情况，但在一些特殊的文本中，例如在网页文本或非标准文本中可能存在连续的字符序列没有空格分隔。

（2）基于标点符号分词：在英文文本中，标点符号（如句号、逗号、分号等）通常表示句子的结束或分隔不同的短语。因此，可以先根据标点符号将文本分割成句子或短语，然后再对每个句子或短语进行单词级别的分词。

（3）词干提取（Stemming）和词形还原（Lemmatization）：词干提取和词形还原是一种将单词归约为其词干或原型的方法。词干提取通过去除单词的后缀来获取其词干，而词形还原则是将单词转化为其在词典中的原型形式。这两种方法通常用于处理单词的不同形态，以便统一表示同一概念的不同变体。

以上是常用的英文分词方法，可根据具体需求和应用场景选择合适的方法进行分词处理。

3.1.3　Token

Token（词元，其读音为['təʊkən]）是自然语言处理世界中的重要概念，这个概念没有官方中文译名，但我们可以根据该词语在众多论文中的语境对其进行如下定义：Token 是当前分词方式下的最小语义单元，根据分词方式的不同，对于英文，它可能是一个单词（如“upstair”）、一个半词（如“up、stair”）或一个字母（如“u、p、s、t、a、i、r”），也可能是一个短语（如“攀登高峰”）、一个词语（如“攀登、高峰”）或一个字（如“攀、登、高、峰”）。如前所述，分词是将连续的文本切分成一个个具有独立意义的词或词组的过程，本质上来说，分词就是分割 Token 的过程，因此文字数据表单中的一行行样本也就是一个个 Token。

Token 在自然语言处理中扮演着重要的角色。首先，它是语义的最小组成部分，也是深度学习算法处理输入数据的基本单元。Token 的数量反映了文本的长度和算法需要处理的数据量，直接影响算法的资源消耗和模型的性能。在 NLP 领域，OpenAI 等模型开发厂商常根据 Token 的使用情况来制定计价和使用限制。因此，Token 的使用量也成为衡量大模型性能的重要指标。随着大模型的不断发展，Token 已成为评估 NLP 模型吞吐量和数据量的标准单位。表 3-1 为 OpenAI API 收费标准（其官方可能会变更收费标准，以实际为准）。

表 3-1　OpenAI API 收费标准

API 版本	功能	收费方式
GPT-3.5	完成基础的 NLP 任务	每 10 万个 Token 收取 4 美分
GPT-4	提供更先进、精准的 NLP 功能	提供两个档位供开发者选择
ChatGPT	完成高质量的自然语言会话任务	基于使用情况的收费模式

在 OpenAI 的官方网站上，可以轻松地找到一个名为 Tokenizer 的工具，它用于计算文本中的 Token 数量。OpenAI 还提供了一篇详细的 GitHub 文档，专门介绍如何计算文本的 Token 数量，以及如何选择最经济的 Token 计算方式以节省成本。对此感兴趣的人可以查阅该文档以获取更多信息。

值得一提的是，在中文中，一个 Token 大致相当于三到五个字符的长度。实际上，Token 就是文本经过分词处理后的样本数量，只是在不同的分词方式下字符的长度会略有不同。

3.1.4　Embedding

长期以来，在自然语言处理领域，我们必须将文本序列编码成数字形式，以便算法能够理解和处理。这意味着文本数据需要经过特定的编码方式转换为数字形式。虽然存在多种编码方法，但它们的本质都是将单词或字符用数字或数字序

列表示。在相同的编码规则下，相同的单词将始终被编码为相同的数字序列，是使用单个数字还是数字序列则取决于算法工程师的选择。

尽管编码的目的是将文本转换为数字形式，但其中涉及的复杂性远非表面看起来那么简单。首先，对于相同的词或字，它们必须在同一套规则下被编码为相同的数字序列或数字，这意味着必须存在一个完全一一对应的词典表来进行编码。然而，除这种直接的一一对应关系外，还有更高层次的境界，即语义编码。这意味着如果两个单词、两个字或两个词组在语义上相似，那么它们被编码成数字后，这些数字在数学上也应该相似。换言之，数字不仅是编码的唯一表示，而且必须代表词的语义信息。举例来说，在语义编码中，“女人”和“女孩儿”应该更接近。但需要注意的是，世界上并不存在唯一正确的语义空间，不同的编码方式会构建不同的语义空间。

为了实现这一目标，需要建立一个高维语义空间，在这个空间中，数学上相邻的点代表着语义上更接近的两个词。大部分与编码相关的研究都是围绕如何构建这样的语义空间展开的，因此编码的实现是具有一定难度的。

目前，深度学习中常见的编码方式包括以下两种。

（1）One-hot（独热）编码：将每个词或字符表示为一个稀疏向量，向量的长度等于词汇表的大小，其中对应于该词或字符的索引位置为 1，其余位置为 0。

（2）Word Embedding（词嵌入）编码：通过训练模型学习得到的词向量表示，将每个词映射到一个低维的实数向量空间中，保留了词之间的语义信息。

如图 3-1 所示，对句子分别进行 Word Embedding 编码和 One-hot 编码后产生的二维表单。

词	×1	×2	×3	×4	×5
小狗	0.5036	0.4943	0.4155	0.7221	0.9132
依偎	0.4716	0.9035	0.1671	0.4957	0.9929
在	0.8539	0.3560	0.7110	0.9761	0.2447
火炉	0.6010	0.7540	0.6639	0.4274	0.8514
旁	0.2665	0.4797	0.8628	0.2134	0.9215
因为	0.0872	0.8268	0.7835	0.3272	0.8871
小狗	0.8556	0.1318	0.2701	0.3320	0.7241
很	0.8781	0.5493	0.2159	0.5839	0.2512
冷	0.9900	0.7401	0.1414	0.4207	0.9985

Word Embedding 编码

词	×1	×2	×3	×4	×5	×6	×7	×8	×9
小狗	1	0	0	0	0	0	0	0	0
依偎	0	1	0	0	0	0	0	0	0
在	0	0	1	0	0	0	0	0	0
火炉	0	0	0	1	0	0	0	0	0
旁	0	0	0	0	1	0	0	0	0
因为	0	0	0	0	0	1	0	0	0
小狗	0	0	0	0	0	0	1	0	0
很	0	0	0	0	0	0	0	1	0
冷	0	0	0	0	0	0	0	0	1

One-hot 编码

图 3-1

3.1.5 语义向量空间

一个通过词嵌入训练得到的语义向量空间如图 3-2 所示。我们可能发现“国王”和“男人”、“女王”和“女人”在向量空间中是相似的。词嵌入通常通过大量的文本数据训练得到，目的是捕捉单词之间的语义关系。

经典的词嵌入方法包括以下 5 种。

（1）Word2Vec：由 Google 研究人员 Tomas Mikolov 等于 2013 年提出，通过训练神经网络模型来学习词向量，其中包括两种模型，即连续词袋（CBOW）和 Skip-gram。

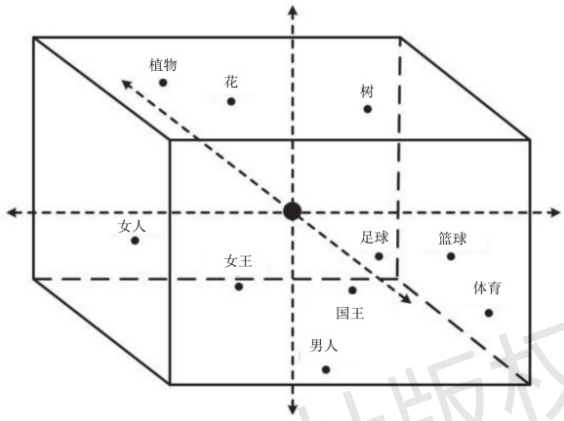

图 3-2

（2）GloVe（Global Vectors for Word Representation，全局向量的词嵌入）：由斯坦福大学的研究人员提出，它结合了全局语料库统计信息和局部上下文窗口中的词共现信息，通过矩阵分解的方式学习词向量。

（3）FastText：由 Facebook 提出，是基于 Word2Vec 的改进版本，通过考虑词的子词信息（字符 n-gram）来学习词向量，从而更好地处理稀有词和形态变化。

（4）固定编码：例如使用 BERT、GPT 等预训练的深度学习模型来编码文本。这些模型通常使用大量数据进行预训练，并可以为新任务进行微调。

（5）基于大模型进行编码：在 OpenAI 研发的大模型生态矩阵中，存在专用于构建语义空间的 Embedding 大模型。它是基于 GPT-3 训练出来的，是专门用于构建语义空间的大模型。

这些经典的词嵌入方法在自然语言处理领域得到了广泛的应用，并为文本表示和语义理解任务提供了重要的基础。

3.2　语言模型历史演进

3.2.1　语言模型历史演进

语言模型的发展可划分为以下三个关键阶段，如表 3-2 所示。

（1）统计语言模型：这一阶段的模型主要基于概率统计原理，通过计算词频、n-gram 等统计方法来预测文本中下一个词的概率，代表了语言模型的早期形态。

（2）神经网络语言模型：随着深度学习技术的兴起，神经网络开始被引入语言模型中，通过多层非线性变换捕捉词汇间的复杂关系，RNNLM（Recurrent Neural Network Language Model，循环神经网络语言模型）、LSTM 等模型在这一时期成为主流，显著提升了语言建模的准确性和流畅度。

（3）基于 Transformer 的大模型：此阶段以 Transformer 架构为核心，尤其是自注意力机制的引入，实现了并行处理和长距离依赖的有效捕获，使得模型规模得以指数级增长，如 BERT、GPT-3、Turing-NLG 等，这些模型不仅在语言理解与生成任务上取得了突破性进展，还展现了跨领域的应用潜力，开启了预训练语言模型的新纪元。

表 3-2　语言模型发展的三个关键阶段

阶段	定义	成果	限制
统计语言模型	初期的语言模型，依赖于统计分布和概率来预测单词。起源于信息论和早期计算机科学	（1）统计机器翻译。 （2）n-gram 模型和 HMM。 （3）最大熵模型	（1）难以捕捉长期依赖性。 （2）需要复杂的特征工程
神经网络语言模型	神经网络在捕捉复杂模式方面的优势使其成为语言模型的一个进步。包括 RNNLM 和 LSTM 在内的模型被用于捕捉序列数据中的依赖关系	（1）神经网络语言模型（RNNLM）。 （2）RNNLM 处理序列数据。 （3）LSTM 在语言模型中的应用	（1）计算效率和扩展性有限。 （2）梯度消失或爆炸问题
基于 Transformer 的大模型	Transformer 架构通过自注意力机制改进了长距离依赖问题的处理，并提高了并行化处理的效率	（1）Transformer 架构。 （2）BERT 模型。 （3）GPT 模型家族等	计算资源消耗大

如表 3-2 所示，第一阶段的统计语言模型可追溯至 20 世纪 50 年代，那时，运用统计学方法完成机器翻译的任务已初露端倪。此时期，统计学方法在该领域占据主导，其核心在于依托人类专家的知识体系；然而，这种方法的“理论上限”相对明确，本质上受限于人类认知能力的边界。相比之下，神经网络模型的兴起标志着一个转折点，它们依赖于海量数据与强大的计算能力，从而在诸多领域，包括视觉识别与自然语言处理领域，实现了对人类能力的超越。这种数据驱动的方法，不需要人工精心设计特征，也能在性能上取得显著突破。

统计语言模型的本质在于依据词语的分布特性和出现频率来进行预测，尽管这一策略在实践中展现出一定成效，但其进步长期受制于对人为特征工程的高度依赖，限制了模型的自主学习与发展潜力。

第二阶段的神经网络语言模型的核心思想是，将神经网络技术应用于语言模型构建中。这一阶段的显著特点是，机器学习，尤其是深度学习技术，充分展现了数据驱动的优势，通过强大的计算能力学习数据分布，从而超越了以往依赖人

工设计特征的局限，突破了模型性能的天花板。神经网络语言模型有能力捕获文本中的复杂模式和隐含意，这与以往技术（如 One-hot 编码）形成对比，后者难以表达深层次的语义信息。

尽管神经网络语言模型带来了显著进步，但仍存在一定的局限性。在视觉任务中广泛应用的循环神经网络（RNN），其天生存在的问题之一在于顺序处理机制，即必须等待上一时间步的神经元输出后，才能处理下一时间步，这导致计算效率低下。另外，RNN 面临梯度消失/爆炸问题，限制了其捕捉长距离依赖关系的能力，例如，早期的基于 RNN 的语言模型往往仅能考虑相邻的几个词，这对于语言理解的深度和连贯性构成了一定阻碍。

第三阶段的基于 Transformer 的大模型目前是该领域的焦点所在。Transformer 的核心创新在于引入了自注意力机制，这一革命性改进有效地解决了长期依赖问题。用传统方法处理句子时，假设每个词语同等重要，导致模型必须均匀记忆所有信息；而自注意力机制使模型能够识别出关键信息——即便在扩展到 50 个或 100 个词语的更长序列中，模型也能精准聚焦于少数几个关键词语，极大地提升了信息处理的效率与针对性，体现了注意力机制的核心价值。此外，Transformer 架构促进了并行计算，进一步加速了处理速度。

然而，基于 Transformer 的模型也面临着显著挑战：推理成本高昂，主要是计算需求与输入序列长度呈现平方关系增长，导致生成长文本时资源消耗急剧上升。另外，这类模型通常直接从大量无标注数据中学习，缺乏明确的监督训练阶段，这意味着如果训练数据蕴含偏见，则模型不仅会吸收这些偏见，而且还可能在预测过程中将其放大，这是在当前研究与应用中亟须关注和解决的问题。

3.2.2　统计语言模型

自然语言处理（NLP）模型使用统计语言模型作为一种建模技术，旨在对自然语言数据的概率分布进行建模和预测。统计语言模型基于一系列统计学方法和概率模型，用于衡量给定一串单词序列的可能性或概率。其基本思想是根据从文本语料库中观察到的频率信息，推断出单词之间的概率分布和语言结构。

在 NLP 模型中，统计语言模型通常采用 n-gram 模型或者隐马尔可夫模型等。

n-gram 模型是一种基于上下文窗口的语言模型，它假设一个单词出现的概率仅与其前面 n−1 个单词相关，通过对大量文本数据进行统计，n-gram 模型可以计算出每个单词在给定上下文中出现的条件概率，从而实现对文本序列的建模。

隐马尔可夫模型即 HMM 是一种用于建模序列数据的统计模型。一个词出现的概率只和它前面出现的一个或有限几个词相关。在 NLP 中，HMM 通常用于词性标注和语音识别等任务，HMM 通过定义状态和状态之间的转移概率及观测状态的发射概率对序列数据进行建模。

统计语言模型在 NLP 中扮演着重要角色，它们可以用于完成语言生成、语言

识别、文本分类和机器翻译等任务。通过对大量文本数据进行分析，统计语言模型能够捕捉到自然语言的规律和结构，为 NLP 提供有效的语言建模基础。然而，统计语言模型在处理复杂的语言结构和语义信息时存在一定的局限性，因此随着神经网络技术的发展，NLP 领域逐渐转向了基于神经网络的语言模型。

3.2.3 神经网络语言模型

NLP 模型从传统的统计语言模型过渡到神经网络语言模型标志着一种技术演进和转变。传统的统计语言模型主要基于规则的方法和统计概率模型，如 n-gram 模型和隐马尔可夫模型。这些模型依赖于人工设计的特征和规则，并且在处理复杂的语言结构和语义信息方面存在一定的局限性。

随着神经网络技术的发展和深度学习算法的出现，NLP 领域开始出现了基于神经网络的语言模型。神经网络语言模型通过大规模语料库的训练，利用深度学习模型自动学习语言的表示和结构特征，从而实现了更加准确和灵活的语言建模能力。

神经网络语言模型采用各种结构，如循环神经网络（RNN）、长短期记忆网络（LSTM）、门控循环单元（Gated Recurrent Unit，GRU）和注意力机制等，以捕捉输入序列的长期依赖关系和语义信息。这些模型能够自动学习语言的表示，不需要人工设计的特征，从而可以更好地适应不同类型和领域的语言数据。

与传统的统计语言模型相比，神经网络语言模型具有以下优势。

（1）更好的语言建模能力：神经网络语言模型能够更好地捕捉语言的复杂结构和语义信息，从而实现更准确和灵活的语言建模。

（2）端到端学习：神经网络语言模型可以通过端到端的方式进行学习，不需要手工设计特征和规则，简化了模型构建的过程。

（3）适应性强：神经网络语言模型可以通过大规模数据的训练，自动学习适应不同领域和类型的语言数据，具有更好的泛化能力。

（4）神经网络语言模型的出现和发展为 NLP 领域带来了革命性变革，推动了 NLP 技术的发展和应用。

3.3 注意力机制

3.3.1 RNN 模型

1. 基本概念

RNN 模型作为一种先进的神经网络模型，专为处理序列数据而设计，在 NLP

领域展现出了广泛应用的潜力。RNN 的独特之处在于其内置的记忆功能，它能接纳可变长度的输入序列，并能依据序列内部的上下文逻辑进行学习与预测，这一特性使之在解析语言序列数据上表现出卓越的适应性。

在 NLP 的众多应用场景中，RNN 发挥着核心作用，涵盖了语言建模、情感分析、命名实体识别、文本生成及机器翻译等多种任务。其优势在于深刻捕捉文本的时间动态性和长期依赖特征，强化了对语言结构的理解和处理能力。例如，在语言建模场景下，RNN 能够基于历史词汇的上下文环境，预测序列中下一个词汇的出现概率；在情感分析任务中，则通过分析文本段落或句子的内在结构，输出对文本情感倾向的判断；在机器翻译任务中，RNN 能够实现源语言到目标语言的高效转换。

特别是，LSTM 作为 RNN 的一种关键演化形式，通过集成门控机制显著增强了模型捕获长距离依赖的能力，成功缓解了标准 RNN 在处理长序列数据时遭遇的梯度消失和梯度爆炸难题，进一步推动了 LSTM 在 NLP 任务中的广泛应用，并显著提高了模型的性能水平。

综上所述，RNN 及其变体如 LSTM 在 NLP 领域的应用，不仅为我们提供了一个强有力的工具来深入解析和理解语言序列数据，也为多样化的文本分析和处理任务带来了高效且有效的解决方案，标志着向高级人工智能交互迈出的重要一步。

2. RNN 架构

图 3-3 展现了经典 RNN 的基本架构。

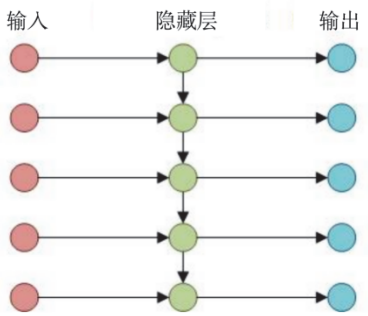

图 3-3

RNN 的输入是 input:$\{x_1, x_2, \cdots, x_t, \cdots, x_T\}$，输出是 input:$\{y_1, y_2, \cdots, y_t, \cdots, y_T\}$。

（1）输入层（Input Layer）：在初始阶段，系统接收的数据输入网络的输入层级。尤其是在自然语言处理的场景下，这些输入数据被转化为单词、字符或其他形式的语言单元编码以便于后续处理。

（2）隐藏层（Hidden Layer）：构成 RNN 核心部分的是其隐含层。在此结构

中，每个时间步不仅包含新接收到的输入信息，还整合了前一时间步的隐含状态信息。通过这一机制，隐含层负责推算并更新当前时间步的隐含状态，该状态实质上保留了过往信息的痕迹，赋予网络记忆功能，使其能有效应对时间序列数据的复杂性。

（3）输出层（Output Layer）：隐含层经过处理产生的信息会进一步传输到输出层，旨在生成模型的终端输出。此输出可体现为对未来值的预测、类别归属的判定，或作为递归至下一时间步的内部状态，继续参与序列的解析与预测过程。

3．缺陷

尽管 RNN 在应对时间序列数据分析方面展现出了一定的优势，但也存在若干局限性。其中两项主要的局限性如下。

（1）限定的等长要求：标准 RNN 架构受限于一个根本假设，即输入序列与预期输出序列需保持严格的时间步一致性。此设计缺陷意味着模型在未经过调整的情况下难以适应变长序列数据，往往强制要求通过补零或剪裁操作来统一序列长度，此举不仅在实际部署中可能削弱模型的灵活性，还可能导致信息的不必要损失或冗余累积，特别是在处理天然长度多样的数据集时。

（2）长序列依赖处理的挑战：RNN 在面对变长序列数据时，经常遭遇由梯度消失引发的长期记忆障碍。随着序列时间跨度的增长，反向传播过程中梯度倾向于急剧衰减，严重阻碍了模型捕获和利用远距离时间步间的关系。这种机制上的不足，直接限制了模型捕获序列中远期关键特征的能力，进而影响其决策质量和整体效能。

上述限制显著制约了基本 RNN 模型在序列数据分析领域的实用性和有效性。有鉴于此，研究界已探索并开发出一系列优化模型，如 LSTM 和 GRU。这些进阶架构通过创新性的门控机制与记忆单元设计，成功解决了梯度消失、序列长度差异的问题，从而大幅度增强了模型捕捉复杂序列模式、提升预测精度及泛化能力的潜力。

3.3.2 Seq2Seq 模型

1．基础概念

Seq2Seq（Sequence-to-Sequence，序列到序列）模型作为一种革命性变体，源自 RNN 体系，其设计初衷在于突破传统 RNN 框架下对输入与输出序列等时长的严格约束。该模型的核心创新在于构建了双阶段（编码阶段与解码阶段）的处理流程，这一架构革新为此类模型赋予了接纳并处理不同长度序列数据的能力。

具体而言，Seq2Seq 模型的前半部分即编码器，完成将任意长度的输入序列转化成一个高维、固定长度的语境向量的任务。此向量被精心设计以封装源序列

的全部语义精髓。随后，该语境向量作为桥梁，传递给模型的后半部分即解码器，解码器逐步推演出目标序列，直至信号指示终止或达到预设的最大输出长度。这一精妙设计不仅确保了模型对于序列长度变化的高度适应性，还极大地拓宽了其在自然语言处理领域的应用场景，涵盖了从机器翻译、文本摘要到对话系统生成等诸多复杂任务。

在实现上，编码器与解码器均可采纳多种神经网络架构，包括但不限于标准 RNN、LSTM 及 GRU，这样的灵活性进一步强化了模型处理序列数据的效能与精确度。总而言之，Seq2Seq 模型凭借其对变长序列数据处理的高效解决方案，在自然语言处理领域确立了其作为关键模型的地位，展现了广泛的应用价值与深远的影响。

2. Seq2Seq 架构

如图 3-4 所示，在 Seq2Seq 架构中，编码器（Encoder）把所有的输入序列都编码成一个统一的语义向量 Context，然后由解码器（Decoder）解码。在解码器解码的过程中，不断地将上一时刻 t–1 的输出作为下一时刻 t 的输入，循环解码，直到输出停止符为止。

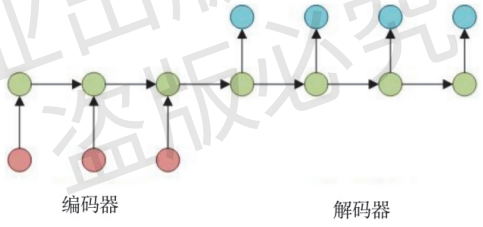

图 3-4

Seq2Seq 模型的训练通常使用教师强制（Teacher Forcing）方法，即将解码器的上一时刻的输出作为当前时间步的输入，直到生成完整的目标序列。训练完成后，该模型可以用于生成未见过的输入序列的对应目标序列，如将一种语言的句子翻译成另一种语言的句子。

3. Seq2Seq 架构的缺点

尽管 Seq2Seq 模型成功破除了输入与输出序列长度不匹配的障碍，但仍需面对若干挑战，主要包括效率问题、Context 限制问题及链式反应效应（蝴蝶效应）问题。

（1）效率问题：在处理延展的序列数据时，Seq2Seq 模型遭遇了效率瓶颈。该模型机制要求每个时间步均需综合考量整个输入序列的信息，致使其运算需求及内存占用随序列增长而急剧上升，从而显著延长了训练与推断周期，并加剧了资源消耗。

（2）Context 限制问题：由于 Context 包含原始序列中的所有信息，所以它的长度就成了限制模型性能的瓶颈。例如机器翻译问题，当要翻译的句子较长时，一个 Context 可能存不下那么多的信息，就会造成精度下降。

（3）链式反应效应（蝴蝶效应）问题：在 Seq2Seq 架构中，让上一时刻的输出作为下一时刻的输入进入网络，这种设计可能导致一个时刻输出的错误会影响后续所有时刻的输出。这是因为在模型的训练过程中，网络尚未完全收敛，因此即使在早期阶段产生的错误也会在后续时间步中逐渐累积和放大，从而导致整个输出序列的错误。轻微的预测误差可在时间序列中不断累积与放大，如同蝴蝶振翅引发的远端风暴，严重影响后续预测的准确性。尤其是在长序列任务中，这种累积错误效应更为显著，增加了模型学习正确模式的难度。

对于上述问题，学术界已探索多种策略以缓解这些问题，例如集成注意力机制以动态聚焦重要信息，利用残差连接保持信息流的连贯性，以及采用更为复杂的网络架构以增强对长期依赖关系的学习能力。这些策略共同作用旨在优化 Seq2Seq 模型处理序列任务时的性能稳定性和长期依赖管理，从而推动该技术边界的拓展。

3.3.3 Attention 注意力机制

如图 3-5 所示，针对 Seq2Seq 模型的优化策略，一个关键切入点在于通过充分利用编码器的所有隐藏层状态，来解决 Context 长度的局限性问题，并有效弱化链式反应效应（蝴蝶效应）。

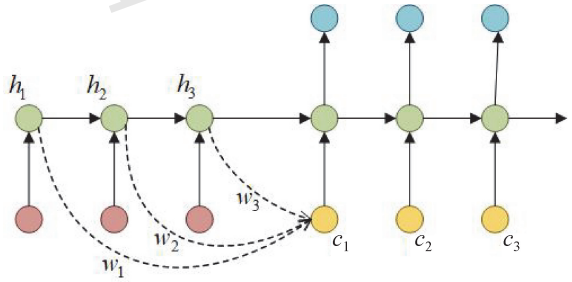

图 3-5

注意力机制作为一种在 Seq2Seq 模型中广泛应用的技术手段，旨在应对序列任务中面临的长距离依赖挑战及减缓信息传播中的细微偏差累积效应（蝴蝶效应）。该机制通过自适应地为输入序列的不同部分分配差异化的注意力权重，确保模型在生成每个阶段的输出时，能够聚焦于与该输出直接相关的关键输入信息，由此增强了模型的有效性和健壮性，减少了误差传播及信息衰减的问题。

图 3-6 直观展示了模型在未采用与采用注意力机制情况下的结构对比。

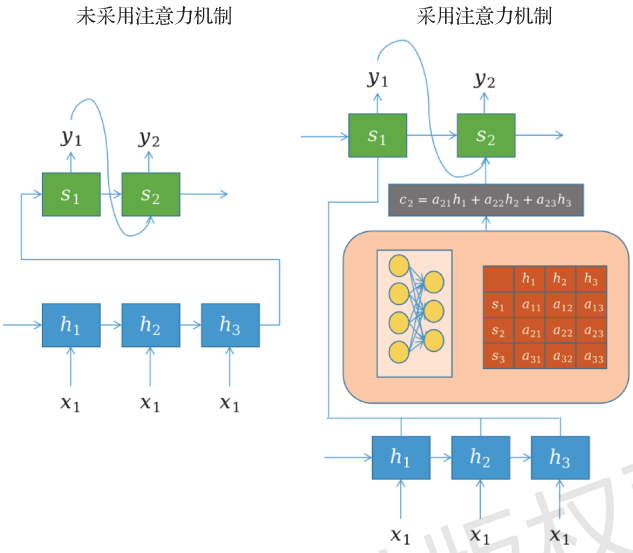

图 3-6

在图 3-6 中，x 表示输入序列，y 表示输出序列，h 表示输入编码器的隐藏层神经元，s 表示解码器中的隐藏层神经元。这些构成了原始 Seq2Seq 模型架构中待优化的核心参数。注意力机制作为一项补充，引入了一个新颖的中间层——注意力模型网络，其核心功能在于构建输入与输出隐藏状态之间的桥梁。

该注意力模型网络执行的关键任务是，通过量化评估每个输入位置对于生成当前输出位置的重要性，依据这一系列重要性评估值，对输入序列的不同组成部分实施加权平均运算，进而提炼出一个能综合反映输入序列与当前输出紧密度的加权上下文向量。此举有效增强了模型在处理序列信息时的关注焦点与上下文感知能力。

$$\boldsymbol{c}_j = \sum_{i=1}^{T} a_{ij} h_i$$

式中，$\boldsymbol{c}$ 表示上下文信息向量，a 表示注意力权重参数，T 表示输入序列的长度。

在训练过程中，注意力机制会自动学习到不同输入位置之间的相关性，从而能够在生成输出时动态地调整注意力权重。这样，即使出现了错误的预测，模型也能够及时纠正，并且不会像传统的 Seq2Seq 模型那样将错误传播到整个输出序列。

此时，解码器的隐藏层神经元 s 的参数如下：

$$s_j = f(s_{j-1}, y_{j-1}, \boldsymbol{c}_j)$$

神经元 s_j 与以下参数有关：上一个解码器隐藏层神经元 s_{j-1}、上一个解码器隐藏层的输出 y_{j-1}、上下文信息向量 $\boldsymbol{c}_j$。

总的来说，注意力机制通过引入动态的上下文信息，使模型在生成输出时能够更加准确地关注输入序列中与当前输出位置相关的信息，从而有效地缓解了长

距离依赖和蝴蝶效应问题。

因此，注意力机制的本质并非繁复难解，其旨在实现输出解码层级与输入编码层级的一种直接且动态的关联。此机制的核心之处在于引入一组经学习优化的权重，即注意力权重（Attention Weight），记作 a，用以量化表征 h（编码器隐藏层状态）与 s（解码器隐藏层状态）两者间的相关性。每项权重 a 均可视为双方交互作用强度的度量，整体架构则构成了注意力模型（Attention Model），此模型另有一个等价表述——对齐函数（Alignment Function），强调了它在跨序列元素匹配上的功能。

关于对齐函数的具体实现方式，论文 *An Attentive Survey of Attention Models* 中提及了几种典型范式，其直观展示参见图 3-7。

Function	Equation	References
similarity	$a(k_i, q) = sim(k_i, q)$	[Graves et al. 2014a]
dot product	$a(k_i, q) = q^T k_i$	[Luong et al. 2015a]
scaled dot product	$a(k_i, q) = \frac{q^T k_i}{\sqrt{d_k}}$	[Vaswani et al. 2017]
general	$a(k_i, q) = q^T W k_i$	[Luong et al. 2015a]
biased general	$a(k_i, q) = k_i(Wq + b)$	[Sordoni et al. 2016]
activated general	$a(k_i, q) = act(q^T W k_i + b)$	[Ma et al. 2017b]
generalized kernel	$a(k_i, q) = \phi(q)^T \phi(k_i)$	[Choromanski et al. 2021]
concat	$a(k_i, q) = w_{imp}^T act(W[q; k_i] + b)$	[Luong et al. 2015a]
additive	$a(k_i, q) = w_{imp}^T act(W_1 q + W_2 k_i + b)$	[Bahdanau et al. 2015]
deep	$a(k_i, q) = w_{imp}^T E^{(L-1)} + b^L$ $E^{(l)} = act(W_l E^{(l-1)} + b^l$ $E^{(1)} = act(W_1 k_i + W_0 q) + b^l$	[Pavlopoulos et al. 2017]
location-based	$a(k_i, q) = a(q)$	[Luong et al. 2015a]
feature-based	$a(k_i, q) = w_{imp}^T act(W_1 \phi_1(K) + W_2 \phi_2(K) + b)$	[Li et al. 2019a]

图 3-7

最终，注意力机制可以抽象为以下公式。

$$A(\boldsymbol{q}, \boldsymbol{K}, \boldsymbol{V}) = \sum_i p(a(k_i, \boldsymbol{q}))v_i$$

在注意力机制中，$\boldsymbol{q}$、$\boldsymbol{K}$、$\boldsymbol{V}$ 是三个核心概念，它们代表了注意力计算过程中的关键元素，其具体含义如下。

（1）$\boldsymbol{q}$（Query）：$\boldsymbol{q}$ 代表当前的关注点或查询向量。在解码器的上下文中，$\boldsymbol{q}$ 通常来源于当前解码步骤的隐藏状态，反映了模型当前关注或查询信息的需求。它的功能是寻找与之最相关的部分，即在输入序列中哪里的信息对生成当前输出最为重要。

（2）$\boldsymbol{K}$（Key）：$\boldsymbol{K}$ 可以理解为每个输入位置的代表性特征或索引量。在编码器的输出中，每个位置的 $\boldsymbol{K}$ 对应该位置的输入信息的某种编码表示，用于与 $\boldsymbol{q}$ 进行匹配。$\boldsymbol{K}$ 有助于确定哪些部分的输入与当前的 $\boldsymbol{q}$ 最为相关，就像数据库查询中的关键词一样。

（3）$\boldsymbol{V}$（Value）：$\boldsymbol{V}$ 则是与 $\boldsymbol{K}$ 对应的实际内容信息或上下文信息向量，代表了输入序列中每个位置的详细信息。一旦通过 $\boldsymbol{q}$ 和 $\boldsymbol{K}$ 的交互确定了哪些部分是重要的（计算出注意力权重），这些权重就会被用来加权 $\boldsymbol{V}$，从而生成一个加权后的上下文信息向量，这个向量包含了根据当前需求筛选出的、对下一步操作最有价值的信息。

注意力机制的工作流程大致可以描述为：首先，通过计算 $\boldsymbol{q}$ 与各个 $\boldsymbol{K}$ 之间的相似度（通常采用点积操作），得到未归一化的注意力分数；然后，应用像 softmax 这样的归一化函数得到注意力权重；最后，将这些权重应用于对应的 $\boldsymbol{V}$ 上进行加权求和，得到最终的上下文信息向量。这一过程使得模型能够动态、有选择性地集中处理输入信息中最相关的部分，从而提高处理序列数据的效率。

3.4　Transformer 架构

3.4.1　整体架构

图 3-8 展示的是神经网络中的 Transformer 架构。在著名的论文 *Attention Is All You Need* 中，编码器和解码器的层数 h 明确设定为 6 层。这一复合架构统称为 Transformer 架构。

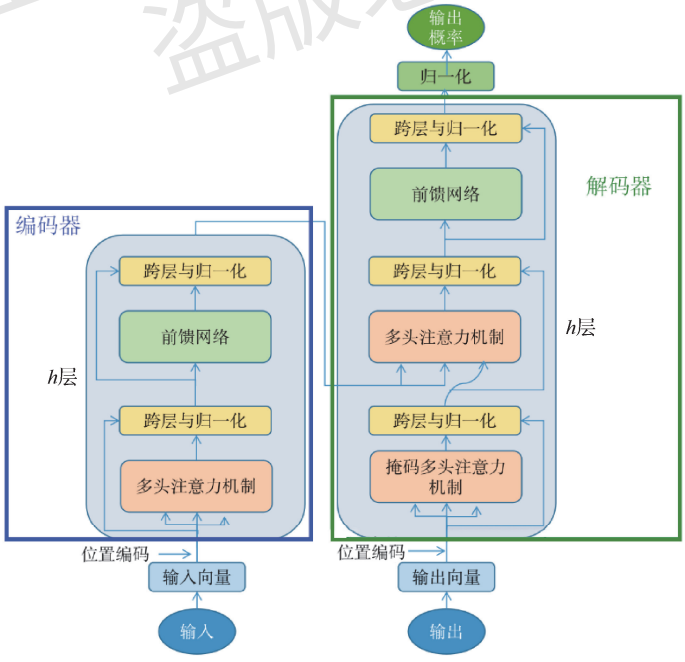

图 3-8

神经网络中的 Transformer 架构是一种先进的序列数据处理模型，最初在自然语言处理领域取得了突破性进展，后来被广泛应用于图像处理等其他领域。

Transformer 架构通过这些精心设计的模块协同工作，实现了对序列数据的强大处理能力，特别是在保留序列中长距离依赖信息的同时，提高了模型的效率和性能。下面介绍它是如何从序列对齐的 RNN 架构变成了 Self-Attention 的 Transformer 架构的。

3.4.2 Self-Attention

Self-Attention 是一种核心机制，其核心目标是解决在序列数据内部不同元素之间建立有效关联的问题，以捕捉并理解这些元素间的语义联系。具体来说，它试图揭示在一个句子“The Law will never be perfect, but its application should be just”中，“its”这个词究竟与哪个词语有着最密切的关联，而不单是简单地用于机器翻译，更多的是服务于深入理解文本的内在意义。通过这一机制，模型在学习了丰富的上下文信息后，能够逐渐掌握“its”是指向“Law”还是“application”这样的语义区分。

在多层的自注意力架构中，每层都在学习数据的不同层面的关联和特征，尽管它们关注的细节可能各异，但所有层的共同追求是提取和整合这些复杂的内在关联性，最终帮助模型整体上更好地理解语言的深层结构和语义。简而言之，自注意力机制的使命在于通过动态地为输入序列中的每个部分分配权重，强调与当前关注点最为相关的部分，从而提炼出关键的语义关系，增强模型的语境理解能力。

1. 公式

Self-Attention 最终的形式表达式如下：

$$\text{Attention}(\boldsymbol{Q},\boldsymbol{K},\boldsymbol{V})=\text{softmax}\left(\frac{\boldsymbol{Q}\boldsymbol{K}^{\mathrm{T}}}{\sqrt{d_k}}\right)\boldsymbol{V}$$

在这个公式中，

（1）$\boldsymbol{Q}$ 表示查询向量，通常是解码器的输出，用于询问编码器输出中的相关信息。

（2）$\boldsymbol{K}$ 表示键向量，与编码器的输出相关，用于与查询向量进行匹配。

（3）$\boldsymbol{V}$ 表示值向量，同样来源于编码器的输出，包含实际需要被加权求和的信息。

（4）d_k 是向量 $\boldsymbol{K}$ 的维度，通常在计算注意力得分前除以它的平方根进行缩放，以稳定 softmax 函数的计算并避免出现大的数值导致的梯度量级联乘积溢出问题。

（5）$\frac{\boldsymbol{QK}^{\mathrm{T}}}{\sqrt{d_k}}$ 计算得出了查询向量与所有键向量之间的相似度量分数。

（6）softmax 函数将这些相似度量转换为概率分布，确保注意力权重之和为 1，使得不同部分的信息可以被加权求和时有所侧重。

在图 3-8 中提到的多头注意力（Multi-Head Attention），会将上述过程多次并行进行，每个头可能关注输入的不同部分，然后将结果合并，以捕获更丰富的上下文信息。

2．缩放点积注意力

softmax 括号里的整体是它使用的对齐函数，这个函数也在图 3-7 中提到过，就是缩放点积注意力（Scaled Dot-Product Attention）。图 3-9 是它的具体模块。

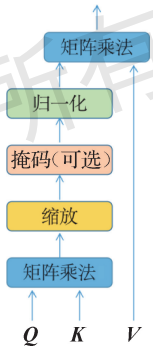

图 3-9

缩放点积注意力的核心流程如下。

（1）计算 $\boldsymbol{Q}$、$\boldsymbol{K}$ 矩阵的点积，并进行逐元素除以 $\sqrt{d_k}$ 的操作（d_k 是 $\boldsymbol{K}$ 的维度），这一操作称为缩放，目的是在向量维度较大时稳定学习过程。

（2）可应用一个 Mask 操作来限制某些位置的注意力（例如，在处理时间序列数据时避免“未来泄露”）。

（3）对缩放后的结果应用 softmax 函数，以此获得每个 $\boldsymbol{Q}$ 位置上关于所有 $\boldsymbol{K}$ 位置的注意力权重分布。

（4）将得到的注意力权重与 $\boldsymbol{V}$ 矩阵进行矩阵乘法，加权求和得到输出，即综合考虑最重要输入信息的上下文向量。这一系列步骤共同构成了 Attention 机制中的 $\boldsymbol{Q}$、$\boldsymbol{K}$ 和 $\boldsymbol{V}$ 变换过程。

3．Attention 与 Self-Attention

Attention（注意力机制）与 Self-Attention（自注意力机制）的差异如下。

（1）Attention：在传统的注意力机制中，模型根据 $\boldsymbol{Q}$ 与 $\boldsymbol{K}$ 之间的相似度来计算注意力权重，然后将这些权重应用于 $\boldsymbol{V}$ 以获取加权的表示。在这种情况下，查询和键通常是来自不同的源，例如编码器—解码器模型中的编码器隐藏状态作为键，解码器隐藏状态作为查询，用于生成上下文表示。

（2）Self-Attention：这是一种特殊形式的注意力机制，其中 $\boldsymbol{Q}$、$\boldsymbol{K}$ 和 $\boldsymbol{V}$ 都来自同一个源序列。这意味着模型可以在序列内部不同位置之间相互作用，并根据序列内部的语义信息来动态地调整权重。自注意力机制允许模型在同一序列中的不同位置之间建立依赖关系，从而更好地捕捉序列内部的长距离依赖关系。

总之，Self-Attention 可以看成 Attention 的一种特殊形式，其独特之处在于它允许模型在同一序列内部进行交互，从而更好地捕捉序列内部的长距离依赖关系。

3.4.3 Multi-Head Attention

选择缩放点积注意力作为对齐函数之后，Transformer 架构如图 3-10 所示。

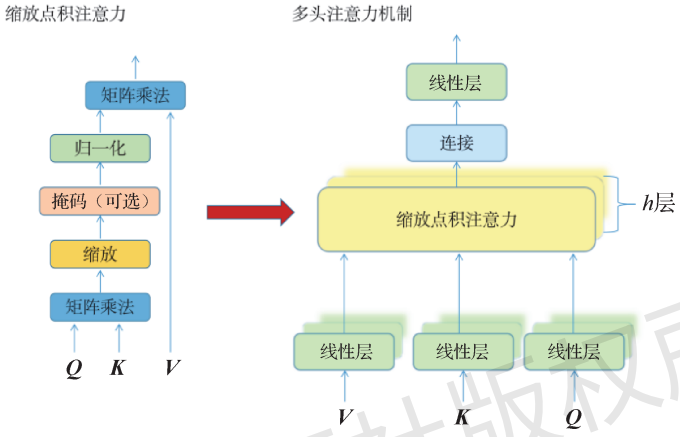

图 3-10

多头注意力机制（Multi-Head Attention）是 Transformer 架构中的一个核心概念，被频繁提及且至关重要。这里的“多头”指的是图 3-10 右侧的“h”，代表着并行存在的 h 个独立的缩放点积注意力模块或者对齐机制。每个头作为一个单独的注意力层，从不同的代表性子空间中捕获信息。这些头并非孤立工作，而是各自运算后，其结果通过连接（Concatenation）操作汇聚起来，随后经过一个权重矩阵 $\boldsymbol{W}^O$ 的变换，旨在整合来自各个注意力头的独特洞察。

这一过程不仅增强了模型捕获多样化特征的能力，还使得模型能够同时关注输入序列的不同方面或特征维度，每个头可能侧重于理解输入数据的不同特征组合或模式。因此，多头注意力通过这种并行且多样化的视角，提升了模型理解和综合信息的灵活性与深度，每个头相当于从一个略微偏移的角度审视输入，共同描绘出一个更为全面和细致的关注图谱。简言之，$\boldsymbol{W}^O$ 不仅用于形式上的整合，更是体现了如何平衡和利用这些多样注意力头产生的不同维度的重要性，共同为后续的计算提供更为丰富和精细的上下文信息。核心公式如下：

$$\text{MultiHead}(\boldsymbol{Q},\boldsymbol{K},\boldsymbol{V}) = \text{Concat}(\text{head}_1,\cdots,\text{head}_h)\boldsymbol{W}^O$$

$$\text{where head}_i = \text{Attention}(\boldsymbol{Q}\boldsymbol{W}_i^{\boldsymbol{Q}},\boldsymbol{K}\boldsymbol{W}_i^{\boldsymbol{K}},\boldsymbol{V}\boldsymbol{W}_i^{\boldsymbol{V}})$$

3.4.4 Encoder

Transformer 的编码器（Encoder）部分是该模型架构中的关键组件，负责接收输入序列并对其进行多层编码处理，以生成对序列内容的丰富表示。具体而言，Transformer 编码器由多个相同的层堆叠而成，每个层又细分为两个主要子层：多

头自注意力（Multi-Head Self-Attention）机制和前馈神经网络（Feed Forward Neural Network，FFNN）之间通常会插入一层归一化操作以稳定训练过程。

如图 3-11 所示，多头注意力机制是指红色箭头的右侧部分。具体而言，这一组件由 Multi-Head Attention 层紧接归一化处理，并随后串联一个全连接的前馈网络（FFN），共同构成了 Transformer 编码器的核心构造模块。

Transformer 编码器的详细描述如下。

（1）输入 Embedding 与位置编码：在进入多层编码结构之前，输入序列首先被转换为词嵌入（Word Embedding），以捕获词汇的语义信息。此外，为了使模型能够理解序列中元素的位置信息，还会添加位置编码（Positional Encoding），这是一种固定或可学习的向量，以确保模型知道输入序列中每个 Token 的相对位置。

（2）多头自注意力：每个编码器层的核心是多头自注意力机制，它允许模型在输入序列的不同部分之间灵活分配注意力，从而捕捉序列的长距离依赖关系。通过将输入分成多个并行的“头”，每个头执行独立的注意力计算，然后将结果合并，模型可以从多个角度并行地审视输入序列，增强其对复杂结构的理解能力。

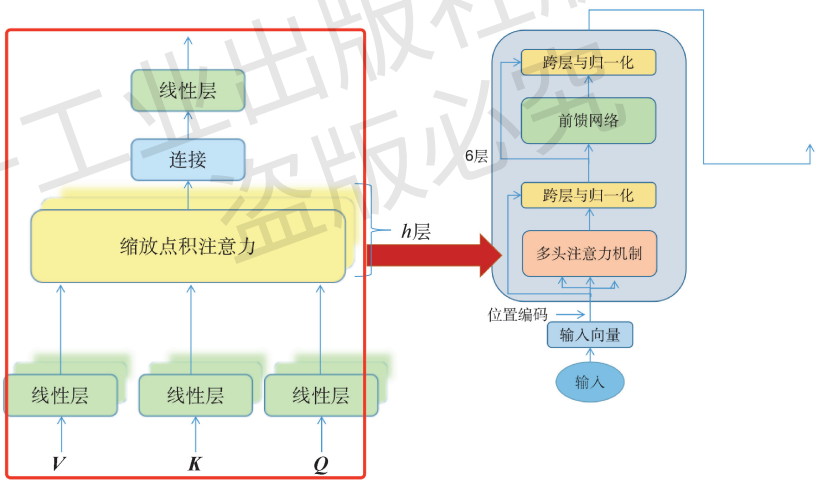

图 3-11

（3）跨层与归一化：为了缓解深度网络训练中的梯度消失问题，每个子层（多头自注意力和前馈神经网络）之后都会跟随一个残差连接（输入与输出相加），保留原始输入信息。随后，应用层归一化（Layer Normalization 或其他归一化技术），确保数据分布稳定，加快训练速度并提高模型性能。

（4）前馈神经网络：每个编码器层的第二个子层是一个全连接的前馈网络，通常包含两个线性变换层，中间夹着一个非线性激活函数（如 ReLU），为模型引入非线性表达能力，能够学习更复杂的特征表示。全连接的前馈网络 FFN 的公式如下：

$$\mathrm{FFN}(x)=\max(0,xW_1+b_1)W_2+b_2$$

（5）重复堆叠：上述结构在整个编码器中重复多次（通常为 6 次），每次迭代都会对序列表示进行更深一层的加工，逐步提炼出更高层次的特征表示。每层的输出作为下一层的输入，通过这一系列的逐步提炼，模型能够构建出对输入序列极其丰富的表示。

Transformer 编码器通过一系列精心设计的层，高效地捕捉和编码输入序列的语义与结构信息，为后续的解码过程或其他下游任务提供强有力的基础表示。

3.4.5 Decoder

1．组成模块

Transformer 的解码器（Decoder）负责生成序列的输出，其结构设计独特，旨在保证预测的顺序性，并且能够有效地利用编码器产生的上下文信息。解码器同样由多个相同的层堆叠而成，每层包含三个关键模块，但与编码器相比，在多头自注意力机制的使用上有所差异。以下内容是解码器结构的详细说明。

（1）Masked Multi-Head Attention（掩码多头注意力）：解码器的第一部分是一个特殊的多头自注意力层，它引入了“未来遮蔽”（Future Masking）机制。这意味着在计算当前单词（或 Token）的注意力权重时，会阻止它看到未来的词，确保预测遵循自然语言的时序性。这一设计强制模型仅基于过去的词来预测下一个词，避免了信息泄露，保证了序列生成的正确顺序。

（2）Encoder-Decoder Attention（编码器-解码器注意力）：解码器的第二部分是另一个多头自注意力层，但这里有所不同。在这个层里，查询（$\boldsymbol{Q}$）矩阵来自上一个解码器块的输出，而键（$\boldsymbol{K}$）和值（$\boldsymbol{V}$）矩阵则直接取自编码器的最终输出矩阵 $\boldsymbol{C}$。这种设置使得解码器能够依据当前解码状态，有选择地从编码器捕获到的全局上下文中提取相关信息，为生成下一步的输出做准备。

（3）Feed Forward Network（前馈网络）：解码器的最后一个模块与编码器相同，是一个全连接的前馈网络。这个网络包含两层线性变换，中间夹着 ReLU 等非线性激活函数，用于进一步转换和丰富每个位置的表示，增加模型的表达能力。

和编码器一样，解码器的每层之间也通过跨层方法，如残差连接（Residual Connections）和层归一化（Layer Normalization）相连，以促进梯度流动并保持输出的稳定性。

总之，解码器通过这三个精心设计的模块协同工作，实现了有序地生成序列，同时有效融合了自身的先前输出，以及编码器提供的输入序列的上下文信息，展现了 Transformer 模型在序列生成任务中的强大效能。

如图 3-12 所示，编码器与解码器的最大不同之处是，解码器使用了红色箭头所指的掩码多头注意力机制。

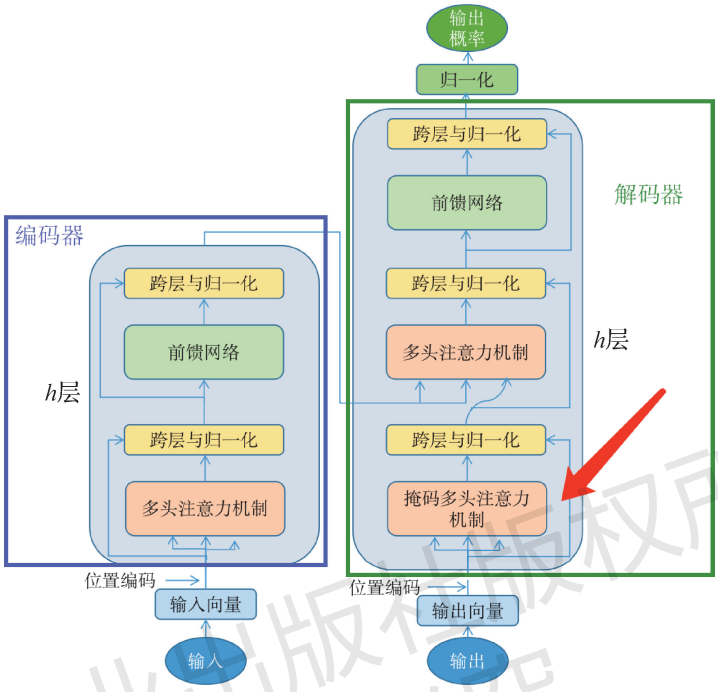

图 3-12

2. Masked Multi-Head Attention

Masked Multi-Head Attention 机制的特殊之处在于它引入了掩码（Mask）机制，以确保模型在处理序列数据时不会泄露未来信息。具体来说，掩码通常用于遮盖序列中当前位置之后的所有信息，使得模型只能基于当前位置之前的信息来进行预测或生成。

在注意力机制中，每个位置都可以与序列中的所有其他位置进行交互，因此掩码被用来限制这种交互只能发生在当前位置之前的位置上。这样做可以确保模型在预测时不会使用未来信息，从而使得模型更加合理和可靠。

总之，Masked Multi-Head Attention 是一种在 Transformer 架构中用于处理序列数据的注意力机制，通过引入掩码机制来确保模型在预测时不会泄露未来信息，从而提高了模型的效率和准确性。

3.4.6　实验效果

最终的模型是 6 层网络结构，将其展开如图 3-13 所示。

把 Transformer 架构庖丁解牛后，从整体上看，Transformer 架构也就这几类。将多个自注意力机制堆成多头注意力机制，加上归一化就构成了编码器（Encoder）。经过掩码操作后的 Masked Multi-Head Attention 加上 Encoder 同款结

构，就构成了解码器（Decoder）。

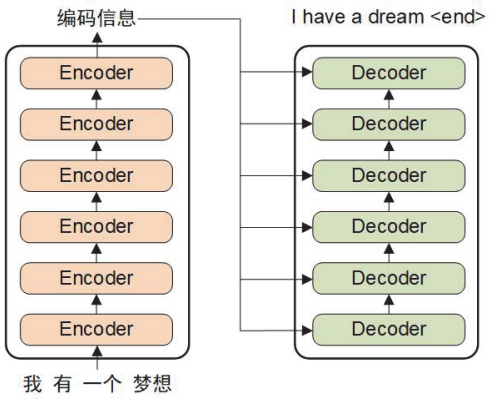

图 3-13

如图 3-14 所示，用这样一个简单的 Transformer 架构，不但训练速度比之前的模型更快，而且在 BLEU 这个英文到法文的测试集上，以及英文到德文的测试集上取得了最好的结果，其得分分别为 28.4 和 41.8，在各种模型中得分高。

Model	BLEU		Training Cost (FLOPs)	
	EN-DE	EN-FR	EN-DE	EN-FR
ByteNet [18]	23.75			
Deep-Att + PosUnk [39]		39.2		$1.0 \cdot 10^{20}$
GNMT + RL [38]	24.6	39.92	$2.3 \cdot 10^{19}$	$1.4 \cdot 10^{20}$
ConvS2S [9]	25.16	40.46	$9.6 \cdot 10^{18}$	$1.5 \cdot 10^{20}$
MoE [32]	26.03	40.56	$2.0 \cdot 10^{19}$	$1.2 \cdot 10^{20}$
Deep-Att + PosUnk Ensemble [39]		40.4		$8.0 \cdot 10^{20}$
GNMT + RL Ensemble [38]	26.30	41.16	$1.8 \cdot 10^{20}$	$1.1 \cdot 10^{21}$
ConvS2S Ensemble [9]	26.36	**41.29**	$7.7 \cdot 10^{19}$	$1.2 \cdot 10^{21}$
Transformer (base model)	27.3	38.1	**$3.3 \cdot 10^{18}$**	
Transformer (big)	**28.4**	**41.8**	$2.3 \cdot 10^{19}$	

图 3-14

3.5 本章小结

本章系统地介绍了大模型的理论基础。首先，讨论了自然语言处理领域中的各种数据类型，包括时间序列数据、分词、Token、Embedding，以及语义向量空间的概念。接着，回顾了语言模型的历史演进，从早期的统计语言模型到现代的神经网络语言模型的发展过程。随后，详细讲解了注意力机制，涵盖了 RNN 模型、Seq2Seq 模型，以及 Attention 机制的原理和应用。最后，深入解析了 Transformer 架构，包括其整体架构、Self-Attention、Multi-Head Attention、Encoder 和 Decoder 的设计，并展示了 Transformer 在实验中的效果。

本章为读者提供了扎实的理论基础，以方便理解大模型的工作原理和技术细节。