**目 录**

[摘要 1](#_Toc134811592)

[Abstract 2](#_Toc134811593)

[1. 绪论 3](#_Toc134811594)

[1.1. 研究背景与意义 3](#_Toc134811595)

[1.2. 国内外研究现状 3](#_Toc134811596)

[1.3. 本文研究内容 5](#_Toc134811597)

[1.4. 论文组织结构 5](#_Toc134811598)

[2. 相关技术背景 6](#_Toc134811599)

[2.1. 自然语言处理 6](#_Toc134811600)

[2.2. 文本表示 7](#_Toc134811601)

[2.2.1. 传统文本表示方法 7](#_Toc134811602)

[2.2.2. 词向量 7](#_Toc134811603)

[2.3. 分词方法 8](#_Toc134811604)

[2.4. 经典语言模型 10](#_Toc134811605)

[2.4.1. 统计语言模型 10](#_Toc134811606)

[2.4.2. 深度神经网络 11](#_Toc134811607)

[2.4.3. 神经网络语言模型 15](#_Toc134811608)

[2.4.4. 词向量层 16](#_Toc134811609)

[2.4.5. 卷积神经网络语言模型 16](#_Toc134811610)

[2.4.6. 循环神经网络语言模型 18](#_Toc134811611)

[2.5. 预训练词向量语言模型 21](#_Toc134811612)

[2.5.1. 静态词向量语言模型 21](#_Toc134811613)

[2.5.2. 动态词向量语言模型 23](#_Toc134811614)

[2.6. 文本分类问题 25](#_Toc134811615)

[2.7. 经典文本分类模型 26](#_Toc134811616)

[2.7.1. FastText 26](#_Toc134811617)

[2.7.2. TextCNN 27](#_Toc134811618)

[2.7.3. TextRNN 28](#_Toc134811619)

[3. 本文研究相关理论 29](#_Toc134811620)

[3.1. Transformer模型 29](#_Toc134811621)

[3.1.1. 传统序列到序列模型 29](#_Toc134811622)

[3.1.2. Transformer模型的突破 30](#_Toc134811623)

[3.1.3. Transformer Encoder详解 30](#_Toc134811624)

[3.1.4. 位置编码 34](#_Toc134811625)

[3.1.5. Transformer Decoder简述 40](#_Toc134811626)

[3.2. BERT模型 41](#_Toc134811627)

[3.2.1. 由GPT得到的灵感 41](#_Toc134811628)

[3.2.2. Bert模型结构 42](#_Toc134811629)

[3.2.3. BERT预训练任务 44](#_Toc134811630)

[3.2.4. BERT模型输入 45](#_Toc134811631)

[3.2.5. BERT模型前向运算过程 48](#_Toc134811632)

[3.2.6. BERT用于文本分类下游任务 49](#_Toc134811633)

[3.2.7. BERT模型的缺陷 50](#_Toc134811634)

[3.3. RoBERTa模型 50](#_Toc134811635)

[3.3.1. RoBERTa模型的改进 50](#_Toc134811636)

[3.3.2. RoBERTa模型用于中文 51](#_Toc134811637)

[3.4. 全词掩码预训练方法 52](#_Toc134811638)

[3.5. BERT增量预训练 53](#_Toc134811639)

[3.5.1. 经典语言模型的增量预训练 53](#_Toc134811640)

[3.5.2. 用于专业领域的BERT模型 53](#_Toc134811641)

[3.5.3. 以专业知识区分领域的增量预训练 54](#_Toc134811642)

[3.5.4. 以任务类型区分领域的增量预训练 54](#_Toc134811643)

[3.6. BERT微调技巧 55](#_Toc134811644)

[3.6.1. 较低的学习率 55](#_Toc134811645)

[3.6.2. 动态学习率调节器 56](#_Toc134811646)

[3.6.3. 层次递减学习率 57](#_Toc134811647)

[3.6.4. 融合不同层提取的特征 57](#_Toc134811648)

[3.6.5. 长文本的处理 57](#_Toc134811649)

[3.6.6. 多任务微调 58](#_Toc134811650)

[3.7. BERT的对抗训练 58](#_Toc134811651)

[3.7.1. FGSM与FGM 58](#_Toc134811652)

[3.7.2. PGD 59](#_Toc134811653)

[3.8. 文本多分类评价标准 60](#_Toc134811654)

[3.9. 改进型损失函数 61](#_Toc134811655)

[3.10. 混合精度训练 63](#_Toc134811656)

[4. 实验部分 64](#_Toc134811657)

[4.1. 实验环境 64](#_Toc134811658)

[4.2. Hugging Face平台介绍 64](#_Toc134811659)

[4.3. 数据集介绍 65](#_Toc134811660)

[4.3.1. 数据清洗 65](#_Toc134811661)

[4.3.2. 数据集分析 66](#_Toc134811662)

[4.4. 实验一 基准实验 67](#_Toc134811663)

[4.4.1. 实验目的 67](#_Toc134811664)

[4.4.2. 实验模型介绍 67](#_Toc134811665)

[4.4.3. 实验内容 68](#_Toc134811666)

[4.4.4. 实验结果 69](#_Toc134811667)

[4.5. 消融实验一 BF16自动混合精度训练 71](#_Toc134811668)

[4.6. 消融实验二 灾难性遗忘 72](#_Toc134811669)

[4.7. 消融实验三 最佳训练epoch 73](#_Toc134811670)

[4.8. 消融实验四 动态学习率调节器的选择 74](#_Toc134811671)

[4.9. 实验二 层次递减学习率 75](#_Toc134811672)

[4.9.1. 实验目的 75](#_Toc134811673)

[4.9.2. 实验内容 75](#_Toc134811674)

[4.9.3. 实验结果 75](#_Toc134811675)

[4.10. 实验三 截断方法 76](#_Toc134811676)

[4.10.1. 实验目的 76](#_Toc134811677)

[4.10.2. 实验内容 76](#_Toc134811678)

[4.10.3. 实验结果 76](#_Toc134811679)

[4.11. 实验四 后四层特征的融合方法 77](#_Toc134811680)

[4.11.1. 实验目的 77](#_Toc134811681)

[4.11.2. 实验内容 77](#_Toc134811682)

[4.11.3. 实验结果 77](#_Toc134811683)

[4.12. 实验五 改进型损失函数对比 78](#_Toc134811684)

[4.12.1. 实验目的 78](#_Toc134811685)

[4.12.2. 实验内容 78](#_Toc134811686)

[4.12.3. 实验结果 79](#_Toc134811687)

[4.13. 实验六 对抗训练 80](#_Toc134811688)

[4.13.1. 实验目的 80](#_Toc134811689)

[4.13.2. 实验内容 80](#_Toc134811690)

[4.13.3. 实验结果 80](#_Toc134811691)

[4.14. 实验七 MLM增量预训练 81](#_Toc134811692)

[4.14.1. 实验目的 81](#_Toc134811693)

[4.14.2. 实验内容 81](#_Toc134811694)

[4.14.3. 实验结果 82](#_Toc134811695)

[4.15. 实验八 NSP增量预训练 85](#_Toc134811696)

[4.15.1. 实验目的： 85](#_Toc134811697)

[4.15.2. 实验内容: 85](#_Toc134811698)

[4.15.3. 实验结果: 85](#_Toc134811699)

[5. 实验结果分析 87](#_Toc134811700)

[6. 总结与展望 88](#_Toc134811701)

[谢辞 89](#_Toc134811702)

[参考文献 90](#_Toc134811703)

**基于BERT的中文期刊论文分类方法研究**

摘要

**：**中文期刊论文分类任务中的文本通常长度较长且具有极强的专业性，给人工分类和自动分类都造成了很大的困难。长期以来，该问题一直缺少的简单且高效的解决方案。近年来，以BERT为代表的预训练语言模型在自然语言处理领域得到了广泛的关注和应用。这些模型利用大规模语料库进行预训练，然后通过微调适应不同的下游任务，取得了很好的效果。这种 “预训练+微调”的方法已经成为了自然语言处理的新范式，使得解决自然语言处理任务不再需要了解非常多的任务背景知识，只要根据任务的输入输出形式应用这些预训练语言模型就能达到一个不错的效果。相关的跟进研究也探索并总结出了许多的设计更合理的模型结构，更有效的预训练任务，更加巧妙的微调策略，以及通过增量预训练优化模型在专业任务上的性能的方法。但是这些研究集中在英文模型与英文数据集领域，缺少针对中文的优化与对比分析。本文以中文期刊论文分类任务为例，选择BERT与RoBERTa模型，对多种微调策略、增量预训练任务、对抗训练方法、改进型损失函数进行了有效性验证和对比分析。通过这些实验结果，为中文领域的预训练语言模型应用积累了经验。

**关键词：**自然语言处理；BERT；文本分类；预训练模型；深度学习

**Research on Chinese journal paper classification method based on BERT**

Abstract

**：**In the task of Chinese journal papers classification, the texts are typically long and highly specialized, posing great challenges for both manual and automatic classification. For a long time, there has been a lack of simple and efficient solutions to this problem. In recent years, pre-trained language models such as BERT have received widespread attention and application in the field of natural language processing. These models are pre-trained on large-scale corpora and then fine-tuned for different downstream tasks, achieving good performance. This "pre-training + fine-tuning" method has become a new paradigm in natural language processing, making it unnecessary to have extensive background knowledge of specific tasks. By applying these pre-trained language models based on the input-output format of the given task, one can achieve good results. Follow-up studies have explored and summarized many model structures, more effective pre-training tasks, more clever fine-tuning strategies, and methods for optimizing the performance of models on professional tasks through further pre-training. However, these studies focus on English models and English datasets, lacking optimization and comparative analysis for Chinese. This article takes the classification task of Chinese journal papers as an example, using BERT and RoBERTa models to conduct effectiveness testing and comparative analysis on multiple fine-tuning strategies, incremental pre-training tasks, adversarial training methods, and novel loss functions. Through these experimental results, experience is accumulated for the application of pre-trained language models in the Chinese domain.

**Key words:** Natural Language Processing; BERT; Text Classification; Pre-training Model; Deep Learning

## 绪论

### 研究背景与意义

根据中国知网提供的数据，自1915年至今，我国已累计发行中文学术期刊8730余种，累计文献量达6000余万篇。对这些浩如烟海的文献进行合理分类，能够极大的提高信息检索的效率与准确性，不仅有助于得同一领域的学者、科研机构等之间查阅和交流彼此的论文，更好地进行知识分享与合作，也有助于发现某个领域目前的研究热点、研究趋势等，从而为未来的研究提供指导和参考。

传统的期刊论文分类方法，一般使用论文的元信息（标题、摘要和关键字），结合论文被刊登的期刊的信息进行推测。由于这些文本整体长度较长，且专业性极强，在包含大量各自领域内部的特定术语和概念的同时，又会涉及到跨领域的知识和广泛的学科背景，给人工分类和自动分类都造成了很大的困难。

近年来，以BERT为代表的预训练语言模型在自然语言处理领域得到了广泛的关注和应用。这些模型利用大规模语料库进行预训练，然后通过微调适应不同的下游任务，取得了很好的效果。这种 “预训练+微调”的方法已经成为了自然语言处理的新范式，使得解决自然语言处理任务不再需要了解非常多的任务背景知识，只要根据任务的输入输出形式应用这些预训练语言模型就能达到一个不错的效果。相关的跟进研究也探索并总结出了许多的设计更合理的模型结构，更有效的预训练任务，更加巧妙的微调策略，以及通过增量预训练优化模型在专业任务上的性能的方法。但是这些研究集中在英文模型与英文数据集领域，缺少针对中文的优化与对比分析。

在中文期刊论文分类任务上应用上述方法，不仅可以快速搭建出一套该问题的高性价比解决方案，同时，中文期刊论文分类任务的特点与难点，也使得其非常适合作为对上述方法进行有效性验证和对比分析的代表性任务，相关实验的结果，将为中文领域的预训练语言模型应用积累经验。

### 国内外研究现状

早期的自然语言处理，主要依赖于专家总结的逻辑规则，或者离散的手工特征。随着深度学习的发展，各种神经网络也被广泛用于解决自然语言处理任务，而预训练一直是学习深度神经网络参数的有效策略。预训练模型在自然语言处理中最初的应用，主要是预训练的单词嵌入，或者说预训练词向量，用于学习单词稠密的分布式表示。这方面的先驱工作是2003年由Bengio等人[1]提出的神经网络语言模型（NNLM）。之后，Collobert等人[2]的工作表明，在未标记数据上预训练的词向量可以显著改善许多NLP任务。2013年，Mikolov等人[3]提出了两种简单的浅层架构，连续词袋模型（CBOW）和跳字模型（Skip-gram），可以学习高质量的预训练词向量，以捕捉单词之间潜在的句法和语义相似性。但是这些词向量是上下文无关的，当用于超过单词级别的任务时，整个模型的其他部分还需要从头开始学习。针对这一问题，大量可以根据上下文语义生成词向量的方法被提出。其中具有代表性的是2018年Peters等[4]使用双向LSTM搭建双向语言模型（BiLM）进行预训练的ELMo模型，该模型输出的上下文表示在广泛的NLP任务上带来了很大的改进。类似ELMo等基于RNN进行序列建模的模型，虽然易于训练，但是一直存在难以捕捉单词之间的远距离的相互作用的长距离依赖问题和低并行度问题。

2017年，Google公司的Vaswani 等人[5]提出的Transformer模型使用self-attention机制作为核心算法，可以直接对序列中每两个词之间的依赖关系进行建模，更适合建模语言的长距离依赖关系，并且易于并行计算，迅速成为了包括预训练语言模型在内的大量自然语言处理模型的主流框架。然而，由于其庞大的结构和较少的归纳偏置，Transformer 通常需要大量的训练语料库，并且很容易在小型或中等大小的数据集上过拟合[6]。2018年，OpenAI公司的Radford等人[6]基于Transformer模型，提出了GPT模型。该模型先在大规模语料库上进行单向语言模型任务的生成式预训练，然后通过微调使其适配到下游任务当中，取得了很好的效果。受到该工作的启发， Devlin等人[7]在同年提出了具有双向语言建模能力的BERT模型，以及掩码语言模型（MLM）和下一句预测（NSP）两个预训练任务。2019年，Yinhan Liu等[8]提出了RoBERTa模型，在BERT基础上进行了多种改进，包括更充分的预训练和去掉NSP任务等等。同年，Zhilin Yang等人[9]将自回归引入了双向语言模型，弥补了BERT模型的缺陷。2021年，哈尔滨工业大学的崔一鸣团队[10]提出了全词掩码预训练方法，可以极大地弥补BERT在用于中文时的词典粒度过细的缺陷，在英文等其他语言上可以改善模型的性能。该团队也在公开社区发布了大量基于全词掩码预训练的中文BERT模型，为相关的后续研究提供了基准。

关于在文本分类问题上应用预训练语言模型的微调技巧的研究，在BERT等大规模预训练模型兴起之前就有过大量研究，其中的代表性工作就是2018年Howard与Ruder[11]总结的通用语言模型微调（ULMFiT）方法。该研究搭建了一个3层LSTM语言模型，经过在大规模通用语料上进行语言模型任务预训练、在分类任务语料上进行语言模型任务增量预训练、在分类任务语料上进行分类任务微调三个步骤，结合逐层递减学习率等微调策略，最终取得了很好的效果。2019年，复旦大学的邱锡鹏团队[12]使用跨3种任务类型的7个分类任务数据集，测试并总结了多种对BERT有效的微调策略，同时总结了领域内增量预训练、任务内增量预训练、跨领域增量预训练对BERT在分类任务上的提升及相关规律。2020年，Gururangan等人[13]在跨4个学科专业领域的8个分类任务上进行对比实验，并提出了领域自适应预训练（DAPT）、任务自适应预训练（TAPT）、增强训练数据的任务自适应预训练（ATD-TAPT）等增量预训练方法，可以使BERT进一步学习专业领域和具体任务的知识。同年，清华大学的刘知远团队[14]在此基础上提出的基于任务引导进行选择性掩码（Selective Mask）的增量预训练方法，使BERT正有针对性地学习对任务更重要的知识。目前，“预训练+增量预训练+微调”已经成为文本分类任务的新范式。

### 本文研究内容

本文以中文期刊论文分类任务为例，选择BERT与RoBERTa模型，对以下方法的有效性进行了验证和对比分析。

1. 微调策略：最佳微调学习率、动态学习率调节器、层次递减学习率、输出层特征融合、长文本截断方法。
2. 改进型损失函数：Weighted Cross-entropy Loss、Focal Loss、Dice Loss
3. 对抗训练方法：FGSM、FGM、二阶PGD
4. 增量预训练方法：MLM、全词掩码MLM、NSP

这些实验结果，将为中文领域的预训练语言模型应用积累经验。

### 论文组织结构

本文分为五个章节，具体章节内容安排如下：

第1章为绪论。该章节将对本文的研究背景与意义进行介绍，包括对中文期刊论文分类任务的特点与难点的分析，解释本文选择其作为代表性任务的原因，以及自然语言处理预训练模型的国内外研究现状，最后对本文研究内容和章节安排进行简要叙述。

第2章为相关技术背景。该章节将对自然语言处理的经典研究内容与方法进行叙述，并介绍了One-Hot、TF-IDF、静态词向量、动态词向量等文本表示方法，以及统计语言模型、神经网络语言模型、基于RNN的语言模型等经典语言模型。之后本章将详细介绍文本分类问题的解决流程，以及基于DNN、CNN、RNN的多种传统文本分类模型。

第3章为本文研究相关理论。该章节将对BERT模型的原理、结构、预训练任务、微调策略进行详细的解析，并且介绍BERT模型存在的几种缺陷与改进方法。之后将对BERT模型用于文本分类任务的增量预训练方法、对抗训练方法、改进型损失函数的原理与实现进行详细的解析。

第4章为实验部分。该章节将首先对本文使用的数据集和实验环境进行介绍，然后详细叙述本文的各实验目的、实验内容，以及对实验结果的分析。

第5章为总结与展望。该章节将对本文研究的结果进行总结，指出本文研究的不足，并对相关研究的未来进行展望。

## 相关技术背景

### 自然语言处理

自然语言（Natural Language），通常指的是人类语言（本文中特指文本符号而非语音信号）。作为人类思维的载体和交流基本工具，它不仅是人类区别于动物的根本标志，更是人类智能发展的重要体现形式。自然语言处理（Natural Language Processing，NLP），主要研究使用计算机理解和生成自然语言的各种理论与方法，是人工智能领域的一个重要甚至核心的分支，也被称为计算语言学（Computational Linguistics）。自然语言处理的目标是使得计算机能像人类一样理解和运用自然语言。

早期的自然语言处理主要采用基于理性主义的规则方法，使用专家总结的符号逻辑知识来处理通用自然语言。但是，由于自然语言的复杂性，这种方法在处理复杂场景时常常表现不佳。从20世纪90年代开始，随着计算机运算速度和存储容量的快速增加，以及统计学习方法的成熟，使得以语料库为核心的统计学习方法在自然语言处理领域大规模应用。大规模的语料库中包含了大量语言的知识，因此基于语言库的自然统计处理方法更加客观。但是，在人类世界中对自然语言的理解是建立在文字基础上的，而从信息的角度来看，文字只是信息的载体，而非信息本身。因此，要让计算机能够处理人类语言，首先需要将文本数据的载体转换为数值，再将数值化后的文本数据输入计算机中进行处理，从而完成特定的计算任务。基于语言库的自然统计处理方法需要事先利用经验性规则将原始的自然语言输入转化为机器可理解的向量形式，这一处理过程称为特征提取（Feature Extraction）。特征提取需要细致的人工操作和一定的专业知识，因此也被称为特征工程（Feature Engineering）。显然，以上方法都需要耗费大量人力，并且需要针对各个具体任务设计专门的特征提取方法，很难迁移到其他任务上，这限制了自然语言处理技术的广泛应用。

在过去的十年里，自然语言处理经历了两次里程碑式的重要发展。第一次是深度学习的兴起使得传统的特征工程方法被摒弃。基于深度神经网络的表示学习技术可以自动地发现有效的特征，消除了不同任务所需的不同特征之间的障碍，并迅速成为自然语言处理的主流方法。但是，基于深度学习的方法存在一个严重的缺点：过度依赖于大规模标注数据。第二次是2018年以来预训练语言模型的应用，开启了基于预训练（Pretrain）加微调（Finetune）的新一代自然语言处理范式。该技术很好的弥补了自然语言处理标注数据不足的缺陷，为自然语言处理系统带来了巨大的进步。

目前人们普遍认为人工智能的发展，经历了从运算智能到感知智能再到认知智能的三个阶段。运算智能关注的是机器的基础运算和存储能力，这方面机器已经远远超过人类。感知智能则强调机器的模式识别能力，比如语音的识别以及图像的识别。目前机器在感知智能上的水平基本达到了，甚至超过了人类的水平。然而在涉及自然语言处理以及长时建模和推理等研究的认知智能上，机器和人类还有很大的差距。这是由于自然语言是人类对现实世界的抽象与总结，每个抽象符号背后都对应着复杂的思维概念。这些符号有着变化无穷的组合性，无处不在的歧义性，不断发展的进化性与难以准确平均的主观性等等难以处理的性质，为自然语言处理造成了众多问题。此外，自然语言是一种典型的序列模型，各抽象符号除了自身的含义外，其时间与空间上的先后顺序也含有大量信息，并对整个序列的含义有着决定性作用。种种难点使得自然语言处理成为了制约人工智能取得更大突破和更广泛应用的瓶颈之一，被称为“人工智能皇冠上的明珠”。

随着互联网的快速发展，网络文本的规模呈爆炸性增长，为自然语言处理提出了巨大的应用需求同时，自然语言处理研究也为人们更深刻的理解自然语言的机理和社会的机制，提供了一条重要途径。综合来看，自然语言处理领域有着广阔的发展空间与应用空间，并将在科技进步与社会发展的过程中发挥越来越重要的作用。

### 文本表示

自然语言中的文本是无法被计算机直接处理的，需要将其转换为数值或者向量形式，这一过程被称为文本表示（Text representation）。按照表示的形式，文本表示可分为离散表示、分布式表示。其发展历程趋势整体上是从高维、离散、稀疏到低维、连续、稠密。

#### 传统文本表示方法

最简单的文本表示方法是独热(One-Hot)编码。对于词典和词, 的独热编码是一个维向量，该向量的第i分量的值为1，其余值为0，记为

独热编码这种简单的编号方式只能描述不同的子词，无法表示词的内部含义。研究者随后提出词频-逆文档频率(Term Frequency-Inverse Document Frequency，TF-IDF)编码。该方法认为一个词的相对重要性与它在一个文档中出现的次数成正比，同时与其在语料库中出现的次数成反比。通过数值化这种相对重要性获取词的表示。但TF-IDF仍存在忽略词语义信息和一定的数据稀疏问题。之后有研究者将n-gram方法与One-Hot编码或TF-IDF编码结合，使得编码结果可以一定程度上获取到词的前后信息。但随着n的增大，词典的维数呈会呈现指数增长，对算力要求很高而且同样会出现数据稀疏问题。、

#### 词向量

近年来，随着神经网络的发展，文本表示方法逐渐倾向于将词映射为连续、稠密的向量的分布式表示。因为向量可以表达更多信息，这种用来表示每个子词的向量被叫作词向量（Word Vector）或称为词嵌入（Word Embedding）。

词向量可以更好的描述词与词之间的关系，体现词语之间的语义关系。在现实生活中词与词之间是有远近关系的，比如狗和猫都属于动物，而桌子这个词却和前两者相去甚远。词向量可以将词与词之间的远近关系映射为向量的距离，从而最大程度的保留单词的原有含义。以图 2.1为例：

图表, 散点图, 气泡图

描述已自动生成

图 2.1 二维词向量示意图

可以看到含义为各种动物、植物和动作的词向量基本分成了三类，类内相对距离较近，类间相对距离较远。上图这种仅使用二维平面进行的可视化的效果可能并不理想，但在实际应用中的高维空间中，这种词向量绝对距离与相对距离可以充分的表达词的含义。

### 分词方法

词（Word）是最小的，能独立使用的音译结合体，是能够独立运用并表达语义或语用内容的最基本单元。进行文本表示，往往要首先对语料库中所有文本进行词切分（Word Segmentation）和去重处理，以各唯一的词构建最终的词典（Vocabulary）。这里得到的词典中的词是用于输入文本表示算法的单元，和自然语言中的词未必相同。字典中一般还会加入一些特殊符号，比如句子的起始符和结束符，未知词标记符，填充符等等，用于满足解决实际问题过程中的需要。之后根据该词典和一定的分词匹配算法，构建分词器（Tokenizer），用于将原始语料转化为文本表示算法的输入。最简单的分词匹配算法叫做正向最大匹配（Forward Maximum Matching，FFM）算法，这是一种贪心算法，其计算过程是从前向后扫描句子中的字符，进可能找到存在于字典中的最长的序列作为分词的结果。

古典分词方法，一般是基于标点符号或语法规则，将原始文本拆分成自然语言中的词汇，并从中选择出现频率最高的部分构成词典。以英语为代表的印欧语系的语言词语之间通常已有分隔符，对其进行切分比较容易。如果仅使用天然的分隔符进行切分不但会造成一定的数据稀疏问题，还会导致词表过大而降低处理速度。但这些语言往往具有复杂的词形变化，比如“computer”、“ computers”、“computing”它们的语义相近却会被认为是完全不同的单词。传统的处理方法是根据人工编写总结的语言规则进行词形还原（Lemmatization）或词干提取（Stemming），但这种方法显然不容易扩展到新的领域或者新的语言上。中文没有复杂的字形变化，但是词语之间没有分隔符，对原始语料句子的切分较为困难，而且中文字与字之间的排列选择可能性极多，词汇量巨大。同样会遇到数据稀疏问题和词表规模问题。古典分词方法还普遍存在无法处理未在词表中出现（Out Of Vocabulary, OOV）的词与词表中的低频词无法得到训练的问题。

针对古典分词方法存在的问题，基于统计的无监督子词（Subword）切分方法应运而生。所谓的子词切分就是将一个单词切分成若干连续的片段，这些片段被称为子词，同时要保证子词尽量长且在语料中出现频次高。以著名的字节对编码（Byte-pair Encoding，BPE）算法[15]为例：

该算法首先会统计语料库内所有单词与频数，然后将原单词拆分到字符（character）级别，根据字符初始化字典。之后统计出现次数最多的字符对，将其插入字典并合并。不断重复合并最高频数字符对和插入字典的步骤，直到字典长度达到预设值。

实际使用BPE算法前，还需要在每个单词的末尾添加“</w>”标记以标识单词边界，避免统计相邻字符对把分别位于两个单词中的字符对算进去。BPE算法的输出将包含字符集与合并规则两部分，分词匹配部分的算法将根据字符集与合并规则执行分词，具体流程与FFM算法类似，也是一种尽可能获取长序列的贪心算法。

另一种常用的字词切分方法是WordPiece算法[16]，该算法会将原始词语切分为词根词缀两部分，粒度介于字符级与词级之间，比如"loves"、“loved”、“loving”这三个单词，会被拆分为“lov”，“ed”,“##ing”,“##es”，其中的"##"符号用于区分词缀。与BPE算法不同的是，WordPiece算法在合并字词时会选择具有最大相邻互信息的相邻字词进行合并，最终得到的输出只有字典，分词匹配算法可以直接采用FFM算法。

BPE与WordPiece算法的字典都是从小到大，属于增量法，与之相反的一种方法是Unigram算法[17]，该算法会先根据语料库中所有序列构建一个尽可能大的词表，然后根据一定的评价准则丢弃部分字词，直到满足字典的预设大小。Unigram算法生成的字典中带有各字词的评价分数，使得其的分词匹配算法能够考虑句子的不同分词可能，并且输出带概率的多个字词片段。

BPE，WordPiece，Unigram等算法都需要事先得到单词才能统计词频，难以中文等语言上直接应用。针对这一问题，Taku Kudo等提出了SentencePiece算法[18]，该算法把一个句子看作一个整体，再拆成片段，因此不需要区分自然语言的词语。SentencePiece算法把空格也当作一种特殊字符来处理，构造词典时则使用BPE或者Unigram算法。

不过，WordPiece算法并非完全不能用于中文。一种可行的方法是先使用中文分词（Chinese Word Segmentation，CWS）工具，将中文文本分割为自然语言中的一个个词，将这些词视作英文的单词，作为WordPiece算法的输入。最终得到的字典粒度为字符级，每一个子词都是一个中文字，出现在原始词语开头或单独成词的字会被视为词根，出现在原始词语中其他字后面的字会被视为词缀。使用这种方法需要高质量的中文分词工具，比如基于多种经典图、树、统计算法的jieba工具库。近年来也出现了一批基于结构化感知器（Structured Perceptron, SP）等深度学习方法的CWS工具。比如清华大学的THULAC（THU Lexical Analyzer for Chinese）工具库[19]，北京大学的pkuseg工具库[20]，哈工大的ltp（Language Technology Platform）工具库[21]。

各种字词切分算法的“词”的概念，已经和自然语言中的词有明显的差别。为了加以区分，这些子词被称为token。本文中涉及到的词这一概念时，一般指的是token。

### 经典语言模型

语言模型（Language Model）是用来描述词、语句乃至于整个文档等语言单元的概率分布的模型，也可以衡量一些已知的语言单元序列是否符合自然语言的分布规律。对语言模型的研究，是自然语言处理领域的基础问题。本章节将介绍几款经典的语言模型。

#### 统计语言模型

统计语言模型（Statistical Language Model）是早期自然语言处理领域的基础模型[22]，也是典型的基于稀疏表示（Sparse Representation）的语言模型。如果把句子看成单词的序列，统计语言模型对应的数学表达可以用概率论的专业术语描述为是：为长度为m的字符串圈定其概率分布其中到依次表示文本中的各个词语。一般采用条件概率链式计算其概率值，如公式（2.1）所示：

|  |  |  |
| --- | --- | --- |
|  |  | （2.1） |

由上式可见，当计算时，需要计算个参数，整体时空复杂度是关于的指数函数。当句子较长时，计算量将会变得很大。因此在统计语言模型中为了减少参数数量，常基于[马尔可夫](https://www.zhihu.com/search?q=%E9%A9%AC%E5%B0%94%E5%8F%AF%E5%A4%AB&search_source=Entity&hybrid_search_source=Entity&hybrid_search_extra=%7B%22sourceType%22%3A%22answer%22%2C%22sourceId%22%3A412730191%7D)假设(Markov Assumption)，认为单词的出现概率仅依赖于前n个词，将其简化为n元(n-gram)统计语言模型。比如时，的时候成为三元模型(tri-gram model)，句子中某单词出现的概率如公式（2.2）所示:

|  |  |  |
| --- | --- | --- |
|  |  | （2.2） |

其中表示序列在语料库中出现的次数。

设为该语言的词汇空间，为词汇量，显然n元模型的空间复杂度是n的指数函数，即,时间复杂度为。作为一种语言的词汇量，一般在几万到几十万之间。很显然，当n比较大时，该模型仍然存在计算量过大的问题。而且即使n很大，该模型依旧不能覆盖所有词与词之间的相关性。

统计语言模型是典型的离散语言模型，存在无法处理新单词或序列的稀疏性问题。比如训练集中不存在的序列会造成上式中分母为0。因此，一般n元模型都会需要配合平滑算法，例如拉普拉斯平滑(Laplace Smoothing)算法，缓解稀疏性问题。如公式（2.3）所示：

|  |  |  |
| --- | --- | --- |
|  |  | （2.3） |

稀疏性问题也表现为，对于训练集中不存在的单词，会由于为0进而导致条件概率为0，无法给出预测。稀疏性问题之外，统计语言模型还存在明显的泛化能力差的问题，不仅依赖于固定单词组合，需要完全的模式匹配，而且无法识别语义相近的句子或词。此外，对于n-gram语言模型来说，马尔可夫假设太强。人在对文字进行处理的时候，能够将很长一段上下文纳入考虑，但是n-gram的离散模型只考虑待预测单词前面的n-1个单词，这个马尔可夫假设与实际情况并不相符。综合来讲，统计语言模型对训练数据集要求很高，不仅需要训练语料和模型应用的领域要一致，而且要在规模尽可能大的基础上降低噪声。IBM 曾进行关于n元语言模型的信息检索性能研究，发现二元语法模型（Bi-gram）和三元语法模型（Tri-gram）分别需要数以亿级和数十亿级别的词汇才能达成饱和，这样大的数据需求限制了统计语言模型的广泛应用。

#### 深度神经网络

神经网络（Neural Networks，NN）又被称为人工神经网络（Artificial Neural Network，ANN），是一种类似生物神经网络的结构和功能的数学模型或计算模型，擅长处理结构规则的多维数据。

##### 多层感知机

最早期最简单的神经网络也被称为多层感知机（Multi-Layer Perception ，MLP）[23]，由感知机（Perceptron）[24]扩展而来。

感知机是一种有若干输入和一个输出的简单分类模型，其结构与生物的神经元高度相似，如图 2.2所示：

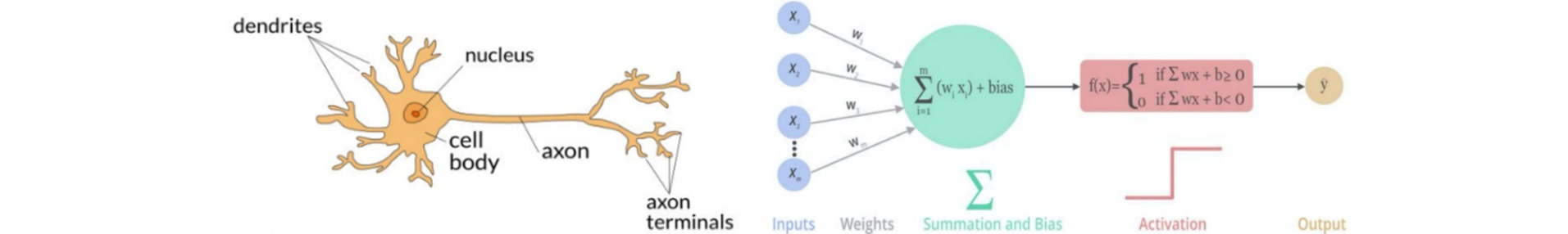


图 2.2 生物神经元的结构与感知机的结构

感知机通过对输入的线性加权得到中间输出。之后对中间使用神经元激活函数得到分类结果-1或1。如公式（2.4）和（2.5）所示，其中为输入个数。

|  |  |  |
| --- | --- | --- |
|  |  | （2.4） |
|  |  | （2.5） |

该模型只能用于二元分类，且无法学习比较复杂的非线性关系，应用范围较为狭窄。比如无法处理有多个候选类的多分类问题。在输出的结果为离散的类别的分类问题外，还有一大类机器学习问题被称为回归问题，其与分类问题的区别在于输出的结果不是离散的类别而是连续的实数值。针对这些缺陷与需求，多层感知机模型通过加入至少一层隐藏层和更复杂的非线性激活函数，提高了模型的学习与表达能力。同时扩展了输出层神经元，使得输出可以有多个维度，这样模型可以灵活的应用于分类，回归，降维和聚类等。如图 2.3的输出层有4个神经元。

形状

中度可信度描述已自动生成

图 2.3 有1个隐藏层和4个输出神经元的多层感知机

计算过程式（2.6）和（2.7）所示,其中为输入层输入，和分别为隐藏层（hidden layer）输出和输出层（output layer）输出，为激活函数，为偏置项（bias），为权重矩阵，该矩阵行数为本层输入维度，列数为本层输出维度。

|  |  |  |
| --- | --- | --- |
|  |  | （2.6） |
|  |  | （2.7） |

由于其中的操作相当于对输入进行线性变换（Linear Transform），因此这样的层可以通称为线性层（Linear Layer）。又由于每层输入的各元素都需要乘以独立的参数，相当于本层每一个节点都与相邻层的所有节点有连接，因此这样的层又被称为全连接层（Fully Connected Layer）或稠密层（Dense Layer）。

神经网络在发展到一定阶度后，模型越来越庞大，结构也越来越复杂，因此人们将其命名为深度神经网络（Deep Neural Networks，DNN）。图 2.4就是一个有3个隐藏层的深度神经网络，实际应用中的神经网络层数还要远多于此。

形状

中度可信度描述已自动生成

图 2.4 有3个隐藏层的神经网络

##### 非线性激活函数

通过对上文公式（2.6）的简单展开可以发现，如果不引入非线性激活函数，仅仅堆叠线性变换的结果最终还是相当于只有一次线性变换，并不能提升模型的复杂度，增强拟合能力。以经典的S形（sigmoid）激活函数（也称logistic函数为例），其函数表达式与导函数表达式如公式（2.8）和（2.9）所示，图像如图 2.5所示。

|  |  |  |
| --- | --- | --- |
|  |  | （2.8） |
|  |  | （2.9） |

sigmoid函数是深度学习领域的经典激活函数，可以把连续的实数值映射为0到1区间的实数。其输出值适合作为概率使用，常用于二分类任务的输出层。由于sigmoid函数的输出在输入值的绝对值比较大时几乎是一条直线，在反向传播时会造成梯度消失；输出值不以零为中心且恒大于零，可能导致模型收敛困难，该现象被称为zigzag现象；包含幂运算导致计算量很大等缺点，目前已经很少用于深度神经网络的隐藏层。

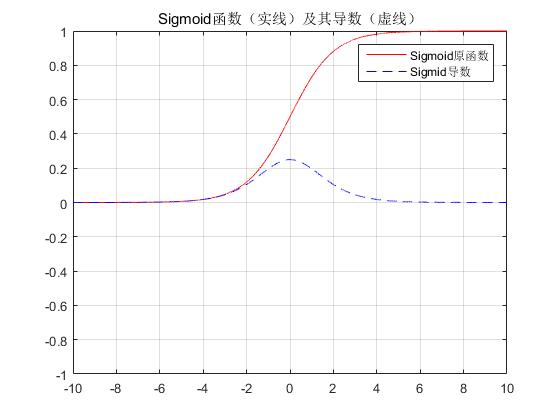


图 2.5 sigmoid函数及其导函数的图像

非线性的激活函数是当前神经网络的不可缺少的部分，随着近年来相关研究的深入，越来越多的激活函数被提出。然而，并没有完整的证据表明如何针对具体应用选择合适的激活函数。本文之后写作中，只对涉及到的具体的激活函数给出详细的介绍。

##### 梯度消失与爆炸

由于深度神经网络是多层非线性函数的堆砌，整个深度网络可以视为是一个复合的非线性多元函数（这些非线性多元函数其实就是每层的激活函数），那么对损失函数（Loss function）求不同层的权值偏导，相当于应用梯度下降的链式法则，链式法则是一个连乘的形式，所以当层数越深的时候，梯度将以指数传播。以一个典型的深度神经网络为例，假设该网络输入层记作式（2.10）：

|  |  |  |
| --- | --- | --- |
|  |  | （2.10） |

中间的多个隐藏层概括为式（2.11）：

|  |  |  |
| --- | --- | --- |
|  |  | （2.11） |

输出层记作式（2.12）：

|  |  |  |
| --- | --- | --- |
|  |  | （2.12） |

则损失为式（2.13）：

|  |  |  |
| --- | --- | --- |
|  |  | （2.13） |

设现在需对输入层进行参数更新，则其后向传播过程为式（2.14）：

|  |  |  |
| --- | --- | --- |
|  |  | （2.14） |

展开为式（2.15）：

|  |  |  |
| --- | --- | --- |
|  |  | （2.15） |

在上式的连乘项中，作为激活函数的导数值，一般大于0且小于1，连乘会使得最终结果越来越趋近于0。当也比较小时，最终梯度会趋近于0，造成模型中靠前的层参数更新缓慢，这种现象叫梯度消失。当，连乘结果也有可能变得很大，造成模型权重大幅抖动，这种现象叫梯度爆炸。梯度爆炸一般出现在模型参数初始化不正确，或者参数分布极度不均匀的情况下。而梯度消失则比较普遍，几乎所有深度学习模型都需要应用一些方法来缓解梯度消失问题。

#### 神经网络语言模型

本章节的神经网络语言模型（Neural Network Language Model，NNLM）专指2003年Bengio Yoshua等[25]发表的神经概率语言模型（Neural Probabilistic Language Model），该模型整体上是一个三层前馈神经网络，包含一个嵌入层、一个隐藏层、一个输出层。同时该模型也是n-gram模型，即基于给定的n-1个上文，预测第n个单词出现的概率。该模型结构如图 2.6所示：

图示

描述已自动生成

图 2.6 神经概率语言模型的结构

该模型首先会将每个输入单词转换为One-Hot编码形式，之后到一个给定维度的致密空间，具体实现方法是使用各单词的One-Hot编码结果乘以如图所示的映射矩阵，该矩阵行数等于词典中的单词数量，列数等于给定的致密空间的维度，得到各单词新的向量表示。通过以上的映射，神经网络语言模型能解决稀疏性的问题。对于一些在训练集中没有遇到过单词，由于其与上文在映射结果的空间中会与相类似的单词处于相近的位置，可以降低出现接近于0的条件概率导致无法预测的问题。

设输入的n-1个词经过处理后得到的向量表示组成的矩阵为，，为隐藏层参数矩阵，为隐藏层偏置，为输出层参数矩阵，为输出层偏置，该模型前馈网络部分的运算最终计算公式如（2.16）所示：

|  |  |  |
| --- | --- | --- |
|  |  | （2.16） |

其中表示下一个位置的单词的概率分布，其对应的真实概率分步是一个One-Hot向量，在正确单词的位置上的值为1，其它位置的值为0。的维度为，其中最大值出现位置对应的单词即为预测结果。

该模型在实际应用过程中虽然表现出了一定的泛化能力，但是没有明确地对超出观察窗口的上下文信息进行处理。不过其优秀表现使得越来越多人开始意识到神经网络在构建语言模型方面的威力，使得深度学习方法成为了学界主流。

#### 词向量层

NNLM模型将稀疏的One-Hot编码矩阵乘以映射矩阵得到词的稠密向量表示的方法，启发了后世关于词向量的研究。映射矩阵等价于一个去掉偏置项仅保留权重矩阵的全连接层，而One-Hot编码向量与映射矩阵相乘的操作等价于根据One-Hot编码向量中数值1所在位置，取出中对应位置的行向量。综合这些性质，研究者设计了专门用于将词映射到词向量的词向量层（Embedding Layer）。

词向量层只有一个可学习参数矩阵，维度为。其中为词典维度，为预设的词向量维度。中每一行都对应着在词典中索引为的词的词向量，使用该词的独热编码向量与相乘即可取出。词向量层的一种优化实现，去掉了生成与保存庞大且稀疏的One-Hot编码矩阵并执行矩阵乘法的步骤，直接以文本中各词的索引作为输入，在中查表（Table look-up）即可。

#### 卷积神经网络语言模型

##### 卷积与池化

在图像识别任务中，如果对每个像素都分配独立的参数会导致模型高度依赖物体的位置和大小，因此在待识别物体移动或缩放时，模型的输出会发生较大变化。在自然语言处理任务中，也存在仅使用全连接层的模型往往无法有效地捕捉到关键的局部信息的问题。比如在情感分析任务中，句子的情感往往由零星的关键词汇或短语决定。为了解决以上问题，一个非常简单的办法就是使用一个小的全连接层提取这些局部信息，如图中固定大小的像素区域或文本中的个别词等。

为了解决这些关键信息位置不同的问题，可以依次扫描输入的每个区域的操作被称为卷积（Convolution）[26]。其中每个小的用于提取局部特征的全连接层又被称为卷积核（Convolution kernel）或过滤器（Filter）。卷积神经网络认为信息具有空间局部性（locality）和空间不变性 (Spatial Invariance)或平移等效性 (Translation Equivariance)，可以用滑动卷积共享权重方式降低参数空间。这一系列对现实生活中观察到的现象进行总结，得出一定的规则(Heuristics)，然后对模型做一定的优化设计的流程，被称为模型的归纳偏置（Inductive Bias）或归纳性偏好。

卷积操作的输出结果还可以进一步聚合，这一过程被称为池化（Pooling）[26]操作。常用的池化有最大池化（Max-pooling），平均池化（Mean-pooling）等。以最大池化为例，其目的是保留最有意义的局部特征。池化操作还可以解决输入样本大小不一致的问题，不同大小的图片或者不同长字的句子包含的词数量并不相同，导致抽取的局部特征个数也不相同，然而经过池化操作后可以保证最终输入的个数相同。

如果仅使用一个卷积核，则只能提取单一的局部特征。在实际问题中往往需要提取多种局部特征，因此在卷积操作时可以使用多个卷积核，分别提取不同局部特征，这一操作被称为多通道卷积（Multi-channel convolution）。

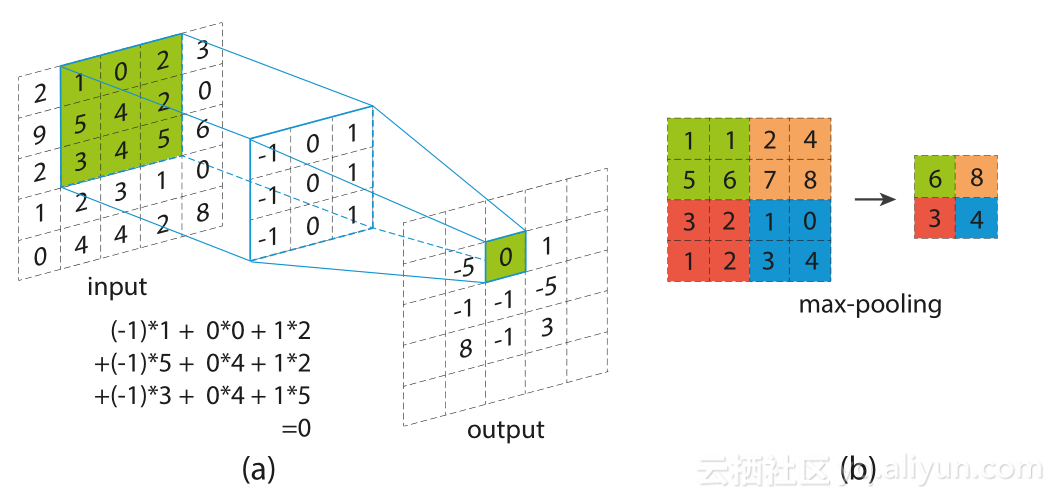


图 2.7 卷积（左）与池化（右）

通过对多个卷积层和池化层的堆叠形成更深的网络，这类网络统称为卷积神经网络。与前文提到的多层感知机模型，类似卷积神经网络中的信息也是从输入层经过隐含层最后传递给输出层，按照一个方向流动，因此它们都被称为前馈神经网络（Feedforward Neural Network，FNN）。

使用卷积神经网络解决自然语言处理问题，几乎相当于是把文本当成一种图片来处理。与自然语言作为一种序列模型的归纳并不符合。同时考虑到自然语言的信息密度较高，池化操作造成的信息损失较为明显。因此卷积神经网络在自然语言处理领域的主要应用是作为特征的提取与降维工具，除非是处理超长文本的情况，一般不作为语言模型的主干网络使用。

#### 循环神经网络语言模型

##### 循环神经网络

上文提到的NNLM与n-gram统计语言模型都假设当前词仅依赖前n个词，因此要求输入序列长度一致，都无法解决长期依赖问题。同时，自然语言的时序信息应当被一同考虑进来，强调顺序重要性。基于以上归纳，研究者提出了用来解决序列问题的序列模型中经典的潜变量自回归模型，将之前所有词的信息总结为潜变量,使得。潜变量自回归模型的一种典型实现方法循环神经网络(Recurrent Neural Networks, RNN)，如图 2.8所示：

图示

描述已自动生成

图 2.8 循环神经网络展开后的结构

图中h、x、y分别代表RNN神经元的输入、潜变量、输出。U、W、V是对向量x、h、y进行线性变换的可学习参数矩阵。这些参数矩阵在所有时刻的计算中共享，因此RNN中的参数量不会随着输入序列的长度增长。表示t时刻的输入向量(例如第t个单词的词向量)。表示t时刻的潜变量(包含了从开始一直到t时刻的相关信息)。表示t时刻的输出向量。潜变量的更新过程符合公式（2.17），其中为偏置项，为非线性激活函数：

|  |  |  |
| --- | --- | --- |
|  |  | （2.17） |

当使用RNN构建语言模型时，其输出如式（2.18）所示：

|  |  |  |
| --- | --- | --- |
|  |  | （2.18） |

其中表示下一个位置的单词的概率分布,为偏置项。

与前文提到的多层感知机和卷积神经网络不同，RNN中的信息循环流动，并不是前馈神经网络。不过将其展开后相当于堆叠多个共享隐含层参数的前馈神经网络。每个时刻的隐藏层承载了从开始到t时刻的全部输入信息，因此也被称为成为记忆单元。RNN的输出的计算是基于当前隐藏层状态的，而隐藏层状态在每个时间步都会更新，其更新是基于当前时刻t的输入向量以及上一时刻的隐藏层状态计算得到的。因此，每个隐藏状态理论上都包含了之前所有时刻的信息，这使得RNN能够捕捉序列中的长期依赖关系。

##### RNN的梯度消失与梯度爆炸

RNN也存在梯度消失和梯度爆炸现象，原因是其反向传播使用的是随时间反向传播(back propagation through time, BPTT)方法。以损失loss对参数求梯度为例，梯度等于loss在各时间步对求导之和，同时各时间步之间需要依次求偏导再对参数矩阵求偏导，最终求解过程中出现激活函数导数和参数矩阵的多次连乘。用公式表示如（2.19）所示：

|  |  |  |
| --- | --- | --- |
|  |  | （2.19） |

在上式的一系列连加项中取其中一项的部分展开如式（2.20）所示：

|  |  |  |
| --- | --- | --- |
|  |  | （2.20） |

RNN中常用的激活函数的导数始终在(0,1]范围内，受值大小的影响，在该段序列内各连乘项的值可能始终大于1，或者始终在[0,1]范围内，导致梯度消失或梯度爆炸现象的出现。很显然,在式（2.20）中越是长的连乘项出现梯度消失或梯度爆炸现象的可能性就越大，反映在式（2.19）中就是时间步跨度较远的子项相比于较近的子项更容易出现问题。

RNN在出现梯度爆炸时的表现与深度神经网络类似。但RNN的梯度消失导致的结果是较早的输入以及其对应的潜变量对当前的参数的更新的影响将会变得极小，使模型难以学习到较早的信息，因此该现象也被形象地称为“遗忘”。

改善RNN的梯度消失和梯度爆炸问题，一种经典的方法是采用导数值域更广的双曲正切（hyperbolic tangent，tanh）函数。其函数表达式与导函数表达式如公式（2.21）和（2.22）所示，图像图 2.9如所示：

|  |  |  |
| --- | --- | --- |
|  |  | （2.21） |
|  |  | （2.22） |

tanh函数是sigmoid函数的改进，其输出值均值为0，避免了zigzag现象。导数值的取值范围也更大有利于缓解梯度消失，但并不能从根本上解决。而且tanh函数同样存在幂运算导致计算量很大的缺点。

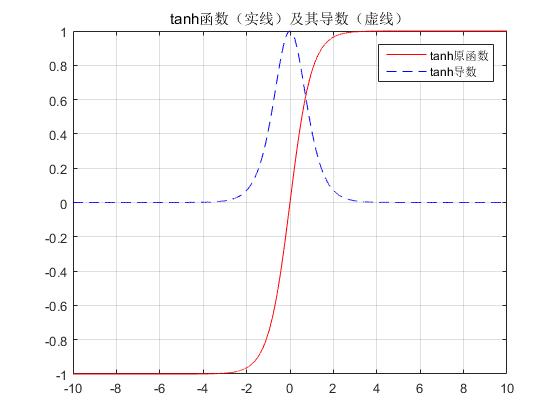


图 2.9 tanh函数及其导函数图像

##### 改进型RNN

RNN有多种可以大幅缓解梯度消失和梯度爆炸的改进型，比如长短期记忆（Long short term memory，LSTM）[27]网络与门控制单元GRU（Gate Recurrent Unit）[28]。

LSTM通过加入输入门（Input Gate）、输出门（Output Gate)、遗忘门（Forget Gate），使得求导后的连乘项中的每一项都可训练学习，避免其全体同时出现在之间或全部大于1，进而缓解了梯度消失和梯度爆炸。

GRU是LSTM的一种简化变体，其内部只有重置门（Reset Gate）和更新门（Update Gate），在实现了与LSTM相当的效果的同时，参数量更少，更容易训练。

图示

描述已自动生成

图 2.10 LSTM与GRU的结构

##### 双向RNN

无论是RNN还是LSTM、GRU等，信息流动都是单向的。在一些任务中，我们希望模型不仅能记住之前的输入，还能理解整个句子的上下文。对此提出了一种RNN改进型，被称为双向循环神经网络（Bidirectional RNN，BiRNN）。BiRNN可以被视作是一个有两套参数的RNN。在训练或推理时，先将输入序列正序输入并使用正向参数进行计算。整个正向输入与计算过程完成后，将输入序列反转，并使用反向参数再次进行计算，最终得到两组输出。通过整合这两组输出作为的后续模型的输入，即可实现对整个句子上下文的综合学习。

### 预训练词向量语言模型

模型预训练，即首先在一个替代任务上预先训练一个初始模型，然后在下游任务（也称目标任务）上继续对该模型进行微调，从而达到提升下游任务准确率的目的。在本质上，这也是迁移学习（Transfer Learning）思想的一种应用。比如计算机视觉（computer vision，CV）领域著名的VGG-16模型[29]，在用于新的具体任务时，只需要下载预先在大规模数据及训练好的大部分权重，并在具体任务的有标注数据集上微调并训练输出层即可。然而VGG-16的预训练阶段同样需要大量人工标注的数据，导致替代任务数据的规模往往也非常有限

在NLP领域，文本的顺序性就是一种天然的标注数据，通过若干连续出现的词预测下一个词本身就能构成一项替代任务。由于现成的图书、网络等文本数据的规模几乎无限，因此可以非常容易的获得超大规模的预训练数据，有人将这种不需要人工标注的预训练学习方式称为无监督学习（Unsupervised learning）。实际上这是并不准确的，因为学习的过程仍是有监督的，只不过监督信号来自输入自身，所以更准确的做法应该叫自监督学习（Self-supervised learning）。

#### 静态词向量语言模型

传统的基于神经网络的语言模型，只重视利用历史词的信息作为输入，而失去了与“未来”上下文之间的贡献信息。针对这一问题，Mikolov等人[3]提出了一类效率更高，表达能力更强的模型——word2vec，其中包括两个模型：连续词袋模型(Continuous Bag of Words，CBOW)和跳字模型(Continuous Skip-gram Model，Skip-gram)。该系列模型完全基于词与词之间的共现信息实现词向量的学习，在学术界和工业界被广泛应用。

CBOW模型的思路是根据上下文对目标词进行预测。对于一段文本。CBOW的任务是根据给定长度的窗口内的上下文，对 时刻的词进行预测。如下图所示，窗口大小为5时。。该模型的输入层将中各词转化为独热编码表示。映射层是一个词向量层，将中各词的独热编码转化为词向量，其后没有非线性激活函数。输出层先将中各词的词向量取平均得到的上下文表示，再根据对目标词进行预测，除了没有偏置项外，其他方面与NNLM的输出层的基本一致。

很显然，CBOW模型不考虑中各单词的位置或顺序信息，其输入仅是一个“词袋”而非序列，这就是其名字“词袋模型”的来历。

CBOW模型的输出参数矩阵与输出层参数矩阵都可以作为下游任务的词向量矩阵使用，他们分别描述了词表中的词在作为条件上下文或目标词时的不同性质。在实际应用中，通常只使用E就能满足需求，但在一些任务中，对二者进行组合得到的向量可能会有更好的表现。

Skip-gram模型则是根据当前词预测其上下文。该模型的输入层与映射层与CBOW相同，目的是将转化为词向量。输出层根据对上下文窗口内的词进行独立的预测，同样没有偏置项。

与CBOW模型类似，Skip-gram模型的参数矩阵和都可以作为词向量矩阵使用。

图示

描述已自动生成

图 2.11 CBOW与Skip-gram模型结构

在训练时，word2vec模型还引入了两种高效的加速技巧。第一种是基于哈夫曼树的层次softmax（hierarchical softmax），可以将分类复杂度从降低到。另一种是负采样（Negative Sample），将Skip-gram任务转化为预测与1个窗口内正样本和个窗口外负样本是否共现的个二分类问题，每轮学习只更新这组参数，而不是像原版任务那样每次都更新组参数。

word2vec模型一个明显的缺点是只能利用一定窗口内的上下文信息，即局部信息，而不能利用整个语料库的全局信息。针对这一点，斯坦福大学的研究者提出了基于全局“词-上下文”共现矩阵的GloVe模型[30]，通过对全局共现矩阵的回归预测训练词向量。

word2vec模型或GloVe模型训练得到的词向量，可以直接保存下来，通过对下游任务模型的词向量层的初始化过程简单修改，就能实现对已经训练好的词向量的利用。既可以直接用于下游任务，也可以作为模型的参数在下游任务的训练过程中进行微调。合理使用预训练词向量，可以大幅缩短模型的训练过程，提升模型的泛化能力。

#### 动态词向量语言模型

无论是word2vec模型还是GloVe模型，其训练得到的词向量可以被认为是静态的。即对于给定的语料库中任意一个词，得到的向量表示是稳定的，不随其上下文变化而变化。这种静态词向量难以刻画一个词在不同上下文或不同语境中不同的语义信息，而一词多义在自然语言中又是普遍存在的现象。为了解决这一问题，研究者提出了上下文相关词向量（Contextualized Word Embedding），又称动态词向量（Dynamic Word Embedding）。顾名思义，在这种表示方法中，一个词的词向量将结合其自身语义和所处的上下文计算获得，因此是随着上下文动态变化的。

RNN恰好提供了这样一种对词的上下文信息以及自身信息进行有效的语义结合方式。在RNN中，每个时间步的隐藏层恰好可以作为当前词在当前上下文条件下的向量表示，即动态词向量。同时，RNN又正好可以根据文本作为序列数据的特点，通过语言模型任务进行自监督学习，而不需要额外的人工标注数据。基于该思想以及对相关研究的总结完善，Peters等[3]提出了深度上下文相关词向量的思想，以及预训练模型ELMo（Embeddings from Language Models）。

在包括自动问答、文本蕴涵和信息抽取等多项自然语言处理任务上的实验表明，采用ELMo生成动态词向量能够直接有效地为当时最好的模型带来显著的提升。同时ELMo模型还被推广到多语言场景，在CoNLL-2018国际多语言通用依存句法分析的评测任务中取得了优异的表现。

图示

描述已自动生成

图 2.12 ELMo模型结构

ELMo模型的输入部分采用基于字符的向量化（embedding）层，目的是为了尽可能减少未登录词（Out-Of-Vocabulary，OOV）的影响。首先由分词器将输入句子拆分为单词，再将各单词拆分为字符，期间还会加入<bow>（单词的开始）、<eow>（单词的结束）、 <bos>（句子的开始）、<eos>（句子的结束）、<pow>（单词补齐符）和<pos>（句子补齐符）等特殊字符。之后由字符向量层将个字符转化为稠密的向量表示。

接下来利用卷积神经网络对字符向量表示序列进行语义组合（Semantic Composition）。具体是使用一维卷积神经网络，以字符向量的维度作为输入通道数，使用多个不同宽度的卷积核进行多分支（Multi-branch）多通道卷积，获取不同粒度的字符级上下文信息。之后通过最大池化整合各卷积结果，并拼接起来，最终得到该单词的定长向量表示，向量维度为总的卷积核通道数。

图示, 表格

描述已自动生成

图 2.13 ELMo中对字符向量的多分支多通道卷积

之后，模型使用两层Highway神经网络对卷积神经网络的输出作进一步线性变换，将其映射到预设的维度，得到单词的最终向量表示。Highway神经网络可以在输入与输出之间直接建立“通道”，使得输出层可以直接将梯度回传到输入层，从而缓解因为层数过多导致的梯度爆炸或梯度消失。目前Highway神经网络已经被更高效的残差网络取代，本文也不再赘述。

在得到上下文无关词向量表示后，ELMo模型采用多层Bi-LSTM实现的双向语言模型分别编码前向与后向上下文信息，从而得到每一个时刻的动态词向量表示。最简单的做法是使用两组LSTM的最后一层隐藏层的输出作为词的动态向量表示。但是在ELMo中，不同隐藏层的向量蕴含了不同层次或者不同粒度的文本信息。例如，越接近输入层的LSTM隐藏层通常编码了更多的词法、句法信息，而靠近输出层的LSTM隐藏层则更偏重语义信息。对于不同的下游任务，对词的表示的信息粒度有所不同。因此，ELMo采用了对不同层次的向量进行加权平均的机制。权重可以在下游任务的训练中学习，为不同的下游任务提供更高的组合自由度。

以ELMo为代表的动态词向量模型能够弥补静态词向量对于一词多义现象的表达能力不足的问题，还能够提供一定的词义消歧能力，提供更丰富更灵活的语义信息。动态词向量也可以直接作为输入特征供目标任务模型使用，无需改变任务模型已有的结构，这种“即插即用”的特点也是动态预训练词向量模型广受欢迎的原因之一。

但是，动态词向量模型在应用时，每次都需要完整执行整个模型的推理流程，实时获取输出，而不是像静态词向量那样可以直接查表。这就使得动态词向量模型在计算资源的消耗上要明显更高，尤其是在结合下游任务进行微调时。因此在实际应用时，要结合具体任务的需求选择合适的词向量化方法。

### 文本分类问题

文本分类（text classification或者text categorization）是最简单也最基础的自然语言处理问题。即针对一段文本输入，输出该文本所属的类别，其中类别是事先定义好的一个封闭的集合。根据一条输入文本可以对应的类别标签数量，可以分为单标签（single-label）文本分类与多标签（multi-label）文本分类两种任务。根据所有备选类别标签的总数，又可以分为二分类任务与多分类任务。根据输入文本的预期长度，也可以分为长文本分类与短文本分类任务。如果预设类别间存在分层包含关系，也可区分出层次文本分类（Hierarchical text classification，HTC）任务。文本分类有许多应用场景，比如垃圾邮件分类：将邮件分为垃圾和非垃圾两类；新闻分类：将新闻分为政治经济体育娱乐等类别；或者文本情感分类任务：识别文本中蕴含的情感类型以及强度。

解决一个文本分类问题的流程大致如图 2.14所示：

图示, 示意图

描述已自动生成

图 2.14 文本分类流程

文本预处理：数据集中的原始语料可能包含各种无意义数据或者损坏，比如html标签或者多余的标点符号，需要进行针对性清洗。除此之外，还可以采用去除停用词（Removing stop words）之类的方法实现数据增强。停用词指语言中一类没有多少实际语义，信息密度较低的词语，比如中文的“的”，“吗”，“吧”，英文的“the”，“an”等。从预料中去掉这些词语有利于节省计算资源，凸显那些对语义有决定性意义的词。但是去标点符号与停用词的合理性与有效性存在争议，应当具体问题具体分析。

文本表示：该步骤的任务使用类似前文介绍的文本表示技术，是将输入的一段文本转化为特征向量。在文本分类任务中，文本表示也被称为文本的特征提取（Feature extraction）。特征提取得到的特征向量不仅仅是对文本中所有词的合理表示，更强调综合考虑整段文本的内容，充分体现该段文本的语义信息。

特征降维：文本特征提取得到的特征可能不全是有效特征，其中可能包含一些冗余或无关的信息，部分特征之间可能存在过强的相关性。这些问题可能会对算法的学习产生一定的干扰，执行特征降维（Feature dimensionality reduction）有利于提取有效信息，同时提高计算效率。在文本分类任务中，一般在处理特征向量维度较多或处理长文本时执行特征降维。

分类模型：分类模型，又称分类器，负责从特征向量到预设分类标签的映射，其核心在于采用的分类算法。在传统机器学习阶段比较常用的有朴素贝叶斯 (Naive Bayes，NB)[31]、随机森林 (Random Forest，RF)[32]、支持向量机Support Vector Machine，SVM)[33]、决策树(Decision Tree，DT)[34]等模型。在深度学习阶段，则有基于深度神经网络、卷积神经网络、循环神经网络，以及注意力机制的多种模型。

评价：通过统计分类模型的输出结果与输入文本在数据集中的标签，计算一系列量化指标，对整个模型的分类效果进行评价。比较常用的指标有精度（Accuracy）、准确率（Precision）、召回率（Recall）和F1分数（F1 score）等。

### 经典文本分类模型

#### FastText

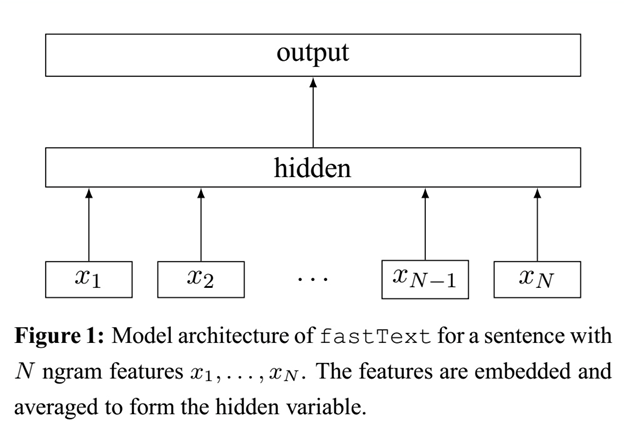


图 2.15 FastText模型结构

FastText[35]是一款非常高效的文本分类模型，其结构和CBOW有诸多相似之处，二者最大的不同是FastText使用了字符级别的n-gram向量来表示一个单词。以n等于3时的单词“apple”为例，它的trigram有“<ap”, “app”, “ppl”, “ple”, “le>”。其中，<表示前缀，>表示后缀。在隐藏层，FastText将5个trigram的向量取平均来表示“apple”的词向量。这样做一方面可以使得低频词也得到足够的训练，因为它的n-gram与其他单词共享，另一方面是对于未登录词仍然可以通过组合n-gram向量构建其词向量。在输出层，FastText也采用了层次softmax，与CBOW不同的是，此处预测的目标是分类标签。

FastText如同其名字那样，是一款非常快速的分类模型。简单的结构和较小的参数量使其对计算资源的需求很少。而且FastText的n-gram向量层也可以采用预训练词向量，使其只需要很短的时间就能完成训练。直到今天，FastText仍是处理长文本分类任务，或者时间敏感资源敏感型任务最常用模型之一。

#### TextCNN

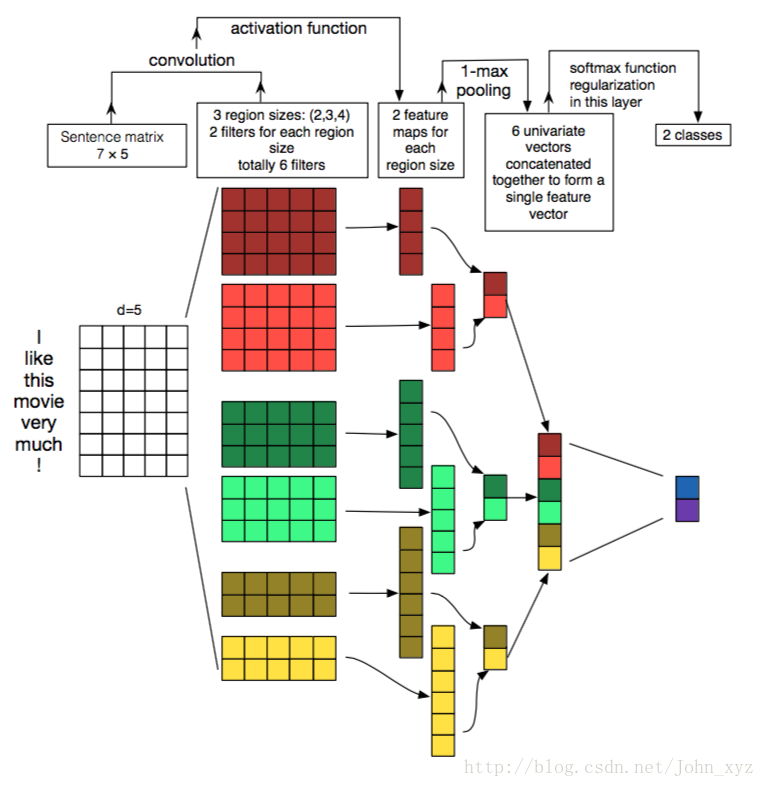


图 2.16 TextCNN模型结构

TextCNN[36]是卷积神经网络用在文本处理方面的知名模型，通过多分支卷积技术实现了对文本的分类功能。该模型的多分支卷积层的各分支会使用不同大小的卷积核对句子分别进行多通道卷积操作，形成大小不同的特征数据。之后的多分支最大池化层会分别对各分支的特征数据进行最大池化处理，之后由全连接层输出。该模型结构灵活，拥有强大的特征提取与降维能力，同样是处理长文本分类最常用的模型之一。由于该模型中使用了池化操作，在这一过程中会丢失一些信息，导致该模型所能表征的句子特征有限，在处理语义相近的文本时需要对其进行进一步调整。

#### TextRNN

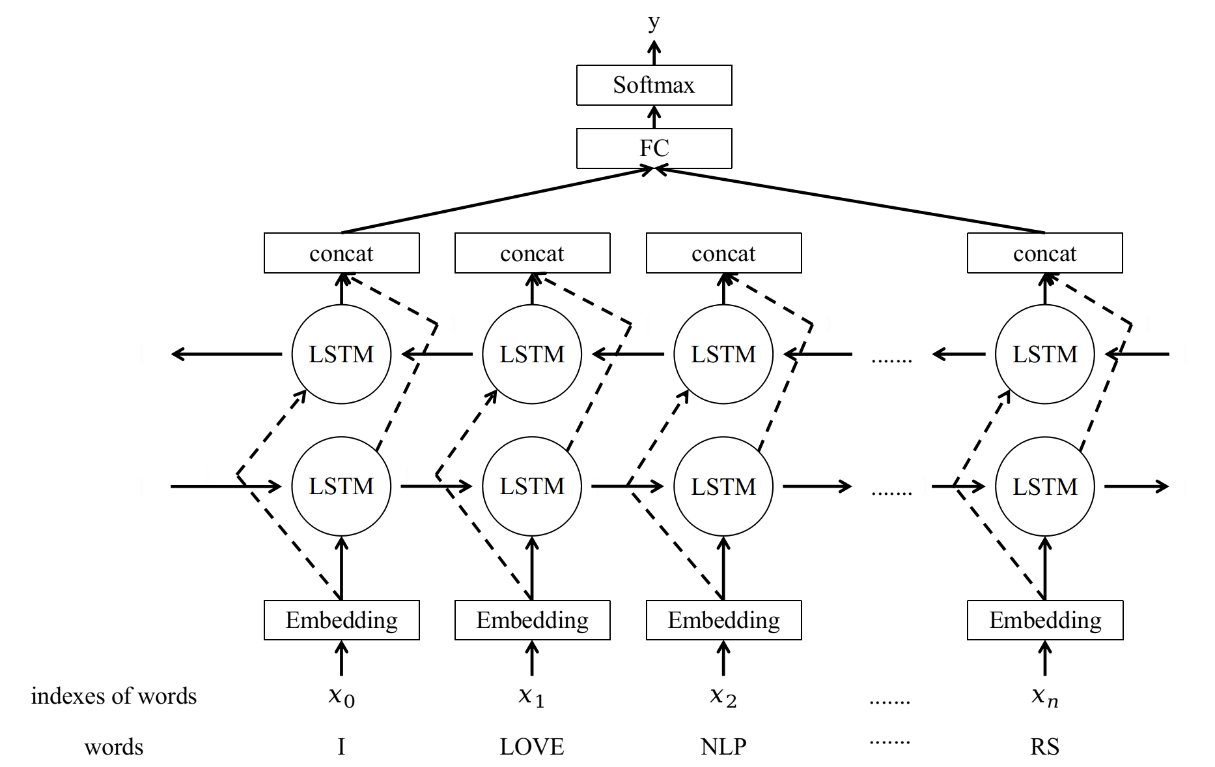


图 2.17 TextRNN模型结构

著名的TextRNN[37]模型是一款简单的模型，使用BiRNN的一种改进型BiLSTM为主干进行搭建。该模型将各时间步步的正向输出与反向输出进行拼接，分别取输入序列的头尾两个输出，作为句子的最终特征，再次拼接后输入全连接层分类器，得到分类结果。

## 本文研究相关理论

### Transformer模型

Transformer模型是[5]Google的研究者在2017年提出的一款机器翻译（Machine Translation）模型，其突破性地以自注意力（Self-Attention）机制为核心对文本进行建模，突破了以往序列模型与RNN绑定的惯例，并且在实验中全面击败了当时的SOTA（State of the arts，最佳）模型。

BERT模型的基本结构由多层Transformer Encoder Layer堆叠组成，其核心算法自注意力（self-attention）机制也与Transformer模型相同，区别仅限于部分涉及模型规模的超参数。理解了Transformer，就理解了大部分BERT模型结构的相关知识。本章节将详细解析Transformer模型，此基础上指出BERT的模型的改进。

#### 传统序列到序列模型

为解决以机器翻译为代表的序列到序列（sequence to sequence，seq2seq）任务而设计的模型，一般采用编码器-解码器（Encoder-Decoder）结构，如下图所示。

其中的编码器和解码器一般都采用RNN（GRU，LSTM）实现。Encoder-Decoder结构的核心思想是将输入序列通过Encoder转换成一个定长的语义特征c，c中包含了原始序列中的所有信息，再通过Decoder将c恢复为目标序列。得到c的方法有很多种，最直接的方法就是把Encoder最后的隐藏层状态赋值给c，也可以对最后的隐藏层状态做一个线性变换得到c，还可以综合所有的隐藏层状态通过加权求和得到c。

图片包含 游戏机, 物体, 钟表

描述已自动生成

图 3.1 经典seq2seq模型结构

当输入句子较长时，c的中包含的信息量就成了限制模型性能的瓶颈。除此之外，这种实现方式只用到了编码器的最后一个隐藏层状态，信息利用率低下。后来的研究者在原始seq2seq模型中加入了注意力（Attention）机制，使得Decoder能更好地利用Encoder的信息。以著名的Luong Attention[38]为例，改进后的Decoder会利用Encoder在编码过程中所有时间步的信息，根据可学习权重矩阵A，在解码的不同阶段生成侧重点不同的语义信息。

手机屏幕截图

描述已自动生成

图 3.2 加入了Luong Attention的seq2seq模型中的生成与Decoder的输入

各种注意力机制的加入，大幅提升了seq2seq模型的表现。

#### Transformer模型的突破

Transformer团队在总结前人工作的基础上，认为可以只保留有着优秀表现的注意力部分，尝试不使用RNN进行seq2seq建模，并且取得了惊人的突破。论文题目《Attention is all you need》也由此而来。

Transformer模型也采用了经典的Encoder-Decoder结构。分别由Transformer Encoder Layer与Transformer Decoder Layer堆叠而成。具体结构见图 3.3：

Transformer最大的贡献是突破了传统RNN需要逐步递归训练缓慢的问题，指出了一条可以高度并行的模型发展方向，使得参数量远超以往的大模型（Large Model）成为了可能，启发了后世大量工作，从某种程度上甚至开启了一个新的时代。

#### Transformer Encoder详解

##### 多头注意力层

###### 自注意力机制

我们在看到一张图片时，大脑会把注意力放在主要的特征上。同样，当我们读一句话时，大脑也会首先记住重要的词汇分词。研究者通过总结人脑处理信息过载的方式，提出了注意力（Attention）机制。注意力机制是一种用于解决序列数据中对特定部分进行加权关注的方法，其目的是从大量信息中有筛选出少量重要信息，并聚焦到这些重要信息上，忽略大多不重要的信息。

图示

描述已自动生成

图 3.3 Transformer模型结构

对于目前大多数注意力机制的实现方法而言，它可以被归纳为两个步骤：第一步是根据输入的Query和Key计算权重系数，第二步是根据这些权重系数对Value进行加权求和。第一步可以进一步细分为两个阶段。第一个阶段是基于Query和Key计算它们之间的相似性或相关性，这个过程通常是通过向量点积或其他可学习的相似度度量来完成的。第二个阶段是对第一阶段得到的原始相似度分值进行归一化处理，转化成权重。这个归一化的过程可以使用不同的函数，例如softmax函数或sigmoid函数，以确保权重总和为1或者范围在0到1之间。权重越大，就越聚焦于对应的Value值上，即权重代表了信息的重要性，而Value则是对应的信息本身。

注意力机制由于其良好的效果和灵活性，在各种自然语言处理任务中都得到了广泛的应用，如机器翻译，问答系统和文本分类等。Transformer中的自注意力机制（Self -attention）机制的self体现在于，其运算过程中的、、矩阵是同源的且都来自于输入。相比之下的其他注意力机制，比如交叉注意力机制，其、、矩阵就分别来自不同的运算结果。

根据注意力权重的计算方法不同，常见的注意力模型有加性模型（Additive attention），点积模型（Dot-Product Attention）等等。具体到Transformer使用注意力计算方法，被称为放缩点积模型（Scaled Dot-Product Attention），可以由以下公式表达：

|  |  |  |
| --- | --- | --- |
|  |  | （3.1） |

式中、、三个矩阵由输入矩阵分别乘以可学习参数矩阵、、得到。输入矩阵由输入序列中的所有词向量组成，维度为，其中为输入序列长度，为词向量维度；矩阵、、的维度分别为、、，其中为同样词向量维度；；没有数学上的硬性要求，一般为了保持输入与输出维度相同，会采用和、相同的数值。在Transformer中，，，统称为。BERT模型中称为，在基础模型中。

电脑屏幕的照片

低可信度描述已自动生成

图 3.4 Transformer中放缩点积注意力的运算过程

公式中引入的三个可学习参数矩阵、、可以增强模型的拟合能力。的结果除以的目的是进行放缩（Scaling），减小方差使得结果中各元素的值落在softmax函数比较敏感的区间。这一数值是的大量实验总结的比较合理的结果。以的情况为例，假设Q,K矩阵中各个向量互相独立，则计算结果的方差会扩大倍，即，除以正好可以使计算结果的方差回到，实现方差与参数矩阵维度的解耦，使得训练过程中梯度值保持稳定。

###### 自注意力机制的优点

表 3.1 自注意力机制与循环神经网络，卷积神经网络的对比。其中n为输入序列长度，d为编码维度，k为卷积核尺寸。

|  |  |  |  |
| --- | --- | --- | --- |
| 层类型 | 时空复杂度 | 运算序列长度 | 最短路径 |
| 自注意力层 |  |  |  |
| 循环层 |  |  |  |
| 卷积层 |  |  |  |

自注意力机制在时空复杂度方面的优势相比于循环层和卷积层并不明显，只在处理短文本的时候有所体现。其最大的优势在于突破了传统RNN网络的低并行度（Low parallelism）问题和长距离依赖（long-range dependence）问题。在使用自注意力机制计算时，词向量两两之间直接计算权重与加权结果，不存在跨越时间步的距离，同时以上过程互相之间可以完全并行，可以充分调动现代大规模分布式计算硬件设备的算力资源。

###### 多头注意力机制

在卷积神经网络中，常常会使用多通道卷积（Multi-channel convolution）操作，利用多个不同的卷积核，分别提取不同局部特征。Transformer模型中，也使用了多组参数，将模型的注意力计算模块分为多个头（Head），形成多个子空间，使模型可以去关注不同方面的信息。

多头自注意力机制的计算流程如图 3.5所示：

图示

描述已自动生成

图 3.5 多头注意力机制的运算流程

首先使输入词向量组成的矩阵分别与可学习参数矩阵、、相乘，得到一系列、、,其中。在Transformer基础模型中，=8，即总共有8个不同的注意力头; 各注意力头的参数矩阵、、维度相同，为，即[512,64]。在BERT基础模型中，则有12个不同的注意力头，对应的可学习参数矩阵维度为[768,64]。

之后各组、、分别进行注意力运算，得到一系列，其中i同上。

|  |  |  |
| --- | --- | --- |
|  |  | （3.2） |

之后拼接各注意力头运算结果，乘以输出权重矩阵得到输出Z。

|  |  |  |
| --- | --- | --- |
|  |  | （3.3） |

在具体代码实现中，参数矩阵、、可以拼接为、、保存在文件中，并且输入词向量组成的矩阵直接与、、相乘，得到的、、矩阵相当于由、、拼接而成，之后的注意力运算必须拆分各注意力头分别进行运算，总共重复次。

不过，这些不同注意力头对于模型而言，并非全部是必要的。有研究发现，在训练完成的模型中即使去掉绝大部分注意力头，模型的性能也未必会出现明显的下降[39]。相关的研究为基于注意力机制的模型的剪枝，即去掉部分多余的头以加快推理速度，提供了理论依据。

#### 位置编码

纯粹的Attention模块是无法捕捉输入顺序的，即无法区分不同位置的词。导致其与词袋模型类似，两个句子只要包含的词相同，哪怕它们的顺序完全不同，最终的表示结果也完全相同。为了解决这一问题，位置编码（Position Encoding）被引入进来，为系列中的每绝对一个位置赋予一个连续、低维、稠密的向量表示，作为模型输入的一部分，使得模型有能力区分不同位置的词。

在Transformer中，采用使用正弦和余弦函数，分别对位于偶数位和奇数位的词进行编码，具体公式如（3.4）所示：

|  |  |  |
| --- | --- | --- |
|  |  | （3.4） |

式中为该token在序列中的实际位置，表示该为偶数，表示该为奇数，。

对于不同的，将会由不同周期不同相位的三角函数进行编码，得到不同的结果，这就引入了绝对位置信息。相隔 个token的两个位置 和，可以表示成和的组合形式，这就引入了相对位置信息。具体推导公式如（3.5）所示：

|  |  |  |
| --- | --- | --- |
|  |  | （3.5） |

将位置编码后的向量和输入词向量进行对位相加，作为整个transformer的输入，即可使输入包含完整的位置信息。

Transformer这种依靠公式确定各位置编码的值的方式被称为固定式(Fixed)位置编码，而BERT模型采用的是学习式(Learned)位置编码。关于BERT模型的位置编码方式，以及为什么简单的相加即可使输入包含位置信息，本文之后的关于BERT模型的相关章节将会给出详细解释。

##### 位置前馈网络

位置前馈网络（Position-wise Feed-Forward network）是一个全连接网络，包含两个全连接层和一个ReLU激活函数。

其公式如（3.6）所示，其中、的维度分别为[]和[]：

|  |  |  |
| --- | --- | --- |
|  |  | （3.6） |

###### ReLU激活函数

线性修正函数（Rectified Linear Unit，ReLU）[40]是近几年兴起的分段激活函数，其函数表达式与导函数表达式如公式（3.7）和（3.8）所示，图像如图 3.6所示：

|  |  |  |
| --- | --- | --- |
|  |  | （3.7） |
|  |  | （3.8） |

图表, 折线图

描述已自动生成

图 3.6 ReLU函数及其导函数的图像

ReLU激活函数其导数在输入值大于零时恒等于1，能有效缓解梯度消失问题，但也一定程度上增大了梯度爆炸的风险。在输入值小于零时，该函数使得神经网络梯度不再更新，增加了网络的稀疏性，从而使模型更具有稳健性，但也有可能因此导致神经单元进入假死状态。该函数最大的优点仅需要if-else实现，运算方便快速。ReLU的输出均值大于0，容易改变输出的分布，一般需要搭配其他方法解决这一问题。

实验证明，相比sigmoid和tanh，ReLU可明显改善梯度消失的问题，且计算高效，因此在业界被广泛使用。ReLU函数也有一些改进型，比如Leak ReLU（LReLU）、ELU、SELU等函数，其改进思路都是对ReLU函数负半轴的输出进行调整，使其不为0，以缓解神经元永久性假死问题。不过这些函数的优势在实际场景中大多并未得到明显证明，因此本文不再赘述。

###### 位置前馈网络的作用

在位置前馈网络中，输入的维向量会先被线性变换为维，经过ReLU函数激活与Dropout后，再次经过线性变换映射回维。通过这一系列映射到高维空间再恢复的操作，可以增强模型的拟合与表达能力。谷歌的研究者[41]通过实验证明了，纯粹注意力的表达能力主要来自短路径，并且随着深度以双倍指数（Doubly Exponentially）方式衰减，最终导致产生一个秩为1的输出，这种结果被称为秩崩溃（Rank Collapse）。而残差连接和位置前馈网络的加入可以阻止或大大减缓秩崩溃，在Transformer模型中起着极为关键的作用。

图表, 折线图

描述已自动生成

图 3.7 ReLU、Leak ReLU、ELU、SELU的函数图像

##### Dropout机制

Dropout[42]是一种深度学习中被广泛的应用的，解决模型过拟合问题的策略。其目的是在训练的过程中每次都随机选择让一部分节点不进行学习，以此模拟现实世界中可能存在的数据异常或数据缺失，使得模型学习到将异常数据过滤只关心有效的有规律数据的能力，增强模型的稳健性[43]，但是因此Dropout会使得模型学习训练速度变慢。

同时，数学上可以证明，最小化含有Dropout网络的损失等价于最小化带有正则项的普通网络。这解释了使用Dropout操作可以用简单的操作有效缓解神经网络的过拟合问题。

图示

描述已自动生成

图 3.8 Dropout前（左）与Dropout后（右）的神经网络

Dropout在训练时会把以的概率将隐藏层的神经元置为零，同时会将其他神经元乘以，保证输出值均值的一致性，即对于隐藏层h执行公式（3.9）所示的操作：

|  |  |  |
| --- | --- | --- |
|  |  | （3.9） |

需要注意的是，在经过Dropout操作后，的均值不会发生变化，但是方差会发生变化，具体为。如果之后经过非线性层，会导致非线性层的输出发生偏移，最终导致整个网络的输出值发生偏移。尤其是在回归问题中使用Dropout时要注意这一点

在比较深的网络中，使用0.5的丢失率是比较好的选择，因为这时Dropout有最大的正则化效果；在比较浅层的网络中，丢失率应该低于0.2，因为过多的丢失率会导致丢失过多的输入数据对模型的影响比较大；不建议使用大于0.5的丢失率，因为它在丢失过多节点的情况下并不会取得更好的正则效果。

在Transformer中，Dropout操作被用于处理模型的输入，以及各线性层之后，但是多头注意力中对应的线性层之前和之后都用出现，且使用的丢失率较小，具体为0.1。

##### 归一化

归一化（Normalization）是一种用于深度神经网络的预处理技术，目的是通过规范化每层神经网络任意神经元的输入值的分布，使得它们呈现出均值为0、方差为1的标准正态分布。该操作一般应用在深度神经网络中激活层之前，以加快模型训练时的收敛速度，使得模型训练过程更加稳定，并避免梯度爆炸或者梯度消失问题的出现，同时起到一定的正则化作用。

归一化的有效性最初被解释为缓解了神经网络的“内部协方差漂移（Internal Covariance Shift，ICS）”[44]，但是该解释目前被认为是不可靠的[45]。另一种解释是归一化使得激活函数的输入值落在比较敏感的区域，这样输入的小变化就会导致损失函数较大的变化，从而可以避免梯度消失问题的出现，同时也能够加快学习收敛速度。迄今为止，依然没有一个数学上非常严谨的证明来解释其具体过程，但是归一化在实践中表现出的优越性能受到了广泛认可，并在几乎所有新兴模型中得到了应用。

###### 批次归一化

批归一化（Batch Normalization，BN）[44]是Google在2015年提出的数据归一化方法。其处理对象是对一批样本，是对这批样本的同一维度特征做归一化。如图 3.9所示，Batch Normalization即在batch方向上，对同一通道（C）内不同样本的特征图[N, H, W]做归一化。在CNN中，同一通道内的特征图来自同一个卷积核，相当于对不同样本的同一局部特征进行归一化。

图示

描述已自动生成

图 3.9 批次归一化、CNN中的层次归一化、Transformer中的层次归一化、实例归一化的输入数据范围。图中参数C、N、H、W在不同应用场景代表不同数值。在CV领域N为batch\_size，C为通道数（channel），H、W为图片维度。在NLP领域，C为词向量维度(embedding\_size)，H、W为句子长度（seq\_len）。

BN计算的过程可以概括为：对输入向量，计算其中元素的均值与方差后，对各元素进行如下操作。式中参数、为可学习参数；是一个极小的数，用于防止分母为0。

|  |  |  |
| --- | --- | --- |
|  |  | （3.10） |

可以看到，BN首先会将中每个特征的数据进行规范化，转换为一个均值为0、方差为1的分布。然而，如果只进行这样的规范化操作，会造成数据表达能力的缺失，因为原有数据的分布中可能隐藏了一些重要的信息。另外，当均值为0、方差为1的输入经过sigmoid或tanh激活函数时，容易陷入非线性激活函数的线性区域，从而损害模型的非线性拟合能力。

因此，BN又引入了两个可学习的参数、，对规范化后的数据进行放缩和平移，恢复数据本身的表达能力。比如，当、时，可以实现等价变换（Identity Transform）保留了原始输入特征的分布信息。

BN有一些明显的缺点，比如当batch size过小时，对均值的估算会不准确。这一点在用于推理时尤其明显。因此，BN在训练时会采用滑动平均（Moving Average）方法近似计算出整个训练集上的均值与方差并保存下来，在推理时直接使用。

此外，对于训练和推理时batch size都很小，比如经常只用一张图片作为训练集，且重视每个单独像素的信息的图片风格迁移任务，有研究者提出了实例归一化（Instance Normalization，IN）[46]，只在通道方向上进行归一化，在具体任务时表现良好。

###### 层次归一化

BN方法并不适合在RNN中使用。以使用RNN的NLP任务为例，BN相当于对不同句子中同一位置的词进行归一化。这样的操作是不符合直觉的，在数学上也是没有意义的。因为不同句子中同一位置的词的特征分布大概率是不同的。针对这个问题，研究者提出了层次归一化（Layer Normalization，LN）方法[47]。

LN的处理对象是单个样本，是对这单个样本的所有特征做归一化。即在通道数方向上，对[C, H, W] 归一化，可以在RNN中应用，并且表现优秀。这一方法同样可以在CNN中使用，比如前文提到的图片风格迁移任务，其风格依赖于单个样本，BN在这种情况下会造成混淆，而LN可以保证每个图像样本之间的独立。

在CV领域中，对于长宽不同的图片，一般可以采用缩放、裁剪、或者池化操作使得同批次内图片长宽统一。但是在NLP领域中，则很难保证各批次中的每个句子长度相等。句子不能像图片那样使用像素插值等方法。对于较短的句子，往往需要进行填充（padding）。这就造成了大量的无用数据，会影响归一化操作的效果。

根据Transformer论文的相关章节中，Transformer模型使用的归一化方法是LN。不过要注意的是，NLP领域的LN与CV领域的含义并不完全相同。具体来说，Transformer模型中的归一化，是以每一个词向量为单位进行归一化。这样的归一化仅纳入了有实际意义的数据，避免了填充（padding）符对均值和方差的干扰。该过程与同样是对单一维度进行归一化的IN非常相似，需要注意辨析避免混淆。

由于归一化会对数据的正负比例产生影响，因此适合放在sigmoid激活函数的前面，ReLU激活函数的后面，或者具体问题具体分析。归一化不能放在Dropout操作后面使用，否则Dropout层的计算结果会改变批量归一化的输入数据的分布，导致归一化之后的偏差更大。

##### 残差连接

残差连接(Residual Connection)，有时也写作跳跃连接（Skip Connection）或捷径连接（Shortcut Connection)，最早由何凯明提出[47]，是一种用于缓解深度神经网络中梯度消失的方法。该方法通过在模型中加入恒等映射（Identity Mapping），使得梯度在后向传播的过程中能有效传播到深度神经网络中靠前的层。

如图 3.10所示，使用了残差连接的多个层被称为残差块（Residual Block），多个残差块堆叠形成的深度神经网络被称为残差网络（Residual Net）

###### 残差连接原理

如果在章节2.4.2.3的例子中的输出层加上第一层的输出，即式（3.11）：

|  |  |  |
| --- | --- | --- |
|  |  | （3.11） |

则对第一层可学习参数矩阵的求梯度过程为式（3.12）：

|  |  |  |
| --- | --- | --- |
|  |  | （3.12） |

展开为式（3.13）：

|  |  |  |
| --- | --- | --- |
|  |  | （3.13） |

可以看到，新加入的常数1大幅缩短了梯度传播到输入层所经过的运算过程。即使连乘项导致了梯度消失，残差连接也能保证部分梯度的存在，使得深层神经网络的靠前的层的参数也能及时更新。

手机屏幕截图

低可信度描述已自动生成

图 3.10 残差连接的结构

#### Transformer Decoder简述

由于Transformer中Decoder与Encoder部分高度相似，且本文主要涉及BERT模型的相关内容，所以对于Decoder部分不作过于详细的解析，仅简述其与Encoder部分的主要区别。

##### 交叉注意力层

Decoder相比于Encoder，其结构上的区别在于加入了交叉注意力层。在该层中，与、相乘获取K、V的输入将来自最后一层Encoder的输出。这种Q、K、V矩阵分别来自不同模块的注意力机制被称为交叉注意力机制，引入该机制的目的是使得Decoder能充分利用Encoder编码得到的原始文本的特征信息。

要注意的是，每一层Decoder的交叉注意力层的输入都是相同的，即最后一层Encoder的输出，而非一一对应。由于翻译任务中源语言序列与目标语言序列的长度常常不相等，因此还需要一起转送填充符所在位置信息。

##### 带掩码的多头注意力机制

Decoder部分的注意力机制采用了带掩码的多头注意力机制（Masked Multihead self-attention），使得在预测每个词时只能看到此时刻之前的信息，而无法访问此时刻之后的信息，防止模型在生成输出时窃取来自未来的信息。

图表

描述已自动生成

图 3.11 Decoder对输入序列进行掩码的流程

在训练时，Decoder的输入是目标语言的句子经过分词器编码后的输入序列。该输入序列的开头会被加入用于表示句子起始的特殊符号</bos>。长度为的输入序列会被转换成个训练样本，构成一个维度为的矩阵，在计算多头自注意力时，生成的权重矩阵会与Mask矩阵求哈达玛(Hadamard)积，得到一个类似于下三角矩阵的结果。该过程中，保留的部分不会发生变化，被Mask的部分会被替换为一个非常大的负数，使得这部分在经过softmax函数后注意力权重接近于0，相当于不参与注意力运算。这种掩码方法使得各样本将可以并行地进行训练，训练目标都是预测未被掩码的部分后的第一个单词。

在推理时，Decoder部分将会从开始符</bos>开始，串行地每轮输出一个单词，直到某次输出为终结符</eos>。

### BERT模型

BERT（Bidirectional Encoder Representations from Transformers）是由Jacob Devlin等人[7]在2018年提出的一款基于深层Transformer的预训练模型。该模型能够充分地利用大规模无标注文本挖掘和学习通用的语言知识，并将这些知识应用于各种特定任务中。

以BERT为代表的预训练模型打破了不同自然语言处理任务之间的壁垒，使得搭建一个面向特定任务的自然语言处理模型不再需要了解非常多的任务背景知识，只要根据任务的输入输出形式应用这些预训练语言模型就能达到一个不错的效果。因此，该范式迅速成为自然语言处理的新范式，使得自然语言处理的模型搭建变得不再复杂。

#### 由GPT得到的灵感

OpenAI公司在2018年结合Transformer模型的Decoder部分与预训练模型的相关工作，提出了生成式预训练（Generative Pre-Training，GPT）模型[6]，提出了“生成式预训练+判别式任务微调（Discriminative Fine-tuning）”的自然语言处理新范式，正式将自然语言处理带入了“预训练”时代。该模型的相关工作，极大地启发了BERT模型的出现。

图示

描述已自动生成

图 3.12 GPT模型结构（左）与其中Transformer块的结构（右）

GPT模型的结构是12层堆叠的Transformer Decoder，由于只有Decoder部分，所以去掉了交叉注意力层。该模型的输入部分采用了字符级BPE分词器，以及与Transformer的位置编码方法不同的可学习位置编码。这里的可学习位置编码与接下来介绍的BERT模型的位置编码方式相同。除了层数更多外，GPT模型的词向量编码维度和注意力机制的可学习参数矩阵、、维度，即与Transformer中对应的超参数为768。单个注意力头的维度仍是64，总共12个注意力头。位置前馈网络的隐藏层维度保持了和词向量维度的四倍关系，即3072维。模型总参数量约1.17亿。

GPT模型的预训练任务是单向语言模型任务，基于继承自Transformer的掩码多头注意力机制实现并行化。原论文中将其称为生成式预训练，通过在约5G数据量的大规模的文本数据上训练一个学习更加丰富的语义表示。

判别式任务微调，指的是将训练好的有通用语义表示能力的模型适配到下游任务（Downstream task）当中，使用有标注的数据进行判别式学习，使得其与下游任务的领域和形式更加契合。通过对GPT模型的输入与输出数据结构的设计，并使用线性层对输出数据进行合理的整合与映射，就能实现GPT在下游任务的应用。

#### Bert模型结构

BERT模型的主要结构与GPT高度相似，都是多层堆叠的Transformer Block。主要区别在于GPT基于其单向语言模型预训练任务，选择了带有掩码注意力机制的Transformer Decoder层，而BERT模型为了实现对自然语言的双向建模，采用了Transformer Encoder层，并针对性地设计了新的预训练任务，并改进了输入层。

表格

描述已自动生成

图 3.13 GPT用于多种下游任务时的输入与输出结构

图示

描述已自动生成

图 3.14 BERT模型结构

BERT模型根据其参数规模，一般分为两个版本。基础版模型BERT-base的模型规模超参数与GPT相同，即词向量编码维度，，堆叠12层。模型总参数量约1.17亿。此外还有大参数量版本BERT-large，词向量编码维度，，堆叠24层。模型总参数量约3.40亿。BERT-large可以更充分地学习与抽取文本的信息，在绝大部分实验中的表现都略好于base版模型，但是对计算资源的消耗量远大于base版模型。本文中除非具体说明，默认的BERT模型规格为基础版。

#### BERT预训练任务

GPT的预训练任务是经典的语言模型任务，属于自回归（Auto-Regressive，AR）预训练。而BERT的研究者突破性地引入了基于自编码（Auto-Encoding，AE）的预训练，具体包含掩码语言模型（Mask Language Model，MLM）和下一句预测（Next Sentence Prediction，NSP）任务。这些任务使得BERT可以充分双向学习与表示大规模无标准文本中的信息，同时进一步加深了自然语言模型的深度。

##### MLM任务

传统基于条件概率的语言模型，只能正序或者逆序对文本进行建模。如果同时进行正序和逆序建模，会使得对于正序模型而言需要预测的“未来”的词被逆序模型暴露，对于逆序模型而言需要预测的“过去”的词被正序模型暴露。使得两个模型的都倾向于从对方直接读取需要预测的词并直接输出，导致模型过于简单，无法学习深层次的语义信息。因此，如ELMo等模型会采用两个互相独立的前向与后向语言模型分别对文本进行建模。

为了实现真正的双向语言建模，即当前时刻的预测同时依赖于“过去”和“未来”的上下文信息，BERT的MLM任务采用了一种类似于完形填空（Clonze）的做法，对输入文本的部分单词进行掩码（Mask），迫使模型同时使用被掩码的单词的上下文还原被掩码位置的词。

在掩码时，模型根据一定比例，用预设的特殊token——[MASK]替换原单词，以表示该位置已经被掩码。然而，这样会造成与训练阶段和下游任务微调阶段之间的数据不一致性，因为专为MLM任务人为引入的[MASK]标记并不会在实际的下游任务中出现。为了缓解这个问题，在对输入序列进行掩码时，并非总是将其替换为[MASK]标记，这会按照概率选择以下三种操作的一种：

表 3.2 BERT的掩码方式举例

|  |  |
| --- | --- |
| 原文 | 我的狗是毛茸茸的 |
| 80％的概率：用[MASK]词块替换单词 | 我的[MASK]是毛茸茸的 |
| 10％的概率：用随机词替换遮蔽词 | 我的苹果是毛茸茸的 |
| 10％的概率：保持单词不变 | 我的狗是毛茸茸的 |

这使得模型并不知道具体哪些单词已经被随机替换并且需要预测，迫使模型学习每个词的语义表示。模型不仅需要理解当前空缺位置之前的词，同时还要理解空缺位置之后的词，从而达到双向语言建模的目的。

实验显示，选择15%的掩码比例时BERT模型的表现最好。这是一个较小的比例，使得BERT模型的收敛略慢于传统的单向语言模型。对这一比例的解释是，自然语言较高的信息密度使得较大的掩码比例对模型而言理解与还原的难度过大，会损害模型的理解能力。

##### NSP任务

对于阅读理解、文本蕴涵等需要两端或多段文本输入的任务来说，BERT模型无法仅依靠MLM任务显式学习多段文本的关联。因此BERT模型引入了第二个预训练任务NSP任务。NSP任务是个二分类任务，要求模型判断句子B是否是句子A的下一句。训练的正样本来自语料库中自然相邻的两个句子A和B，负样本则是将句子B替换为语料库中任意一个其他句子。

表 3.3 BERT的NSP任务样本举例

|  |  |
| --- | --- |
| 输入 | Label |
| [CLS]男子到商店[SEP]他买了一加仑牛奶[SEP] | IsNext |
| [CLS]男子到商店[SEP]企鹅是飞行鸟类[SEP] | NotNext |

NSP任务的正负样本比例控制在1:1。由于其较为简单的设计，因此可以自动生成大量训练样本，而且可以达到很高（97%—98%）的准确率。

#### BERT模型输入

对于BERT模型而言，需要在充分理解模型主要算法与预训练任务的基础，才能理解BERT模型的输入部分的设计原理。BERT的输入层由三个部分组成，分别为词向量（Token Embedding），块向量（Token Embedding）和位置向量（Token Embedding）。以上三个向量的维度相同，在基础模型中维度，最终的输入为以上三个向量按位置相加的得到。如图 3.15所示。

图示

描述已自动生成

图 3.15 BERT模型的输入层结构

用公式表示如式（3.14）所示：

|  |  |  |
| --- | --- | --- |
|  |  | （3.14） |

##### 词向量部分

BERT模型的词向量部分与传统模型的词向量部分类似，通过词向量矩阵将输入的token转化为向量表示。如式（3.15）所示：

|  |  |  |
| --- | --- | --- |
|  |  | （3.15） |

式中是依据token在词典中位置下标构造的one-hot编码表示，字典长度。

BERT模型在用于中文时，依旧使用的与英文版相同的WordPiece算法关于具体的字典生成与分词器训练过程，研究者表示其使用的是Google内部的WordPiece工具，并没有开源。推测其主要过程与前文提到的方法类似，先使用CWS将中文训练语料转化为自然词之后，再输入WordPiece算法。

BERT字典中含有一些特殊token。如表 3.4所示：

表 3.4 BERT字典中的特殊token及其位置下标和含义

|  |  |  |
| --- | --- | --- |
| 特殊token | 位置下标 | 含义 |
| [PAD] | 0 | 填充（padding）位，用于填充数据样本中的各句子到相同的长度 |
| [UNK] | 100 | 未知（unknown）的输入，用于表示字典外输入 |
| [CLS] | 101 | 句子的起始，也是句子分类（Classification）任务的特征向量 |
| [SEP] | 102 | 句子的结束，也在NSP等有两个输入序列的任务中作为分隔符 |
| [MASK] | 103 | 用于在MLM任务中表示被掩码的token |

除此之外，字典中位置下标1到99为保留位置，可以由根据具体任务手动加入新的token。

BERT字典中除了中文字符外，还有一些常用的英文单词、数字、标点符号等等，这些内容可能包含有Google团队凭借经验加入的部分。从结果上来看，这套字典的使用非常广泛，是大量后续研究和生产应用的默认标准字典。

但是，一个汉字作为一个子词的分词粒度，相当于字符（Character）级分词。与中文的自然语言使用习惯相比，明显偏细。针对这一问题，一种直接的解决思路是扩展词典，加入大量符合中文使用习惯的多字词和短语。比如腾讯技术工程部门曾尝试设计并训练了词典大小在80K和500K级别的BERT，并使用自适应（Adaptive）Softmax结构避免词向量矩阵参数量过大。但是最终模型的表现并不理想，甚至不能有效解决未登录词的问题。此外也有一些研究发现对于中文语言模型而言，词（Word）级别的分词并不是必须的[49]。本文的3.4章节将会详细介绍一种能有效解决字级词典粒度偏细的问题的方法。

##### 块向量部分

由于BERT模型只有Encoder部分，即每条输入样本只能有一个序列，在用于NSP等有两个输入序列的任务时，就需要将两个序列拼接起来，中间插入[SEP]作为分隔符，同时对拼接后序列的各token属于哪一个序列进行标记，并且使模型可以学习这一部分隶属信息。如式（3.16）所示：

|  |  |  |
| --- | --- | --- |
|  |  | （3.16） |

依据token在两个序列中隶属关系构造的one-hot编码表示，，表示最多可以拼接两个输入序列。

##### 位置向量部分

与Transformer中的固定式位置编码不同，BERT采用了可学习式位置编码。这是一种非常简单直观的位置编码方式，即让模型在训练过程中自行学习各位置的信息并使用位置向量进行表示。如式（3.17）所示：

|  |  |  |
| --- | --- | --- |
|  |  | （3.17） |

式中是根据token在句子中的位置下标构造的one-hot编码表示，表示最长可以编码的序列长度。

很显然，BERT这种位置编码方式无法处理超过预设最长可以编码的序列长度的输入。如果为了避免无法处理长输入序列，预设一个很大的，则会由于缺乏足够长的预训练语料导致靠后的位置编码得不到充分的预训练。被认为是一个权衡各方利弊后得到的较为合理的值。对于需要处理大量长度超过512的输入序列的任务，应当考虑采取一些对长文本输入进行处理的方法，或者选择经过后世研究者改进的模型，比如引入了块级循环Segment-level Recurrence）和相对位置编码（Relative Position Encoding）的XLNet[9]。

##### 输入特征融和的原理

将词向量（Token Embedding），块向量（Token Embedding）和位置向量（Token Embedding）进行特征融合，最符合的方法应该是将三个向量拼接（Concat）后作为最终的输入向量。即：

|  |  |  |
| --- | --- | --- |
|  |  | （3.18） |

通过对求和特征融合的运算公式，即式（3.19）：

|  |  |  |
| --- | --- | --- |
|  |  | （3.19） |

进行简单展开就可以知道，这种拼接特征融合方法与下式展示的求和特征融合方法在数学上相同的，如图 3.16所示。

这个维的空间维度足够高，对于“任意一个token位于任意一个位置属于任意一块”的情况，只要训练语料足够多，都可以找到一个合适的点，充分表示并进行区分。综上所述，BERT模型的输入设计是巧妙、合理，而且高效的。

图示

描述已自动生成

图 3.16 先映射再相加（左）与先拼接再映射 （右）

通过直观的图示可以看到，无论是先拼接再映射，还是先映射再求和，其本质都是将空间的点映射到一个维的特征空间。这个维的特征空间即是模型的输入特征空间。

图示

描述已自动生成

图 3.17 特征空间的映射

#### BERT模型前向运算过程

在开始运算前，首先要使用分词器（tokenizer）对文本进行字词切分与编码。由于训练时同一个batch中往往存在不同长度的文本，分词器一般还需要将过长的截断到预设的最大长度。对长度小于最短长度的文本，将会句末加上一系列填充符[PAD]，填充到预设最大长度。分词器还需要生成保存了填充符所在位置信息的向量，用于在计算注意力时将填充符替换为一个非常大的负数，使得这部分在经过softmax函数后注意力权重接近于0，相当于不参与注意力运算。

Bert Embedding层根据分词器编码结果对输入进行词向量化，并且与位置编码、块编码相加，经过Layer Normalization与Dropout后，得到最终的模型输入。

之后进入Bert Encoder Layer中的多头自注意力层进行注意力计算，注意力计算的结果的输出经过Dropout后，再经过一个线性层并再次进行Dropout，得到多头注意力层的输出，这里的线性层对应。多头注意力层的输出与输入相加，形成残差连接，之后进行Layer Normalization。接下来经过位置前馈网络运算，将H维输入映射到4H维，经过ReLU函数激活后再映射回H维。位置前馈网络的输出同样与输入相加，形成残差连接。最后进行Layer Normalization，得到Bert Encoder Layer的输出。

Bert Encoder Layer块的输入与输出维度被设计为相同的值。可以多层串联堆叠，在BERT基础模型中堆叠了12层，有利于对文本信息进行充分的提取。

#### BERT用于文本分类下游任务

与GPT模型类似，通过在BERT模型后面增加线性层，并对模型输出进行合理的整合与映射，即可应用于多种下游任务，如图 3.18 BERT用于文本分类任务的输入与输出所示。

图示

描述已自动生成

图 3.18 BERT用于文本分类任务的输入与输出

以本文的文本分类任务（Sentence Classification Tasks）为例，无论是类似于NSP预训练任务的句子对分类任务，还是经典的单句分类任务，都只需要将模型编码后的整个序列开头的[CLS]符对应的向量视作整个句子的特征，输入线性层分类器映射到目标类别即可。

自注意力机制会使用文本中的其它词来增强目标词的语义表示，但是目标词本身的语义还是会占主要部分。而[CLS]符本身没有语义，经过模型编码后，得到的是所有词通过自注意力机制的加权平均，与文本中已有的其它词相比，[CLS]这个无明显语义信息的符号会更“公平”地融合文本中各个词的语义信息，从而更好的表示整句话的语义。

另一种应用方式是将BERT对整个句子的编码结果都利用起来，作为动态词向量输入给类似于TextRNN等经典分类模型。实验显示，相比于ELMo等经典动态词向量模型和其他静态词向量模型，BERT输出的动态词向量表现更好。但是在单句分类任务中，这种方法并不比仅使用[CLS]符的方法有明显的提升。这一现象可以用“水桶效应”来解释：整个系统的短板在于BERT对句子特征的编码与提取能力，更复杂的输出层并不能提升BERT模型本身的表现。

#### BERT模型的缺陷

BERT模型在后世的使用与研究中，被发现存在有一些缺陷[9]。这些缺陷大多与BERT为了实现双向语言模型而设计的MLM任务有关。具体如下：

微调不匹配（Pretrain-Finetune Discrepancy）问题。MLM任务中添加的[MASK]在下游任务样本中一般不会出现，这就使得预训练过程与微调过程存在数据不一致问题，影响迁移学习的效果。虽然BERT模型在最初设计MLM任务时就考虑到了这一点，并进行了处理，但是只能很有限地缓解问题。

独立性假设（Independence Assumption）问题。BERT模型在预测每个被[MASK]的token的时候认为所有的[MASK]互相独立的。这相当于给输入的句子一个默认的假设：所有的子词都是互相独立。然而在自然言语中该假设并不成立。比如英文的“giraffe”和中文的“苹果”，假设二者在经过WordPiece算法切分为字词并掩码后的结果分别是“gira[MASK]”和“苹[MSAK]”。很显然，两个词的剩余部分的限定性都非常强，在这种情况下模型会倾向于学习单词级的短距离固定序列的出现规律，而非从上下文语义信息中学习并还原被掩码的词，可能导致对模型长距离建模能力与语义理解能力的损害。

缺乏细粒度的语义表示能力。研究者们发现，BERT模型在部分子词的语义表示上仍有一些问题，比如在降维“sky”和“sea”的编码特征之后可以发现二者的特征距离很近，而在自然语言的语义上二者实际上相差很远。缺乏细粒度的语义表示能力反映了BERT模型本身的一种能力缺陷，可能在某些真实任务上造成很大影响，如果被针对性攻击更是无法应对。

### RoBERTa模型

RoBERTa[8]是在论文《RoBERTa: A Robustly Optimized BERT Pretraining Approach》中被提出的BERT改进版本，该模型在预训练数据、预训练方法等方面对BERT进行了改进，并且取得了非常优异的表现。对之后的许多BERT模型相关的研究与应用都起到了启发作用。

#### RoBERTa模型的改进

RoBERTa模型相比BERT的改进主要有以下几点：

更多的训练数据。BERT预训练使用了约16GB大小的 BookCorpus数据集和英语维基百科数据集。RoBERTa模型的预训练在此基础上增加了CC-NEWS、OPENWEBTEXT、STORIES等数据集，最终数据集大小达到了160G。大规模的语料库使之后更充分的训练成为了可能。

更大的batch size。RoBERTa在训练过程中尝试了与BERT相同的256，以及2k、8k等更大batch size。论文中通过实验，证明了使用更大的batch size并适度增大学习率减少训练总步数，可以得到更好的结果。论文作者在考虑了并行计算等因素后，最终选择了batch size=8k进行训练。

动态（Dynamic）掩码。原版的BERT在创建MLM任务样本时，仅在第一次加载数据时执行掩码，之后的每一个epoch都使用了相同的数据样本。这种掩码方式被称为静态（Static）掩码。很显然，这种掩码方法对语料的利用存在大量重复。RoBERTa作者对此提出了动态掩码方法，即在每一次生成模型输入时才随机进行掩码，大幅提升了训练语料的利用率，同时使模型表现更佳。

去掉了NSP任务。RoBERTa作者在实验中发现，在使用了足够大的语料库进行了足够充分的MLM预训练后，再进行NSP预训练，在下游任务上几乎没有提升，甚至还有所损害。在尝试了多种NSP预训练方法后的实验结果同样支持这一结论。近年来的大量其他研究也支持这一结论。目前公认的解释是，使模型学习到句子级的长距离依赖关系的思路并没有错误，但BERT的NSP任务设计可能并不合理，或者说过于简单，并不能使模型进行符合预期的学习。NSP任务在原版BERT的消融实验中被证明有效是因为原版BERT的MLM预训练不够重复，在更大规模语料库上进行了足够充分的MLM预训练后，NSP任务应该被去掉。

字节级BPE（Byte-level BPE，BBPE）分词方法。RoBERTa采用了与GPT2相同的BBPE算法，该方法用字节（Bytes）作为基础的子词单元，词表的大小是50K，大于英文BERT的30K。在UTF-8编码中，每一个字符会被编码到长度为1到4个字节的序列中，BBPE算法会将一个原始的Unicode 字节序列（Byte Sequence）分割成字节级子词（Byte-level Subwords）。之后使用BPE算法，提取出个出现次数最多的子序列作为词典。此外，不同于BERT在子词开头加入“##”表示该子词为后缀，RoBERTa在子词开头加入特定的 Unicode 字符(比如\u0120，带点的G)表示该子词为前缀。采用BBPE编码的好处是可以在不需要引入任何未知字符前提下对任意文本进行编码，RoBERTa作者相信这个编码方式更通用。

#### RoBERTa模型用于中文

RoBERTa模型在重要结构上相比于BERT几乎没有修改，仅因为不再包含NSP任务所以去掉了块编码层，但是在将RoBERTa模型用于中文时，BBPE算法并不是较为理想的中文表示方法。对于在UTF-8中一个字符占一个字节的英文字母而言，BBPE算法的表现近似于优化版的BPE方法。但对于一个字占两个字节的中文而言，BBPE算法的粒度显然过于细了，对中文字的语义的混淆与破坏较为严重，生成的词典也几乎完全不可读。

为了避免出现前文提到的，类似BERT模型缺乏细粒度的语义表示能力的问题，大部分中文RoBERTa模型沿用了BERT的WordPiece算法与词典，或者说使用了与BERT相同的tokenizer。由于输入相同，部分中文RoBERTa模型直接使用了与BERT完全相同的模型。虽然依旧保留了块编码层，但中文RoBERTa模型同样不包含NSP任务，在训练时只使用单句。

### 全词掩码预训练方法

全词掩码（Whole Word Masking，WWM）是由哈工大的研究者[10]提出的一种改进型MLM预训练方法，主要更改了原版MLM的训练样本生成策略。简单来说，原有基于WordPiece的分词方式会将一个完整的词切分成若干个子词，并在生成训练样本对这些分开的子词随机进行掩码。在全词掩码中，如果一个完整的词的部分WordPiece子词被掩码，则同属该词的其他部分也会被掩码。需要注意的是，这里的掩码指的是广义的掩码（替换成[MASK]；保持原词汇；随机替换成另外一个词），并非只局限于单词替换成[MASK]标签的情况。下表是一个例子，为方便理解，表中只考虑替换成[MASK]标签的情况：

表 3.5 全词掩码操作与样例

|  |  |
| --- | --- |
| **说明** | **样例** |
| 原文 | 我的狗是毛茸茸的 |
| 分词文本 | 我的 狗 是 毛茸茸 的 |
| 原始Mask输入 | 我的 狗 是 毛茸[MASK] 的 |
| 全词Mask输入 | 我的 狗 是 [MASK] [MASK] [MASK] 的 |

WWM方法可以缓解前文提到的独立性假设问题。同时对于中文BERT而言，一个汉字作为一个子词的分词粒度相比于英文子词的词根与词缀，本身就有些偏细。WWM预训练方法可以在一定程度上弥补中文BERT的细粒度表示的缺陷，增强BERT模型在词级别学习和理解中文语义的能力。

WWM方法在用于英文训练样本时较为简单，可以根据某个将被掩码的子词，向前匹配开头不带“##”的词根或向后匹配开头带“##”的词缀即可，也可以根据空格和标点符号对原句子进行英文分词，根据英文单词匹配需要被一起掩码的子词。但是WWM预训练方法在用于中文时，必须先使用中文分词（CWS）工具对原句进行中文分词，根据中文分词结果与原句的子词序列逐个比对，标记那些在WWM时需要一起掩码的子词。

WWM方法同样适用于RoBERTa模型。实验证明，该方法在中文和英文场景下，对包括BERT、RoBERTa在内的多种使用MLM预训练任务的预训练模型的性能都有明显的提升。

此外，百度的研发团队在其类BERT模型ERINE中提出过与WWM方法类似的实体级 (Entity-leve)掩码和短语级 (Phrase-level)掩码策略。 实体级掩码通常会掩盖由多个单词组成的实体，短语级会掩盖由几个词共同作为概念单元的整体短语。这两种方法的实现相比于WWM较为困难，但也可以提升模型的性能。

### BERT增量预训练

#### 经典语言模型的增量预训练

在BERT等大规模预训练模型兴起之前，就存在大量在文本分类任务上应用预训练语言模型的相关研究，其中的代表性工作就是2018年Howard与Ruder[11]总结的通用语言模型微调（Universal Language Model Fine-Tuning ，ULMFiT）方法。

图示

描述已自动生成

图 3.19 ULMFiT模型结构与流程

该研究搭建了一个3层LSTM语言模型，经过在大规模通用语料上进行语言模型任务预训练、在分类任务语料上进行语言模型任务增量预训练、在分类任务语料上进行分类任务微调三个步骤，结合逐层递减学习率等微调策略，最终取得了很好的效果。以下三个阶段，实现的很好的文本分类效果。

（1）. 大规模通用语料上进行语言模型（LM）任务预训练。

（2）. 在分类任务的小规模语料上进行语言模型（LM）任务微调。

（3）. 在分类任务的小规模语料上进行分类任务微调。

其中的第（2）阶段可以被视为增量预训练，目的是实现模型的任务迁移（Task transfer）。在实践中，一般还会在（1）（2）之间加入在分类任务语料同领域的语料上进行语言模型任务增量预训练阶段，目的是实现模型的领域迁移（Domain transfer）。

#### 用于专业领域的BERT模型

原版BERT的预训练语料主要集中在常识（Commonsense Knowledge）和开放领域知识（Open Domain Knowledge），但是，有很多重要的任务需要用到专业领域知识（Specialized Domain Knowledge）。因此，大量使用具体专业领域语料进行预训练的BERT相继出现。比如使用生物医药领域语料库训练的BioBERT[50]、使用金融领域语料库训练的FinBERT[51]、使用诊断文本和出院报告语料库训练的ClinicalBERT[52]、以及使用计算机和生物医药语料训练的SciBERT[53]等等。这些用于具体专业领域的BERT有的选择在原版BERT的基础上进一步训练，也有的选择重新构建词典从头开始训练，并且都在对应领域的相关任务上获得了提升。

#### 以专业知识区分领域的增量预训练

2020年，Gururangan等人[13]针对这一问题进行了统一的对比实验，在生物医学领域（BioMed）、计算机科学领域（CS）、新闻领域（News）、评论领域（Reviews）这四个领域上选择了8个分类任务（每个领域上各有2个分类任务）。通过这四个领域的语料与原版RoBERTa训练语料库的高频词重合度的对比，可以发现其两两之间的相似度普遍较低，这也为增量预训练的合理性与潜力做出了介绍。

Gururangan等人尝试了领域自适应预训练（Domain-adaptive Pretraining, DAPT）与任务自适应预训练（Task-adaptive Pretraining, TAPT）两种常用方法。DAPT就是在预训练完成的语言模型RoBERTa的基础上，分别使用4个领域的数据接着做第二阶段的预训练，生成4个领域自适应的语言模型。TAPT就是将各个任务本身的训练数据当作无标签数据来进行第二阶段的预训练，得到8个任务自适应的预训练语言模型。与DAPT相比，TAPT使用的是更小的，但与目标任务更相关的预训练数据集。

从结果上来看，DAPT与TAPT两种方法单独使用都能显著增强BERT在目标领域上的表现，先DAPT 再TAPT的结合使用效果优于单独使用，并且可以通过实验证明性能的提升不是因为新增的训练数据或步数。由于数据与任务相关度更高，TAPT在部分任务上的表现甚至优于DAPT。基于这一结论，作者又尝试了增强训练数据的任务自适应预训练（Augmenting Training Data for Task-adaptive Pretraining），使用人工筛选或算法自动选择的方法，获取更多与任务数据同分布的预训练数据，进行更充分的TAPT，同样进一步取得了显著的提升。

#### 以任务类型区分领域的增量预训练

2019年，复旦大学的邱锡鹏团队[12]进一步研究了BERT在多种类型的文本分类问题上的增量预训练问题。他们将7个英文文本分类数据集按照任务类型分为三个领域：主题、情感、问题。并实验了领域内增量预训练（In-Domain Further Pre-Training，IDPT）和任务内增量预训练（Within-Task Further Pre-Training，ITPT），以及使用其他领域语料进行跨领域增量预训练的领域内增量预训练（Cross-Domain Further Pre-Training，CDIT）。

卡通人物

描述已自动生成

图 3.20 预训练语料、任务领域语料和任务语料的交叉关系。图中灰色部分是与任务语料同分布的增强数据语料。

通过分析实验结果可以发现，经过IDPT与ITPT的BERT模型在目标任务上的表现均优于原版BERT。通常来讲，IDPT的表现由于ITPT，但ITPT的表现差距一般不大，且在部分任务上由于IDPT。而CDIT一般不能给BERT带来显著的性能提升，比较BERT在预训练阶段就已经获取了通用领域的知识。

虽然ITPT有着很高的的性价比，但是实验发现ITPT步数在10万步以内时才会对模型在具体下游任务上的性能起到改善作用。当ITPT步数过多时，反而会造成模型在下游任务上的性能下降。该现象可以用前文提到的，BERT模型的微调不匹配问题等缺陷进行解释：预训练任务只是下游任务的替代任务，而不是与下游任务完全相同。过度预训练反而会导致BERT在预训练任务上出现过拟合。

图表, 折线图

描述已自动生成

图 3.21 对BERT进行ITPT后的分类错误率变化曲线

### BERT微调技巧

#### 较低的学习率

灾难性遗忘（Catastrophic forgetting）[54]，是迁移学习中的一个常见问题，指的是预训练模型在学习新知识的时候，几乎遗忘了在训练阶段学到的知识。有研究者对BERT可能出现的灾难性遗忘问题进行了规律总结，发现采用较低的学习率（Learning Rate）缓解灾难性遗忘问题[12]。

通过使用多种不同的学习率进行对比实验可以发现，当微调学习率达到4e-4时，BERT模型会出现明显的灾难性遗忘现象，甚至无法在下游任务上收敛。随着学习率降低，灾难性遗忘现象会有明显的缓解。从结果上来看，2e-5是一个比较合理的数值。在该学习率下，模型可以稳定收敛且兼顾训练速度。

#### 动态学习率调节器

虽然已经确定BERT的微调学习率应控制在2e-5，但这并不意味着无法对其做进一步的调整。

在训练的过程中，保持学习率全程为一个固定值未必是最理想的选择。刚开始训练时，模型输出的loss值一般很高，梯度值较大，此时应该控制学习率较低，避免模型参数大幅抖动。待模型初步收敛后再将学习率调高。在训练接近结束时，同样应该降低学习率以避免模型参数抖动，帮助模型收敛到最优解。

为了实现以上流程，一种名为学习率调节器（Learning Rate Schedule）的实用工具被提出，并得到了广泛的应用。比如图 3.22所示的，带有预热的线性学习率调节器（Linear Schedule With Warmup）：

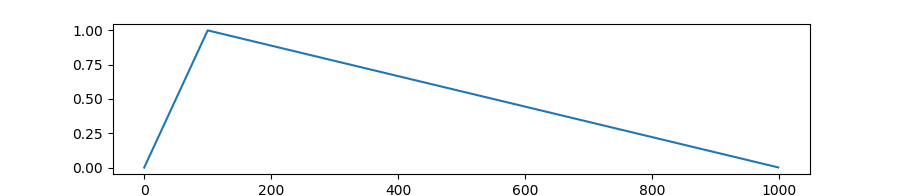


图 3.22 带有预热的线性学习率调节器的学习率调节曲线。图中横轴为步数，纵轴为相较于预设最大值的比例。

这种学习率调节器会在预热阶段控制学习率从预设最小值线性上升到预设最大值，并在之后控制学习率线性下降到预设最小值。可以简单直接地实现上文所述的理想学习率调整流程。此外还有一些改进型的学习率调节器，比如图 3.23所示的，带有预热和硬重启的余弦学习率调节器（Cosine With Hard restarts Schedule With Warmup）：

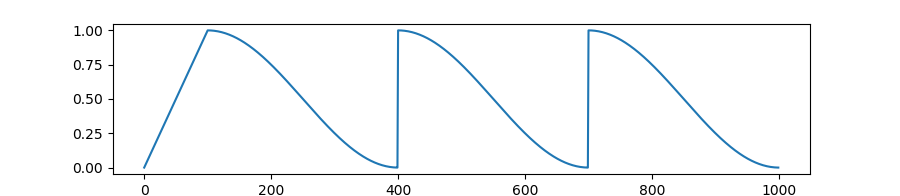


图 3.23 带有预热和硬重启的余弦学习率调节器的学习率调节曲线。图中横轴为步数，纵轴为相较于预设最大值的比例。

该学习率调节器在下降阶段采用余弦曲线，且会根据预设重启次数，在学习率最初几次下降到最小值时，重新从最大学习率开始训练。这些重启有利于使模型跨过鞍点（Saddle Point），一定程度上避免模型收敛在局部最优解。

#### 层次递减学习率

一般认为，BERT模型中靠近输入层的Encoder层包含更多的关于语言的广泛性通用知识，而靠近输出层的层包含更多与具体任务相关的知识。在微调时，我们可以对不同的层使用不同的学习率，以保留底层的通用知识，调整具体任务相关的知识。具体的实现方法，就是自顶层向底层逐层降低学习率。如式（3.20）所示，设表示第k层的学习率，则第k-1层的学习率为：

|  |  |  |
| --- | --- | --- |
|  |  | （3.20） |

其中为递减率，是一个小于1大于0的数。通过实验总结认为，当顶层学习率为2e-5，时，BERT模型有着最佳表现。

#### 融合不同层提取的特征

BERT的每一层都能捕捉到输入文本不同的特征。一般情况下，越靠近输出层的Encoder层捕捉到的特征在分类任务中表现越好。不过，仅使用最后一层输出的特征未必会有最佳的性能。

对于12层的BERT基础版而言，融合后四层输出的特征并作为分类器的输入能获得最好的性能表现。常见的特征融合方法有求均值（mean）、最大池化（max）、拼接（concat）等。三种融合方法互相间的性能差距，以及和仅使用最后一层的特征的默认分类方法的性能差距一般不是很大，具体采用哪种方法应该具体任务具体分析。

#### 长文本的处理

由于BERT位置向量的设计，使得其只能处理除了[CLS]与[SEP]符之外，最大长度为510的输入序列。虽然这个长度已经足以满足很大一部分常见任务的需求，但是对于某些长样本，或者特殊任务，仍需要设计并应用一些合理的长文本处理方法。

一种简单直接的长文本处理方法是截断法（Truncation methods），顾名思义，该方法会将过长的输入序列截断，保留一个长度为510的输入序列。截断法又可以具体细分为只保留前510个输入token的头（Head）截断法、只保留后510个输入token的尾（Tail）截断法、以及在前后分被选择一部分token的头尾（Head and Tail）截断法。

另一种处理长文本的方法被称为层次法（Hierarchical Methods）。该方法将长输入序列切分为个子序列，依次输入BERT得到k个特征表示，之后采用求均值、最大池化、自注意力等方法进行特征融合再输入分类器。

在具体任务中采用哪一种长文本处理方法，应当根据具体文本长度和数据分布规律进行具体分析。

#### 多任务微调

在拥有多个用于文本分类的数据集时，可以采用多任务微调（Multi-task Fine-Tuning）这一方法，来充分利用这些数据，充分优化BERT在文本分类任务上的表现。具体实现流程是首先在所有的数据集上都进一遍微调，然后用更小的学习率分别在目标任务数据集上微调额外的步数。

从实验结果来看，多任务微调通常都能带来性能的提升。但是多任务微调与前文提到的CDPT方法结合使用时性能提升不明显，这可能是因为经过CDPT的BERT已经获取了足够多的跨领域知识。

### BERT的对抗训练

对抗训练（Adversarial Training）这一概念，在2015年由GAN之父Ian Goodfellow提出[55]。该方法简而言之，就是在原始输入样本上加一个扰动，构造对抗样本后，用其进行训练，提高模型在遇到对抗样本时的稳健性，同时一定程度也能提高模型的表现和泛化能力。

#### FGSM与FGM

关于扰动的计算，Goodfellow认为，神经网络由于其线性的特点，很容易受到线性扰动的攻击。并基于此提出了快速梯度符号法（Fast Gradient Sign Method，FGSM）[55]，来计算输入样本的扰动。在FGSM中，扰动被定义为：

|  |  |  |
| --- | --- | --- |
|  |  | （3.21） |

其中，为输入，为目标标签，为符号函数，为损失函数，为输入层参数，表示求梯度，为放缩系数。显然，是向最大化损失的方向增加的扰动。实验发现当，时，就足以给单层分类器造成99.9%的错误率。

FGSM被提出时主要应用于CV领域。CV任务的输入是连续的RGB的值，而NLP任务中，输入是离散的单词序列。因此，将FGSM应用于BERT等NLP模型的方法是，将扰动添加在连续的向量（Embedding）层上，具体到BERT模型则是词向量（Word Embedding）层。整个对抗训练的流程为：

对于每个输入：

（1）. 正常进行前向运算，根据输出反向传播，求词向量层的梯度。

（2）. 备份当前词向量层参数。根据公式计算出，然后与当前的词向量层参数相加。

（3）. 前向计算添加了扰动的词向量层输出的对抗样本的输出，并根据输出反向传播，求词向量层的梯度，并累加在（1）的梯度上。

（4）. 恢复（2）中备份的词向量层参数，并根据（3）的梯度进行更新。

在以上过程中生成的对抗样本并不对应某个具体的单词，无法通过修改原始输入得到恶意样本。因此，在NLP任务中，对抗训练的作用不再是为了防御基于梯度的恶意攻击，反而更多的是作为一种正则化，提高模型的泛化能力。

2017年，FGSM的作者提出了该方法的改进版本，快速梯度算法（Fast Gradient Method，FGM）[56]。将计算的过程改为：

|  |  |  |
| --- | --- | --- |
|  |  | （3.22） |
|  |  | （3.23） |

式中为梯度，表示的第二范数正则化。

FGM与FGSM相比，取消了符号函数，通过对梯度进行合理放缩以计算扰动。除此之外，FGM的对抗训练流程与FGSM没有区别。

FGM与FGSM对抗训练每一步都需要两轮前向与后向计算，其运算量大致是正常训练的两倍，在实际应用时要注意时间成本。

#### PGD

FGSM与FGM通过参数一次性算出了对抗扰动，这样得到的可能不是最优的。对此，有研究者提出了投影梯度下降（Projected Gradient Descent，PGD）方法[57]。PGD计算扰动的方法可以概括为：“小步走，多走几步”，如果走出了扰动半径为的球形空间，就映射回“球面”上，以保证扰动不要过大。用公式表示为：

|  |  |  |
| --- | --- | --- |
|  |  | （3.24） |

式中为第t步的梯度。若某步之后，则

使用PGD方法进行对抗实验的流程为：

对于每个输入，以及预设总步数T：

（1）. 正常进行前向运算，根据输出反向传播，求词向量层的梯度。同时备份当前词向量层参数与梯度。

对于每步t：

（2）.根据公式计算出。若走出了预设球形空间，则映射回“球面”上。然后与当前的词向量层参数相加。

（3）.若t不是最后一步，则将梯度归零。 前向计算添加了扰动的词向量层输出的对抗样本的输出，并根据输出反向传播，求词向量层的梯度，然后回到（2）。

（4）.若t是最后一步，则恢复（1）中备份的梯度。前向计算添加了扰动的词向量层输出的对抗样本的输出，并根据输出反向传播，求词向量层的梯度，并叠加在当前的梯度上。

（5）. 恢复（1）中备份的词向量层参数，并根据（4）的梯度进行更新。

PGD方法可以通过多次迭代，找到最优的构造对抗样本的扰动，但是其多轮迭代的流程也使得其运算量大约是正常训练的倍。在实际使用时应当控制T的值，综合考虑时间成本。

### 文本多分类评价标准

在二分类问题中，假设该样本一共有两种类别：Positive和Negative，则分类结果可以分为如下四种：True Positive (TP)、True Negative (TN)、False Positive (FP)、False Negative (FN)。

同样的评价标准也可以扩展到多分类场景，根据混淆矩阵分别对各类的分类结果进行统计，如式（3.25）所示，其中n为类数量。

|  |  |  |
| --- | --- | --- |
|  |  | （3.25） |

表格

低可信度描述已自动生成

图 3.24 多分类任务中TP、FP、TN、FN的定义（左）与混淆矩阵举例（右）

针对某一类而言，常用的评价标准Accuracy，Precision，Recall和F1 score的定义为：

|  |  |  |
| --- | --- | --- |
|  |  | （3.26） |
|  |  | （3.27） |
|  |  | （3.28） |
|  |  | （3.29） |

之后要对各类的评价结果进行统一，常见的方法有以下三种。

第一种是宏平均（Macro-average）方法。该方法直接对各类别的评估指标（Precision/ Recall/ F1-score）求平均，所有类别的权重相等。该方法的值易受罕见类的影响，一般用于各类别重要性相等的情况。

第二种是加权平均（Weighted-average）方法。该方法是宏平均方法的变体，给不同类别分配不同权重（权重根据该类别的真实分布比例确定），每个类别的评估指标乘以权重后再进行相加。该方法考虑了类别不平衡情况，但它的值更容易受到常见类（majority class）的影响。

第三种方法是微平均（Micro-average）方法。该方法将所有类别的TP,、FP、FN、FN先相加，然后直接计算最终的评估指标。这种方法明显会忽略稀有类，同样易受常见类影响。

一般情况下，宏F1分数（Macro-F1）是最能反映模型综合性能的评价指标，该指标也是最常用的指标。对于预期应用场景存在数据类不平衡的情况，也可以使用加权F1分数（Weight-F1）为平均指标。微平均则一般用于类别数量较少且比例相近的情况。

### 改进型损失函数

损失函数（Loss function）是衡量模型预测值与真实值之间差异程度的一种数学函数。损失函数值越小，意味着模型能够更准确地拟合数据样本，通常模型的性能越好。在训练阶段，一般将训练数据划分成一个个批次，对于每个批次的数据，使用模型进行预测，并通过损失函数计算出预测值和真实值之间的损失值。然后，使用反向传播算法计算出梯度，根据梯度更新模型参数，使得模型的预测能力能够逐步提高。不同的模型与任务用的损失函数一般也不一样。这里以分类任务常用的交叉熵（Cross-entropy，CE）损失函数，又称负对数似然损失（Negative Log Likelihood，NLL）函数为例，其公式如（3.30）所示。

|  |  |  |
| --- | --- | --- |
|  |  | （3.30） |

其中，N为样本的数量。M为类别的数量；为符号函数，如果样本i真实类别为c则取1，否则取0. 为模型输出的样本i为类别c的概率。

交叉熵是一种用于衡量两个概率分布之间差异程度的度量方法，它可以在信息论和机器学习中都有着广泛应用。在信息论中，交叉熵被用于衡量使用不同概率分布进行编码所需要的平均编码长度。在机器学习中，交叉熵被用作损失函数来评估模型预测与真实标签之间的相似度。交叉熵的值越小，表明模型预测结果与真实标签之间的误差越小，模型的性能也就越好。

交叉熵损失函数在用于二分类问题时时，一般用sigmoid函数作为分类器输出函数。在多分类时，一般用softmax函数作为分类器输出函数。softmax函数是sigmoid函数的扩展，如公式（3.31）所示：

|  |  |  |
| --- | --- | --- |
|  |  | （3.31） |

其中输入x为一个向量，表示x中的一个元素。Softmax函数可以将x的原始数据进行归一化，转化为一个之间的数值，这些数值可以被当做概率分布，用来作为多分类的目标预测值。

CE损失函数对所有样本和类别都一视同仁，不管当前样本是简单还是复杂，也不管某类别是常见类还是罕见类。这使得模型的训练将被简单样本和常见类主导，难以从复杂稀有样本中学习。

一种直观的缓解不平衡训练数据对模型性能影响的方法，是给损失函数中添加权重因子，提高少数类别在损失函数中的权重。具体的实现被称为加权交叉熵损失函数（Weighted Cross-entropy）。如下公式（3.32）所示：

|  |  |  |
| --- | --- | --- |
|  |  | （3.32） |

其中权重系数可以根据各类别的出现频率求负权重得到。

另一种从改善难分类样本的处理的角度，对CE损失函数进行改进的方法被称为Focal Loss（FL）[58]。如公式为（3.33）所示：

|  |  |  |
| --- | --- | --- |
|  |  | （3.33） |

式中为调制因子（Modulating Factor），为大于0的放缩系数。

当样本i真实标签为c，且趋向于1时，样本i为已准确分类样本，趋向于0，减少了该样本对损失值的贡献。当样本i真实标签为c，但趋向于0时，样本i为分类不准确样本，趋向于1，保留了该样本对损失值的贡献。放缩系数越大，则易分类样本的调制因子相对越小。

FL函数使得模型聚焦于那些难分类样本，同时还可以通过加强稀有类样本的权重，缓解训练数据不平衡的问题，有助于模型的整体性能提高。

近年来，又出现了一种专门用于缓解大量NLP任务中的数据不平衡问题，能够提高基于F1分数的Dice Loss[59]。如公式（3.34）所示，该损失函数的核心是Sørensen–Dice系数，在分类任务中等效于F1分数。

|  |  |  |
| --- | --- | --- |
|  |  | （3.34） |

式中是与FL中相同的调制因子，是为了防止分母为0的平滑因子。剩余的部分等效于软化后的F1分数。

Dice Loss在许多数据集与任务上都表现出了较明显的F1分数提升，但是其可能造成精度的下降，这是因为Dice Loss会倾向于保持类间的“平衡”，而不是笼统地考虑所有的数据。

### 混合精度训练

浮点（Float Point）数据类型主要分为双精度（FP64）、单精度（FP32）、半精度（FP16）格式。在传统神经网络模型的训练过程中，一般默认采用FP32浮点数据类型，来表示网络模型权重和其他参数。混合精度训练[60]是在尽可能减少精度损失的情况下，利用半精度浮点数加速训练。它使用FP16即半精度浮点数存储权重和梯度，在运算时采用FP32权重备份（Weight Backup），FP32累加（Accumulated）损失缩放（Loss Scale）等方法，在减少占用内存的同时起到了加速训练的效果，并且尽可能避免舍入误差(Rounding Error)与数值溢出(Grad Overflow / Underflow)导致的精度损失。自动混合精度（Automatic Mixed Precision）训练，指的是是框架会根据当前进行的运算，自动对数据类型进行调整，启动混合精度运算。

BF16，有时也被称为BFloat16或Brain Float16，是一种针对人工智能或深度学习应用程序进行优化的新数据格式。Google Brain团队在长期研究和实验中发现，神经网络对指数的大小比尾数敏感得多，并基于此结论提出了BF16数据格式[60]。与FP16相比，BF16牺牲了部分精度，获取了更大的数值空间（Dynamic Range），更适合深度学习的应用场景。目前，BF16在Google第三代Tensor Processing Unit（TPU），以及Intel、Nvidia、Arm和许多其他公司的人工智能加速器上得到了广泛应用，已经成了公认的替代FP16的混合精度运算新数据格式。

图形用户界面, 应用程序

描述已自动生成

图 3.25 FP32、FP16、BF16格式对比

## 实验部分

### 实验环境

本次实验硬件环境为AutoDL AI算力云平台租赁的云GPU服务器，硬件配置为：

* + - CPU：15 vCPU Intel(R) Xeon(R) Platinum 8358P CPU @ 2.60GHz
    - GPU：RTX 3090
    - 内存：80G

软件环境为docker容器，容器对应的镜像内主要软件及版本为:

* + - 操作系统：ubuntu 20.04
    - Cuda版本：11.3
    - Python版本：3.8
    - Pytorch版本：1.11.0

使用的主要工具库的版本为：

* + - transformers版本：4.24.0
    - datasets版本：2.6.1
    - accelerate版本：0.15.0
    - ltp版本：4.2.13

### Hugging Face平台介绍

在使用Bert或其他预训练模型时，预训练参数与对应的模型文件，以及相关的分词器等周边工具的来源极为重要。本文采用了Hugging Face公司开源的transformers预训练模型库，以及其同名开源社区中公开的预训练参数。Hugging face最初是一家总部位于纽约的聊天机器人创业公司，目前致力于NLP技术的平民化(democratize)，维护了大型开源NLP社区，提供海量的预训练模型参数与数据集。目前，由transformers库与社区内其他开源工具库组成的Hugging face平台已经成为了NLP领域默认的开发与分享标准。其整体代码风格偏向于科研与学习，在各大期刊的文献中都有应用，也是NLP研究者首选的成果发布与开源平台。但是这种倾向也使得transformers库的运行效率受到了影响，在用于实际生产环境时，一般要转换为ONNX等通用模型格式，再部署到其他专业平台。

本文实验使用的BERT模型（BertModel）与分词器（BertTokenizer）均来自transformers库，预训练模型参数来自Hugging face Hub。datasets库与accelerate库同样来自于Hugging face平台，分别用于数据预处理和加载，以及简化模型训练过程。

### 数据集介绍

本文实验采用的数据集名为Chinese Scientific Literature Dataset（下文简称CSL），

是首个大规模中文科学文献数据集。数据获取自[国家科技资源共享服务工程技术研究中心](https://nstr.escience.net.cn/)，包含 2010-2020 年发表的期刊论文元信息（标题、摘要和关键词）。根据中文核心期刊目录进行筛选， 并标注学科和门类标签，分为13个门类（一级标签）和67个学科（二级标签）。最近更新时间为2022年，总数据条数约40万条。单数据结构如图 4.1所示：

该数据集已被权威中文任务测评基准网站CLUE benchmark收录，为排行榜任务-论文关键词识别（keyword Recognition）的标准数据集，质量受到了广泛认可。

本文将采用其中每条数据的标题（Title）、摘要（Abstract）和关键字（Keywords）为模型输入（input），13个门类（category）为分类目标（label），进行文本分类任务。

图形用户界面, 文本, 应用程序, 电子邮件

描述已自动生成

图 4.1 CSL数据集数据样本格式

#### 数据清洗

本实验在下载数据集时，选择了数据集论文中网址提供的原始tsv文件。下载后文件共两份，一份文件约396k行，是数据集总集。另一份文件约40k行，从前者中随机提取得到。经过粗略的初步人工筛查，并总结后续测试中出现的一些错误，发现数据集中存在以下问题并依次予以修正：

部分数据行存在数据项类型错误、数据项缺损、数据内容过短或残缺，判断为损坏数据，直接删除。

部分数据中存在2个或更多偶数个@符号组成的乱码（@@），去掉其中的乱码后，其中大部分数据可以挽救。其中少部分数据整个数据项都被乱码填充，去掉乱码后会出现数据项缺损，该部分数据无法挽救，直接删除。

最终在40k文件中删除了44行，396k文件中删除了380行

在后续的观察与实验中，发现了大量引数据集中存在英文引号（"）错误使用导致的问题，具体问题与解决方法如下：

读取tsv文件使用的pandas工具库对中文的支持存在兼容性bug，当英文引号（"）和中文字符连续出现在tsv文件中的时候，即使手动设置了以制表符（\t）为分隔符读取文件，pandas依旧有可能按照csv文件的标准，将英文引号误认为分隔符造成数据读取错误，在保存时也有可能出现1个英文引号以后变成2个（"”）的情况。对此需要在调用读取和保存函数时，手动设置pandas忽略英文引号（"）的特殊含义，直接原样读取或保存。

部分数据行中存在连续出现的多个英文引号（"），且大多出现在句首句尾。多次执行字符串替换函数，将多个英文引号缩减为一个，即可修复这一问题并保留正确语句。

部分数据行存在整个数据项字段两端同时存在英文引号的现象，这种格式与csv文件默认格式非常相似。删除两端多余的英文引号即可恢复数据。

部分数据行中有英文引号（"）和中文引号（“”）错误混用的问题，比如英文引号搭配中文右引号（”）或者中文左引号（“）搭配英文引号，且大多出现在句首句尾。通过正则表达式匹配，找出存在这一问题的数据行，并用正确的另一半中文引号替换掉英文引号即可解决这一问题。

个人推测这一系列引号相关的问题发生的原因是原作者在整理该数据集的时候，部分原始数据是以英文以号为分隔符的csv格式。但是正常的文本数据当中也存在英文引号，会导致分隔符错误，因此强行将原始数据写入以制表符（\t）为分隔符的tsv文件，而原本csv文件中的英文引号分隔符并没有被合理地清除。这些问题的出现规律都比较明显，格式也比较规整统一，处理的难度并不高。

由于pandas工具库对中文的支持存在兼容性bug，除了保存英文引号时会出现问题，有时也会出现一个空格保存为多个的问题，为确保数据清洗彻底，本人之后又使用VS code编辑器的查找替换功能，对连续出现的引号或空格进行了清理和检查，确保数据清洗彻底。

#### 数据集分析

本次实验的13个目标类在数据集中的分布如图 4.2所示：

可以看到数据分布存在明显的不均衡现象，主要表现为存在一个常见类。考虑到本次数据集的来源是中科院权威平台，以及我国科学研究长期偏重于工程应用的特点，可以认为这种数据不均衡现象是符合现实的。换而言之，本文实验中设计的中文期刊论文分类系统，如果应用在实际生产场景中，预期的输入数据的类别分布规律大概率相同。因此本文相关实验将在测试集中不对这种数据不平衡现象进行处理，保留其原始类别分布比例。同时在评价模型性能时采用加权F1分数。

图表, 饼图

描述已自动生成

图 4.2 CSL数据集类别分布图

将每条数据的标题（Title）、摘要（Abstract）和关键字（Keywords）字段拼接，中间加入[PAD]符隔开，使用BertTokenizer转化为输入序列，并对长度进行统计，结果图 4.3和图 4.4所示：

可以看到输入序列的长度整体上符合左偏正态分布。40k数据集的平均长度为241.2，396k数据集的平均长度为240.6。仅有约1.9%的输入序列长度超过512。考虑到绝大部分输入序列的长度都在BERT模型可以处理的范围内，本文的实验将不会采取过于复杂的长文本处理方法，对于长度超过512的样本将直接采取截断处理。

### 实验一 基准实验

#### 实验目的

实验的主要目的是对常用中文BERT模型进行筛选以及评估基准性能。在对原始模型不做任何修改的前提下，通过反复测试，总结最优超参数以及最佳流程。同时获取模型对计算的空间资源与时间资源的占用情况，为后续实验的成本评估提供参考。实验中各模型的最佳性能，将作为后续实验的基准性能，起到对照组的作用。

#### 实验模型介绍

本实验选择的各组预训练参数如下：

bert-base-chinese：Google发布的原版BERT模型，训练语料为中文维基，总词数约0.4B。

chinese-bert-wwm：哈工大讯飞联合实验室发布的使用WWM方法训练的BERT模型，训练语料为中文维基。

chinese-bert-wwm-ext：哈工大讯飞联合实验室发布的使用WWM方法训练的BERT模型。其中ext表示训练语料extend数据集，包括：中文维基百科，其他百科、新闻、问答等数据，总词数达5.4B。

chinese-roberta-wwm-ext：哈工大讯飞联合实验室发布的使用WWM方法训练的RoBERTa模型。其中ext表示训练语料extend数据集。

chinese-roberta-wwm-ext-large：哈工大讯飞联合实验室发布的使用WWM方法训练的RoBERTa模型Large版。其中ext表示训练语料extend数据集。参数量约325M，属于大参量模型。

rbt6：哈工大讯飞联合实验室发布的使用WWM方法训练的仅6层Encoder的RoBERTa模型。参数量约60M，属于小参量模型。

图表, 直方图

描述已自动生成

图 4.3 CSL 40k数据集输入序列长度统计。图中横轴为序列长度，纵轴为数量。

图表, 直方图

描述已自动生成

图 4.4 CSL 396k数据集输入序列长度统计。图中横轴为序列长度，纵轴为数量。

#### 实验内容

本次实验设置了（128，256，512）这3个档位的输入序列长度，分别对比个模型在这3个长度下的表现。如前文所说，中文Roberta将采用与中文BERT完全相同的字典（vocab），分词器（BertTokenizer）与模型（BertModel）。因此本实验模型全部使用Hugging Face提供的BertForSequenceClassification模型。该模型是对BertModel的进一步封装，加入了一层768维到num\_labels维（本实验中为13维）的线性层（nn.linear）为输出层，用于将BertModel最后一层输出的[CLS]字符对应的词向量映射到分类结果。并且封装了交叉熵损失函数（CrossEntropyLoss），便于评估和训练。

微调数据集为CSL 40k，将每条数据的标题（Title）、摘要（Abstract）和关键字（Keywords）字段拼接，中间加入[PAD]符隔开，使用BertTokenizer转化为输入序列，并将加入的[PAD]符的attention\_mask标记为0，使其不参与注意力运算。采用datasets库提供的dataset类实现对数据集的保存与加载，并将微调数据集按照9:1随机拆分为训练集和测试集。

#### 实验结果

通过大量测试，最终总结的最优超参数为学习率2e-5，加入线性下降（Linear Schedule）学习率调节器，并且开启BF16自动混合精度运算。各模型在三个预设输入序列长度下的最佳训练轮数epoch和训练步数step，以及batch\_size如下表 4.1所示：

表 4.1 各模型在三种输入序列长度下的训练轮数epoch、batch\_size、训练步数step

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 模型 | 句长 | epoch | batch\_size | step |
| bert-base-chinese  chinese-bert-wwm  chinese-bert-wwm-ext  chinese-roberta-wwm-ext | 128 | 16 | 128 | 4480 |
| 256 | 8 | 64 | 4488 |
| 512 | 4 | 32 | 4492 |
| chinese-roberta-wwm-ext-large | 128 | 8 | 64 | 4488 |
| 256 | 4 | 32 | 4492 |
| 512 | 1 | 8 | 4494 |
| rbt6 | 128 | 16 | 128 | 4480 |
| 256 | 8 | 64 | 4488 |
| 512 | 4 | 32 | 4492 |

各模型对显存的占用情况如图 4.5所示：

各模型每轮训练消耗的时间如图 4.6所示：

各模型的基准性能如图 4.7所示：

各模型对显存的占用和训练所需的时间多随输入序列的增加而增加，同时各模型在输入序列更长时性能更优。大部分模型在输入长度为512时表现最佳，也有部分模型在输入序列为256时表现最佳。这与前文分析的数据集文本长度分布规律相吻合。基于此，之后的实验将不再涉及输入序列长度为128的情况。

图表, 条形图

描述已自动生成

图 4.5 各模型的显存占用情况

图表, 条形图

描述已自动生成

图 4.6 各模型单轮训练耗时

使用了WWM预训练方法的两款bert模型整体表现均优于原版bert。chinese-bert-wwm-ext与chinese-roberta-wwm-ext预训练数据集更大预训练更充分，理论上应当优于chinese-bert-wwm，但在本次实验中却表现相反。原因是ext数据集中新增的大部分数据偏向于日常用语，而中文维基数据集相比之下以正式书面文字居多。在wwm系列模型的原作者的github中也能找到实验支持这一结论。

chinese-roberta-wwm-ext的性能并没有优于三款原版BERT模型，甚至略低。查阅原作者的github中提供的预训练参数发现，该模型在MLM预训练使采用了和其他BERT模型一样的batch size和预训练步数，并没有完全实现RoBERTa模型的改进，仅相当于chinese-bert-wwm-ext模型去掉了NSP任务，其性能表现不佳是可以解释的。

图表, 条形图

描述已自动生成

图 4.7 各模型基准性能

chinese-roberta-wwm-ext-large的性能理论上更强，但是在本次实验中并明显展示，推测是微调batch\_size过小，不能起到较好的训练效果。该模型较大的参数量使得其对计算资源的占用也很大，单张3090的算力已经难以以支撑进一步尝试，每个epoch的训练时间已经超出了可接受的范围，因此数据仅留作对照组，之后的实验将不再涉及该模型。

rbt6作为小参量模型，对显存的占用明显更小，训练所需的时间也明显更短。但是小参量模型以及蒸馏、剪枝等问题不是本文的主要研究方向，该模型实验数据仅留作对照组，之后的实验将不再涉及该模型。

### 消融实验一 BF16自动混合精度训练

本实验中维持所有超参数及其他设置与实验一中相同，以bert-base-chinese与chinese-bert-wwm为例，关闭BF16混合精度运算，观察训练过程计算资源占用情况的变化，以及最终的性能。

在实验过程中发现，关闭BF16混合且输入序列长度为512时，batch\_size=32会导致显存溢出。最终该项实验设置batch\_size=24，训练步数也因此增加到5992步。

实验结果如图 4.8和图 4.9所示：

需要注意，实验结果图中bert-base-chinese-fp32在输入序列长度为512的情况下，batch\_size只有bert-base-chinese-bf16的3/4，该项实验数据参考价值较低。

很显然，BF16混合精度训练在节约显存占用和减少训练时间上都有显著的效果，并且没有对模型性能造成明显影响。

考虑到BF16混合精度训练已经成为深度学习领域公认的新标准，因此本文只进行了两个模型的代表性实验，验证这一结论。

图表, 条形图

描述已自动生成

图 4.8 bert-base-chinese在开启与关闭BF16混合精度运算的情况下的显存占用与单轮训练耗时。其他模型表现基本相同。

图表, 条形图

描述已自动生成

图 4.9 bert-base-chinese与chinese-bert-wwm在开启与关闭BF16混合精度运算的情况下性能

### 消融实验二 灾难性遗忘

本实验在实验一的基础上，将学习率修改为4e-4，同时将去掉学习率调节器，保持学习率为恒定值。其他条件相同的情况下，对chinese-bert-wwm模型进行微调，整个微调过程中的loss函数值变化如图 4.10所示：

而实验一中loss函数值的变化图为图 4.11：

可以观察到，在学习率为4e-4的整个训练过程中，模型完全没有收敛，出现了明显的灾难性遗忘现象。

图形用户界面

描述已自动生成

图 4.10 chinese-bert-wwm模型在4e-4学习率下微调的loss变化曲线。图中横坐标为训练步数，纵坐标为loss值

图形用户界面, 图表

描述已自动生成

图 4.11 chinese-bert-wwm模型在2e-5学习率下微调的loss变化曲线。图中横坐标为训练步数，纵坐标为loss值

图表, 条形图

描述已自动生成

图 4.12 各模型在减少了一半训练轮数后的性能

### 消融实验三 最佳训练epoch

本实验在实验一的基础上将所有模型在两种可能的最优输入序列长度的情况下的训练epoch减小为一半，尝试是否可以进一步节约计算资源。实验结果如图 4.12所示。

可以看到更小的epoch所有情况下都没有使模型充分收敛，而更大的epoch成本太高，同时在前期测试中已经被证明提升不明显。实验一中选择的epoch已经是权衡各种利弊后得到的最优解。

### 消融实验四 动态学习率调节器的选择

本实验在实验一的基础上，尝试了多种其他学习的调节器，尝试得到动态学习率调节器到最佳选择。

本实验测试了以下几种动态学习率调节器：

constant：维持学习率为恒定值，相当于去掉学习率调节器。

linear：线性下降学习率调节器，与实验一中相同。

linear\_wramup：带有预热的线性学习率调节器，设定训练总步数的前10%为预热部分。

consine\_restart：带有重启的余弦下降学习率调节器，在训练步数一半时有一次重启。

实验结果，如图 4.13和图 4.14所示：

首先可以显著观察到的一点是，各模型在不加入学习率调节器的情况下，这种表现都明显差于加入学习率调节器的情况。

带有预热的线性下降学习率调节器的表现整体上弱于不预热的情况，原因可能是由于预热阶段的主要目的是为了防止刚开始训练的模型发生大幅抖动，帮助模型初步收敛。而本实验中采用了预训练模型和较小的学习率，模型本身就已经初步收敛，因此预热阶段没有起到作用。

线性下降学习率调节器和带有重启的余弦下降学习率调节器的表现整体上比较接近，考虑到让后续实验有更好的可解释性，本文最终选择了只保留一款最简单的线性下降学习率调节器。

图表, 条形图

描述已自动生成

图 4.13 各模型在各动态学习率调节器下的性能表现（1）

图表, 条形图

描述已自动生成

图 4.14 各模型在各动态学习率调节器下的性能表现（2）

### 实验二 层次递减学习率

#### 实验目的

本实验的目的是验证层次递减学习率这一微调技巧在本文的研究中是否有效

#### 实验内容

本实验参考文献中给出的最优解，在学习率2e-5的情况下，设置每层衰减率为0.95，同时设置最后的线性层分类器学习率为1e-4。

#### 实验结果

实验结果如图 4.15所示，其中不加入层次递减学习率直接的情况，实验数据直接沿用实验一的基准数据。

可以看到在输入序列长度为256的情况下，层次递减学习率整体上并没有较好的表现。但是在输入序列长度为512的情况下，层次递减学习率对各模型都有显著的性能提升。

综合分析全部实验数据，可以看到在输入序列长度为512且使用层次递减学习率的情况下，所有模型是最优表现。之后的实验也将沿用该实验条件。

图表, 条形图

描述已自动生成

图 4.15 加入层次递减学习率后各模型的性能

### 实验三 截断方法

#### 实验目的

本实验将在实验二的基础上进一步探究在输入序列长度为512的情况下，对于占比约1.9%的长度超过512的样本的截断方法。

#### 实验内容

在实验二中采用了BertTokenizer默认的头截断法，即只保留前512个token。本实验中新增了头尾截断法组，对于长度超过512的序列，保留前256和后256个token。由于本实验的输入序列开头是待分类论文的标题和关键字等重要信息，显然不适合尾截断法，因此没有进行相关实验。除此之外的超参数与其他设定与实验二相同。

#### 实验结果

实验结果如图 4.16所示，其中head的组沿用实验二的数据。

对于bert-base-chinese模型而言，头尾截断法表现更好。对于三款wwm预训练的模型而言，头截断法表现更好。考虑到本文主要研究更适用于中文的wwm预训练模型，且头截断法是BertTokenizer的默认方法，实现更简单。因此在之后的实验中，都将继续沿用头截断法。

图表, 条形图

描述已自动生成

图 4.16 采用头尾截断法后各模型的性能

### 实验四 后四层特征的融合方法

#### 实验目的

本实验将探究将模型后四层提取的句子特征进行合理的融合，并作为分类器的输入的最佳方法。

#### 实验内容

在BertModel的配置类BertConfig中设置output\_hidden\_states为True，即可使模型输出全部12层提取的特征。从中取出后四层的特征，并进一步取出[CLS]符对应的特征，并进行融合。本次实验设置了求均值（mean）、最大池化（max）、拼接（concat）三种融合方法进行对比。

#### 实验结果

实验结果如图 4.17所示，其中default项沿用实验二的数据。

对于bert-base-chinese模型而言，mean与max两种融合方法都使得模型有更好的表现。但是对于三款wwm预训练的模型而言，各种特征融合方法对模型都没有明显性能提升。考虑到本文主要研究更适用于中文的wwm预训练模型，之后的实验不会再应用这些特征融合方法。

图表, 条形图

描述已自动生成

图 4.17 各模型在应用各特种融合方法后的性能

### 实验五 改进型损失函数对比

#### 实验目的

如前文所说，本文实验的数据存在明显的分布不均衡问题，主要表现为有一个常见类。本实验将尝试使用改进型损失函数对这一问题进行处理。

#### 实验内容

由于本文采用的是预训练模型，不能对模型结构做过多修改。而且学习率等超参数已经确定为前人总结的最优解。因此，可能的改进方向除了对训练数据进行平衡处理外，就是对损失函数进行一些改进，使得模型在数据分布不均衡的情况下，依旧能对所有样本进行充分的学习。

实验的数据平衡处理，采用的方法是消除常见类。将原本微调数据集中的工程类样本的数量从17906条删除到和样本数量第二名一样的3587条。由于训练样本数量减少，该实验扩大到6个epoch使训练步数为4402步，和原本的4492步基本相等。

另外，本实验尝试了Weighted Cross-entropy Loss，Focal Loss，Dice Loss三款具有较强的处理数据分布不均衡的能力的改进型损失函数，并对这些函数的表现进行对比分析。

#### 实验结果

实验结果如下图所示，default项采用的是Cross-entropy Loss，沿用实验二的数据。

图表, 条形图

描述已自动生成

图 4.18 各模型在使用了平衡训练数据集或者各改进型损失函数后的性能

很显然，对数据集进行平衡处理这一方法，不仅没有使得模型能充分学习其他少见类的知识，甚至损害了模型原本已经掌握的关于常见类的知识。

Dice Loss同样对模型的性能造成了一定损害，通过分析图 4.19的loss值曲线，可以看到Dice Loss的损失值一直较高（0.3以上），模型整体上并没有较好的收敛。为了解决这一问题，本实验又尝试了。将Cross-entropy Loss与Dice Loss组合使用，取各自函数值的一半作为最终损失值。但最终没有性能的改善，组合损失值同样一直维持在高位。就实验结果而言，Dice Loss不适合本次任务。

图表

描述已自动生成

图 4.19 chinese-bert-wwm模型的Dice loss变化曲线。图中横坐标为训练步数，纵坐标为loss值

Weighted Cross-entropy Loss和Focal Loss的表现，整体上和Cross-entropy Loss比较接近，但是性能仍然有所下降。对于这一结果，个人认为合理的解释是这样的：学术论文这种文本本身就有可能同时属于两个甚至多个类别，即交叉学科。而本文在评价分类结果时，只采纳了预测概率值最高的一个类别，在这种情况下的难分类样本即使被模型充分学习，依旧很难被确定类别。而这些能够迫使模型更多的关注少见类和难分类样本的损失函数，反而导致了模型对常见类的忽略，造成了一定的性能下降。如果采用类似于top2预测概率值的评价标准，也许就能观察到模型上整体更好的表现。

综合以上实验结果与分析过程考虑，之后的实验将继续采用Cross-entropy Loss。

### 实验六 对抗训练

#### 实验目的

本实验尝试将对抗训练引入本文的研究任务中，通过增强模型的泛化能力，提升模型的性能。

#### 实验内容

本实验中分别实现了适用于FGSM、FGM、二阶PGD三种对抗训练方法的BERT模型，以及相关的训练函数。同时记录了三种对抗训练方法各自对运算资源的消耗情况。

#### 实验结果

三种对抗训练方法每个epoch的耗时，如图 4.20所示：

图表, 条形图

描述已自动生成

图 4.20 bert-base-chinese在应用各对抗训练方法后单轮训练耗时

各模型的性能表现如图 4.21所示：

各对抗训练方法对计算资源的消耗量，大致和其前后向运算次数成正比。

整体上来讲，二阶PGD方法的表现优于FGM，FGM优于FGSM。这与三种方法各自获取扰动值的细致程度和准确程度相吻合。但是如实验五中分析的那样，由于本文研究任务的特殊性，更强的泛化能力未必能非常显著地展现出来。

考虑到对抗训练计算资源的消耗量比较大，因此本文之后的实验将不再采取对抗训练。只在最终部署的时候通过对抗训练进行加强。

图表, 条形图

描述已自动生成

图 4.21 各模型在应用各对抗训练方法后的性能表现

### 实验七 MLM增量预训练

#### 实验目的

本实验将尝试使用MLM ITPT提升各模型在本文研究任务上的表现，同时对比测试WMM方法用于增量与训练表现。

#### 实验内容

本次实验设计了两种语料长度的增量预训练数据集。一种被称为short数据集，通过将原数据样本的摘要部分，使用ltp工具拆分为句子，并且加入原数据样本的摘要和关键字组成。摘要单独为一个句子。关键词拼接为一个句子，中间加入[PAD]符隔开。最终得到的数据集有2050062行，平均长度为51个字符。

另一种被称为long数据集，和前文的微调数据集类似，通过将原数据样本的标题，关键字和摘要拼接起来，中间加入[PAD]符隔开，作为一个句子。最终得到的数据集有395829行，平均长度为277个字符。

short与long两个数据集之后都会通过BertTokenizer转化为输入序列，预设最大长度分别为128和512，同时将加入的[PAD]符的attention\_mask标记为0，使其不参与注意力运算。

在对bert-base-chinese进行MLM增量预训练时，将采用和原版bert相同的样本生成方法。在对三款采用了WMM预训练的模型进行MLM增量预训练时，将采用WMM预训练方法。子词Mask比例将沿用默认的15%

本实验的模型采用transformers库提供了BertForMaskedLM模型，通过对BertModel的进一步封装，可以实现MLM任务。同时采用了DataCollatorForLanguageModeling工具类，实现对输入序列的动态Mask。在WWM的MLM任务中，则还需要搭配ltp工具和prepare\_ref函数，先对语料进行一次中文分词，根据中文分词结果与原句的子词序列逐个比对，标记那些在WWM时需要一起Mask的子词，保存在名为chinese\_ref的数组中，和输入序列一起经过DataCollatorForWholeWordMask类，处理为用于WWM的训练样本。

两种语料长度的增量预训练都设置batch\_size=32，与模型发布者在预训练阶段采用的384相比这是一个很小的值，因此学习率采用了发布者给出的推荐值中最小的2e-5。

short训练的训练集长度为2050062行，训练20万步，约4个epoch，显存占用7323MB，耗时约10个小时。long训练的训练集长度为395829行，训练20万步约17个epoch，显存占用22875MB，耗时约23小时21分，启用WWM后耗时约25小时13分钟。

#### 实验结果

本次实验由于成本原因，没有进行较为密集的数据采样，在增量预训练的过程中每隔25000步保存一次检查点，并评测模型此时在下游任务上的性能。为了更直观地分析这些稀疏的数据点，采用了二次多项式进行拟合。实验结果如图 4.22和图 4.23所示。

图表

描述已自动生成

图 4.22 使用short语料进行MLM任务增量预训练后各模型的F1分数变化趋势

图表

描述已自动生成

图 4.23 使用long语料进行MLM任务增量预训练后各模型的F1分数变化趋势

本实验最开始只训练了20万步，观察到无论是在short数据集上还是long数据集上的增量预训练，各模型的性能都还没有明显的达到峰值的迹象。但是已经可以观察到使用long语料的增量预训练效果明显好于short，因此之后仅将使用long语料的增量预训练增加了10万步，约8个epoch。接下来的分析也将基于这30万步使用long语料的增量预训练的实验的相关数据。

首先从图中可以直观地看到，MLM ITPT这款模型的性能表现都有着显著的提升。bert-base-chinese、chinese-bert-wwm-ext、chinese-roberta-wwm-ext在大约20万步的时候性能到达峰值。根据文献给出的数据，各模型的性能应当在10万步左右达到峰值。但是本次实验中采用了2e-5这一较小的学习率，大致是BERT模型预训练常用的学习率3e-5到5e-5的一半，因此各模型性能的峰值出现在20万步左右。

chinese-bert-wwm模型的性能一直呈现上升趋势，这可能是因为该模型的WWM MLM预训练本身就没有另外两款采用了ext数据集的模型那样充分。虽然bert-base-chinese 的预训练情况与其相同，但WWM MLM预训练难度更高，需要更多的步数才能模型学习到饱和。

三款采用了WMM预训练的模型，最初的性能都要优于原版BERT，采用WWM方法ITPT性能反而明显弱于原版BERT。出现这种现象的原因可能有很多种，比如本次实验采用的数据集专业性太强或质量不够高，使得模型在ITPT的过程中，忘记了之前学到的通用语言知识。但是bert-base-chinese较好的表现可以证明这种推测并不可靠。

为了进一步解释该现象，补充一个不使用WMM对chinese-bert-wwm模型进行MLM增量预训练的实验，结果如图 4.24中橙色线所示：

图表, 散点图

描述已自动生成

图 4.24 不使用WMM对chinese-bert-wwm模型进行增量预训练

可以看到，chinese-bert-wwm模型的性能相较于使用WWM时有明显提升，但是依旧达不到bert-base-chinese增量预训练之后的水平。

本人倾向于使用WWM MLM预训练高于原版MLM任务的难度来解释。本次实验中，由于单张3090的性能限制，预训练batch\_size只有32，远远小于模型发布者在Google TPU集群上预训练时采用的384。过小的batch\_size和训练任务的高难度不匹配，给模型的学习和收敛造成了一定困难。这一问题在预训练阶段采用了大batch\_size的roberta上面更加明显。

最终，本实验充分展示了ITPT的有效性，以及WWM MLM预训练在ITPT中的相对有效性，这同样是很有价值的结论。

### 实验八 NSP增量预训练

#### 实验目的：

本实验将尝试使用NSP ITPT提升各模型在本文研究任务上的表现，同时对比测试WMM方法用于增量与训练表现。

#### 实验内容:

本实验使用ltp工具，将原始数据样本的摘要部分拆分为句子，用于构建NSP增量预训练数据集。在构建数据集的过程中，与每一条不是段落中最后一句的句子，有50%的概率选择它和它的下一句组成一条训练样本，也有50%的概率选择它和任意一条不是它的下一句的句子组成训练样本，最终得到的数据集有833133行。

本实验的模型采用transformers库提供了BertForNextSentencePrediction模型，通过对BertModel的进一步封装，可以实现NSP任务。训练时设置batch\_size=32，学习率为2e-5，训练10万步。

#### 实验结果:

在增量预训练的过程中，每隔25000步保存一次检查点，并评测模型此时在下游任务上的性能。通过图就能很明显地看出，NSP增量预训练模型的性能有害。这一结果也和大量相关研究以及经验结论相吻合。

图表, 折线图

描述已自动生成

图 4.25 进行NSP任务增量预训练后各模型的F1分数变化趋势

## 实验结果分析

通过对本文实验进行总结分析，可以得到以下结论：

在微调阶段采用较低学习率、动态学习率调节器以及层次递减学习率对于中文BERT模型而言通常能获得较好的性能提升。

长文本的截断方法与输出层特征融合的方法，整体上偏向于工程技巧（Engineering Trick），是否采纳应当根据具体下游任务具体分析。

生成扰动值更细致更准确的对抗学习方法，对中文BERT模型的性能改善效果越好，但是这样的方法也会消耗更多计算资源。

由于本文任务的特殊性与评价标准的不全面，未能成功利用改进型损失函数解决数据类别分布不均匀的问题。实际生产中，应当考虑采用多模型集成的方法，或者针对交叉学科文献设计专门的评价标准。

对中文BERT模型进行合理步数的MLM任务内增量预训练（ITPT），可以显著提升模型在具体任务上的性能。但是NSP任务的增量预训练会严重损害中文BERT模型的性能，

MLM增量预训练应当采用尽量长的文本。在算力资源不充分时，应当谨慎考虑在增量预训练中使用WWM方法。

## 总结与展望

本文基于BERT模型，以较高的效率和较低成本，实现了对中文期刊论文的分类。同时，本文对多种微调策略、增量预训练任务、对抗训练方法、改进型损失函数进行了有效性验证和对比分析，总结了一些有参考价值的结论。由于不完善的评价标准和算力限制，本文未能较好地解决数据类别不均衡问题，也未能充分发掘WWM方法的潜力，这些问题还需要在未来进一步研究。

中文是世界上使用人数最多的语言之一，具有巨大的应用潜力和市场需求。在未来，中文预训练语言模型将继续是整个自然语言处理领域的研究热点。期待有更多高质量的中文数据集，以及针对中文语言特点进行优化设计的模型能够出现。

参考文献

1. Yoshua Bengio, et al. “A Neural Probabilistic Language Model.” Journal of Machine Learning Research, Mar. 2003.
2. Ronan Collobert, et al. “Natural Language Processing (Almost) from Scratch.” Journal of Machine Learning Research, Feb. 2011.
3. Tomas Mikolov, et al. “Efficient Estimation of Word Representations in Vector Space.” International Conference on Learning Representations, Jan. 2013.
4. Matthew Peters, et al. “Deep Contextualized Word Representations.” Human Language Technologies, Volume 1 (Long Papers), 2018.
5. Ashish Vaswani, et al. “Attention Is All You Need.” Neural Information Processing Systems, June 2017.
6. Alec Radford, et al. “Improving Language Understanding by Generative Pre-Training.” 2018.
7. Jacob Devlin, et al. “BERT: Pre-Training of Deep Bidirectional Transformers for Language Understanding.” Proceedings of the 2019 Conference of the North, 2019.
8. Yinhan Liu, et al. “RoBERTa: A Robustly Optimized BERT Pretraining Approach.” Cornell University - arXiv, July 2019.
9. Zhilin Yang, et al. “XLNet: Generalized Autoregressive Pretraining for Language Understanding.” Neural Information Processing Systems, June 2019.
10. Yiming Cui, et al. “Pre-Training with Whole Word Masking for Chinese BERT.” IEEE/ACM Transactions on Audio, Speech, and Language Processing, Nov. 2021.
11. Jeremy Howard, and Sebastian Ruder. “Universal Language Model Fine-Tuning for Text Classification.” Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics, 2019.
12. Chi Sun, et al. “How to Fine-Tune BERT for Text Classification?” Lecture Notes in Computer Science, Chinese Computational Linguistics, 2019, pp.194–206.
13. Suchin Gururangan, et al. “Don’t Stop Pretraining: Adapt Language Models to Domains and Tasks.” Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, 2020.
14. Yuxian Gu, et al. “Train No Evil: Selective Masking for Task-Guided Pre-Training.” Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP), 2020.
15. Philip Gage. “A New Algorithm for Data Compression.” The C Users Journal Archive, Feb. 1994.
16. Yonghui Wu, et al. “Google’s Neural Machine Translation System: Bridging the Gap between Human and Machine Translation.” Cornell University - arXiv, Sept. 2016.
17. Taku Kudo. “Subword Regularization: Improving Neural Network Translation Models with Multiple Subword Candidates.” Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics, 2019.
18. Taku Kudo, and John Richardson. “SentencePiece: A Simple and Language Independent Subword Tokenizer and Detokenizer for Neural Text Processing.” Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing: System Demonstrations, 2019.
19. Zhongguo Li, and Maosong Sun. “Punctuation as Implicit Annotations for Chinese Word Segmentation.” Computational Linguistics, vol. 35, no. 4, Oct. 2009, pp. 505–12.
20. Ruixuan Luo, et al. “PKUSEG: A Toolkit for Multi-Domain Chinese Word Segmentation.” Cornell University, arXiv, June 2019.
21. Wanxiang Che, et al. “LTP: A Chinese Language Technology Platform.” International Conference on Computational Linguistics, Aug. 2010.
22. Jerome R Bellegarda. “Statistical Language Model Adaptation: Review and Perspectives.” Speech Communication, vol. 42, no. 1, Sept. 2003, pp. 93–108.
23. David E Rumelhart, et al. “Learning Representations by Back-Propagating Errors.” Nature, Nov. 2004, pp. 533–36.
24. F Rosenblatt. “The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain.” Psychological Review, June 2006, pp. 386–408.
25. Yoshua Bengio, et al. “A Neural Probabilistic Language Model.” Journal of Machine Learning Research, Mar. 2003.
26. Cun Le, Y., et al. “Backpropagation Applied to Handwritten Zip Code Recognition.” Neural Computation, Mar. 2008, pp. 541–51.
27. Sepp Hochreiter, and Jürgen Schmidhuber. “Long Short-Term Memory.” Neural Computation, May 2006, pp. 1735–80.
28. Junyoung Chung, et al. “Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling.” arXiv: Neural and Evolutionary Computing, Dec. 2014.
29. Karen Simonyan, and Andrew Zisserman. “Very Deep Convolutional Networks for Large-Scale Image Recognition.” International Conference on Learning Representations, Jan. 2015.
30. Jeffrey Pennington, et al. “Glove: Global Vectors for Word Representation.” Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP), 2015.
31. J Chen, Huang H, Tian S, et al. “Feature selection for text classification with Naïve Bayes.” Expert Systems with Applications,2009,36(3):5432-5435.
32. Ameni Bouaziz, et al. “Short Text Classification Using Semantic Random Forest.” Data Warehousing and Knowledge Discovery, Lecture Notes in Computer Science, 2014, pp. 288–99.
33. Mesleh A Moh'd, Kanaan G. “Support vector machine text classification system: Using Ant Colony Optimization based feature subset selection.” International Conference on Computer Engineering&Systems.IEEE,2008:143-148.
34. S Bahassine, Madani A, Kissi M. “An improved Chi-sqaure feature selection for Arabic text classification using decision tree.” 11th International Conference on Intelligent Systems: Theories and Applications(SITA).IEEE,2016:1-5.
35. Armand Joulin, et al. “Bag of Tricks for Efficient Text Classification.” arXiv: Computation and Language, July 2016.
36. Yoon Kim. “Convolutional Neural Networks for Sentence Classification.” Conference on Empirical Methods in Natural Language Processing (2014).
37. Siwei Lai, et al. “Recurrent Convolutional Neural Networks for Text Classification.” Proceedings of the AAAI Conference on Artificial Intelligence, June 2022.
38. Minh-Thang Luong, et al. “Effective Approaches to Attention-Based Neural Machine Translation.” Cornell University - arXiv, Aug. 2015.
39. Paul Michel, et al. “Are Sixteen Heads Really Better than One.” arXiv: Computation and Language, May 2019.
40. Vinod Nair, and GeoffreyE. Hinton. “Rectified Linear Units Improve Restricted Boltzmann Machines.” International Conference on Machine Learning, June 2010.
41. Yihe Dong, et al. “Attention Is Not All You Need: Pure Attention Loses Rank Doubly Exponentially with Depth.” arXiv: Learning, Mar. 2021.
42. Nitish Srivastava, et al. “Dropout: A Simple Way to Prevent Neural Networks from Overfitting.” Journal of Machine Learning Research, Jan. 2014.
43. Xavier Bouthillier, et al. “Dropout as Data Augmentation.” arXiv: Machine Learning, June 2015.
44. Sergey Ioffe, and Christian Szegedy. “Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift.” arXiv: Learning, Feb. 2015.
45. Shibani Santurkar, et al. How Does Batch Normalization Help Optimization? (No, It Is Not About Internal Covariate Shift). May 2018.
46. Dmitry Ulyanov, et al. “Instance Normalization: The Missing Ingredient for Fast Stylization.” arXiv: Computer Vision and Pattern Recognition, July 2016.
47. Jimmy Ba, et al. “Layer Normalization.” arXiv: Machine Learning, July 2016.
48. Kaiming He, et al. “Deep Residual Learning for Image Recognition.” Cornell University - arXiv, Dec. 2015.
49. Xiaoya Li, et al. “Is Word Segmentation Necessary for Deep Learning of Chinese Representations.” Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics, 2019.
50. Jinhyuk Lee, et al. “BioBERT: A Pre-Trained Biomedical Language Representation Model for Biomedical Text Mining.” Bioinformatics, Sept. 2019, pp. 1234–40.
51. Yi Yang, et al. “FinBERT: A Pretrained Language Model for Financial Communications.” Cornell University - arXiv, June 2020.
52. Kexin Huang, et al. “ClinicalBERT: Modeling Clinical Notes and Predicting Hospital Readmission.” Cornell University - arXiv, Apr. 2019.
53. Iz Beltagy, et al. “SciBERT: A Pretrained Language Model for Scientific Text.” Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP), 2019.
54. Michael McCloskey, and Neal J. Cohen. “Catastrophic Interference in Connectionist Networks: The Sequential Learning Problem.” Psychology of Learning and Motivation, 2008.
55. Ian Goodfellow, et al. “Explaining and Harnessing Adversarial Examples.” Cornell University - arXiv, Dec. 2014.
56. Takeru Miyato, et al. Adversarial Training Methods for Semi-Supervised Text Classification. May 2016.
57. Aleksander Madry, et al. “Towards Deep Learning Models Resistant to Adversarial Attacks.” International Conference on Learning Representations, Feb. 2018.
58. Tsung-Yi Lin, et al. “Focal Loss for Dense Object Detection.” 2017 IEEE International Conference on Computer Vision (ICCV), 2017.
59. Xiaoya Li, et al. “Dice Loss for Data-Imbalanced NLP Tasks.” Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, 2020.
60. Paulius Micikevicius, et al. “Mixed Precision Training.” International Conference on Learning Representations, Oct. 2017.
61. Shibo Wang, et al. “BFloat16: The secret to high performance on Cloud TPUs.” 2021.
62. Yudong Li, et al. “CSL: A Large-Scale Chinese Scientific Literature Dataset.”- arXiv, 2022.