

Transformers For Machine Learning

John Yuan

2024 年 4 月 29 日

目录

0.1	前言	6
0.2	序言	7
0.3	为什么选择这本书?	7
0.4	这本书是为谁编写的?	8
0.5	本书涵盖的内容	8
0.6	作者	10
0.7	贡献者	10
1	深度学习和 Transformer: 简介	11
1.1	深度学习: 历史视角	11
1.2	Transformers 和分类学	13
1.2.1	改进的 Transformer 架构	13
1.2.2	预训练方法及应用	18
1.3	资源	18
1.3.1	库和实现	18
1.3.2	书籍	18
1.3.3	课程、教程和讲座	19
1.3.4	案例研究和详细信息	20

2 Transformer: 基础知识和简介	20
2.1 编码器-解码器架构	20
2.2 序列到序列	21
2.2.1 编码器	21
2.2.2 解码器	22
2.2.3 训练	23
2.2.4 基于 RNN 的编码器-解码器的问题	23
2.3 注意力机制	23
2.3.1 背景	23
2.3.2 Based-Score 的注意力类型	25
2.3.3 基于注意力的序列到序列	27
2.4 Transformer	28
2.4.1 源和目标表示	29
2.4.2 注意力层	31
2.4.3 残差和层归一化	34
2.4.4 位置前馈网络	35
2.4.5 编码器	35
2.4.6 解码器	35
2.5 案例研究: 机器翻译	36
2.5.1 目标	36
2.5.2 数据、工具和库	36
2.5.3 实验、结果和分析	36
3 双向编码器表示	52
3.1 BERT	52
3.1.1 架构	52
3.1.2 预训练	53
3.1.3 微调	55
3.2 BERT 变体	57
3.2.1 RoBERTa BERT 预训练方法 (RoBERTa)	57
3.3 应用	57

3.3.1	TaBERT	57
3.3.2	BERTopic	59
3.4	BERT 见解	59
3.4.1	BERT 句子表示	59
3.4.2	BERTology	60
3.5	案例研究：用 Transformer 主题建模	61
3.5.1	目标	61
3.5.2	数据、工具和库	61
3.5.3	实验、结果和分析	63
3.6	案例研究：微调	72
3.6.1	目标	72
3.6.2	数据、工具和库	72
3.6.3	实验、结果和分析	75
4	多语言 Transformer 架构	83
4.1	多语言 Transformer 架构	83
4.1.1	基本多语言 Transformer	84
4.1.2	单编码器多语言 NLU	85
4.1.3	双编码器多语言 NLU	96
4.1.4	多语言 NLU	100
4.2	多语言数据	101
4.2.1	预训练数据	101
4.2.2	多语言基准	102
4.3	多语言迁移学习见解	103
4.3.1	零样本跨语言学习	103
4.3.2	与语言无关的跨语言	106
4.4	案例研究	107
4.4.1	目标	107
4.4.2	数据、工具和库	107
4.4.3	实验、结果和分析	108

5 Transformer Block 修改	119
5.1 Transformer Block 修改	119
5.1.1 轻量级 Transformer	119
5.1.2 Transformer 之间的连接	124
5.1.3 自适应计算时间	125
5.1.4 Transformer Blocks 之间的循环关系	127
5.1.5 分层 Transformer	131
5.2 改良 Multi-HEAD 自注意力 Transformer	131
5.2.1 多头自注意力的结构	131
5.2.2 降低自注意力的复杂性	135
5.2.3 改进多头注意力	149
5.2.4 偏向先验的注意力	152
5.2.5 原型查询	152
5.2.6 压缩键值内存	153
5.2.7 低阶近似	155
5.3 训练任务效率的修改	157
5.3.1 ELECTRA	157
5.4 Transformer 子模块修改	158
5.4.1 Switch Transformer	159
5.5 案例研究：情绪分析	160
5.5.1 目标	160
5.5.2 数据、工具和库	160
5.5.3 实验、结果和分析	162
6 预训练和特定应用	167
6.1 文本处理	167
6.1.1 特定领域的 Transformer	168
6.1.2 文本到文本 Transformer	169
6.1.3 文本生成	170
6.2 计算机视觉	174
6.2.1 视觉 Transformer	175

6.3	自动语音识别	176
6.3.1	Wav2vec 2.0	176
6.3.2	Speech2Text2	177
6.3.3	HuBERT: 隐藏单元 BERT	177
6.4	多模式和多任务 Transformer	178
6.4.1	视觉和语言 BERT (VilBERT)	178
6.4.2	Unified Transformer (UniT)	179
6.5	使用 TIMESFORMER 进行视频处理	180
6.5.1	补丁嵌入	181
6.5.2	自注意力	181
6.6	图 Transformer	183
6.6.1	图中的位置编码	184
6.6.2	图 Transformer 输入	184
6.7	强化学习	189
6.7.1	决策转换器	190
6.8	案例研究: 自动语音识别	192
6.8.1	目标	192
6.8.2	数据、工具和库	192
6.8.3	实验、结果和分析	192
7	Transformer 的可解释性和可解释性技术	199
7.1	可解释系统的特征	199
7.2	影响可解释性的相关领域	200
7.2.1	可视化方法	202
7.2.2	模型蒸馏	206
7.2.3	内在方法	209
7.3	注意和解释	212
7.3.1	注意力不是解释	212
7.3.2	注意力是不是解释	216
7.4	量化注意力流	218
7.4.1	作为 DAG 的信息流	218

7.4.2	注意力的推出	219
7.4.3	注意力流	220
7.5	案例研究：文本分类	220
7.5.1	目标	220
7.5.2	数据、工具和库	221
7.5.3	实验、结果和分析	221

0.1 前言

著名人工智能先驱、诺贝尔奖获得者赫伯特·西蒙强调，“注意力”是信息经济最宝贵的资源，需要在过剩的信息资源中有效分配注意力。在写了关于意义感知人工智能的基础论文后，最近担任 MIT-Princeton-USAF-AFRL AI Faculty-SME，我有幸受邀在同一期刊的 ASQ 特刊上发表文章，并成为马尔科姆·鲍德里奇（Malcolm Baldrige）国家质量奖管理员，并与西蒙博士在同一全球学术引文影响研究中并列。

鉴于上述背景，我很高兴与您分享最全面和最新的研究、实践、案例研究概要，当今可用的应用程序可以在 Transformer 的最新人工智能技术进步上提供最佳的投资回报率，灵感来自论文“Attention is All You Need.”。自从谷歌在 2017 年推出 Transformer 架构以来，Transformer 在以上下文为中心的实现方面为意义感知 AI 提供了指数级的改进，作为基于注意力机制（例如点积注意力和多头注意力）的深度（神经网络）学习模型。在增强的顺序数据并行处理方面取得的进步使得高效的上下文敏感，因此对于更大的数据集来说更“有意义”，并且比以前更可行。

涵盖与跨自然语言处理（NLP）等应用程序的 Transformer 相关的神经网络架构的最新进展）、语音识别、时间序列分析以及计算机视觉和跨越科学、医学和金融的特定领域模型，本书旨在满足学术界和工业界多个受众的理论、研究、应用和实践需求，包括研究生和研究人员、本科生、行业从业者和专业人士。本书通过实践案例研究完善了理论驱动的应用和实践，关注人工智能的可解释性，这是实践中一个日益重要的主题，因为更加关注道德人工智能和可信人工智能等问题。

–Yogesh Malhotra 博士创始人兼首席执行官美国风险投资和私募股权

公司 Global Risk Management Network LLC 科学家 www.yogeshmalhotra.com

0.2 序言

0.3 为什么选择这本书?

自 2012 年以来，深度学习架构开始主导机器学习领域。然而，大多数突破都是在计算机视觉应用中。这一成功的主要推动力是卷积神经网络 (CNN) 架构。CNN 的效率和并行性使计算机视觉架构能够对大量数据进行预训练，这被证明是其成功的关键因素。此后多年，自然语言处理 (NLP) 应用并未受到新的深度学习革命的太大影响。传统的序列建模架构，例如循环神经网络 (RNN) 和长短期记忆 (LSTM)，已用于 NLP 应用程序。这种架构的顺序性质限制了在相同规模的数据上进行训练的可能性，而这些数据对计算机视觉显示出价值。

2017 年，Google 推出了 Transformer 架构，以更加并行化的方式处理顺序数据。这种架构允许在比以前更大的数据集上进行有效的训练。这使得 Transformer 彻底改变了 NLP 领域，就像 CNN 彻底改变了计算机视觉一样。

Transformers 现在正成为许多神经架构的核心部分，广泛应用于 NLP、语音识别、时间序列和计算机视觉等领域。OpenAI 在其 GPT2/GPT3 中使用 Transformer，在各种 NLP 任务中具有最先进的性能水平。DeepMind 击败顶级职业星际争霸选手的 AlphaStar 程序也采用了 Transformer 架构。Transformers 经历了多次改编和改动，产生了更新的技术和方法。没有一本可以在一个地方捕获 Transformer 的基础知识和各种变化。

本书是为数据科学家和研究人员（学术界和工业界）提供的独特资源。

- 一本全面的参考书详细解释了与 Transformer 相关的所有算法和技术。
- 全面涵盖了 60 多种 Transformer 架构。
- 一本用于了解如何在不同的 NLP 应用、语音、时间序列和计算机视觉中应用 Transformer 技术的书。
- 实用的提示和技巧每种架构以及如何在现实世界中使用它。

- 实践案例研究为不同主题的现实场景提供实用见解，例如机器翻译、主题挖掘、零样本多语言分类、情感分析、自动语音从任务、过程和分析角度，对识别、文本分类/分类进行了足够的详细介绍，所有这些都可以在 Google Colab 中运行。

0.4 这本书是谁编写的?

最先进的 Transformer 架构的理论解释这些讲座将吸引研究生和研究人员（学术界和工业界），因为它将提供简单的切入点，对快速发展的领域进行深入讨论。实用的实践案例研究和代码将吸引本科生、从业者和专业人士，因为它允许快速实验并降低进入该领域的门槛。

Transformers 已经是 NLP 深度学习架构的基石。它们也迅速应用于计算机视觉和音频等其他应用中。任何有关神经网络、深度学习或人工智能的课程都必须深入讨论 Transformer 作为关键的最先进架构。这本书可以作为读者的参考书，温习他们的具体理解，或者探索 Transformer 的用途来应对特定的挑战。我们希望本书成为读者在面临不同挑战或缺乏理解时可以多次参考、获得见解和使用的资源。

0.5 本书涵盖的内容

本书通过数学理论和实际用例深入介绍 Transformer 的基础知识。下面给出了每章的简要说明。

1. 第一章将从时间轴、历史以及它对学术界和工业界的影响向读者介绍变形金刚。然后，我们将根据分类法以及每章如何从理论、实践和应用角度呈现来制定完整的路线图。然后本章对其他章节中将使用的资源、工具、书籍和课程等实际方面进行了全面的讨论。

2. 第 2 章首先介绍序列到序列模型及其局限性。然后，本章逐步介绍了 Transformer 的各种构建模块，例如注意力、多头注意力、位置编码、残差连接和编码器-解码器框架。所有这些功能单元都从理论和实践的角度进行了详细的处理，以便读者全面掌握该主题。最后，本章总结了一个使用 Transformer 进行机器翻译任务的实际案例研究，显示了操作方面的内容。3. BERT 的出现彻底改变了自然语言处理（NLP）领域，并帮助在许多传统的

挑战性任务中接近人类水平。

1. 第三章介绍了 BERT 架构的细节，以及如何针对经典 NLP 任务（例如单/对文本分类、标记标记和问题回答）对其进行预训练和微调。本章还讨论了 BERT 学领域，这是与 BERT 内部工作原理及其如何处理和分析文本和信息相关的研究。最后，本章介绍了一些深度学习架构，这些架构修改了 BERT 以提高效率（例如 RoBERTa）和其他类型的 NLP 应用程序（例如用于表格数据的 NLP - TaBERT）。本章最后提供了使用 BERT 进行情感分类和主题建模应用的真实案例研究。
4. 多语言迁移学习是 Transformer 架构对机器学习领域产生重大影响的一个领域。第 4 章介绍了基于 Transformer 的概述多语言架构以及如何针对 NLP 任务预训练和微调跨语言迁移学习。本章还概述了用于多语言 NLP 的最先进基准。本章进一步提供了一些关于识别 NLP 中影响跨语言和零样本迁移学习的因素的研究和技术的见解。最后，提出了使用多语言通用句子编码器进行零样本跨语言情感分类的实际案例研究。
5. 在第五章中，我们讨论了对标准转换器架构进行的各种修改，以处理内存有限的较长序列，构建更快、更高质量的转换器模型，并在文本生成和摘要方面表现更好。我们还讨论了模型架构和方法之间的主要差异，这些差异以关键思想为中心，例如知识蒸馏和通过降低注意机制复杂性来提高计算效率。本章包括一个使用预先训练的 Transformer 模型进行情感分类的案例研究，包括对该模型的多头注意力机制的内容的介绍。
6. 自 BERT 以来，不同领域已经提供了多种预训练模型，提供了可以针对科学、医学和金融领域特定领域数据进行微调的模型。此外，特定于语言的预训练模型在下游特定于语言的任务上提供了越来越有竞争力的结果。在第六章中，我们讨论可用的预训练模型，展示它们的好处以及在计算机视觉、语音、时间序列和文本等特定领域的应用。本章包括一个案例研究，比较了三种基于 Transformer 的自动语音识别模型的性能。
7. 在许多关键应用中，考虑到基于 Transformer 的模型的黑盒性质，需要从可解释性的角度来理解模型。在第七章中，我们将介绍解决可解释性的模型的特征、影响可解释性的相关领域、应用于基于 Transformer 和基于注

意力的系统的可解释方法的分类，最后，在电子健康记录系统中使用具有不同可解释技术的 Transformer 的详细案例研究，以使其变得更加实用。

0.6 作者

Uday Kamath 花费了二十多年的时间来开发分析产品，并将这些经验与统计、优化、机器学习、生物信息学和进化计算方面的学习相结合。他为许多期刊、会议和书籍做出了贡献，是《XAI：可解释 XAI 简介》、《NLP 和语音识别深度学习》、《掌握 Java 机器学习》和《机器学习：Java 端到端指南》的作者开发商。他担任过许多高级职务：Digital Reasoning 的首席分析官、Falkonry 的顾问以及 BAE Systems Applied Intelligence 的首席数据科学家。卡马斯博士拥有多项专利，并在合规性、网络安全、金融犯罪和生物信息学等领域使用人工智能构建了商业产品。他目前担任 Smarsh 的首席分析官。他负责数据科学、利用深度学习、Transformer、可解释的人工智能以及金融领域和医疗保健领域的语音和文本现代技术进行分析产品的研究。

Kenneth L. Graham 拥有二十年解决多个领域的定量问题的经验。领域，包括蒙特卡洛模拟、自然语言处理、异常检测、网络安全和行为分析。在过去的十年中，他专注于为政府和行业构建可扩展的 NLP 解决方案，包括实体共指解析、文本分类、主动学习、自动语音识别和时间标准化。他目前在 AppFolio 担任高级机器学习工程师。Graham 博士在自然语言处理方面拥有五项专利、七篇研究出版物和凝聚态物理学博士学位。

Wael Emara 在学术界和工业界拥有二十年的经验。他拥有计算机工程和计算机科学博士学位，重点研究机器学习和人工智能。他的技术背景和研究涵盖信号和图像处理、计算机视觉、医学成像、社交媒体分析、机器学习、作者和自然语言处理。Emara 博士为许多机器学习主题的同行评审出版物做出了贡献，并且活跃于大纽约地区的技术社区。他目前是 Smarsh, Inc. for Digital Reasoning 的首席数据科学家，致力于最先进的人工智能 NLP 系统的研究。

0.7 贡献者

Krishna Choppella BAE Systems AI Toronto, Canada

Mitch NaylorSmarsh , Smarsh, Inc. Nashville, Tennessee
Vedant Vajre Stone Bridge High School Ashburn, Virginia

1 深度学习和 Transformer：简介

Transformers 是深度学习模型，在自然语言处理、计算机视觉和语音识别等多个领域取得了最先进的性能。事实上，最近提出的 Transformer 模型变体的大量涌现意味着研究人员和实践者都发现很难跟上步伐。在本章中，我们提供了与 Transformer 创新直接或间接相关的各种研究的简要历史。接下来，我们讨论基于计算、内存、应用程序等效率的架构变化的分类法，这可以帮助导航复杂的创新空间。最后，我们提供了工具、图书馆、书籍和在线课程等资源，读者可以在追求目标的过程中受益 [1]。

1.1 深度学习：历史视角

20 世纪 40 年代初，S. McCulloch 和 W. Pitts 使用简单的电子计算机称为“阈值逻辑单元”的电路，通过模拟大脑的工作方式来模拟智能行为。这个简单模型的第一个神经元具有输入和输出，当“加权和”低于阈值时，它会生成输出 0，否则生成 1，这后来成为所有神经架构的基础。权重不是学习的而是调整的。唐纳德·赫布 (Donald Hebb) 在其著作《行为的组织》(1949) 中奠定了复杂神经处理的基础，提出神经通路如何让多个神经元随着时间的推移而放电和增强 [2]。Frank Rosenblatt 在他的开创性工作中扩展了 McCulloch-Pitts 神经元，将其称为“Mark I Perceptron”；给定输入，它使用线性阈值逻辑生成输出。

感知器中的权重是通过重复传递输入并减少预测输出和期望输出之间的差异来“学习”的，从而诞生了基本的神经学习算法。Marvin Minsky 和 Seymour Papert 后来出版了《Perceptrons》一书，该书揭示了感知器在学习简单异或函数 (XOR) 方面的局限性，从而引发了所谓的“第一个人工智能冬天 [3]”。约翰·霍普菲尔德介绍了“Hopfield Networks”，它是第一个用作内容可寻址记忆系统的循环神经网络 (RNN) [4]。1986 年，David Rumelhart、Geoff Hinton 和 Ronald Williams 发表了开创性著作“通过反向传播误差学习表征”

[5]。他们的工作证实了使用许多“隐藏”层的多层神经网络如何克服感知器在通过相对简单的训练程序学习复杂模式方面的弱点。这项工作的构建模块是 S. Linnainmaa 多年来的各种研究奠定的, P. Werbos、K. Fukushima、D. Parker 和 Y. LeCun [6, 7, 8, 9, 10]。

LeCun 等人通过他们的研究和实施, 首次广泛应用神经网络来识别美国使用的手写数字。S.PostalService[11]。这项工作是深度学习历史上的关键里程碑, 证明了卷积运算和权重共享在学习计算机视觉特征中的实用性。

反向传播作为关键的优化技术, 遇到了梯度消失、梯度爆炸等一系列问题, 以及无法学习长期信息, 仅举几例 [12]。Hochreiter 和 Schmidhuber 在他们的工作“长短期记忆 (LSTM)”架构中, 演示了如何克服长期依赖性问题随着时间的推移克服反向传播的缺点 [13]。

2006 年 Hinton 等人发表突破性论文“深度信念网络的快速学习算法”;这是深度学习复兴的原因之一 [14]。该研究强调了使用无监督方法进行逐层训练的有效性, 然后进行监督“微调”, 以实现字符识别方面的最先进结果。Bengio 等人在随后的开创性工作中提出为什么多层深度学习网络与浅层神经网络相比可以分层学习特征 [15]。在他们的研究中, Bengio 和 LeCun 强调了通过卷积神经网络 (CNNs)、受限玻尔兹曼机 (RBMs) 和深度信念网络 (DBNs) 等架构进行深度学习的优势。RBM), 并通过无监督预训练和微调等技术, 从而激发下一波深度学习浪潮 [16]。斯坦福大学人工智能实验室负责人李飞飞与其他研究人员一起推出了 ImageNet, 它产生了最广泛的图像集合, 并首次强调了数据在学习基本任务, 例如对象识别、分类和聚类。计算机硬件的改进 (主要通过 GPU) 使吞吐量每五年增加近 10 倍, 以及大量可供学习的数据的存在导致了该领域的范式转变。深度学习网络不再是许多复杂应用程序主要关注的手工设计特征, 而是通过从大量训练数据中学习, 出现必要的特征, 成为许多最先进技术的基础。

Mikolov 等人和 Graves 提出了使用 RNN 和长短期记忆的语言模型, 后来成为许多自然语言处理 (NLP) 架构的构建块 [17, 18]。Collobert 和 Weston 的研究论文有助于演示许多概念, 例如预训练词嵌入、文本 CNN 以及多任务学习的嵌入矩阵共享 [19]。Mikolov 等人进一步改进了 Bengio 等人提出的 enbed-dings 一词的训练效率。通过消除隐藏层并制定一个近似

的学习目标，产生“word2vec”，这是一种高效的大规模词嵌入实现 [20] [21]。Sutskever 的研究提出了一种无 Hessian 优化器来有效地训练 RNN 的长期依赖性，这是一项突破复兴 RNN 的使用，特别是在 [22] [?]。因此，序列到序列框架成为各种 NLP 任务的核心架构，例如选区解析、命名实体识别 (NER)、机器翻译、问答和摘要等。此外，甚至谷歌也开始用具有序列到序列神经机器翻译模型更换其基于短语的 Deep Divemontolithic 机器翻译模型。为了克服序列到序列框架的瓶颈问题，Bahdanau 等人的开创性工作。提出了注意力机制，它在 transformer 及其变体中发挥着至关重要的作用 [23]。

1.2 Transformers 和分类学

Transformer 架构于 2017 年在论文《Attention Is All You Need》中引入，用于解决序列到序列问题。它是使用循环层或卷积层的替代方案。自其推出以来，人们对改进标准 Transformer 的各种方法进行了广泛的研究。两项调查 [24, 25] 对与 Transformer 相关的论文进行了分类。Transformer 研究主要集中在三个方面：架构修改、预训练方法和应用。在本书中，我们将花时间研究架构修改的子集、大型语言模型的预训练方法（如 BERT[26]）以及一些应用。

1.2.1 改进的 Transformer 架构

Transformer 架构修改的 Transformer 架构可以分为两种广义类别 [27]：Transformer 块的内部布置的更改以及 Transformer 块组成层的更改 表 1.1。

1. Transformer block 改变

Transformer 模块更改总结了 Transformer 修改的类型。到目前为止，对 Transformer 模块的修改已分为五类 [24]

- 减少内存占用和计算
- 添加 Transformer 模块之间的连接
- 自适应计算时间（例如，允许在训练期间提前停止）
- 循环或分层结构

- 更彻底地改变架构（例如，神经架构搜索）

Transformer block 改变的类型

Modification	Transformer
Lightweight transformers	Lite Transformer [28] Funnel Transformer [29] DeLighT[30]
Cross-block connectivity	Realformer[31] Transparent Attention[32]
Adaptive computation time	Universal Transformer[33] Conditional Computation Transformer[34] DeeBERT[35]
Recurrent	Transformer-XL [36] Compressive Transformer[37] Memformer[38]
Hierarchical	HIBERT[39] Hi-Transformer[40]
Different architectures	Macaron Transformer[41] Sandwich Transformer[42] Differentiable Architecture Search[43]

在本书中，我们将重点关注几种架构修改，这些修改允许 Transformer 处理更长的序列和/或降低注意机制的计算复杂性。我们在 表 1.1

2. Transformer 子层改变

在第 2 章中，我们将详细了解 Transformer 块的结构，涵盖其四个组件，以便稍后我们可以讨论研究人员的方法修改了它们。一般来说，Transformer 块 [44] 有四个部分：位置编码、多头注意力、具有层归一化的残差连接 [45] 和位置明智的前馈网络。Transformer 子层的变化集中这四类组件，其中大部分都集中在多头注意力的改变方面 [24, 25]。表 1.2 显示了经过修改的多头注意力机制的 Transformer。

Modification	Transformer
Low-rank	Performer[46] Nystromformer [47] Synthesizer[48]
Attention with prior	Gaussian Transformer[49] Realformer[31] Synthesizer[48] Longformer[50]
Improved multi-head attention	Talking-heads Attention[51] Multi-Scale Transformer[52]
Complexity reduction	Longformer[50] Reformer[53] Big Bird[54] Performer[46] Routing Transformer[55]
Prototype queries	Clustered Attention[56] Informer[57]
Clustered key-value memory	Set Transformer[58] Memory Compressed Transformer[59] Linformer[60]

多头注意力人们在多头注意力机制方面付出了很多努力；研究其二次计算复杂性、解决所述复杂性的方法以及如何针对特定类型的问题改变它。这项工作大部分分为两大类：降低注意力机制的计算复杂性，或者改变注意力机制，使其能够学习更多东西。正如参考文献 [27, 25] 中

所讨论的，有很多方法可以解决注意力机制的复杂性。有低阶近似，例如 Linformer[60] 和 Performer[46]。有多种方法可以稀疏注意力机制，其中一些方法有效地将注意力机制的复杂性降低到与序列长度呈线性关系。例如，Longformer[50] 和 BigBird[54] 通过固定给定标记可以出现的位置来增加稀疏性。其他一些转换器，如 Reformer[53]，通过对输入标记进行排序或聚类来引入可学习的稀疏性。还有其他一些可以缩小尺寸注意力矩阵 [27, 27]。

还有各种工作试图改进多方面的头部注意力机制 [27]。例如，注意力头被允许相互“交流”和/或共享信息 [61, 51, 62, 63]，学习参与的最佳跨度 [64]，并在不同的任务中使用不同的注意力跨度。注意头 [65]。此列表并不详尽。我们在第 5 章中讨论了几种这样的方法。

位置编码 [44] 是一种将序列顺序编码到转换器中的方法。它们还包括修改 transformer 块组件的另一种途径。到目前为止，已经使用了四种位置编码 [27]：绝对位置编码（如标准 Transformer 中的编码）、相对位置编码（如 Transformer-XL 中）、具有绝对和相对位置的混合编码信息，以及以其他方式提供有关序列顺序信息的隐式编码。如表 1.3 所示。我们在第 2 章中讨论标准 Transformer 中使用的绝对位置编码，在第 5 章中讨论 Transformer-XL 中使用的相对编码。

表 1.3 位置编码的更改修改

Modification	Transformer
Absolute position	Original Transformer[44]
Relative position	Transformer-XL[36]
Absolute/relative hybrid	Roformer[66]
Other representations	R-Transformer[17]

深入探讨残差连接和位置前馈网络一些工作包括对多头注意力机制和位置前馈网络之后的残差块进行更改，包括层的位置归一化，用其他东西交换层归一化，完全删除层归一化 [27]，或者引入或可逆残余层以节省内存（在 Reformer 中使用）[53]。Reformer 将在第 5 章中讨

论。其他工作已经研究了改变位置明智前馈网络的方法，包括改变激活函数、增加其表示能力或删除前馈网络。

1.2.2 预训练方法及应用

大量工作集中在如何预训练 Transformer 上预训练。有仅编码器模型，例如 BERT[26]，仅解码器模型，例如著名的生成预训练 Transformer 模型 GPT-3 [67]，以及编码器-解码器模型，例如 T5[68] 和 ByT5[69]。BERT 在第 3 章中详细讨论，第 5 章中的 T5 以及第 6 章中的 ByT5。

已经有许多针对特定数据域（例如，金融或医学文本）和特定类型的数据（例如，图像）的应用程序和特定领域的转换器或视频。我们在第 6 章资源中讨论了几个这样的应用程序

1.3 资源

在本节中，我们将讨论一些对研究人员和从业者有用的资源。

1.3.1 库和实现

这里有一些有用的库、工具和实现（表 1.4）

表 1.4 库和实现工具

Organization	Language and Framework	API	Pre-trained
AllenNLP	Python and PyTorch	Yes	Yes
HuggingFace	Jax, PyTorch, and TensorFlow	Yes	Yes
Google Brain	TensorFlow	Yes	
GluonNLP	MXNet	Yes	Yes

1.3.2 书籍

我们发现有用的一些书籍有

- Transfer Learning for Natural Language Processing by Paul Azunre [70]

- Transformers for Natural Language Processing by Denis Roth-man[71]
- Deep Learning Algorithms: Transformers, gans, encoders, rnns, cnns, and more by Ricardo A. Calix [35]
- Python Transformers By Huggingface Hands On by Joshua K.Cage[72]//
- Deep Learning for NLP and Speech Recognition by Uday Kamath, John Liu, and James Whitaker[73]

1.3.3 课程、教程和讲座

- The Annotated Transformer by Alexander Rush et al. <http://nlp.seas.harvard.edu/2018/04/03/attention.html>
- HuggingFace course on transformers <https://huggingface.co/course>
- DeepLearning.AI course on sequence models <https://www.coursera.org/learn/nlp-sequence-models>
- DeepLearning.AI course on transformers and BERT <https://www.coursera.org/learn/attention-models-in-nlp>
- Stanford CS224N: NLP with Deep Learning by Christopher Manning <http://web.stanford.edu/class/cs224n/>
- UC Berkeley's Applied Natural Language Processing by David Bamman <https://people.ischool.berkeley.edu/~dbamman/info256.html>
- Advanced NLP with spaCy by Ines Montani <https://course.spacy.io/en/>
- Deep Learning for Coders with fastai and PyTorch by Sylvain Gugger and Jeremy Howard <https://course.fast.ai/>
- Jay Alammar's visual explanation of transformers and related architectures <https://jalammar.github.io/>

1.3.4 案例研究和详细信息

在第 2 章至第 7 章的末尾，我们提供了一个案例研究，使读者能够了解如何应用本章中讨论的一个或多个模型和方法，或者它们在应用于以下领域时如何相互比较：一样的问题。案例研究的目的是为使用 transformer 模型提供一个小起点，以便可以进一步扩展。每个案例研究都被选择在至少与 NVIDIA K80 一样强大的 GPU 上运行大约一小时（Google 合作实验室免费提供这些）。本书随附的 Github 存储库中也提供了案例研究：<https://github.com/CRCTransformers/deepdive-book>。

2 Transformer：基础知识和简介

许多自然语言处理和语音识别技术需要能够处理大型序列作为输入并将其转换为特定的输出。循环神经网络 (RNN) 等传统方法存在一些缺点阻碍了现实世界的解决方案。transformer 已成为克服大多数这些限制的基本设计作品，并且它们具有最先进的结果。本章首先介绍序列到序列模型及其局限性。然后，本章逐步介绍了 Transformer 的各种构建模块，例如注意力、多头注意力、位置编码、残差连接和编码器-解码器框架。所有这些功能单元都从理论和实践的角度进行了详细的处理，以便读者全面掌握该主题。最后，一个真实世界的案例研究通过使用众所周知的库和数据集展示操作方面来结束本章。

2.1 编码器-解码器架构

许多 NLP 问题，例如机器翻译、问答和文本摘要，仅举几例，都使用对序列作为输入来训练模型。一般来说，序列的长度是可变的。使用编码器-解码器架构（如图 2.1 所示）是解决该问题的常见做法。编码器组件采用可变长度序列并将其转换为固定长度输出状态，解码器组件采用固定长度状态并将其转换回可变长度输出。

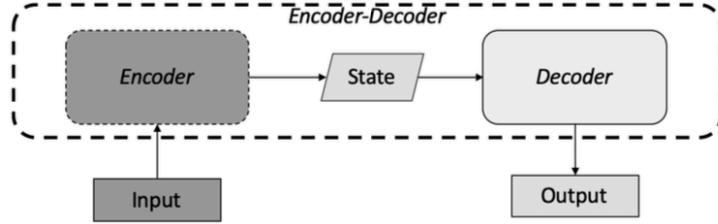


Figure 2.1 Encoder-Decoder architecture.

2.2 序列到序列

2.2.1 编码器

序列到序列机器翻译和许多其他 NLP 问题使用序列产生了有希望的结果-to-sequence 架构（缩写为 seq2seq），具有基于循环神经网络（例如 RNN）的编码器和解码器 [74, 75]。图 2.2 展示了一个机器翻译的例子，其中一个法语句子，J'aime le thé.，它的英语等效句子，I love tea. 形成句子对作为训练输入的示例。标记 <eos> 和 <bos> 分别是处理句子结尾和开头的特殊方法。如图 2.2 所示，编码器的最终隐藏状态充当解码器的输入。

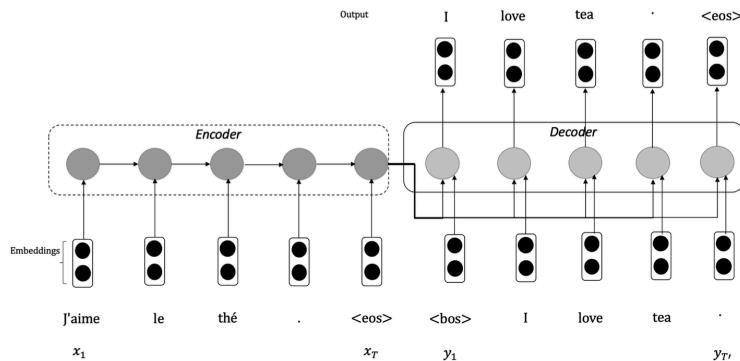


Figure 2.2 RNN-based sequence-to-sequence for machine translation.

输入句子被标记为单词，单词被映射到特征向量，特征向量是编码器的输入。假设输入序列由 x_1, \dots, x_T 表示，使得输入文本序列中的标记 x_t 是第

t 个标记。嵌入映射变换输入标记 x_1, \dots, x_T 到 $\mathbf{x}_1, \dots, \mathbf{x}_T$ 向量。任何时间 t 的单向 RNN 具有先前的隐藏状态 \mathbf{h}_{t-1} 和输入 \mathbf{x}_t 生成新的隐藏状态

$$\mathbf{h}_t = f(\mathbf{h}_{t-1}, \mathbf{x}_t) \quad (2.1)$$

(2.1) 上面的方程可以更具体地写成

$$\mathbf{h}_t = \tanh(\mathbf{W}^{(hh)}\mathbf{h}_{t-1} + \mathbf{W}^{(hx)}\mathbf{x}_t) \quad (2.2)$$

\mathbf{h}_t 称为上下文变量或上下文向量，对整个输入序列的信息进行编码，由下式给出：

$$\mathbf{c} = m(\mathbf{h}_1, \dots, \mathbf{h}_T) \quad (2.3)$$

其中 m 是映射函数，在最简单的情况下将上下文变量映射到最后一个隐藏状态

$$\mathbf{c} = m(\mathbf{h}_1, \dots, \mathbf{h}_T) = \mathbf{h}_T \quad (2.4)$$

增加架构的复杂性，RNN 可以是双向的，因此隐藏状态不仅取决于先前的隐藏状态 \mathbf{h}_{t-1} 和输入 \mathbf{x}_t ，还取决于下一个状态 \mathbf{h}_{t+1} 。

2.2.2 解码器

解码器具有编码器的输出，即上下文变量 \mathbf{c} ，和给定的输出序列 y_1, \dots, y'_T 生成解码输出。在 Sutskever 等人中，来自编码器的上下文变量——最终的隐藏状态——启动解码器，而在 Cho 等人中。上下文变量被传递到每个时间步。

与编码器类似，解码器在任何时间 t' 的隐藏状态由

$$\mathbf{s}_{t'} = g(\mathbf{s}_{t-1}, \mathbf{y}_{t'-1}, \mathbf{c}) \quad (2.5)$$

解码器的隐藏状态流向输出层， t' 处的下一个标记的条件分布由

$$P(\mathbf{y}_{t'} | \mathbf{y}_{t'-1}, \dots, \mathbf{y}_1, \mathbf{c}) = \text{softmax}(\mathbf{s}_{t-1}, \mathbf{y}_{t'-1}, \mathbf{c}) \quad (2.6)$$

2.2.3 训练

解码器预测每个时间步输出标记的概率分布，softmax 给出单词的分布。这样，编码器和解码器联合训练，交叉熵损失用于优化，由公式 2.7 给出

$$\max_{\theta} \frac{1}{N} \sum_{n=1}^N \log p_{\theta}(y^{(n)} | x^{(n)}) \quad (2.7)$$

连接 `<bos>` 和原始输出序列（不包括最终标记）作为训练期间解码器的输入称为 **teacher forcing**。**teacher forcing** 有助于解决训练 RNN 时收敛缓慢和不稳定的问题。

2.2.4 基于 RNN 的编码器-解码器的问题

如上一节所述，有关源句子的完整信息被压缩并编码在解码器组件使用的一个上下文变量中。随着输入大小的增加，压缩输入时会出现信息丢失。句子中的单词也可能具有复杂的结构和基于语言的长距离关联。以单个向量的压缩方式捕获这些也会导致效率低下。另一方面，编码器侧每个时间步的隐藏变量都是可用的，并且携带解码器网络使用的信息。解码器中的每个时间步都会受到编码器中隐藏变量的不同影响。RNN 还存在梯度消失和爆炸的问题 [12]。RNN 的计算问题之一是重复或依赖于先前的时间步使得架构非常难以并行化。

2.3 注意力机制

2.3.1 背景

注意力机制涉及选择性地关注特定元素，同时过滤掉不太相关的元素。人类视神经每秒接收数十亿比特的信息，而大脑的处理能力却要低得多。视觉注意力是注意力的一种形式，涉及对刺激物（例如人或无生命物体或特定任务）的定向和持续关注，从而使大脑能够进行有效的处理。因此，注意力机制使得人类能够只关注一小部分感兴趣的信息，从而实现资源的优化利用，从而获得更好的生存和成长。

“美国心理学之父”威廉·詹姆斯创建了一个由两部分组成的框架来解释视觉注意机制 [76]。在这个框架中，注意力的聚光灯使用非意志（非自愿）

和意志（自愿）线索来偏向感觉输入。非意志线索是非自愿的，并且基于目标的显着性和可注意到性环境。相比之下，意志提示是基于主体有意识地专注于目标的自愿努力。例如，通过给特定物体涂上不同的颜色来吸引注意力，或者关注哭泣的婴儿，都是非意志性线索。相反，为了回答问题或解决特定问题而关注特定文本，则是意志性线索。

在深度学习中的注意力机制的背景下，意志性线索映射查询、非意志线索的按键以及价值的感官输入。每个感觉输入（值）都映射到该感觉输入的非意志提示（键）。因此，注意力机制可以被认为是通过注意力池，使用查询（意志线索）和键（非意志线索）对值（感觉输入）进行偏向选择的过程，如图 2.3 所示。

注意力机制的设计方式是为了克服基于 RNN 的编码器-解码器架构描述的问题。

如图 2.3 所示，注意力机制可以被视为具有键和值的存储器以及一个层，当有人查询它时，该层从值生成输出，其键映射输入 [23]。为了形式化，让我们考虑由 n 个键值对 $(\mathbf{k}_1, \mathbf{v}_1), \dots, (\mathbf{k}_n, \mathbf{v}_n)$ 组成的存储单元，其中 $\mathbf{k}_i \in \mathbb{R}^{d_k}$ 和 $\mathbf{v}_i \in \mathbb{R}^{d_v}$ 。注意力层接收一个输入作为查询 $\mathbf{q} \in \mathbb{R}^{d_q}$ 并返回一个输出 $\mathbf{o} \in \mathbb{R}^{d_v}$ ，其形状与值 v 相同。注意力层使用得分函数 α 来测量查询和键之间的相似度，该函数返回得分 a_1, \dots, a_n 键值 $\mathbf{k}_1, \dots, \mathbf{k}_n$ 由

$$a_i = \alpha(\mathbf{q}, \mathbf{k}_i) \quad (2.8)$$

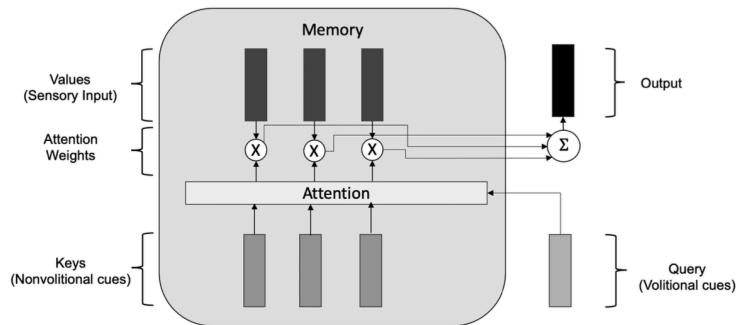


Figure 2.3 Attention mechanism showing query, keys, values, and output vector interactions.

注意力权重计算为分数上的 softmax 函数

$$\mathbf{b} = \text{softmax}(\mathbf{a}) \quad (2.9)$$

\mathbf{b} 的每个元素

$$b_i = \frac{\exp(a_i)}{\sum_j \exp(a_j)} \quad (2.10)$$

输出是注意力权重的加权和值

$$\mathbf{o} = \sum_{i=1}^n b_i \mathbf{v}_i \quad (2.11)$$

图 2.4 捕获从输入查询到输出的各种向量和标量之间的相互作用。

2.3.2 Based-Score 的注意力类型

正如所讨论的，分数函数 (\mathbf{q}, \mathbf{k}) 可以采取各种形式，这引起了许多注意力机制，各有各的优点。

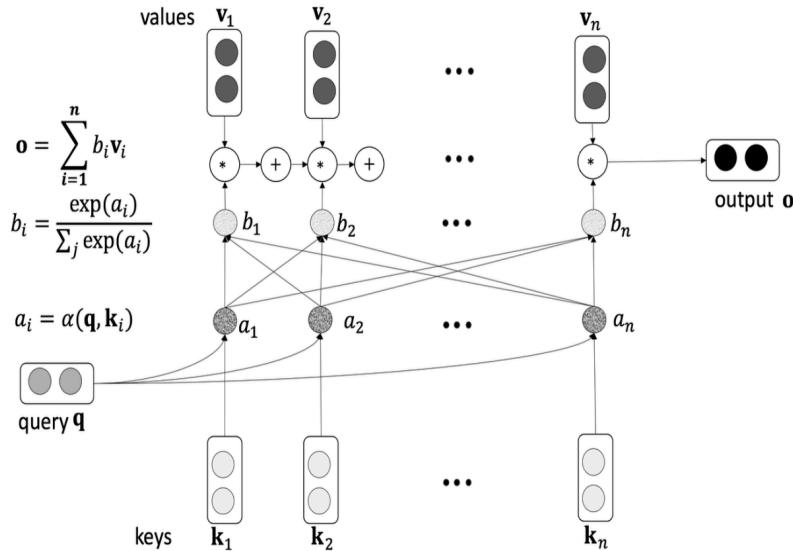


Figure 2.4 Attention mechanism showing query, keys, values, and output vector interactions.

1. 点积（乘法）

基于点积的评分函数是最简单的一种，没有需要调整的参数 [77]。

$$\alpha(\mathbf{q}, \mathbf{k}) = \mathbf{q} \cdot \mathbf{k} \quad (2.12)$$

2. 缩放点积或乘法

基于缩放点积的评分函数将点积除以 $\sqrt{d_k}$ ，以消除维度 d_k 的影响 [44]。根据 Vaswani 等人的研究，随着维数的增加，点积变得更大，这将 softmax 函数推入具有极端梯度的区域。

$$\alpha(\mathbf{q}, \mathbf{k}) = \frac{\mathbf{q} \cdot \mathbf{k}}{\sqrt{d_k}} \quad (2.13)$$

3. 线性、MLP 或加法

Luong 等人还进行了实验，将查询和键投影到维度 h 的隐藏层，学习权重 $(\mathbf{W}_k, \mathbf{W}_q)$ ，并使用具有注意层的编码器-解码器。将它们与值组合起来的 sigmoid 函数，如给出的

$$\alpha(\mathbf{q}, \mathbf{k}) = \mathbf{v}^T \tanh(\mathbf{W}_k \mathbf{k} + \mathbf{W}_q \mathbf{q}) \quad (2.14)$$

缩放点积，或基于点积的评分，比附加注意力机制更快、内存效率更高。

2.3.3 基于注意力的序列到序列

Encoder Decoder

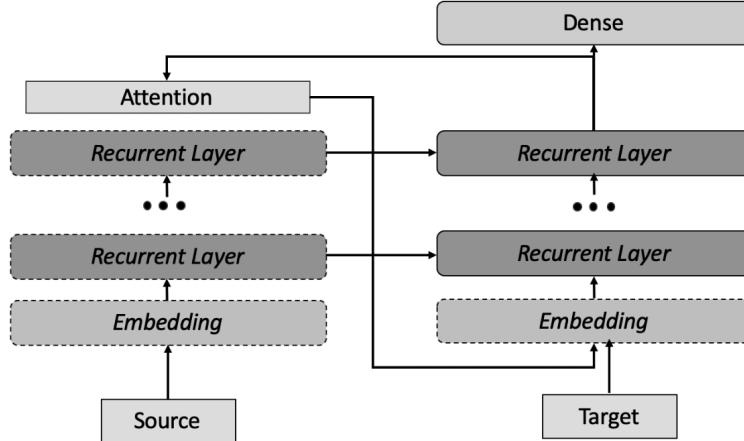


Figure 2.5 Encoder-decoder with attention layer.

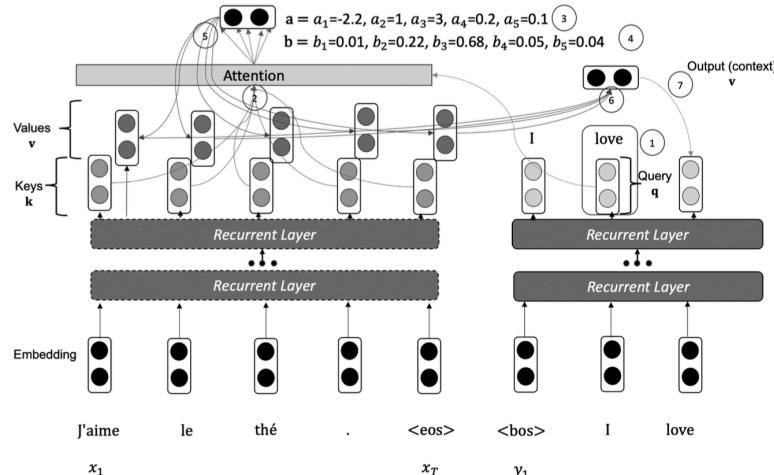


Figure 2.6 Sequence-to-sequence, at a given time t , when the decoder has generated *love* at $t-1$ and the token *tea* will be generated as the output.

图 2.6 序列到序列，在给定时间 t ，当解码器在 $t-1$ 生成 *love* 时，将生成标记 *tea* 作为输出。

编码器-解码器的一般变化，增加了注意力层和查询映射，键和值如图 2.5 所示。

1. 最后编码器状态的输出用作键和值 v
2. 最后一个解码器状态在时间 $t - 1$ 的输出用作查询 q
3. 注意力层 o 的输出（上下文变量）用于下一个解码器状态 t

为了更详细地理解流程，让我们考虑法语翻译示例，其中键和值是标记 $\{J'aime, l, the, .\}$ 的编码器状态，并且解码器在时间 $t - 1$ 生成了标记 I,love。如图 2.6 所示，解码器在时间 $t - 1$ 时的输出 love，作为查询流入注意力层，它与键结合并生成非归一化分数 $a = \alpha(q, k)$ ，然后将其归一化以给出注意力权重 b 。这些注意力权重进一步与值（编码器输出）结合以给出上下文变量或输出向量 o 。然后输出与先前的解码器状态结合以在时间 t 生成下一个标记 tea。

2.4 Transformer

如图 2.7 所示，Vaswani 等人提出的初级 transformer 结构。基于之前描述的编码器-解码器架构。Transformer 结合了卷积神经网络 (CNN) 的并行计算和循环神经网络 (RNN) 的优点来捕获长范围、可变长度的顺序信息。Transformer 架构中提出了许多系统性的改变，在接下来的几小节中，我们将详细介绍每一个改变。

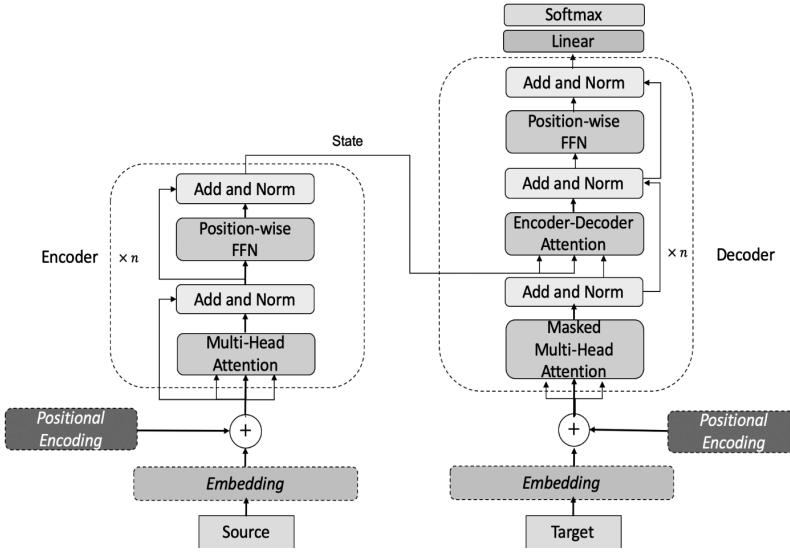


Figure 2.7 Transformer architecture.

2.4.1 源和目标表示

源和目标表示源词和目标词都被标记化，标记通过词嵌入和位置编码为所有句子提供位置编码表示。

1. 词嵌入

句子中标记的标准词嵌入查找可以将长度为 l 的句子转换为维度为 (l, d) 的矩阵 \mathbf{W} ，即 $\mathbf{W} \in \mathbb{R}^{l \times d}$ 。

2. 位置编码

	0	0	0
Position	0	0	1
	0	1	0
	0	1	1
	1	0	0
	1	0	1
	1	1	0
	1	1	1

Depth

Figure 2.8 Positional encoding for 8 positions with dimensionality 3.

词顺序和位置在大多数情况下起着至关重要的作用。NLP 任务的一部分。通过一次处理一个单词，循环神经网络本质上整合了单词顺序。在 Transformer 架构中，为了获得速度和并行性，循环神经网络被多头注意力层取代，我们稍后将详细介绍。因此，有必要将有关词序的信息显式传递到模型层作为捕获它的一种方法。这种词序信息的编称为位置编码。人们可以得出有效位置编码的各种要求。他们是

- (a) 每个时间步长的唯一编码值（句子中的单词）。
- (b) 不同长度句子的两个时间步之间的距离一致。

3. 编码结果的概括与句子的长度无关。4. 编码是确定性的。实现位置编码的所有要求的一种简单方法是使用二进制表示。图 2.8 突出显示了如何使用大小或深度为 3 的向量，我们可以使用满足上述所有要求的二进制值生成 8 个位置编码。每个位以灰色 (0) 和白色 (1) 表示，显示每个位置如何不同并且具有恒定的差异。从记忆的角度来看，使用二进制值的成本非常高。

如果句子的长度由 l 给出，则嵌入维度/深度由 d 给出，位置编码 \mathbf{P} 是相同维度的二维矩阵，即 $\mathbf{P} \in \mathbb{R}^{l \times d}$ 。每个位置都可以用方程表示，即沿 l 维度的 i 和沿 d 维度的 j

$$\mathbf{P}_{i,2j} = \sin(i/1000^{2j/d}) \quad (2.15)$$

$$\mathbf{P}_{i,2j+1} = \cos(i/1000^{2j/d}) \quad (2.16)$$

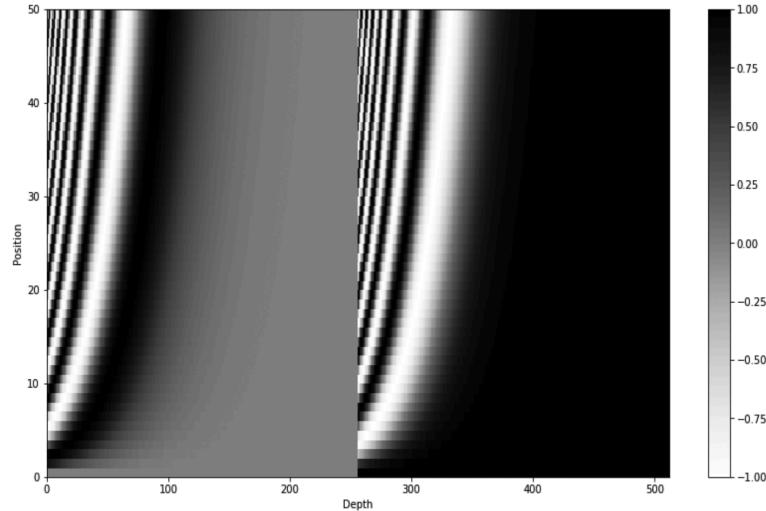


Figure 2.9 Positional encoding for 50 positions with dimensionality 512.

图 2.9 维度为 512 的 50 个位置的位置编码。for $i = 0, \dots, l - 1, j = 0, \dots, \lfloor (d - 1)/2 \rfloor$ 。上述函数定义表明，频率沿矢量维度递减，并在波长上形成从 2π 到 $10000 \cdot 2\pi$ 的几何级数。对于 $d = 512$ 维，最大位置长度 $l = 50$ ，位置编码可视化如图 2.9 所示。

如图 2.7 所示，两个矩阵，即词嵌入 \mathbf{W} 和位置编码 \mathbf{P} 相加，生成输入表示 $\mathbf{X} = \mathbf{W} + \mathbf{P} \in \mathbb{R}^{l \times d}$ 。

2.4.2 注意力层

在编码器和解码器中，Transformer 的基本构建块是自注意力。根据编码器和解码器端的使用方式和位置，它有微小的变化。在接下来的小节中，我们从自注意力开始逐步构建注意力层。

1. 自注意力 为了理解多头注意力，必须分解计算并理解它的单头部分，即自注意力。图 2.10 显示了输入向量 \mathbf{x}_i 如何通过自注意力层转换为

输出向量 \mathbf{z}_i 。每个输入向量 \mathbf{x}_i 生成三个不同的向量：查询、键和值 $(\mathbf{q}_i, \mathbf{k}_i, \mathbf{v}_i)$ 。查询、键和值向量是通过将输入向量 \mathbf{x}_i 投影到可学习权重矩阵 \mathbf{W}_q 、 \mathbf{W}_k 和 \mathbf{W}_v 分别得到 \mathbf{q}_i 、 \mathbf{k}_i 和 \mathbf{v}_i 来获得的。这些查询/键/值权重矩阵是随机初始化的，并且权重是联合学习的培训过程。对于编码器和解码器的第一个注意力层，输入是词嵌入和位置编码的总和。

类似于 2.3 节中的注意力讨论，我们讨论了查询、键和值，以及它们如何影响最终的注意力分数，注意力为每个输入生成所有三个向量，以下是它们的关键作用：

- (a) 标记 i , \mathbf{q}_i 的查询向量的作用是与所有其他键向量 $\sum_{j=0}^l \mathbf{q}_i \mathbf{k}_j^T$ 相结合，以影响其自身输出的权重 \mathbf{z}_i
- (b) 标记 i 、 \mathbf{k}_i 的关键向量的作用是与所有其他查询向量匹配，以获得与查询的相似性，并通过查询关键产品评分影响输出。

3. 标记 i 的值向量 \mathbf{v}_i 的作用是通过结合查询键分数的输出来提取信息以获得输出向量 \mathbf{z}_i 。

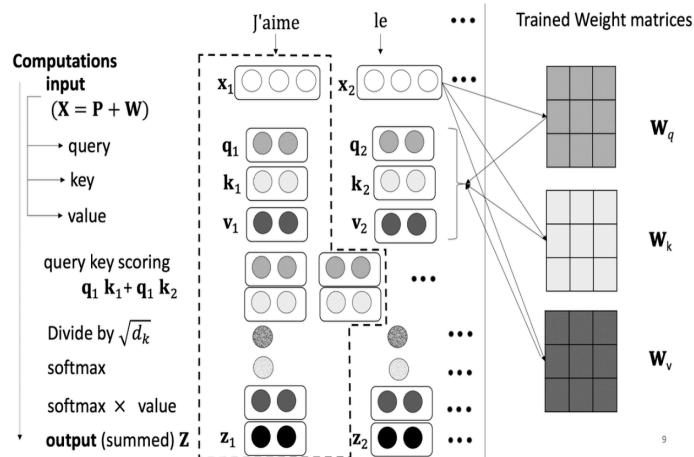


Figure 2.11 The dotted lines show the complete flow of computation for one input through a self-attention layer.

图 2.11 虚线显示了一个输入通过自注意力层的完整计算流程。图 2.11 演示了对每个标记从输入到输出执行的所有计算的逻辑流程。输入矩阵 $\mathbf{X} \in \mathbb{R}^{x \times d}$ (其中 l 是句子的最大长度, d 是输入的维数) 不是对每个标记 i 进行向量计算, 而是与每个查询、键和值矩阵组合给出的单个计算,

$$\text{attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right) \quad (2.17)$$

2. 多头注意力

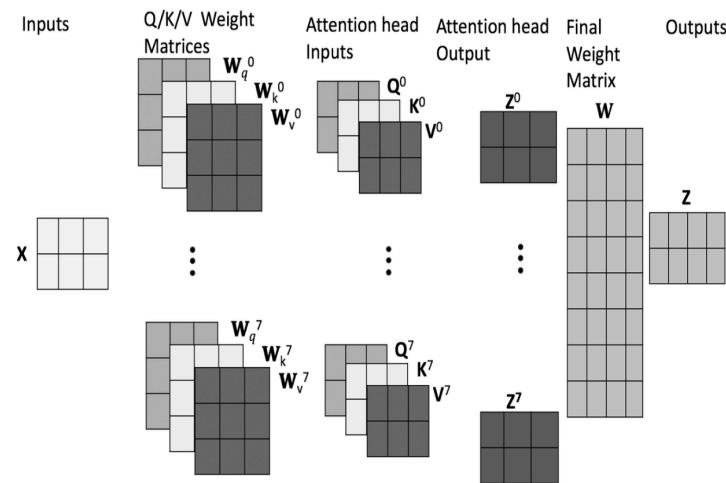


Figure 2.12 Multi-head attention.

可以有 h 并行的自注意力头, 而不是单个自注意力头; 这称为多头注意力。在最初的 transformer 论文中, 作者使用了 $h = 8 \text{ heads}$ 。多头注意力为输入提供不同的子空间表示, 而不仅仅是个表示, 这有助于捕获相同输入的不同方面。它还有助于模型将焦点扩展到不同的位置。每个头脑可以学习不同的东西, 例如, 在机器翻译中, 它可能是关于学习语法、时态、词形变化等。

。多头注意力有多组查询/键/值权重矩阵, 每组都会为输入产生不同的查询/键/值矩阵, 最终生成输出矩阵 \mathbf{z}_i 。这些来自每个头的输出矩

阵被连接并相乘加上一个额外的权重矩阵 \mathbf{W}_O ，以获得单个最终矩阵 \mathbf{Z} ，其中每个输入 \mathbf{x}_i 都有向量 \mathbf{z}_i 作为输出。所有头的并行输入到输出变换如图 2.12 所示。

$$head_i = \text{attention}(\mathbf{W}_q^i \mathbf{Q}, \mathbf{W}_k^i \mathbf{K}, \mathbf{W}_v^i \mathbf{V}) \quad (2.18)$$

$$\text{multihead}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \mathbf{W}_O \text{concat}(head_1, \dots, head_h) \quad (2.19)$$

3. 掩码多头注意力 我们希望解码器从编码器序列和模型已经看到的特定解码器序列中学习，以预测下一个单词/字符。因此，对于解码器的第一层，类似于序列到序列架构，只需要存在先前的目标标记，而其他目标标记需要被屏蔽。这种特殊的改变导致了屏蔽多头注意力。这是通过使用掩蔽权重矩阵 \mathbf{M} 来实现的，该矩阵对于未来的标记具有 $-\infty$ ，对于之前的标记具有 0。该计算被插入到 \mathbf{Q} 和 \mathbf{K}^T 乘法的缩放之后以及 softmax 之前，以便 softmax 得出先前标记的实际缩放值和未来标记的值 0。

$$\text{maskedAttention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^T + \mathbf{M}}{\sqrt{d_k}}\right) \quad (2.20)$$

4. 编码器-解码器多头注意力 在解码器端需要学习给定时间整个源输入和目标输出之间的注意力关系。因此，来自目标序列（给定时间之前）的查询向量以及键和值编码器的整个输入序列被传递到解码器中的自注意力层，如图 2.7 所示。

2.4.3 残差和层归一化

与 ResNet 类似，输入 \mathbf{X} 与输出 \mathbf{Z} 短路，两者相加并通过层归一化 $\text{addAndNorm}(\mathbf{X} + \mathbf{Z})$ [78]。层归一化确保每层具有 0 均值和 $\text{unit}(1)$ 方差。对于每个隐藏单元 h_i ，我们可以计算

$$h_i = \frac{g}{\sigma}(h_i - \mu) \quad (2.21)$$

其中 g 是增益变量 (可以设置为 1), μ 是均值由 $\frac{1}{H} \sum_{i=1}^H h_i$ 给出, σ 是由 $\sqrt{\frac{1}{h}(h_i - \mu)^2}$ 给出的标准差。层归一化减少了协方差偏移, 即每层之间的梯度依赖性, 因此由于需要更少的迭代而加快了收敛速度 [45]。这与批量归一化有关, 其中批量归一化发生在一个隐藏单元级别, 并且在该批次上实现 0 均值和单位 (1) 方差 [79]。层归一化的优点是它的工作与批量大小无关, 即, 可以给出单个示例, 小批量或大批量。

2.4.4 位置前馈网络

编码器和解码器都在注意子层之后包含一个完全连接的前馈网络。对于每个位置, 执行类似的线性变换, 其间有 ReLU 激活。

$$FFN(\mathbf{x}) = \max(0, \mathbf{x}\mathbf{W}_i + b1)\mathbf{W}_2 + b2 \quad (2.22)$$

Transformers: 基础知识和简介 27 每个位置都会经历相同的变换, 仅在层级别有所不同。

2.4.5 编码器

Transformer 中的编码器块由 n 个块 {multi-headAttention, addAndNorm, FFN,addAndNorm } 组成, 如下所示如图 2.7 所示。编码器端的每一层多头注意力都关注输入或源, 即输入和输入之间的注意力。坦尼等人。表明编码器侧的每一层 Transformer 执行经典意义上的不同 NLP 任务, 例如词性、成分、依赖关系、实体解析等。[80]

2.4.6 解码器

transformer 中的解码器块由 {maskedMultiheadAttention, addAndNorm, encoderDecoderAttention, addAndNorm, FFN, addAndNorm } 的 n 个块组成如图 2.7 所示。解码器侧的第一层多头注意力关注目标, 即屏蔽输出与其自身之间的注意力。编码器-解码器注意力层在源和目标之间创建注意力。

2.5 案例研究：机器翻译

2.5.1 目标

本节将介绍 NLP 领域中序列到序列的实际用例——英语和法语句子之间的机器翻译。我们将应用基于注意力和基于 transformer 的技术来理解、比较和对比两种架构。

2.5.2 数据、工具和库

来自 <https://www.manythings.org/anki> 的英法平行语料库用于机器翻译。对于必要的数据整理和可视化，我们使用标准 Python 库，如 Pandas、NumPy 和 Matplotlib。我们使用 TorchText 和 spaCy 进行基本文本处理，例如标记化。基于注意力和基于 Transformer 的序列到序列是使用 PyTorch 库实现的。²⁸

2.5.3 实验、结果和分析

1. 探索性数据分析 基本数据预处理（例如将句子转换为 ASCII 并过滤掉太长的对）作为文本规范化步骤执行。然后我们对句子进行标记，将单词转换为标记 ID，并将 `<bos>` 和 `<eos>` ID 附加到标记 ID 序列的开头和结尾。使用 `<pad>` 标记将可变长度序列填充到批次中的最大观察长度，确保了用于训练和评估的固定大小张量。

过滤后总共 135,842 个语言对减少到 131,951 个，我们进一步将其拆分为 80% 的训练、10% 验证数据和 10% 测试数据，即分别为 105,460、13,308 和 13,183。图 2.13 和图 2.14 将英语/法语和联合分布的分布图显示为直方图。平行语料库中的大多数句子长度在 4 到 8 个标记/单词之间。

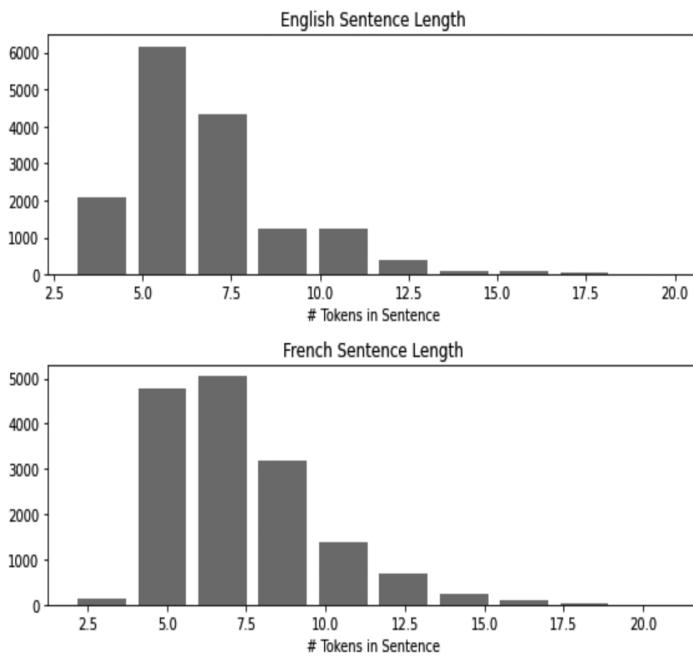


Figure 2.13 Sentence length distribution for English and French sentences.

图 2.13 英语和法语句子的句子长度分布。

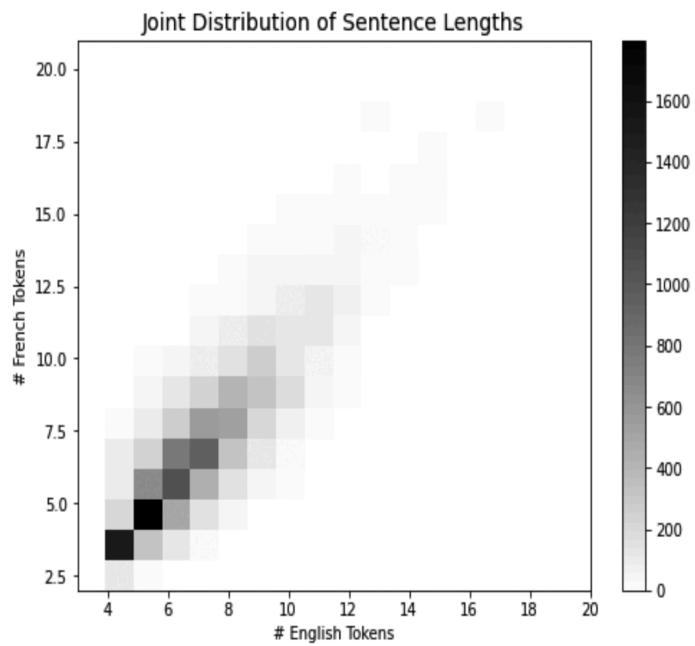


Figure 2.14 Joint distribution for English and French sentences based on length.

图 2.14 基于长度的英语和法语句子的联合分布。

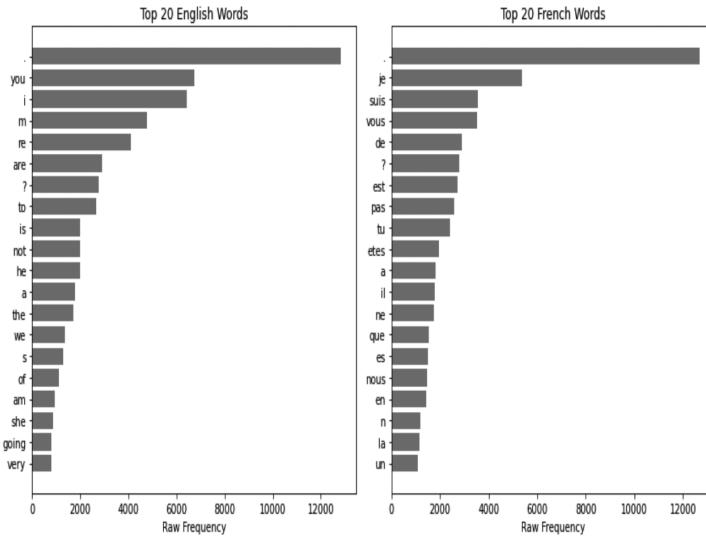


Figure 2.15 Top 20 words from English and French sentences.

图 2.15 英语和法语句子中的前 20 个单词。

英语和法语的前 20 个单词如图 2.15 所示。频率和分布显示了一些预期的常见单词，例如“the-le”、“is-est”等。它还突出显示了某些缩写，例如“don”代表“don’t”，以及底层分词器 (spaCy) 如何执行预处理。迭代地改进和分析数据有助于整体模型结果。

基于 Bahdanau 注意的序列到序列和基于 transformer 的将用于对数据进行训练/验证，并且将在测试集上评估两者最佳模型。

2. 注意力

```
class BahdanauEncoder(nn.Module):
    def __init__(self, input_dim, embedding_dim,
                 encoder_hidden_dim,
                 decoder_hidden_dim, dropout_p):
        super().__init__()
```

```

        self.input_dim = input_dim
        self.embedding_dim = embedding_dim
        self.encoder_hidden_dim = encoder_hidden_dim
        self.decoder_hidden_dim = decoder_hidden_dim
        self.dropout_p = dropout_p

        self.embedding = nn.Embedding(input_dim, embedding_dim)
        self.gru = nn.GRU(embedding_dim, encoder_hidden_dim,
                         bidirectional=True)
        self.linear = nn.Linear(encoder_hidden_dim * 2,
                               decoder_hidden_dim)
        self.dropout = nn.Dropout(dropout_p)

    def forward(self, x):
        embedded = self.dropout(self.embedding(x))
        outputs, hidden = self.gru(embedded)
        hidden = torch.tanh(self.linear(
            torch.cat((hidden[-2, :, :], hidden[-1, :, :]),
                      dim=1)))
        return outputs, hidden

```

清单 2.1 中给出的编码器模型类 **BahdanauEncoder** 使用双向门控循环单元 (GRU) 对源语言中的句子进行编码。2.1 Bahdanau 编码器

```

class BahdanauAttentionQKV(nn.Module):
    def __init__(self, hidden_size, query_size=None,
                 key_size=None, dropout_p=0.15):
        super().__init__()

```

```

        self.hidden_size = hidden_size
        self.query_size = hidden_size if query_size is None else query_size
        # assume bidirectional encoder, but can specify otherwise
        self.key_size = 2*hidden_size if key_size is None else key_size
        self.query_layer = nn.Linear(self.query_size, hidden_size)
        self.key_layer = nn.Linear(self.key_size, hidden_size)
        self.energy_layer = nn.Linear(hidden_size, 1)
        self.dropout = nn.Dropout(dropout_p)

    def forward(self, hidden, encoder_outputs, src_mask=None):
        # (B, H)
        query_out = self.query_layer(hidden)
        # (Src, B, 2*H) --> (Src, B, H)
        key_out = self.key_layer(encoder_outputs)
        # (B, H) + (Src, B, H) = (Src, B, H)
        energy_input = torch.tanh(query_out + key_out)
        # (Src, B, H) --> (Src, B, 1) --> (Src, B)
        energies = self.energy_layer(energy_input).squeeze(2)
        # if a mask is provided, remove masked tokens from
        softmax calc
        if src_mask is not None:
            energies.data.masked_fill_(src_mask == 0, float("-inf"))
        # softmax over the length dimension
        weights = F.softmax(energies, dim=0)
        # return as (B, Src) as expected by later
        multiplication
        return weights.transpose(0, 1)

```

清单 2.2 中给出的编码器模型类 BahdanauAttentionQKV 使用查询和关键张量计算注意力权重。

解码器模型类 BahdanauDecoder 如清单 2.3 所示，使用隐藏层、编码器输出和注意力权重来生成下一个标记。

```
class BahdanauDecoder(nn.Module):
    def __init__(self, output_dim, embedding_dim,
                 encoder_hidden_dim, decoder_hidden_dim,
                 attention, dropout_p):
        super().__init__()
        self.embedding_dim = embedding_dim
        self.output_dim = output_dim
        self.encoder_hidden_dim = encoder_hidden_dim
        self.decoder_hidden_dim = decoder_hidden_dim
        self.dropout_p = dropout_p
        self.embedding = nn.Embedding(output_dim, embedding_dim)
        self.attention = attention # allowing for custom attention
        self.gru = nn.GRU((encoder_hidden_dim * 2) +
                          embedding_dim, decoder_hidden_dim)
        self.out = nn.Linear((encoder_hidden_dim * 2) +
                           embedding_dim + decoder_hidden_dim, output_dim)
        self.dropout = nn.Dropout(dropout_p)

    def forward(self, input, hidden, encoder_outputs,
               src_mask=None):
        # (B) --> (1, B)
        input = input.unsqueeze(0)
        embedded = self.dropout(self.embedding(input))
        attentions = self.attention(hidden, encoder_outputs,
                                    src_mask)
        # (B, S) --> (B, 1, S)
        a = attentions.unsqueeze(1)
        # (S, B, 2*Enc) --> (B, S, 2*Enc)
```

```

encoder_outputs = encoder_outputs.transpose(0, 1)
# weighted encoder representation
# (B, 1, S) @ (B, S, 2*Enc) = (B, 1, 2*Enc)
weighted = torch.bmm(a, encoder_outputs)
# (B, 1, 2*Enc) --> (1, B, 2*Enc)
weighted = weighted.transpose(0, 1)
# concat (1, B, Emb) and (1, B, 2*Enc)
# results in (1, B, Emb + 2*Enc)
rnn_input = torch.cat((embedded, weighted), dim=2)
output, hidden = self.gru(rnn_input, hidden.unsqueeze(0))
assert (output == hidden).all()
# get rid of empty leading dimensions
embedded = embedded.squeeze(0)
output = output.squeeze(0)
weighted = weighted.squeeze(0)
# concatenate the pieces above
# (B, Dec), (B, 2*Enc), and (B, Emb)
# result is (B, Dec + 2*Enc + Emb)
linear_input = torch.cat((output, weighted, embedded),
dim=1)
# (B, Dec + 2*Enc + Emb) --> (B, 0)
output = self.out(linear_input)
return output, hidden.squeeze(0), attentions

```

清单 2.3 Bahdanau 解码器

```

enc = BahdanauEncoder(input_dim=len(en_vocab),
embedding_dim=ENCODER_EMBEDDING_DIM,
ncoder_hidden_dim=ENCODER_HIDDEN_SIZE,

```

```

decoder_hidden_dim=DECODER_HIDDEN_SIZE,
opout_p=0.15)

attn = BahdanauAttentionQKV(DECODER_HIDDEN_SIZE)
dec = BahdanauDecoder(output_dim=len(fr_vocab),
embedding_dim=DECODER_EMBEDDING_DIM,
encoder_hidden_dim=ENCODER_HIDDEN_SIZE,
decoder_hidden_dim=DECODER_HIDDEN_SIZE,
attention=attn, dropout_p=0.15)
seq2seq = BahdanauSeq2Seq(enc, dec, device)

```

清单 2.4 给出了序列到序列的整体结构。编码器和解码器的嵌入固定尺寸为 256，隐藏层大小为 256。正则化是通过为编码器和解码器添加 0.15 的 dropout 来完成的，并且梯度范数被裁剪为 10。

清单 2.5 显示了使用编码器和解码器进行的模型训练 Adam 的 AdamW 变体是一种改进，其中权重衰减与优化分离 [81]。编码器和解码器的学习率均手动网格搜索并固定为 0.0001。

```

enc_optim = torch.optim.AdamW(seq2seq.encoder.parameters(), lr=1e-4)
dec_optim = torch.optim.AdamW(seq2seq.decoder.parameters(), lr=1e-4)
optims = MultipleOptimizer(enc_optim, dec_optim)
best_valid_loss = float("inf")
for epoch in tqdm(range(N_EPOCHS), leave=False, desc="Epoch"):
    train_loss = train(seq2seq, train_iter, optims, loss_fn, device, clip=CLIP)
    valid_loss = evaluate(seq2seq, valid_iter, loss_fn, device)
    if valid_loss < best_valid_loss:
        best_valid_loss = valid_loss
    torch.save(seq2seq.state_dict(), model_path)

```

清单 2.5 Bahdanau 基于注意力的网络的训练

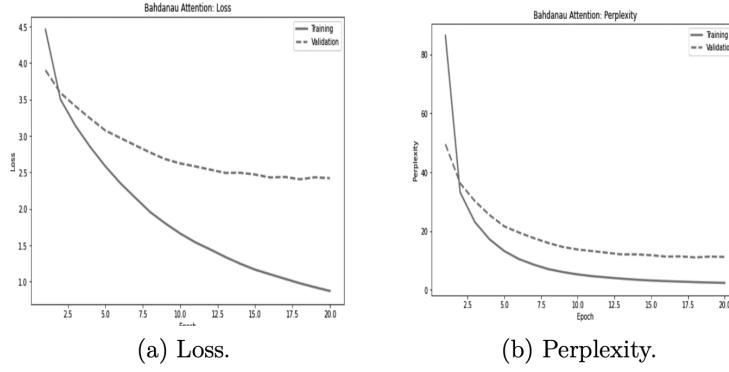
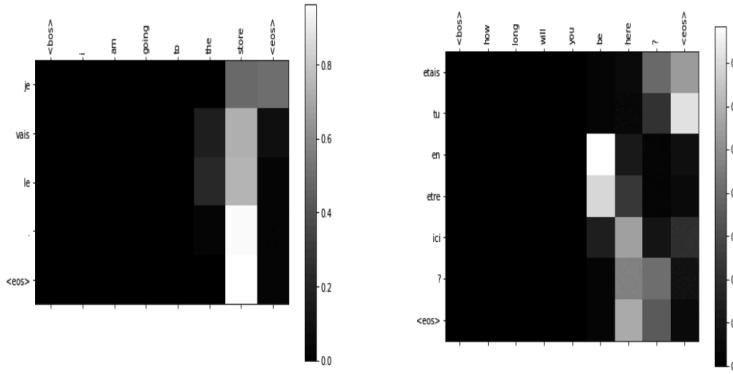


Figure 2.16 Attention-based seq2seq loss and perplexity on training and validation sets.

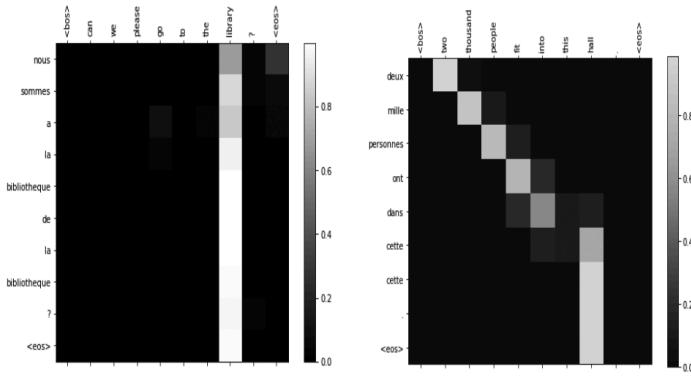
图 2.16 基于注意力的 seq2seq 损失和训练和验证集的困惑。

困惑度是一种内在的评估方法，通常用于测量语言建模和机器翻译等任务的性能 [82]。在图 2.16 中，我们展示了注意力模型的困惑度和训练/验证损失图，在训练过程中每个时期进行测量。有趣的是，验证损失在第 14 轮左右开始变平，并且训练损失进一步减少，从而表明过度拟合。两种方法的最佳模型都是根据验证数据的最佳拟合来选择的。



(a) Example showing the English word “going” pays attention to “je” and “vais”.

(b) Example showing the English word “be” pays attention to “en” and “etre”.



(c) English word “library” pays attention to multiple words.

(d) The words “two” and “deux” are matched.

Figure 2.17 Attention examples and plots.

为了了解注意力机制在翻译过程中如何运作，我们绘制了一些解码注意力输出的示例，突出显示解码器正在关注的位置，如图所示 2.17。输出有助于可视化和诊断数据和模型中的问题。例如，图 2.17(a) 显示了英语单词“going”如何关注“je”和“vais”，类似地“store”单词如何关注“au”、“magasin”、“.”。和“<eos>”。

3. Transformer

```
class TransformerModel(nn.Module):
    def __init__(self, input_dim, output_dim, d_model,
                 num_attention_heads, num_encoder_layers,
                 num_decoder_layers, dim_feedforward,
                 max_seq_length, pos_dropout, transformer_dropout):
        super().__init__()
        self.d_model = d_model
        self.embed_src = nn.Embedding(input_dim, d_model)
        self.embed_tgt = nn.Embedding(output_dim, d_model)
        self.pos_enc = PositionalEncoding(d_model,
                                         pos_dropout, max_seq_length)
        self.transformer = nn.Transformer(d_model,
                                         num_attention_heads, num_encoder_layers,
                                         num_decoder_layers, dim_feedforward,
                                         transformer_dropout)
        self.output = nn.Linear(d_model, output_dim)
    def forward(self, src, tgt, src_mask=None,
               tgt_mask=None, src_key_padding_mask=None,
               tgt_key_padding_mask=None, memory_key_padding_mask=None):
        src_embedded = self.embed_src(src) * np.sqrt(self.d_model)
        tgt_embedded = self.embed_tgt(tgt) * np.sqrt(self.d_model)
        src_embedded = self.pos_enc(src_embedded)
        tgt_embedded = self.pos_enc(tgt_embedded)
        output = self.transformer(src_embedded,
                                 tgt_embedded, tgt_mask=tgt_mask,
                                 src_key_padding_mask=src_key_padding_mask,
                                 tgt_key_padding_mask=tgt_key_padding_mask,
                                 memory_key_padding_mask=memory_key_padding_mask)
        return self.output(output)
```

清单 2.6 显示了包装 PyTorch trans-former 块的 transformer 模型。

```
transformer = TransformerModel(input_dim=len(en_vocab),  
                               output_dim=len(fr_vocab), d_model=256,  
                               num_attention_heads=8, num_encoder_layers=6,  
                               num_decoder_layers=6, dff=2048,  
                               max_seq_length=32, pos_dropout=0.15,  
                               transformer_dropout=0.3)  
transformer = transformer.to(device)
```

清单 2.7 带有 attention 的 Transformer 模型头、编码和解码

4. 结果与分析

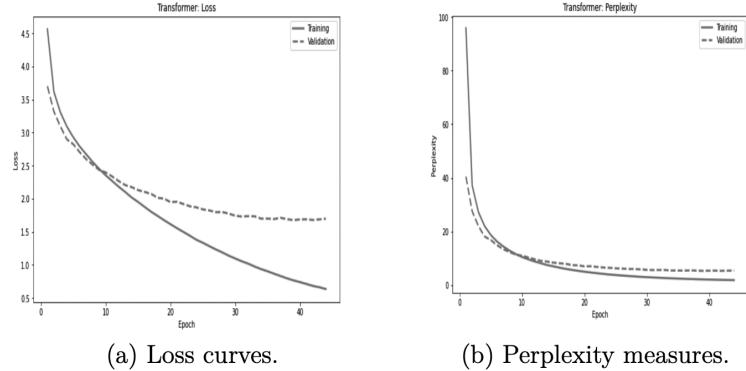


Figure 2.18 Transformer loss and perplexity on training and validation sets.

图 2.18 是训练过程中每轮测量的 Transformer 模型的困惑度和训练/验证损失图。验证损失在第 20 轮中的值小于 2，而注意力机制中的值约为 2.5。此外，注意力的困惑度几乎是验证集中 Transformer 模型的两倍。(a) 损失曲线。(b) 困惑度测量。图 2.18 训练和验证集上的 transformer 损耗和困惑度。2.19 显示了基于注意力的模型和基于

转换器的模型在同一测试数据集上的性能比较。变换器的困惑度几乎是注意力的三倍，证明了该架构在现实世界翻译问题中的好处。

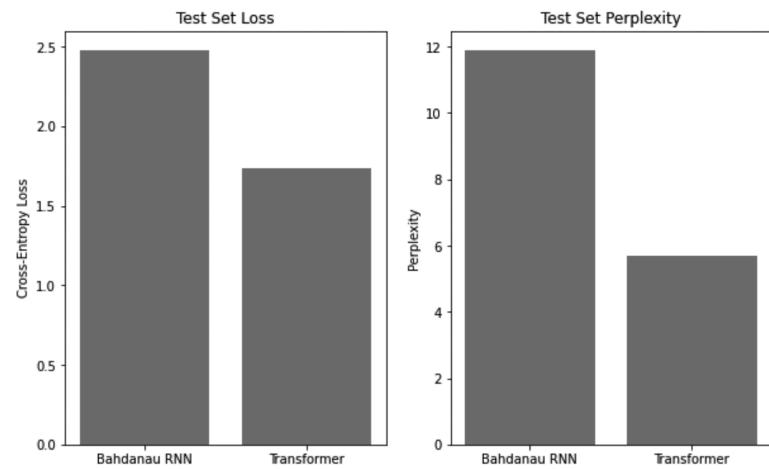


Figure 2.19 Loss and perplexity on the test set.

5. 可解释性

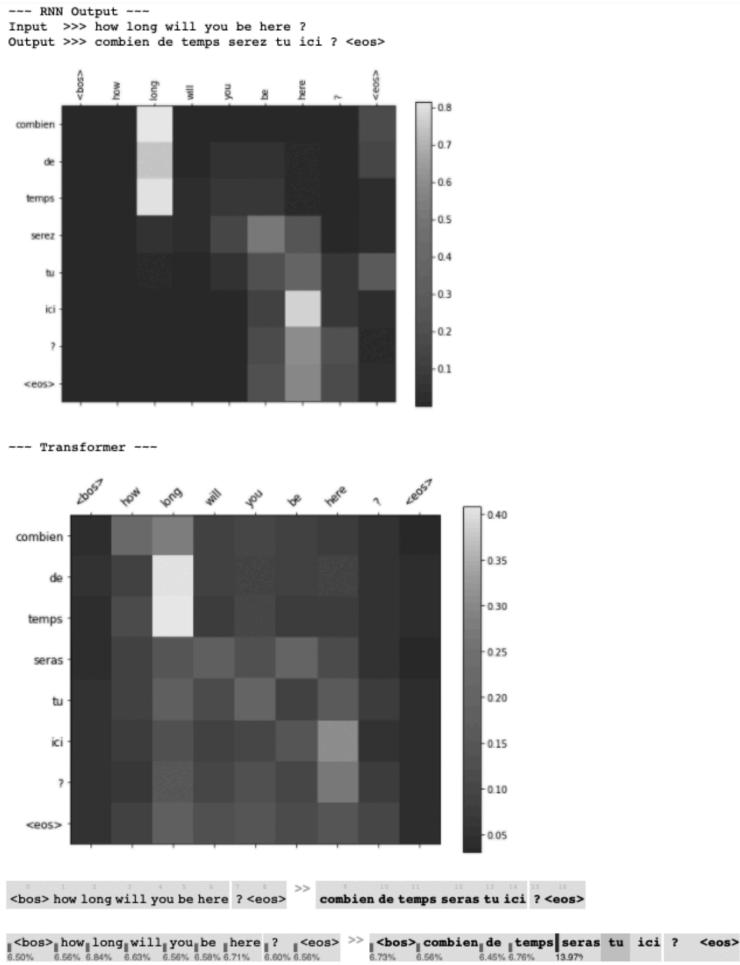


Figure 2.20 Explaining translations for—How long you will be here?

图 2.20 解释翻译——您将在这里呆多久？

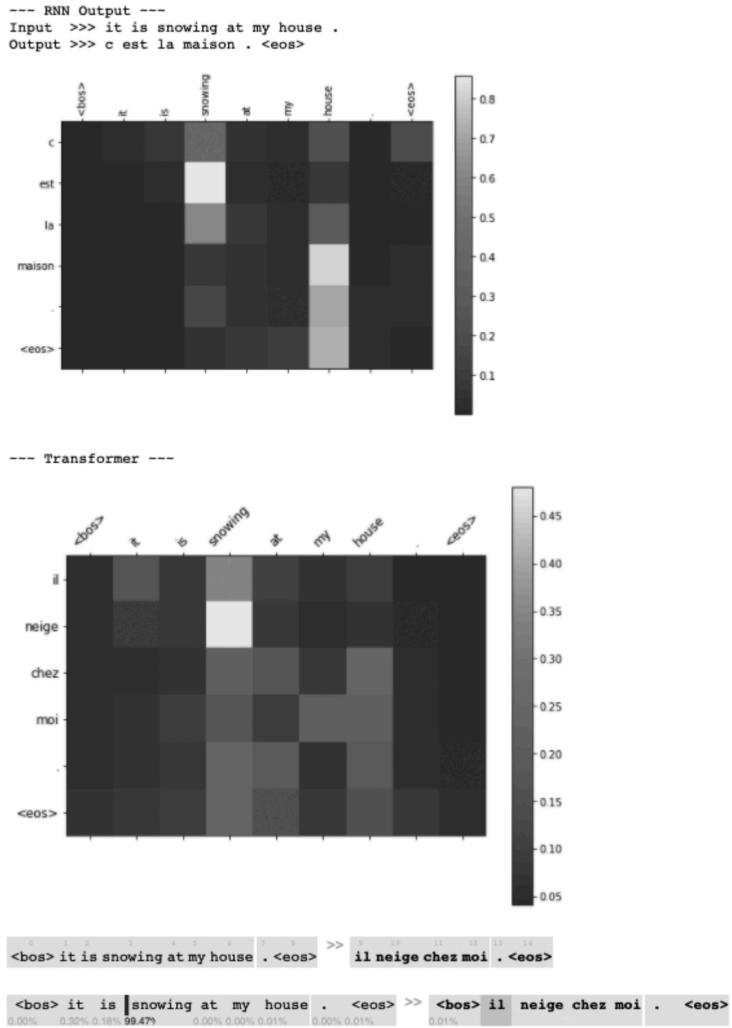


Figure 2.21 Explaining translations for—It is snowing at my house.

图 2.21 解释翻译——我家正在下雪。

我们可以使用输入序列的梯度值来说明每个生成的标记对每个输入标记的依赖性。我们首先对嵌入式输入执行前向传递。然后，我们获取具有最高 Logit 值的标记（与之前的贪婪解码方式相同），并从最高 Logit 值执行向后传递。这会将梯度通过模型填充回嵌入输入，显示最终的分布。最后，我们对每个生成的标记重复此过程并可视化结果矩

阵。2.20 显示了 RNN 与 Transformer 的翻译结果，可以看出 RNN 将形式/复数“serez”与非正式/单数“tu”配对，而 Transformer 匹配“seras tu”。

2.21 显示了另一个 RNN 与 Transformer 的翻译对比，可以看出 RNN 没有捕获句子的“下雪”部分，并产生了“我的房子”的不同措辞。

3 双向编码器表示

Transformer 的双向编码器表示 (BERT) [26] 的出现被认为是自然语言处理 (NLP) 领域革命的开始。BERT 使用未标记的文本来预训练深度双向上下文表示。这产生了丰富的预训练语言模型，可以通过简单的附加输出层和合理大小的标记数据集进行微调，以在广泛的 NLP 任务中产生最先进的性能。这些发展降低了广泛采用这些强大的预训练模型的准入门槛。现在，人工智能领域的常见做法是共享预先训练的模型并以最小的成本对其进行微调，这与设计特定任务架构的旧模式相反。在本章中，我们将介绍 BERT 的基本设计概念、发展和应用。

3.1 BERT

3.1.1 架构

核心层 BERT 的主要贡献之一是设计的简单性和所涵盖的下游任务的多样性。BERT 的架构由多层双向 Transformer 编码器组成 [44]。BERT 架构的容量特征为 (i) transformer 层数 L ，(ii) 隐藏表示的大小 H ，以及双向自注意力头的数量 A 。

输入和输出表示 鉴于 BERT 核心架构的简单性，只需一个双向 Transformer 编码器的堆栈，BERT 的独创性在于输入和输出表示的设计，可用于训练具有相同架构的许多下游 NLP 任务。为此，BERT 的输入被设计为使用相同的输入表示设计来明确表示涉及单个或一对句子的 NLP 下游任务。大多数主要的 NLP 下游任务可以使用单个句子（例如，文本分类、序列标记、摘要等）或成对句子（例如，问答、自然语言推理等）来涵盖。

对于任何输入序列，BERT 都会前缀一个特殊的 [CLS] 标记。最后一个 BERT 层中该 token 的隐藏向量将用作整个输入序列的聚合表示，通常用于分类任务。对于具有配对句子的 NLP 任务，BERT 将句子连接成一个序列，中间有一个分隔符标记 [SEP]，这是 BERT 用来区分两个句子的一种方式。BERT 还使用学习的分段嵌入来指示标记属于哪个句子。BERT 使用 WordPiece [83] 对输入序列进行标记和嵌入。最后，BERT 使用位置嵌入层对输入中标记的顺序进行编码。BERT 的输入是上述每个 token 的 token 嵌入、分段嵌入和位置嵌入的总和，如图 3.1 所示。

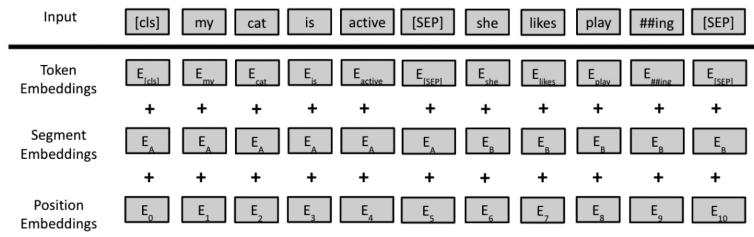


Figure 3.1 BERT input configuration [71].

图 3.1 BERT 输入配置 [26]。

3.1.2 预训练

Masked Language Model (MLM) 这个想法是随机屏蔽一定比例的输入序列标记，用特殊的 [MASK] 标记替换它们。在预训练期间，修改后的输入序列通过 BERT 运行，然后将屏蔽标记的输出表示馈送到 WordPiece 词汇表上的 softmax 层，如标准语言模型预训练中所实践的那样。Transformer 编码器的双向注意力迫使 [MASK] 预测任务使用序列中其他非屏蔽标记提供的上下文。BERT 以 15% 的屏蔽率进行预训练。这个简单的 MLM 任务有一个缺点，即预训练和微调任务之间会产生不匹配，因为特殊的 [MASK] 标记在微调期间不会出现。为了克服这个问题，15% 的被屏蔽掉的标记中的每个标记都受到以下启发：

- 以 80% 的概率，该 token 被替换为特殊的 [MASK] token。

- 以 10% 的概率, token 被随机 token 替换。
- 以 10% 的概率, token 保持不变。

偶尔插入随机 token (即噪声) 会促使 BERT 减少对屏蔽的 token 的偏见, 尤其是当 maskedtoken 保持不变时, 在其双向上下文注意力中。MLM 任务仅在屏蔽标记上使用交叉熵损失, 并忽略所有非屏蔽标记的预测。

Next Sentence Prediction (NSP) 许多下游 NLP 任务需要理解两个句子之间的关系, 例如问答 (QA) 和自然语言推理 (自然语言学)。标准语言模型无法获取此类知识。这激发了 NSP 任务, 其中 BERT 被输入句子对并进行预训练, 以预测第二个句子是否应该在连续上下文中跟随第一个句子。如前所述, 第一个句子以 [CLS] 标记为前缀, 然后两个句子由特殊标记 [SEP] 分隔。在 NSP 任务预训练期间, 模型会得到句子对, 其中 50% 的情况下第二个句子出现在第一个句子之后, 另外 50% 的情况下第二个句子是来自完整训练语料库的随机句子。Transformer 层的自注意力鼓励 [CLS] token 的 BERT 表示对两个输入句子进行编码。因此, NSP 预训练是通过在 [CLS] token 表示之上添加带有 softmax 的单层 MLP 来进行预测二进制 NSP 标签。BERT 预训练涉及通过在组合损失函数上优化模型参数来与 MLM 和 NSP 任务进行组合训练。图 3.2 说明了 BERT 的预训练任务。

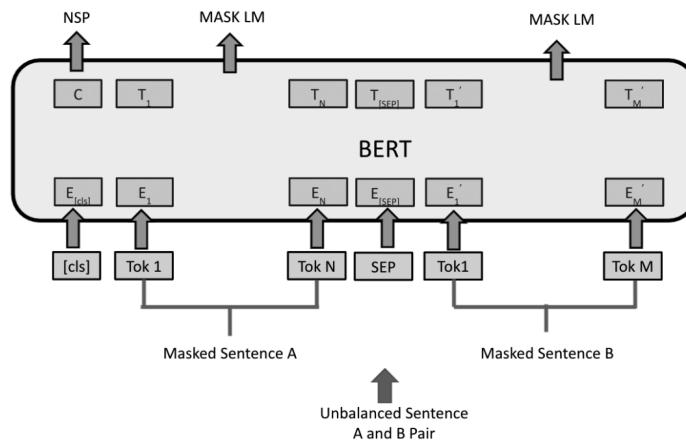


Figure 3.2 BERT pre-training architecture [71].

图 3.2 BERT 预训练架构 [26]。

3.1.3 微调

通过 BERT 简单而通用的设计，为许多下游任务微调预训练模型是一项简单的任务，只需插入正确的输入和输出并端到端地微调模型的所有参数。正如前面所讨论的，许多 NLP 任务可以使用输入到 BERT 的两个句子来建模。对于问答 (QA) 和自然语言推理 (NLI) 等任务，BERT 的输入分别是连接的问答句子对和假设前提句子对。其他使用单输入句子的任务，例如文本分类和序列标记（例如，POS、NER），BERT 的输入再次被建模为连接的句子，但其中一个为空。对于 token 级别的任务（例如 QA、POS 和 NER），BERT 输出 token 表示被馈送到输出层以进行 token 标记。图 3.3 提供了针对 token 级别任务的 BERT 微调的说明。对于文本分类和 NLI 等任务，使用特殊标记 [CLS] 的输出表示作为输出层的输入以生成标签。图 3.4 和 3.5 描述了单序列和配对序列分类任务的 BERT 微调架构

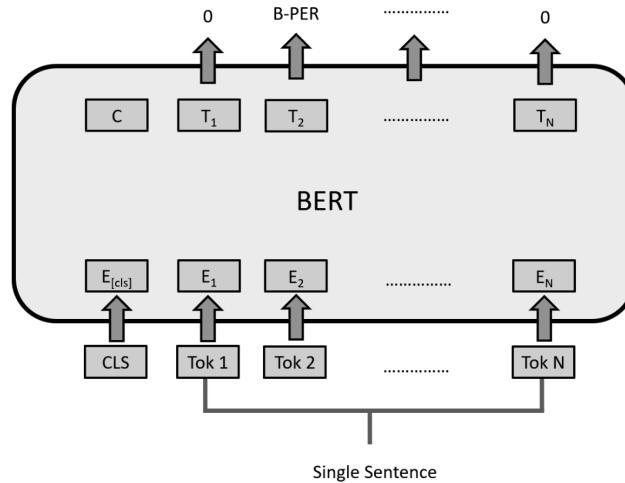


Figure 3.3 BERT fine-tuning architecture for token tagging tasks (e.g., POS, NER, and QA) [71].

图 3.3 用于标记标记任务（例如 POS、NER 和 QA）的 BERT 微调架构 [26]。

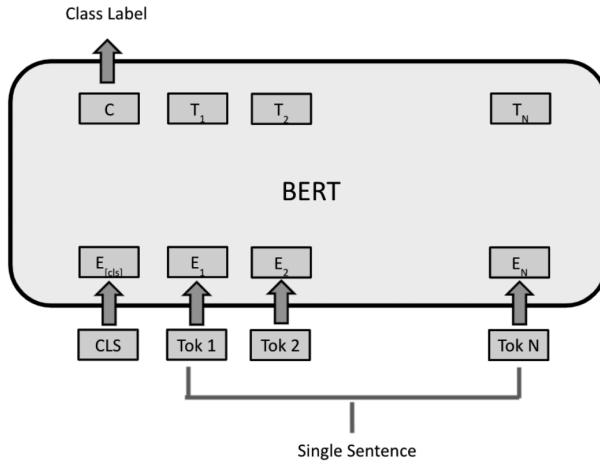


Figure 3.4 BERT fine-tuning architecture for single sequence classification tasks [71].

图 3.4 单序列分类任务的 BERT 微调架构 [26]。

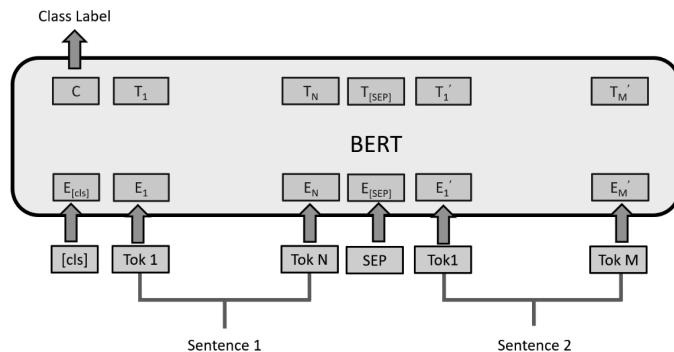


Figure 3.5 BERT fine-tuning architecture for paired-sequence classification tasks (e.g., NLI) [71].

图 3.5 用于配对序列分类任务（例如 NLI）的 BERT 微调架构 [26]。

3.2 BERT 变体

3.2.1 RoBERTa BERT 预训练方法 (RoBERTa)

RoBERTa [84] 是一种鲁棒优化的 BERT 预训练方法，在预训练任务、训练超参数和预训练数据方面改进了原始 BERT 模型。如前所述，BERT 使用两个预训练任务：Masked Language Model (MLM) 和 Next Sentence Prediction (NSP)。对于 MLM 任务，BERT 在数据预处理阶段随机屏蔽 token。因此，掩模在整个模型训练过程中保持静态。另一方面，RoBERTa 遵循动态屏蔽策略，其中为每个训练时期随机选择屏蔽标记。RoBERTa 还放弃了 NSP 预训练任务，仅使用动态 MLM 任务。RoBERTa 的另一个贡献是输入序列的大小，其中最多 512 个标记的完整句子是从一个或多个文档中连续采样的。如果在获得所需的输入序列大小之前到达文档末尾，则在添加额外的分隔符标记后，句子采样会继续下一个文档。

在超参数方面，RoBERTa 表明，在预训练期间使用具有更高学习率的大型小批量可以改善困惑度动态 MLM 任务以及下游任务绩效。RoBERTa 还对比 BERT 更多数量级的数据进行了预训练，预训练数据由 160 GB 组成，包括图书语料库、英语维基百科和 CommonCrawl 新闻数据集 Transformers 的双向编码器表示 (BERT) 49 (6300 万篇文章，76 GB)、Web 文本语料库 (38 GB) 和 Common Crawl 的故事 (31 GB)。

总而言之，使用与 BERT 相同的基本架构，RoBERTa 表明，以下设计更改显着提高了 RoBERTa 相对于 BERT 的性能。

1. 对模型进行更长时间的预训练 2. 使用更大的批次 3. 使用更多的训练数据 4. 删除 NSP 预训练任务 5. 较长序列的训练 6. 动态改变应用于训练数据的掩蔽模式。

3.3 应用

3.3.1 TaBERT

aBERT [85] 是第一个在自然语言句子和表格数据格式上进行预训练的模型。这些表示对于涉及自然语言句子和表格的合作推理的问题是有利的。数据库语义解析是一个示例，其中自然语言查询（例如，“哪个国家拥有最大

的 GDP? ”) 被翻译为可在数据库 (DB) 表上执行的程序。这是第一种跨越结构化和非结构化领域的预训练技术，它为语义解析开辟了新的可能性，其中主要问题之一是理解数据库表的结构以及它如何与查询相匹配。TaBERT 在包含 2600 万张表格的语料库及其随附的英语短语上进行了训练。从历史上看，语言模型仅在自由形式的自然语言文本上进行训练。虽然这些模型非常适合仅需要使用自由格式自然语言进行推理的工作，但它们对于基于数据库的问答等活动来说是不够的，这些活动涉及自由格式语言和数据库表的推理。两个常用的数据库基准 - 利用了基于问题的回答：传统的监督文本到 SQL 作业对来自 Spider 数据集的数据进行过度结构化，以及对来自 WikiTableQuestions 数据集的半结构化数据集进行弱监督解析任务。弱监督学习比监督学习要困难得多，因为解析器无法访问标记的问题，而必须探索非常巨大的查询搜索空间。TaBERT 是一个建立在 BERT 模型之上的自然语言处理 (NLP) 模型，接受自然语言查询和表格作为输入。TaBERT 以这种方式获取句子的上下文表示以及 DB 表的组成部分。然后可以利用这些表示在其他神经网络中生成真实的数据库指令。然后，可以使用该作业的训练数据进一步微调 TaBERT 的表示。

TaBERT 通过内容快照进行训练，其中模型仅对表中与查询最相关的位进行编码。由于某些数据库表中的行数巨大，对它们进行编码是一个计算密集型且低效的过程。通过仅编码与话语最相关的材料部分，内容快照使 TaBERT 能够处理巨大的表。例如，“皮奥特上次在哪个城市获得第一名？”这句话。(从 WikiTableQuestions 数据集中提取的样本) 可能会附有一个表格，其中包括有关年份、地点、位置和事件的信息。内容快照将采用三行的子集。该子集不会包括表的所有数据，但模型足以理解，例如，场地列包含城市。TaBERT 使用传统的水平自注意力 (捕获特定行中的单元格之间的依赖关系) 和垂直自注意力 (捕获不同行中的单元格之间的信息流) 的组合来描述表结构。这种水平和垂直自注意力层的最终输出是话语标记和表列的分布式表示，可以利用下游语义解析器来计算数据库查询。

3.3.2 BERTopic

主题建模是 NLP 中具有挑战性的主题之一。BERT 及其变体的进步促使 NLP 社区在主题建模中利用 BERT。Grootendorst [86] 引入了一个优雅的想法，BERTopic，使用 BERT 和转换器嵌入来提取文档中的可解释主题。因此，利用了最先进的预训练 Transformer 模型。该方法首先使用 BERT 模型创建感兴趣文档的嵌入。由于文档可能非常大，而 BERT 模型有限制标记大小，在提取嵌入之前对文档进行预处理。预处理将文档划分为比 Transformer 模型的标记大小更小的段落或句子。然后，对文档嵌入进行聚类，将具有相似主题的所有文档聚类在一起。值得注意的是，建议在执行聚类之前对嵌入进行降维。诸如 t 分布随机邻域嵌入 (t-SNE) 或统一流形逼近和投影 (UMAP) 之类的降维算法可用于降低维度。该主题是使用 Grootendorst [86] 引入的新颖的基于类的度量从聚类文档中衍生出来的。基于类的词频-逆文档频率 (c-TF-IDF) 是经典词频-逆文档频率 (TF-IDF) 的修改，其中某个类别中的所有文档被视为单个文档，然后计算 TF-IDF。因此，这种情况下的 TF-IDF 将代表主题中单词的重要性，而不是文档中的重要性。基于类的词频-逆文档频率 (c-TF-IDF) 计算如下：

$$c - TF - IDF_i = \frac{t_i}{w_i} \times \log \frac{m}{\sum_j^n t_j} \quad (3.1)$$

其中为每个类提取每个单词的频率并通过单词总数进行归一化这是根据所有类别 n 中单词 t 的总频率的归一化文档数进行加权的。换句话说，(3.1) 中的度量不是计算文档中单词与其他文档相比的相对重要性，而是计算类别（即主题）中单词与其他主题相比的相对重要性。

3.4 BERT 见解

3.4.1 BERT 句子表示

由于 Transformer 编码器中的自注意力，特殊标记 [CLS] 的 BERT 表示对输入中的两个句子进行编码。因此，MLP 分类器的输出层将 X 作为输入，其中 X 是 MLP 隐藏层的输出，其输入是编码的 [CLS] 标记。因为预训练的模型尚未针对任何下游任务进行微调。在这种情况下，[CLS] 的隐

藏状态不是一个好的句子表示。如果稍后您对模型进行微调，您也可以使用 [CLS]。

BERT 句子表示也可以通过合并各个 token 表示来获得。然而，最后一层在预训练过程中与目标函数（即 Masked Language Model 和 Next Sentence Prediction）太接近，因此可能会对这些目标产生偏差。

3.4.2 BERTology

BERT 对 NLP 领域的巨大推动引发了许多研究了解它的工作原理以及它通过大量预训练提取的知识类型。BERTology 旨在回答一些关于为什么 BERT 在如此多的 NLP 任务上表现良好的问题。BERTology 解决的一些主题包括 BERT 学习的知识类型及其代表的位置。一般来说，BERT 获取三种类型的知识：句法知识、语义知识、世界知识。

BERT 句法知识的表示是分层的而不是线性的，即除了词序信息之外，它们还包括句法树结构。此外，BERT 嵌入存储有关语音片段、语法块和角色的信息。另一方面，BERT 对语法的理解是不完整的，因为探测分类器无法检索语法树中遥远父节点的标签。就句法信息的表示方式而言，自注意力权重似乎并不直接编码句法结构，但它们可能会被改变以反映它。在执行完形填空任务时，BERT 会考虑主谓一致性。此外，研究还表明 BERT 不能“理解”否定，并且对错误的输入并不关心。

在语义知识方面，BERT 已经展示了一些语义角色的知识。BERT 还对有关实体类型、关系、语义角色和原型角色的信息进行编码。然而，人们发现 BERT 在数字表示方面遇到了困难。对 BERT 世界知识能力的研究表明，对于某些关系类型，普通 BERT 与依赖知识库的方法相比具有竞争力。然而，BERT 无法根据其世界知识进行推理。本质上，BERT 可以“猜测”许多物体的可供性和属性，但没有有关它们交互的信息（例如，它“知道”人们可以走进房子，房子很大，但它不能推断房子是比人更大。）

此外，BERTology 还关注 BERT 架构内语言信息的本地化，无论是在自注意力头或层级别的双向编码器表示。结果表明，大多数自注意力头不会直接编码任何重要的语言信息，因为只有不到一半的自注意力头表现出“异质”模式 2。垂直模式存储在模型的很大一部分中（注意 [CLS]、[SEP] 和标

点符号)。此外，某些 BERT 头似乎专门研究特定类型的句法关系，头更关注特定句法位置的单词，而不是随机基线。其他研究发现没有一个头部包含整个句法树。此外，注意力权重可以说明主语一致和反身照应。此外，研究表明，即使注意力头专门监视语义关系，它们也并不总能帮助 BERT 在相关任务上表现良好。

对于层级知识定位，假设 BERT 第一层得到以下形式的表示：标记、段和位置嵌入的混合作为输入。它的原因是底层包含有关词序的最线性信息。结果表明，线性词序知识在 BERT 基础的第 4 层周围减少。接下来是对句子层次结构的加深理解。大量研究表明，middleBERT 层包含最多的句法信息，lastBERT 层包含最多的任务特定信息。此外，研究表明，虽然大多数句法信息可能集中在几个级别上，但语义信息分布在整个模型中，这解释了为什么一些重要案例最初处理错误，然后在更高层成功处理。

3.5 案例研究：用 Transformer 主题建模

3.5.1 目标

在本章中，我们研究了 Transformer 架构的几种应用。在本案例研究中，我们了解如何使用预训练（或微调）的 Transformer 模型进行主题建模。如果正在探索新的数据集，则可以在探索性数据分析期间使用此方法。

3.5.2 数据、工具和库

```
pip install -U datasets bertopic
```

清单 3.1 Python 环境设置

我们将使用预先训练的 Transformer 来探索 Yelp 评论数据集（来自 https://huggingface.co/datasets/yelp_review_full）并查看评论中的主题类型。我们将使用句子转换器库生成句子嵌入（请参阅 <https://github.com/UKPLab/sentence-transformers>）。它提供针对特定任务（例如语义搜索）进行预训练的模型。最后，我们将使用 BERTopic[86]（参见 <https://github.com/MaartenGr/BERTopic>）进行主题建模，并使用 Huggingface 数据集来加载数据。我们可以安装它们，如清单 3.1 所示。

1. 数据

```
from datasets import load_dataset
N = 10_000
dataset = load_dataset("yelp_review_full", split=f"train[:{N}]")
```

清单 3.2 加载数据集

我们首先通过清单 3.2 中的 HuggingfaceDatasets 加载 Yelp Review Full 数据集。数据集中有 650,000 条评论。为了使本案例研究的运行时间保持在合理范围内，我们将仅处理前 10,000 条评论。要处理更多评论，只需更改清单 3.2.

2. 计算嵌入

```
from sentence_transformers import SentenceTransformer
import numpy as np
# SentenceTransformer automatically checks if a GPU is available
embeddings_model = SentenceTransformer("all-mpnet-base-v2")
batch_size = 64
def embed(batch):
    batch["embedding"] = embeddings_model.encode(batch["text"])
    return batch

dataset = dataset.map(embed, batch_size=batch_size, batched=True)
dataset.set_format(type='numpy', columns=['embedding'],
                   output_all_columns=True)
```

清单 3.3 计算嵌入

我们将使用清单 3.3 中的句子转换器的“all-mpnet-

base-v2”模型生成嵌入。它的构建目的是在嵌入句子和较长文本范围时在语义搜索方面表现良好。

3.5.3 实验、结果和分析

1. 构建主题

```
from bertopic import BERTopic
topic_model = BERTopic(n_gram_range=(1, 3))
topics, probs = topic_model.fit_transform
(np.array(dataset["embedding"]))
print(f"Number of topics: {len(topic_model.get_topics())}")
```

清单 3.4 计算主题现在我们已经计算了 Yelp 评论的指定子集的嵌入，我们可以继续进行主题建模，如清单 3.4.

2. 主题大小分布

```
topic_sizes = topic_model.get_topic_freq()
topic_sizes
```

清单 3.5 对主题进行采样现在我们已经计算了主题分布，我们需要查看每个主题的评论类型。让我们首先看看主题大小的分布。清单 3.5 中的命令将输出主题大小的汇总表。请注意，主题的大小是包含该主题的评论数量。该表如图 3.6 所示。

Topic	Count	Name
0	-1	-1_food_good_place_were
1	0	0_italian_pasta_sauce_it_was
2	1	1_our_she_we_us
3	2	2_pittsburgh_in_pittsburgh_sandwich_bar
4	3	3_pizza_the_pizza_crust_cheese
...
145	144	144_brix_pizza_gluten_free_gluten
146	145	145_cream_ice_cream_ice_klavors
147	146	146_thai_pad_thai_house_pad_thai
148	147	147_massage_spas_the_spas_massages
149	148	148_store_the_produce_pick_save_produce

Figure 3.6 Table showing sizes and names of largest and smallest topics.

图 3.6 显示最大和最小主题的大小和名称的表。

注意 id 为 -1 的主题。这对应于 HDBSCAN 算法的未分配簇输出。未分配的簇由无法分配给其他簇之一的所有事物组成。它通常可以被忽略，但如果它太大，则表明我们选择的参数可能对我们的数据不利。可以查看未分配簇的内容以验证它是否包含不相关的单词。

3. 主题可视化

```
topic_model.visualize_barchart(top_n_topics=10, n_words=5, width=1000, height=80)
```

清单 3.6 可视化 10 个最大主题

BERTopic 包括多种方法来可视化主题分布的不同方面。在清单 3.6 中，我们对 10 个最大主题进行了采样，如图 3.7 所示。

```
topic_model.visualize_heatmap(top_n_topics=20, n_clusters=5)
```

清单 3.7 可视化 20 个最大主题的余弦相似度

BERTopic 还可以显示以下内容的余弦相似度热图主题嵌入，使用可视化热图方法，如列表 3.7 所示。如图 3.8 所示，它显示了每对主题的嵌入向量之间的重叠程度。

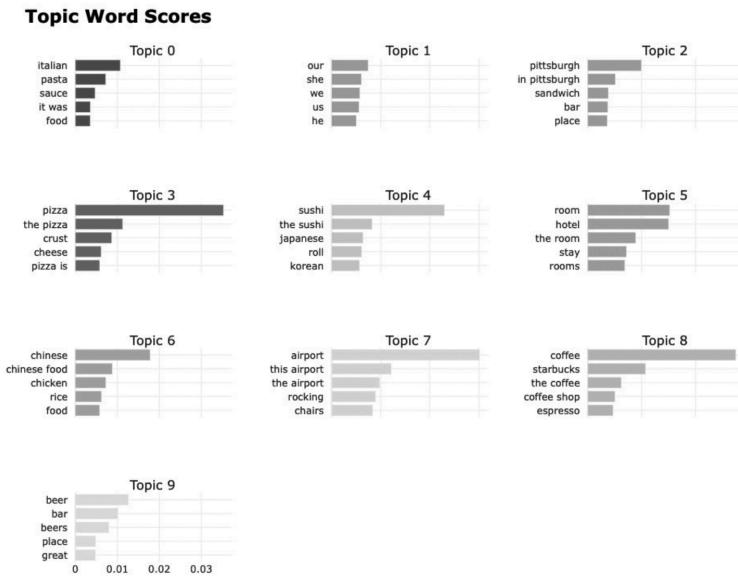


Figure 3.7 The 10 largest topics.

图 3.7 10 个最大主题。

4. 主题内容

```

def dump_topic_and_docs(text, topic_id):
    print(f"{text} size: {topic_sizes['Count'][topic_id + 1]}\n")
    n = len(topic_sizes) - 1

    if topic_id != -1:
        reviews = topic_model.get_representative_docs(topic_id)

        for review in reviews:
            print(review, "\n")

    return topic_model.get_topic(topic_id)[:10]

```

清单 3.8 打印主题大小和代表词

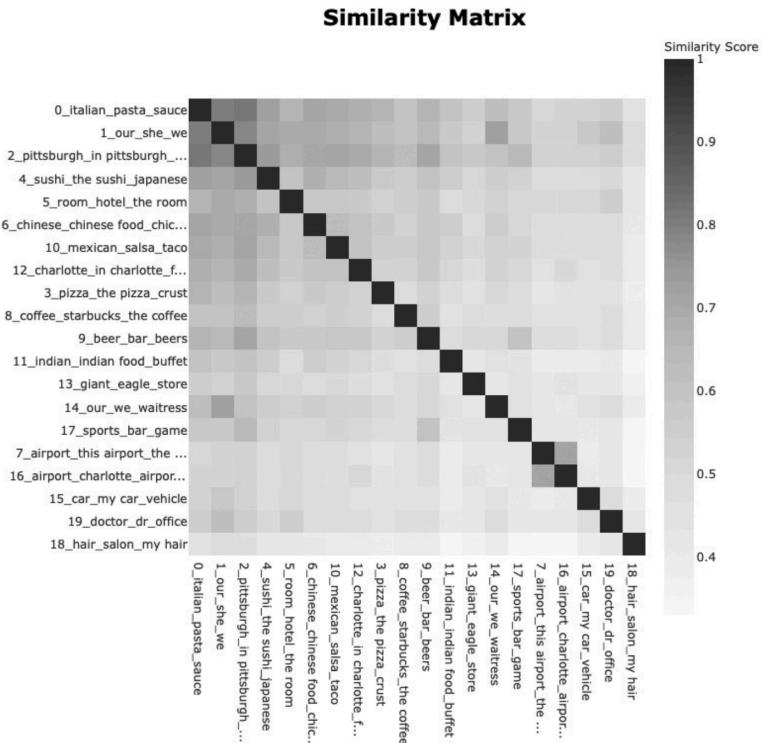


Figure 3.8 Heatmap of the cosine similarity of the 20 largest topics.

图 3.8 20 个最大主题的余弦相似度热图。

现在我们将看一个主题示例。我们在图 3.7 中查看了 10 个最大的主题，因此在这里我们将查看最大和最小的主题、具有中值大小的主题以及未分配簇的内容。清单 3.8 显示了一个简单的辅助函数，我们可以使用它来打印主题的大小以及与该主题最相关的 10 个单词。

```
>>> dump_topic_and_docs("Unassigned cluster", -1)
Unassigned cluster size: 2588
```

```
[('food', 0.0028285252847909887),  
 ('good', 0.0027913860678148716),  
 ('place', 0.0027706150477682947),  
 ('were', 0.0026233026252016375),  
 ('here', 0.0025870065464529087),  
 ('had', 0.002496872164775201),  
 ('great', 0.0024961777546672285),  
 ('it was', 0.002469218043237805),  
 ('there', 0.0024523636574226805),  
 ('service', 0.0024482400423906816)]
```

清单 3.9 未分配的集群

首先，我们将查看主题 id -1 的未分配集群，以验证我们可以安全地忽略其内容。如清单 3.9 所示。

正如我们所看到的，未分配簇的内容包含不强烈属于任何主题的单词。最大、最小和中位数主题分别如清单 3.10、3.11 和 3.12 所示。

```
>>> dump_topic_and_docs("Largest topic", 0)  
Largest topic size: 349  
  
****Representative reviews****  
This place makes me cringe! I've dined here with large groups of friends  
when we needed to have a big table and they all wanted to be bursting  
full of cheap food and that is really the only excuse to go to this place.  
\n\nOne reviewer mentioned the 90's music and the goofy food art on the  
walls. I could not agree more that this is so funny. Whoa and talk about  
noisy. This place is deafening inside on a Friday or Saturday night,  
worse than a cafeteria. I think that everyone with a City-Pass crams in  
there in search of the best two-for-one deal on a massive mound of macaroni
```

slathered in dreadful red sauce and salty cheese.

I actually ordered a salad as my main the last time that I dined there because I know how universally disappointing the pasta dishes were and they actually screwed up a salad. I am not sure what on earth it was supposed to be, but they called it a chopped salad and it had a little M next to it in the menu as if it were a specialty of the house. I asked for grilled chicken on top and received a dried out piece of leather sitting above a mess of lettuce, beans, nuts, cheese and peppers. Just plain salty and awful. Everything was either from a can or a jar.

I do agree with others who have said that the service is very fast and friendly. They kept the beer and wine flowing at our table at every visit. I think that is why so many of my friends just shovel this stuff down.

Ah well, Arrivederci (no more) Mama Ricotta

I met up with friends for a birthday gathering at Frankie's. It was my first time and, while I usually don't go out of my way for Italian, I was very impressed with Frankie's. I felt like I stepped back in time. The ambiance and decor seemed elegant from the 50s era, yet the friendliness of the server and the atmosphere was casual.

The menu contained everything you'd expect on an Italian restaurant menu and everything from the bread to the appetizer to the entree to the wine tasted delicious. Frankie's is definitely a place you can take friends and family to impress them, but not spend a fortune doing so.

When you think of a nice Italian restaurant, you don't think it would come in a strip mall, but Mama Ricotta's bucks the trend. Not only does the atmosphere & decor give the impression of a nicer Italian place, the food is pretty good.

While you may be thinking that this is a dinner only place, this is actually a really popular lunch place. There is usually a line during lunch, but it moves pretty quickly,

especially if the outside seating is open. While the food can be a tad on the pricey side, I have yet to have a meal I haven't been happy with. They have plenty of selections for all Italian lovers so don't expect just the obvious options. \n\nI'd suggest this place as more of a dinner place, mainly because of the prices along with the portion sizes. If you lunch it here, it may be a long afternoon at work trying to stay awake. And with their wine selection, making this a date destination isn't a bad idea either.

```
[('italian', 0.010707434311063687),  
 ('pasta', 0.007218630048706305),  
 ('sauce', 0.004690392541116093),  
 ('it was', 0.003576349729937027),  
 ('food', 0.0035416017180294685),  
 ('restaurant', 0.0034094836517629345),  
 ('salad', 0.003321322452779836),  
 ('olive', 0.0032739980714160824),  
 ('bread', 0.0032417620081978916),  
 ('italian food', 0.0031995754647714428)]
```

清单 3.10 最大话题：意大利美食餐厅

```
>>> dump_topic_and_docs("Smallest topic", n-1)  
Smallest topic size: 10  
****Representative reviews****  
There is no doubt that the ambiance is better than many other optical's  
shops but the prices are very high. \n\nWe saw the same product in  
Carolina's mall in sears Optical's at a way lower price point. I am  
definitely disappointed.
```

Meh. I hate how this store is laid out, and everything is tackily displayed. I also hate how they gouge you for printer ink. Huge store, not much going on inside. Too bad it's so close.

A very oddly laid out supermarket. the Deli isle is not very wide, the the walls are closing in on you, and they put the DIY olive cart in front to make it more cluttered. Just a poorly laid out store which is showing it's age.

```
[('store', 0.02825717462668952),  
 ('the produce', 0.01719937359158404),  
 ('pick save', 0.017115781758613058),  
 ('produce', 0.017000435507400078),  
 ('this store', 0.016450186193483884),  
 ('laid out', 0.01360566214521891),  
 ('opticals', 0.012361992937075754),  
 ('coppers', 0.011805417634250547),  
 ('hot cocoa', 0.011805417634250547),  
 ('produce quality', 0.011805417634250547)]
```

清单 3.11 最小主题：店铺氛围和布置

```
>>> dump_topic_and_docs("Median topic", n//2)  
Median topic size: 28  
****Representative reviews****  
Cosmos Cafe is yet another venue that is really a restaurant, but doubles  
as a nightlife hot spot on the weekends. North Carolina's laws on serving  
alcohol make it nearly impossible to just open a \"club\", so you see a lot
```

of these in Charlotte, especially downtown. \n\nThe staff is super friendly and doesn't hassle you for walking in and realizing \"Saturday night\" hasn't started yet because the dining patrons still haven't cleared out. So don't be afraid to check it at 11pm and then come back again at midnight or 1 if it's still slow. I've never been here to eat but it's a great place to stop in for a late night weekend drink to see what's going on. People dress nice and behave well. So you can actually get an expensive drink and not worry about someone elbowing it across the room. \n\nThere's plenty of seating on both the first and second floors and they have a respectable, if predictable, Scotch selection that helps me keep them in mind when I'm downtown, so I'm sure the rest of the liquor options are pretty good. Coming from Illinois where you can buy booze at grocery stores, it's depressing to walk into a bar in Charlotte and only have one or two choices. Cosmos expands the horizons and I appreciate that.

Need a place in Charlotte to get black out drunk without fear of waking up missing vital organs? Than the Westin Charlotte is the place for you. Was witness to an amateur drunk who passed out. Rather than doing the typical bounce and toss the staff gave genuine care and concern to the help him through the this moment of self reflection. One employee went so far as to ice his neck in hope to help him sober up, if nothing else at least remeber his name. Even when the EMTs arrived and drunk punches were thrown the staff stood stoically ready to show their support for their customer. So, it became a dinner and a show. Five stars. NERD RAGE!

Festivals. Fun. Beer. Lots of beer. Charlotte Center City Partners (or Find Your Center) puts on a lot of these types of festivals Uptown and in South End. When you check out their website or their weekly newsletter you'll be able to see lots of events coming up like Beer, Blues and BBQ, Taste of Charlotte, Speedstreet and the like. \n\nMany of these events and

festivals usually have beer available, hence why I'm a fan. And, yeah, I also really like supporting the local organization that's responsible for Uptown's development. If only there was a PBR festival...!

```
[('charlotte', 0.017158486331587473),  
 ('in charlotte', 0.013092184112676906),  
 ('beer', 0.01118926729742822),  
 ('city', 0.007059710231581003),  
 ('dance', 0.005752330716241153),  
 ('selection', 0.005730672906147966),  
 ('liquor', 0.005587858949299897),  
 ('center city', 0.005496678910160935),  
 ('beers', 0.005368697666709216),  
 ('events', 0.005089779403417317)]
```

清单 3.12 中值主题：夜生活夏洛特，NC

3.6 案例研究：微调

3.6.1 目标

本案例研究的目标是为任何句子分类提供微调标准 BERT 模型的分步演示，我们选择情感分类作为流行样本任务。

3.6.2 数据、工具和库

我们选择了 Google Play 应用程序评论数据集。该数据集包括三个类别的 15,746 个样本；即负、中性和正。我们使用 Huggingface 转换器库来执行微调任务，以及用于所有其他数据处理和可视化的标准 Python 数据科学堆栈 [87]（图 3.9-3.11）。

我们使用 BERT BASE 作为起始预训练 BERT 模式。Listing 3.6.3 显示了如何从 Trans-(a) (b) 加载预训练的 BERT 模型。formers 库并说明了将

用于微调 BERT 模型的标记化的输出 [87] 。

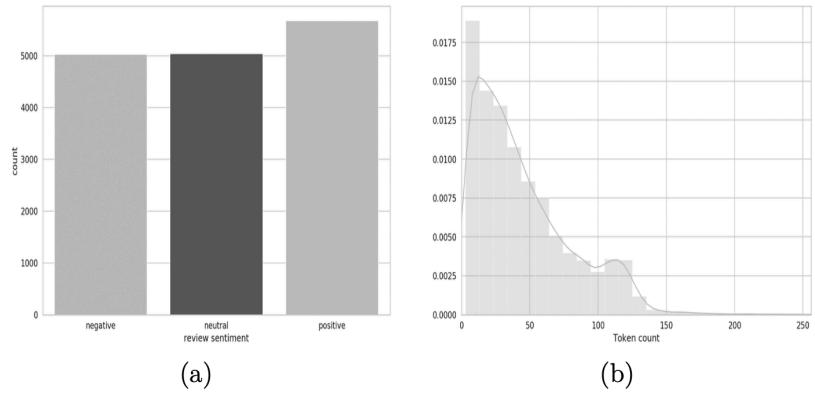


Figure 3.9 Exploratory data analysis of the sentiment classification dataset.

图 3.9 情感分类数据集的探索性数据分析。

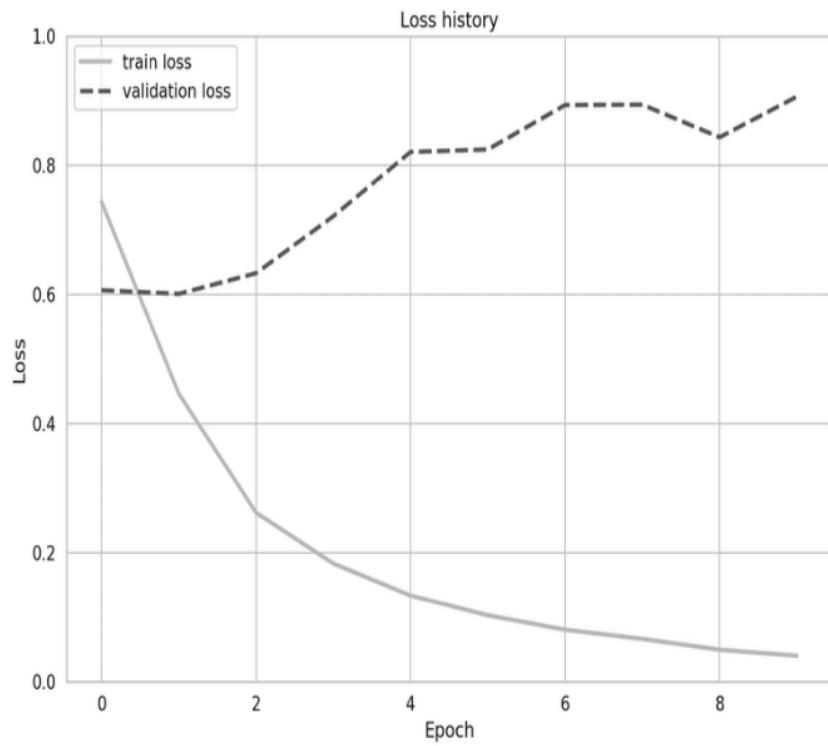


Figure 3.10 Training and validation loss throughout training.

图 3.10 整个训练过程中的训练和验证损失

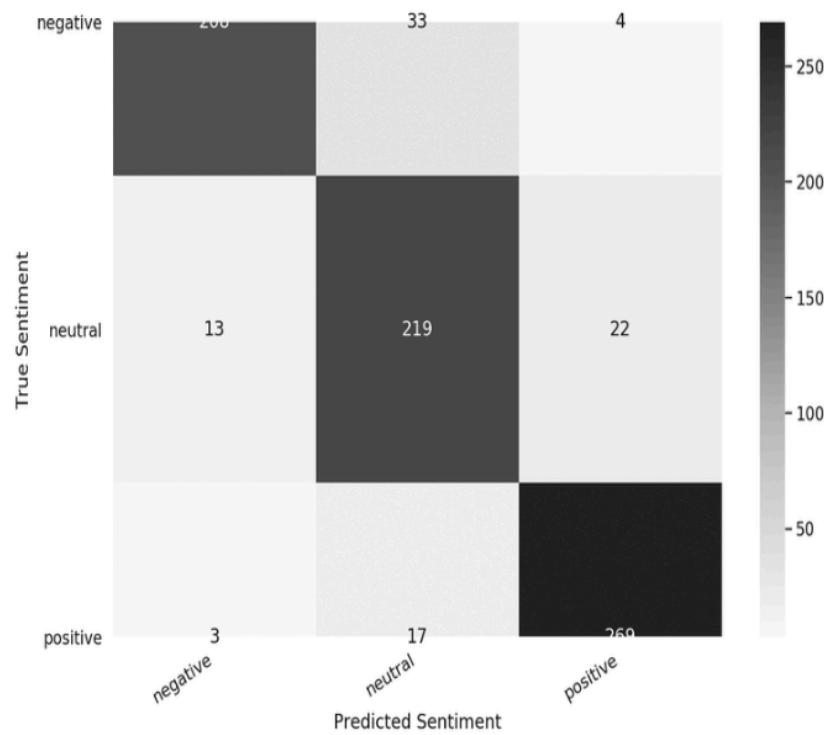


Figure 3.11 Confusion matrix of test set.

图 3.11 测试集的混淆矩阵。

3.6.3 实验、结果和分析

```

PRE_TRAINED_MODEL_NAME = 'bert-base-cased'
tokenizer = BertTokenizer.from_pretrained(PRE_TRAINED_MODEL_NAME)
sample_txt = 'When was I last outside? I am stuck at home for 2 weeks.'
tokens = tokenizer.tokenize(sample_txt)
    
```

```

token_ids = tokenizer.convert_tokens_to_ids(tokens)
print(f' Sentence: {sample_txt}')
print(f' Tokens: {tokens}')
print(f' Token IDs: {token_ids}')
encoding = tokenizer.encode_plus(
    sample_txt,
    max_length=32,
    truncation=True,
    add_special_tokens=True, # Add '[CLS]' and '[SEP]'
    return_token_type_ids=False,
    pad_to_max_length=True,
    return_attention_mask=True,
    return_tensors='pt', # Return PyTorch tensors
)
print(f'Encoding keys: {encoding.keys()}')
print(len(encoding['input_ids'][0]))
print(encoding['input_ids'][0])
print(len(encoding['attention_mask'][0]))
print(encoding['attention_mask'])
print(tokenizer.convert_ids_to_tokens(encoding['input_ids'][0]))

```

```

MAX_LEN = 160
class GPReviewDataset(Dataset):
    def __init__(self, reviews, targets, tokenizer, max_len):
        self.reviews = reviews
        self.targets = targets
        self.tokenizer = tokenizer
        self.max_len = max_len

```

```

def __len__(self):
    return len(self.reviews)

def __getitem__(self, item):
    review = str(self.reviews[item])
    target = self.targets[item]
    encoding = self.tokenizer.encode_plus(
        review, add_special_tokens = True, max_length =
        self.max_len, return_token_type_ids = False,
        return_attention_mask = True, truncation = True,
        pad_to_max_length = True, return_tensors = 'pt',)
    return {
        'review_text':review,
        'input_ids': encoding['input_ids'].flatten(),
        'attention_mask':encoding['attention_mask'].flatten(),
        'targets':torch.tensor(target, dtype=torch.long)
    }

def create_data_loader(df, tokenizer, max_len, batch_size):
    ds = GPReviewDataset(
        reviews = df.content.to_numpy(), targets =
        df.sentiment.to_numpy(), tokenizer =
        tokenizer, max_len = max_len)
    return DataLoader(ds, batch_size=batch_size, num_workers=4)

df_train, df_test = train_test_split(df,

```

```

    test_size = 0.1, random_state = RANDOM_SEED)
df_val, df_test = train_test_split(df_test,
    test_size = 0.5, random_state = RANDOM_SEED)
print(df_train.shape, df_val.shape, df_test.shape)
BATCH_SIZE = 16
train_data_loader = create_data_loader(df_train,
    tokenizer, MAX_LEN, BATCH_SIZE)
val_data_loader = create_data_loader(df_val,
    tokenizer, MAX_LEN, BATCH_SIZE)
test_data_loader = create_data_loader(df_test,
    tokenizer, MAX_LEN, BATCH_SIZE)

#Testing to see if the data loader works appropriately
data = next(iter(train_data_loader))
print(data.keys())
print(data['input_ids'].shape)
print(data['attention_mask'].shape)
print(data['targets'])v

```

```

EPOCHS = 10
optimizer = AdamW(model.parameters(),
    lr= 2e-5, correct_bias=False)
total_steps = len(train_data_loader) * EPOCHS
scheduler = get_linear_schedule_with_warmup( optimizer,
    num_warmup_steps = 0, num_training_steps=total_steps)
loss_fn = nn.CrossEntropyLoss().to(device)

```

```

def train_epoch(model, data_loader, loss_fn,
    optimizer, device, scheduler, n_examples):

```

```

model=model.train()
losses = []
correct_predictions = 0
for d in data_loader:
    input_ids = d["input_ids"].to(device)
    attention_mask = d["attention_mask"].to(device)
    targets = d["targets"].to(device)
    outputs = model(input_ids=input_ids,
                    attention_mask=attention_mask)
    _, preds = torch.max(outputs, dim = 1)
    loss = loss_fn(outputs, targets)
    correct_predictions += torch.sum(preds == targets)
    losses.append(loss.item())
    loss.backward()
    nn.utils.clip_grad_norm_(model.parameters(), max_norm=1.0)
    optimizer.step()
    scheduler.step()
    optimizer.zero_grad()
return correct_predictions.double()/n_examples, np.mean(losses)

def eval_model(model, data_loader, loss_fn, device, n_examples):
    model = model.eval()
    losses = []
    correct_predictions = 0
    with torch.no_grad():
        for d in data_loader:
            input_ids = d["input_ids"].to(device)
            attention_mask = d["attention_mask"].to(device)
            targets = d["targets"].to(device)

```

```

        outputs = model(input_ids = input_ids,
                          attention_mask = attention_mask)
        _,preds = torch.max(outputs, dim = 1)
        loss = loss_fn(outputs, targets)
        correct_predictions += torch.sum(preds == targets)
        losses.append(loss.item())

    return correct_predictions.double()/n_examples, np.mean(losses)

```

```

history = defaultdict(list)
best_accuracy = 0
for epoch in range(EPOCHS):
    print(f'Epoch {epoch + 1}/{EPOCHS}')
    print('-'*15)
    train_acc, train_loss = train_epoch(model, train_data_loader,
                                         loss_fn, optimizer, device, scheduler, len(df_train))
    print(f'Train loss {train_loss} accuracy {train_acc}')
    val_acc, val_loss = eval_model(model, val_data_loader,
                                    loss_fn, device, len(df_val))
    print(f'Val loss {val_loss} accuracy {val_acc}')
    history['train_acc'].append(train_acc)
    history['train_loss'].append(train_loss)
    history['val_acc'].append(val_acc)
    history['val_loss'].append(val_loss)
    if val_acc>best_accuracy:
        torch.save(model.state_dict(), 'best_model_state.bin')
        best_accuracy = val_acc

plt.plot(history['train_acc'], label='train accuracy')

```

```
plt.plot(history['val_acc'], label='validation accuracy')
plt.title('Training history')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend()
plt.ylim([0,1])
```

```
plt.plot(history['train_loss'], label='train loss', linewidth=3)
plt.plot(history['val_loss'], '--', label='validation loss', linewidth=3)
plt.title('Loss history')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend()
plt.ylim([0, 1]);
```

```
def get_predictions(model, data_loader):
    model = model.eval()
    review_texts = []
    predictions = []
    prediction_probs = []
    real_values = []
    with torch.no_grad():
        for d in data_loader:
            texts = d["review_text"]
            input_ids = d["input_ids"].to(device)
            attention_mask = d["attention_mask"].to(device)
            targets = d["targets"].to(device)
```

```

outputs = model(input_ids = input_ids,
                 attention_mask = attention_mask)
_, preds = torch.max(outputs, dim=1)
probs = F.softmax(outputs, dim =1)
review_texts.extend(texts)
predictions.extend(preds)
prediction_probs.extend(probs)
real_values.extend(targets)
predictions = torch.stack(predictions).cpu()
prediction_probs = torch.stack(prediction_probs).cpu()
real_values = torch.stack(real_values).cpu()

return review_texts, predictions, prediction_probs, real_values

def show_confusion_matrix(confusion_matrix):
    hmap = sns.heatmap(confusion_matrix, annot=True, fmt="d", cmap="Blues")
    hmap.yaxis.set_ticklabels(hmap.yaxis.get_ticklabels(),
                             rotation = 0, ha='right')
    hmap.xaxis.set_ticklabels(hmap.xaxis.get_ticklabels(),
                             rotation = 30, ha='right')
    plt.ylabel('True Sentiment')
    plt.xlabel('Predicted Sentiment')
    y_review_texts, y_pred, y_pred_probs, y_test =
    get_predictions(model, test_data_loader)
    print(classification_report(y_test, y_pred,
                                target_names=class_names))
    cm = confusion_matrix(y_test, y_pred)
    df_cm = pd.DataFrame(cm, index=class_names, columns = class_names)
    show_confusion_matrix(df_cm)

```

4 多语言 Transformer 架构

BERT [26] 的引入改变了自然语言处理 (NLP) 领域，并在广泛的任务上取得了最先进的性能。最近，训练 NLP 算法仅使用标记数据，由于每个新任务和每个新项目收集数据的可用性和成本有限，这多年来阻碍了它们的进展。BERT 的一个主要贡献是引入了一种用于训练 NLP 系统的新渠道，其中算法可以从大量廉价的未标记数据中学习核心和通用自然语言概念，这一过程也称为自监督学习或模型预训练。然后使用更小的特定于任务的标记数据集对预训练的模型进行微调，以适应任何特定的下游任务。这种预训练和微调的新流程构成了机器学习领域大多数进步的核心。BERT 在英语 NLP 任务中取得的巨大成功激励了它在其他语言中的使用。然而，使用 BERT 的管道仅适用于具有足够大的未标记数据进行预训练的语言。这促进了多语言模型的开发，其中模型在多种语言上进行预训练，希望模型能够将核心 NLP 知识从高资源语言转移到低资源语言，我们最终得到一个跨语言对齐多语言表示的多语言模型。本章介绍多语言 Transformer 架构的最新发展。多语言 transformer 架构完全定义为：

- 模型架构（层和组件）
- 预训练任务
- 预训练数据
- 微调任务和基准

本章从用于自然语言理解 (NLU) 的不同多语言 transformer 架构开始) 和自然语言生成 (NLG)。第 4.2 节继续描述多语言数据的最新技术，其中包括未标记的预训练数据和下游 NLP 任务的微调基准。第 4.3 节提供了对多语言 Transformer 模型内部工作的一些见解。最后，第 4.4 节提供了零样本多语言情感分类的实际案例研究。

4.1 多语言 Transformer 架构

本节讨论多语言 Transformer 的不同架构设计。我们将多语言转换器分为 (i) 自然语言理解 (NLU) 架构和 (ii) 自然语言生成 (NLG) 架构。表

4.1[88] 总结了本书出版时可用的有影响力的多语言 Transformer 模型。

4.1.1 基本多语言 Transformer

多语言 Transformer 模型通常基于 mBERT 架构 [26]。在本节中，我们将描述多语言转换器架构的基本组件。

表 4.1 多语言语言模型摘要 [88]

Model	Architecture	Objective Function	Mono	Parallel	Lang	Ref
IndicBERT	33M	MLM	IndicCorp	x	12	[89]
mBERT	172M	MLM	Wikipedia	x	104	[26]
Amber	172M	MLM, TLM, CLWA, CLSA	Wikipedia	✓	104	[90]
MuRIL	236M	MLM, TLM	CommonCrawl+ Wikipedia	✓	17	[91]
VECO-small	247M	MLM, CS-MLM	CommonCrawl	✓	50	[92]
Unicoder	250M	MLM, TLM, CLWR, CLPC, CLMLM	Wikipedia	✓	15	[93]
XLM-15	250M	MLM, TLM	Wikipedia	✓	15	[94]
InfoXLM-base	270M	MLM, TLM, XLCO	CommonCrawl	✓	94	[95]
XLM-R-base	270M	MLM	CommonCrawl	x	100	[96]
HiCTL-base	270M	MLM, TLM, HICTL	CommonCrawl	✓	100	[97]
Ernie-M-base	270M	MLM, TLM, CAMLM, BTMLM	CommonCrawl	x	100	[98]
HiCTL-Large	559M	MLM, TLM, HICTL	CommonCrawl	✓	100	[97]
Ernie-M-Large	559M	MLM, TLM, CAMLM, BTMLM	CommonCrawl	✓	100	[98]
InfoXLM-Large	559M	MLM, TLM, XLCO	CommonCrawl	✓	94	[95]
XLM-R-Large	559M	MLM	CommonCrawl	x	100	[96]
RemBERT	559M	MLM	CommonCrawl+ Wikipedia	x	110	[99]
X-STILTS	559M	MLM	CommonCrawl	x	100	[100]
XLM-17	570M	MLM, TLM	Wikipedia	✓	17	[94]
XLM-100	570M	MLM, TLM	Wikipedia	x	100	[94]
VECO-Large	662M	MLM, CS-MLM	CommonCrawl	✓	50	[92]

1. 输入层

提供一系列标记作为多语言转换器的输入。标记输入源自有限词汇表（通常是子词词汇表）的一次性表示。通常，该词汇表是通过使用 BPE[101]、WordPiece[83] 或 SentencePiece[102] 等算法连接来自多种语言的单语数据来学习的。为了保证词汇表中能够很好地表示不同的语言和文字，可以使用指数加权平滑对数据进行采样，或者可以通过划分词汇表大小来学习语言簇的不同词汇表 [103]。

2. transformer 层

通常，多语言语言模型 (MLM) 构成多语言 transformer 的编码器。它由一堆 Nlayer 组成，每个 Nlayer 都有一个注意力头，后面跟着一个前馈神经网络。注意力头通过使用注意力加权线性组合句子中所

有其他标记的表示来计算输入序列中每个标记的嵌入。连接所有注意力头的嵌入并将它们传递给前馈网络，为每个输入单词生成一个 d 维嵌入。

3. 输出层

通常，最终转换器层的输出被用作每个标记的上下文表示，而与 [CLS] 标记关联的嵌入是被认为是整个输入文本的嵌入。或者可以通过将标记嵌入汇集在一起 来生成文本嵌入。输出层有一个基本的线性变换，后面跟着一个 Softmax，它接受来自前一个变换层的标记嵌入作为输入，并生成词汇表标记的概率分布。

4.1.2 单编码器多语言 NLU

1. mBERT

多语言 BERT (mBERT) 架构与原始 BERT 架构相同。它不是仅在具有英语衍生词汇的单语英语数据上进行训练，而是在 104 种语言的维基百科页面上进行训练，并具有通用的 WordPiece 词汇。它没有参考输入语言，也缺乏确保翻译等效句子具有紧密表示的明确技术。所有 104 种语言共享 110k WordPiece 词汇表。为了解决维基百科的内容不平衡问题，例如英语维基百科的数量是其 120 倍像库尔德语维基百科这样的文章，次要语言被过度采样，而主要语言被不足采样。除了其他因素之外，相同的词汇可以实现某种跨语言表示。接下来，我们简单介绍一下 mBERT 的预训练任务，即多语言 Masked Language Model (mMLM) 和 NextSentence Prediction (NSP)。

Masked Language Model (MLM) 这是 BERT 引入的标准 token 级无监督预训练任务。然而，在多语言转换器的上下文中，它与多种语言一起使用而没有对齐。它遵循相同的 MLM 程序，其中随机选取 15% 的标记，然后 (i) 以 80% 的概率将它们替换为 [MASK] 标记，或 (ii) 以 10% 的概率将它们替换为随机标记，(iii) 它们以 10% 的概率保持不变。MLM 任务的目标是使用输入标记序列两侧的未屏蔽标记来预测屏蔽标记。形式上，设 x 为输入标记序列， M_x 为屏蔽标记集合。

MLM 损失定义如下：

$$\mathcal{L}_{MLM}^{(x)} = -\frac{1}{|M_x|} \sum_{i \in M_x} \log P(x_i | x \setminus M_x) \quad (4.1)$$

4.1 说明了 MLM 任务，其中 x 和 y 代表不同语言的单语言输入句子。尽管该任务没有考虑有关池屏蔽输入的多种语言的任何信息，但它在生成跨语言良好对齐的跨语言表示方面显示出了巨大的价值。见解部分将提供有关此行为背后因素的更多详细信息。

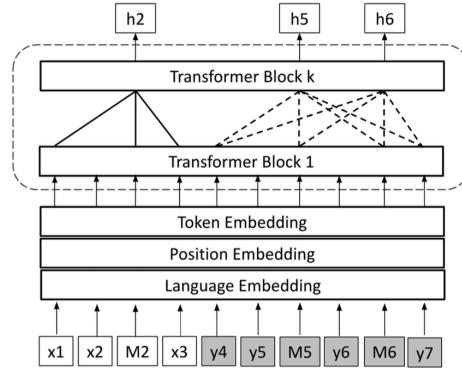


Figure 4.1 Illustration of multilingual MLM task [71, 194]. x and y are monolingual sentences in different languages that are not parallel.

图 4.1 多语言 MLM 任务图示 [26, 98]。 x 和 y 是不并行的不同语言的单语句子。

Next Sentence Prediction (NSP) 在句子级预训练任务的背景下，NSP 辅助模型学习短语之间的关联 [26]。这是一个学习识别连续句子的二元句子对分类问题。对于两个句子 x and y ，表示两个句子 (x, y) 聚合表示的 [CLS] 标记向量被传递到 Sigmoid 层以获得连续句子的概率。为了准备训练，创建短语对时，百分之五十的出现是连续的，其余百分之五十是不连续的。在句子级别对模型进行预训练有利于下游任务，例如问答 (QA)、自然语言推理 (NLI) 和语义文本相似性 (STS)，

这些任务需要句子对作为输入。令 $l \in \{1, 0\}$ 表示两个句子 (x, y) 是否连续, NSPloss 定义如下:

$$\mathcal{L}_{NSP}^{(x,y)} = -\log P(l|x, y) \quad (4.2)$$

2. XLM

跨语言语言模型 (XLM) [94] 是通过从单语和并行语料库学习来改进 mBERT 架构。为了从单语言数据中学习, XLM 使用 mBERT 使用的标准 MLM 预训练任务。XLM 还建议使用 Causal Language Modeling (CLM) 进行单语预训练。因果语言模型 (CLM) CLM 是经典语言建模的核心预训练任务, 其中模型根据先前的单词上下文正式预测下一个单词。设 $x = \{x_1, x_2, x_3, \dots, x_{|x|}\}$ 表示标记序列, 其中 $|x|$ 表示序列中标记的数量。CLM 损失定义为:

$$L(x)CLM = -1|x||x|/summationdisplayi = 1logP(xi|x < i) \quad (4.3) \quad (1)$$

其中 $x_{<i} = x_1, x_2, x_3, \dots, x_{i-1}$

XLM 的主要贡献是引入了使用并行企业的预训练任务, 明确鼓励跨语言表示的对齐; 即翻译语言模型 (TLM)。

Translation Language Model (TLM) TLM 是 MLM 的一种变体, 允许在跨语言预训练中使用并行语料库 [94]。编码器的输入是一对并行句子 (x, y) 。与 MLM 类似, 所有 k 个标记都被屏蔽, 以便它们可能属于 x 或 y 句子。为了预测 x 中的屏蔽词, 模型可能依赖于 x 中的周围上下文或翻译 y , 对于 y 中的屏蔽标记反之亦然。这隐含地迫使编码器获取跨语言对齐的表示。本质上, 如果 x 中的屏蔽标记的上下文不足, 模型将更多地关注 y 中的上下文, 以更好地预测屏蔽标记。图 4.2 说明了 TLM 如何执行跨语言注意力。为了定义并行句子对 (x, y) 的 TLM, 令 M_x 和 M_y 表示两个句子中的屏蔽位置集, 然后 $x \setminus M_x$ 和 $y \setminus M_y$ 表示 x 和 y 的屏蔽版本。TLM 损失函数 [24] 定义如下:

$$\mathcal{L}_{TLM}^{(x,y)} = -\frac{1}{|M_x|} \sum_{i \in M_x} \log P(x_i | x_{\setminus M_x}, y_{\setminus M_y}) - \frac{1}{|M_y|} \sum_{i \in M_y} \log P(y_i | x_{\setminus M_x}, y_{\setminus M_y}) \text{tag4.4} \quad (2)$$

XLM 为 MLM 和 CLM 预训练任务如何提供强大的跨语言特征提供了广泛的证据，这些特征可用于无监督机器翻译和其他下游监督跨语言分类任务。跨语言语言模型 (XLM) 还显示了将 TLM 与 MLM 结合使用对跨语言表示对齐质量的巨大影响，这已通过其在 XNLI 基准测试中的表现得到证明。

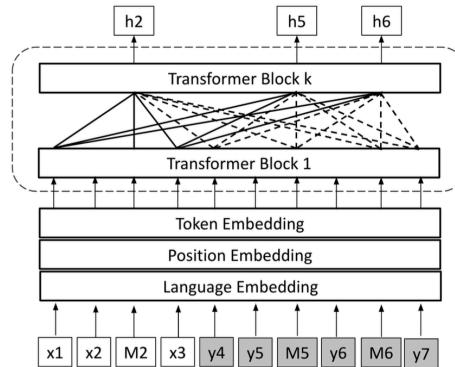


Figure 4.2 Illustration of TLM [146, 194] showing the attention mechanism across parallel sentences (x, y) which promotes cross-lingual alignment of the produced representations.

图 4.2 TLM [94, 98] 的插图显示了跨并行句子 (x, y) 的注意机制，这促进了生成的表示的跨语言对齐。

3. XLM-RoBERTa

XLM-RoBERTa (XLM-R) [96] 是仅基于 MLM 预训练任务的多语言 transformer 模型。XLM-R 使用 100 种语言的 2.5 TB CommonCrawl 数据进行训练。XLM-R 是多语言 Transformer 模型进展谱系中的一项非常重要的技术，因为它在重要的下游任务（例如序列标记和问题解答）上比 mBERT 和 XLM 提供了强大的性能提升。尽管事实上它没

有使用并行语料库进行预训练。XLM-R 遵循 XLM 的架构设计，没有使用语言嵌入，从而可以更好地处理语码转换。XLM-R 的主要贡献是使用单语言数据实现最先进的性能，并证明对更干净和更大的训练数据、词汇量以及正确水平的超参数调整的价值的深刻见解。

4. ALM

交替语言模型 (ALM) ALM [104] 是跨语言表示的预训练任务，其工作方式与 TLM 预训练任务相同，除了输入数据的性质。TLM 使用平行句子进行输入，但是 ALM 使用从平行句子生成的代码转换句子。在语码转换句子中，每个句子对的深度潜水短语与相应的翻译短语随机交换。对于平行句子对 (x, y) ，其中源句子 $x = \{x_1, x_2, \dots\}$ 且目标句子 $y = \{y_1, y_2, \dots\}$ ，ALM 模型的输入是代码转换序列 $u = \{u_1, u_2, \dots\}$ 由 (x, y) 对组成。 $u_{[i,j]}$ 被分配给 $x_{[a,b]}$ 或 $y_{[c,d]}$ ，其中约束是这两个序列是并行对 (x, y) 使得 $1 \leq a \leq b \leq |x|$ 且 $1 \leq c \leq d \leq |y|$ 。语码转换序列 u 中源标记 x 的比例为 α 。换句话说，如果 $\alpha = 1$ ，则整个输入序列是单语言的，由源句子 x 组成，如果 $\alpha = 0$ ，则输入序列就是目标句子 y ，我们通过使用 $0 < \alpha < 1$ 来构造代码-交换数据。

使用工具对平行语料库中的源句子和目标句子进行词对齐，并使用机器翻译技术提取双语短语表。接下来，根据比率 α 替换源短语和目标短语来构造语码转换数据。鉴于 α 随机决定从源句子和目标句子中选取哪些短语，每个句子对可以产生许多语码转换序列，这对于克服可用并行语料库通常有限的大小非常有用。然后，ALM 对代码交换数据使用标准 MLM 预训练任务。形式上，对于平行句子对 (x, y) ，令 z 为由 x 和 y 生成的代码转换句子，则 ALM 损失定义为：

$$\mathcal{L}_{ALM}^{(z(x,y))} = -\frac{1}{|M|} \sum_{i \in M} \log P(z_i | z \setminus M) \quad (4.5)$$

其中 $z \setminus M$ 是 z 的屏蔽版本， M 是 $z \setminus M$ 中屏蔽标记位置的集合。

4.3 说明了 ALM 如何使用并行句子对创建一组代码转换训练数据，然后将其用于训练 MLM transformer。

5. Unicoder

Unicoder[93] 是使用并行语料库学习跨语言表示的另一种架构。它使用与 XLM 相同的架构，并引入了三个新的跨语言预训练任务；即跨语言单词恢复 (CLWR)、跨语言释义分类 (CLPC) 和跨语言屏蔽 LM (CLMLM)。

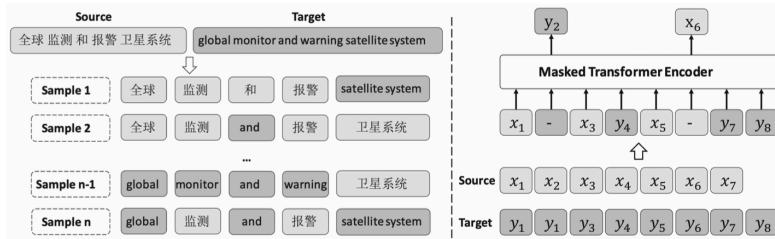


Figure 4.3 Illustration of Alternating Language Model (ALM) [282] where a parallel sentence pair (x, y) is used to generate multiple code-switch examples that are used to train an MLM transformer.

图 4.3 交替语言模型 (ALM) 图示 [104]，其中并行句子对 (x, y) 用于生成多个代码转换示例，用于训练 MLM 转换器。

跨语言单词恢复 (CLWR) CLWR[93] 使用平行语料库来学习两种语言之间的单词对齐。预训练任务从可训练的注意力矩阵 [23] 开始，该矩阵学习用目标语言标记嵌入来表示源语言标记嵌入。然后，将注意力矩阵中学习到的变换提供给跨语言模型，以学习重建源标记嵌入。形式上，给定一个平行句子对 (x, y) ，其中 $x = (x_1, x_2, \dots, x_m)$ 是一个带有来自源语言 s 的 m 个单词的句子， $y = (y_1, y_2, \dots, y_n)$ 是一个包含来自目标语言 t 的 n 单词的句子。CLWR 首先通过 y [93] 的所有词嵌入来表示每个 x_i ($x_i^t \in R^h$)：

$$x_i^t = \sum_{j=1}^n \text{softmax}(A_{ij}) y_j^t \quad (4.6)$$

其中 $x_i^s \in R^h$ 和 $y_j^t \in R^h$ 分别是 x_i 和 y_j 的表示， h 表示表示维度， $A \in R^{m \times n}$ 的注意力矩阵定义为：

$$A_{ij} = W \left[x_i^s, y_j^t, x_i^s \odot y_j^t \right] \quad (4.7)$$

注意， $W \in R^{3 \times h}$ 是一个可训练的权重矩阵，而 \odot 是逐元素乘法。然后 Unicoder 将 $X^t = (x_1^t, x_2^t, \dots, x_n^t)$ 作为输入，并预测原始单词序列 x 。

跨语言释义分类 (CLPC) CLPC [93] 通过提出平行句子对 (x, y) 的释义分类预训练任务，引入了使用平行数据的不同方式，其中平行句子对 (x, y) 为正类，非平行句对 (x, z) 为负类。使用较小的释义检测模型来挑选训练数据样本，使得对于每个 (x, y) 和 (x, z) 句子对， z_i 选择接近 x 但不等于 y 。

跨语言 Masked LM (CLMLM) CLMLM [93] 是 TLM 的扩展，具有相同的跨语言注意方案。然而，虽然 TLM 接受成对的并行句子，但 CLMLM 的输入是在文档级别创建的，其中并行文档（即具有多个并行翻译的完整文档）中的多个句子被其他语言翻译替换。这是受到一些报道的使用较长单语文本预训练成功的启发。Unicoder 模型使用 XLM 模型进行初始化，然后使用组合的 MLM、TLM、CLWR、CLPC 和 CLMLM 任务进行预训练，并在 XNLI 和 XQA 基准上进行微调。

6. INFOXLM

信息论跨语言模型 (INFOXLM) [95] 从信息论的角度处理跨语言表示对齐问题，它展示了多语言预训练任务实际上可以被视为最大化多语言之间的相互信息不同粒度的上下文。例如，InfoXLM 表明，多语言 Masked Language Model (mMLM) 预训练任务本质上最大化了同一语言中掩码标记与其上下文之间的相互信息。另一方面，跨语言的锚标记促使跨语言上下文紧密相关。TLM 还有一个信息论解释，它被证明可以最大化掩码标记和并行语言上下文之间的相互信息，作为隐式跨语言表示对齐机制。使用所提供的信息论框架，InfoXLM 提出了一种新的基于对比学习的跨语言预训练任务，即跨语言对比学习。

跨语言对比学习 (XLCO) XLCO [95] 提出了一种信息论方法来产生跨语言表示使用平行语料库。XLCO 鼓励 Transformer 编码器与非

平行句子相比最大化平行句子表示之间的互信息，从而执行对比学习。TLM 在信息理论框架内被证明可以最大化标记序列互信息。另一方面，XLCO 最大化并行句子之间的序列级互信息。设 (x_i, y_i) 为平行句子对，其中 y_i 是 x_i 的翻译， $\{y_j\}_{j=1, j \neq i}^N$ 是与 y_i 相同语言但不是 x_i 的翻译的句子集合。XLCO 使用基于 InfoNCE [105] 的损失函数最大化 x_i 和 y_i 之间的信息内容，定义为：

$$\mathcal{L}_{XLCO}^{(x_i, y_i)} = -\log \frac{\exp(f(x_i)^\top f(y_i))}{\sum_{j=1}^N \exp(f(x_i)^\top f(y_j))} \quad (4.8)$$

其中 $f(x_i)$ 和 $f(y_i)$ 分别是句子 x_i 和 y_i 的 transformer 编码器表示。对于负采样，XLCO 不会从 $\{y_j\}_{j=1, j \neq i}^N$ 显式采样，而是使用混合对比 [95]

INFOXLM 使用 XLM-R 模型参数进行初始化，然后使用组合的 mMLM、TLM 和 XLCO 任务进行预训练，这些任务被制定为互信息最大化任务；单语言标记序列互信息 (MMLM)、跨语言标记-序列互信息 (TLM) 和跨语言序列级互信息 (XLCO)。

7. AMBER

对齐多语言双向编码器 (AMBER) [90] 是一种创建跨语言序列级互信息 (XLCO) 的技术。使用并行数据的语言表示与词级和句子级对齐预训练任务，即跨语言词对齐 (CLWA) 和跨语言句子对齐 (CLSA) 任务。

跨语言词对齐 (CLWA) 与 CLWR 类似，CLWA [90] 还旨在使用并行语料库学习不同语言之间的单词对齐。CLWA 使用两个不同的注意力掩码作为 transformer 模型的输入。训练后，在顶部 transformer 层获得两个注意力矩阵，即源到目标注意力 $A_{x \rightarrow y}$ 和目标到源注意力 $A_{y \rightarrow x}$ 。假设 $A_{x \rightarrow y}$ 测量源到目标转换的对齐，对于 $A_{y \rightarrow x}$ 反之亦然，希望使用 $A_{x \rightarrow y}$ 或 $A_{y \rightarrow x}$ 注意力矩阵对单词进行类似的对齐。为此，模型鼓励在训练期间最小化 $A_{x \rightarrow y}$ 和 $A_{y \rightarrow x}$ 之间的距离。

跨语言句子对齐 (CLSA) CLSA [90] 旨在使用并行数据强制跨语言句

子表示的对齐。 \mathcal{M} 和 \mathcal{P} 分别是单语言数据和并行数据。对于每个平行句子对 (x, y) ，分别生成 x 和 y 的不同句子嵌入，即 c_x 和 c_y 。 c_x 和 c_y 是通过对编码器最后一层的嵌入进行平均而获得的。嵌入对齐是通过使用以下损失函数鼓励模型预测输入句子 x 的正确翻译 y 来进行的：

$$\mathcal{L}_{CLAS}^{(x,y)} = -\log \frac{e^{\mathbf{c}_x^T \mathbf{c}_y}}{\sum_{y' \in \mathcal{M} \cup \mathcal{P}} e^{\mathbf{c}_x^T \mathbf{c}_{y'}}} \quad (4.9)$$

其中 $\mathcal{L}_{CLSA}^{(x,y)}$ 的 y' 可以是任何句子，没有语言限制。

AMBER 使用 MLM、TLM、CLWA、CLSA 任务的组合进行训练，并在 XTREME 基准上进行评估，其性能优于 XLM-R 和 Unicoder。

8. ERNIE-M

许多技术已经显示了使用并行和单语言语料库来获得更好对齐的跨语言表示的价值。然而，平行语料库相对于单语言数据的大小总是有限的，除非我们收集大量的平行语料库，否则限制了可能的进展。ERNIE-M [98] 使用大型单语语料库和有限的平行语料库来生成跨语言表示。为了增强生成的表示，ERNIE-M 使用大量可用的单语语料库生成伪并行句子，然后将它们与并行数据一起使用以改进跨语言表示的对齐。ERNIE-M 提出了两个预训练任务，Cross-Attentionmasked Language Modeling (CAML) 和 Back-Translation MaskedLanguage Modeling (BTMLM)

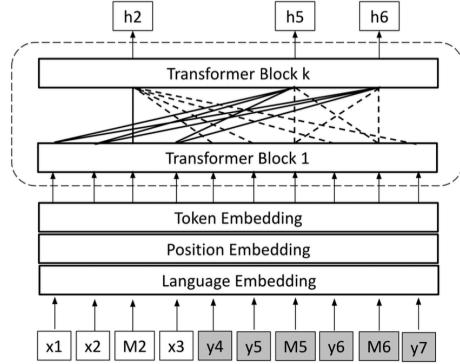


Figure 4.4 Cross-attention Masked LM (CAML) where a masked token can only attend to the opposite sentences for prediction [194].

Cross-attention Masked LM (CAML) Cross-attention MaskedLM (CAML) [98] 是另一个并行语料库与 TLM 密切相关的预训练任务。如前所述，对于给定的并行句子对 (x, y) ， x 中的屏蔽标记具有访问权限，并且可能关注 x 的上下文和 y 的上下文，这同样适用于 y 中的 masked token。另一方面，CAML 通过仅保留对相应并行句子的访问来限制 TLM。因此， x 中的 masked token 只能使用 y context 进行预测，反之亦然。禁止关注与屏蔽标记相同的句子的上下文会迫使算法学习并行语言的表示。否则，算法可能会采取更简单的路径，并且更倾向于使用屏蔽标记的相同句子来进行预测。在 TLM 中使用相同的设置，其中对于句子对 (x, y) ，令 M_x 和 M_y 表示两个句子中的屏蔽位置集，然后 $x_{\setminus M_x}$ 和 $y_{\setminus M_y}$ 表示 x 和 y 的屏蔽版本。CAML 的损失函数定义如下：

$$\mathcal{L}_{CAML}^{(x,y)} = -\frac{1}{|M_x|} \sum_{i \in M_x} \log P(x_i | y_{\setminus M_y}) - \frac{1}{|M_y|} \sum_{i \in M_y} \log P(y_i | x_{\setminus M_x}) \quad (4.10)$$

图 4.4 说明了 CAMLM 跨语言注意力。如图所示，标记 h_2 关注句子 y 中的标记但不关注句子 x 的标记，而标记 h_5 和 h_6 只关注句子 x 中的标记而不关注句子 y 的标记。

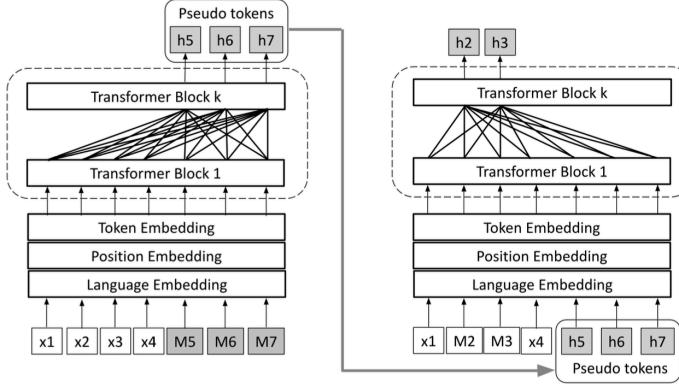


Figure 4.5 Illustration of BTMLM [194] pre-training task. The first step on the left is where a pre-trained CAMLM is used to generate pseudo-parallel sentences. The second step on the right is where the generated pseudo-parallel sentences are then used to further train the CAMLM.

4.5 BTMLM [98] 预训练任务的插图。左边的第一步是使用预训练的 CAMLM 生成伪并行句子。右边的第二步是使用生成的伪并行句子进一步训练 CAMLM。

反向翻译标记语言模型 (BTMLM) BTMLM [98] 利用反向翻译 [106] 来规避缺乏并行学习语料库的问题跨语言表示。它由两个步骤组成；第一步从给定的单语语料库生成伪并行数据。ERNIE-M [98] 首先使用 CAMLM 预训练模型，然后在原始单语句子的末尾添加占位符掩码以显示模型应生成的位置和语言，从而构建伪并行句子。第二步屏蔽原始单语句子中的标记，然后将其与创建的伪并行句子连接起来。最后，模型应该预测屏蔽标记。图 4.5 显示了 BTMLM 的两个步骤。

由于它们在 XLM 架构中的优越性能，mMLM 和 TLM 预训练任务也被用作训练 ERNIE-M 的一部分。ERNIE-M 使用 96 种语言的单语和并行语料库进行训练，并使用 XLM-R 权重进行初始化。ERNIE-M 在一系列不同的下游任务上表现出了最先进的性能，包括 NER、跨语言问答、跨语言释义识别和跨语言句子检索。

9. HITCL

(HICTL) 分层对比学习 (HICTL) [97] 是另一个使用基于 InfoNCE [105] 损失函数的跨语言预训练任务。尽管 XLCO 是基于句子的, HICTL 提供句子和单词级跨语言表示。

句子级表示的构建方式与 XCLLO 相同,除了负采样,而不是从 $\{y_j\}_{j=1, j \neq i}^N$ 收集样本平滑线性插值 [107, 108] 句子表示之间用于构造硬负样本。

对于句子级表示, 对比损失相似度得分在平行句子 (x_i, y_i) [CLS] 标记表示和换句话说。对于每个并行句子对输入 (x_i, y_i) , 维护一个词袋 \mathcal{W} , 其中 \mathcal{W} 中的所有单词被视为正样本, 词汇表中的所有其他单词被视为负样本。为了高效的否定词采样, HICTL 不是从整个词汇表中采样, 而是构造一组非常接近平行句子 (x_i, y_i) [CLS] 标记表示的否定词。

4.1.3 双编码器多语言 NLU

1. LaBSE

语言-不可知的 BERT 句子嵌入 (LaBSE) 与语言无关的 BERT 句子嵌入 (LaBSE) [109] 是一种用于训练跨语言句子表示的架构, 它结合了 Masked Language Model (MLM) 和翻译语言模型 (TLM) 预来自 XLM [94] 的训练任务以及使用带有附加边际 softmax loss [110] 的双向双编码器的翻译排名任务。如图 4.6 所示, 双编码器由两个成对的 mBERT 编码器组成。来自两个编码器的 [CLS] 标记句子表示被馈送到评分函数。图 4.6 提供了 LaBSE 架构的说明。

具有附加 Margin Softmax 的双向双编码器双编码器架构 [110] 已被证明对于排序任务非常有效, 例如 LaBSE 用于跨语言嵌入的翻译检索任务。形式上, 对于并行句子对 (x_i, y_i) , 翻译检索是一个排名问题, 旨在将 y_i (x_i 的真实翻译) 放置在翻译空间中高于所有句子的位置 $\mathcal{Y}P(y_i|x_i)$ 的公式为:

$$P(y_i|x_i) = \frac{e^{\phi(x_i, y_i)}}{\sum_{\bar{y} \in \mathcal{Y}} e^{\phi(x_i, \bar{y})}} \quad (4.11)$$

其中 ϕ 是 x_i 和 y_i 表示之间相似度的评分函数，在训练期间 $P(y_i|x_i)$ 近似为从同一批次的翻译对中采样负数 y_n ：

$$P_{approx}(y_i|x_i) = \frac{e^{\phi(x_i, y_i)}}{e^{\phi(x_i, y_i)} + \sum_{n=1, n \neq i}^N e^{\phi(x_i, y_n)}} \quad (4.12)$$

因此，对于并行源和目标对 (x_i, y_i) ，可以使用对数似然目标来优化模型 [110]：

$$\mathcal{L}_s = -\frac{1}{N} \sum_{i=1}^N \log \frac{e^{\phi(x_i, y_i)}}{e^{\phi(x_i, y_i)} + \sum_{n=1, n \neq i}^N e^{\phi(x_i, y_n)}} \quad (4.13)$$

对于每个 x_i ，损失 \mathcal{L}_s 旨在识别正确的 y_i 。这意味着使用 \mathcal{L}_s 将导致 x_i 和 y_i 表示之间的不对称关系。为了解决这个问题，Yand 等人 [110] 引入了双向学习目标 $\bar{\mathcal{L}}_s$ ，它明确优化了并行对 (x_i, y_i) 的前向和后向排序：

$$\mathcal{L}'_s = -\frac{1}{N} \sum_{i=1}^N \frac{e^{\phi(y_i, x_i)}}{e^{\phi(y_i, x_i)} + \sum_{n=1, n \neq i}^N e^{\phi(y_i, x_n)}} \quad (4.14)$$

$$\bar{\mathcal{L}}_s = \mathcal{L}_s + \mathcal{L}'_s$$

为了在正确的翻译和非常接近的最接近的非翻译之间增加分离，评分函数 ϕ 使用附加边距 softmax 进行扩展，它在正平行对 (x_i, y_i) 周围引入了边距 m 。

$$\phi'(x_i, y_j) = \begin{cases} \phi(x_i, y_j) - m & \text{if } i = j \\ \phi(x_i, y_j) & \text{if } i \neq j \end{cases} \quad (4.15)$$

将 $\phi'(x_i, y_i)$ 应用于双向损失 $\bar{\mathcal{L}}_s$ ，附加边际双向损失变为如下 [110]：

$$\mathcal{L}_{ams} = -\frac{1}{N} \sum_{i=1}^N \frac{e^{\phi(x_i, y_i) - m}}{e^{\phi(x_i, y_i) - m} + \sum_{n=1, n \neq i}^N e^{\phi(x_i, y_n)}} \quad (4.16)$$

$$\bar{\mathcal{L}} = \mathcal{L}_{ams} + \mathcal{L}'_{ams}$$

如上所述早些时候，LaBSE 分别使用单语和并行语料库通过 Masked Language Model (MLM) 和翻译语言模型 (TLM) 预训练任务来预训练其基于 BERT 的编码器。LaBSE 使用三阶段渐进堆叠方法来训练 transformer 编码器。对于 Llayer 编码器，它首先学习 fracL 4layers 的模型，然后学习 fracL 2layers，最后是所有 Llayers。在连续的训练步骤中，前面步骤模型的参数被复制到后续步骤中。

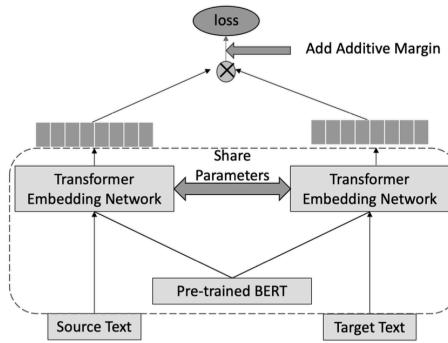


Figure 4.6 Illustration of Language-agnostic BERT Sentence Embedding (LaBSE) architecture [88].

图 4.6 与语言无关的 BERT 句子嵌入 (LaBSE) 架构的图示 [109]。

2. mUSE

多语言通用句子编码器 (mUSE) [111, 112] 提出了一种跨语言句子嵌入的方法通过结合单语言句子表示的多任务学习和用于并行文本检索的多语言句子嵌入学习的双编码器来进行学习。通过结合这两种方法，mUSE 将在单语言任务中学到的强大性能转移到其他语言的零样本学习中。基本上，对于特定语言对（源语言和目标语言），该模型使用多任务架构来学习 (i) 源语言单语任务，(ii) 单语目标语言任务，以及 (iii) 源语言到目标语言翻译任务作为桥接任务，鼓励所有训练任务之间的跨语言表示对齐。图 4.7 展示了 QA、NLI 和桥接翻译任务的架构。图 4.7 用于 QA、NLI 和桥接翻译任务的 mUSE 架构图解 [111]。

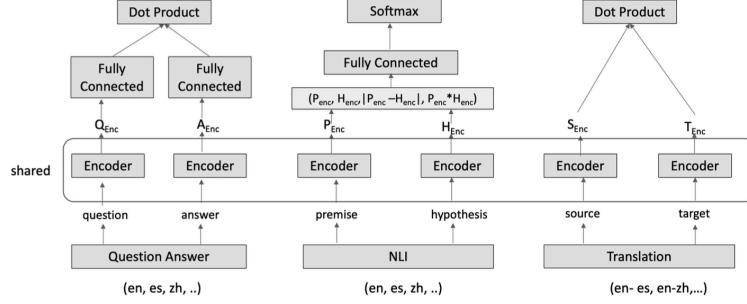


Figure 4.7 Illustration of mUSE architecture for QA, NLI and the bridging translation tasks [284].

图 4.7 用于 QA、NLI 和桥接的 mUSE 架构图示翻译任务 [111]。

mUSE 使用双编码器对可以表示为排序输入响应句子的任务执行多任务学习对。有些任务自然落入此框架内，例如 QA 和翻译任务。其他任务可能需要特殊的重新制定，例如 NLI。输入响应排序可以正式描述如下：对于输入响应对 (s_i^I, s_i^R) ，目标是使 s_i^R 高于所有可能的响应 $s_j^R \in \mathcal{S}^R$ 。换句话说，条件概率 $P(s_i^R | s_i^I)$ 应定义为：

$$P(s_i^R | s_i^I) = \frac{e^{\phi(s_i^I, s_i^R)}}{\sum_{s_j^R \in \mathcal{S}^R} e^{\phi(s_i^I, s_j^R)}} \quad (4.17)$$

$$\phi(s_i^R, s_j^R) = g^I(s_i^I)^\top g^R(s_j^R)$$

其中 g^I 是输入句子编码函数和 g^R 响应句子编码函数， g^I 和 g^R 组成双编码器。 g^I 和 g^R 都使用 transformer 实现，并针对每个任务使用对数似然损失函数 $\tilde{P}(s_i^R | s_i^I)$ 进行训练。

如图 4.7 所示，mUSE 使用单个共享编码器支持多个下游任务。训练任务包括：QA 预测任务、NLI 任务、翻译排序。在共享编码器之后添加了 QA 和 NLI 任务的特定于任务的层，以帮助满足每个任务的表征需求。

4.1.4 多语言 NLU

上一节讨论的所有跨语言预训练任务仅用于自然语言理解 (NLU) 任务。本节讨论 MASS 等生成多语言模型使用的一些预训练任务 [113]、BART [114]、mBART [115] 和 XNLG[116]。

序列到序列 LM (Seq2SeqLM) 如前所述, Masked Language Model (MLM) 处理了掩码标记上的 token 级分类问题, 其中原始单词通过将掩码标记向量馈送到词汇表上的 Softmax 层来预测。MLM 还用于预训练仅编码器的 Transformer 架构, 然后将其用于 NLU 任务。Seq2SeqLM 将传统的 MLM 扩展到预训练编码器-解码器变换器模型, 例如 T5 [68]、mT5[117] 和 MASS[113]。虽然 MLM 的上下文包括输入序列中的所有标记, 但 Seq2SeqLM 的上下文包括输入屏蔽序列中的所有标记, 并且预测目标序列中的左侧标记。以编码器的屏蔽序列作为输入, 解码器从左到右顺序预测屏蔽词。这使得使用 Seq2SeqLM 预训练的 Transformer 架构能够用于 NLG 任务。对于给定的标记 x 输入序列, 令 \hat{x} 为 x 的屏蔽版本, 即具有屏蔽 n 元语法跨度的 x 。将 l_s 定义为屏蔽 n 元语法跨度的长度, 则 Seq2SeqLM 损失定义为

$$\mathcal{L}_{Seq2SeqLM}^{(x)} = -\frac{1}{l_s} \sum_{s=i}^j \log P(x_s | \hat{x}, x_{i:s-1}) \quad (4.18)$$

去噪自动编码器 (DAE) 类似于 Seq2SeqLM, DAE[114, 115] 是编码器-解码器转换器架构的预训练任务。虽然 Seq2SeqLM 基于预测屏蔽标记, 但 DAE 并不屏蔽文本元素, 而是破坏它们, 然后重新构造原始值。DAE 通过标记删除和标记屏蔽在标记级别破坏文本, 通过句子排列突变在句子级别破坏文本, 通过文本填充在文本跨度级别破坏文本, 或者通过文档轮换在文档级别破坏文本。DAE 比 Seq2SeqLM 样本效率更高, 因为它为模型训练提供了更多信号。DAE 会生成更多训练信号, 因为它重建了完整的原始文本, 而 Seq2SeqLM 仅重建了屏蔽标记。BART [114] 使用双向编码器对损坏的输入序列进行编码, 并使用从左到右的解码器重建原始文本。类似于 Seq2SeqLM 的定义, 对于给定的输入文本 x , 令 \hat{x} 为 x 的损坏版本, 则 DAE 损失定义为

$$\mathcal{L}_{DAE}^{(x)} = -\frac{1}{|x|} \sum_{i=1}^{|x|} \log P(x_i | \hat{x}, x < i) \quad (4.19)$$

跨语言自动编码 (XAE) 虽然 DAE 在多种语言上进行训练，但编码输入和解码输出始终相同。结果，模型检测到语言符号和生成的短语之间的不正确相关性。换句话说，模型可能会忽略提供的语言符号并直接用输入语言创建句子。XNLG [116] 提供跨语言自动编码 (XAE) 工作来克服这个问题。与 DAE 不同，XAE 的编码输入和解码输出采用不同的语言，类似于机器翻译。此外，XNLG 对参数进行了两阶段优化。它最初使用 MLM 和 TLM 作业训练编码器。然后，在第二步中，它修复编码器并使用 DAE 和 XAE 任务训练解码器。该方法有效地预训练了所有参数，并弥合了 MLM 预训练和自回归解码微调之间的差距。

4.2 多语言数据

4.2.1 预训练数据

多语言语言模型 (MLM) 在预训练期间使用不同的数据源阶段。更具体地说，大型单语语料库通常用于个别语言，而平行语料库则用于某些语言之间。不同现有 MLM 的单语语料库的来源。例如，mBERT [26] 使用维基百科进行预训练。鉴于，在 [96] 中，Conneau 等人。使用更大的通用爬行语料库来训练 XLM-R。其他模型使用自定义的爬网数据。例如，IndicBERT[89] 接受了印度语言的自定义爬取数据的训练。表 4.1 提供了用于预训练不同 mLM 的数据集的摘要。

一些模型（例如 IndicBERT[89] 和 MuRIL[91]）被设计为仅支持一小部分语言。其他 MLM（例如 XLM-R）是大规模多语言的，可以支持 ~100 种语言。管理大量语言带来了用于不同语言预训练的数据量之间可能不平衡的挑战。例如，在使用 Wikipedia 和 CommonCrawl 进行预训练时，很明显，英语可用文章的数量远高于波斯语或乌尔都语等其他语言。类似地，可用于语言对的并行数据量在很大程度上取决于这些语言的流行程度。为了克服这些挑战，大多数 ML 在创建预训练数据时使用指数平滑的数据加权。这种权重可以防止低资源语言的代表性不足。更具体地说，如果总预训练数据

的 $m\%$ 属于语言 i , 则该语言的概率为 $p_i = k \cdot 100$ 。然后用指数因子 α 对每个概率进行调制, 然后对结果值进行归一化以提供语言上的最终概率分布。该概率分布用于对不同语言的预训练数据进行采样。因此, 低资源语言将被过度采样, 而高资源语言将被欠采样。这样的过程保证了在低资源语言中使用合理的词汇集 (同时训练 Word-Piece [118] 或 SentencePiece 模型 [102])。表 4.1 总结了不同 MLM 支持的语言数量和总词汇量被他们使用。通常, 支持更多语言的 MLM 具有更大的词汇量。

4.2.2 多语言基准

MLM 最常用的方法是下游任务的跨语言性能, 这需要使用来自高资源的特定于任务的数据来微调模型语言, 例如英语, 并在其他语言上对其进行评估。XGLUE [119]、XTREME [120]、XTREME-R [121] 是一些最先进的跨语言学习基准。如表 4.2 所示, 这些基准测试为各种任务和语言提供了训练/评估数据。这些任务可以分为: 分类、结构预测、问答和语义检索。

表 4.2 按 NLP 任务分类的多语言语言模型基准 [88]

task	Corpus	Train	Dev	Test	TestSets	#Langs.	Task	Metric	Domain	Benchmark
Classification	XNLI	392,702	2,490	5,010	Translations	15	NLI	Acc.	Misc.	XT, XTR, XG
	PAWS-X	49,401	2,000	2,000	Translations	7	Paraphrase	Acc.	Wiki/Quora	XT, XTR, XG
	XCOPA	33,410+400	100	500	Translations	11	Reasoning	Acc.	Misc.	XTR
	NC	100k	10k	10k	-	5	Sent.Labeling	Acc.	News	XG
	QADSM	100k	10k	10k	-	3	Sent.Relevance	Acc.	Bing	XG
	WPR	100k	10k	10k	-	7	Sent.Relevance	nDCG	Bing	XG
	QAM	100k	10k	10k	-	7	Sent.Relevance	Acc.	Bing	XG
Struct.Pred	UD-POS	21,253	3,974	47-20,436	Ind.annot.	37(104)	POS	F1	Misc.	XT, XTR, XG
	WikiANN-NER	20,000	10,000	1,000-10,000	Ind.annot.	47(176)	NER	F1	Wikpedia	XT, XTR
	NER	15k	2,8k	3,4k	-	4	NER	F1	News	XG
QA	XQuAD	87,599	34,736	1,190	Translations	11	Span Extraction	F1/EM	Wikpedia	XT, XTR
	MLQA			4,517-11,590	Translations	7	Span Extraction	F1/EM	Wikpedia	XT, XTR
	TyDiQA-GoldP	3,696	634	323-2,719	Ind.annot.	9	Span Extraction	F1/EM	Wikpedia	XT, XTRf
Retrieval	BUC	-	-	1,896-14,330	-	5	Sent.Retrieval	F1	Wiki/News	XT
	Tatoeba	-	-	1,000	-	33(122)	Sent.Retrieval	Acc.	Misc.	XT
	Mewsl-X	116,903	10,252	428-1,482	ind.annot.	11(50)	Lang.agn.retrieval	mAP@20	News	XTR
	LAReQA XQuAD-R	87,599	10,570	1,190	translations	11	Lang.agn.retrieval	mAP@20	Wikpedia	XTR

1. 分类

对于分类, 下游任务是将由单个句子或一对句子组成的输入分类为 k 类之一一个例子是自然语言推理 (NLI) 问题, 它要求输入是一对短语, 输出是三类之一: 暗示、中性或矛盾。XNLI [122]、PAWS-X [123]、XCOPA [124]、NC [119]、QADSM [119]、WPR [119] 和 QAM [119] 是一些著名的跨语言文本分类数据集用于评估 mLM。

2. 结构预测

该任务是预测输入句子中每个单词的标签。两个经常执行的任务是词性 (POS) 标记和命名实体识别 (NER)。WikiANN-NER [125]、CoNLL2002[126] 和 CoNLL 2003 [127] 共享任务数据集在 NER 中很受欢迎，而通用依赖关系数据集 [125] 用于词性标记。

3. 问答

QA 任务涉及从上下文和问题中提取响应范围。通常，训练数据只能用英语访问，但评估集可在多种语言中使用。XQuAD [128]、MLQA [129] 和 TyDiQA-GoldP [130] 数据集常用于 QA 评估。

4. 语义检索

语义检索涉及从给定源语言句子的句子集合中找到目标语言中匹配的句子。该任务使用以下数据集进行评估：BUCC [131]、Tateoba [132]、Mewsli-X [121] 和 LAReQA XQuAD-R [133]。Mewsli-X 和 LAReQA XQuAD-R 被认为更具挑战性，因为它们涉及从多语言句子池中匹配目标语言的句子恢复。

4.3 多语言迁移学习见解

4.3.1 零样本跨语言学习

多语言迁移学习的最终目标之一是创建具有与监督单语模型相称的零样本跨语言性能的 MLM。创建零样本跨语言模型的标准工作流程需要使用多种语言来预训练 amLM，其中 MLM 学习跨语言表示。然后使用来自源语言的特定于任务的监督数据集对该模型进行微调。最后，使用微调后的模型对来自多种不同目标语言的同一任务进行推理。Transformer 在跨语言迁移学习方面取得了显着的进步，特别是在零样本学习方面。变形者如何进行跨语言迁移学习的细节仍然是一个活跃的研究领域。本节讨论活跃的研究领域，以了解影响跨语言迁移学习的主要因素 [88]。讨论的因素包括：(i) 数据相关因素，(ii) 模型架构因素，以及 (iii) 训练任务因素。

1. 数据因素

Shared Vocabulary 在训练 MLM 之前，使用 WordPiece 对所有语言的文本进行标记 [118] 或 SentencePiece[102] 模型。主要概念是将每个单词标记为其最常出现的子词。然后，模型的词汇表由所有语言中可识别的所有这些子词组成。这种组合的标记化确保了子词在大量密切相关甚至遥远的语言之间共享。因此，如果目标语言测试集中存在的子词也出现在源语言训练集中，则某些模型性能可能会通过这个公共词汇来传递。在 [134] 和 [135] 中研究了跨语言零样本可迁移性与语言之间词汇重叠大小之间的关系，发现了很强的正相关性，并且对下游任务的积极影响在跨语言中是一致的。几个不同的领域，包括 NER [126, 127]、POS [125]、XNLI [122]、MLDoc[136] 和依赖解析 [125]。这些观察结果很直观，并为使用单语语料库执行多语言训练时的跨语言表示对齐提供了一些解释。然而，这一领域的研究仍然是新的，并且仍在检查其他证据以验证这些观察结果。例如，Karthikeyan 等人 [137] 综合地将词汇重叠减少到零，但是在跨语言零样本学习中观察到的性能下降很小，这削弱了词汇重叠对跨语言零样本的观察到的影响。另一项工作 [128] 表明，即使没有词汇重叠，跨语言迁移学习仍然可以通过适当微调除输入嵌入层之外的所有转换器层来进行。

预训练数据大小 预训练数据大小对交叉语言的影响 [138] 中对语言进行了评估，表明当在大语料库上对 mBERT 进行预训练时，跨语言迁移得到了改善。[139] 表明零样本传输的性能与目标语言中用于为更高级别任务（例如 XNLI 和 XQuAD）预训练 MLM 的数据量密切相关。对于较低级别的任务（如 asPOS、依存句法分析和 NER）也存在积极的联系，尽管它不如 XNLI 和 XQuAD 的联系那么强。

使用并行语料库 虽然 MLM 即使没有经过跨语言信号的专门训练也能表现良好，但直观地说，利用此类信号进行有目的的训练应该会提高表现。事实上，XLMSin [94] 和 InfoXLM [95] 证明，将并行语料库与 TLM 预训练任务结合使用可以提高性能。如果没有可用的平行语料库，[140] 认为使用相似的语料库（例如 Wikipedia 或 CommonCrawl）训练 mLm 比使用跨语言的不同来源的语料库更好。

2. 模型架构因素

多语言模型的容量取决于其层数，注意头的数量以及隐藏表示的大小。Karthikeyan 等人。[137] 证明跨语言迁移学习的性能高度依赖于网络的深度，而不太依赖于注意力头的数量。他们特别证明，即使单头注意，足够的跨语言迁移也是可能的。此外，模型中参数的总数对跨语言迁移效率的影响小于层数。有趣的是，在 [140] 中，mBERT 的跨语言迁移被认为是其参数集较小且容量有限的结果，这迫使它使用共享结构来对齐跨语言的表示。另一个关键的架构决策是自注意力上下文窗口，它指的是在训练期间给予 MLM 的标记数量。Liu 等 [138] 证明尽管使用较小的上下文窗口更有效当预训练数据稀疏时，使用较长的上下文窗口会更好，当有大量预训练数据可用时，使用较长的上下文窗口会更好。在 [141] 和 [142] 中，有人认为，由于 MLM 的模型容量有限，可供多种语言使用，因此它们不能像预训练的单语言模型一样捕捉多种语言的所有微妙之处。他们展示了单语言模型的知识蒸馏如何增强 MLM 的跨语言性能。

3. 模型任务因素

微调策略在 [143] 中讨论了微调 amLM 改变其参数，通过擦除部分损害其跨语言能力预训练期间学到的对齐方式。他们通过证明当 MLM 针对词性标注进行微调时，其跨语言检索性能显着下降，从而证明了这一点。为了克服这个问题，他们建议利用持续学习框架来微调模型，以便它不会忘记训练它的原始任务（MLM）。他们声称使用这种微调方法可以改善跨语言 POS 标记、NER 和句子检索的结果。

表示对齐在 [144, 145] 中，使用通过学习的隐式对齐表示来检查零样本跨语言迁移的性能 MLM 和单语言模型的表示随后使用并行语料库进行显式对齐。他们指出，显式对齐可以提高性能。考虑到这一点，王等人。[146] 提供了一种明确的策略，用于在 mBERT 训练期间跨语言对齐匹配单词对的表示。这是通过包含一个损失函数来实现的，该函数可以最小化对齐单词嵌入之间的欧几里得距离。Zhao 等人。当词对的表示显式对齐并且向量空间进一步标准化时，[147] 还报告了可比较的结果。

4.3.2 与语言无关的跨语言

尽管 BERT 等 Transformer 模型在推进 NLP 领域和机器学习方面取得了巨大成功，但此类模型如何提取数据的内部工作原理而商店知识目前还没有被完全理解，是积极研究的焦点。多语言 Transformer 当然也不例外，但更有趣的是，它们似乎可以跨多种语言使用和共享知识；即跨语言迁移学习。这就提出了一个问题：多语言转换器是否学习与语言无关的跨语言表示。人们提出了几种方法来回答这个问题，每种方法都有不同的攻击角度。

(i) 消融研究，(ii) 任务探针，以及 (iii) 并行语料库表示。

消融研究已经提出了几种消融研究来检验关于多语言转换器中与语言无关的表示的有效性的几种假设。^[134] 中测试的第一个假设是，高资源语言之间的联合脚本是良好的多语言转换器性能的混杂因素。然而，这被证明是正确的，因为多语言迁移发生在不共享脚本的语言之间，例如用阿拉伯脚本编写的乌尔都语和用 De-vanagari 脚本编写的印地语^[134]。其他工作^[148] 将输入标记化作为混杂因素进行了检查，发现使用子词标记化比单词级或字符级标记化更能学习与语言无关的表示。预训练任务也被认为是一个可能的混杂因素，结果表明，使用并行语料库预训练任务（例如 XLM）训练的模型比在单语言语料库（例如 mBERT 和 XLMR^[149] ）。

并行语料库表示实现与语言无关的表示假设的另一种方法是使用不同语言的并行语料库，并检查它们的并行表示在模型嵌入空间中的对齐情况。典型相关分析（CCA）是一种用于分析并行表示对齐的技术。在^[148] 中，CCA 分析表明 mBERT 不会在同一空间上投影并行表示。机器翻译也被用来检查与语言无关的表示，一方面提供源语言的句子，另一方面提供目标语言中的多个句子，任务是使用最接近的邻居算法来识别目标候选语言中的正确翻译。在^[134] 中可以看出，正确的平移（即表示对齐）取决于模型中生成表示的层。结果表明，mBERT 中间层 5-8 对于来自同一语系的语言（例如英语-德语和印地语-乌尔都语）产生了 75% 的最近邻精度。有人认为，一个正确的结论是，多语言转换器模型的大容量允许创建和保留与语言无关和特定于语言的表示。与语言无关的表示足够丰富，可以正确对齐多语言单词并检索语义相似的句子。然而，这些表示还不足以解决机器翻译等难题^[150]。

探测任务研究与语言无关的表示的另一种方法是对在各个层获取的表

示使用探测任务。例如，一致的依赖树可以从 mBERT 中指示句法抽象的中间层的表示中学习 [151, 125]。然而，主谓宾 (SVO) 语言（例如英语、法语和印尼语）的依存关系树比 SOV 语言（例如土耳其语、韩语和日语）的依存关系树更准确。SOV 和 SVO 语言之间的这种差异也可以在 POS 的标签中看到 [134]。每一层都有独特的特性，因此混合来自多个层的数据以获得最佳结果是有利的，而不是根据整体性能选择单个层，正如 Dutch 在各种 NLU 任务中所证明的那样 [152]。在同一个实验中，我们发现，与单语言荷兰语模型相比，多语言模型在早期层的词性标注方面具有更多的信息表示。

4.4 案例研究

4.4.1 目标

在本节中，我们提供了一个训练多语言情感分类器的实际示例。我们的目标是说明多语言模型的零样本分类能力，其中模型仅训练英语数据，然后用于预测非英语数据而无需进一步训练。

4.4.2 数据、工具和库

对于本案例研究，我们使用二元情感分类 Yelp Polarity 数据集 [153]。该数据集包含 560K 个用于训练的高极性 Yelp reviews 和 38K 个用于测试的评论。Yelp 原始评论的评分为 1 到 5 星。该数据集是通过将 1 星和 2 星评论分组为负面情绪类，将 3 星和 4 星评论分组为积极情绪类来构建的。我们的模型将使用多语言通用句子编码器 (mUSE) [112, 111] 进行特征生成。mUSE 是经过训练的 Transformer 编码器，使得不同语言但具有相似含义的文本将产生相似的编码。这类似于具有相似含义（和用法）的两个单词将具有相似的单词嵌入。mUSE 支持 16 种语言：阿拉伯语、简体中文、繁体中文、英语、法语、德语、意大利语、日语、韩语、荷兰语、波兰语、葡萄牙语、西班牙语、泰语、土耳其语、俄语。在本案例研究中，我们将使用用于加载 mUSEmodel 的 TensorFlow Hub、用于加载 Yelp Polarity 数据集的 Huggingface Datasets 以及用于使训练变得更简单的 Py-Torch Lightning。mUSE 内部使用 TensorFlow Text 进行标记化，因此我们也安装了它。可以通过运行清单 4.1. 中所示的 shell 命令来安装它们。

```
pip install tensorflow_hub tensorflow_text>=2.0.0rc0
pytorch_lightning==1.4.7 datasets==1.12.1
```

清单 4.1 Python 环境设置

4.4.3 实验、结果和分析

一旦环境设置完毕，可以加载编码器，如清单 4.2 所示。我们还定义了一个简单的函数，它接受字符串列表作为输入并返回 Numpy 数组列表。请注意，为了清楚起见，我们包含了 Python 3 的类型注释。不必使用它们。

```
import tensorflow_hub as hub
import tensorflow_text
import numpy as np
from typing import List, Dict
model_URL =
    "https://tfhub.dev/google/universal-sentence-encoder-multilingual-large/3"
encoder = hub.load(model_URL)

def embed_text(text: List[str]) -> List[np.ndarray]:
    "Encode text and convert TensorFlow tensors to Numpy arrays"
    vectors = encoder(text)
    return [vector.numpy() for vector in vectors]
```

清单 4.2 加载通用句子编码器

1. 数据预处理

为了使模型训练变得简单，本例研究使用 PyTorchLightning。PyTorch Lightning 让您实现 LightningDataModule 类，该类封装了所有数据加载、预处理和批处理。我们展示了如何对清单 4.3 中的 Yelp Polarity 数据集执行此操作。我们使用完整训练集和测试集的一小部分来将模型训练时间保持在最低限度。我们将其作为练习留给读者，以扩展到更大的子集或整个数据集。

```
import pytorch_lightning as pl
import torch
from torch.utils.data import DataLoader
from datasets import Dataset, load_dataset
# set seed (this is optional)
pl.seed_everything(445326, workers=True)

class YelpDataModule(pl.LightningDataModule):
    def __init__(self, batch_size: int = 32, num_workers: int = 2):
        super().__init__()
        self.batch_size = batch_size
        self.num_workers = num_workers
        self.pin_memory = torch.cuda.is_available()

    def prepare_data(self):
        """
        This method loads a subset of the train and test sets.
        It uses the first 2% of the train and test sets to
        train and test, respectively.
        It uses the last 1% of the training set for validation.
        """

```

```

        self.test_ds = load_dataset('yelp_polarity', split="test[:2%]")
        self.train_ds = load_dataset('yelp_polarity', split="train[:2%]")
        self.val_ds = load_dataset('yelp_polarity', split="train[99%:]")

        # Map class labels to an integer
        self.label_names = self.train_ds.unique("label")
        label2int = {str(label): n for n, label in
                     enumerate(self.label_names)}
        self.encoder = encoder_factory(label2int)

def setup(self):
    # Compute embeddings in batches, to speed things up
    self.train = self.train_ds.map(self.encoder,
                                   batched=True, batch_size=self.batch_size)
    self.train.set_format(type="torch", columns=["embedding", "label"],
                          output_all_columns=True)
    self.val = self.val_ds.map(self.encoder, batched=True,
                               batch_size=self.batch_size)
    self.val.set_format(type="torch", columns=["embedding", "label"],
                        output_all_columns=True)
    self.test = self.test_ds.map(self.encoder, batched=True,
                                batch_size=self.batch_size)
    self.test.set_format(type="torch", columns=["embedding", "label"],
                         output_all_columns=True)

def train_dataloader(self):
    return DataLoader(self.train, batch_size=self.batch_size,
                     num_workers=self.num_workers,

```

```
pin_memory=self.pin_memory, shuffle=True)

def val_dataloader(self):
    return DataLoader(self.val, batch_size=self.batch_size,
                      num_workers=self.num_workers,
                      pin_memory=self.pin_memory)

def test_dataloader(self):
    return DataLoader(self.test, batch_size=self.batch_size,
                      num_workers=self.num_workers)

def encoder_factory(label2int: Dict[str, int]):
    "Returns a function that encodes each text example and each label"
    def encode(batch):
        batch["embedding"] = embed_text(batch["text"])
        batch["label"] = [label2int[str(x)] for x in
                         batch["label"]]
        return batch
    return encode
```

清单 4.3 加载模型和分词

2. 实验

接下来，我们定义清单 4.4 中的模型架构。

```
import torch
```

```

import torch.nn as nn
import torch.nn.functional as F
from datasets import load_metric
class Model(pl.LightningModule):
    def __init__(self,
                 hidden_dims: List[int] = [768, 128],
                 dropout_prob: float = 0.5,
                 learning_rate: float = 1e-3):
        super().__init__()
        self.train_acc = load_metric("accuracy")
        self.val_acc = load_metric("accuracy")
        self.test_acc = load_metric("accuracy")
        self.hidden_dims = hidden_dims
        self.dropout_prob = dropout_prob
        self.learning_rate = learning_rate
        self.embedding_dim = 512
        layers = []
        prev_dim = self.embedding_dim
        if dropout_prob > 0:
            layers.append(nn.Dropout(dropout_prob))

    for h in hidden_dims:
        layers.append(nn.Linear(prev_dim, h))
        prev_dim = h
        if dropout_prob > 0:
            layers.append(nn.Dropout(dropout_prob))
        layers.append(nn.ReLU())
        if dropout_prob > 0:
            layers.append(nn.Dropout(dropout_prob))

```

```

# output layer
layers.append(nn.Linear(prev_dim, 2))

self.layers = nn.Sequential(*layers)

def forward(self, x):
    # x will be a batch of USEm vectors
    logits = self.layers(x)
    return logits

def configure_optimizers(self):
    optimizer = torch.optim.Adam(self.parameters(),
        lr=self.learning_rate)
    return optimizer

def __compute_loss(self, batch):
    "Runs the forward pass and computes the loss"
    x, y = batch["embedding"], batch["label"]
    logits = self(x)
    preds = torch.argmax(logits, dim=1).detach().cpu().numpy()
    loss = F.cross_entropy(logits, y)
    return loss, preds, y

def training_step(self, batch, batch_idx):
    "Computes forward pass, loss, and
    logs metrics for a training batch"

```

```

        loss, preds, y = self.__compute_loss(batch)
        self.train_acc.add_batch(predictions=preds, references=y)
        acc = self.train_acc.compute()["accuracy"]
        values = {"train_loss": loss, "train_accuracy": acc}
        self.log_dict(values, on_step=True, on_epoch=True,
                      prog_bar=True, logger=True)
        return loss

    def validation_step(self, batch, batch_idx):
        "Computes forward pass, loss, and logs metrics
         for a validation batch"
        loss, preds, y = self.__compute_loss(batch)
        self.val_acc.add_batch(predictions=preds, references=y)
        acc = self.val_acc.compute()["accuracy"]
        values = {"val_loss": loss, "val_accuracy": acc}
        self.log_dict(values, on_step=True, on_epoch=True,
                      prog_bar=True, logger=True)
        return loss

    def test_step(self, batch, batch_idx):
        "Computes forward pass, loss, and logs
         metrics for a test batch"
        loss, preds, y = self.__compute_loss(batch)
        self.test_acc.add_batch(predictions=preds, references=y)
        acc = self.test_acc.compute()["accuracy"]
        values = {"test_loss": loss, "test_accuracy": acc}
        self.log_dict(values, on_step=False, on_epoch=True,
                      prog_bar=True, logger=True)
        return loss

```

清单 4.4 多语言二元分类器架构

现在模型已经定义好了，我们可以加载数据，然后训练和评估模型。我们将训练最多五个时期，将在训练期间监视验证损失，将保存包含最低验证损失的三个模型检查点（每个时期一个）。检查点将存储在名为“model”的目录中。这些选项通过 ModelCheckpoint 回调传递给 PyTorch Lightning 的 Trainer 对象。

```
data = YelpDataModule()
data.prepare_data()
data.setup()
print(len(data.train)) # >> 11200
print(len(data.val)) # >> 5600
print(len(data.test)) # >> 760
model = Model()
MAX_EPOCHS = 5

checkpoint_callback = pl.callbacks.ModelCheckpoint(
    monitor="val_loss", dirpath="model",
    filename="yelp-sentiment-multilingual-{epoch:02d}"
    "-{val_loss:.3f}", save_top_k=3,
    mode="min")

# Create the Trained, and use a GPU (if available)
# It is best to train this model with a GPU.
if torch.cuda.is_available():
    trainer = pl.Trainer(gpus=1, max_epochs=MAX_EPOCHS,
        callbacks=[checkpoint_callback])
else:
    trainer = pl.Trainer(max_epochs=MAX_EPOCHS,
```

```
callbacks=[checkpoint_callback])

# Train the model, passing it the train and
# validation data loaders
trainer.fit(model, data.train_dataloader(),
data.val_dataloader())
# Test the model
trainer.test(test_dataloaders=data.test_dataloader())
#>> [{}'test_accuracy': 0.8644737005233765,
    'test_loss': 0.32756760716438293}]
```

清单 4.5 加载数据、训练和评估模型

我们已经训练了一个模型，现在让我们尝试一下。在清单 4.6 中，我们定义了一个函数，使用验证损失最低的 modelcheckpoint 来预测输入文本的情绪。

```
best_model = Model.load_from_checkpoint(
    checkpoint_callback.best_model_path)
def predict(text: List[str]):
    embeddings = torch.Tensor(embed_text(text))
    logits = best_model(embeddings)
    preds = torch.argmax(logits, dim=1).
        detach().cpu().numpy()
    scores = torch.softmax(logits, dim=1).
        detach().cpu().numpy()
    results = []
    for t, best_index, score_pair in zip
        (text, preds, scores):
        results.append({"text": t, "label":
```

```

    "positive" if best_index == 1 else
    "negative", "score": score_pair[best_index]})

return results

predict(["I love that restaurant!",
         "I hate italian food."])
#>> [{"label": 'positive', "score": 0.99751616,
        "text": 'I love that restaurant!'},
       # {"label": 'negative', "score": 0.9791407,
        "text": 'I hate italian food.'}]

```

清单 4.6 加载最佳模型并运行推理

因为我们使用了 USEm 嵌入，我们应该能够预测非英语语言的情绪。让我们尝试一下。如前所述，USEm 支持 16 种语言：阿拉伯语、简体中文、繁体中文、英语、法语、德语、意大利语、日语、韩语、荷兰语、波兰语、葡萄牙语、西班牙语、泰语、土耳其语、俄语。在清单 4.7 中，我们比较了语言对之间的情绪预测，发现即使我们的模型是在 Yelp 极性训练集的一小部分子集上进行训练的，它仍然可以表现良好。我们还发现该模型可以对至少一种不属于 16 种受支持语言的语言进行准确预测。我们在清单 4.7 中使用四种语言的输入文本：英语、德语、意大利语和芬兰语。

```

from pprint import PrettyPrinter
pp = PrettyPrinter()

# English vs. German
english_text = "Our server was horrid. He messed up the order and didn't even ap-
                he spilled wine on my sister's hair!"
german_translation = "Unser Server war schrecklich. Er hat die Bestellung durche-

```

gebracht und sich nicht einmal entschuldigt, als er Wein in die Haare meiner Schwester verschüttet hat!"

```
pp pprint(predict(best_model, [english_text, german_translation]))  
#>> [{"label": "negative",  
# "score": 0.9564845,  
# "text": "Our server was horrid. He messed up the order and "  
# "didn't even apologize when he spilled wine on my "  
# "sister's hair!"},  
# {"label": "negative",  
# "score": 0.9694613,  
# "text": "Unser Server war schrecklich. Er hat die Bestellung "  
# "durcheinander gebracht und sich nicht einmal "  
# "entschuldigt, als er Wein in die Haare meiner "  
# "Schwester verschüttet hat!"}]  
# English vs. Italian & Finnish  
english_text = "My least favorite film is Showgirls. I hate it so much. In fact,  
bad that it makes me angry."  
italian_translation = "Il mio film meno preferito è Showgirls. Lo odio così tanto  
effetti, e così brutto che mi fa arrabbiare."  
finnish_translation = "Minun lempi elokuva on Showgirls. Vihaan sitä niin paljon  
asiassa se on niin paha, etta se saa minut vihaiseksi."  
pp pprint(predict(best_model, [english_text,  
italian_translation, finnish_translation]))  
#>> [{"label": "negative",  
# "score": 0.98994666,  
# "text": "My least favorite film is Showgirls. I hate it so much."  
# "In fact, it's so bad that it makes me angry."},  
# {"label": "negative",  
# "score": 0.974451,
```

```
# "text": "Il mio film meno preferito e Showgirls. Lo odio così "
# "tanto. In effetti, è così brutto che mi fa arrabbiare."},
# {"label": "negative",
# "score": 0.7616636,
# "text": "Minun lempi elokuva on Showgirls. Vihaan sitä niin paljon."
# "Itse asiassa se on niin paha, etta se saa minut vihaiseksi."}]
```

清单 4.7 加载最佳模型并运行推理

多语言 Transformer 架构 107USEm even 在芬兰工作。但是为什么？如果不深入研究，就很难确定。我们的猜测是，在训练过程中，USEm 标记化中使用的子词单元让 Transformer 学习跨语言使用哪些子词单元，类似于 wav2vec2 [154] 中的语音子单元。我们添加到 USEm 上的层经过分类训练，然后让模型学习哪些子词单元与积极或消极情绪相关。也许芬兰语中使用的子词单位与 USEm 支持的 16 种语言之一中的子词单位足够接近。

5 Transformer Block 修改

对 Attention Is All You Need [44] 中介绍的 Transformer 架构进行了许多修改。正如第 1.2.1 节中所讨论的，修改有两种主要类型：对 Transformer 块本身的组织的更改以及对 Transformer 块内部子模块的更改。本章讨论 1.2.1.1 节中 Transformer 块更改的具体示例以及 1.2.1.2. 中 Transformer 子模块更改的具体示例。

5.1 Transformer Block 修改

在本节中，我们将详细介绍具有更改的 Transformer 块的几个模型。

5.1.1 轻量级 Transformer

1. Funnel-Transformer

Funnel-Transformer [29] 在传递到下一层之前通过池化 transformer 编码器层的输出进行压缩，从而降低计算成本。然后，它使用节省的计算来支持更深或更宽的模型，从而增加模型容量。

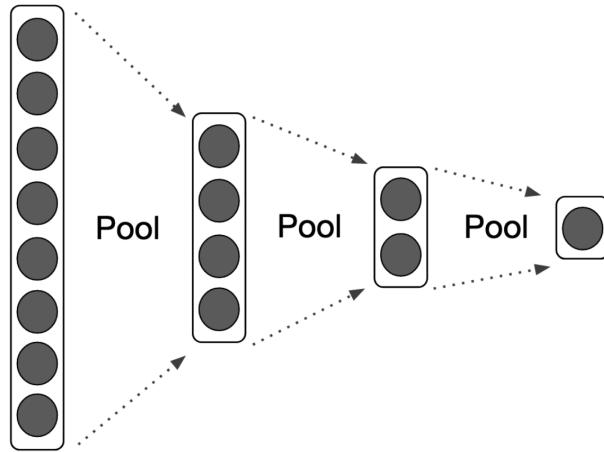


Figure 5.1 Schematic architecture diagram for Funnel-Transformer’s encoder. Each layer represents a block composed of several transformer layers with the same sequence length. Shows three pooling operations between blocks, with each decreasing the sequence length of the output by half.

图 5.1 Funnel-Transformer 编码器的架构示意图。每层代表一个由多个具有相同序列长度的 TransformerLayer 组成的块。显示块之间的三个池化操作，每个池化操作之前将输出的序列长度减少一半。

编码器标准转换器对所有层使用相同的序列长度。Funnel-Transformer 通过在 transformer 层之间放置一个池化层来改变这一点，以减少序列长度。它通常在相同序列长度的块中具有多个在池化操作之前，它通常在具有相同序列长度的块中具有多个 transformer 层，如图 5.1 所示。如果给定层的输出为 \mathbf{h} ，则池化层的输出为 $h' = \text{Pooling}(h)$ ，其中 $h \in \mathbb{R}^{T \times d}$ 和 $h' \in \mathbb{R}^{T' \times d}$ ，对于一些 $T' < T$ 。

\mathbf{h}' 用于构造自注意力块的查询和残差连接， \mathbf{h} 用于键和值向量：

$$\mathbf{Q} = \mathbf{h}' \mathbf{W}_Q, \in \mathbb{R}^{T' \times k} \quad (5.1)$$

$$\mathbf{K} = \mathbf{h}' \mathbf{W}_K, \in \mathbb{R}^{T \times k} \quad (5.2)$$

$$\mathbf{V} = \mathbf{h}' \mathbf{W}_V, \in \mathbb{R}^{T \times v} \quad (5.3)$$

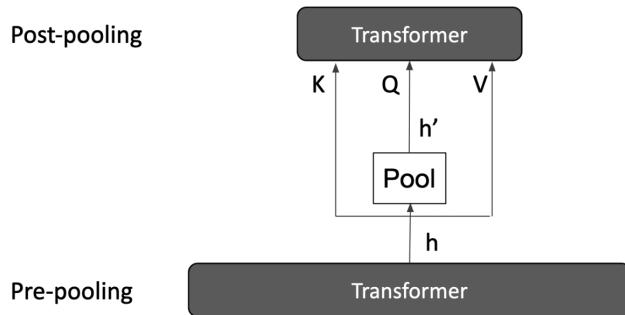


Figure 5.2 Shows how the pooling operation between Funnel-Transformer's encoder layers affect the input of the next layer. \mathbf{h} is the output of the layer before the pooling and \mathbf{h}' is the output of the pooling operation. The query matrix for the next layer is constructed from the pooled output, \mathbf{h}' . The key and value matrices for the next layer are made from the unpooled output, \mathbf{h} .

图 5.2 显示了 Funnel-Transformer 编码器层之间的池化操作如何影响下一层的输入。他是池化之前层的输出， \mathbf{h} /primei 是池化操作的输出。下一层的查询矩阵是根据池化输出 \mathbf{h}' 构造的。下一层的键和值矩阵由未池化的输出 \mathbf{h} 组成。

非池化和池化输出与下一层的查询、键和值矩阵之间的关系如图 5.2 所示。 $(n + 1)^{st}$ 层的输出为

$$\mathbf{h}^{(n+1)} = \text{LayerNorm}(\mathbf{h}'^{(n)} + \text{multihead}(\mathbf{Q}(\mathbf{h}'^{(n)}, \mathbf{K}(\mathbf{h}^{(n)}, \mathbf{V}(\mathbf{h}^{(n)}))) \quad (5.4)$$

每个注意力头的注意力权重矩阵为 $(T' \times T)$ ，每个连续层的复杂度递减。多头注意力的输出与 \mathbf{h}' 具有相同的维度。

通过从池化序列和非池化序列的键和值构造查询，注意力机制尝试学习池化和非池化序列应如何最好地相互关注产生高质量的压缩。Funnel-Transformer 使用步幅和窗口大小均设置为 2 的均值池。

解码器为了支持模型需要生成完整输出序列的标记级预测任务（例如机器翻译），Funnel-Transformer 有一个可选的解码器，可以对压缩后的数据进行上采样编码器输出为完整序列长度。M 解码器层将具有长度 $T_M = T/2^{M-1}$ 的输出序列 $\mathbf{h}^{(M)}$ 。通过重复每个隐藏向量 2^{M-1} 次，它将一步上采样到 $h^{(up)} = [h_1^{(up)}, \dots, h_T^{(up)}]$ ：

$$h_i^{up} = h_{i/2^{N-1}}^{(M)}, \forall i = 1, \dots, T \quad (5.5)$$

$$x//y = \text{floor}(x/y) \quad (5.6)$$

为了解决 Funnel-Transformer 上采样过程中的重复信息，第一编码器层的隐藏状态输出被添加到上采样表示中： $\mathbf{g} = \mathbf{h}^{(up)} + h^{(1)}$ 。这充当剩余连接。然后，他们在 \mathbf{g} 之上添加一些转换器层，以使其学习如何最好地组合特征。

缩放漏斗变换的时间复杂度为 $O(d \cdot T^2 + T \cdot d^2)$ 。由于 T 在连续的编码器层上减少了一半，因此复杂度降低了一个因子每层四个。由于 $O(T \cdot d^2)$ 具有较大的常数 d^2 ，因此它往往占主导地位，提供线性加速，而不是二次加速。由于层之间的池化降低了复杂性，因此可以添加额外的编码器层或使现有层更宽，而不会以任何显着的方式增加计算负载。

性能将三种尺寸的标准 transformer 与 Funnel-Transformer 的几种配置进行了比较，每种配置与所比较的 transformer 相比，每种尺寸的预期浮点运算数量较少或相似：

大：24 层，d= 1024 基础：12 层，d= 768 小：6 层，d= 768

针对 GLUE 进行质量比较，当 Funnel-Transformer 减少序列长度并添加更多层时，它的性能优于文本分类和所有 GLUE 数据集 (STS-B)

除外) 的标准转换器。当序列长度减少但深度没有增加时, GLUEtext 分类数据集的性能会下降。

2. DeLightT

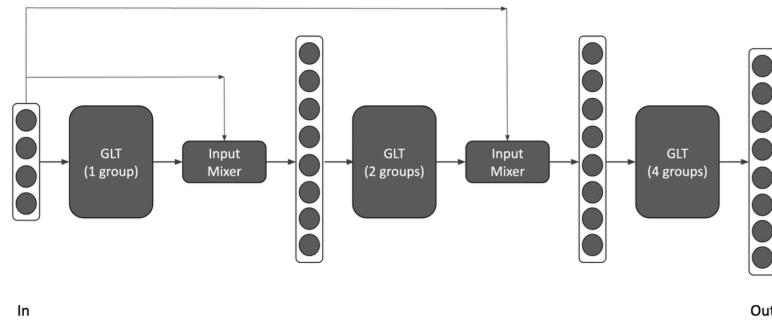


Figure 5.3 The expansion phase of the DeLight transformation, showing the three grouped linear transformations (GLT).

图 5.3 DeLight 变换的扩展阶段, 显示了三组线性变换 (GLT)。

DeLightT [30] 是一种改进的 Transformer 架构, 在机器翻译和语言建模方面的性能与标准 Transformer 一样; 同时使用更少的参数和 FLOPs。它引入了对 Transformer 块的输入的转换, 该转换发生在投影到查询、键、值空间之前。**DeLightT 模块** DeLightT 模块使用 N 层分组线性变换 (GLT) [155] 首先变换 d 维在第一个 $N/2$ 层中将向量转换为 $d_{max} = w_m d_{in}$ 的高维空间, 然后使用其他 $N - N/2$ 层将 d_{max} 向量转换为低维空间。 d_o 向量是随后在查询、键和值空间中投影的内容。靠近模型输出的 DeLightT 块比靠近模型输入的块更宽、更深。DeLightT 使用 $d_o = d/2$ 并且还对 GLT 中各组之间的特征进行了调整, 使用混合器连接 [156] 将调整后的特征与输入组合起来, 类似于剩余连接, DeLightT 块的扩展阶段如图 5.3 所示。

Mehta 等人。提出通过增加 DeLightT 块的深度及其中间 GLT 层的宽度, transformer 将具有增加的表示能力, 然后可以用单头注意力代替多头注意力。同样, 他们提出 DeLightT 块的宽线性层可以将前馈层的大小减少多达 16 倍。

性能尽管参数少得多，但 DeLighT 在机器翻译方面的表现与标准 Transformer 一样甚至更好。但是，当 DeLighT 被赋予更多参数时，它的性能优于标准 transformer。它还在机器翻译方面获得了与 SOTA 模型相似或更好的质量。此外，性能随着 DeLighT 块中网络参数数量的增加而提高。

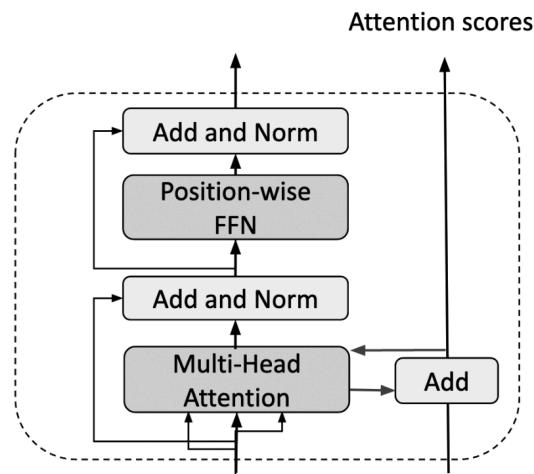


Figure 5.4 RealFormer’s residual attention connection.

图 5.4 RealFormer 的剩余注意力连接。

5.1.2 Transformer 之间的连接

1. RealFormer

RealFormer，剩余注意力层 transformer [31]，将剩余分数添加到原始注意力逻辑中（查询来自 transformer 前一层的所有注意力头的关键点积）。注意力权重就是注意力逻辑总和的 softmax。这在图 5.4 中示意性地示出，在 (5.7)–(5.9) 中以数学方式示出。

$$ResidualMultiHead(\mathbf{Q}, \mathbf{K}, \mathbf{V}, \mathbf{Prev}) = concat(head_1, \dots, head_h) \mathbf{W}_o \quad (5.7)$$

$$head_i = ResidualAttn(\mathbf{Q}_i, \mathbf{K}_i, \mathbf{V}_i, \mathbf{Prev}_i) \quad (5.8)$$

$$ResidualAttn(\mathbf{Q}', \mathbf{K}', \mathbf{V}', \mathbf{Prev}') = \text{softmax}\left(\frac{\mathbf{Q}'\mathbf{K}'^T}{\sqrt{(d_k)}} + \mathbf{Prev}'\right)\mathbf{v}' \quad (5.8)$$

其中 $\mathbf{Prev} \in \mathbb{R}^{h \times d_{in} \times d_{out}}$ 是来自前一个 Transformer 层的注意力逻辑。 d_{in} 是层的输入序列维度， d_{out} 是层的输出序列维度。Softmax 的参数（注意力逻辑或注意力分数）被传递到下一个 RealFormerlayer。

此方法可以应用于其他 Transformer 架构，包括解码器层。当 transformer 中存在多种类型的注意力模块时，可以跳过边缘以支持每种类型的注意力的一个残差连接。

性能 RealFormer 通常比标准 Transformer 架构表现更好，包括它在 BERT 中的使用，所有这些都无需增加模型参数的数量。

5.1.3 自适应计算时间

1. 通用 Transformer

如前文所示在各章中，Transformers 克服了与 RNN 相关的问题，即顺序计算的瓶颈和众多问题中的梯度消失问题。此外，自注意力（Transformer 中的关键创新）有助于并行化每个符号基于上下文的向量的计算，并创建一个全局感受野，让符号从所有符号中获取信息。另一方面，当解决具有固有层次结构的任务时，或者当训练和模型预测的未见数据之间的长度显着变化时，RNN 不存在循环归纳偏差就成为一个问题。此外，transformer 中的顺序计算数量与输入大小无关，而仅取决于层数，这使得它在计算上不通用或图灵不完整。Transformers 对

所有输入应用相同数量的计算，导致在许多情况下效率低下，其中计算可以以复杂性为条件。

Universal Transformers (UT) by Dehghani et al. [33] 是 Transformers 的扩展，其中并行性和全局感受野优势通过 RNN 的循环归纳偏差得到补充，同时具有计算通用性。通用变换器没有固定数量的变换器层，而是具有通用变换器块，即一个自注意力机制，后跟一个循环变换，为每个输入符号并行提供电流感应偏差。如图 5.5 所示，通用变换器是一个循环函数，不是在时间上而是在深度上，它演化对应的隐藏状态。并行地处理每个输入，每一步都基于先前隐藏状态的序列。

UT 与现有的神经架构有许多共同点，例如神经 GPU [157] 和神经图灵机 [158]。它也可以被证明相当于一个多层 transformer，其各层都有相关的参数。Graves 提出了自适应计算时间 (ACT)，它允许 RNN 动态学习在接受输入和发出输出（思考时间）之间要采取多少计算步骤，克服每个符号固定数量的计算步骤的问题。受 Graves ACT 启发，UT 在每个位置都采用了动态 ACT 停止机制，以根据复杂性调节每个符号的计算。研究表明，UT 在各种 NLP 和 NLU 上都优于标准 Transformer，并在 LAMBADA 语言建模等复杂任务中达到了新的最先进水平。

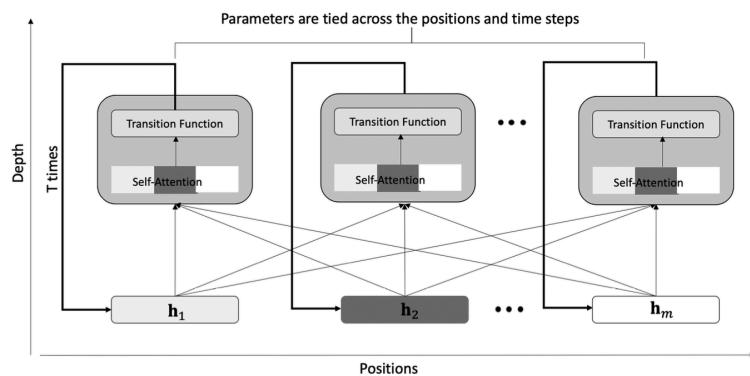


Figure 5.5 Universal Transformer with information from different positions using self-attention and applying a recurrent transition function.

图 5.5 通用 transformer，其中包含来自不同位置的信息，使用自注意力并应用循环转换函数。

5.1.4 Transformer Blocks 之间的循环关系

1. Transformer-XL

Transformer-XL [36] 对超出固定长度的依赖关系进行建模，同时尊重句子边界。引入它是因为标准 Transformer 架构的固定宽度上下文窗口阻止它学习在固定窗口之外的范围内对依赖项进行建模。Transformer-XL 可以处理比标准 Transformer 长 450% 的依赖关系，并且推理速度比 Transformer 快 1800 倍。

段级递归 Transformer-XL 通过使用称为段级递归的方法将其上下文扩展到固定窗口之外。递归的工作原理是在处理当前文本段时使用前一个文本段作为附加上下文。为了 Transformer Modifications 117 要工作，必须缓存前一个段以及该段每层的 Transformer 输出。在标准 Transformer 中，第 n 个 Transformer 层将前一层 (n-1) 的输出作为输入：

$$\mathbf{h}_t^{(n)} = \text{Transformer}(\mathbf{h}_t^{(n-1)}) \quad (5.10)$$

并从 $\mathbf{h}_t^{(n-1)}$ 内部生成查询、键和值：

$$\begin{aligned} \mathbf{Q}_t^{(n)} &= h_t^{(n-1)} \mathbf{W}_q \\ \mathbf{K}_t^{(n)} &= h_t^{(n-1)} \mathbf{W}_k \\ \mathbf{V}_t^{(n)} &= h_t^{(n-1)} \mathbf{W}_v \end{aligned} \quad (5.11)$$

注意，n=1 被视为第一个 transformer 层，n=0 是第一个 transformer 层的输入，所以 $h_t^{(0)} = \mathbf{X}$ 。

Eqns (5.11) 当包含前一段时发生变化。考虑两个连续的文本段，它们嵌入了表示 \mathbf{X}_t 和 \mathbf{X}_{t+1} ，其中仅用于表示段顺序。当计算当前段的

第 n 个 transformer 层的输出 \mathbf{X}_{t+1} 时，我们有来自 $\mathbf{h}_t^{(n-1)}$ 的贡献，它是前一个段的前一个 transformer 层的输出，

$$\mathbf{h}_{t+1}^{(n)} = \text{TransformerXL}(\mathbf{h}_{t+1}^{(n-1)}, \mathbf{h}_t^{(n-1)}) \quad (5.12)$$

计算当前段的键和值时，会出现对前一段的依赖关系：

$$\begin{aligned} \mathbf{Q}_{t+1}^{(n)} &= \mathbf{h}_{t+1}^{(n-1)} \mathbf{W}_q \\ \mathbf{K}_{t+1}^{(n)} &= \tilde{\mathbf{h}}_{t+1}^{(n-1)} \mathbf{W}_k \\ \mathbf{V}_{t+1}^{(n)} &= \tilde{\mathbf{h}}_{t+1}^{(n-1)} \mathbf{W}_v \end{aligned} \quad (5.13)$$

$$\tilde{\mathbf{h}}_{t+1}^{(n-1)} = [\text{StopGradient}(\mathbf{h}^{(n-1)_t}); \mathbf{h}_{(t+1)}^{(n-1)}] \quad (5.14)$$

其中 StopGradient 表示在操作过程中不计算梯度。等式 (5.13) 和 (5.14) 表明注意力机制被用来计算修改后的注意力，该注意力结合了先前输入序列的信息来计算当前输入序列的表示。然而，事情还不止于此。当前序列的 transformer 输出取决于前一个序列的 transformer 的输出

表 5.1 在层深为三时，第四段 transformer 的输出有来自第一段的贡献。为简洁起见，上面所示的 T 是 (5.12)

Layer(n)	S_1	S_2	S_3	S_4
0(input)	$h_1^{(0)} = S_1$	$h_2^{(0)} = S_2$	$h_3^{(0)} = S_3$	$h_4^{(0)} = S_4$
1	$h_1^{(1)} = T(h_1^{(0)})$	$h_2^{(1)} = T(h_2^{(0)}, h_1^{(0)})$	$h_3^{(1)} = T(h_3^{(0)}, h_2^{(0)})$	$h_4^{(1)} = T(h_4^{(0)}, h_3^{(0)})$
2	$h_1^{(2)} = T(h_1^{(1)})$	$h_2^{(2)} = T(h_2^{(1)}, h_1^{(1)})$	$h_3^{(2)} = T(h_3^{(1)}, h_2^{(1)})$	$h_4^{(2)} = T(h_4^{(1)}, h_3^{(1)})$
3	$h_3^{(2)} = T(h_1^{(2)})$	$h_2^{(2)} = T(h_2^{(2)}, h_1^{(2)})$	$h_3^{(3)} = T(h_3^{(2)}, h_2^{(2)})$	$h_4^{(3)} = T(h_4^{(2)}, h_3^{(2)})$

前一个序列的 transformer 输出取决于该序列之前的序列和前一个 transformer 层。这意味着随着转换器层数的增加，模型可以处理的

有效上下文大小也会增加。形式上，我们可以说第 t 个段上第 n 个 transformer 层的输出有来自早至 $t-n$ 的段的贡献。表 5.1 中显示了一个示例，其中包含四个文本段和三个 Transformer-XL layer。

Transformer-XL 中的位置编码位置编码正如第 1 章中所讨论的。由于 Transformer 不包含循环层或卷积层，因此词序是通过位置编码显式内置的。每个输入序列都被赋予一个位置编码， $\mathbf{P} \in \mathbb{R}^{L \times d}$ ，其中 L 是最大序列长度， d 是嵌入维度，以及词嵌入 $\mathbf{E} \in \mathbb{R}^{L \times d}$ 。注意，完整的输入序列由 $\mathbf{X} = \mathbf{E} + \mathbf{P}$ 表示；这与本章中使用的 \mathbf{X} 这是本章中使用的相同 \mathbf{X} ，例如，如 (5.22) 所示。

\mathbf{P} 的第 i 行， \mathbf{p}_i 中所示，包含标记在第 i 个位置编码。类似地， e_i 是位置 i 中标记的词嵌入。由于位置编码是确定性的并且独立于序列中的标记，因此位置 i 中的任何单词都将具有相同的位置编码。这引入了段级递归的问题。

考虑两个连续的文本序列 \mathbf{X}_{t+1} 和 \mathbf{X}_t ，以及每个序列的第一个 TransformerXL 层的输出。(5.12) 告诉我们

$$\mathbf{h}_{t+1}^{(1)} = \text{TransformerXL}(\mathbf{X}_{t+1}, \mathbf{X}_t) \quad (5.15)$$

(5.14) 表示

$$\tilde{\mathbf{h}}_{t+1}^{(0)} = [\text{StopGradient}(\mathbf{X}_t; \mathbf{X}_{t+1})] \quad (5.16)$$

由于 $\mathbf{X}_t = \mathbf{E}_t + \mathbf{P}$ 并且位置编码与序列中的标记无关，因此 $\tilde{\mathbf{h}}_{t+1}^{(0)}$ 是连接位置编码的相同副本。这是有问题的，因为 $\tilde{\mathbf{h}}_{t+1}^{(0)}$ 表示扩展的有效上下文大小，因此序列中的每个位置都应该具有不同的位置编码。如果没有不同的位置编码，模型就会丢失有关词序的信息。这个问题的解决方案是使用相对位置编码。

相对位置编码早期的工作 [159, 160] 引入了相对位置编码作为一种让注意力机制从序列中两个位置之间的距离中学习的方法，而不是使用绝对位置编码来自标准转换器的编码，它使注意力机制偏向于认为绝

对位置是重要的。相反，有关 \mathbf{q}_i 和 \mathbf{k}_i 之间相对距离的信息将被纳入注意力权重计算中，使注意力机制偏向于将距离 $i-j$ 视为重要量，而不是位置 i 和 j 。

TransformerXL 修改了注意力矩阵计算以使用正弦相对位置编码 [36]。我们可以通过首先展开 (5.23) 中 $\mathbf{Q}\mathbf{K}^T$ 的 (i,j) 项， $\mathbf{q}_i\mathbf{k}_j^T$ 来了解如何实现。回顾 (5.22)、(5.24) 和 (5.25)，我们可以看到 \mathbf{X} 的第 i 行是 $\mathbf{x}_i = \mathbf{e}_i + \mathbf{p}_i$ ，并且

$$\begin{aligned}\mathbf{q}_i &= \mathbf{x}_i \mathbf{W}_q \\ \mathbf{k}_j &= \mathbf{x}_j \mathbf{W}_k\end{aligned}\tag{5.17}$$

利用 (5.17)，我们将 $\mathbf{q}_i\mathbf{k}_j^T$ 展开为

$$\begin{aligned}A_{ij} &= (\mathbf{e}_i \mathbf{W}_q + \mathbf{p}_i \mathbf{W}_q)(\mathbf{W}_k^T \mathbf{e}_j^T + \mathbf{W}_k^T \mathbf{p}_j^T) \\ &= \mathbf{e}_i \mathbf{W}_q \mathbf{W}_k^T \mathbf{e}_j^T + \mathbf{e}_i \mathbf{W}_q \mathbf{W}_k^T \mathbf{p}_j^T + \mathbf{p}_i \mathbf{W}_q \mathbf{W}_k^T \mathbf{e}_j^T + \mathbf{p}_i \mathbf{W}_q \mathbf{W}_k^T \mathbf{p}_j^T\end{aligned}\tag{5.18}$$

Transformer XL 重新参数化五个新量的 A_{ij} 项： \mathbf{R}_{i-j} , \mathbf{u} , \mathbf{v} , $\mathbf{W}_{k,E}$ 和 $\mathbf{W}_{k,R}$ [36]

1. \mathbf{R}_{i-j} 是距离 $i-j$ 的标记对的相对位置编码，并替换 (5.18) 中的 \mathbf{p}_j 。
 \mathbf{R}_{i-j} 是相对位置编码矩阵 $\mathbf{R} \in \mathbb{R}^{L \times d}$ 的第 $(i-j)$ 行
2. \mathbf{u} 替换 (5.18) 的第三项中的 $\mathbf{p}_i \mathbf{W}_q$

- (a) \mathbf{v} 替换 (5.18). 的第四项中的 $\mathbf{p}_i \mathbf{W}_q$ 。
- (b) TransformerXL 用两组密钥替换了密钥 K：基于内容的密钥和基于位置的密钥。此更改导致将密钥权重矩阵 \mathbf{W}_k 替换为两个权重矩阵 $\mathbf{W}_{k,E}$ 和 $\mathbf{W}_{k,R}$ ，其中 $\mathbf{W}_{k,E}$ 生成基于内容的密钥， $\mathbf{W}_{k,R}$ 生成基于位置的密钥 [36]。因此，(5.18) 变为

$$A_{ij} = \mathbf{e}_i \mathbf{W}_q \mathbf{W}_{k,E}^T \mathbf{e}_j^T + e_i \mathbf{W}_q \mathbf{W}_{k,R}^T \mathbf{R}_{i-j}^T + \mathbf{u} \mathbf{W}_{k,E}^T \mathbf{e}_j^T + \mathbf{v} \mathbf{W}_{k,R}^T \mathbf{R}_{i-j}^T \quad (5.19)$$

请注意，通过消除显式位置依赖性， $\mathbf{x}_i = \mathbf{e}_i$ 意味着 $\mathbf{X} = \mathbf{E}$ ，因此 $\mathbf{q}_i = \mathbf{e}_i \mathbf{W}_q$ 和 $\mathbf{k}_j = \mathbf{e}_j \mathbf{W}_{k,E}$ ，

使用 (5.19)，我们得到第 n 个模型层的 Transformer-XL 注意力机制的最终形式

$$\mathbf{Attn}_t^{(n)} = \text{softmax} \left(\frac{\mathbf{A}_t^{(n)}}{\sqrt{d_k}} \right) \mathbf{V}_t^{(n)} \quad (5.20)$$

其中 $\mathbf{A}_t^{(n)}$ 是 (5.19) 按照 (5.13) 的规定修改，

$$A_{t,ij}^{(n)} = \mathbf{q}_{t,i}^{(n)} \mathbf{k}_{t,j}^{(n)T} + \mathbf{q}_{t,i}^{(n)} \mathbf{W}_{k,R}^{(n)T} \mathbf{R}_{i-j}^T + \mathbf{u} \mathbf{k}_{t,j}^{(n)T} + \mathbf{v} \mathbf{W}_{k,R}^{(n)T} \mathbf{R}_{i-1}^T \quad (5.21)$$

请注意，softmax 分母中的总和是查询涉及的关键位置集 S_i 上的屏蔽总和，如图所示 (5.27) 中。另外，由于 $\mathbf{x} = \mathbf{e}_i$ ， $\mathbf{h}_t^{(0)} = \mathbf{X}_t$ 。

5.1.5 分层 Transformer

分层 transformer 的两个例子是用于图像识别的 Vision Transformer[161] 和用于 TimeSformer [162] 视频分类/动作识别。这两种模型都将在第 6.5.2 章修改多头自注意力的 transformer 中介绍。

5.2 改良 Multi-HEAD 自注意力 Transformer

5.2.1 多头自注意力的结构

多头注意力是一种组合多种注意力机制的方法，可以让模型学习输入和输出之间的依赖关系的不同类型。多头注意力机制是每个 Transformer 块的核心。编码器和解码器块都使用多头自注意力，解码器块具有第二个多头注意力，它关注编码器块的输出，并具有适当的因果约束。

在本节中，我们将描述 L tokens 的单个输入序列的多头自注意力计算，嵌入维度为 d。输入序列可以用矩阵 $\mathbf{X} \in \mathbb{R}^{L \times d}$ 表示

$$\mathbf{x} = \begin{bmatrix} \mathbf{x}_1 \\ \vdots \\ \mathbf{x}_n \end{bmatrix} \quad (5.22)$$

其中 $\mathbf{x}_i = (x_0^{(i)}, \dots, x_{d-1}^{(i)})$ 是第 i 个 token 的嵌入向量。正如我们在 2.4.2.1 节中看到的，注意力机制的输出（在头连接之前）可以用

$$\text{Attn}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d_k}}\right)\mathbf{V} \quad (5.23)$$

其中 \mathbf{Q} , \mathbf{K} , \mathbf{V} 分别是查询矩阵、键矩阵和值矩阵。每个矩阵都是将输入序列变换到不同向量空间的结果：

$$\begin{aligned} \mathbf{Q} &= \mathbf{X}\mathbf{W}_q, \in \mathbb{R}^{L \times d_k} \\ \mathbf{K} &= \mathbf{X}\mathbf{W}_k, \in \mathbb{R}^{L \times d_k} \\ \mathbf{V} &= \mathbf{X}\mathbf{W}_v, \in \mathbb{R}^{L \times d_v} \end{aligned} \quad (5.24)$$

其中 d_k 是查询和键空间的维度，通常设置为 d, d_v 为值维度。矩阵 \mathbf{W}_q 、 $\mathbf{W}_k \in \mathbb{R}^{d \times d_k}$ 和 $\mathbf{W}_v \in \mathbb{R}^{d \times d_v}$ 基本上是旋转矩阵。查询/键/值矩阵的每一行对应于第 i 个标记的查询/键/值向量：

$$\mathbf{Q} = \begin{bmatrix} \mathbf{q}_1 \\ \vdots \\ \mathbf{q}_L \end{bmatrix}, \mathbf{K} = \begin{bmatrix} \mathbf{k}_1 \\ \vdots \\ \mathbf{k}_L \end{bmatrix}, \mathbf{V} = \begin{bmatrix} \mathbf{v}_1 \\ \vdots \\ \mathbf{v}_L \end{bmatrix} \quad (5.25)$$

请注意，(5.24) 可以适用于长度分别为 L_1 和 L_2 的两个序列 \mathbf{X}_1 和 \mathbf{X}_2 之间的多头注意力的情况。

对于两个序列，查询矩阵由 \mathbf{X}_2 构成，键值矩阵由 \mathbf{X}_2 构成：

$$\begin{aligned}\mathbf{Q} &= \mathbf{X}_1 \mathbf{W}_k, \in \mathbb{R}^{L_1 \times d_k \times h} \\ \mathbf{K} &= \mathbf{X}_2 \mathbf{W}_k, \in \mathbb{R}^{L_2 \times d_k \times h} \\ \mathbf{V} &= \mathbf{X}_2 \mathbf{W}_v, \in \mathbb{R}^{L_2 \times d_v \times h}\end{aligned}\tag{5.26}$$

其中 $\mathbf{X}_1 \in \mathbb{R}^{L_1 \times d}$ 和 $\mathbf{X}_2 \in \mathbb{R}^{L_2 \times d}$ 。这通常是 transformer 解码器块中发生的情况。 $\mathbf{X}_1 \in \mathbb{R}^{L_1 \times d}$

(5.23) 的 softmax 部分是注意力权重矩阵 A_{ij} :

$$A_{ij} = \frac{\exp(\mathbf{q}_i \mathbf{k}_j^T)}{\sum_{r \in S_i} \exp(\frac{\mathbf{q}_i \mathbf{k}_r^T}{\sqrt{d_k}})}\tag{5.27}$$

其中 S_i 是查询 \mathbf{q}_i 可以关注的关键位置集合.

1. 多头自注意力

到目前为止，我们只讨论了单头自注意力。多头注意力主要是将上面所示的矩阵划分为 h 个片段，其中 h 是注意力头的数量。

每个注意力头都有自己的查询/键/值，这些查询/键/值是通过将单头版本分解为相同大小的片段而获得的，这些片段被索引令 $n = 1, \dots, h$:

$$\begin{aligned}\mathbf{Q}_n &= \mathbf{X} \mathbf{W}_n^{(q)}, \in \mathbb{R}^{L \times d_k/h} \\ \mathbf{K}_n &= \mathbf{X} \mathbf{W}_n^{(k)}, \in \mathbb{R}^{L \times d_k/h} \\ \mathbf{V}_n &= \mathbf{X} \mathbf{W}_n^{(v)}, \in \mathbb{R}^{L \times d_v/h}\end{aligned}\tag{5.28}$$

这并不意味着我们现在有了 h 查询、键和值矩阵，而是 (5.28) 中所示的矩阵是 (5.24) 中所示矩阵的一部分。图 5.6 中的查询明确显示了这一点。

键和值矩阵以与 \mathbf{Q} 类似的方式划分为注意力头，第 n 个头的注意力计算按预期进行：

$$\text{Attn}^{(n)}(\mathbf{Q}_n, \mathbf{K}_n, \mathbf{V}_n) = \text{softmax} \left(\frac{\mathbf{Q}_n \mathbf{K}_n^T}{\sqrt{d_k/h}} \right) \mathbf{V}_n\tag{5.29}$$

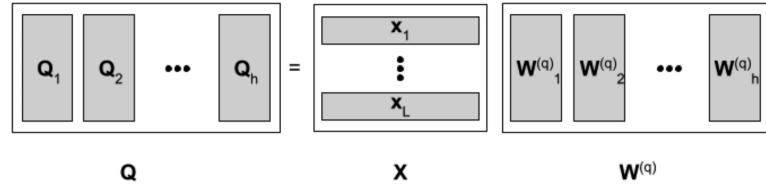


Figure 5.6 The query matrix, \mathbf{Q} , can be partitioned into h heads, as described in (5.28).

图 5.6 查询矩阵 \mathbf{Q} 可以划分为 h 个 head，如 (5.28) 中所述。

请注意，对于多头情况， $\mathbf{Q}_n \mathbf{K}_n^T$ 除以 $\sqrt{d_k/h}$ 而不是 $\sqrt{d_k}$ 。这一变化说明了查询有效维度和键空间到 d_k/h 的变化。然后如 (2.19) 中所述组合注意力头。

2. 空间和时间复杂度

计算第 5.27 节中描述的注意力权重矩阵需要 $O(L^2 \times d_k)$ 矩阵乘法，计算第 5.23 节中的上下文向量需要 $O(L^2 \times d_v)$ 矩阵乘法，因此 self-attention 的时间复杂度为 $O(L^2 \times d_k + L^2 \times d_k)$ 。

考虑 L 个 tokens 的单个输入序列，并且查询、键和值共享相同的维度，因此 $d_k = d_v = d_{model}$ 。这意味着 \mathbf{Q} 、 \mathbf{K} 、 \mathbf{V} 是 $L \times d_{model}$ 模型矩阵，(5.23) 中的注意力权重矩阵是 $L \times L$ 。假设 32 位浮点数，内存使用量增长很快，如表 5.2 所示。包含批量大小时需要更多内存。例如，如果批量大小为 32，序列长度为 20,000，则需要 51.2 GB 来存储自注意力权重。时间复杂度在 L 中也是二次方。随着序列的长度增加，这种缩放变得令人望而却步 [53]。例如，如果序列长度加倍，计算和存储注意力权重所需的时间将增加四倍。

L	Memory
600	4MB
1K	1.4MB
20K	1.6GB
64K	16.384GB

5.2.2 降低自注意力的复杂性

本节讨论了几种降低多头自我注意的复杂性的 Transformer 模型。

1. Longformer

在计算自注意力时（忽略编码器和解码器块之间自注意力的因果要求），通常对序列中的哪些位置可以相互关注没有限制。这意味着，原则上，每个头的注意力权重矩阵可能是密集的。当将其视为图时，它对应于完全连接的加权二分图。如果序列有 L 个标记，那么就会有 $L(L-1)/2$ 个边。Longformer [50] 通过根据特定模式限制哪些位置可以相互关注来改变这一点。这导致所有头的注意力权重稀疏，相当于从注意力图中删除边缘。

Longformer 将全局注意力与两种类型的短程注意力（滑动窗口和扩张滑动窗口）结合起来。每个注意力模式对应于一种类型的注意力掩码。

滑动窗口注意力序列中的每个标记都被赋予一个固定大小的上下文窗口 w ，而不是能够关注序列中的任何标记，因此它只能关注其自身的上下文窗口。邻居。例如，如果我们有九个标记的序列 (w_1, \dots, w_9) ，并且窗口大小 $w=4$ ，则中间标记 w_5 的上下文将向右 (w_3, w_4) 和向左 (w_6, w_7) 各扩展两个标记。如图 5.7 所示。

在图形视图中，这意味着每个顶点最多链接到 $w-1$ 个顶点。因此，这种变化将边的数量从 $L(L-1)/2$ 减少到 $L(w-1)$ 。这个简单的改变将复杂性从序列长度的二次函数降低到序列长度的线性函数 $O(Lw)$ ，因为滑动窗口注意力权重矩阵在每行（或列）中将具有 Lw 非零值。

$$\begin{bmatrix} \bullet & \bullet & \bullet & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \bullet & \bullet & \bullet & \bullet & \cdot & \cdot & \cdot & \cdot & \cdot \\ \bullet & \bullet & \bullet & \bullet & \bullet & \cdot & \cdot & \cdot & \cdot \\ \cdot & \bullet & \bullet & \bullet & \bullet & \bullet & \cdot & \cdot & \cdot \\ \cdot & \cdot & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \cdot \\ \cdot & \cdot & \cdot & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet \\ \cdot & \cdot & \cdot & \cdot & \bullet & \bullet & \bullet & \bullet & \bullet \\ \cdot & \cdot & \cdot & \cdot & \cdot & \bullet & \bullet & \bullet & \bullet \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \bullet & \bullet & \bullet \end{bmatrix}$$

Figure 5.7 Sliding window attention pattern, for $L = 9$ and $w = 4$. Row i corresponds to query i . Columns with a \bullet are keys that query i attends to and \cdot represents a lack of attention (a missing edge).

图 5.7 滑动窗口注意模式，对于 $L= 9$ 且 $w= 4$ 。Row i 对应于查询 i 。带有 \bullet 的列是查询我关注的键，并且 \cdot 表示缺乏关注（缺少边缘）。

扩张的滑动窗口注意力这种注意力模式通过在上下文窗口中添加 d 大小的间隙来扩展滑动窗口注意力的广度。这实际上扩展了上下文窗口的有效大小。例如，在上面使用的九个标记序列中，假设我们添加了一个扩张 $d= 2$ 。在生成 w_5 的上下文时，我们现在会跳过中间的一个单词。因此， w_5 的左上下文将是 (w_1, w_3) ，右上下文将是 (w_7, w_9) 。请注意，即使整体上下文宽度已经增加，这种注意模式也会阻止标记关注其直接邻居，或距窗口中心距离 $d-1$ 的任何标记，或它关注的窗口内的任何位置。不同的注意力头可以使用不同的 d 值，这可以缓解上述问题。图 5.8 显示了 $L= 2$ 、 $w= 4$ 和 $d= 2$ 时的扩张滑动窗口注意力。

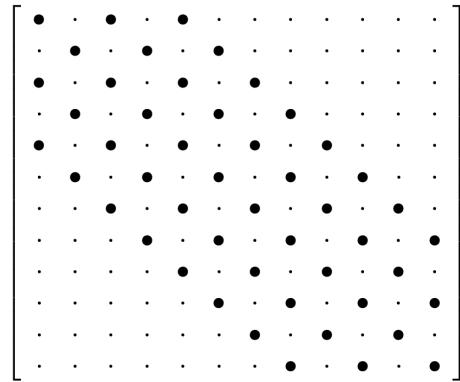


Figure 5.8 Dilated sliding window attention pattern, for $L = 12$, $w = 4$, and $d = 2$. Row i corresponds to query i . Columns with a • are keys that query i attends to and · represents a lack of attention (a missing edge).

图 5.8 扩张滑动窗口注意力模式， $L= 12$, $w= 4$, $d= 2$ 。Rowic 对应于查询 i。具有“是查询 i 关注的键”和“表示缺乏关注（缺失边）”的列。全局注意力全局注意力模式选择让一些标记关注序列中的任何其他标记。在这种情况下，序列中的所有标记都会关注该标记。这对应于选择注意力权重矩阵的特定行及其转置列。

Longformer 根据训练任务决定允许哪些标记具有全局注意。例如，问题和答案任务，问题中的所有标记都具有全局注意。具有全局注意力的标记数量通常与序列长度无关，因此全局注意力与序列长度也是线性的。Longformer 将全局注意力与滑动窗口注意力结合起来。

通过为全局和滑动窗口模式提供自己的查询、键和值矩阵，分别计算全局和短程注意力模式的权重。分别为 \mathbf{Q}_g 、 \mathbf{K}_g 、 \mathbf{V}_g 和 \mathbf{Q}_s 、 \mathbf{K}_s 、 \mathbf{V}_s

Longformer 使用较小的窗口大小较低层和较高层的较大窗口尺寸。这赋予了较高层分层性质。事实上，扩张的滑动窗口仅用于较高层，以便较低层可以专注于局部上下文。

2. Reformer Reformer，在 [53] 中介绍，通过修改注意力机制来解决注意

力机制的复杂性，并通过使用可逆残差网络。这些技巧让 Reformer 包含比 Transformer 大几个数量级的上下文窗口（最多 1,000,000 个单词）。

注意力和局部敏感散列正如 5.2.1.1 节中所讨论的，作为 Transformer 核心的缩放点积注意力的时间复杂度为 $O(L^2)$ ，随着序列 L 中标记数量的增加，这变得令人望而却步。Reformer 模型通过使用局部敏感哈希修改注意力机制来解决这个问题。这将时间复杂度更改为 $O(L \log L)$ 。

回想一下，在缩放点积注意力中，查询、键和值矩阵是将 d 模型维度输入向量矩阵转换为维度 d_k 的查询和键以及维度 d_v 的值的结果。

在 A 的方程中，计算成本较高的项是乘积 $\mathbf{Q}\mathbf{K}^T$ ，此外，一旦应用了 softmax 函数，只有沿每个 d_{model} 维度的最大项才重要。这意味着对于 \mathbf{Q} 中的每个查询向量，我们只需要 \mathbf{K} 中最接近它的键。为了使这更容易，他们设置 $\mathbf{Q} = \mathbf{K}$ ，这意味着对于每个查询向量，我们只需要找到最接近的查询。这是一个近似最近邻问题，因此我们可以使用局部敏感哈希 (LSH)。

局部敏感哈希局部敏感哈希 (LSH) 于 1998 年在 [163] 中引入，作为一种基于哈希的近似相似性搜索方法。用正式术语来说，LSH 基于一系列哈希函数 \mathcal{F} ，这些函数对项目集合进行操作，其中，如果有任何两个这样的项目 x 和 y ，则 $Prob_{h \in \mathcal{F}}/[h(x) = h(y)] = sim(x, y)$ ，其中 $sim(x, y) \in [0, 1]$ 在项目集合上定义的相似函数中 [164]。或者，换句话说，每当您可以对数据集中的项目进行散列时，您就有一个 LSH 方法，这样当这些项目靠近时，任何两个项目的碰撞概率比它们相距较远时要高得多。

有多种方法可以实现 LSH 计划。Reformer 使用 [165] 中定义的角度 LSH 方案。回想一下，查询矩阵 \mathbf{Q} （和密钥矩阵 \mathbf{K} ）的形状为 $l \times d_k$ 。要计算 \mathbf{Q} 的哈希值，首先生成一个随机 $d_k \times b/2$ 矩阵 \mathbf{R} ，其中每列都有一个从 ± 1 均匀采样的非零值，并且 b 是哈希值的数量。接下来，计算 \mathbf{QR} ，它旋转查询（和密钥）矩阵。最后，哈希为 $h(Q) = \arg \max([QR; -QR])$ [53]。一旦查询和键被散列，您就可以排

列索引，以便散列到同一存储桶中的序列位置并排排列。然后，在每个哈希桶内，计算完整的注意力。

我们可以使用此信息来查看 LSH 如何影响注意力计算，方法是首先重写自注意力方程 (5.23)，使用 (5.27)：

$$a_i = \sum_{j \in S_i} \exp \left(\frac{\mathbf{q}_i \mathbf{k}_j^T}{\sqrt{d_k}} - \log Z(i, S_i) \right) \mathbf{v}_j, \quad (5.30)$$

其中 S_i 是查询 i 关注的关键位置集合， $Z(i, S_i) = \sum_{r \in S_i} \exp \left(\frac{\mathbf{q}_i \mathbf{k}_r^T}{\sqrt{d_k}} \right)$ 是 softmax 归一化项。

在不失一般性的情况下，我们可以将 (5.30) 重写为一组扩展的关键位置的总和， $\tilde{S}_i \times S_i$ ，其中可以包括 \mathbf{q}_i 不涉及的位置：

$$a_i = \sum_{j \in \tilde{S}_i} \exp \left(\frac{\mathbf{q}_i \mathbf{k}_j^T}{\sqrt{d_k}} - m(i, S_i) - \log Z(i, S_i) \right) \mathbf{v}_j \quad (5.31)$$

$$m(i, S_i) = \begin{cases} \infty, & j \notin S_i \\ 0, & \text{otherwise} \end{cases} \quad (5.32)$$

(5.31) 中的项 $m(i, S_i)$ 是一个掩蔽项，可确保 q 不参与的关键位置对总和没有贡献。

如上所述，集合 S_i 是查询 i 所关注的关键位置的集合。在上面定义的 LSH 方案下， S_i 应该只包含散列到与查询相同的桶中的关键位置，或者换句话说

$$S_i = \{j : h(\mathbf{q}_i) = h(\mathbf{k}_j)\} \quad (5.33)$$

先验，不能保证查询将有任何需要注意的键。为了解决这个问题，并确保 $h(\mathbf{q}_i) = h(\mathbf{k}_j)$ ，[53] 修复了键 \mathbf{k}_j ，使得 $\mathbf{k}_j = \frac{\mathbf{q}_j}{\|\mathbf{q}_j\|}$ 。为了使计算更加高效，Reformer 做了两件简单的事情

- (a) 查询经过排序，因此同一散列桶中的查询是相邻的。在散列桶内，保留原始序列顺序
- (b) 排序后的查询被分组为 m 个连续查询的块，

$$m = \frac{2L}{\text{Number of buckets}} \quad (5.34)$$

在每个块内，每个位置都可以处理该块中的其他位置以及前一个块中的位置。这两个更改定义了查询 i 可以关注的一组新的关键位置：

$$\tilde{S}_i = \left\{ j : \left\lfloor \frac{s_i}{m} \right\rfloor - 1 \leq \left\lfloor \frac{s_j}{m} \right\rfloor \leq \left\lfloor \frac{s_i^{(r)}}{m} \right\rfloor \right\} \quad (5.35)$$

其中 s 是排序矩阵中的位置我被调到了那个位置。(5.35) 可以在 (5.31) 中使用来计算上述 LSH 方案下的注意力。

多轮 LSH 由于 LSH 固有的随机性，很容易将相似的东西放入不同的桶中。为了解决这个问题，进行多轮 LSH 是常见的做法。例如，当使用 LSH 逼近特征向量时，可能会对向量进行多次哈希处理以生成 LSH 签名。然后，您可以使用汉明距离来比较它们的 LSH 特征，而不是使用余弦相似度来比较两个特征向量。类似地，在 Reformer 中，您可以计算查询和键的多个哈希值，然后将它们组合起来。

如上所述，位置 i 处的查询可以关注 S_i 中的键位置。然而，如果我们执行 n 轮 LSH，那么我们最终会得到查询位置 i 可以参与的关键位置的 n 组

$$S_i = \bigcup_{r=1}^n S_i^{(r)}, \quad (5.36)$$

其中 $S_i^{(r)} = \{j : h^{(r)}(\mathbf{q}_i) = h^{(r)}(\mathbf{q}_j)\}$ 。然后，对于按哈希桶排序查询，(5.35) 变为

$$\tilde{S}_{ii}^{(r)} = \left\{ j : \left\lfloor \frac{s_i^{(r)}}{m} \right\rfloor - 1 \leq \left\lfloor \frac{s_j^{(r)}}{m} \right\rfloor \leq \left\lfloor \frac{s_i^{(r)}}{m} \right\rfloor \right\} \quad (5.37)$$

我们可以类似地更新多轮 LSH 的 (5.31):

$$a_i = \sum_r^n \exp(\log Z(i, S_i^{(r)}) - \log Z(i, S_i)) a_i^{(r)} \quad (5.38)$$

$$a_i^{(r)} = \sum_{j \in S_i^{(r)}} \exp \left(\frac{\mathbf{q}_i \mathbf{k}_j^T}{\sqrt{d_k}} - m_{i,j}^{(r)} - \log Z(i, S_i^{(r)}) \right) \mathbf{v}_j \quad (5.39)$$

$$m_{i,j}^{(r)} = \begin{cases} \infty & j \notin S_i^{(r)} \\ 10^5 & \text{if } i = j \\ \log N_{i,j} & \text{otherwise} \end{cases} \quad (5.40)$$

$$N_{i,j} = |r' : j \in S_i^{(r')}| \quad (5.41)$$

请注意，在多轮 LSH 情况下，Reformer 修改 mask-ing 项以包含（并降低权重）位置 i 处的查询涉及位置 i 处的键的情况。添加这种情况是因为，虽然标准转换器允许位置关注自身，但当 $\mathbf{Q} = \mathbf{K}$ 时这是没有帮助的，就像 LSH 关注中的情况一样。这是没有帮助的，因为任何位置 i 的 LSH 哈希 $h(\mathbf{q}_i)$ 都简单地等于其自身。

可逆性和内存 Reformer 还使用可逆残差层解决了标准 Transformer 的内存使用问题。残差层的设计是为了解决训练误差随网络深度增加的问题，同时避免梯度消失的问题。通过用残差函数重写神经网络，训练精度可以随着网络深度的增加而增加：

$$y = x + F(x) \quad (5.42)$$

其中 x 是层输入和 y 是层输出 [78]。ResNet 是这些残差层的堆栈。

在标准 Transformer 中，每个 Transformer 层都有多个残差层。例如，单个变换器编码器块具有两个剩余块。第一个残差块是多头注意力 [44]，然后是层归一化 [45]。第二个残差块是位置前馈网络 (FFN) [44]

，然后是层归一化。请注意，层归一化操作的形式为 $x + Sublayer(x)$ ，其中 Sublayer 表示传入层。对于第一个残差块，子层是多头注意力，对于第二个残差块，它是 FFN 层。

可逆残差层是图像识别中使用的残差层的可逆变体，被引入作为内存密集程度较低的替代品 [166]。每个可逆层采用一对 (x_1, x_2) 作为输入并返回一对 (y_1, y_2) 作为输出：

$$\begin{aligned} y_1 &= x_1 + F(x_2) \\ y_2 &= x_2 + G(y_1) \end{aligned} \tag{5.43}$$

要反转该层，只需简单地两边减去残差：

$$\begin{aligned} x_2 &= y_2 - G(y_1) \\ x_1 &= y_1 - F(x_2) \end{aligned} \tag{5.44}$$

现在，假设我们有 n 个这样的可逆层的堆栈，使得第 n 层的输入是该层的输出 $n - 1$ 。方程 (5.43) 和 (5.44) 成为递推关系：

$$y_1^{(n)} = y_1^{(n-1)} + F(y_2^{(n-1)}) \tag{5.45}$$

$$y_2^{(n)} = y_2^{(n-1)} + G(y_1^{(n)}) \tag{5.46}$$

$$y_2^{(n-1)} = y_2^{(n)} + G(y_1^{(n)}) \tag{5.47}$$

$$y_1^{(n-1)} = y_1^{(n)} + F(y_2^{(n-1)}) \tag{5.48}$$

因此，第 $n-1$ 层的输出激活可以根据第 n 层的输出激活和第 n 层的残差 F 和 G 的值来计算。这就是为什么不需要存储可逆残差层的激活来执行反向传播的原因。因为不需要存储中间层的激活，所以模型将使用更少的内存（而不牺牲性能）。**transformer** 中的**可逆层**现在我们了解了什么是可逆残差层以及它们如何节省内存，我们可以看看它们

在 Reformer 中如何使用 [53]。注意力机制作为残差函数 F，位置前馈网络为 G。层归一化成为残差块的一部分，因为它具有 $x + \text{Sublayer}(x)$ 的形式，其中 Sublayer 表示适当的残差函数。参考号 [53] 表明使用可逆残差层的 transformer 具有与标准 transformer 相同的性能。

3. Performer

The Performer[46] 模型使用一种称为“Fast Attention Via positive Orthogonal Random features (FAVOR+)”的方法来降低注意力机制的复杂性。表演者不会通过使用注意力模式提前偏置注意力机制，而是使用 FAVOR+ 使用内核来近似 softmax 计算。

将注意力作为内核通过重写自注意力公式 ((5.23))，内核形式主义变得显而易见：

$$\text{Attn}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax} \left(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d_k}} \right) \mathbf{V} \quad (5.49)$$

$$\Rightarrow \left[\frac{\exp \left(\frac{\mathbf{q}_i \mathbf{k}_j^T}{\sqrt{d_k}} \right)}{\sum_{r \in S_i} \exp \left(\frac{\mathbf{q}_i \mathbf{k}_r^T}{\sqrt{d_k}} \right)} \right] \mathbf{V} \quad (5.50)$$

$$= \mathbf{D}^{-1} \left[\exp \left(\frac{\mathbf{q}_i \mathbf{k}_k^T}{\sqrt{d_k}} \right) \right] \mathbf{V} \quad (5.51)$$

其中 \mathbf{D} 是项的对角矩阵 $D_{ij} = \delta_{ij} \sum_{r \in S_i} \exp \left(\frac{\mathbf{q}_i \mathbf{k}_r^T}{\sqrt{d_k}} \right)$ 。请注意，每个对角元素都是 softmax 分母中总和的一部分，并且 \mathbf{D} 的倒数只是倒数矩阵。在 Reformer 中，我们可以将注意力权重重新定义为 $A_{ij} = \exp \left(\frac{\mathbf{q}_i \mathbf{k}_j^T}{\sqrt{d_k}} \right)$ ，因此 $D_{ij} = \delta_{ij} \sum_{r \in S_i} A_{ir}$ 。因此 $\text{Attn}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \mathbf{D}^{-1} \mathbf{A} \mathbf{V}$ 。查询和键可以缩放，以便将键维度吸收到它们中， A_{ij} 简单地是 $\exp \mathbf{q}_i \mathbf{k}_j^T$ 。

由于 A_{ij} 取决于查询和键向量的内积，因此它是 \mathbf{q}_i 和 \mathbf{k}_j 之间相似性的度量。这些注意力权重可以使用 FAVOR+ 算法来近似：

$$A_{ij} = \langle \phi(\mathbf{q}_i^T)^T \phi(\mathbf{k}_j^T) \rangle \quad (5.52)$$

其中映射 ϕ 分别映射 $\mathbf{Q}, \mathbf{K} \in \mathbb{R}^{L \times d_k}$ 到 $\mathbf{Q}', \mathbf{K}' \in \mathbb{R}^{L \times r}$ ，当 $r > 0$ 。 \mathbf{Q}' 的行是 $\phi(\mathbf{q}_i^T)^T$ \mathbf{K}' 的行是 $\phi(\mathbf{k}_i^T)^T$ ；则 (5.49) 变为

$$\widehat{\text{Attn}}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \hat{\mathbf{D}}^{-1}(Q'((\mathbf{K}')^T V)), \hat{D}_{ij} = \delta_{ij} \sum_{m \in S_i} \phi(\mathbf{q}_i^T)(\mathbf{k}_m^T)^T \quad (5.53)$$

当核函数 $\phi(x)$ 如 [46] 中定义时，注意力权重 $A_{ij} = \exp(\mathbf{q}_i \mathbf{k}_j^T)$ 可以近似为

$$\exp(\mathbf{q}_i \mathbf{k}_j^T) = \Lambda \langle \cosh(\omega^T (\mathbf{q}_i + \mathbf{k}_j)) \rangle_\omega, \quad (5.54)$$

其中 $\Lambda = \exp(-(\|\mathbf{q}_i\|^2 + \|\mathbf{k}_j\|^2)/2)$ 并且 ω 从 d_k 维标准正态分布中采样。如果将 (5.54) 中的 ω 替换为 $\sqrt{d} \frac{\omega}{\|\omega\|}$ ，则 ω 是半径为 $\sqrt{d_k}$ 的 d_k 维球体表面上的任意点。

softmax 计算的这种核近似将二次复杂度降低到序列长度上接近线性的复杂度，并且可以通过定期重采样来减少近似误差 ω 。

4. Big Bird

Big Bird 是为 Transformer 提供允许线性缩放的稀疏注意机制的另一项努力。它也被证明是图灵完备的和通用序列函数逼近器 [54]

注意力机制是一个有向图 Big Bird 将注意力机制描述为一个有向图。顶点表示输入序列中的 L 个位置。有向边表示 softmax 之前的注意力权重。换句话说，有向边是查询向量和关键向量之间的内积，其中第 i 个顶点只会有有向边，其顶点对应于它可以关注的关键位置，如集合 S_i 在 (5.27) 所描述的。

邻接矩阵图表示将降低注意力机制复杂性的问题投射为图解析问题，可以用图论来解决。Big Bird 用它的 $L \times L$ adjacency 矩阵来描述注意力图， \mathbf{A} ：

$$A(i, j) = \begin{cases} 1, & j \in S_i \\ 0, & \text{otherwise} \end{cases} \quad (5.55)$$

当邻接矩阵全为 1 时， $A(i, j) = 1, \forall(i, j)$ ，那么我们就有了一个全连接图，其中每个顶点都连接到其他每个顶点，并且具有二次复杂度的注意力机制。当 $A(i, j) = 0$ 时，意味着边 (i, j) 不存在并且因此查询位置 $i(\mathbf{q}_i)$ 无法关注关键位置 $j(\mathbf{k}_j)$ 。

注意力图模式使用图论框架，Big Bird 结合了三种注意力模式：随机注意力、滑动窗口注意力和全局注意力。每个注意力模式都对应于初始化邻接矩阵的特定方式。

随机注意力 Big Bird 的随机注意力模式受到 Erdős-Renyi 随机图 [167] 类型的启发，该图以均匀概率随机连接顶点。在 Big Bird 的随机注意力中，每个查询 \mathbf{q}_i 关注随机数量的键。就邻接矩阵而言，对于以相等概率随机选择的关键位置， $A(i,:) = 1$ 。随机注意力模式的邻接矩阵示例如图 5.9 所示。

滑动窗口注意力 Big Bird 的局部注意力源自环格上的 Watts-Strogatz [168] 小世界图，其中每个顶点都连接到 w 个邻居，因此有 $w/2$ 两边的邻居。这允许位置 i 处的查询关注窗口 $[i-w/2, i+w/2]$ 中位置处的键。就邻接矩阵而言， $A(i, i-w/2 : i+w/2) = 1$ 。这类似于第 2 节中讨论的 Longformer 中的滑动窗口注意力。5.2.2.1. 滑动窗口注意力的邻接矩阵示例如图 5.10 所示。

$$\begin{bmatrix} \bullet & \bullet & \bullet & \bullet & \cdot & \cdot & \cdot & \cdot & \bullet & \bullet \\ \cdot & \bullet \\ \cdot & \bullet \\ \bullet & \cdot & \cdot & \bullet & \cdot & \bullet & \cdot & \bullet & \bullet & \bullet \\ \cdot & \bullet \\ \bullet & \bullet \\ \cdot & \bullet \\ \bullet & \bullet \\ \cdot & \bullet \\ \bullet & \bullet \\ \cdot & \bullet \\ \bullet & \bullet \end{bmatrix}$$

Figure 5.9 Random adjacency matrix, for $L = 12$ and $r = 7$. Row i corresponds to query i . Columns with a \bullet are keys that query i attends to and \cdot represents a lack of attention (a missing edge).

图 5.9 随机邻接矩阵， $L= 12$ 且 $r= 7$ 。第 i 行对应于查询 i 。带有 \bullet 的列是查询 i 关注的键，并且 \cdot 表示缺乏关注（缺失边缘）

$$\begin{bmatrix} \bullet & \bullet & \bullet & \cdot \\ \bullet & \bullet & \bullet & \bullet & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \bullet & \bullet \\ \cdot & \bullet \\ \cdot & \bullet \\ \cdot & \bullet \\ \cdot & \cdot & \bullet \\ \cdot & \cdot & \bullet \\ \cdot & \cdot & \bullet \\ \cdot & \cdot & \bullet \\ \cdot & \cdot & \bullet \\ \cdot & \cdot & \bullet \end{bmatrix}$$

Figure 5.10 Sliding window attention adjacency matrix, for $L = 12$ and $w = 4$. Row i corresponds to query i . Columns with a \bullet are keys that query i attends to and \cdot represents a lack of attention (a missing edge).

图 5.10 滑动窗口注意力邻接矩阵，对于 $L= 12$ 且 $w= 4$ 。第 i 行对应于查询 i 。带有 \bullet 的列是查询关注的键， \cdot 表示缺乏关注（缺少边缘）。

$$\begin{bmatrix} \cdot & \cdot & \bullet & \cdot & \cdot & \bullet & \cdot & \cdot & \cdot \\ \cdot & \cdot & \bullet & \cdot & \cdot & \bullet & \cdot & \cdot & \bullet \\ \bullet & \cdot & \cdot & \bullet & \cdot & \cdot & \bullet & \cdot & \bullet \\ \cdot & \cdot & \cdot & \cdot & \bullet & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \bullet & \cdot & \cdot & \bullet \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \bullet & \cdot & \bullet \\ \cdot & \bullet & \cdot \\ \cdot & \bullet \\ \cdot & \cdot \\ \cdot & \cdot \\ \cdot & \cdot \\ \cdot & \cdot \end{bmatrix}$$

Figure 5.11 Global attention adjacency matrix for the internal transformer construction, for $L = 12$ and $G = 3, 4, 7$. Row i corresponds to query i . Columns with a \bullet are keys that query i attends to and \cdot represents a lack of attention (a missing edge).

图 5.11 全局注意力邻接矩阵对于内部 transformer 结构, $L= 12$ 且 $G= 3,4,7$ 。第 i 行对应于 query i 。带有 \bullet 的列是查询 i 关注的键，并且 \cdot 表示缺乏关注 (缺少边缘)。

全局关注 Big Bird 还允许一些标记关注序列中的所有标记。这些全局标记也由所有的标记关注。Big Bird 使用两种类型的全局标记：内部 transformer 构造和外部 transformer 构造。

在内部 transformer 构造中, L vertex 的子集 G 被提升为全局标记。因此这些位置中的查询或密钥会涉及所有其他位置。这里, $A(i,:) = A(:,i) = 1$, $i \in G$ 。扩展的邻接矩阵 Bis 如图 5.11 所示。

外部 transformer 结构向现有的 L 标记添加了附加标记。附加标记是全局的。示例包括 transformer 中使用的特殊标记，例如 [CLS]。这本质上创建了一个新的邻接矩阵 B , 其中通过在 A 上预先添加增长和列来包含特殊标记。此处, $B(i,:) = B(:,i) = 1$, 其中 $i = 1, \dots, g$, 且 $B(g+i, g+j) = A(i,j)$, 其中 i 和 $j = 1, \dots, L$ 。扩展的邻接矩阵如图 5.12 所示。

最后, 随机、滑动窗口和全局注意力 (外部构造) 组合的邻接矩阵示例如图 5.13 所示。

$$\begin{bmatrix} \bullet & \bullet \\ \bullet & \bullet \\ \bullet & \bullet \\ \bullet & \bullet & \bullet & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \bullet & \bullet & \bullet & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \bullet & \bullet & \bullet & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \bullet & \bullet & \bullet & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \bullet & \bullet & \bullet & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \bullet & \bullet & \bullet & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \end{bmatrix}$$

Figure 5.12 Global attention adjacency matrix for the external transformer construction, for $L = 9$ and $g = 3$. Row i corresponds to query i . Columns with a \bullet are keys that query i attends to and \cdot represents a lack of attention (a missing edge).

图 5.12 外部 transformer 构造的全局注意力邻接矩阵， $L= 9$ 且 $g= 3$ 。
第 i 行对应于查询 i 。带有 \bullet 的列是查询 i 关注的键，并且 \cdot 表示缺乏关注（缺失边缘）

$$\begin{bmatrix} \bullet & \bullet \\ \bullet & \bullet \\ \bullet & \bullet \\ \bullet & \bullet \\ \bullet & \bullet \\ \bullet & \bullet \\ \bullet & \bullet \\ \bullet & \bullet \\ \bullet & \bullet \end{bmatrix}$$

Figure 5.13 Big Bird attention adjacency matrix for the external transformer construction, for $L = 12$, $g = 2$, $w = 4$, and $r = 4$. Row i corresponds to query i . Columns with a \bullet are keys that query i attends to and \cdot represents a lack of attention (a missing edge).

图 5.13 外部 transformer 构造的 Big Bird 注意力邻接矩阵，对于 $L=$

12, g = 2, w = 4, and r = 4。带有 • 的列是查询 i 关注的键，并且 · 表示缺乏关注（缺失边缘）。

高效稀疏矩阵乘法通过使邻接矩阵（和相应的注意力矩阵）稀疏，GPU 提供的加速被消除。这是因为任意稀疏矩阵的乘法在 GPU 上效率不高。然而，稀疏性被分组为块的矩阵可以在 GPU 上有效地相乘 [169]。为了解决这个问题，Big Bird 将查询块和键分组在一起，然后按块而不是按单个序列位置添加稀疏性 [54]。选择块大小 b，然后将序列长度除以 b, L/b 块。因此，将不再有 L 查询和键，而是 L/b 查询和键。这以相对简单的方式修改了上面讨论的每个注意力模式：

- (a) 随机注意力查询要关注的键的随机数量，r，成为查询块关注的关键块的随机数量。
- (b) 滑动窗口关注查询块 i 关注到关键块 $i - (w - 1)/2$ 到 $i + (w - 1)/2$
- .
- (c) 全局注意力全局注意力的定义没有改变，只是它是根据块而不是序列位置来定义的。

5.2.3 改进多头注意力

现在我们关注一些改变注意力机制以提高 Transformer 性能的方式。

1. Talking-heads 的注意力

Vaswani 等人 [44] 表明多头注意力允许 transformer 执行 h 个（注意力头数量）单独的注意力计算。相反，Talking-Heads Attention [51] 允许注意力头共享信息。它的工作原理是添加两个线性层，将查询矩阵和关键矩阵的乘积 $\mathbf{Q}\mathbf{K}^T$ （注意力逻辑）投影到一个新空间，并将注意力权重 $\text{Softmax}(\mathbf{Q}\mathbf{K}^T)$ 投影到一个新空间。

Talking-Heads Attention (THA) 还将注意力头分为三种类型：查询和键的头、值的头、注意力逻辑和注意力权重的头。让我们详细看看这个。

回顾第 5.2.1.1 节，长度为 L1 和 L2 的两个序列 $\mathbf{X}_1, \mathbf{X}_2$ 之间的多头注意力为：

$$\mathbf{Q} = \mathbf{X}_1 \mathbf{W}_q, \in \mathbb{R}^{L_1 \times d_k \times h} \quad (5.56)$$

$$\mathbf{K} = \mathbf{X}_2 \mathbf{W}_k, \in \mathbb{R}^{L_2 \times d_k \times h} \quad (5.57)$$

$$\mathbf{V} = \mathbf{X}_2 \mathbf{W}_v, \in \mathbb{R}^{L_2 \times d_v \times h} \quad (5.58)$$

其中 $\mathbf{X}_1 \in \mathbb{R}^{L_1 \times d}$, $\mathbf{X}_2 \in \mathbb{R}^{L_2 \times d}$, $\mathbf{W}_q, \mathbf{W}_k \in \mathbb{R}^{d \times d_k \times h}$ 且 $\mathbf{W}_v \in \mathbb{R}^{d \times d_v \times h}$

$$\alpha = \mathbf{Q} \mathbf{K}^T, \in \mathbb{R}^{L_1 \times L_2 \times h} \quad (5.57)$$

$$\mathbf{A}(\mathbf{Q}, \mathbf{K}) = \text{softmax}\left(\frac{\alpha}{\sqrt{d_k}}\right), \in \mathbb{R}^{L_1 \times L_2 \times h} \quad (5.58)$$

$$\mathbf{C}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \sum_{L_2} A(\mathbf{Q}, \mathbf{K}) \mathbf{V}, \in \mathbb{R}^{L_1 \times d_v \times h} \quad (5.59)$$

其中 α 是注意力逻辑, $\mathbf{A}(\mathbf{Q}, \mathbf{K})$ 是注意力权重, $\mathbf{C}(\mathbf{Q}, \mathbf{K}, \mathbf{V})$ 是“上下文”向量, 表示注意力串联之前的注意力头的输出头和最终投影层。

对注意力头进行分区 THA 可以通过 (5.56) - (5.59) 中所示的几种方式修改注意力机制。首先, 将 \mathbf{Q} 和 \mathbf{K} 的注意力头维度更改为查询键注意力头 h_k 的数量, 并将 \mathbf{V} 的注意力头维度更改为值注意力头 h_v 的数量。这是通过更改从输入序列生成查询矩阵、键矩阵和值矩阵的投影矩阵的维度来实现的。换句话说, (5.56) 变为

$$\mathbf{Q} = \mathbf{X}_1 \mathbf{W}_q, \in \mathbb{R}^{L_1 \times d_k \times h_k} \quad (5.60)$$

$$\mathbf{K} = \mathbf{X}_2 \mathbf{W}_k, \in \mathbb{R}^{L_2 \times d_k \times h_k}$$

$$\mathbf{V} = \mathbf{X}_2 \mathbf{W}_v, \in \mathbb{R}^{L_2 \times d_v \times h_v}$$

其中 $\mathbf{W}_q, \mathbf{W}_k \in \mathbb{R}^{d \times d_k \times h_k}$, 并且 $\mathbf{W}_v \in \mathbb{R}^{d \times d_v \times h_v}$ 。

投影注意力逻辑接下来, 注意力 logit α 被投影到一个线性层, 该线性层将查询关键注意力头与注意力 logit/weigh 头混合, $\mathbf{W}_\alpha \in \mathbb{R}^{h_k \times h}$, 以及注意力权重用一个线性层进行投影, 该线性层将值注意力头与注

意力 logit/weight 头混合, $W_A \in \mathbb{R}^{h \times h_v}$ 。换句话说, (5.57)–(5.59) 变成

$$\alpha = \mathbf{Q}\mathbf{K}^T, \in \mathbb{R}^{L_1 \times L_2 \times h_k} \quad (5.61)$$

$$\mathbf{P}_\alpha = \alpha \mathbf{W}_\alpha \mathbb{R}^{L_1 \times L_2 \times h} \quad (5.62)$$

$$\mathbf{A}(\mathbf{Q}, \mathbf{K}) = \text{softmax} \left(\frac{\mathbf{P}_\alpha}{\sqrt{d_k}} \right), \in \mathbb{R}^{L_1 \times L_2 \times h} \quad (5.63)$$

$$\mathbf{P}_A(\mathbf{Q}, \mathbf{K}) = \mathbf{A}\mathbf{W}_A, \in \mathbb{R}^{L_1 \times L_2 \times h_v} \quad (5.64)$$

$$\mathbf{C}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \sum_{L_2} \mathbf{P}_A \mathbf{V}, \in \mathbb{R}^{L_1 \times d_v \times h_v} \quad (5.65)$$

其中 \mathbf{P}_α 是投影注意力逻辑, $\mathbf{P}_A(\mathbf{Q}, \mathbf{K})$ 是投影注意力权重。

性能通过使用 THA 和与 T5 论文相同的超参数（除了在预训练期间省略 dropout）训练 T5 模型并类似地训练 ALBERT 模型来评估头部注意力 (THA)。作者发现 THA 的表现始终优于多头注意力 [51]。仅投影注意力逻辑或仅投影注意力权重仅比使用纯多头注意力稍好一些。使用这两种投影可以显着提高性能。在编码器块的自注意力层上使用头部说话注意力比在解码器块的注意力层上使用头部说话注意力对模型性能有更大的影响。

多头注意力已经带来了额外的成本, 而 THA 添加的投影 \mathbf{W}_α 和 \mathbf{W}_A 增加了成本。计算预计注意力逻辑 \mathbf{P}_α 会添加 $L_1 \cdot L_2 \cdot h_k \cdot h$ 矩阵乘法, 计算预计注意力权重 \mathbf{P}_A 会添加 $L_1 \cdot L_2 \cdot h \cdot h_v$ 矩阵乘法, 计算预矩阵乘法。总之, THA 添加了 $L_1 \cdot L_2 \cdot h \cdot (h_k + h_v)$ 矩阵乘法, 这将使计算成本更高。作者指出, 如果 $h < d_k$ 且 $h < d_v$, 则 THA 投影的成本小于现有多头注意力计算的成本。因此, 总之, Talking-Heads Attention 提高了模型质量, 但代价是增加了计算时间; 允许用户决定他们希望做出什么样的权衡。

5.2.4 偏向先验的注意力

注意力本节讨论一些通过固定哪些位置可以关注哪些位置来偏置注意力机制的模型。第 5.2.2.1 节中讨论的 Longformer 模型和第 5.2.2.4 节中讨论的 Big Bird 模型都是带有先验的注意力的示例，因为每个模型都使用特定的注意力模式，例如第 5.2.2.1 节和 5.2.2.4 节中的滑动窗口注意力。我们还在第 5.1.2.1 节中讨论了带有先验的偏置注意力的另一个例子，Realformer.

5.2.5 原型查询

- 集群注意力 集群注意力 [56] 是一种避免自注意力 $O(L^2 \cdot d_k + L^2 \cdot d_v)$ 通过使用 k 均值聚类算法对 LSH 哈希查询进行聚类来线性化自注意力权重计算的时间复杂度。并使用查询质心作为查询来计算注意力矩阵。

聚类查询向量聚类注意力发生在两个阶段。首先，每个查询向量都使用局部敏感哈希进行哈希处理。然后，散列查询会使用 k-means 分组为 C 个 cluster。k-means 使用的距离度量是汉明距离。第 j 个簇的质心由

$$\mathbf{q}_j^c = \frac{\sum_{i=1}^L S_{ij} \mathbf{q}_i}{\sum_{i=1}^L S_{ij}} \quad (5.66)$$

给出，其中 \mathbf{q}_j^c 是第 j 个簇的质心，矩阵 $S \in \{0, 1\}^{L \times C}$ 将查询向量划分为 C 个不重叠的簇，因此如果 $S_{ij} = 1$ ，则 \mathbf{q}_i 在簇 j 中。质心查询被分组为质心向量矩阵 $\mathbf{Q}^c \in \mathbb{R}^{C \times d_k}$ 。然后我们可以用查询质心矩阵替换真实查询矩阵， \mathbf{Q}^c 并计算聚类注意力矩阵：

$$\mathbf{A}^c = \text{softmax} \left(\frac{\mathbf{Q}^c \mathbf{K}^T}{\sqrt{(d_k)}} \right), \in \mathbb{R}^{C \times L} \quad (5.67)$$

你可以停在这里，只使用聚类注意力权重计算注意力机制的输出。该计算的时间复杂度为 $O(CL \cdot d_k + LC \cdot d_v)$ ，与序列长度显式呈线性关系。但是，可以改进聚类注意力近似，以找到在每个 C 簇具有最高注

意力得分的 k 个键。并且，对于集群的每个 top- k 键，使用该集群中的查询计算注意力：

$$A_{ij}^t = \begin{cases} \frac{\hat{m}_j \exp(\mathbf{q}_i \mathbf{k}_l^T)}{\sum_{r=1}^L T_{jr} \exp(\mathbf{q}_i \mathbf{k}_r^T)}, & \text{if } T_{ij} = 1 \\ A_{jl}^c, & \text{otherwise} \end{cases} \quad (5.68)$$

其中 $\hat{m}_j = \sum_{i=1}^L T_{ij} A_{ij}^c$ 且 $T \in \{0, 1\}^{C \times L}$ ：如果 $T_{ij} = 1$ ，则 \mathbf{k}_i 是聚类 j 中的前 k 个键之一。

然后计算聚类注意力的上下文向量（值的加权平均值）并将其作为值矩阵： $\hat{\mathbf{V}} = \mathbf{A}^t \mathbf{V}, \in \mathbb{R}^{L \times d_v}$ 。这使得聚类注意力计算的复杂度为 $O(CL \cdot d_k + LC \cdot d_v + kL \max(d_k, d_v))$ ，与序列长度成线性关系。性能集群注意力在《华尔街日报》和 Switch-board 音频数据集的自动语音识别方面优于标准 Transformer 和 Reformer。它还近似于在 GLUE 和 SQuAD 上预训练的 RoBERTa，但性能损失很小。它在 GLUE 上的表现优于 RoBERTa，但不是 SQuAD，后者稍差。

随着簇数量的增加，近似值变得更加准确。对于长序列长度，它的收敛速度是标准 Transformer 的两倍，而对于短序列长度，簇注意力并不比标准 Transformer 快。

5.2.6 压缩键值内存

1. Luna：线性统一嵌套注意力 Luna [170]，代表线性统一嵌套注意力，用两个嵌套线性注意力计算替换每个注意力头中的热注意力权重计算，使用额外的、可学习的输入序列来学习编码上下文信息： $P \in \mathbb{R}^{l \times d}$ ，其中 l 是序列的长度。前面讨论过，查询序列 $X \in \mathbb{R}^{n \times d}$ 和上下文序列 $C \in \mathbb{R}^{m \times d}$ 之间的注意力头的输出可以写为

$$\mathbf{Y} = Attn(\mathbf{X}, \mathbf{C}) = \text{softmax} \left(\frac{\mathbf{X} \mathbf{W}_q (\mathbf{C} \mathbf{W}_k)^T}{\sqrt{d_k/h}} \right) \mathbf{C} \mathbf{V}, \in \mathbb{R}^{n \times d} \quad (5.69)$$

其中 $\mathbf{W}_q, \mathbf{W}_k, \mathbf{W}_v$ 是分别将输入序列投影到查询、键和值空间的线性投影，如 (5.24) 所示。

Luna 注意

包注意 第一个嵌套注意层计算额外序列 \mathbf{P} 和上下文序列之间的注意力。C. Luna 称之为“包注意力”。它的输出是“打包上下文”， $\mathbf{Y}_P = Attn(\mathbf{P}, \mathbf{C})$, $\in \mathbb{R}^{l \times d}$ ，它与 \mathbf{P} 具有相同的维度。它的复杂度是 $O(lm)$ 。

解包注意 力嵌套注意力层的第二个计算输入序列和打包上下文序列 \mathbf{Y}_P 之间的注意力。Luna 将其称为“解压注意力”： $\mathbf{Y}_X = Attn(\mathbf{X}, \mathbf{Y}_P)$, $\in \mathbb{R}^{n \times d}$ 。它的输出与输入序列 \mathbf{X} 具有相同的维度。其复杂度为 $O(ln)$ 。

Luna 注意力 Luna 按顺序组合了打包和解包注意力层，形成 Luna 层： $\mathbf{Y}_X, \mathbf{Y}_P = LunaAttn(\mathbf{X}, \mathbf{P}, \mathbf{C})$ ，它包含图 5.14 所示的两个多头注意力层。 \mathbf{P} 被初始化为 Transformer 的位置编码，对于后续的 Transformer 块，使用 \mathbf{Y}_P 代替 \mathbf{P} 。由于打包注意力层和解包注意力层是按顺序组成的，因此它们的组合复杂度为 $O(lm + ln)$ 。由于额外的输入序列 Ph 的长度是固定的，Luna 的多头注意力机制的组合复杂度与输入序列 \mathbf{X} 的长度呈线性关系。稍作修改，Luna 还可以支持因果交叉注意力。

性能 Luna 与 Long Range Arena 基准 (LRA) [171] 上的标准 transformer 和 11 个其他高效 transformer。它在所有任务上都表现良好，优于大多数（但不是全部）模型；获得最高的平均准确度。Lunawa 始终比标准 transformer 以及与其进行比较的许多（但不是全部）高效 transformer 更快。它的内存使用率也比标准 Transformer 更高，内存使用量与其他高效 Transformer 相当，甚至比其他高效 Transformer 还要好。

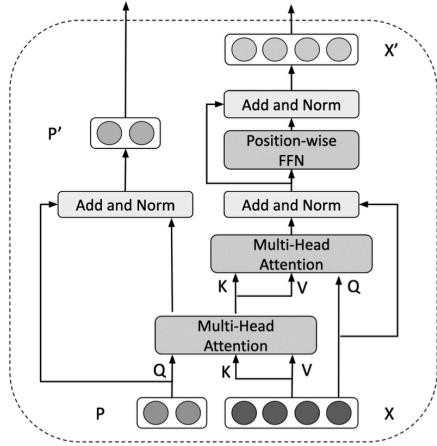


Figure 5.14 Architecture of a Luna encoder block. \mathbf{P} is the extra sequence for the block and \mathbf{X} is the input sequence of the block. \mathbf{P}' will be the extra sequence for the next Luna block and \mathbf{P}' will be the input sequence of the next Luna block.

图 5.14 Luna 编码器块的架构。 \mathbf{P} 是块的额外序列， \mathbf{X} 是块的输入序列。 \mathbf{P}' 将是下一个 Luna 块的额外序列， \mathbf{P}' 将是下一个 Luna 块的输入序列。

5.2.7 低阶近似

1. Linformer 在 [60] 中，Wang 等人。证明标准 Transformer 的 [44] 自注意力权重矩阵可以用低秩矩阵 [172] 来近似，将复杂度从 $O(L^2)$ 降低到 $O(L)$ ，其中 L 是序列长度。

Linformer 引入了两个线性变换第 i 个注意力头的键矩阵和值矩阵的投影矩阵： $\mathbf{E}_i = \delta \mathbf{R}$ 和 $\mathbf{F}_i = \exp^{-\delta} \mathbf{R}$ ，其中 $\mathbf{R} \in \mathbb{R}^{k \times L}$ 其分量取自 $\mathcal{N}(0, 1/k)$, $\delta = \theta(1/L)$, $k = 5 \log(dL)(\epsilon^2 - \epsilon^3)$ ， ϵ 是近似误差。另外， $d_k = d_v = d$ 。这种近似意味着机制中每个注意力头的输出可以近似为 $\text{head}_i = \overline{\mathbf{A}}_i \cdot \mathbf{F}_i \mathbf{V}_i$ ，其中

$$\overline{\mathbf{A}}_i = \text{softmax} \left(\frac{\mathbf{Q}_i (\mathbf{E}_i \mathbf{K}_i)^T}{\sqrt{d_k/h}} \right), \in \mathbb{R}^{L \times k} \quad (5.70)$$

\bar{A}_i 是注意力头的注意力权重矩阵的近似。它的复杂度是 $O(kL)$ ，这就是 Linformer 也是一种降低注意力机制的空间和时间复杂度的方法的原因，因此适合第 5.2.2 节中讨论的方法。

作者考虑了参数共享的三种变体，以使 Linformer 更加高效：

- (a) 在所有注意力头之间共享 E 和 F (头向共享): $\mathbf{E}_i = \mathbf{E}, \mathbf{F}_i = \mathbf{F}$
- (b) 跨头和键/值矩阵共享投影: $\mathbf{E}_i = \mathbf{F}_i = \mathbf{E}$
- (c) 跨头、键/值矩阵和模型层共享投影 (分层共享)

推理速度首先针对标准转换器进行测试 [44]。随着序列长度的增加，标准 transformer 变得更慢。Linformer 的速度基本保持恒定，并且对于长序列来说速度明显更快。王等人。保持序列长度固定并改变批量大小来处理文本序列。由于序列长度保持固定，因此这种行为是预期的。

为了测试性能，Linformer 以与 RoBERTa [84] 相同的方式进行训练，并使用验证困惑度和性能进行比较曼斯对下游任务。基于困惑度，Linformer 的性能随着 k 的增加而增加， $L=512, k=128$ 和 $L=1024, k=256$ 时，Linformer 的质量几乎与标准 Transformer 相当。基于下游任务性能，Linformer 的性能与 RoBERTa 相当。当 $L=512, k=128$ 时，在 $k=256$ 时优于 RoBERTa。 $L=1024, k=256$ 和 $L=512, k=256$ 的性能相似，表明 Linformeris 的性能更多地受 k 控制而不是 L/k 控制。有两个关键结果：

1. 分层参数共享效果最好。
2. 使用分层共享时的性能几乎与不使用参数共享时的性能相同。

这表明可以减少 Linformer 引入的参数数量，而不会影响质量。第 5.2.2.3 节中讨论的 Performer 模型也基于注意力的低秩近似。Transformer 修改 145

5.3 训练任务效率的修改

5.3.1 ELECTRA

ELECTRA [173] 引入了一个新的预训练任务，替换的标记检测，它结合了最好的 Masked Language Model 和常规语言模型，使 ELECTRA 比其前辈更有效地学习。ELECTRA 是一个双向模型，就像一个 Masked Language Model。然而，与 Masked Language Model 不同，它从输入序列中的所有位置进行学习。

1. 替换标记检测 替换标记检测 (RTD) 帮助模型了解哪些标记最有可能位于序列中的给定位置，因为模型学习存储位置并识别训练数据中的每个标记。ELECTRA 就像生成对抗网络 (GAN) 中的判别器。与 GAN 不同，生成器不会被训练来欺骗鉴别器，并且噪声向量不会添加到生成器的输入中。

ELECTRA 使用生成器和鉴别器的组合进行训练。生成器是一个 Masked Language Model，判别器是 ELECTRA 本身。结果，ELECTRA 比早期的 Masked Language Model (如 BERT) 需要更少的训练数据，因为 Masked Language Model 不会掩码训练数据中的每个标记。生成器和判别器都是 transformer 编码器。

训练开始于生成器从输入序列中选择一组随机的 k 位置 $\mathbf{m} = (m_1, \dots, m_k)$ ，使得每个位置 m_i ，其中 $i = 1, \dots, k$ ，从 $[1, L]$ 统一得出，其中列出了序列长度。输入序列 $\mathbf{x} = (x_1, \dots, x_L)$ 中位于 \mathbf{m} 位置的每个标记将被替换为 [MASK] 标记。屏蔽序列为

$$\mathbf{x}^{masked} = REPLACE(\mathbf{x}, \mathbf{m}, [MASK]) \quad (5.71)$$

然后生成器学习 \mathbf{x} 中屏蔽标记的未屏蔽版本。生成器输出特定标记的概率 x_t 是

$$Prob_G(x_t | \mathbf{x}) = \frac{\exp(\mathbf{e}_t^T \mathbf{h}_G(\mathbf{x})_t)}{\sum_{t'} \exp(\mathbf{e}(x')^T \mathbf{h}_G(\mathbf{x})_T)}, \quad (5.72)$$

其中 e^t 是嵌入向量对于代币 x_t 。

鉴别器学习辨别序列中的哪些标记是由生成器生成的。这是通过创建一个序列来实现的，其中屏蔽的标记已被生成器中的样本（“损坏的”标记）替换：

$$\mathbf{x}^c = \text{REPLACE}(\mathbf{x}, \mathbf{m}, \hat{\mathbf{x}}_i) \quad (5.73)$$

其中 $\hat{\mathbf{x}}_i$ 是生成器生成的标记：

$$\hat{\mathbf{x}}_i \sim \text{Prob}_G(x_i | \mathbf{x}^{\text{masked}}) \quad (5.74)$$

其中 $i \in m$ 。然后，判别器学习识别原始输入 \mathbf{x} 中包含哪些 \mathbf{x}^c 的标记。在这里，与 GAN 不同，生成器没有经过训练来欺骗鉴别器，并且噪声向量也没有添加到生成器的输入中。

2. T5 T5（文本到文本传输转换器 [68]）是一项针对以下问题的调查的结果：了解哪种迁移学习方法最有效。它将标准 NLP 任务重新定义为文本到文本的转换，其中输入和输出都是字符串。这与 BERT 等屏蔽语言模型形成对比，后者输出类标签或输入范围。

训练任务的重新表述意味着您可以在任何 NLP 任务上使用相同的模型和损失函数。使用文本到文本转换（序列转换）可以让您同时针对多个任务训练一个模型，并重用模型架构、损失函数和超参数。Raffel 等人的附录 D。有关于如何为 T5 模型训练或微调的每个数据集格式化输入的示例。

5.4 Transformer 子模块修改

本节讨论对 Transformer 的修改，但不会修改注意力机制或模型的内存配置文件。

5.4.1 Switch Transformer

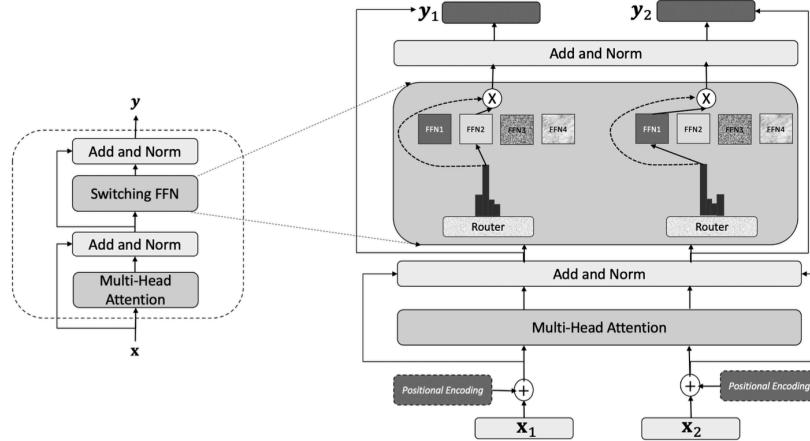


Figure 5.15 Switch Transformer encoder block illustrating two input tokens x_1 and x_2 being processed through the network. The dense FFN is replaced with switching FFN as one of the experts.

图 5.15 Switch Transformer 编码器块说明了通过网络处理的两个输入 tokens x_1 和 x_2 。在稀疏模型中，稠密 FFN 被交换 FFN 取代，作为专家之一，具有大量计算成本和训练不稳定。

专家混合 (MoE) 模型以复杂性和培训成本为代价促成了许多成功 [174]。然而，专家模型的混合不共享参数，导致稀疏模型具有大量计算成本和训练不稳定。开关变压器通过专家之间的新颖路由算法解决了大部分问题，从而在不增加计算成本的情况下增加参数数量 [175]。开关变压器的核心创新在于用开关前馈层代替变压器中的前馈层，如图 5.15 所示。

在标准 transformer 中，单个前馈网络遵循多头注意力层的输出。它负责将表示逐个标记转换为下一个转换器输入块。如图 5.15 所示，在开关 transformer 中，不是一个前馈网络，而是多个前馈网络，也称为专家网络。在多头注意力之后，每个标记表示 x 仅被路由到一个专家。通过计算专家的概率分布，从一组 N 专家中选择专家， $\{E_i(\mathbf{x})\}_{i=1}^N$ ：

$$\mathbf{p}(\mathbf{x}) = \text{softmax}(\mathbf{W}_r \cdot \mathbf{x}) \quad (5.75)$$

其中 $\mathbf{W}_r \in \mathbb{R}^{N \times d}$ 是一个可学习的权重矩阵，决定哪个选择专家并确定标记表示 x 的维度。概率最大的专家对应于 5.75 的最大组成部分： $p(\mathbf{x}) = (p_1, \dots, p_N)$ ，称之为 p_j ，用于计算更新后的标记表示。因此，尽管由于四个前馈网络，参数增加了四倍，硬路由保证计算成本保持不变。因此，切换前馈网络可以将专家扩展到任意数量（受内存容量限制），而不会增加计算成本，并且在分片过程中不需要任何传输帮助。

模型并行性在消耗时跨设备（核心/机器）对模型进行分片批次中的相同数据，导致模型较大，由于顺序流而处理速度较慢；瓶颈是由通信引入的。另一方面，数据并行性使模型权重保持恒定并将数据分片，从而提高计算速度，但由于尺寸而导致模型容量较低。SwitchTransformer 采用数据和专家并行性，即每个数据都被分片给一台设备上的一名专家，该设备使用其专家权重进行计算。专家本身是分布式的，彼此之间不进行通信，从而实现完全并行性并降低计算开销。作者使用专家、数据和模型并行性来扩展到一个巨大的模型，即使是单个专家也无法容纳一个设备。

与 T5-Base 和 T5-large 相比，开关 transformer 表明，给定相同的 FLOPS，这些模型可以在不同的自然语言任务和不同的训练方案中，例如预训练、微调和多任务训练，其规模相当大，并且表现出色。多语言结果更令人印象深刻，因为它们显示了与单一语言相比的增益。跨 101 种语言的等效 MT5 模型。

5.5 案例研究：情绪分析

5.5.1 目标

此案例研究检查了针对情感跨度提取进行了微调的 T5 模型的注意力权重。该跨度提取模型的输入是包含正面或负面情绪的文本跨度。输出是导致该跨度具有指定情绪的输入文本的子序列。

5.5.2 数据、工具和库

```
pip install torch transformers sentencepiece bertviz
```

清单 5.1 Python 环境设置 Transformer Modifications 149 要使用这个模型，我们需要三个 Python 库：Huggingface 的 transformers、Google 的 sentpiece 和 bertviz 库。可以通过运行清单 5.1 中所示的 shell 命令来安装它们。一旦环境设置完毕，模型和分词器就可以加载到内存中，如清单 5.2 所示。

```
import torch
from transformers import T5ForConditionalGeneration,
AutoTokenizer
import numpy as np

# Use GPU, if available
device = torch.device("cuda" if torch.cuda.is_available()
else "cpu")
model_name = "mrm8488/t5-base-finetuned-span-sentiment-extraction"
tokenizer = AutoTokenizer.from_pretrained(model_name)
model = T5ForConditionalGeneration.from_pretrained(model_name)
model = model.to(device)
```

清单 5.2 加载模型和 tokenizer

要使用此模型提取情感跨度，您可以使用清单 5.3 中定义的函数。

```
def get_sentiment_span(text, sentiment):
    query = f"question: {sentiment} context: {text}"
    input_ids = tokenizer.encode(query, return_tensors="pt",
    add_special_tokens=True)
    input_ids = input_ids.to(device)
    generated_ids = model.generate(input_ids=input_ids,
    num_beams=1, max_length=80).squeeze()
```

```
predicted_span = tokenizer.decode(generated_ids,
    skip_special_tokens=True,
    clean_up_tokenization_spaces=True)
return predicted_span

text = "You're a nice person, but your feet stink."
get_sentiment_span(text, "positive")
# > 'nice person,'  
get_sentiment_span(text, "negative")
# > 'your feet stink.'
```

清单 5.3 从输入文本中提取情绪范围该模型采用以下形式输入：问题：[正面/负面] 上下文：[指定情绪的文本]。

本案例研究重点关注一系列包含正面和负面的文本中的正面情绪情绪。文字是“你是个好人，但你的脚很臭。”因此，对于积极情绪，跨度提取模型的完整输入文本是“问题：积极上下文：你是一个好人，但你的脚很臭。”

5.5.3 实验、结果和分析

有多种方法已经被证实。用于分析注意力机制对模型输出的影响。我们只看一个：简单地可视化注意力权重（使用 BertViz [176] ）。

1. 可视化注意力头权重 T5 模型有 12 层，每层有 3 个层注意机制：
 1. 编码器自注意力
 - (a) 解码器自注意力
 - (b) 交叉注意力

每个注意力机制有 12 个头，因此有 144 组注意力权重，每个层和注意力头的选择都有一组。我们将使用 BertViz [176] 库来查看注意力头的权重，如清单 5.4 所示。

```

from bertviz import head_view

def view_cross_attn_heads(text, sentiment,
    layer=None, heads=None):
    query = f"question: {sentiment} context: {text}"
    input_ids = tokenizer.encode(query, return_tensors="pt",
        add_special_tokens=True)
    input_ids = input_ids.to(device)
    with torch.no_grad():
        output = model.forward(input_ids=input_ids,
            decoder_input_ids=input_ids,
            output_attentions=True,
            return_dict=True)
    tokens = tokenizer.convert_ids_to_tokens(input_ids[0])

    head_view(output.cross_attentions, tokens,
        layer=layer, heads=heads)

def view_decoder_attn_heads(text, sentiment,
    layer=None, heads=None):
    query = f"question: {sentiment} context: {text}"
    input_ids = tokenizer.encode(query,
        return_tensors="pt", add_special_tokens=True)
    input_ids = input_ids.to(device)
    with torch.no_grad():
        output = model.forward(input_ids=input_ids,
            decoder_input_ids=input_ids,
            output_attentions=True, return_dict=True)

    tokens = tokenizer.convert_ids_to_tokens(input_ids[0])

```

```

head_view(output.decoder_attentions, tokens,
          layer=layer, heads=heads)

def view_encoder_attn_heads(text, sentiment,
                            layer=None, heads=None):
    query = f"question: {sentiment} context: {text}"input_ids =
        tokenizer.encode(query, return_tensors="pt",
                         add_special_tokens=True)
    input_ids = input_ids.to(device)
    with torch.no_grad():
        output = model.forward(input_ids=input_ids,
                               decoder_input_ids=input_ids,
                               output_attentions=True, return_dict=True)

    tokens = tokenizer.convert_ids_to_tokens(input_ids[0])
    head_view(output.encoder_attentions, tokens,
              layer=layer, heads=heads)

view_cross_attn_heads(text, "positive")
view_decoder_attn_heads(text, "positive")
view_encoder_attn_heads(text, "positive")

```

清单 5.4 可视化注意力权重

对于初学者来说，我们想要看看以下任意一个的权重：注意力头在输入文本中显示“正”一词，关注提取的子序列中的任何标记。

我们可以看到，“正”在编码器注意力的三个头中关注“好”。这些如图 5.16 所示。在交叉注意力机制的五个头中，“积极”和“好”都关注“人”。如图 5.17.

2. 分析

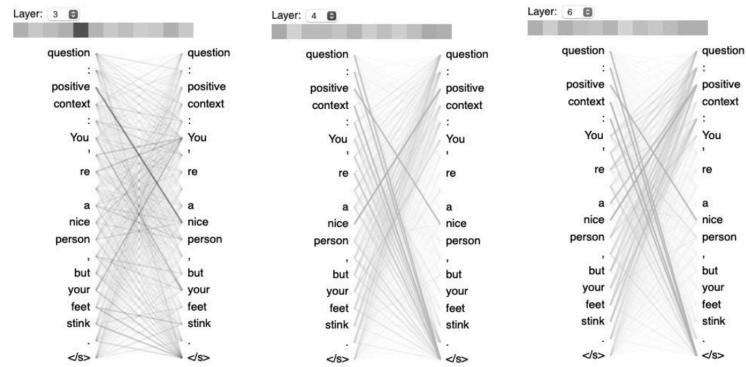


Figure 5.16 Encoder attention heads from layers 3 (head 5), 4 (head 11), and 6 (head 11), from left to right.

图 5.16 从左到右第 3 层（头 5）、4（头 11）和 6（头 11）的编码器注意力头。

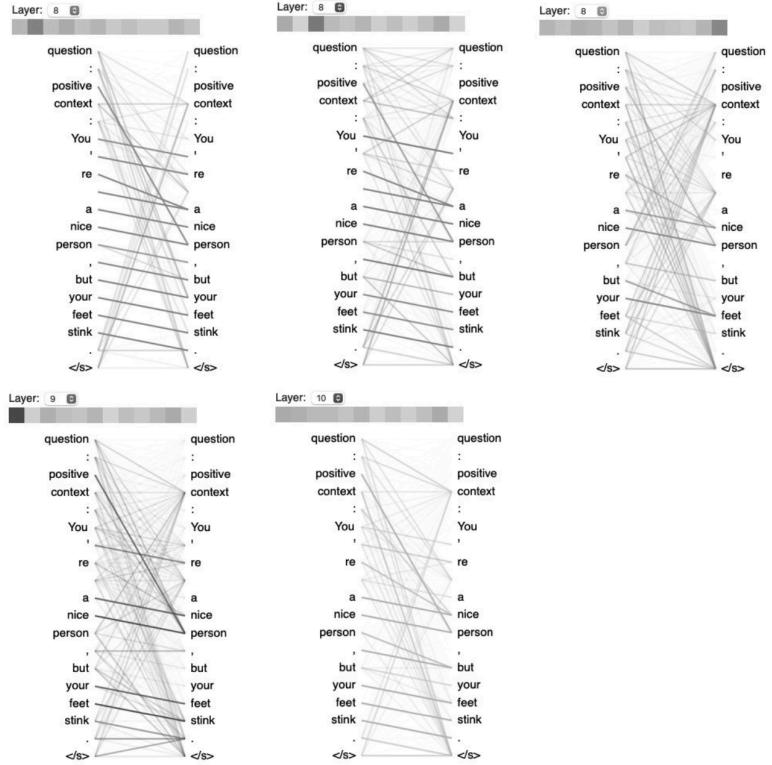


Figure 5.17 Cross-attention heads from layers 8, 9, and 10. The top row shows heads 1, 2, and 11 of layer 8 (left to right). The bottom row shows head 0 of layer 9 and head 1 of layer 10.

图 5.17 第 8、9 和 10 层的交叉注意力头。顶行显示第 8 层的头 1、2 和 11（从左到右）。底行显示第 9 层的 head 0 和第 10 层的 head 1。

虽然有 11 个注意力头（共 432 个）在“积极”和注意力之间表现出较强的注意力，但没有的甚至更多。跨度提取模型中的许多其他注意力头都属于 [177] 中所示的注意力模式类型，特别是每个标记关注几乎所有其他标记的广泛关注以及每个标记关注后续标记的类型。

上面的分析调用使用注意力权重本身来解释预测的能力受到质疑。最近，[178] 发现上面所示的注意力权重分析是一种很差的方法，用于分析注意机制。有几篇论文提出了其他分析方法。我们将在下面描述其中之一。

注意机制输出范数 [178] 建议使用其输出组件的范数来分析注意机制。回想一下 (5.29)，它展示了如何在标准转换器中为单个注意力头定义自注意力。如 (2.19) 所示，在连接注意力头之后，我们应用线性层来获得注意力机制的输出。线性层具有权重 \mathbf{W}_O ，为该机制提供以下输出：

$$\mathbf{y}_i = \left(\sum_{j=1}^L A_{ij} \mathbf{v}(\mathbf{x}_j) \right) \mathbf{W}_O \quad (5.76)$$

$$= \sum_{j=1}^L A_{ij} \mathbf{f}(\mathbf{x}_j) \quad (5.77)$$

其中 A_{ij} 是来自 (5.27) 的注意力权重， $\mathbf{v}(\mathbf{x}_j)$ 是来自 5.25 的值向量， $\mathbf{f}(\mathbf{x}_j) = \mathbf{v}(\mathbf{x}_j) \mathbf{W}_O$ 。 $\mathbf{f}(\mathbf{x}_j)$ 是对应于第 j 个输入标记 x_j 的注意力机制的输出。

在 [178]，Kobayashi 等人特别建议使用 $\|A_{ij}\mathbf{f}(\mathbf{x}_j)\|$ 来表示标记 i 关注标记 j 的程度。他们还表明 $\|\mathbf{f}(\mathbf{x}_j)\|$ 和 A_{ij} 是有意义的量，有时在确定模型行为方面发挥相反的作用。

6 预训练和特定应用

转换器架构被引入作为一种语言模型，将一种语言的文本转换为另一种语言的文本 [44]。自从最初应用于机器翻译以来，Transformer 架构已应用于计算机视觉、音频处理和视频处理以及 NLP 中的其他问题。transformer 架构已被证明是一种接收输入序列并将其转换为其他内容的可靠方法。

6.1 文本处理

本节介绍 transformer 应用于特定领域文本处理的一些方法以及文本序列中最近取得的进展文本序列处理。

6.1.1 特定领域的 Transformer

通过微调 BERT 等预训练模型，为特定领域的应用程序制作了 Transformer。我们将看一些。

1. BioBERT

BioBERT [179] 是通过在大量生物医学文本上微调 BERT 构建的特定领域语言模型。BioBERT 被创建的原因是 ELMo 和 BERT 在生物医学文本上表现不佳，因为它们的训练数据不包含生物医学文本。BioBERT 在对生物医学文本挖掘有用的三个 NLP 任务上优于以前的模型：命名实体识别 (NER)、关系提取和问答 (QA)。从预训练的 BERT 模型到 BioBERT 模型需要三个步骤。起点是预训练的 BERT 模型。下一步是在 PubMed 摘要和 PubMedCentral 全文文章上预训练模型。最后，在预训练结束后，BioBERT 使用特定于任务的数据集对 NER、关系提取和问答任务进行进一步微调。BioBERT 使用了 cased BERT 词汇，未做任何更改。如果他们不这样做，他们将无法从预训练的 BERT 模型开始。BioBERT 在 8 个 NVIDIA VoltaV100 GPU 上训练了 23 天。

2. SciBERT

SciBERT [180] 是一种为处理科学文本而构建的语言模型。它使用 BERT 架构，但接受了 Semantic Scholar 网站 (seman-ticscholar.org) 140 万篇论文的全文和摘要的训练。与 BERT 一样，SciBERT 使用具有 30,000 个 token 的 WordPiece 标记化，但它没有使用 BERT 的词汇。相反，SciBERT 的词汇是根据语义学者语料库构建的。SciBERT 在五项 NLP 任务上进行了评估：NER、PICO 提取、文本分类、关系提取和依存分析。PICO 提取是一项序列标记任务，应用于有关临床试验的论文，其中模型提取提及临床试验的参与者、干预措施、比较或结果的跨度。在单个 8 核 TPU 上的训练花费了一周时间。

3. FinBERT

使用 ULMFit 模型 [181]，Howard 和 Ruder 表明，在目标域的语料库

上调整已经预先训练的语言模型可以改善分类指标。FinBERT [182] 是研究相同的想法是否可以应用于 BERT 并取得相同或更大成功的结果。FinBERT 通过两种方式进行尝试：(1) 在大型金融语料库上进一步预训练；(2) 仅对金融情绪分析语料库中的句子进行预训练。对于预训练，FinBERT 使用路透社 TRC2 数据集的子集，已针对金融关键字进行过滤。生成的语料库包含超过 2900 万个单词，涵盖 46,143 个文档。对于情感分析，使用金融短语库数据集。它有来自 LexisNexis 的 4845 个句子。

6.1.2 文本到文本 Transformer

如第 5.3.2 节中讨论的，来自 Raffel 等人的 T5 模型。[68] 将标准 NLP 任务重新构建为文本到文本的转换，其中输入和输出是字符串。自 T5 推出以来，出现了两项重要的进步：名为 mT5[117] 的多语言版本，支持 101 种语言，以及直接在 UTF-8 字节上运行的无标记变体 ByT5[69]。在本节中，我们讨论 ByT5。

1. ByT5

ByT5[69] 是一个字节级模型，以利用被编码为 UTF-8bytes 序列的字符串。ByT5 的部分动机是无标记模型对于词汇表外的单词更加稳健、拼写错误、大小写和形态变化。无标记模型不会使用固定词汇表将测试映射到标记。字节级模型只需要 256 个嵌入向量，而不是使用固定词汇表时所需的大量向量。使用字节带来的词汇大小的大幅减少意味着可以在模型中的其他地方使用更多参数。

由于字节序列比标记序列长，并且变换器计算复杂度是序列长度的二次方（除了线性化注意力），早期的工作试图减轻使用字节级和字符级序列带来的复杂性增加。ByT5 从 mT5 架构开始，但在一些方面与 mT5 不同。首先，没有 SentencePiece 标记化。使用原始 UTF-8 字节作为输入，并学习 256 个字节中每个字节的嵌入。此外，词汇表中还添加了三种特殊代币：填充代币、EOS 代币和未知代币。改变预训练任务，使得输入跨度的屏蔽部分的长度比 T5 中的长。在 T5 中，编码

器和解码器模块具有相同的层数。ByT5 中删除了该限制，编码器的层数是解码器的三倍。最后，从输出中删除非法的 UTF-8 字节。

如上所述，鲁棒性是使用字节而不是标记的假设好处之一。为了测试假设的鲁棒性，他们作者向数据中添加了六种噪声，并观察它如何影响性能：1. 通过给每个字符有 10% 的机会被删除来进行删除。2. 给每个字符 10% 的机会添加、删除或变异（可能性相等）。3. 给每个字符 20% 的机会被重复 1-3 次。4. 每个字符大写并用空格填充。5. 当语言使用大小写时，将每个字符设为大写。

(a) 当语言使用大小写时，随机设置每个字符的大小写。

噪声以两种方式之一注入：注入微调和评估数据或仅注入评估数据。对于问答任务，噪声会添加到上下文中，但不会添加到问题或答案中。对于句子蕴含训练任务，在前提和假设中添加噪声。对于这两种类型的噪声，论文表明 ByT5 确实比 mT5 对噪声更鲁棒。

6.1.3 文本生成

基于 Transformer 的语言模型最著名的成功之一是使用 GPT-2 和 GPT-3 语言模型进行文本生成，它们是由 transformer 解码器层堆栈构建的。

1. GPT：生成预训练

生成预训练模型系列是语言建模趋势的一部分，其中模型首先以与任务无关的方式在无监督数据上进行训练，然后针对特定任务进行微调。该系列中的第一个模型是同名的 GPT [183]，它首先在大量未标记文本上预训练一堆 Transformer 解码器层，然后在标记的、特定于任务的数据上进行微调。GPT 是一种自回归模型，这意味着它使用序列前面步骤的输入来预测序列后面的值。GPT 的评估有四种自然语言理解任务：自然语言推理、问答、语义相似性和文本分类。

无监督预训练在此阶段，GPT 从标记语料库开始，并通过它，学习如何在给定一些先前的上下文的情况下预测下一个标记。更正式地说，给定一个未标记的语料库 $U = (w_1, \dots, w_n)$ ，模型学习给定前面的 k tokens，

$P(w_t|w_{t-1}, \dots, w_{t-k})$ 的情况下预测 token w_t 的条件概率，通过最小化负对数似然

$$L_1(U) = - \sum_t \log P(w_t|w_{t-1}, \dots, w_{t-k}; \Theta) \quad (6.1)$$

其中 Θ 表示模型参数。优化采用随机梯度下降。

监督微调在此阶段，模型在标记的、特定于任务的语料库 C 上进行微调，其中每个数据点都是一个标记序列 $\mathbf{x} = (x_1, \dots, x_m)$ 和一个类标签 y 。预训练的解码器模型用作标记数据的特征生成器，并附加一个带有 softmax 激活和权重矩阵 W 的全连接线性层，并通过最小化第二负对数似然

$$L_2(C) = - \sum_{(\mathbf{x}, y)} \log P(y|\mathbf{x}; \mathbf{W}) \quad (6.2)$$

Radford 等人发现当将无监督阶段的语言建模目标 (6.1) 添加到 (6.2) 时，模型收敛得更快并且泛化得更好。因此，完整的目标是加权和 $L_2(C) + \lambda L_1(C)$ 。

格式化数据以进行微调四个训练任务中每一个的数据格式不同：

- 文本分类数据具有简单的格式；每个实例都用开始和结束标记括起来，因此输入的格式类似于 $[\langle s \rangle, text, \langle /s \rangle]$ 。
- 自然语言推理 (NLI) 实例有两部分：前提 p 和假设 h 。标签可以是蕴含的、矛盾的或中性的。输入的格式类似于

$$[\langle s \rangle, p, \$, h, \langle /s \rangle]$$

$\$$ ，其中 $\$$ 是分隔符标记。

- 文本相似性实例有两个文本序列，可以是任意顺序；所以这两个订单都包括在内。输入的格式类似于蕴涵实例。

- 对于问答和常识推理，又名多项选择问题，每个实例都是上下文序列、问题和一组 k 个可能的答案 $\{ak\}$ 。从这样一个实例中，`kmodelinputs` 是通过连接上下文、问题、\$ 分隔符标记和可能的答案之一来构造的。每个都用作单独的输入。

模型训练该模型是 12 个 Transformer 解码器层的堆栈，具有 12 个屏蔽自注意力头。模型维度为 768，前馈层使用 $dff = 3072$ 。学习了位置嵌入，而不是标准 Transformer 中使用的固定嵌入。

预训练数据语言模型预训练在 BooksCorpus [184] 和 1B Word Benchmark 上进行 [185] 数据集。

微调数据一些数据来自 GLUE 多任务基准测试 [186]。使用了五个 NLI 数据集：SNLI、MNLI、QNLI、SciTail 和 RTE。多项选择数据来自 RACE 数据集 [187]。评估基于 Story Cloze Test 数据集 [188]。语义相似度位于 Microsoft Paraphrase 语料库 (MRPC) [189]、QuoraQuestion Pairs (QQP) 数据集 [43] 和语义文本相似度基准 (STS-B) [190] 上。文本分类在语言可接受性语料库 (CoLA) [191] 和斯坦福情感树库 (SST-2) [192] 上进行了评估。

2. GPT-2

如果第一个 GPT 模型证明了生成预训练的强大功能，那么 GPT-2 [193] 表明语言模型可以学习特定的 NLP 任务，而无需在这些任务上进行显式训练。这与第 3 章和第 4 章中讨论的模型的零样本应用相同。

模型架构与 GPT 类似，GPT-2 使用一堆 transformer 解码器块。解码器略有修改。在标准 transformer 中，层范数模块位于多头注意力和位置前馈网络之后，作为剩余连接的一部分。在 GPT-2 中，层规范模块位于多头注意力之前和位置前馈之前。剩余连接现在只包括加法，而不是加法和层范数。在多头注意力之后，一个附加的层规范模块被放置在最终的解码器块中。残差层的权重初始化方式与 GPT 模型不同。残差层的权重除以 $1/\sqrt{N}$ ，其中 N 是整个模型中残差层的数量。

GPT-2 使用字节对编码 (BPE) 标记化 [194]，以便可以使用词汇表来表示任何 UTF-8 字符串只有 256 字节。此处未使用原始 UTF-8 字节

进行计算，因为字节级语言模型未在字级语言模型的级别上执行。训练了具有所描述架构的四种变体。

这四种型号中最大的型号是“GPT-2”。它有 15.42 亿个参数，使用 48 个 Transformer 解码器层。每个数据集都在多个语言建模数据集上进行了评估，无需任何额外的训练。GPT-2 在八个数据集中的七个上实现了最先进的水平。

3. GPT-3

GPT-3[67] 是 Transformer 语言模型趋势的一部分其中参数数量的增加会导致语言模型执行下游任务的能力增强，而几乎不需要特定于任务的训练。

模型架构 GPT-3 使用与 GPT-2 相同的模型架构，但有一个例外：transformer 层中的注意力机制在密集和局部带状之间交替稀疏的图案。稀疏带状模式在稀疏变换器 [195] 中使用，仅计算稀疏选择的查询键对之间的注意力。

为了研究上下文学习中参数计数的影响，Brown 等人。训练了八个不同大小的模型。最小的模型“GPT-3 Small”有 1.25 亿个参数，12 个 Transformer 解码层，模型维度为 768，12 个注意力头，每个注意力头的维度为 64 ($d_k = d_v = 64$)，批量处理的 tokens 为 50 万个，并使用学习率 6e-4。最大的模型“GPT-3 175B”或简称“GPT-3”，有 1750 亿个参数，96 个 Transformer 解码层，模型维度为 12288，96 个注意力头，头维度为 128($d_k = d_v = 128$)，已处理批量 320 万个 token，使用学习率 0.6e-4。GPT-3 的参数数量是 GPT-2 的 1000 多倍。所有模型均使用 3000 亿个 token 进行训练，并使用 2048 个 token 的上下文窗口。请注意，位置前馈网络的 $d_{ff} = 4d_{model}$ 。使用较大批量大小的较大模型使用较小的学习率。

性能 针对零次、一次和几次学习，对几种大小的 GPT-3 模型进行了评估，Brown 等人。描述为不同类型的情境学习。通过小样本学习类型，GPT-3 为模型提供了适合上下文窗口的所需任务的许多示例（10-100 个示例）。一次性学习提供一个示例和任务描述，而零次学习不提供示

例，仅提供任务描述。没有梯度更新或微调。模型在 12 个 NLP 数据集上进行评估。

结果表明，随着模型参数数量的增加，模型需要更少的演示来学习如何执行任务（从而达到给定的精度）目标）。当使用少样本时，模型性能随着模型大小的增加而提高得更快，这表明较大的模型更适合上下文学习。

模型的评估和调节对于少样本， K 个示例是从训练集（或开发集，如果有的话）中随机抽取的。没有标记的训练集）用作调节，由 1-2 个换行符分隔。 K 的变化范围为 0 到上下文窗口的大小。一些示例仅描述了用作调节的任务。对于多项选择任务， K 个具有正确选择的上下文示例被用作条件，并且仅附加一个上下文示例作为要完成的项目。与 GPT-2 一样，对每个可能完成的语言模型可能性评分进行评分。二元分类通常被视为多项选择题，其中可能的答案是有意义的单词，例如“肯定”或“否定”。在某些情况下，此任务的格式与 T5 模型的格式相同。使用波束搜索完成填空任务。请参阅开创性论文，了解用于评估 GPT-3 的每个数据集的示例。

每个尺寸的 GPT-3 模型还使用零次、一次或几次条件条件在其他几个任务上进行了测试：解读单词、SAT 类比、算术、在一个例子中使用不常见的单词。定义单词后生成句子，生成新闻文章文本，并纠正英语语法。

6.2 计算机视觉

Transformer 在图像处理中最明显的应用是图像识别（也称为图像分类）。在 Transformer 出现之前，图像识别的最高质量来自卷积神经网络（CNN）[11]。有一些应用于图像识别的纯 Transformer 模型可以与最先进的 CNN 模型相媲美。在本节中，我们重点关注视觉变换器（ViT）[161]，引入它是为了了解纯变换器模型如何有效地用于计算机视觉。

6.2.1 视觉 Transformer

给定分辨率为 $H \times W$ 和 C 通道的图像，该图像可以用 $\mathbf{x} \in \mathbb{R}^{H \times W \times C}$ 表示。ViT 首先将二维图像分解为一系列 N 个图像块， $\mathbf{x}_p \in \mathbb{R}^{N \times P^2 \cdot C}$ ，分辨率为 $P \times P$ ，其中 $N = HW/P^2$ 。图像补丁的序列就像标准转换器 [44] 中的标记序列。

在通过嵌入层发送补丁序列之前，可学习的嵌入类似于 BERT 中的 [CLS] 标记， \mathbf{x}_{cls} 被添加到每个补丁向量上。所以 $\mathbf{x}_p \rightarrow [\mathbf{x}_{cls}; \mathbf{x}_p]$ 。使用嵌入层 $\mathbf{E} \in \mathbb{R}^{(P^2 \cdot C) \times D}$ 和一维位置编码 $\mathbf{E}_{pos} \in \mathbb{R}^{(N+1) \times D}$ ，我们计算 Transformer 的输入。 D 是 Transformer 隐藏大小。

$$\mathbf{z}_0 = [\mathbf{x}_{cls}; \mathbf{x}_p^{(1)} \mathbf{E}; \dots; \mathbf{x}_p^{(N)} \mathbf{E}] + \mathbf{E}_{pos} \quad (6.3)$$

从这里开始， \mathbf{z}_0 被传递到带有 L 编码器层的大多数标准 Transformer 编码器架构中：

$$\mathbf{z}'_l = MultiHeadAttn(LayerNorm(\mathbf{z}_{l-1}) + \mathbf{z}_{l-1}) \quad (6.4)$$

$$\mathbf{z}_l = FeedFwd(LayerNorm(\mathbf{z}_l) + \mathbf{z}'_l) \quad (6.5)$$

$$\mathbf{y} = LayerNorm(\mathbf{z}_L^0) \quad (3)$$

其中 $l = 1, \dots, L$ 且 \mathbf{z}_L^0 是最终编码器层的分类嵌入。此外，前馈层有两个全连接层，后面跟着一个 GELU 激活函数。ViT 与标准 transformer 的区别之一是，在 ViT 中，在发生剩余连接之前，将层范数运算应用于前一层的输出。在标准 Transformer [44] 中，在 LayerNorm 之前添加了残差连接。

通过 ViT 的一系列实验，[161] 证明 CNN 引入的归纳偏差对于小型数据集很有用，但对于较大的数据集则无效。对于更大的数据集，模型可以自行学习相关相关性，正如各种 Transformer 所显示的那样。ViT 还表明，补丁之间的空间关系（图像内的距离）是通过位置编码来学习的。彼此接近的补丁最终会具有相似的位置编码。二维空间相关性也是通过位置编码来学习的，即同一行或列中的块具有相似的位置编码。实验还表明，将图像块的二维结构硬编码为位置编码并不能提高质量。这可能是因为将归纳偏差构建到

像 Transformer 这样通用的模型中会阻止它自己学习什么是重要的或什么是不重要的。

最后，Vision Transformer 研究了对自注意力机制的修改，即轴向注意力 [196, 197]。轴向注意力，注意力集中在同一行或同一列的块之间。ViT actually 创建了轴向变换器块，其中有一个行注意机制，后面有一个列注意机制。

6.3 自动语音识别

自动语音识别（ASR），即系统自动将语音音频信号转换为文本，提出了计算机视觉和 NLP 任务中不存在的独特挑战。语音信号是连续值的序列，而不是像图像或文本中看到的离散值。这意味着您无法以与计算机视觉或 NLP 相同的方式进行预训练。例如，在 NLP 中，您有一个离散字或子字单元，在输入信号本身中定义。同样，在计算机视觉中，您在输入信号中拥有图像子单元，即像素。语音信号中没有内置子单元。这使得预训练变得困难，无论是有掩饰的还是无掩饰的。本节介绍 transformer 应用于 ASR 的三种方式。预训练和特定于应用的 transformer

6.3.1 Wav2vec 2.0

wav2vec 2.0 [154] 的第一步使用卷积特征编码器（它是 CNN）将输入语音音频 \mathbf{X} 转换为表示序列 (z_1, \dots, z_T) 。序列的长度 T 是音频中的时间步数。然后，语音表示序列中的一些跨度被屏蔽。

编码能够被学习，因为语音被分解为离散语音单元类似于用作文本转换器输入的 WordPiece 标记。语音单元是音频序列的离散单元的有限集合，并且比音素短（它们的长度为 25 毫秒）。潜在语音编码为类似于在文本转换器的初始嵌入层中学习的嵌入。这些屏蔽编码被传递到转换器中以构建上下文化表示。对比损失函数 [198, 105] 让 wav2vec 2.0 转换器学习。

请注意，离散语音单元还可以进行跨语言训练，其中模型学习哪些单元仅用于特定语言以及哪些单元跨多种语言使用。

训练过程正如我们在本书中所看到的，跨语言-前者可以使用大型语料库的预训练来学习自然语言的各种特征，然后针对给定的任务集进行微调。

这就是 wav2vec 2.0 所发生的情况，只不过它结合了无监督和自监督预训练，只需要相对少量的标记数据进行微调。

6.3.2 Speech2Text2

Speech2Text2 [199] 是无监督预训练的另一个强大应用。将 Transformer 训练为 ASR，其中将解码器块添加到 wav2vec 2.0 编码器之上。预训练后，模型使用 CTC 损失进行微调。解码器仅在微调期间进行训练，而不是预先训练。

由于 Speech2Text2 使用解码器，因此它还可以用于组合机器翻译和 ASR，即音频可以是英语，转录可以是法语。王等人。还针对此任务对 Speech2Text 进行了微调，在某些语言对上实现了新的最先进技术。

6.3.3 HuBERT：隐藏单元 BERT

隐藏单元 BERT 为了解决没有预定义语音单元的问题，Hsu 等人。介绍 HuBERT（隐藏单元 BERT）[200]。HuBERT 使用聚类来生成噪声语音子单元，然后可以将其视为可以进行类似 BERT 的预训练的子词。

HuBERT 使用噪声标签作为输入标记序列，可以对这些标记进行屏蔽以进行 BERT 预训练。因此，就像 BERT 学习未屏蔽文本的表示以便正确预测屏蔽标记一样，HuBERT 学习未屏蔽的噪声语音子单元的表示以便它可以正确预测屏蔽语音单元。

在 HuBERT 中，原始语音信号为通过 CNN 编码器产生长度为 T 的编码语音信号 $X = [x_1, \dots, x_T]$ ，与 wav2vec 2.0 非常相似。然后对编码语音信号中的一组索引进行掩码。该屏蔽序列用作类 BERT 转换器的输入。HuBERT 还以与 wav2vec 2.0 相同的方式执行掩蔽。

HuBERT 在原始语音信号的 MFCC 特征 [201] 上使用 k 均值来查找非平凡的离散隐藏单元 [202]。它称这些为 $[z_1, \dots, z_T]$ ，其中 $zt \in [C]$ 是 C 类的分类变量。这些 zt 被用作 BERT 预训练准确预测所需的输出“标记”。许等人。还尝试使用聚类模型集合来代替单个 k 均值，然后进行乘积量化 [203]。预训练后，HuBERT 使用标准联结时间分类 (CTC) 损失来使用标记的模型来微调模型音频数据。请注意，CNN 特征编码在微调过程中会冻结。

6.4 多模式和多任务 Transformer

Transformer 在许多 NLP 任务中被证明非常有效之后，Transformer 甚至在涉及图像、文本和视频等各种模态的组合任务中也显示出其有效性。特别是围绕图像和文本相结合的任务，人们提出了许多 Transformer 架构来扩展基本 BERT 模型，以将视觉输入与语言输入结合起来 [204, 205]。在本节中，我们将介绍一些更通用和采用的架构。

6.4.1 视觉和语言 BERT (VilBERT)

视觉和语言 BERT (VilBERT) 是一种用于学习与图像和文本任务无关的表示的联合模型 [204]。首先使用对象检测网络将图像转换为映射到特征向量的区域序列流。文本遵循映射到位置和单词嵌入的标记的正常流作为序列流。如图 6.1 所示，VilBERT 使用标准变换器模块 (TRM) 和共同注意变换器模块 (Co-TRM)，提供两种模态之间的稀疏交互。Co-TRM 模块计算查询、键和值矩阵，类似于标准转换器块，用于任何中间层的视觉和语言表示。然而，每个模态的键和值都作为另一个模态的多头注意力块的输入提供。因此，Co-TRM 的多头注意力模块为每种模式产生以另一种模式为条件的注意力池特征，并实现联合学习。该架构允许多层 TRM 和 Co-TRM 进行细化，并且与任务相关。输出与任务相关，并且可以像多层感知器 (MLP) 一样简单，后跟一个软最大层来计算给出图像和文本之间相似性的分数。可以有多个输出对应于多个任务，所有输出都是并行学习的。

VilBERT 的训练通过与 inception 网络训练类似的各个阶段进行。首先，文本和图像独立训练。BERT 模型在大型语料库上针对两项任务进行端到端训练：掩码语言建模 (MLM) 和 Next Sentence Prediction (NSP)。基于 Faster R-CNN 的预训练对象检测网络从图像中提取边界框及其视觉特征，并通过网络对其进行处理。接下来，Conceptual Captions 168 Transformers for Machine Learning：包含大量图像示例及其标题的 Deep Dive 数据集，用于联合学习任务，例如将图像中的区域映射到文本以及从图像中预测屏蔽字。最后，该模型经过训练，用于解决涉及视觉和文本模式的特定任务，例如视觉问答 (VQA)、视觉常识推理 (VCR)、图像检索和相位接地。研究表明，与现有的特定任务模型相比，VilBERT 在所有四项任务中都取得了最

先进的结果。

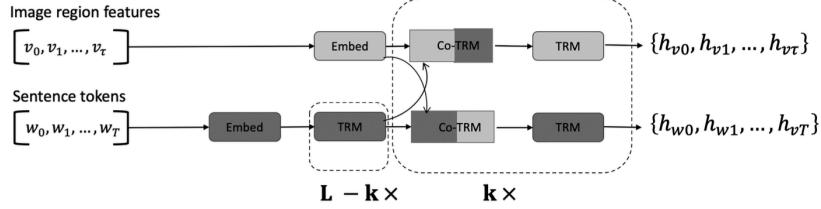


Figure 6.1 ViLBERT 处理文本和图像，用于学习多模态任务的联合表示。

图 6.1 ViLBERT 处理文本和图像，用于学习多模态任务的联合表示。

6.4.2 Unified Transformer (UniT)

在他们的工作中，Transformer is All You Need: Multimodal Multitask Learning with a Unified Transformer [205]，Hu 和 Singh 提出了一种 UnifiedTransformer (UnitT)，它可以跨文本和图像的模态联合学习并执行七个任务。

如图 6.2 所示，对于每种模态，都有一个编码器网络，并且编码器网络的输出被连接起来给出联合代表。组合的编码器输出流经具有多头交叉注意块的解码器网络，用于特定于任务的输入嵌入，将结果输出到每个特定于任务的头，例如对象类检测、视觉问题回答、情感分类器等。

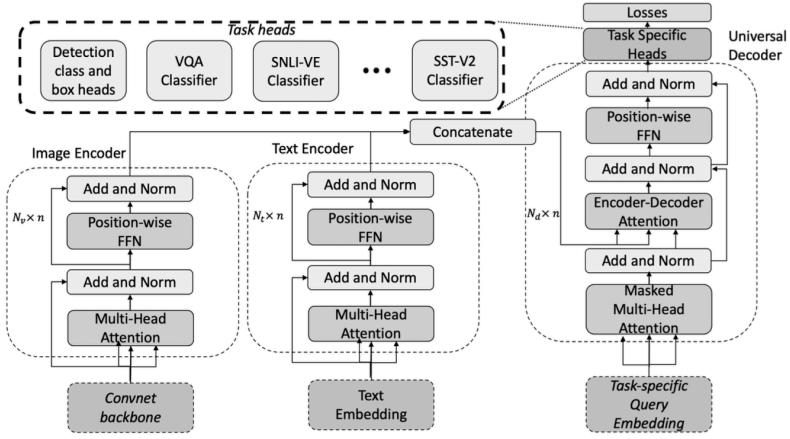


Figure 6.2 UniT processing text and image for learning joint representation for multi-modal tasks.

6.2 用于学习多模态任务联合表示的文本和图像处理单元。

对于训练仅视觉和特定于视觉语言的任务，图像首先由卷积神经网络块处理，然后由 Transformer 编码器处理以获得隐藏的视觉表示。此外，考虑到不同的视觉相关任务（例如目标检测和 VQA）可能需要特定于任务的信息，任务嵌入向量被添加到 transformer 编码器中。文本编码器使用预先训练的 BERT 和文本嵌入以及与图像编码器类似的具体于任务的嵌入向量。解码器与领域无关，从流入多头交叉注意块的模态中获取串联输入，并将具体于任务的查询流入解码器 transformer 的输入多头注意。训练联合发生在多个任务上，其中在每次训练迭代时选择一个随机任务和一个数据集来填充一批样本。Unit 模型在 8 个数据集的 7 个任务上显示出与每个领域的既定先前工作相当且稳健的性能，并保持参数相似。

6.5 使用 TIMESFORMER 进行视频处理

Transformer 可以通过多种方式应用于视频。本节将讨论这样一种应用程序，视频分类/动作识别。

TimeSformer [162] 回答了是否可以使用自注意力而不是卷积来进行高性能视频分类或动作识别。TimeSformer 使用改进的注意力机制，并使用 Vision Transformer 将图像分解为一系列补丁的方法作为其起点。

6.5.1 补丁嵌入

将视频剪辑分解为一系列图像补丁，类似于第 6.2 节中 Vision Transformer 的方式。1 将二维图像分解为图像补丁序列。模型的输入是具有 FRGB 帧、分辨率为 $H \times W$ 的视频。可以用一个四维矩阵 $X \in \mathbb{R}^{H \times W \times 3 \times F}$ 来表示。输入中的每个帧 ($\mathbb{R}^{H \times W \times 3}$) 然后被分解为大小为 $P \times P$ 的 N 个不重叠的块，其中 $N = HW/P^2$ 。每个补丁都变成一个向量 $\mathbf{x}(p, t) \in \mathbb{R}^{3P^2}$ ，其中 $p = 1, \dots, N$ 是补丁索引， $t = 1, \dots, F$ 是时间/帧索引。

下一步是生成一个嵌入向量 $\mathbf{z}_{(p,t)}^{(0)} \in \mathbb{R}^D$ ，对于每一个补丁，其中 D 是 Transformer 的隐藏大小：

$$\mathbf{z}_{(p,t)}^{(0)} = \mathbf{x}(p, t)\mathbf{E} + \mathbf{e}_{(p,t)}^{pos} \quad (6.7)$$

其中 $\mathbf{e}_{(p,t)}^{pos} \in \mathbb{R}^D$ 是一个位置编码，包括空间位置和帧索引， \mathbf{E} 是一个学习嵌入层， D 是一个隐藏层 den 尺寸（类似于标准 transformer 中的 d_{model} ）。请注意， $\mathbf{z}_{(p,t)}^{(0)}$ 的上标 (0) 表示这是第一个 TimeSformer 层，其索引为 0。就像在 Vision Transformer 中一样，TimeSformer 在 $\mathbf{z}_{(p,t)}^{(0)}$ 对应于 BERT 的特殊分类标记 $z_{0,0}^{(0)}$ 。

6.5.2 自注意力

在 TimeSformer 中，自注意力机制几乎与所有其他转换器相同。在大多数 Transformer 中，多头注意力的输入是查询、键和值矩阵。与 5.24 类似，each 是将输入嵌入变换到不同向量空间的结果。每个向量空间分别由矩阵 \mathbf{W}_Q 、 \mathbf{W}_K 和 \mathbf{W}_V 参数化。 \mathbf{W} 矩阵用于将输入嵌入旋转到查询、键和值空间中。因此，TimeSformer 层中单个补丁的查询、键和值向量 1 是

$$\mathbf{q}_{(p,t)}^{(l,h)} = \text{LayerNorm}\left(\mathbf{z}_{(p,t)}^{(l-1)}\right) \mathbf{W}_Q^{(l,h)} \in \mathbb{R}^{D/H} \quad (6.8)$$

$$\mathbf{k}_{(p,t)}^{(l,h)} = \text{LayerNorm}\left(\mathbf{z}_{(p,t)}^{(l-1)}\right) \mathbf{W}_K^{(l,h)} \in \mathbb{R}^{D/H} \quad (6.9)$$

$$\mathbf{v}_{(p,t)}^{(l,h)} = \text{LayerNorm}\left(\mathbf{z}_{(p,t)}^{(l-1)}\right) \mathbf{W}_V^{(l,h)} \in \mathbb{R}^{D/H} \quad (6.10)$$

H 是注意力头的数量, l 是层索引, 并且这是注意力头指数。 D/H 类似于香草 Transformer 中使用的键/值维度。请注意, TimeSformer 赋予查询、键和值相同的维度, 并且每个 W 矩阵是 $\in \mathbb{R}^{H \times D}$ 。请注意, 前一层的输出在查询、键和值计算之前经过 LayerNorm 操作; 这就像 Vision Transformer 中的一样, 如第 6.6 节所示。

1. 时空自注意力

TimeSformer 使用缩放的点积注意力, 如第 5.23 节所示。使用上面定义的查询、键和值向量, 我们可以计算 patch (p, t) 和第 l 层中的补丁

$$a_{(p,t),(p',t')}^{(l,h)} = \text{softmax} \left(\mathbf{q}_{(p,t)}^{(l,h)} \cdot \mathbf{k}_{(p',t')}^{(l,h)T} \right) \quad (6.11)$$

$$= \frac{\exp \left(\mathbf{q}_{(p,t)}^{(l,h)} \cdot \mathbf{k}_{(p',t')}^{(l,h)T} \right)}{\sum_{(u,r) \in S_{(p,t)}} \exp \left(\mathbf{q}_{(p,t)}^{(l,h)} \cdot \mathbf{k}_{(u,r)}^{(l,h)T} \right)} \quad (6.12)$$

其中 $S(p, t)$ 是补丁 (p, t) 可以参与的补丁集。注意, 由于 $\mathbf{z}_{(p,t)}^{(l)}$ 包含在 $(0, 0)$ 处嵌入的分类, 所以查询、键和值包括分类标记项: $\mathbf{q}_{(p,t)} = [\mathbf{q}(0, 0); \mathbf{q}(p, t)]$ 。为了简洁起见, 第 6.12 节中的平方根因子已被省略。

补丁 (p, t) 的 $NF + 1$ 权重可以分组为简单向量

$$\mathbf{a}_{(p,t)}^{(l,h)} = \text{softmax} \left(\mathbf{q}_{(p,t)}^{(l,h)} \cdot \left[\mathbf{k}_{(0,0)}^{(l,h)T}; \mathbf{k}_{(p',t')}^{(l,h)T} \right] \right) \quad (6.13)$$

通过观察补丁 (p, t) 和补丁 (p', t') 的相对位置, 可以看出 6.12 节中有两种类型的注意力。 (p, t) 只允许关注同一帧中的块的情况, 即 $t' = t$, 是空间注意力。这减少了需要从 $NF + 1$ 到 $N + 1$ 计算的点积的数量。如果 (p, t) 只允许处理不同帧中的补丁, 但在相同的位置 $(p' = p)$, 那么我们有暂时的关注。这减少了需要从 $NF + 1$ 到 $F + 1$ 计算的点积的数量。上述两种简化相当于一个注意力掩码, 并且在计算 softmax 操作的归一化因子时需要添加到 $S_{(p,t)}$ 中。多头自注意力计算的其余部分按照 transformer 中的预期进行。

2. 时空注意力块 论文 [162] 考虑了四种限制 $S(p,t)$ 的方法：仅空间注意力、分割时空注意力、稀疏注意力局部和全局注意力、轴向注意力。

时空分离注意力是指具有独立的空间和时间注意力机制；时间块，随后是空间块。计算多头时间注意力的输出嵌入 $z'_{(p,t)}^{(l)}$ ，然后将其作为输入馈送到多头空间注意力块中。在该方案中，时间块和空间块都有自己独立的查询、键和值空间。由于块是不顺序的，因此每个补丁 (p,t) 有 $F + 1 + N + 1 = N + F + 2$ 个查询键比较。请注意，注意力块的顺序是通过实验发现的。稀疏局部和全局注意力是局部时空注意力的组合，其中局部性被定义为相邻的 $F \times H/2 \times W/2$ 块，以及时间和空间维度上步长为 2 的全局注意力。根据定义，稀疏局部和全局注意力是对完整时空注意力的粗粒度近似。最后，轴向注意力类似于视觉变换器中的注意力，其中同一行或同一帧内的补丁之间存在空间注意力。柱子；不同帧中同一位置的块之间也存在时间注意力。

随着空间分辨率的增加和视频变长，分割时空注意力的尺度比联合时空注意力更好，因为空间和时间分量在分割情况下是分离的。请注意，对于单个补丁，空间和时间注意力都是线性的。空间为 $O(N)$ ，时间为 $O(F)$ ，因此划分的尺度为 $O(N+F)$ 。联合注意力为 $O(N \cdot F)$ 。

论文发现，在很多情况下，空间注意力比时间注意力更重要。但是，在某些情况下，时间注意力非常重要。另一个发现是，分割的时空注意力能够比完整的联合时空注意力学到更多，因为分割的情况将它们视为两个独立的注意力机制，因此原则上它有两倍的参数并且可以学到更多。正因为如此，推荐的注意力方法是时空注意力划分。

6.6 图 Transformer

Transformer 可以应用于图数据集吗？当 transformer 使用完全注意机制时，这意味着它没有硬编码的稀疏性，它将输入序列视为全连接图。对于文本、图像、视频等来说都是如此。我们在 5.2.2.4 节中使用 Big Bird 看到了文本数据，在 6.2.1 节中看到了使用 Vision Transformer 的图像，在 6.5 节中看到了 TimeSformer 的视频。全连接注意力让模型了解每个连接的相对重要性。然而，对于图数据集，任何给定的节点都可能有如此多的链接，

以至于使用全连接注意力在计算上会很困难，因为对于简单序列来说，全注意力已经具有二次复杂度。这就是 [206] 中引入的图变换器的目的。它通过让节点关注其本地邻域中的其他节点来解决自注意力的复杂性。

6.6.1 图中的位置编码

如第 5.2.1 节中所述，缩放点积注意力机制在时间和内存方面都具有二次复杂性。由于图可以有大量的节点，为了使图变换器在计算上可行，任何节点的注意力都必须存在局部稀疏性。问题在于，一般图没有节点之间距离的概念，这使得使用位置编码来提供距离或局部性的测量变得很重要，这在 Transformer 中很常见。如 [206] 中所述，这个问题通过使用拉普拉斯位置编码 [207] 来解决，拉普拉斯位置编码是通过谱嵌入到欧几里得空间中生成的。

1. 拉普拉斯位置编码

具有 n 个节点的图的拉普拉斯位置编码是根据图的邻接矩阵和度矩阵计算的：

$$\Delta = \mathbb{I} - \mathbf{D}^{-1/2} \mathbf{A} \mathbf{D}^{-1/2} \quad (6.14)$$

$$= \mathbf{U}^T \boldsymbol{\Lambda} \mathbf{U}, \quad (6.15)$$

其中 \mathbf{A} 是邻接矩阵， \mathbf{D} 是度矩阵， Δ 是特征值矩阵（对角线）， \mathbf{U} 是特征向量矩阵。位置编码对于节点 $i \lambda_i$ 来说，是通过取 k 个最小的非平凡特征向量来构造的，每个特征向量都经过归一化，并随机分配一个 -1 因子给每个特征向量。

6.6.2 图 Transformer 输入

我们可以将图 \mathcal{G} 描述为节点和节点之间的边的集合。每个节点 i 有特征 $i \in \mathbb{R}^{d_n \times 1}$ ，其中 d 是节点特征的数量。节点 i 和 j 之间的每条边都有特征 $\beta_{ij} \in \mathbb{R}^{d_e \times 1}$ ，其中 d_e 是边特征的数量。图 Transformer 的输入层有两个嵌入层，一个用于嵌入节点，一个用于嵌入边。两个嵌入层都产生 d - 维嵌入，导致节点 i , $\tilde{\mathbf{h}}_i^{(0)}$ 的节点嵌入和节点 i 和 j 之间的边缘嵌入

$$\tilde{h}_i^{(0)} = \mathbf{A}^{(0)}\alpha_i + a^{(0)} \quad (6.16)$$

$$\tilde{e}_{ij}^{(0)} = \mathbf{B}^{(0)}\beta_i + b^{(0)} \quad (6.17)$$

其中 $\mathbf{A}(0) \in \mathbb{R}^{d \times d_n}$ 和 $\mathbf{B}(0) \in \mathbb{R}^{d \times d_n}$ 分别是节点和边嵌入矩阵， $a^{(0)}$ 和 $b^{(0)}$ 分别是节点和边的偏置项。上标 (0) 表示这是输入层。

拉普拉斯位置编码 $\boldsymbol{\lambda}$ 也可以通过附加的可学习嵌入层 $C^{(0)} \in \mathbb{R}^{d \times k}$ 嵌入到 d -维空间中，以生成拉普拉斯位置嵌入 $\boldsymbol{\lambda}_i^{(0)}$ ：

$$\boldsymbol{\lambda}_i^{(0)} = C^{(0)}\boldsymbol{\lambda}_i + \mathbf{c}^{(0)} \quad (6.18)$$

$$\mathbf{h}_i^{(0)} = \tilde{\mathbf{h}}_i^{(0)} + \boldsymbol{\lambda}_i^{(0)} \quad (6.19)$$

注意 $\mathbf{c}^{(0)} \in \mathbb{R}^d$ 是一个偏置项对于拉普拉斯位置嵌入， $\mathbf{h}(0)$ 是全节点嵌入，拉普拉斯位置嵌入仅针对输入层计算，不在 Transformer 层内使用。

1. 没有边属性的图

有两种构造方法一个图 Transformer，取决于图是否有边属性。当没有边缘属性时， l 层多头注意力机制的输出为：

$$\tilde{\mathbf{h}}_i^{(l+1)} = [\mathbf{Attn}(1, l, i); \dots; \mathbf{Attn}(H, l, i)]\mathbf{O}^{(l)} \quad (6.20)$$

$$\mathbf{Attn}(m, l, i) = \sum_{j \in S_i} w_{ij}^{(m,l)} \mathbf{W}_v^{(m,l)} \mathbf{h}_j^{(l)} \quad (6.21)$$

$$w_{ij}^{(m,l)} = \text{softmax}_j \left(\frac{\mathbf{W}_q^{(m,l)} \mathbf{h}_i^{(l)} \cdot \mathbf{W}_k^{(m,l)} \mathbf{h}_j^{(l)}}{\sqrt{d_k}} \right) \quad (6.22)$$

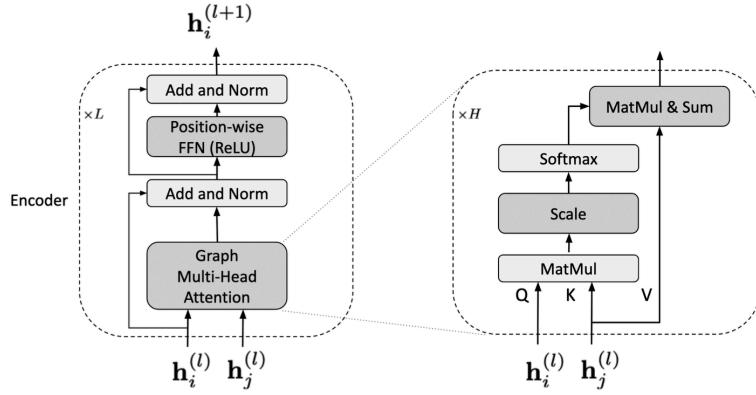


Figure 6.3 Diagram of the Graph Transformer encoder, without edge attributes. The encoder structure is shown on the left. The multi-head attention mechanism is shown on the right. As described in section 6.6.1.1, Laplacian embeddings are only applied to the input layer, $l = 0$.

图 6.3 Graph Transformer 编码器图, 不带边缘属性。编码器结构如左图所示。多头注意力机制如右图所示。如第 6.6.1.1 节所述, 拉普拉斯嵌入仅应用于输入层, $l= 0$ 。

其中 S_i 是查询 i 关注的关键位置集合, m 个索引注意头, $\mathbf{W}_q^{(m,l)}, \mathbf{W}_k^{(m,l)}, \mathbf{W}_v^{(m,l)} \in \mathbb{R}^{d_k \times d}$ 为键值维度, $\mathbf{O}^{(l)} \in \mathbb{R}^{d \times d}$ 。请注意, 为了数值稳定性, 上述 softmax 计算中的指数被限制为 $[-5,5]$ 。还请注意, 与节点嵌入相乘以创建查询、键和值矩阵的权重矩阵的定义是转置本书其他地方使用的定义。

$\tilde{\mathbf{h}}_i^{(l+1)}$ 如下传递到 transformer 层的其他部分, 导致 transformer 层的输出 $\mathbf{h}_i^{(l+1)}$:

$$\mathbf{h}_i^{(l+1)} = \text{Norm}(\mathbf{h}'_i^{(l+1)} + \mathbf{h}''_i^{(l+1)}) \quad (6.23)$$

$$\mathbf{h}'_i^{(l+1)} = \text{Norm}(\mathbf{h}_i^{(l)} + \tilde{\mathbf{h}}''_i^{(l+1)}) \quad (6.24)$$

$$\mathbf{h}''_i^{(l+1)} = \mathbf{W}_2 \text{ReLU}(\mathbf{W}_1^{(l)} \mathbf{h}'_i^{(l+1)}) \quad (6.25)$$

其中 $\mathbf{W}_1^l \in \mathbb{R}^{2d \times d}, \mathbf{W}_2^{(l)} \in \mathbb{R}^{d \times 2d}$ 和 Norm 可以是层归一化或批量归

一化。图中 (6.23) - (6.25) 中描述的图 transformer 的结构如图 6.3 所示。

2. 具有边属性的图 对于边上具有属性的图, (6.21)-(6.25) 被修改和扩展以考虑边嵌入。节点嵌入主要更新为道路。唯一的变化是, 进入 softmax 操作的注意力权重现在有以下来自边缘嵌入的贡献:

$$\tilde{\mathbf{h}}_i^{(l+1)} = [\mathbf{Attn}(1, l, i); \dots; \mathbf{Attn}(H, l, i)] \mathbf{O}^{(l)}$$

(6.26)

$$\tilde{\mathbf{e}}_{ij}^{(l+1)} = \left[\tilde{w}_{ij}^{(1,l)}, \dots, \tilde{w}_{ij}^{(H,l)} \right] \mathbf{O}_{edge}^{(l)} \quad (6.27)$$

$$\mathbf{Attn}(m, l, i) = \sum_{j \in S_i} w_{ij}^{(m,l)} \mathbf{V}^{(m,l)} \mathbf{h}_j^{(l)} \quad (6.28)$$

$$w_{ij}^{(m,l)} = \text{softmax}(\tilde{w}_{ij}^{(m,l)}) \quad (6.29)$$

$$\tilde{w}_{ij}^{(m,l)} = \left(\frac{\mathbf{W}_q^{(m,l)} \mathbf{h}_i^{(l)} \cdot \mathbf{W}_k^{(m,l)} \mathbf{h}_j^{(l)}}{\sqrt{d_k}} \right) \mathbf{W}_E^{(m,l)} \mathbf{e}_{ij}^{(l)} \quad (6.30)$$

其中, S_i 是查询 i 关注的关键位置集合, m 索引注意力头, $\mathbf{W}_q^{(m,l)}, \mathbf{W}_k^{(m,l)}, \mathbf{W}_v^{(m,l)} \in \mathbb{R}^{d_k \times d}$, d_k 为键值维度。 $\mathbf{W}_E^{(m,l)}$ 是边的值层的权重; 它转换边缘嵌入的方式类似于 $\mathbf{W}_v^{(m,l)}$ 转换节点嵌入的方式。

请注意, 虽然当存在边缘属性时存在单个多头注意力机制, 但存在一个单独的线性层用于组合节点的最终注意力和对边的最终注意力。这可以从 $\mathbf{O}^{(l)}, \mathbf{O}_{edge}^{(l)} \in \mathbb{R}^{d \times d}$ 的存在性看出。与没有边缘属性的情况一样, 为了数值稳定性, 上面的 softmax 计算中的指数被限制为 $[-5, 5]$ 。 $\tilde{\mathbf{h}}_i^{(l+1)}$ 被传递到 Transformer 层的其他剩余部分, 如下所示, 导致 Transformer 层的两个输出, $\mathbf{h}_i^{(l+1)}$ 对于节点 (如 (6.31) 所示) 和 $\mathbf{e}_{ij}^{(l+1)}$ 对于边:

$$\mathbf{h}_i^{(l+1)} = \text{Norm}(\mathbf{h}'_i^{(l+1)} + \mathbf{h}''_i^{(l+1)}) \quad (6.31)$$

$$\mathbf{h}'_i^{(l+1)} = \text{Norm}(\mathbf{h}_i^{(l)} + \tilde{\mathbf{h}}_i^{(l+1)}) \quad (6.32)$$

$$\mathbf{h}''_i^{(l+1)} = \mathbf{W}_{n,2}^{(l)} \text{ReLU}(\mathbf{W}_{n,1}^{(l)} \mathbf{h}'_i^{(l+1)}) \quad (6.33)$$

$$\mathbf{e}_i^{(l+1)} = \text{Norm}(\mathbf{e}'_i^{(l+1)} + \mathbf{e}''_{ij}^{(l+1)}) \quad (6.34)$$

$$\mathbf{e}'_i^{(l+1)} = \text{Norm}(\mathbf{e}_i^{(l)} + \tilde{\mathbf{e}}_{ij}^{(l+1)}) \quad (6.35)$$

$$\mathbf{e}''_i^{(l+1)} = \mathbf{W}_{e,2}^{(l)} \text{ReLU}(\mathbf{W}_{e,1}^{(l)} \mathbf{e}'_{ij}^{(l+1)}) \quad (6.36)$$

其中 $\mathbf{W}_{n,1}^{(l)}, \mathbf{W}_{e,1}^{(l)} \in \mathbb{R}^{2d \times d}$, Norm 可以是层归一化或批量归一化。下标 n 和 e 分别表示节点和边。图 6.4 示意性地显示了这一点。

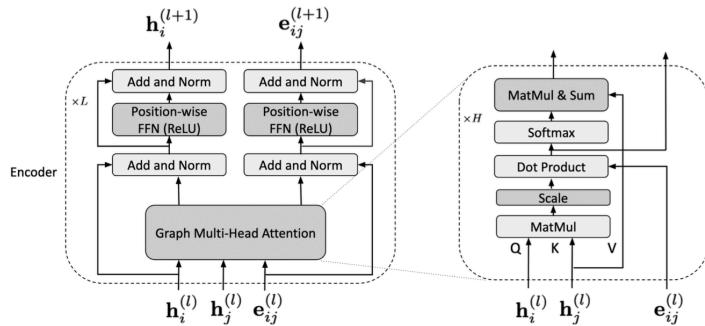


Figure 6.4 Diagram of the Graph Transformer encoder, with edge attributes. The encoder structure is shown on the left. The multi-head attention mechanism is shown on the right. As described in section 6.6.1.1, Laplacian embeddings are only applied to the input layer, $l = 0$.

图 6.4 Graph Transformer 编码器图，具有边缘属性。编码器结构如左图所示。多头注意力机制如右图所示。如第 6.6.1.1 节所述，拉普拉斯嵌入仅应用于输入层， $l = 0$ 。

6.7 强化学习

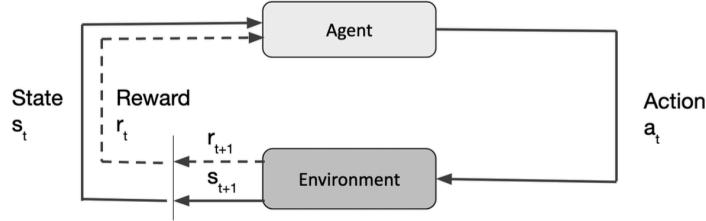


Figure 6.5 Depiction of the interaction between the agent and environment in reinforcement learning. At time t , the state observed by the agent is s_t . The agent then selects an action a_t . In the next time step, $t + 1$, the environment updates its state to s_{t+1} and issues a reward to the agent, r_{t+1} . After this, the cycle can repeat.

图 6.5 描述强化学习中代理与环境之间的交互。在时间 t 时，智能体观察到的状态为 s_t 。然后代理选择一个动作。在下一个时间步 $t+1$ 中，环境将其状态更新为 s_{t+1} 并向代理发出奖励 r_{t+1} 。此后，循环可以重复。

强化学习 (RL) 是一种机器学习方法，使用奖励来训练一个自主代理，该代理可以选择其下一步行动，以便在给定其当前状态的情况下获得最佳累积奖励。要理解强化学习的过程，我们可以开始具有有限马尔可夫决策过程 [208]，这是一种正式的方式与环境交互的代理可以采取的一系列决策。代理和环境按照一系列时间步骤 $t = 0, \dots, T$ 进行交互。在每次，代理获取环境的 state $s_t \in \mathcal{S}$ ，然后选择一个动作 $a_t \in \mathcal{A}(s)$ 。在时间 $t + 1$ ，智能体从环境 $r_{t+1} \in \mathbb{R}$ 中获得奖励，并且环境的状态更新为 s_{t+1} 。这个简单的想法如图 6.5 所示。请注意，一般来说，奖励将是状态和动作的函数， $r_t = R(s_t, a_t)$ 。随着时间的推移，一系列状态、动作和奖励不断累积： $(s_0, a_0, r_1, s_1, a_1, r_2, s_2, a_2, r_3, \dots)$ 。这个序列可以称为轨迹。

在实践中，我们观察具有概率分布的随机变量 s_t 和 r_t 。转移到状态 s' 并获得奖励 r 的概率，假设在状态 s 中采取的行动 a 由

$$p(s', r | s, a) = \text{Prob}(s_t = s', r_t = r | s_{t-1} = s, a_{t-1} = a) \sum_{s' \in \mathcal{S}} \sum_{r \in \mathcal{R}} p(s', r | s, a) = 1, \forall s \in \mathcal{S}, a \in \mathcal{A}(s) \quad (6.37)$$

转移概率充分表征了环境的动态性并且智能体不知道奖励集合或转移概率。强化学习的任务是学习智能体在状态 s 中选择动作的概率。这种概率分布称为策略。

6.7.1 决策转换器

决策转换器 [209] 是尝试用一个变换器取代强化学习 (RL) 方法的结果，该变换器对状态、动作和奖励的序列进行建模，用于训练一个自主代理。这个应用程序与大多数转换器所应用的语言建模任务相去甚远。

上面概述的有限马尔可夫决策过程与决策转换器使用的过程之间有一个重要的区别。决策转换器应用于一种称为离线强化学习的强化学习类型。Inonline RL 不是让代理与环境交互并在采取行动并获得奖励后进行状态更新，而是有一个固定的数据集，其中包含从任意策略得出的轨迹。对于智能体来说，这是一种更难学习的方法。对于离线强化学习，数据集中的轨迹的形式为

$$(\hat{R}_1, s_1, a_1, \hat{R}_2, s_2, a_2, \dots, \hat{R}_T, s_T, a_T) \quad (6.38)$$

$$(R^{1,s1,a1}, R^{2,s2,a2}, \dots, R^{T,sT,aT}) \quad (6.38)$$

其中 $\hat{R}_t = \sum_{t'=t}^T r_{t'}$ 是“return-to-go”，即从 t 到达轨迹终点所需生成的奖励量。

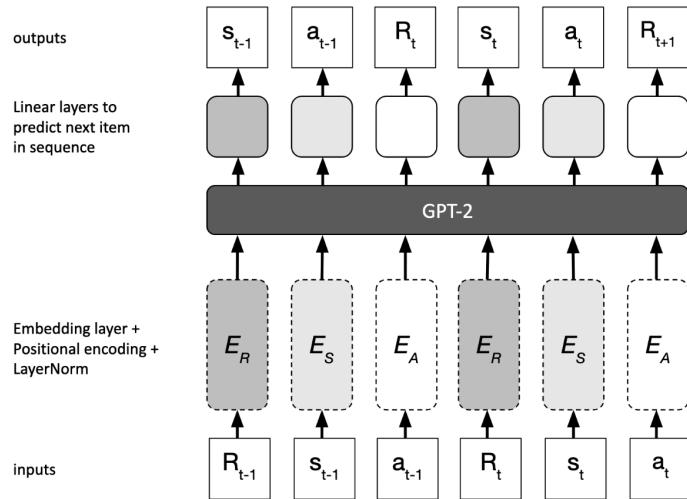


Figure 6.6 Decision Transformer architecture for offline RL. The first two time steps are shown. E_R , E_S , and E_A represent the combined embedding layer positional encoding, and layer norm for the returns-to-go, states, and actions, respectively. The layer after GPT-2 contains three linear submodules for predicting the next element of the sequence. Next state, action, and return-to-go are predicted using different submodules.

图 6.6 用于离线 RL 的决策 Transformer 架构。显示了前两次步骤。 E_R 、 E_S 和 E_A 分别表示组合嵌入层位置编码和返回、状态和动作的层范数。GPT-2 之后的层包含三个线性子模块，用于预测序列的下一个元素。使用不同的子模块来预测下一个状态、动作和返回。

这些固定轨迹用于自回归训练。前面的 K 个步骤被传递到决策转换器。每个步骤包含三个“标记”：状态、操作和返回。每种类型的 token 都有自己的线性嵌入层，后面跟着一个 Layer-Norm。每种类型的标记嵌入中还添加了学习的位置嵌入，这些标记嵌入与标准转换器中的不同，因为每个时间步长有三个标记而不是一个。然后将嵌入传递给 GPT-2 模型，进行自回归训练。请注意，对于图像输入，嵌入层是计算机视觉中常用的卷积编码器。Decision Transformer 架构的示意图如图 6.6 所示。

由于输入 token 是奖励、状态和动作的序列，因此多头自注意力机制计算奖励、状态和动作之间的注意力。这包括输入标记和部分生成的输出

标记之间的交叉注意力，其中仅仅是行动。Chen 等人。推测注意力机制使决策转换器能够学习状态、动作和奖励之间的关系。切内塔尔。我还相信 DecisionTrans-former 对于在线 RL 方法可能很有用，它可用于生成多样的行为。DecisionTransformer 实现了最先进的各种 RL 基准。

6.8 案例研究：自动语音识别

6.8.1 目标

在本章中，我们研究了 transformer 架构。在本案例研究中，我们比较了 Wav2vec 2.0 [154] 与 Speech2Text2 [199] 和 HuBERT [200] 的转录质量，它们都是基于 Transformer 的，并且已在本章前面讨论过。我们将使用单词来测量转录质量错误率和字符错误率指标。

6.8.2 数据、工具和库

```
pip install -U transformers datasets librosa jiwer
torchaudio sentencepiece
```

清单 6.1 Python 环境设置

我们将使用 Huggingface Transformers 加载预训练模型，使用 Huggingface Datasets 加载自动语音识别（ASR）数据集，TIMIT [210]。在环境中加载必要的库后，如清单 6.1 所示，我们可以加载数据集

```
from datasets import load_dataset
timit = load_dataset("timit_asr", split="test")
```

清单 6.2 加载数据集

6.8.3 实验、结果和分析

1. 预处理语音数据

```

import librosa

def process_data(batch):
    batch["speech"], batch["sampling_rate"] =
        librosa.load(batch["file"], sr=16000)
    return batch

timit = timit.map(process_data,
remove_columns=['file', 'audio', 'phonetic_detail', 'word_detail',
'dialect_region', 'sentence_type', 'speaker_id', 'id'])

```

清单 6.3 预处理演讲

2. 评估

有我们在转录过程中可能会犯三种类型的错误：

- 替换：将正确的单词或字符替换为错误的单词或字符
 - 删除：删除正确的单词或字符
 - 插入：插入单词或字符
- 我们可以通过以下方式计算单词错误率和字符错误率指标：将三种错误相加并除以单词（或字符）数：

$$WER = \frac{S + D + I}{N} \quad (6.39)$$

$$CER = \frac{S + D + I}{N} \quad (6.40)$$

其中 S 是替换数， D 是删除数， I 是插入的数量， N 是转录本中单词（或字符，对于 CER）的数量。Huggingface 数据集库内置了两个指标，因此我们将使用它们而不是编写自己的指标。

```
from datasets import load_metric
```

```
wer = load_metric("wer")
cer = load_metric("cer")
```

清单 6.4 加载 WER 和 CER 指标现在我们加载每个指标模型，然后转录并计算质量指标。**Wav2vec 2.0** 运行清单 6.5 之前的列表后，我们发现 Wav2vec 2.0 在 TIMIT 测试集上的 WER 为 0.09291 (9.29%)，CER 为 0.02212 (2.2%) .

```
import torch
from transformers import Wav2Vec2Processor, Wav2Vec2ForCTC

device = torch.device("cuda" if torch.cuda.
    is_available() else "cpu")
processor = Wav2Vec2Processor.from_pretrained
    ("facebook/wav2vec2-large-960h")
model = Wav2Vec2ForCTC.from_pretrained
    ("facebook/wav2vec2-large-960h") .to(device)

def remove_punc(text):
    text = text.replace('。', ' ')
    text = text.replace('，', ' ')
    text = text.replace('？', ' ')
    text = text.replace('；', ' ')
    text = text.replace('！', ' ')
    return text

def wav2vec2_predict(batch):
    features = processor( batch["speech"] ,
        sampling_rate=batch["sampling_rate"][0] ,
        padding=True, return_tensors="pt")
```

```

        input_values = features["input_values"].to(device)
        with torch.no_grad():
            logits = model(input_values).logits

        predicted_ids = torch.argmax(logits, dim=-1)
        transcription = processor.batch_decode(predicted_ids)
        batch["transcription"] = transcription
        # Wav2vec 2's base model doesn't produce
        # punctuation and uppercases text
        batch["target"] = [remove_punc(x.upper()) for x in
                           batch["text"]]
    return batch

BATCH_SIZE = 16
result = timit.map(wav2vec2_predict, batched=True,
                    batch_size=BATCH_SIZE,
                    remove_columns=["speech", "sampling_rate"])
print("WER: ", wer.compute(predictions=result["transcription"],
                           references=result["target"]))
print("CER: ", cer.compute(predictions=result["transcription"],
                           references=result["target"]))

```

清单 6.5 使用 Wav2vec 进行转录和评估

Speech2Text2 [199] 是一个转换器解码器模型，可以与任何语音编码器一起使用，例如 Wav2Vec2 或 HuBERT [200]。我们将使用与清单 6.5 中使用的 Wav2vec2 模型大小相当的 Speech2Text2 模型。

```

from transformers import Speech2TextProcessor,
Speech2TextForConditionalGeneration

```

```

s2t_processor = Speech2TextProcessor.
    from_pretrained("facebook/s2t-large-librispeech-asr")
s2t_model = Speech2TextForConditionalGeneration.
    from_pretrained
        ("facebook/s2t-large-librispeech-asr").to(device)
def remove_punc(text):
    text = text.lower()
    text = text.replace(' . , , ')
    text = text.replace(' , , , ')
    text = text.replace(' ? , , ')
    text = text.replace(' ; , , ')
    text = text.replace(' ! , , ')
    return text

def s2t_predict(batch):
    features = s2t_processor(batch["speech"],
        sampling_rate=batch["sampling_rate"][0],
        padding=True, return_tensors="pt")
    input_features = features["input_features"].to(device)
    # including the attention mask is important for this model
    # if it is omitted, then the model may generate
    # transcription that is noticeably longer than the target
    attention_mask = features["attention_mask"].to(device)
    with torch.no_grad():
        generated_ids = s2t_model.generate(input_ids=input_features,
            attention_mask=attention_mask)

    batch["transcription"] = s2t_processor.
        batch_decode(generated_ids,

```

```

    skip_special_tokens=True)

    # Speech2Text2 model doesn't produce punctuation and lowercases text
    batch["target"] = [remove_punc(x) for x in batch["text"]]
    return batch

BATCH_SIZE = 16
result = timit.map(s2t_predict, batched=True,
                    batch_size=BATCH_SIZE,
                    remove_columns=["speech", "sampling_rate"])
print("WER: ", wer.compute(predictions=result["transcription"],
                           references=result["target"]))
print("CER: ", cer.compute(predictions=result["transcription"],
                           references=result["target"]))

```

清单 6.6 使用 Speech2Text2 进行转录和评估

在清单 6.6 中，您会注意到我们显式地将注意力掩码传递给模型。事实证明，这对于 Speech2Text2 很重要。如果省略注意掩码，则模型可能会生成比目标长的转录。如果 GPU 的速度与 K80 相当，则列表运行时间应约为 4 分钟。Speech2Text2 模型在 TIMIT 测试集上实现了 0.10283 (10.28%) 的 WER 和 0.03624 (3.62%) 的 CER。

HuBERT [200]，Hidden-Unit BERT 的缩写，是一种基于 Transformer 的自监督语音表示学习模型能够直接从标记的音频数据中学习声学模型和语言模型。

```

from transformers import Wav2Vec2Processor, HubertForCTC

hb_processor = Wav2Vec2Processor(
    from_pretrained("facebook/hubert-large-ls960-ft"))

```

```

hb_model = HubertForCTC.from_pretrained
("facebook/hubert-large-ls960-ft").to(device)
def remove_punc(text):
    text = text.upper()
    text = text.replace(' .', ' ')
    text = text.replace(' ,', ' ')
    text = text.replace(' ?', ' ')
    text = text.replace(' ;', ' ')
    text = text.replace(' !', ' ')
    return text
def hb_predict(batch):
    features = hb_processor( batch["speech"],
        sampling_rate=batch["sampling_rate"][0],
        padding=True, return_tensors="pt")
    input_values = features["input_values"].to(device)
    with torch.no_grad():
        logits = model(input_values).logits

    predicted_ids = torch.argmax(logits, dim=-1)
    transcription = processor.batch_decode(predicted_ids)
    batch["transcription"] = transcription
    # HuBERT doesn't produce punctuation and uppercases text
    batch["target"] = [remove_punc(x) for x in batch["text"]]
    return batch

BATCH_SIZE = 16
result = timit.map(hb_predict, batched=True,
    batch_size=BATCH_SIZE,
    remove_columns=["speech","sampling_rate"])
print("WER: ", wer.compute(predictions=result["transcription"]),

```

```

    references=result["target"]))
print("CER: ", cer.compute(predictions=result["transcription"],
    references=result["target"]))

```

清单 6.7 使用 HuBERT 进行转录和评估

表 6.1 Wav2vec2.0、Speech2Text2 和 HuBERT 的 TIMIT 测试集的 WER 和 CER 指标。最佳得分为黑体字

Model	WER	CER
Wav2vec 2.0	0.09291	0.02212
Speech2Text2	0.10283	0.03624
HuBERT	0.11544	0.02872

HuBERT 在 TIMIT 测试集。为了进行比较，将 WER 和 CER 分数汇总到表 6.1 中，其中显示 Wav2vec 2.0 在 TIMIT 测试中表现最好，Speech2Text2 位居第二；胡伯特排名第三。

7 Transformer 的可解释性和可解释性技术

在医疗保健、立法、执法或金融等领域的非关键应用中，除了预测之外，还需要从可解释性的角度来理解模型。不幸的是，人们可以将本书中介紹的大多数最先进的 transformer 模型和技术归类为“黑匣子”，这会严重阻碍采用。因此，迫切需要从理解和诊断的角度围绕这些复杂的最先进模型建立可解释性。正如 Xie 等人提出的，我们将涵盖解决可解释性的模型的特征、影响可解释性的相关领域、应用于基于 transformer 和基于注意力的系统的可解释方法的分类，最后是一个详细的案例使用具有不同可解释技术的 transformer 来研究电子健康记录系统以获得实际见解 [211] .

7.1 可解释系统的特征

可解释系统的决定性目标之一是它允许系统的最终用户理解输入之间的关系和输出。因此，特征可以定义为系统的属性，用户可以评估和测量的

深入研究。正如 Xie 等人提出的，四个必要的特质是：

1. **自信**当最终用户（决策者）能够根据输入和输出将基于 transformer 的模型的处理与他们的思维过程结合起来时，对系统的信心就会增加。图像或文本上的注意力显着图突出显示了从模型角度来看对决策（分类、识别、问答等）很重要的输入部分，输出模仿了受过训练的人类如何将基于焦点的机制作为一种解释形式 [212, 213, 44, 214, 215] .
2. **安全**当部署在直接或间接影响人类生活的应用中时，基于 transformer 的模型应该被认为是安全的。人们可以根据以下方面来评价安全性：
(i) 一致且确定性的行为，即，给定相同的输入，输出每次都保持相同，
(ii) 稳健且可靠的理解和异常条件，以及 (iii) 能够警惕对整个社会产生负面影响的选择 [216] 。
3. **信任**可靠的模型是不需要验证的模型。事实证明，预测置信度高的模型并不能保证可信度 [217, 216, 218, 219]]] 。谢特等人。考虑两个重要的可信度标准：(i) 满意度测试和 (ii) 经验。令人满意的测试是模型获得与其训练相似的性能的能力。从数量上来说，当模型性能指标（例如准确度或精度）在训练和未见过的测试数据上相似时，信任就会增加。经验是一种更加定性的评估，人们可以在不检查的情况下判断系统足以执行任务。现实世界部署中的数据和概念漂移可能会进一步降低对系统的信任。近年来有许多研究从模型预测的角度探讨安全性 [220, 221, 222, 223] 。
4. **伦理**如果模型不违反最终用户设定的任何道德原则，则可以在其决策过程中被认为是道德的。为基于道德的人工智能制定框架和指南本身作为一个领域正在不断发展 [224, 225, 226, 227] 。

7.2 影响可解释性的相关领域

Xie 等人确定与可解释性密切相关并与上一节中定义的特征相关的四个领域。

学习机制：提供对模型训练过程的见解的能力有助于增加对模型的信心和信任。可以通过以下方式进一步探索导致模型学习状态的过程：从过滤器、

权重和激活的角度理解学习概念或统计模式中不同层的演化 [228, 229]。基于输入和权重的不同层的收敛性研究，以及从记忆和统计力学角度研究网络的泛化特性 [230]。

模型调试：类似于软件调试，模型调试对应于对模型架构的检查，通过模型进行数据处理。网络，以及训练和运行过程中引入的错误 [231]。一种常见的研究方法是建立充当诊断和检查探针的辅助模型 [232]。Amershi 等人提出的 ModelTracker 允许视觉交互，包括错误标记数据、缺失特征识别、对标签学习训练数据不足的洞察、异常值的影响、特征空间可视化、性能模型摘要等，这是一种与模型无关的调试方法 [233]。Fuchs 等人的神经听诊器。是另一个通用框架，它通过促进和抑制信息来量化影响因素的重要性来分析学习过程 [234]。

对抗性攻击和防御：对抗性示例是经过精心设计的输入，用于输入模型并判断其区分能力。构建对抗性示例需要了解输入空间和类之间的边界。分类问题。对抗性攻击和防御是从可解释性和诊断角度探索模型的两种不同方法 [219, 235]。黑盒和白盒攻击是对抗性攻击的两种主要类型，它们生成示例来欺骗模型 [236, 237, 238]。最近的研究表明，通过扰动输入添加难以察觉的噪声揭示了模型的漏洞 [239]。对抗性防御是为了使模型变得稳健 190 用于机器学习的 Transformers：针对对抗性示例的深入研究。对抗性防御的两种常见方法是 (i) 对抗性训练，其中训练数据集通过对抗性示例进行增强以引入鲁棒性，以及 (ii) 扰动去除，其中模型识别对抗性示例并拒绝它们 [217, 240]。

公平性和偏见：在影响人类的关键领域部署的许多模型的主要目标之一是在决策过程中“公平”和无偏见。尽管是一个不断发展的领域，但公平意味着 (i) 群体公平，也称为人口平等或统计平等，重点关注种族、性别、性取向等。[241]，以及 (ii) 个体公平性，重点关注具有相似特征和生成相似模型输出的特征的个体 [242]。解决公平性的不同技术可以分为 (i) 预处理方法，从数据中删除敏感特征，(ii) 过程中方法，其中添加公平性约束，以及 (iii) 后处理方法，用于在训练后调整模型预测 [243, 244, 245, 246]。** 可解释的方法分类有许多关于可解释人工智能的调查，其中不同的策略生成了许多用于对可解释性技术进行分类的分类法。谢等人。分类学基于基本的可

解释性方法，仅限于通用或特定的基于 transformer 的模型，在本节中讨论，如图 7.1.

7.2.1 可视化方法

可视化方法通过强调输入和输出之间的影响来揭示解释。黑盒模型的输出如图 7.2 所示。可视化和解释的常见方法是通过显着性图，突出显示最大程度地影响模型行为的最显着的输入。谢等人。进一步将可视化方法分为基于反向传播和基于扰动的可视化。

1. 基于反向传播

使用训练期间从输出传递到输入的梯度信号来理解输入数据的显着性是本组中的典型过程。

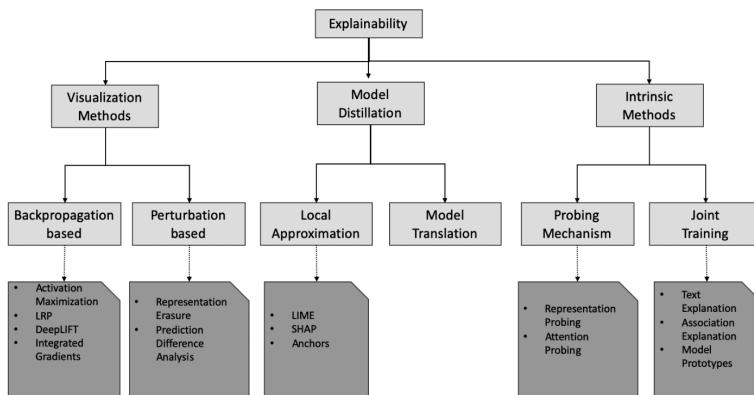


Figure 7.1 Explainable methods taxonomy.

图 7.1 可解释方法分类。

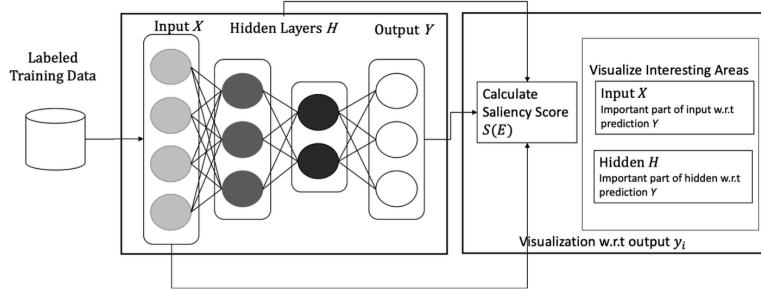


Figure 7.2 Visualization methods. Vary based on different saliency techniques $S(E)$ applied to inputs and hidden layers with respect to the output.

图 7.2 可视化方法。根据应用于输入和相对于输出的隐藏层的不同显着性技术 $S(E)$ 而变化。**Activation maximization**：在他们的开创性工作中，Erhan 等人引入了一种通用方法来可视化网络任何层的重要特征，方法是注意到激活 a_j^i （对应于单元 i 和层 j 的激活）可以通过保持参数 θ 固定和优化输入 \mathbf{x} [247]。一般原理是通过计算 $a_j^i(\mathbf{x}, \theta)$ 的梯度并更新输入 \mathbf{x} 的梯度方向来优化输入 \mathbf{x} 。

逐层相关性传播 (LRP)：基于 LRP 的技术测量每个输入特征与网络的输出 [248, 249, 250, 251]。LRP 的最通用类型是深度泰勒分解，其基础是假设网络 f 是可微的，并且可以通过某个根 $\hat{\mathbf{x}}$ 的泰勒展开来近似，其中 $f(\hat{\mathbf{x}}) = 0$ 。根的选择点是这样的，即它与输入 \mathbf{x} 没有显着差异， f 为其生成不同的预测 [252]。

$$f(\mathbf{x}) = f(\hat{\mathbf{x}}) + \Delta_{\hat{\mathbf{x}}} f \cdot (\mathbf{x} - \hat{\mathbf{x}}) + \epsilon \quad (7.1)$$

$$f(\mathbf{x}) = \sum_i^N \frac{\partial f}{\partial x_i}(\hat{x}_i) \cdot (x_i - \hat{x}_i) + \epsilon \quad (7.2)$$

其中 ϵ 对泰勒展开式中的所有二阶及更高项求和。输入的相关性得分可以从上面的方程得出：

$$r_i = \frac{\partial f}{\partial x_i}(\hat{x}_i) \cdot (x_i - \hat{x}_i) \quad (7.3)$$

对于具有多层的深度网络，深度泰勒分解位置假设相关性得分的分解，从输出开始经过中间层到输入。假设连接到层 $l+1$ 处的 M 个节点的节点 i 层的相关性分数是可分解的，并由下式给出：

$$r_i^l = \sum_j^M r_{ij}^l \quad (7.4)$$

深度学习重要特征 (DeepLIFT)：Shrikumar 等人提出的深度学习重要特征其中根据输入 \mathbf{x} 和“参考”输入 \mathbf{x}' 之间的差异将相关性分数分配给输入特征，其中 \mathbf{x} 是特定于问题的 [253]。参考输入 \mathbf{x}' 的选择是特定于域的。例如，在 MNIST 分类中，全零表示共同背景的输入可以是参考输入的选择之一。 $\Delta t = f(\mathbf{x}) - f(\mathbf{x}')$ 是输入 \mathbf{x} 和参考输入 \mathbf{x}' 之间神经元输出的差异。为输入特征 x_i 分配的相关性得分 $R_{\Delta x_i \Delta t}$ 和计算所需的神经元由下式给出：

$$\Delta t = \sum_{i=1}^N R_{\Delta x_i \Delta t} \quad (7.5)$$

Shrikumar 等人。使用 Linear 规则、Rescale 规则和 Reveal Cancel 规则提供不同的方法来计算 Δx_i 对 Δt 的影响之间的权重。定义乘数 $m_{\Delta x \Delta t}$ ，测量 $\Delta \mathbf{x}$ 相对于 Δt 的相关性，通过 $\Delta \mathbf{x}$ 求平均值为：

$$m_{\Delta x \Delta t} = \frac{R_{\Delta x \Delta t}}{\Delta \mathbf{x}} \quad (7.6)$$

采用链式法则逐层计算相关性得分在 DeepLIFT 论文中。Arkhangelskaia 和 Dutta 对 BERT 模型预测应用 DeepLIFT 并测试结果，以监控问答应用程序输入的注意力值的变化 [254]。

积分梯度 (IG) Sundarajan 等人的积分梯度。是一种基于两个公理计算网络相关性的解释技术：敏感性和实现方差 [255]。敏感性公理：对

于与某个基线输入 \mathbf{x}' 不同的输入 \mathbf{x} ，以及特征 x_i 且 $f(\mathbf{x}) \neq f(\mathbf{x}')$ ，则 \mathbf{x}_i 应具有非零相关性。实现不变性：对于两个网络 f_1 和 f_2 ，当所有可能的输入的输出相等时，每个输入特征 x_i 的相关性得分在 f_1 和 f_2 上应该相同。

如果是具有输入 \mathbf{x} 和基线输入 \mathbf{x}' 的深度网络，其中输出为 [0,1]，输入 \mathbf{x} 的特征 x_i 的相关性被视为通过 \mathbf{x}' 的路径的梯度积分，由下式给出：

$$IG_i(\mathbf{x}) = (\mathbf{x}_i - \mathbf{x}'_i) \int_0^1 \frac{\partial f(\mathbf{x}' + \alpha(\mathbf{x}) - \mathbf{x}')}{\partial \mathbf{x}_i} \quad (7.7)$$

其中 α 平滑地分布在 [0,1] 范围内，并且与 \mathbf{x}' 到 \mathbf{x} 的路径无关。上述方程可以使用论文中描述的适当步数的黎曼求和来近似。

2. 基于扰动

输入并通过比较原始输入和改变输入之间的输出差异来计算特征相关性是所有方法的核心基于扰动的方法。

表示擦除：Li 等人提出了一种通用技术来进行解释，通过检查擦除输入表示的影响来了解这些变化如何影响输出 [256]。通过分析影响，我们可以识别对模型决策有重大贡献的基本表征。擦除可以在不同级别的表示上执行，例如输入词向量维度、输入词或词组以及中间隐藏单元。直接的技术是计算擦除表示时标签上对数似然的差异。更复杂的方法使用强化学习模型来找到必须删除的最小单词集，以改变模型的决策。

有意义的扰动：Fong 和 Vedaldi 提出了一种元预测器，使用扰动技术根据敏感信息给出局部解释。输入中预测输出的区域 [257]。作者提出了三类扰动来生成图像分类的视觉解释：(i) 常数，用恒定值替换图像中的区域，(ii) 噪声，向该区域添加小噪声，以及 (iii) 模糊，模糊图像例如，预测狗的黑盒图像分类器 (f) 的解释为

$$D(\mathbf{x}, f) = \{\mathbf{x} \in \mathbf{X}_d \iff f(\mathbf{x}) = 1\} \quad (4)$$

其中 $f(\mathbf{x}) = 1$ 图像中狗的预测， \mathbf{X}_d 是分类器预测为狗的所有图像。对于新图像 \mathbf{x}^0 ，解释技术使用上述技术在输入中创建扰动，以根据解释规则找到结果为 $f(\mathbf{x}^0)$ 的局部敏感区域。

预测差异分析: Zintgraf 等人提出了一种基于 Robnik-Sikonja 和 Kononenko 研究的方法，用于测量基于改变基于概率的分类器的输入信息的影响 [258, 259]。它通过使用边际概率确定 $p(c|\mathbf{x}-i)$ 和 $p(c|\mathbf{x})$ 之间的差异来评估输入特征 x_i 对于类 c 的影响

$$p(c|\mathbf{x}_{-i}) = \sum_{x_i} p(x_i|\mathbf{x}_{-i})p(c|\mathbf{x}_{-i}, x_i) \quad (7.9)$$

其中 \mathbf{x} 对应于所有输入特征， \mathbf{x}_{-i} 对应于除 x_i 之外的所有特征。特征 x_i 的重要性通过以下方式测量：

$$\text{Diff}_i(c|\mathbf{x}) = \log_2(\text{odds}(c|\mathbf{x})) - \log_2(\text{odds}(c|\mathbf{x}_{-i})) \quad (7.10)$$

7.2.2 模型蒸馏

Xie 等人将模型蒸馏类别称为训练后方法，其中模型中的编码知识被蒸馏成便于用户解释的表示，如图 7.3 所示。Xie 等人进一步将这一类别分为两个子类别：局部逼近和模型翻译。

1. 局部逼近

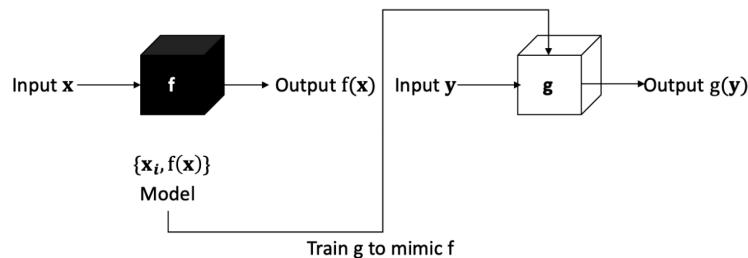


Figure 7.3 Model distillation.

图 7.3 模型蒸馏

所有局部逼近方法都假设学习流形中的局部边界比学习整个空间上的全局判别模型要简单得多 [260]。例如，一个简单的线性分类器可以区分流形中密集区域的数据子集，并为该区域中的实例提供简单的解释。我们将讨论一些可以在模型无关方式中使用的流行技术。**Local Interpretable Model-agnostic Explanations(LIME)**：Ribeiro 等人。开发了一种最流行的局部近似方法，称为局部可解释模型不可知解释 (LIME) [261]。LIME 通过扰动任何给定的感兴趣实例的流形邻近度来生成新的样本实例，测量每个样本的模型输出，并将局部线性可解释模型拟合到新样本。因此，通过覆盖全局模型中的大多数实例和预测，可以构建许多本地易于解释的模型，例如线性加权分类器/回归器或决策树。让我们通过函数 f 、感兴趣的数据实例 x 、从一类可解释模型 \mathcal{H} （例如线性、逻辑或决策树）中选择的解释模型 h 来表示整个深度学习模型，那么识别局部可解释模型的问题就相当于针对目标 $\zeta(x)$ 进行优化：

$$\zeta(x) = \arg \min_{h \in H} \mathcal{L}(f, h, \pi_x(z)) + \Omega(h) \quad (7.11)$$

其中 π_x 是一个核函数，根据样本 z 到感兴趣的实例 x 的距离对其进行加权， \mathcal{L} 是一个损失函数， Ω 是复杂性惩罚，例如决策树中的树深度。损失函数 \mathcal{L} 提供了解释模型在实例 x 的局部性上与真实模型的接近程度的度量。

SHapley 加法解释 (SHAP)：Shapley 值具有博弈论基础，特别是合作博弈论。人们可以将每个功能视为游戏中的玩家，目标是根据玩家对总支出的贡献，在组成联盟的玩家之间公平分配支出。每个特征在所有可能的特征联盟集中平均的边际贡献给出了 Shapley 值。

让我们通过函数 f 表示整个深度学习模型，输入 d 个特征 $x = x_1, x_2, \dots, x_d$ ，即 Shapley 值第 j 个特征的 ϕ_j 由下式给出：

$$\phi_j(f) = \frac{|S|!(d - |S| - 1)!}{d!}[f(S \cup \{x_j\}) - f(S)] \quad (7.12)$$

锚点: 锚点克服了许多使用线性近似来定义局部行为的局部解释方法的局限性。对于许多现实世界的领域，线性近似可能会导致不准确的决策。锚点是一种高精度的解释技术，用于通过推断特征空间中的一组 if-then 规则来生成局部解释 [262]。这些规则充当一组特征的“锚点”，使其对变化和未见数据更加鲁棒。

如果 x 表示解释的实例，则表示一组谓词，即结果规则或锚点，使得 $A(x) = 1$ ， f 表示黑盒模型， D 表示已知的扰动分布， z 是规则 A 应用 $D(z|A)$ 时从条件分布中抽取的样本， τ 是给定的精度阈值，则锚定定义为：

$$\mathbb{E}_{D(z|A)} = [\mathbb{1}_{f(x)=f(z)}] \quad (7.13)$$

一般来说，在连续空间中找到锚点是不可行的，通过引入参数 δ ，可以改变定义：对于某些任意小的 δ ，满足此精度阈值的概率如下：

$$\Pr[\rho(A) \geq \tau] \geq 1 - \delta \quad (7.14)$$

由于可能有多个锚点满足此标准，因此优先考虑具有最高覆盖范围的锚点 $cov(A)$ ，定义为

$$cov(A) = \mathbb{E}_{D(z)}[A(z)] \quad (7.15)$$

如果 A' 是满足 (7.14) 的锚点集合，则锚点生成则成为一个组合优化问题，由下式给出：

$$\max_{A \in A'} cov(A) \quad (7.16)$$

在实践中，使用了各种基于启发式的方法。例如，贪婪搜索技术（例如自下而上的方法）从一个空规则集开始，通过一个附加特征谓词迭代扩展锚来生成一组候选规则，以贪婪的方式创建此规则集。集束搜索可用于通过在迭代搜索过程中维护一组候选规则来改进这种贪婪搜索方法。

2. 模型翻译 模型翻译涉及在整个数据集上训练一个替代的“较小”模型作为输入，并将来自黑盒模型的输出作为训练数据。因此，与上述局部逼近技术相比，这以可解释的方式捕获了全局行为。替代模型的选择可以基于模型大小、部署难易程度和可解释的算法（例如决策树、基于规则的分类器、图形等）[263, 264, 265, 266]。谭等人。建议通过将黑盒模型分解为可解释模型（例如样条线或袋装树）然后使用它们构建可加性模型来构建通用可加性可解释模型，这是一种此类技术 [267]。

7.2.3 内在方法

内在方法使用模型架构为是，并且使用表示提供解释，例如隐藏层的权重或注意力权重 [268]。内在方法可以进一步分类为 (i) 基于探测机制和 (ii) 基于联合训练。

1. 探测机制

在过去的几年里，在注意力头或任何中间层之上插入一个简单层（例如，单层神经分类器）的想法深度学习模型已经在主要任务上进行了训练，以验证“信息”（辅助任务）作为诊断或解释的方式已经成为一种新兴趋势。例如，在针对情感分类进行训练的 BERT 模型上，可以在训练模型的中间层之上插入一个探针分类器，以衡量识别语言信息的能力。探测分类器可用于验证句法级别（例如动词、名词等）和语义级别（例如识别实体、关系、蕴涵等）的信息，这对于情感分类的总体主要任务很有用。创建一个数据集，其中包含用于情感分类的原始句子和识别动词、名词、实体等的标签。然后，用于测量任何层性能的探针分类器通过冻结这些层的信息来接受新辅助任务的训练和测试，从而在识别或解释中给出该层的定量测量 [269]。

Representation Probing 在最早的作品之一中，Hupkes 等人。提出了使用“诊断分类器”进行探测的概念，以揭示循环和递归网络如何使用具有精确语法和语义以及有限词汇量的算术语言任务来处理层次结构 [270]。此外，它还展示了神经架构如何处理具有分层组合语义的语言，并探索用于 NLP 任务的黑盒神经架构。

Conneau 等人引入探测任务来对不同任务上的不同神经架构进行全面评分，以理解和量化在编码器上学习的句子嵌入的语言属性 [96]。他们创建了探测任务，分为表面信息、句法信息和语义信息。表面信息探测任务回答问题；例如，“句子嵌入是否保留句子长度？”，使用以句子长度作为标签的数据集。句法信息探测任务研究基于句法的属性，例如，“嵌入对词序敏感吗？”，使用分类数据集，其中二元词移位作为正数，非移位作为负数。最后，语义信息探测任务研究嵌入中保留的基于语义的属性，例如，“嵌入可以理解时态吗？”，使用时态分类数据集，其中 VBP/VBZ 形式被标记为现在时，VBD 被标记为过去时。这项工作中使用不同架构和下游任务进行的综合实验为模型架构及其保留不同语言属性的能力提供了深入的见解。

Tenney 等人。引入了“边缘探测”来理解 ELMO、GPT 和 BERT 等深度学习架构中的隐藏表示 [271]。它通过冻结各层并使用神经分类器来训练和测试各种任务（例如词性），研究单词在每个位置的作用，以编码结构、句法、语义甚至远程现象标签（POS）、成分标签、依赖标签、命名实体标签、语义角色标签（SRL）、共指、语义原角色和关系分类。他们表明，语境化嵌入比非语境化等价物有所改进，与语义任务相比，主要是在句法任务上。

Tenney 等人在他们的工作中进一步发现，像 BERT 这样的模型可以以可解释和可本地化的方式重新发现类似于传统 NLP 管道的语言信息 [80]。他们发现了以下序列：POS 标记、解析、NER、语义角色、共指是整个 BERT 模型的一部分。他们引入了两个补充指标：标量混合权重和累积评分。标量混合权重是一种依赖于训练语料库的度量，突出显示与整个 BERT 模型中的探测分类器最相关的层。累积评分是一种依赖于评估集的指标，用于估计通过引入每一层在探测任务中可以获得多高的分数。

Hewitt 和 Liang 在他们的工作中设计了一种控制任务策略作为解决探测混杂因素的措施问题 [272]。探测混杂问题可以定义为——给定需要解释的主要神经架构、探测分类器（例如 MLP 或逻辑回归）和监督辅助任务，我们如何将性能（例如测试准确性）的信用分配给这三

个任务中的一个他们设计的控制任务可以通过探测诊断分类器轻松学习，但不会编码在表示（实际的神经模型或层）中。使用不同的探测分类器选择来评估控制任务和辅助任务之间的性能（测试准确性）差异作为选择性度量，可以轻松地将学习的功劳分配给表示或探测分类器。这项工作还回答了诸如“探针设计如何影响探测任务性能？”等问题。注意力探测通过在现有神经架构之上添加注意力层或使用深度学习层中的现有注意力权重，将其作为“注意力图”映射到输入来探索关系，从而进行探测两者之间的关系很快就会发展成为一种有效的解释技术。

Rocktäschel 等人。提出了序列到序列网络中的神经逐词注意机制，用于对单词和短语对的蕴涵进行推理 [273]。前提和假设之间逐字注意力的可视化表明，前提的不相关部分，例如没有什么意义的单词，被正确地忽略了蕴涵。通过更深层次语义连接的前提和假设通过注意力权重显示出适当的相关性。

Xu 等人使用注意力机制进行自动图像字幕任务 [274]。这项工作表明，注意力机制不仅取得了最先进的结果，而且突出了图像中的显着对象，同时在输出序列中生成相应的单词，因此对于解释很有用。

Transformers 201Yang 等人的可解释性和可解释性技术。采用分层注意力机制进行文档分类 [275]。两个级别的注意力机制（一个在单词级别，另一个在句子级别）使模型在构建文档表示时能够不同地关注内容。注意力层的可视化表明该模型选择信息丰富的单词和句子进行分类。

Jesse Vig 引入了一个工具“BertViz”，用于可视化 Transformer 中各个级别的注意力，即。在整个模型级别、注意力头级别和单个神经元级别 [176]。模型视图为给定输入提供跨所有层和头的整个模型的单一高级视图。模型视图有助于可视化所有层的注意力头中注意力模式如何演变。注意力头视图可视化给定层中一个或多个注意力头产生的注意力模式，可以帮助检测性别偏见等偏见。神经元视图可视化查询和关键向量中的各个神经元以及它们产生注意力的交互，从而详细了解模式是如何形成的。

2. 联合培训

在基于联合训练的方法中，解释任务与实际学习任务相关联以便模型共同学习解释和实际任务。解释任务可以以自然语言格式生成解释，或将输入与人类可理解的概念相关联，或学习可用作解释的数据原型。

通过使用文本解释生成增强深度学习架构，在预测时创建用户（专家或外行）友好的解释组件并随着原始任务学习解释是一种不断发展的可解释性方法 [276, 277, 278, 279, 280]。大多数方法需要解释作为训练数据的一部分以及主要任务的标记数据，这使得这种方法成本高昂。

将输入特征（单词或图像组件）或潜在特征与概念或对象相关联作为解释方法，或者通过将显着图与输入相关联或使用正则化项或模型更改的语义概念的潜在激活是此类中的另一种方法 [281, 282, 283, 284]。许多基于计算机视觉的体系结构，尤其是分类问题，从训练过程中学习原型，然后将其用于与新的未见过的实例关联作为解释 [285, 286]。

7.3 注意和解释

如上一节所述，新兴模式之一，特别是在 NLP 中，是将注意力权重的大小与输入相关联，并用它来解释模型行为。接下来，我们讨论一些影响人们如何看待注意力机制及其对可解释性贡献的论文和研究。

7.3.1 注意力不是解释

在本文中，Jain 和 Wallace 试图提出关于注意力及其解释的基本问题 [287]。例如，当我们创建如图 7.4 所示的将注意力权重直接与输入标记或权重相关联的注意力图时，没有考虑许多转换或计算（例如中间隐藏状态、查询向量、注意力技术）的影响。该论文提出了两个关键问题——(i) 注意力热图是否揭示了单词/标记的重要性？(ii) 注意力机制是否为我们提供了透明度，即模型是如何达到预测的？这些抽象问题可以映射到以下假设进行验证：

1. 注意力权重应与具有相关语义的特征重要性度量相关（例如，基于梯度的度量和留一法）

2. 如果我们关注不同的输入，预测会有所不同吗？

不同的 NLP 任务，例如文本分类、问答 (QA) 和自然语言推理 (NLI)，使用上下文 BiLSTM 和具有标准的非上下文前馈网络使用注意力机制进行分析。

1. 注意力权重和特征的重要性

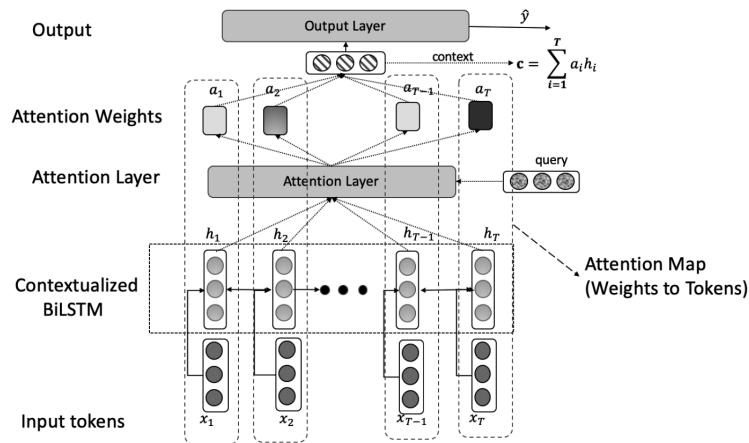


Figure 7.4 Attention probing with contextualized bidirectional LSTM as the neural model and an attention layer.

图 7.4 使用上下文双向 LSTM 作为神经模型和注意力层进行注意力探测。

对于第一个假设，输入中的单词根据注意力权重和特征重要性得分（梯度和留一法）进行排名，Kendal tau 度量用于测量相关性。与特征重要性分数相关的语义，实验通过扰动具有相同注意力和上下文的单个单词来测量输出变化。

该算法遵循一个简单的过程，首先对输入进行编码以获得隐藏状态 ($h \leftarrow Enc(\mathbf{x})$)，计算注意力 ($\hat{a} \leftarrow \text{softmax}(\phi(h, \mathbf{Q}))$) 和输出 ($\hat{y} \leftarrow Dec(h, \mathbf{Q})$)。对于基于梯度的方法，梯度是针对每个输入的梯度分数和相关分数：

$$g_t \leftarrow \left| \sum_{w=1}^{|V|} 1[\mathbf{x}_{tw} = 1] \frac{\partial y}{\partial \mathbf{x}_{tw}} \right|, \forall t \in [1, T] \quad (7.17)$$

$$\tau_g \leftarrow \text{Kendall}\tau(\alpha, g) \quad (7.18)$$

对于留一法，一次删除一个单词，并使用总变化距离（TVD）作为计算输出分布和相关得分的变化的度量：

$$\Delta \hat{y}_t \leftarrow \text{TVD}(\hat{y}(\mathbf{x}_{-t}), \hat{y}(\mathbf{x})), \forall t \in [1, t] \quad (7.19)$$

实验表明，所有数据集中的注意力权重和特征重要性得分之间的相关性始终较低，特别是对于上下文编码器。

2. 反事实实验

为了验证第二个假设，作者提出两个实证问题

- (a) 如果注意力分数随机排列，输出会改变多少？
- 2. 我们能否找到最大程度不同的注意力，并且不会使输出改变超过预定义的阈值 ϵ ？

注意力排列算法 2 捕捉了表征注意力权重变化时模型行为的实验

```

h  $\leftarrow$  Enc(x),  $\hat{\alpha}$   $\leftarrow$  softmax( $\phi(\mathbf{h}, \mathbf{Q})$ )
 $\hat{y}$   $\leftarrow$  Dec(h,  $\hat{\alpha}$ )
for  $p \leftarrow 1$  to 100 do
     $\alpha^p \leftarrow$  Permute( $\hat{\alpha}$ )
     $\hat{y}^p \leftarrow$  Dec(h,  $\hat{\alpha}$ )
     $\Delta \hat{y}^p \leftarrow$  TVD[ $\hat{y}^p$ ,  $\hat{y}$ ]
end
 $\Delta \hat{y}^{med} \leftarrow$  Median $p$ ( $\Delta \hat{y}^p$ )
Algorithm 1: Permuting attention weights.
```

算法 1：置换注意力权重。

对于不同的数据集，该实验的结果是不同的。一些数据集强调，注意力权重可能会通过一小组特征来推断解释输出，但扰乱注意力对预测几乎没有影响，而对于其他数据集，则没有观察到这一点。

对抗性注意力这个实验的直觉是探索新的注意力权重，该权重与观察到的注意力分布尽可能不同，但仍能给出准确的预测。指定 ϵ 值是为了简化定义预测中微小变化的优化过程。一旦给定 ϵ 值，目标是找到 k 个对抗分布 $\{\alpha^{(1)}, \dots, \alpha^{(k)}\}$ ，使得每个 $\alpha^{(i)}$ 最大化与原始 $\hat{\alpha}$ 的距离，但不会改变输出超过 ϵ 。The Jensen-Shannon 散度用于衡量分布之间的差异。优化方程由下式给出：

$$\text{maximize}_{\alpha^{(1)}, \dots, \alpha^{(k)}} f \left(\left\{ \alpha^{(i)} \right\}_{i=1}^k \right) \quad (7.20)$$

其中 $f(\{\alpha^{(i)}\}_{i=1}^k)$ 是：

$$\sum_{i=1}^k \text{JSD}[\alpha^{(i)}, \hat{\alpha}] + \frac{1}{k(k-1)} \sum_{i < j}^k \text{JSD}[\alpha^{(i)}, \alpha^{(j)}] \quad (7.21)$$

等式第一部分 $\sum_{i=1}^k \text{JSD}[\alpha^{(i)}, \hat{\alpha}]$ 发现与观察到的 $\hat{\alpha}$ 最大不同的注意力，并且第二部分 $\frac{1}{k(k-1)} \sum_{i < j}^k \text{JSD}[\alpha^{(i)}, \alpha^{(j)}]$ 彼此最大不同。对抗性注意力发现算法可以概括为：

```

h  $\leftarrow$  Enc(x),  $\hat{\alpha} \leftarrow \text{softmax}(\phi(\mathbf{h}, \mathbf{Q}))$ 
 $\hat{y} \leftarrow \text{Dec}(\mathbf{h}, \hat{\alpha})$ 
 $\alpha^{(1)}, \dots, \alpha^{(k)} \leftarrow \text{Optimize7.20}$  for  $i \leftarrow 1$  to  $k$  do
     $\left| \begin{array}{l} \hat{y}^{(i)} \leftarrow \text{Dec}(\mathbf{h}, \hat{\alpha}^{(i)}) \\ \Delta \hat{y}^{(i)} \leftarrow \text{TVD}[\hat{y}, \hat{y}^{(i)}] \\ \Delta \alpha^{(i)} \leftarrow \text{JSD}[\hat{\alpha}, \alpha^{(i)}] \end{array} \right.$ 
end
 $\epsilon \max \text{JSD} \leftarrow \max_i 1[\Delta \hat{y}^{(i)} \leq \epsilon] \Delta \alpha^{(i)}$ 
Algorithm 2: Finding adversarial attention weights.

```

算法 2：寻找对抗性注意力权重。

与排列注意力权重实验类似，对抗性注意力实验的结果也根据数据而变化。注意力分布并不能唯一地描述为什么一个模型做出了特定的预测，因为你可以找到给出相同预测的替代注意力热图。从研究中得出的总体结论是——注意力并没有对模型做出特定预测的原因提供一致的解释。

7.3.2 注意力是不是解释

Wiegreff 和 Pinter 对 Jainand Wallace 在上述研究中提出的主张和假设提出质疑，并通过实验证明了注意机制对于可解释性的有用性 [288]。Wiegreffe 和 Pinter 对 Jain 和 Wallace 研究的主要论点是，他们的“解释”研究含糊不清，相关性实验不充分，对抗性权重实验模糊且不切实际。

AI 文献和 NLP 领域中的可解释性情境化显示了两种形式的解释——(i) 合理的解释和 (ii) 忠实的可解释性。合理解释的目标是为模型的行为提供人类可以理解的基本原理。合理的解释是主观的，需要人在回路中来判断解释是否合理并建立对模型的信任。忠实的解释能力是一种更严格的解释形式，并且与透明度和可解释性的期望属性以及作者在工作中探索的内容更紧密地结合在一起。该技术的目标是量化预测任务的输入（标记/单词）和输出（预测）之间的相关性。这种解释形式的必要条件之一是——模型解释是排他性的，即预测模型应该为一个推理过程表现出一组行为。作者为注意力机制的忠实解释提出了以下三个要求：

1. 注意机制应该是良好模型性能的必要组成部分。2. 注意力分布应该难以操纵，即，如果任何训练模型可以改变注意力权重的分布，但仍具有相似的预测，则它们可能不适合解释。这直接对应于忠实解释所需的排他性，并将指导对抗性模型的搜索。
1. 注意力分配应该在无情境的环境中发挥良好作用。由于注意力权重通常是在上下文化隐藏层输出上学习的，为了查看对输入标记的影响，需要使用非上下文化设置来判断其有用性。
1. 所有任务都需要集中注意力吗

作者使用使用来自 Jain 和 Wallace 设置的 BiLSTM 模型使用相同的三组任务和六个分类数据集，并创建另一个模型，其中注意力权重与

学习权重相比是均匀分布的。根据所有六个分类数据集上的 F1 分数比较统一的和学习的注意力权重，新闻数据集没有显示任何变化，因此不用于后续的两次分析。斯坦福情绪树库 (SST) 是一个边缘案例，与 MIMIC (III) 和 IMDB 数据集相比显示出很小的差异。

2. 搜索对抗模型

找到模仿基本模型预测的注意力权重分布，作者提出了一种模型一致的训练协议，用于通过适用于所有训练示例的组合参数化来查找对抗性注意力分布。他们用于对抗性训练的两个度量是总变异距离 (TVD) 和詹森-香农散度 (JSD)。总变异距离 (TVD) 用于比较两个模型的类别预测，由下式给出：

$$\text{TVD}(\hat{y}_1, \hat{y}_2) = \frac{1}{2} \sum_{i=1}^{|y|} |\hat{y}_{1i} - \hat{y}_{2i}| \quad (7.22)$$

Jensen-Shannon Divergence (JSD) 用于比较两个注意力分布，由下式给出：

$$JSD(\alpha_1, \alpha_2) = \frac{1}{2} \text{KL}[\alpha_1 \parallel \bar{\alpha}] + \frac{1}{2} \text{KL}[\alpha_2 \parallel \bar{\alpha}]$$

其中

$$\bar{\alpha} = \frac{\alpha_1 + \alpha_2}{2} \quad (7.23)$$

对抗训练算法首先训练一个基础模型 (Mb)。然后训练一个对抗模型 (Ma)，最小化基础模型的输出预测分数，并使用以下实例范围损失函数最大化基础模型的学习注意力分布的变化：

$$\mathcal{L}(M_a, M_b) = \text{TVD}(\hat{y}_1^{(i), \hat{y}_2^{(i)}}) - \lambda \text{KL}(\alpha_a^{(i)} \parallel \alpha_b^{(i)}) \quad (7.24)$$

其中 $\hat{y}^{(i)}$ 和 $\alpha(i)$ 表示实例 i 的预测和注意力分布，并且 λ 控制预测距离和注意力分布变化之间的权衡。

对于糖尿病数据集（以及贫血症和 IMDB），他们发现很难找到产生不同注意力权重且不损失预测性能的对抗性权重，从而支持将注意力用于忠实的解释。相反，斯坦福情感树库（SST）数据集显示没有使用注意力权重来进行忠实的解释。

3. 注意力探测 为了验证注意力分布在非上下文环境中是否运行良好，来自 BiLSTM 的注意力权重被施加到带有词向量表示包的非上下文训练 MLP 层上。因此，任务的高性能意味着注意力分数捕获了输入和输出之间的关系。除了斯坦福情感树库（SST）数据集外，每个任务和数据集都显示 BiLSTM 训练的注意力权重优于 MLP 和统一权重，表明注意力权重的有用性。

总之，研究已经奠定了验证注意力机制有用性的三个基本组成部分以及对其进行量化以进行忠实解释的三种方法。注意力机制的有用性被证明是依赖于任务的。

7.4 量化注意力流

如前两节所述，将注意力权重与具有单个注意力的简单 BiLSTM 中的解释相关联输出本身之前的层是一个开放的研究课题。在编码器中具有自注意力、多个注意力头和许多注意力层的 transformer 中，这个问题变得更加困难。在他们的研究中，Abnar 和 Zuidema 提出了几种技术来解决这个问题，即 Attention Rollout 和 Attention Flow，都计算从该层到输入到-kens（即标记注意力）的每一层的注意力分数，以进行可视化和解释 [289]

7.4.1 作为 DAG 的信息流

作为 DAG 任何层的 transformer 中的注意力模块都有一个残差 con-来自输入的连接，因此任意层 $l+1$ 处的值 (V) 由下式给出： $V_{l+1} = V_l + W_{att}V_l$ ，其中 W_{att} 是注意力矩阵。作者提出添加单位矩阵 I 并计算由残差连接更新的原始注意力为：

$$A = 0.5 \cdot W_{att} + 0.5 \cdot I \cdot A \quad (7.25)$$

该方程可用于为 Transformer 模型在任何节点或层创建信息流图。该研究建议使用有向无环图（DAG），其中输入标记或隐藏嵌入作为图节点，原始注意力权重作为边。

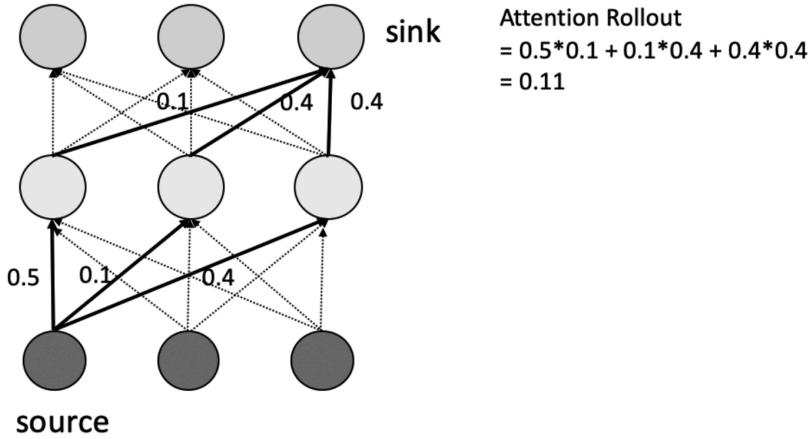


Figure 7.5 Example of attention rollout mechanism between two nodes.

图 7.5 两个节点之间的注意力推出机制示例。

7.4.2 注意力的推出

一旦从 Transformer 模型构建了 DAG，注意力推出机制就会通过计算比例来量化注意力对输入标记的影响可以通过每个链接递归地传播到所有输入标记的信息的数量。给定层 i 和前面各层中的所有注意力权重的注意力推出机制通过递归地乘以注意力权重矩阵来计算对输入标记的注意力，从输入层开始一直到第 i 层，可以写为：

$$\tilde{A}(l_i) = \begin{cases} A(l_i)\tilde{A}(l_{i-1}) & \text{if } i > j \\ A(l_i) & \text{if } i = j \end{cases} \quad (7.26)$$

7.5 显示了使用权重递归乘法在输入标记（源）和输出之间进行注意力展开计算。

7.4.3 注意力流

流网络是一个有向图，根据图论，其“容量”与图中的每个边相关。最大流量算法在给定流量网络的情况下找到任何给定源和接收器之间具有最大可能值的流量。在这种技术中，当图网络映射到 transformer 的流网络时，边作为注意力权重；最大流算法可以计算从任意层任意节点到输入节点（tokens）的最大注意力流。

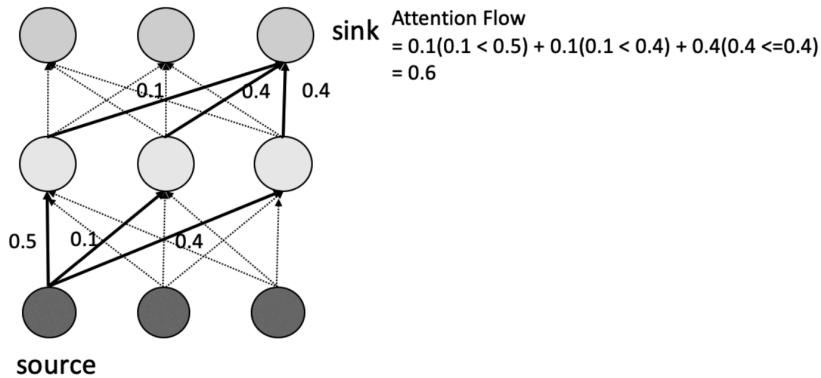


Figure 7.6 Example of attention flow mechanism between two nodes.

图 7.6 两个节点之间的注意力流机制示例。

7.6 显示了使用容量（最小权重）输入标记（源）和输出之间的注意力流计算。

研究表明，这两种方法给出了互补的观点，并且与吸引注意力相比，两者都与输入的重要性分数产生了更高的相关性使用基于梯度和消融方法的标记。

7.5 案例研究：文本分类

7.5.1 目标

在医疗保健领域，机器学习模型的透明度和可解释性对其采用至关重要。本节将通过一个在最先进的转换器上使用可解释性技术的用例来说明可解释性技术的有用性。目标是将逻辑回归等简单的传统可解释机器学习算法

与最先进的机器学习算法进行比较。最先进的 BERT 和临床 BERT 可以看到性能增益和事后技术的应用来解释黑盒模型。

7.5.2 数据、工具和库

由于 HIPAA 隐私法规，医疗保健数据的使用受到许多限制和限制。然而，MTSamples.com 收集转录的医疗报告以及四十种专业和工作类型的转录报告样本，以克服这些限制并帮助转录更容易获得。Kaggle 的医学转录数据集和分类任务基于该数据集。Medical-NLP 项目使用医学分类法进一步将数据转换为四个标签或专业（外科、医疗记录、内科和其他）。我们将使用转换后的数据集和分类任务进行案例研究。

我们使用 pandas 进行基本文本处理和探索性数据分析。Sklearn 库用于传统的 NLP 管道和逻辑回归模型。我们采用 BERT 的 Huggingface 实现和 BioClinicalBERT 作为我们的转换器实现。Captum 库用于通过显着性方法执行输入归因。为了理解和可视化 BERT 层和头部的工作方式，我们使用 exBERT 可视化包。

7.5.3 实验、结果和分析

1. 探索性数据分析

我们执行一些基本的 EDA 来理解来自分布和语料库的数据。

7.7 到 7.10 显示了词云、文档长度分布、热门词和类别分布的有趣输出。词云（图 7.7）和最高频率词图（图 7.9）清楚地显示出对诊断、伤害、慢性等术语的偏见，这些术语构成了大多数医疗转录和记录的基本语言。文档长度分布图（图 7.8）显示出长尾分布，超过 25% 的文档超出了 BERT 的最大序列长度。最后，40 个类别到 4 个类别的转换（图 7.10）显示了几乎平衡的分布，并且由于不平衡的影响现已最小化，因此在分类器比较中变得很有帮助。

2. 实验

我们使用 90-10% 的训练和测试分割，并进一步创建超参数和学习曲线的验证集为 10%。我们首先将基础 BERT 与 BIOClinicalBERT 进行

比较，两者均在训练数据上进行微调，并在测试数据上进行评估。



Figure 7.7 Word cloud.

图 7.7 词云。

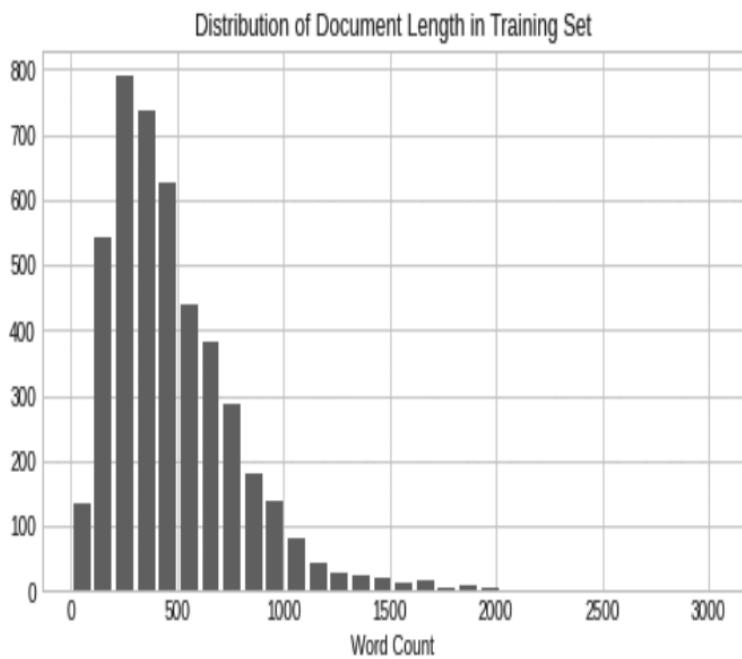


Figure 7.8 Document length distribution.

图 7.8 文档长度分布。

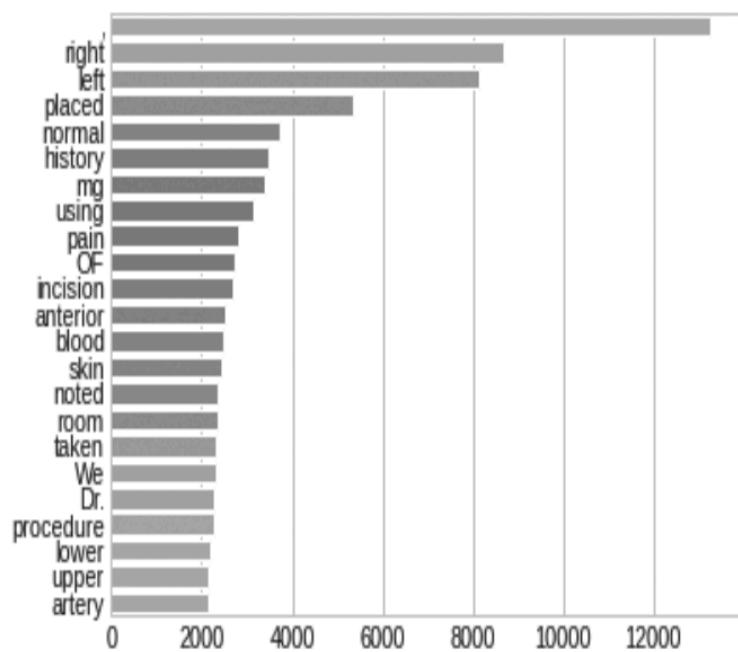


Figure 7.9 Top words.

图 7.9 热门词。

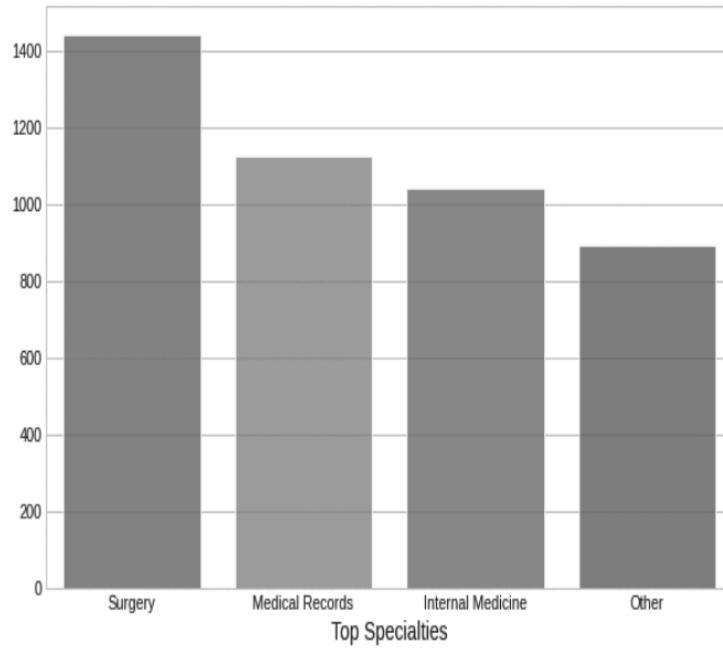


Figure 7.10 Class distribution.

图 7.10 类别分布。

表 7.1 和 7.2 显示,在大型医疗记录语料库上进行预训练的 BIO_{ClinicalBERT} 比基础 BERT 的分类稍有优势。

TABLE 7.1 Fine-tuned base BERT on Test Data

Class/Metrics	Precision	Recall	F1
Surgery	0.792	0.768	0.779
Medical Records	0.612	0.510	0.556
Internal Medicine	0.588	0.642	0.614
Other	0.587	0.670	0.626
accuracy			0.67
macro avg	0.645	0.648	0.644
weighted avg	0.673	0.670	0.670

TABLE 7.2 Fine-tuned Bio_Clinical BERT on Test Data

Class/Metrics	Precision	Recall	F1
Surgery	0.796	0.788	0.792
Medical Records	0.605	0.676	0.639
Internal Medicine	0.690	0.633	0.660
Other	0.589	0.582	0.586
<hr/>			
accuracy			0.694
macro avg	0.670	0.670	0.669
weighted avg	0.696	0.694	0.694

3. 错误分析和可解释性

接下来，我们将尝试进行错误分析使用可解释的技术来获得更多见解。

对 500 个测试数据的误差分析表明：1. BIO_{ClinicalBERT} 在 51 个实例上的表现优于基本模型。

- (a) BIO_{ClinicalBERT} 的表现比 39 个实例上的基础模型。
 - (b) 两个模型均未命中 114 个实例。
 - (c) 两个模型均在 296 个实例上正确预测。
-

```
def do_attribution(text, tokenizer,
    model, device, class_idx=0):
    inputs = tokenizer(text, return_tensors='pt',
        truncation=True, max_length=512)
    tokens = tokenizer.convert_ids_to_tokens(
        inputs['input_ids'].squeeze().tolist())
    logits = model(inputs['input_ids'].to(device)).logits
    probas = logits.softmax(-1).squeeze(0)
    model.zero_grad()
    attr_raw = sal.attribute(embed_text(text,
        tokenizer, model, device),
```

```

additional_forward_args=(model, class_idx), abs=False)
attr_sum = attr_raw.sum(-1).squeeze(0)
attr_norm = attr_sum / torch.norm(attr_sum)
record = viz.VisualizationDataRecord(attr_norm,
                                      pred_prob=probas.max(),
                                      pred_class=classes[probas.argmax().item()],
                                      true_class=classes[class_idx], # change if you
# aren't passing the true (labeled) class index
                                      attr_class=classes[class_idx],
                                      attr_score=attr_norm.sum(),
                                      raw_input=tokens,
                                      convergence_score=None)
return record

```

清单 7.1 显着性解释清单

7.1 显示了视觉的基于显着性的实现-输入属性的化。

在图中。在 7.11 和 7.12 中，我们可以看到同一记录上每个模型的解释有所不同——我们将每个标记归因于正确的外科手术类别。基本 BERT 模型对此示例给出了错误的预测。乍一看，我们可以看到，baseBERT 模型最显着的负面向因来自与肾脏相关的术语（kidney、renal 等），这可能表明 baseBERT 模型将这些术语与外科手术以外的类别相关联。域适应的 BioClinical BERT 对膀胱有积极和消极的归因，并对程序的描述有积极的解释（例如，“置于背侧截石位置”和“以标准方式准备和覆盖”）。这可以让我们深入了解额外的预训练如何帮助 BioClinical BERT 实现相对更好的性能。

从图 7.14 和 7.12 可以看出， $\text{BIO}_{\text{ClinicalBERT}}$ 正确预测了医疗记录，而基础 BERT 则正确预测了内科。在 $\text{BIO}_{\text{ClinicalBERT}}$ 显着性可视化中突出显示“既往病史”、“现在”、“入院”等上下文词，进一步强调了为什么“医疗记录”是 $\text{BIO}_{\text{ClinicalBERT}}$ 的预测类别。

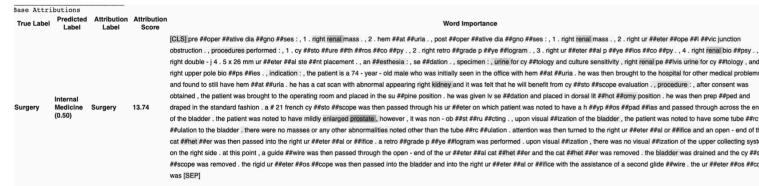


Figure 7.11 Saliency-based visualization for a correct prediction for the base BERT

图 7.11 基于显着性的可视化，用于正确预测基本 BERT。

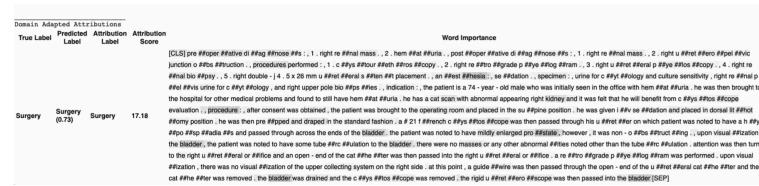


Figure 7.12 Saliency-based visualization for a correct prediction for the domain adapted BIO_ClinicalBERT

图 7.12 基于显着性的可视化，用于对适应领域的 BIOClinicalBERT 进行正确预测。

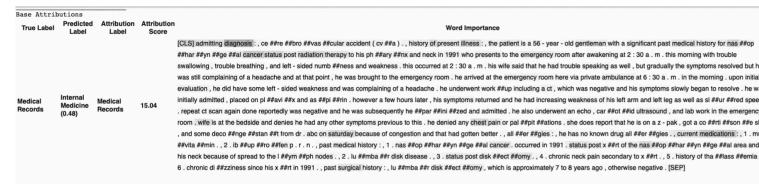


Figure 7.13 Saliency-based visualization for an incorrect prediction for the base BERT

图 7.13 基于显着性的可视化，用于对基本 BERT 进行错误预测。

`./Images/` 图 7.14 基于显着性的可视化，用于对适应 BIO_{ClinicalBERT} 的领域进行错误预测。

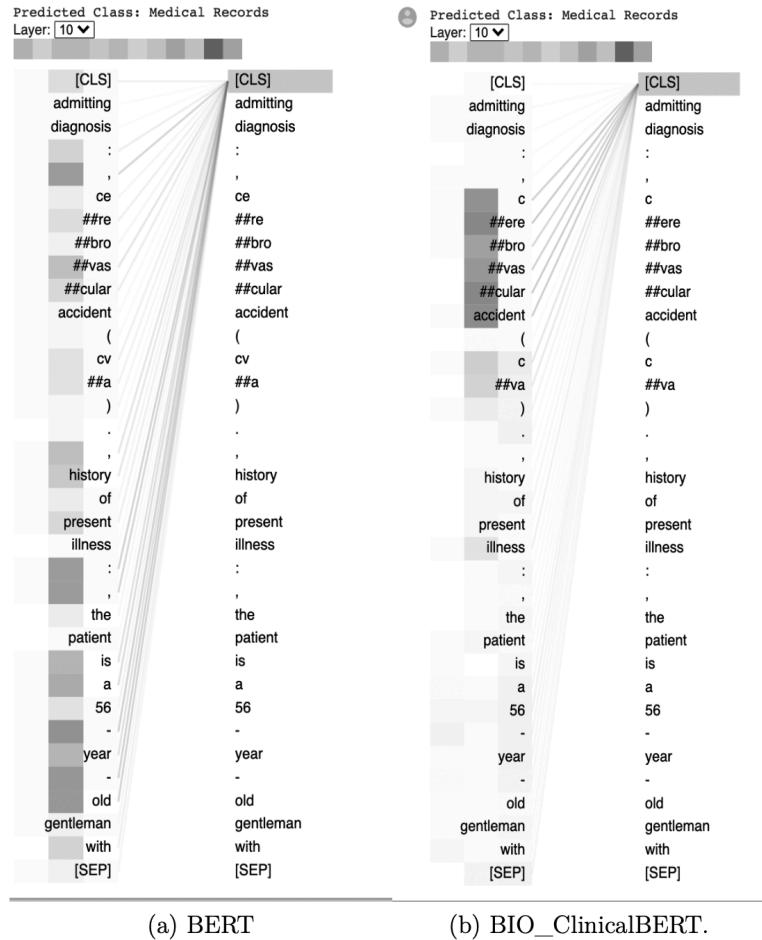


Figure 7.15 Sample example visualized at a layer by two different architectures:
(a) BERT and (b) BIO_ClinicalBERT.

图 7.15 将两个不同架构的可视化数据层示例:(a) BERT 和 (b) BIO_ClinicalBERT。

我们还可以可视化自注意力权重以获得另一个视图转换为 BERT 模型学习的表示。BertViz 库提供了一个交互式前端来探索这些标记权重。不幸的是, BertViz 不能很好地扩展到长序列, 因此我们截断了输入以说明 BERT 和 BIO_ClinicalBERT 的局部注意力效应, 以突出标记权重和类别标记之间的关系各层如图 7.15 所示。

参考文献

- [1] Warren S. McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The Bulletin of Mathematical Biophysics*, 5(4):115–133, December 1943.
- [2] D.O. Hebb. *The Organization of Behavior*. Psychology Press, 0 edition, April 2005.
- [3] Marvin Minsky and Seymour A. Papert. *Perceptrons: an introduction to computational geometry*. The MIT Press, Cambridge/Mass., 2. print. with corr edition, 1972.
- [4] J J Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences*, 79(8):2554–2558, April 1982.
- [5] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, October 1986.
- [6] Seppo Linnainmaa. *The representation of the cumulative rounding error of an algorithm as a Taylor expansion of the local rounding errors*. PhD thesis, Master’s Thesis (in Finnish), Univ. Helsinki, 1970.
- [7] Paul Werbos. Beyond regression: New tools for prediction and analysis in the behavioral sciences. *PhD thesis, Committee on Applied Mathematics, Harvard University, Cambridge, MA*, 1974.
- [8] Kunihiko Fukushima. Neural network model for a mechanism of pattern recognition unaffected by shift in position-neocognitron. *IEICE Technical Report, A*, 62(10):658–665, 1979.
- [9] David B Parker. Learning-logic. *Tech. Rep.*, 47, 1985.
- [10] Yann LeCun. Une procedure d’apprentissage pour reseau a seuil asymetrique. *Proceedings of Cognitiva 85*, pages 599–604, 1985.

- [11] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural Computation*, 1(4):541–551, December 1989.
- [12] Sepp Hochreiter. The vanishing gradient problem during learning recurrent neural nets and problem solutions. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 06(02):107–116, April 1998.
- [13] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, November 1997.
- [14] Geoffrey E. Hinton, Simon Osindero, and Yee-Whye Teh. A fast learning algorithm for deep belief nets. *Neural Computation*, 18(7):1527–1554, July 2006.
- [15] Yoshua Bengio, Pascal Lamblin, Dan Popovici, and Hugo Larochelle. Greedy layer-wise training of deep networks. In *Advances in Neural Information Processing Systems*, volume 19. MIT Press, 2006.
- [16] Yoshua Bengio and Yann LeCun. *Scaling Learning Algorithms toward AI*, page 321–360. The MIT Press, August 2007.
- [17] Zhiwei Wang, Yao Ma, Zitao Liu, and Jiliang Tang. R-transformer: Recurrent neural network enhanced transformer. (arXiv:1907.05572), July 2019. arXiv:1907.05572 [cs, eess].
- [18] Alex Graves. Generating sequences with recurrent neural networks. (arXiv:1308.0850), June 2014. arXiv:1308.0850 [cs].
- [19] Ronan Collobert and Jason Weston. A unified architecture for natural language processing: deep neural networks with multi-task learning. In *Proceedings of the 25th international conference on Machine learning - ICML '08*, page 160–167, Helsinki, Finland, 2008. ACM Press.

- [20] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Advances in Neural Information Processing Systems*, volume 26. Curran Associates, Inc., 2013.
- [21] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. (arXiv:1301.3781), September 2013. arXiv:1301.3781 [cs].
- [22] Ilya Sutskever. *Training recurrent neural networks*. University of Toronto Toronto, ON, Canada, 2013.
- [23] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. (arXiv:1409.0473), May 2016. arXiv:1409.0473 [cs, stat].
- [24] Katikapalli Subramanyam Kalyan, Ajit Rajasekharan, and Sivanesan Sangeetha. Ammus: A survey of transformer-based pretrained models in natural language processing. (arXiv:2108.05542), August 2021. arXiv:2108.05542 [cs].
- [25] Yi Tay, Mostafa Dehghani, Dara Bahri, and Donald Metzler. Efficient transformers: A survey. (arXiv:2009.06732), March 2022. arXiv:2009.06732 [cs].
- [26] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. (arXiv:1810.04805), May 2019. arXiv:1810.04805 [cs].
- [27] Tianyang Lin, Yuxin Wang, Xiangyang Liu, and Xipeng Qiu. A survey of transformers. (arXiv:2106.04554), June 2021. arXiv:2106.04554 [cs].
- [28] Zhanghao Wu, Zhijian Liu, Ji Lin, Yujun Lin, and Song Han. Lite transformer with long-short range attention. (arXiv:2004.11886), April 2020. arXiv:2004.11886 [cs].

- [29] Zihang Dai, Guokun Lai, Yiming Yang, and Quoc V. Le. Funnel-transformer: Filtering out sequential redundancy for efficient language processing. (arXiv:2006.03236), June 2020. arXiv:2006.03236 [cs, stat].
- [30] Sachin Mehta, Marjan Ghazvininejad, Srinivasan Iyer, Luke Zettlemoyer, and Hannaneh Hajishirzi. Delight: Deep and light-weight transformer. (arXiv:2008.00623), February 2021. arXiv:2008.00623 [cs].
- [31] Ruining He, Anirudh Ravula, Bhargav Kanagal, and Joshua Ainslie. Realformer: Transformer likes residual attention. (arXiv:2012.11747), September 2021. arXiv:2012.11747 [cs].
- [32] Ankur Bapna, Mia Xu Chen, Orhan Firat, Yuan Cao, and Yonghui Wu. Training deeper neural machine translation models with transparent attention. (arXiv:1808.07561), September 2018. arXiv:1808.07561 [cs].
- [33] Mostafa Dehghani, Stephan Gouws, Oriol Vinyals, Jakob Uszkoreit, and Łukasz Kaiser. Universal transformers. (arXiv:1807.03819), March 2019. arXiv:1807.03819 [cs, stat].
- [34] Ankur Bapna, Naveen Arivazhagan, and Orhan Firat. Controlling computation versus quality for neural sequence models. (arXiv:2002.07106), April 2020. arXiv:2002.07106 [cs, stat].
- [35] Ji Xin, Raphael Tang, Jaejun Lee, Yaoliang Yu, and Jimmy Lin. Deebert: Dynamic early exiting for accelerating bert inference. (arXiv:2004.12993), April 2020. arXiv:2004.12993 [cs].
- [36] Zihang Dai, Zhilin Yang, Yiming Yang, Jaime Carbonell, Quoc V. Le, and Ruslan Salakhutdinov. Transformer-xl: Attentive language models beyond a fixed-length context. (arXiv:1901.02860), June 2019. arXiv:1901.02860 [cs, stat].
- [37] Jack W. Rae, Anna Potapenko, Siddhant M. Jayakumar, and Timothy P. Lillicrap. Compressive transformers for long-

- range sequence modelling. (arXiv:1911.05507), November 2019. arXiv:1911.05507 [cs, stat].
- [38] Qingsong Wu, Zhenzhong Lan, Kun Qian, Jing Gu, Alborz Geramifard, and Zhou Yu. Memformer: A memory-augmented transformer for sequence modeling. (arXiv:2010.06891), April 2022. arXiv:2010.06891 [cs].
 - [39] Xingxing Zhang, Furu Wei, and Ming Zhou. Hibert: Document level pre-training of hierarchical bidirectional transformers for document summarization. (arXiv:1905.06566), May 2019. arXiv:1905.06566 [cs].
 - [40] Chuhan Wu, Fangzhao Wu, Tao Qi, and Yongfeng Huang. Hi-transformer: Hierarchical interactive transformer for efficient and effective long document modeling. (arXiv:2106.01040), December 2021. arXiv:2106.01040 [cs].
 - [41] Yiping Lu, Zhuohan Li, Di He, Zhiqing Sun, Bin Dong, Tao Qin, Liwei Wang, and Tie-Yan Liu. Understanding and improving transformer from a multi-particle dynamic system point of view. (arXiv:1906.02762), June 2019. arXiv:1906.02762 [cs, stat].
 - [42] Ofir Press, Noah A. Smith, and Omer Levy. Improving transformer models by reordering their sublayers. (arXiv:1911.03864), April 2020. arXiv:1911.03864 [cs].
 - [43] Yuekai Zhao, Li Dong, Yelong Shen, Zhihua Zhang, Furu Wei, and Weizhu Chen. Memory-efficient differentiable transformer architecture search. (arXiv:2105.14669), May 2021. arXiv:2105.14669 [cs].
 - [44] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. (arXiv:1706.03762), August 2023. arXiv:1706.03762 [cs].

- [45] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer normalization. (arXiv:1607.06450), July 2016. arXiv:1607.06450 [cs, stat].
- [46] Krzysztof Choromanski, Valerii Likhoshesterov, David Dohan, Xingyou Song, Andreea Gane, Tamas Sarlos, Peter Hawkins, Jared Davis, Afroz Mohiuddin, Lukasz Kaiser, David Belanger, Lucy Colwell, and Adrian Weller. Rethinking attention with performers. (arXiv:2009.14794), November 2022. arXiv:2009.14794 [cs, stat].
- [47] Yunyang Xiong, Zhanpeng Zeng, Rudrasis Chakraborty, Mingxing Tan, Glenn Fung, Yin Li, and Vikas Singh. Nyströmformer: A nyström-based algorithm for approximating self-attention. *Proceedings of the AAAI Conference on Artificial Intelligence*, 35(16):14138–14148, May 2021.
- [48] Yi Tay, Dara Bahri, Donald Metzler, Da-Cheng Juan, Zhe Zhao, and Che Zheng. Synthesizer: Rethinking self-attention in transformer models. (arXiv:2005.00743), May 2021. arXiv:2005.00743 [cs].
- [49] Maosheng Guo, Yu Zhang, and Ting Liu. Gaussian transformer: A lightweight approach for natural language inference. *Proceedings of the AAAI Conference on Artificial Intelligence*, 33(0101):6489–6496, July 2019.
- [50] Iz Beltagy, Matthew E. Peters, and Arman Cohan. Longformer: The long-document transformer. (arXiv:2004.05150), December 2020. arXiv:2004.05150 [cs].
- [51] Noam Shazeer, Zhenzhong Lan, Youlong Cheng, Nan Ding, and Le Hou. Talking-heads attention. (arXiv:2003.02436), March 2020. arXiv:2003.02436 [cs, eess, stat].

- [52] Sandeep Subramanian, Ronan Collobert, Marc’Aurelio Ranzato, and Y.-Lan Boureau. Multi-scale transformer language models. (arXiv:2005.00581), May 2020. arXiv:2005.00581 [cs].
- [53] Nikita Kitaev, Łukasz Kaiser, and Anselm Levskaya. Reformer: The efficient transformer. (arXiv:2001.04451), February 2020. arXiv:2001.04451 [cs, stat].
- [54] Manzil Zaheer, Guru Guruganesh, Avinava Dubey, Joshua Ainslie, Chris Alberti, Santiago Ontanon, Philip Pham, Anirudh Ravula, Qifan Wang, Li Yang, and Amr Ahmed. Big bird: Transformers for longer sequences. (arXiv:2007.14062), January 2021. arXiv:2007.14062 [cs, stat].
- [55] Aurko Roy, Mohammad Saffar, Ashish Vaswani, and David Grangier. Efficient content-based sparse attention with routing transformers. *Transactions of the Association for Computational Linguistics*, 9:53–68, February 2021.
- [56] Apoorv Vyas, Angelos Katharopoulos, and François Fleuret. Fast transformers with clustered attention. (arXiv:2007.04825), September 2020. arXiv:2007.04825 [cs, stat].
- [57] Haoyi Zhou, Shanghang Zhang, Jieqi Peng, Shuai Zhang, Jianxin Li, Hui Xiong, and Wancai Zhang. Informer: Beyond efficient transformer for long sequence time-series forecasting. *Proceedings of the AAAI Conference on Artificial Intelligence*, 35(12):11106–11115, May 2021.
- [58] Juho Lee, Yoonho Lee, Jungtaek Kim, Adam Kosiorek, Seungjin Choi, and Yee Whye Teh. Set transformer: A framework for attention-based permutation-invariant neural networks. In *Proceedings of the 36th International Conference on Machine Learning*, page 3744–3753. PMLR, May 2019.
- [59] Peter J. Liu, Mohammad Saleh, Etienne Pot, Ben Goodrich, Ryan Sepassi, Lukasz Kaiser, and Noam Shazeer. Generating

- wikipedia by summarizing long sequences. (arXiv:1801.10198), January 2018. arXiv:1801.10198 [cs].
- [60] Sinong Wang, Belinda Z. Li, Madian Khabsa, Han Fang, and Hao Ma. Linformer: Self-attention with linear complexity. (arXiv:2006.04768), June 2020. arXiv:2006.04768 [cs, stat].
 - [61] Jian Li, Zhaopeng Tu, Baosong Yang, Michael R. Lyu, and Tong Zhang. Multi-head attention with disagreement regularization. (arXiv:1810.10183), October 2018. arXiv:1810.10183 [cs].
 - [62] Jean-Baptiste Cordonnier, Andreas Loukas, and Martin Jaggi. Multi-head attention: Collaborate instead of concatenate. (arXiv:2006.16362), May 2021. arXiv:2006.16362 [cs, stat].
 - [63] Noam Shazeer. Fast transformer decoding: One write-head is all you need. (arXiv:1911.02150), November 2019. arXiv:1911.02150 [cs].
 - [64] Sainbayar Sukhbaatar, Edouard Grave, Piotr Bojanowski, and Armand Joulin. Adaptive attention span in transformers. (arXiv:1905.07799), August 2019. arXiv:1905.07799 [cs, stat].
 - [65] Qipeng Guo, Xipeng Qiu, Pengfei Liu, Xiangyang Xue, and Zheng Zhang. Multi-scale self-attention for text classification. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(0505):7847–7854, April 2020.
 - [66] Jianlin Su, Yu Lu, Shengfeng Pan, Ahmed Murtadha, Bo Wen, and Yunfeng Liu. Roformer: Enhanced transformer with rotary position embedding. (arXiv:2104.09864), November 2023. arXiv:2104.09864 [cs].
 - [67] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon

- Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. (arXiv:2005.14165), July 2020. arXiv:2005.14165 [cs].
- [68] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. (arXiv:1910.10683), September 2023. arXiv:1910.10683 [cs, stat].
- [69] Linting Xue, Aditya Barua, Noah Constant, Rami Al-Rfou, Sharan Narang, Mihir Kale, Adam Roberts, and Colin Raffel. Byt5: Towards a token-free future with pre-trained byte-to-byte models. (arXiv:2105.13626), March 2022. arXiv:2105.13626 [cs].
- [70] Paul Azunre. *Transfer Learning for Natural Language Processing*. Simon and Schuster, August 2021. Google-Books-ID: bGI7EAAAQBAJ.
- [71] Denis Rothman. Transformers for natural language processing : build innovative deep neural network architectures for nlp with python, pytorch, tensorflow, bert, roberta, and more. (*No Title*).
- [72] Joshua K. Cage. *Python Transformers By Huggingface Hands On: 101 practical implementation hands-on of ALBERT/ViT/BigBird and other latest models with huggingface transformers*.
- [73] Deep learning for NLP and speech recognition₂₀₁₉. SpringerScience + BusinessMedia, New York, NY, 2019.
- [74] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In *Advances in Neural*

Information Processing Systems, volume 27. Curran Associates, Inc., 2014.

- [75] Kyunghyun Cho, Bart van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. (arXiv:1406.1078), September 2014. arXiv:1406.1078 [cs, stat].
- [76] William James and Frederick H Burkhardt. The principles of psychology, the works of william james. 1983.
- [77] Minh-Thang Luong, Hieu Pham, and Christopher D. Manning. Effective approaches to attention-based neural machine translation. (arXiv:1508.04025), September 2015. arXiv:1508.04025 [cs].
- [78] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. page 770–778, 2016.
- [79] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. (arXiv:1502.03167), March 2015. arXiv:1502.03167 [cs].
- [80] Ian Tenney, Dipanjan Das, and Ellie Pavlick. Bert rediscovers the classical nlp pipeline. (arXiv:1905.05950), August 2019. arXiv:1905.05950 [cs].
- [81] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. (arXiv:1711.05101), January 2019. arXiv:1711.05101 [cs, math].
- [82] Stanley F Chen, Douglas Beeferman, and Roni Rosenfeld. Evaluation metrics for language models. 1998.
- [83] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah,

- Melvin Johnson, Xiaobing Liu, Łukasz Kaiser, Stephan Gouws, Yoshiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Greg Corrado, Macduff Hughes, and Jeffrey Dean. Google’s neural machine translation system: Bridging the gap between human and machine translation. (arXiv:1609.08144), October 2016. arXiv:1609.08144 [cs].
- [84] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized bert pretraining approach. (arXiv:1907.11692), July 2019. arXiv:1907.11692 [cs].
- [85] Pengcheng Yin, Graham Neubig, Wen-tau Yih, and Sebastian Riedel. Tabert: Pretraining for joint understanding of textual and tabular data. (arXiv:2005.08314), May 2020. arXiv:2005.08314 [cs].
- [86] Maarten Grootendorst. Bertopic: Leveraging bert and c-tf-idf to create easily interpretable topics. *Zenodo, Version v0*, 9(10.5281), 2020.
- [87] Venelin Valkov. Sentiment analysis with bert and transformers by hugging face using pytorch and python, 2020.
- [88] Sumanth Doddapaneni, Gowtham Ramesh, Mitesh M. Khapra, Anoop Kunchukuttan, and Pratyush Kumar. A primer on pre-trained multilingual language models. (arXiv:2107.00676), December 2021. arXiv:2107.00676 [cs].
- [89] Divyanshu Kakwani, Anoop Kunchukuttan, Satish Golla, Gokul N.C., Avik Bhattacharyya, Mitesh M. Khapra, and Pratyush Kumar. Indicnlpsuite: Monolingual corpora, evaluation benchmarks and pre-trained multilingual language models for indian

- languages. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, page 4948–4961, Online, 2020. Association for Computational Linguistics.
- [90] Junjie Hu, Melvin Johnson, Orhan Firat, Aditya Siddhant, and Graham Neubig. Explicit alignment objectives for multilingual bidirectional encoders. (arXiv:2010.07972), April 2021. arXiv:2010.07972 [cs].
 - [91] Simran Khanuja, Diksha Bansal, Sarvesh Mehtani, Savya Khosla, Atreyee Dey, Balaji Gopalan, Dilip Kumar Margam, Pooja Aggarwal, Rajiv Teja Nagipogu, Shachi Dave, Shruti Gupta, Subhash Chandra Bose Gali, Vish Subramanian, and Partha Talukdar. MuriL: Multilingual representations for indian languages. (arXiv:2103.10730), April 2021. arXiv:2103.10730 [cs].
 - [92] Fuli Luo, Wei Wang, Jiahao Liu, Yijia Liu, Bin Bi, Songfang Huang, Fei Huang, and Luo Si. Veco: Variable encoder-decoder pre-training for cross-lingual understanding and generation. October 2020.
 - [93] Haoyang Huang, Yaobo Liang, Nan Duan, Ming Gong, Linjun Shou, Dixin Jiang, and Ming Zhou. Unicoder: A universal language encoder by pre-training with multiple cross-lingual tasks. (arXiv:1909.00964), September 2019. arXiv:1909.00964 [cs].
 - [94] Alexis CONNEAU and Guillaume Lample. Cross-lingual language model pretraining. In *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.
 - [95] Zewen Chi, Li Dong, Furu Wei, Nan Yang, Saksham Singhal, Wenhui Wang, Xia Song, Xian-Ling Mao, Heyan Huang, and Ming Zhou. Infoxlm: An information-theoretic framework for cross-lingual language model pre-training. (arXiv:2007.07834), April 2021. arXiv:2007.07834 [cs].

- [96] Alexis Conneau, German Kruszewski, Guillaume Lample, Loïc Barrault, and Marco Baroni. What you can cram into a single vector: Probing sentence embeddings for linguistic properties. (arXiv:1805.01070), July 2018. arXiv:1805.01070 [cs].
- [97] Xiangpeng Wei, Rongxiang Weng, Yue Hu, Luxi Xing, Heng Yu, and Weihua Luo. On learning universal representations across languages. (arXiv:2007.15960), March 2021. arXiv:2007.15960 [cs].
- [98] Xuan Ouyang, Shuhuan Wang, Chao Pang, Yu Sun, Hao Tian, Hua Wu, and Haifeng Wang. Ernie-m: Enhanced multi-lingual representation by aligning cross-lingual semantics with monolingual corpora. (arXiv:2012.15674), September 2021. arXiv:2012.15674 [cs].
- [99] Hyung Won Chung, Thibault Févry, Henry Tsai, Melvin Johnson, and Sebastian Ruder. Rethinking embedding coupling in pre-trained language models. (arXiv:2010.12821), October 2020. arXiv:2010.12821 [cs].
- [100] Jason Phang, Iacer Calixto, Phu Mon Htut, Yada Pruksachatkun, Haokun Liu, Clara Vania, Katharina Kann, and Samuel R. Bowman. English intermediate-task training improves zero-shot cross-lingual transfer too. (arXiv:2005.13013), September 2020. arXiv:2005.13013 [cs].
- [101] Rico Sennrich, Barry Haddow, and Alexandra Birch. Neural machine translation of rare words with subword units. (arXiv:1508.07909), June 2016. arXiv:1508.07909 [cs].
- [102] Taku Kudo and John Richardson. Sentencepiece: A simple and language independent subword tokenizer and detokenizer for neural text processing. (arXiv:1808.06226), August 2018. arXiv:1808.06226 [cs].

- [103] Hyung Won Chung, Dan Garrette, Kiat Chuan Tan, and Jason Riesa. Improving multilingual models with language-clustered vocabularies. (arXiv:2010.12777), October 2020. arXiv:2010.12777 [cs].
- [104] Jian Yang, Shuming Ma, Dongdong Zhang, ShuangZhi Wu, Zhoujun Li, and Ming Zhou. Alternating language modeling for cross-lingual pre-training. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(05):9386–9393, April 2020.
- [105] Aaron van den Oord, Yazhe Li, and Oriol Vinyals. Representation learning with contrastive predictive coding. (arXiv:1807.03748), January 2019. arXiv:1807.03748 [cs, stat].
- [106] Rico Sennrich, Barry Haddow, and Alexandra Birch. Improving neural machine translation models with monolingual data. (arXiv:1511.06709), June 2016. arXiv:1511.06709 [cs].
- [107] Samuel R. Bowman, Luke Vilnis, Oriol Vinyals, Andrew M. Dai, Rafal Jozefowicz, and Samy Bengio. Generating sentences from a continuous space. (arXiv:1511.06349), May 2016. arXiv:1511.06349 [cs].
- [108] Wenzhao Zheng, Zhaodong Chen, Jiwen Lu, and Jie Zhou. Hardness-aware deep metric learning. page 72–81, 2019.
- [109] Fangxiaoyu Feng, Yinfei Yang, Daniel Cer, Naveen Arivazhagan, and Wei Wang. Language-agnostic bert sentence embedding. (arXiv:2007.01852), March 2022. arXiv:2007.01852 [cs].
- [110] Yinfei Yang, Gustavo Hernandez Abrego, Steve Yuan, Mandy Guo, Qinlan Shen, Daniel Cer, Yun-hsuan Sung, Brian Strope, and Ray Kurzweil. Improving multilingual sentence embedding using bi-directional dual encoder with additive margin softmax. (arXiv:1902.08564), June 2019. arXiv:1902.08564 [cs].
- [111] Yinfei Yang, Daniel Cer, Amin Ahmad, Mandy Guo, Jax Law, Noah Constant, Gustavo Hernandez Abrego, Steve Yuan,

- Chris Tar, Yun-Hsuan Sung, Brian Strope, and Ray Kurzweil. Multilingual universal sentence encoder for semantic retrieval. (arXiv:1907.04307), July 2019. arXiv:1907.04307 [cs].
- [112] Muthuraman Chidamaram, Yinfai Yang, Daniel Cer, Steve Yuan, Yun-Hsuan Sung, Brian Strope, and Ray Kurzweil. Learning cross-lingual sentence representations via a multi-task dual-encoder model. (arXiv:1810.12836), August 2019. arXiv:1810.12836 [cs].
- [113] Kaitao Song, Xu Tan, Tao Qin, Jianfeng Lu, and Tie-Yan Liu. Mass: Masked sequence to sequence pre-training for language generation. (arXiv:1905.02450), June 2019. arXiv:1905.02450 [cs].
- [114] Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Ves Stoyanov, and Luke Zettlemoyer. Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. (arXiv:1910.13461), October 2019. arXiv:1910.13461 [cs, stat].
- [115] Yinhan Liu, Jiatao Gu, Naman Goyal, Xian Li, Sergey Edunov, Marjan Ghazvininejad, Mike Lewis, and Luke Zettlemoyer. Multilingual denoising pre-training for neural machine translation. *Transactions of the Association for Computational Linguistics*, 8:726–742, November 2020.
- [116] Zewen Chi, Li Dong, Furu Wei, Wenhui Wang, Xian-Ling Mao, and Heyan Huang. Cross-lingual natural language generation via pre-training. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(0505):7570–7577, April 2020.
- [117] Linting Xue, Noah Constant, Adam Roberts, Mihir Kale, Rami Al-Rfou, Aditya Siddhant, Aditya Barua, and Colin Raffel. mt5: A massively multilingual pre-trained text-to-text transformer. (arXiv:2010.11934), March 2021. arXiv:2010.11934 [cs].

- [118] Mike Schuster and Kaisuke Nakajima. Japanese and korean voice search. In *2012 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, page 5149–5152, Kyoto, Japan, March 2012. IEEE.
- [119] Yaobo Liang, Nan Duan, Yeyun Gong, Ning Wu, Fenfei Guo, Weizhen Qi, Ming Gong, Linjun Shou, Dixin Jiang, Guihong Cao, Xiaodong Fan, Ruofei Zhang, Rahul Agrawal, Edward Cui, Sining Wei, Taroon Bharti, Ying Qiao, Jiun-Hung Chen, Winnie Wu, Shuguang Liu, Fan Yang, Daniel Campos, Rangan Majumder, and Ming Zhou. Xglue: A new benchmark dataset for cross-lingual pre-training, understanding and generation. (arXiv:2004.01401), May 2020. arXiv:2004.01401 [cs].
- [120] Junjie Hu, Sebastian Ruder, Aditya Siddhant, Graham Neubig, Orhan Firat, and Melvin Johnson. Xtreme: A massively multilingual multi-task benchmark for evaluating cross-lingual generalisation. In *Proceedings of the 37th International Conference on Machine Learning*, page 4411–4421. PMLR, November 2020.
- [121] Sebastian Ruder, Noah Constant, Jan Botha, Aditya Siddhant, Orhan Firat, Jinlan Fu, Pengfei Liu, Junjie Hu, Dan Garrette, Graham Neubig, and Melvin Johnson. Xtreme-r: Towards more challenging and nuanced multilingual evaluation. (arXiv:2104.07412), October 2021. arXiv:2104.07412 [cs].
- [122] Alexis Conneau, Guillaume Lample, Ruty Rinott, Adina Williams, Samuel R. Bowman, Holger Schwenk, and Veselin Stoyanov. Xnli: Evaluating cross-lingual sentence representations. (arXiv:1809.05053), September 2018. arXiv:1809.05053 [cs].
- [123] Yinfei Yang, Yuan Zhang, Chris Tar, and Jason Baldridge. Paws-x: A cross-lingual adversarial dataset for paraphrase identification. (arXiv:1908.11828), August 2019. arXiv:1908.11828 [cs].

- [124] Edoardo Maria Ponti, Goran Glavaš, Olga Majewska, Qianchu Liu, Ivan Vulić, and Anna Korhonen. Xcopa: A multilingual dataset for causal commonsense reasoning. (arXiv:2005.00333), October 2020. arXiv:2005.00333 [cs].
- [125] Tomasz Limisiewicz, Rudolf Rosa, and David Mareček. Universal dependencies according to bert: both more specific and more general. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, page 2710–2722, 2020. arXiv:2004.14620 [cs].
- [126] Erik F. Tjong Kim Sang. Introduction to the conll-2002 shared task: Language-independent named entity recognition. (arXiv:cs/0209010), September 2002. arXiv:cs/0209010.
- [127] Erik F. Tjong Kim Sang and Fien De Meulder. Introduction to the conll-2003 shared task: Language-independent named entity recognition. (arXiv:cs/0306050), June 2003. arXiv:cs/0306050.
- [128] Mikel Artetxe, Sebastian Ruder, and Dani Yogatama. On the cross-lingual transferability of monolingual representations. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, page 4623 – 4637, 2020. arXiv:1910.11856 [cs].
- [129] Patrick Lewis, Barlas Oğuz, Ruty Rinott, Sebastian Riedel, and Holger Schwenk. Mlqa: Evaluating cross-lingual extractive question answering. (arXiv:1910.07475), May 2020. arXiv:1910.07475 [cs].
- [130] Jonathan H. Clark, Eunsol Choi, Michael Collins, Dan Garrette, Tom Kwiatkowski, Vitaly Nikolaev, and Jennimaria Palomaki. T y d i qa: A benchmark for information-seeking question answering in ty pologically di verse languages. *Transactions of the Association for Computational Linguistics*, 8:454–470, December 2020.

- [131] Pierre Zweigenbaum, Serge Sharoff, and Reinhard Rapp. Overview of the second bucc shared task: Spotting parallel sentences in comparable corpora. In Serge Sharoff, Pierre Zweigenbaum, and Reinhard Rapp, editors, *Proceedings of the 10th Workshop on Building and Using Comparable Corpora*, page 60–67, Vancouver, Canada, August 2017. Association for Computational Linguistics.
- [132] Mikel Artetxe and Holger Schwenk. Massively multilingual sentence embeddings for zero-shot cross-lingual transfer and beyond. *Transactions of the Association for Computational Linguistics*, 7:597–610, November 2019.
- [133] Uma Roy, Noah Constant, Rami Al-Rfou, Aditya Barua, Aaron Phillips, and Yinfei Yang. Lareqa: Language-agnostic answer retrieval from a multilingual pool. (arXiv:2004.05484), April 2020. arXiv:2004.05484 [cs].
- [134] Telmo Pires, Eva Schlinger, and Dan Garrette. How multilingual is multilingual bert? (arXiv:1906.01502), June 2019. arXiv:1906.01502 [cs].
- [135] Shijie Wu and Mark Dredze. Beto, bentz, becas: The surprising cross-lingual effectiveness of bert. (arXiv:1904.09077), October 2019. arXiv:1904.09077 [cs].
- [136] Holger Schwenk and Xian Li. A corpus for multilingual document classification in eight languages. (arXiv:1805.09821), May 2018. arXiv:1805.09821 [cs].
- [137] Karthikeyan K, Zihan Wang, Stephen Mayhew, and Dan Roth. Cross-lingual ability of multilingual bert: An empirical study. (arXiv:1912.07840), February 2020. arXiv:1912.07840 [cs].
- [138] Chi-Liang Liu, Tsung-Yuan Hsu, Yung-Sung Chuang, and Hung-Yi Lee. A study of cross-lingual ability and language-specific

- information in multilingual bert. (arXiv:2004.09205), April 2020. arXiv:2004.09205 [cs].
- [139] Anne Lauscher, Vinit Ravishankar, Ivan Vulić, and Goran Glavaš. From zero to hero: On the limitations of zero-shot cross-lingual transfer with multilingual transformers. (arXiv:2005.00633), May 2020. arXiv:2005.00633 [cs].
- [140] Philipp Dufter and Hinrich Schütze. Identifying elements essential for bert’s multilinguality. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, page 4423–4437, Online, 2020. Association for Computational Linguistics.
- [141] Xinyu Wang, Yong Jiang, Nguyen Bach, Tao Wang, Fei Huang, and Kewei Tu. Structure-level knowledge distillation for multilingual sequence labeling. (arXiv:2004.03846), May 2020. arXiv:2004.03846 [cs].
- [142] Zewen Chi, Li Dong, Furu Wei, Xian-Ling Mao, and Heyan Huang. Can monolingual pretrained models help cross-lingual classification? (arXiv:1911.03913), November 2019. arXiv:1911.03913 [cs].
- [143] Zihan Liu, Genta Indra Winata, Andrea Madotto, and Pascale Fung. Exploring fine-tuning techniques for pre-trained cross-lingual models via continual learning. (arXiv:2004.14218), October 2020. arXiv:2004.14218 [cs].
- [144] Yuxuan Wang, Wanxiang Che, Jiang Guo, Yijia Liu, and Ting Liu. Cross-lingual bert transformation for zero-shot dependency parsing. (arXiv:1909.06775), September 2019. arXiv:1909.06775 [cs].
- [145] Q. Liu, D. McCarthy, I. Vulić, and A. Korhonen. *Investigating cross-lingual alignment methods for contextualized embeddings with Token-level evaluation*. January 2019.

- [146] Zirui Wang, Jiateng Xie, Ruochen Xu, Yiming Yang, Graham Neubig, and Jaime Carbonell. Cross-lingual alignment vs joint training: A comparative study and a simple unified framework. (arXiv:1910.04708), February 2020. arXiv:1910.04708 [cs].
- [147] Wei Zhao, Steffen Eger, Johannes Bjerva, and Isabelle Augenstein. Inducing language-agnostic multilingual representations. (arXiv:2008.09112), June 2021. arXiv:2008.09112 [cs].
- [148] Jasdeep Singh, Bryan McCann, Richard Socher, and Caiming Xiong. Bert is not an interlingua and the bias of tokenization. In Colin Cherry, Greg Durrett, George Foster, Reza Haffari, Shahram Khadivi, Nanyun Peng, Xiang Ren, and Swabha Swayamdipta, editors, *Proceedings of the 2nd Workshop on Deep Learning Approaches for Low-Resource NLP (DeepLo 2019)*, page 47–55, Hong Kong, China, November 2019. Association for Computational Linguistics.
- [149] Rochelle Choenni and Ekaterina Shutova. What does it mean to be language-agnostic? probing multilingual sentence encoders for typological properties. (arXiv:2009.12862), September 2020. arXiv:2009.12862 [cs].
- [150] Jindřich Libovický, Rudolf Rosa, and Alexander Fraser. How language-neutral is multilingual bert? (arXiv:1911.03310), November 2019. arXiv:1911.03310 [cs].
- [151] Ethan A. Chi, John Hewitt, and Christopher D. Manning. Finding universal grammatical relations in multilingual bert. (arXiv:2005.04511), May 2020. arXiv:2005.04511 [cs].
- [152] Wietse de Vries, Andreas van Cranenburgh, and Malvina Nissim. What’s so special about bert’s layers? a closer look at the nlp pipeline in monolingual and multilingual models. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, page 4339–4350, 2020. arXiv:2004.06499 [cs].

- [153] Xiang Zhang, Junbo Zhao, and Yann LeCun. Character-level convolutional networks for text classification. (arXiv:1509.01626), April 2016. arXiv:1509.01626 [cs].
- [154] Alexei Baevski, Henry Zhou, Abdelrahman Mohamed, and Michael Auli. wav2vec 2.0: A framework for self-supervised learning of speech representations. (arXiv:2006.11477), October 2020. arXiv:2006.11477 [cs, eess].
- [155] Sachin Mehta, Rik Koncel-Kedziorski, Mohammad Rastegari, and Hannaneh Hajishirzi. Pyramidal recurrent unit for language modeling. (arXiv:1808.09029), August 2018. arXiv:1808.09029 [cs].
- [156] Sachin Mehta, Rik Koncel-Kedziorski, Mohammad Rastegari, and Hannaneh Hajishirzi. Define: Deep factorized input token embeddings for neural sequence modeling. (arXiv:1911.12385), February 2020. arXiv:1911.12385 [cs].
- [157] Łukasz Kaiser and Ilya Sutskever. Neural gpus learn algorithms. (arXiv:1511.08228), March 2016. arXiv:1511.08228 [cs].
- [158] Alex Graves, Greg Wayne, and Ivo Danihelka. Neural turing machines. (arXiv:1410.5401), December 2014. arXiv:1410.5401 [cs].
- [159] Peter Shaw, Jakob Uszkoreit, and Ashish Vaswani. Self-attention with relative position representations. (arXiv:1803.02155), April 2018. arXiv:1803.02155 [cs].
- [160] Cheng-Zhi Anna Huang, Ashish Vaswani, Jakob Uszkoreit, Noam Shazeer, Ian Simon, Curtis Hawthorne, Andrew M. Dai, Matthew D. Hoffman, Monica Dinculescu, and Douglas Eck. Music transformer. (arXiv:1809.04281), December 2018. arXiv:1809.04281 [cs, eess, stat].
- [161] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa De-

- hghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. (arXiv:2010.11929), June 2021. arXiv:2010.11929 [cs].
- [162] Gedas Bertasius, Heng Wang, and Lorenzo Torresani. Is space-time attention all you need for video understanding? (arXiv:2102.05095), June 2021. arXiv:2102.05095 [cs].
- [163] Piotr Indyk and Rajeev Motwani. Approximate nearest neighbors: towards removing the curse of dimensionality. In *Proceedings of the thirtieth annual ACM symposium on Theory of computing - STOC '98*, page 604–613, Dallas, Texas, United States, 1998. ACM Press.
- [164] Moses S. Charikar. Similarity estimation techniques from rounding algorithms. In *Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*, STOC '02, page 380–388, New York, NY, USA, 2002. Association for Computing Machinery.
- [165] Alexandr Andoni, Piotr Indyk, Thijs Laarhoven, Ilya Razenshteyn, and Ludwig Schmidt. Practical and optimal lsh for angular distance. (arXiv:1509.02897), September 2015. arXiv:1509.02897 [cs].
- [166] Aidan N Gomez, Mengye Ren, Raquel Urtasun, and Roger B Grosse. The reversible residual network: Backpropagation without storing activations. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.
- [167] P Erdős and A Rényi. On the evolution of random graphs by.
- [168] Duncan J. Watts and Steven H. Strogatz. Collective dynamics of ‘small-world’ networks. *Nature*, 393(6684):440–442, June 1998.

- [169] Scott Gray, Alec Radford, and Diederik P Kingma. Gpu kernels for block-sparse weights. *arXiv preprint arXiv:1711.09224*, 3(2):2, 2017.
- [170] Xuezhe Ma, Xiang Kong, Sinong Wang, Chunting Zhou, Jonathan May, Hao Ma, and Luke Zettlemoyer. Luna: Linear unified nested attention. (arXiv:2106.01540), November 2021. arXiv:2106.01540 [cs].
- [171] Yi Tay, Dara Bahri, Liu Yang, Donald Metzler, and Da-Cheng Juan. Sparse sinkhorn attention. In *Proceedings of the 37th International Conference on Machine Learning*, page 9438–9447. PMLR, November 2020.
- [172] Carl Eckart and Gale Young. The approximation of one matrix by another of lower rank. *Psychometrika*, 1(3):211 – 218, September 1936.
- [173] Kevin Clark, Minh-Thang Luong, Quoc V. Le, and Christopher D. Manning. Electra: Pre-training text encoders as discriminators rather than generators. (arXiv:2003.10555), March 2020. arXiv:2003.10555 [cs].
- [174] Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. (arXiv:1701.06538), January 2017. arXiv:1701.06538 [cs, stat].
- [175] William Fedus, Barret Zoph, and Noam Shazeer. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. (arXiv:2101.03961), June 2022. arXiv:2101.03961 [cs].
- [176] Jesse Vig. A multiscale visualization of attention in the transformer model. (arXiv:1906.05714), June 2019. arXiv:1906.05714 [cs].

- [177] Kevin Clark, Urvashi Khandelwal, Omer Levy, and Christopher D. Manning. What does bert look at? an analysis of bert’ s attention. (arXiv:1906.04341), June 2019. arXiv:1906.04341 [cs].
- [178] Goro Kobayashi, Tatsuki Kurabayashi, Sho Yokoi, and Kentaro Inui. Attention is not only a weight: Analyzing transformers with vector norms. (arXiv:2004.10102), October 2020. arXiv:2004.10102 [cs].
- [179] Jinhyuk Lee, Wonjin Yoon, Sungdong Kim, Donghyeon Kim, Sunkyu Kim, Chan Ho So, and Jaewoo Kang. Biobert: a pre-trained biomedical language representation model for biomedical text mining. *Bioinformatics*, 36(4):1234–1240, February 2020.
- [180] Iz Beltagy, Kyle Lo, and Arman Cohan. Scibert: A pretrained language model for scientific text. (arXiv:1903.10676), September 2019. arXiv:1903.10676 [cs].
- [181] Jeremy Howard and Sebastian Ruder. Universal language model fine-tuning for text classification. (arXiv:1801.06146), May 2018. arXiv:1801.06146 [cs, stat].
- [182] Dogu Araci. Finbert: Financial sentiment analysis with pre-trained language models. (arXiv:1908.10063), August 2019. arXiv:1908.10063 [cs].
- [183] Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. Improving language understanding by generative pre-training.
- [184] Yukun Zhu, Ryan Kiros, Rich Zemel, Ruslan Salakhutdinov, Raquel Urtasun, Antonio Torralba, and Sanja Fidler. Aligning books and movies: Towards story-like visual explanations by watching movies and reading books. page 19–27, 2015.
- [185] Matthew E. Peters, Mark Neumann, Robert L. Logan IV, Roy Schwartz, Vidur Joshi, Sameer Singh, and Noah A.

- Smith. Knowledge enhanced contextual word representations. (arXiv:1909.04164), October 2019. arXiv:1909.04164 [cs].
- [186] Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. Glue: A multi-task benchmark and analysis platform for natural language understanding. (arXiv:1804.07461), February 2019. arXiv:1804.07461 [cs].
- [187] Guokun Lai, Qizhe Xie, Hanxiao Liu, Yiming Yang, and Eduard Hovy. Race: Large-scale reading comprehension dataset from examinations. (arXiv:1704.04683), December 2017. arXiv:1704.04683 [cs].
- [188] Nasrin Mostafazadeh, Michael Roth, Annie Louis, Nathanael Chambers, and James Allen. Lsdsem 2017 shared task: The story cloze test. In *Proceedings of the 2nd Workshop on Linking Models of Lexical, Sentential and Discourse-level Semantics*, page 46–51, Valencia, Spain, 2017. Association for Computational Linguistics.
- [189] Bill Dolan and Chris Brockett. Automatically constructing a corpus of sentential paraphrases. January 2005.
- [190] Daniel Cer, Mona Diab, Eneko Agirre, Iñigo Lopez-Gazpio, and Lucia Specia. Semeval-2017 task 1: Semantic textual similarity - multilingual and cross-lingual focused evaluation. In *Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017)*, page 1–14, 2017. arXiv:1708.00055 [cs].
- [191] Alex Warstadt, Amanpreet Singh, and Samuel R. Bowman. Neural network acceptability judgments. (arXiv:1805.12471), October 2019. arXiv:1805.12471 [cs].
- [192] Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D. Manning, Andrew Ng, and Christopher Potts. Recursive deep models for semantic compositionality over a sentiment treebank. In David Yarowsky, Timothy Baldwin, Anna

- Korhonen, Karen Livescu, and Steven Bethard, editors, *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, page 1631–1642, Seattle, Washington, USA, October 2013. Association for Computational Linguistics.
- [193] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners.
 - [194] Philip Gage. A new algorithm for data compression. *The C Users Journal*, 12(2):23–38, 1994.
 - [195] Rewon Child, Scott Gray, Alec Radford, and Ilya Sutskever. Generating long sequences with sparse transformers. (arXiv:1904.10509), April 2019. arXiv:1904.10509 [cs, stat].
 - [196] Zilong Huang, Xinggang Wang, Lichao Huang, Chang Huang, Yunchao Wei, and Wenyu Liu. Ccnet: Criss-cross attention for semantic segmentation. page 603–612, 2019.
 - [197] Jonathan Ho, Nal Kalchbrenner, Dirk Weissenborn, and Tim Salimans. Axial attention in multidimensional transformers. (arXiv:1912.12180), December 2019. arXiv:1912.12180 [cs].
 - [198] Steffen Schneider, Alexei Baevski, Ronan Collobert, and Michael Auli. wav2vec: Unsupervised pre-training for speech recognition. (arXiv:1904.05862), September 2019. arXiv:1904.05862 [cs].
 - [199] Changhan Wang, Anne Wu, Juan Pino, Alexei Baevski, Michael Auli, and Alexis Conneau. Large-scale self- and semi-supervised learning for speech translation. (arXiv:2104.06678), April 2021. arXiv:2104.06678 [cs].
 - [200] Wei-Ning Hsu, Benjamin Bolte, Yao-Hung Hubert Tsai, Kushal Lakhotia, Ruslan Salakhutdinov, and Abdelrahman Mohamed. Hubert: Self-supervised speech representation learning by masked prediction of hidden units. (arXiv:2106.07447), June 2021. arXiv:2106.07447 [cs, eess].

- [201] Min Xu, Ling-Yu Duan, Jianfei Cai, Liang-Tien Chia, Changsheng Xu, and Qi Tian. *HMM-Based Audio Keyword Generation*, volume 3333 of *Lecture Notes in Computer Science*, page 566–574. Springer Berlin Heidelberg, Berlin, Heidelberg, 2004.
- [202] Chia-ying Lee and James Glass. A nonparametric bayesian approach to acoustic model discovery. In Haizhou Li, Chin-Yew Lin, Miles Osborne, Gary Geunbae Lee, and Jong C. Park, editors, *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, page 40–49, Jeju Island, Korea, July 2012. Association for Computational Linguistics.
- [203] R.M. Gray and D.L. Neuhoff. Quantization. *IEEE Transactions on Information Theory*, 44(6):2325–2383, October 1998.
- [204] Jiasen Lu, Dhruv Batra, Devi Parikh, and Stefan Lee. Vilbert: Pretraining task-agnostic visiolinguistic representations for vision-and-language tasks. (arXiv:1908.02265), August 2019. arXiv:1908.02265 [cs].
- [205] Ronghang Hu and Amanpreet Singh. Unit: Multimodal multitask learning with a unified transformer. (arXiv:2102.10772), August 2021. arXiv:2102.10772 [cs].
- [206] Vijay Prakash Dwivedi and Xavier Bresson. A generalization of transformer networks to graphs. (arXiv:2012.09699), January 2021. arXiv:2012.09699 [cs].
- [207] Vijay Prakash Dwivedi, Chaitanya K. Joshi, Anh Tuan Luu, Thomas Laurent, Yoshua Bengio, and Xavier Bresson. Benchmarking graph neural networks. (arXiv:2003.00982), December 2022. arXiv:2003.00982 [cs, stat].
- [208] Richard S Sutton and Andrew G Barto. Reinforcement learning: An introduction.

- [209] Lili Chen, Kevin Lu, Aravind Rajeswaran, Kimin Lee, Aditya Grover, Michael Laskin, Pieter Abbeel, Aravind Srinivas, and Igor Mordatch. Decision transformer: Reinforcement learning via sequence modeling. (arXiv:2106.01345), June 2021. arXiv:2106.01345 [cs].
- [210] Garofolo J. S. Timit acoustic phonetic continuous speech corpus. *Linguistic Data Consortium*, 1993, 1993.
- [211] Gabrielle Ras, Ning Xie, Marcel van Gerven, and Derek Doran. Explainable deep learning: A field guide for the uninitiated. (arXiv:2004.14545), September 2021. arXiv:2004.14545 [cs, stat].
- [212] Dong Huk Park, Lisa Anne Hendricks, Zeynep Akata, Anna Rohrbach, Bernt Schiele, Trevor Darrell, and Marcus Rohrbach. Multimodal explanations: Justifying decisions and pointing to the evidence. page 8779–8788, 2018.
- [213] Drew A. Hudson and Christopher D. Manning. Compositional attention networks for machine reasoning. (arXiv:1803.03067), April 2018. arXiv:1803.03067 [cs].
- [214] Gaël Letarte, Frédéric Paradis, Philippe Giguère, and François Laviolette. Importance of self-attention for sentiment analysis. In *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, page 267–275, Brussels, Belgium, 2018. Association for Computational Linguistics.
- [215] Ruidan He, Wee Sun Lee, Hwee Tou Ng, and Daniel Dahlmeier. An unsupervised neural attention model for aspect extraction. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 388–397, 2017.

- [216] Anh Nguyen, Jason Yosinski, and Jeff Clune. Deep neural networks are easily fooled: High confidence predictions for unrecognizable images. page 427–436, 2015.
- [217] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. (arXiv:1412.6572), March 2015. arXiv:1412.6572 [cs, stat].
- [218] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, Omar Fawzi, and Pascal Frossard. Universal adversarial perturbations. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1765–1773, 2017.
- [219] Xiaoyong Yuan, Pan He, Qile Zhu, and Xiaolin Li. Adversarial examples: Attacks and defenses for deep learning. *IEEE Transactions on Neural Networks and Learning Systems*, 30(9):2805–2824, September 2019.
- [220] Heinrich Jiang, Been Kim, Melody Guan, and Maya Gupta. To trust or not to trust a classifier. In *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018.
- [221] Kevin Baum, Maximilian A. Köhl, and Eva Schmidt. Two challenges for ci trustworthiness and how to address them. In *Proceedings of the 1st Workshop on Explainable Computational Intelligence (XCI 2017)*, Dundee, United Kingdom, 2017. Association for Computational Linguistics.
- [222] Kush R. Varshney and Homa Alemzadeh. On the safety of machine learning: Cyber-physical systems, decision sciences, and data products. *Big Data*, 5(3):246–255, September 2017.
- [223] Dario Amodei, Chris Olah, Jacob Steinhardt, Paul Christiano, John Schulman, and Dan Mané. Concrete problems in ai safety. (arXiv:1606.06565), July 2016. arXiv:1606.06565 [cs].

- [224] Jess Whittlestone, Rune Nyrup, Anna Alexandrova, and Stephen Cave. The role and limits of principles in ai ethics: Towards a focus on tensions. In *Proceedings of the 2019 AAAI/ACM Conference on AI, Ethics, and Society*, AIES ’19, page 195–200, New York, NY, USA, 2019. Association for Computing Machinery.
- [225] Emily M. Bender, Dirk Hovy, and Alexandra Schofield. Integrating ethics into the nlp curriculum. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics: Tutorial Abstracts*, page 6–9, Online, 2020. Association for Computational Linguistics.
- [226] Solon Barocas and Danah Boyd. Engaging the ethics of data science in practice. *Communications of the ACM*, 60(11):23–25, October 2017.
- [227] Mark Coeckelbergh. *AI ethics*. Mit Press, 2020.
- [228] Bolei Zhou, Aditya Khosla, Agata Lapedriza, Aude Oliva, and Antonio Torralba. Object detectors emerge in deep scene cnns. (arXiv:1412.6856), April 2015. arXiv:1412.6856 [cs].
- [229] Quanshi Zhang, Ruiming Cao, Ying Nian Wu, and Song-Chun Zhu. Growing interpretable part graphs on convnets via multi-shot learning. *Proceedings of the AAAI Conference on Artificial Intelligence*, 31(1), February 2017.
- [230] Maithra Raghu, Justin Gilmer, Jason Yosinski, and Jascha Sohl-Dickstein. Svcca: Singular vector canonical correlation analysis for deep learning dynamics and interpretability. (arXiv:1706.05806), November 2017. arXiv:1706.05806 [cs, stat].
- [231] Daniel Kang, Deepti Raghavan, Peter Bailis, and Matei Zaharia. Model assertions for debugging machine learning.
- [232] Guillaume Alain and Yoshua Bengio. Understanding intermediate layers using linear classifier probes. (arXiv:1610.01644), November 2018. arXiv:1610.01644 [cs, stat].

- [233] Saleema Amershi, Max Chickering, Steven M. Drucker, Bongshin Lee, Patrice Simard, and Jina Suh. Modeltracker: Redesigning performance analysis tools for machine learning. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*, page 337–346, Seoul Republic of Korea, April 2015. ACM.
- [234] F Fuchs, Oliver Groth, A Kosiorek, Alex Bewley, Markus Wulfmeier, Andrea Vedaldi, and H Posner. Neural stethoscopes: Unifying analytic, auxiliary and adversarial network probing. *arXiv*, 2018.
- [235] Wei Emma Zhang, Quan Z Sheng, Ahoud Abdulrahmn F Alhazmi, and Chenliang Li. Generating textual adversarial examples for deep learning models: A survey. *arXiv preprint arXiv:1901.06796*, page 129, 2019.
- [236] Jiawei Su, Danilo Vasconcellos Vargas, and Kouichi Sakurai. One pixel attack for fooling deep neural networks. *IEEE Transactions on Evolutionary Computation*, 23(5):828–841, October 2019.
- [237] Kevin Eykholt, Ivan Evtimov, Earlene Fernandes, Bo Li, Amir Rahmati, Chaowei Xiao, Atul Prakash, Tadayoshi Kohno, and Dawn Song. Robust physical-world attacks on deep learning visual classification. page 1625–1634, 2018.
- [238] Yinpeng Dong, Fangzhou Liao, Tianyu Pang, Hang Su, Jun Zhu, Xiaolin Hu, and Jianguo Li. Boosting adversarial attacks with momentum. page 9185–9193, 2018.
- [239] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. (*arXiv:1706.06083*), September 2019. *arXiv:1706.06083 [cs, stat]*.
- [240] Pouya Samangouei, Maya Kabkab, and Rama Chellappa. Defense-gan: Protecting classifiers against adversarial attacks

- using generative models. (arXiv:1805.06605), May 2018. arXiv:1805.06605 [cs, stat].
- [241] Toon Calders, Faisal Kamiran, and Mykola Pechenizkiy. Building classifiers with independency constraints. In *2009 IEEE International Conference on Data Mining Workshops*, page 13–18, Miami, FL, USA, December 2009. IEEE.
 - [242] Cynthia Dwork, Nicole Immorlica, Adam Tauman Kalai, and Max Leiserson. Decoupled classifiers for group-fair and efficient machine learning. In *Proceedings of the 1st Conference on Fairness, Accountability and Transparency*, page 119–133. PMLR, January 2018.
 - [243] Flavio Calmon, Dennis Wei, Bhanukiran Vinzamuri, Karthikeyan Natesan Ramamurthy, and Kush R Varshney. Optimized pre-processing for discrimination prevention. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.
 - [244] Faisal Kamiran and Toon Calders. Data preprocessing techniques for classification without discrimination. *Knowledge and Information Systems*, 33(1):1–33, October 2012.
 - [245] Paula Gordaliza, Eustasio Del Barrio, Gamboa Fabrice, and Jean-Michel Loubes. Obtaining fairness using optimal transport theory. In *Proceedings of the 36th International Conference on Machine Learning*, page 2357–2365. PMLR, May 2019.
 - [246] Alex Beutel, Jilin Chen, Zhe Zhao, and Ed H. Chi. Data decisions and theoretical implications when adversarially learning fair representations. (arXiv:1707.00075), July 2017. arXiv:1707.00075 [cs].
 - [247] Dumitru Erhan, Y. Bengio, Aaron Courville, and Pascal Vincent. Visualizing higher-layer features of a deep network. *Technical Report, Univeristé de Montréal*, January 2009.

- [248] Sebastian Bach, Alexander Binder, Gr  goire Montavon, Frederick Klauschen, Klaus-Robert M  ller, and Wojciech Samek. On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation. *PLOS ONE*, 10(7):e0130140, July 2015.
- [249] Yanzhuo Ding, Yang Liu, Huanbo Luan, and Maosong Sun. Visualizing and understanding neural machine translation. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, page 1150 – 1159, Vancouver, Canada, 2017. Association for Computational Linguistics.
- [250] Sebastian Lapuschkin, Alexander Binder, Gregoire Montavon, Klaus-Robert Muller, and Wojciech Samek. Analyzing classifiers: Fisher vectors and deep neural networks. page 2912–2920, 2016.
- [251] Gr  goire Montavon, Wojciech Samek, and Klaus-Robert M  ller. Methods for interpreting and understanding deep neural networks. *Digital Signal Processing*, 73:1–15, February 2018.
- [252] Gr  goire Montavon, Sebastian Lapuschkin, Alexander Binder, Wojciech Samek, and Klaus-Robert M  ller. Explaining nonlinear classification decisions with deep taylor decomposition. *Pattern Recognition*, 65:211–222, May 2017.
- [253] Avanti Shrikumar, Peyton Greenside, and Anshul Kundaje. Learning important features through propagating activation differences. In *Proceedings of the 34th International Conference on Machine Learning*, page 3145–3153. PMLR, July 2017.
- [254] Ekaterina Arkhangelskaia and Sourav Dutta. Whatcha lookin’at? deeplifting bert’s attention in question answering. (arXiv:1910.06431), October 2019. arXiv:1910.06431 [cs].

- [255] Mukund Sundararajan, Ankur Taly, and Qiqi Yan. Axiomatic attribution for deep networks. In *Proceedings of the 34th International Conference on Machine Learning*, page 3319–3328. PMLR, July 2017.
- [256] Jiwei Li, Will Monroe, and Dan Jurafsky. Understanding neural networks through representation erasure. (arXiv:1612.08220), January 2017. arXiv:1612.08220 [cs].
- [257] Ruth Fong and Andrea Vedaldi. Interpretable explanations of black boxes by meaningful perturbation. In *2017 IEEE International Conference on Computer Vision (ICCV)*, page 3449–3457, October 2017. arXiv:1704.03296 [cs, stat].
- [258] Marko Robnik-Šikonja and Igor Kononenko. Explaining classifications for individual instances. *IEEE Transactions on Knowledge and Data Engineering*, 20(5):589–600, 2008.
- [259] Luisa M. Zintgraf, Taco S. Cohen, Tameem Adel, and Max Welling. Visualizing deep neural network decisions: Prediction difference analysis. (arXiv:1702.04595), February 2017. arXiv:1702.04595 [cs].
- [260] David Baehrens, Timon Schroeter, Stefan Harmeling, Motoaki Kawanabe, and Katja Hansen. How to explain individual classification decisions.
- [261] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. “why should i trust you?”: Explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD ’16, page 1135–1144, New York, NY, USA, 2016. Association for Computing Machinery.
- [262] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. Anchors: High-precision model-agnostic explanations. *Proceedings*

of the AAAI Conference on Artificial Intelligence, 32(11), April 2018.

- [263] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. (arXiv:1503.02531), March 2015. arXiv:1503.02531 [cs, stat].
- [264] Junho Yim, Donggyu Joo, Jihoon Bae, and Junmo Kim. A gift from knowledge distillation: Fast optimization, network minimization and transfer learning. page 4133–4141, 2017.
- [265] Osbert Bastani, Carolyn Kim, and Hamsa Bastani. Interpreting blackbox models via model extraction. (arXiv:1705.08504), January 2019. arXiv:1705.08504 [cs].
- [266] Bo-Jian Hou and Zhi-Hua Zhou. Learning with interpretable structure from gated rnn. (arXiv:1810.10708), January 2020. arXiv:1810.10708 [cs].
- [267] Sarah Tan, Rich Caruana, Giles Hooker, Paul Koch, and Albert Gordo. Learning global additive explanations for neural nets using model distillation. 2018.
- [268] Gabrielle Ras, Marcel van Gerven, and Pim Haselager. Explanation methods in deep learning: Users, values, concerns and challenges. (arXiv:1803.07517), March 2018. arXiv:1803.07517 [cs, stat].
- [269] Yonatan Belinkov. Probing classifiers: Promises, shortcomings, and advances. (arXiv:2102.12452), September 2021. arXiv:2102.12452 [cs].
- [270] Dieuwke Hupkes, Sara Veldhoen, and Willem Zuidema. Visualisation and “diagnostic classifiers” reveal how recurrent and recursive neural networks process hierarchical structure. *Journal of Artificial Intelligence Research*, 61:907–926, April 2018.

- [271] Ian Tenney, Patrick Xia, Berlin Chen, Alex Wang, Adam Poliak, R. Thomas McCoy, Najoung Kim, Benjamin Van Durme, Samuel R. Bowman, Dipanjan Das, and Ellie Pavlick. What do you learn from context? probing for sentence structure in contextualized word representations. (arXiv:1905.06316), May 2019. arXiv:1905.06316 [cs].
- [272] John Hewitt and Percy Liang. Designing and interpreting probes with control tasks. (arXiv:1909.03368), September 2019. arXiv:1909.03368 [cs].
- [273] Tim Rocktäschel, Edward Grefenstette, Karl Moritz Hermann, Tomáš Kočiský, and Phil Blunsom. Reasoning about entailment with neural attention. (arXiv:1509.06664), March 2016. arXiv:1509.06664 [cs].
- [274] Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhudinov, Rich Zemel, and Yoshua Bengio. Show, attend and tell: Neural image caption generation with visual attention. In *Proceedings of the 32nd International Conference on Machine Learning*, page 2048–2057. PMLR, June 2015.
- [275] Zichao Yang, Diyi Yang, Chris Dyer, Xiaodong He, Alex Smola, and Eduard Hovy. Hierarchical attention networks for document classification. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, page 1480 – 1489, San Diego, California, 2016. Association for Computational Linguistics.
- [276] Lisa Hendricks, Zeynep Akata, Marcus Rohrbach, Jeff Donahue, Bernt Schiele, and Trevor Darrell. *Generating Visual Explanations*, volume 9908. October 2016.

- [277] Rowan Zellers, Yonatan Bisk, Ali Farhadi, and Yejin Choi. From recognition to cognition: Visual commonsense reasoning. page 6720–6731, 2019.
- [278] Hui Liu, Qingyu Yin, and William Yang Wang. Towards explainable nlp: A generative explanation framework for text classification. (arXiv:1811.00196), June 2019. arXiv:1811.00196 [cs].
- [279] Michael Hind, Dennis Wei, Murray Campbell, Noel C. F. Codella, Amit Dhurandhar, Aleksandra Mojsilović, Karthikeyan Natesan Ramamurthy, and Kush R. Varshney. Ted: Teaching ai to explain its decisions. In *Proceedings of the 2019 AAAI/ACM Conference on AI, Ethics, and Society*, AIES ’19, page 123–129, New York, NY, USA, 2019. Association for Computing Machinery.
- [280] Oana-Maria Camburu, Tim Rocktäschel, Thomas Lukasiewicz, and Phil Blunsom. e-snli: Natural language inference with natural language explanations. (arXiv:1812.01193), December 2018. arXiv:1812.01193 [cs].
- [281] David Alvarez-Melis and Tommi S. Jaakkola. Towards robust interpretability with self-explaining neural networks. (arXiv:1806.07538), December 2018. arXiv:1806.07538 [cs, stat].
- [282] Yinpeng Dong, Hang Su, Jun Zhu, and Bo Zhang. Improving interpretability of deep neural networks with semantic information. page 4306–4314, 2017.
- [283] Rahul Iyer, Yuezheng Li, Huao Li, Michael Lewis, Ramitha Sundar, and Katia Sycara. Transparency and explanation in deep reinforcement learning neural networks. In *Proceedings of the 2018 AAAI/ACM Conference on AI, Ethics, and Society*, AIES ’18, page 144–150, New York, NY, USA, 2018. Association for Computing Machinery.

- [284] Tao Lei, Regina Barzilay, and Tommi Jaakkola. Rationalizing neural predictions. (arXiv:1606.04155), November 2016. arXiv:1606.04155 [cs].
- [285] Oscar Li, Hao Liu, Chaofan Chen, and Cynthia Rudin. Deep learning for case-based reasoning through prototypes: A neural network that explains its predictions. *Proceedings of the AAAI Conference on Artificial Intelligence*, 32(1), April 2018.
- [286] Chaofan Chen, Oscar Li, Chaofan Tao, Alina Jade Barnett, Jonathan Su, and Cynthia Rudin. This looks like that: Deep learning for interpretable image recognition. (arXiv:1806.10574), December 2019. arXiv:1806.10574 [cs, stat].
- [287] Sarthak Jain and Byron C. Wallace. Attention is not explanation. (arXiv:1902.10186), May 2019. arXiv:1902.10186 [cs].
- [288] Sarah Wiegreffe and Yuval Pinter. Attention is not explanation. (arXiv:1908.04626), September 2019. arXiv:1908.04626 [cs].
- [289] Samira Abnar and Willem Zuidema. Quantifying attention flow in transformers. (arXiv:2005.00928), May 2020. arXiv:2005.00928 [cs].