

网络预览版



# 自然语言处理导论

张奇 桂韬 黄萱菁

2023 年 2 月 16 日



数与数组

$\alpha$	标量
$\boldsymbol{\alpha}$	向量
$\boldsymbol{A}$	矩阵
$\mathbf{A}$	张量
$\boldsymbol{I}_n$	$n$ 行 $n$ 列单位矩阵
$\boldsymbol{v}_w$	单词 $w$ 的分布式向量表示
$\boldsymbol{e}_w$	单词 $w$ 的独热向量表示: $[0,0,...,1,0,...0]$ , $w$ 下标处元素为 1

索引

$\alpha_i$	向量 $\boldsymbol{\alpha}$ 中索引 $i$ 处的元素
$\boldsymbol{\alpha}_{-i}$	向量 $\boldsymbol{\alpha}$ 中除索引 $i$ 之外的元素
$w_{i:j}$	序列 $w$ 中从第 $i$ 个元素到第 $j$ 个元素组成的片段或子序列
$A_{ij}$	矩阵 $\boldsymbol{A}$ 中第 $i$ 行、第 $j$ 列处的元素
$\boldsymbol{A}_{i:}$	矩阵 $\boldsymbol{A}$ 中第 $i$ 行
$\boldsymbol{A}_{:,j}$	矩阵 $\boldsymbol{A}$ 中第 $j$ 列
$A_{ijk}$	三维张量 $\mathbf{A}$ 中索引为 $(i, j, k)$ 处元素
$\mathbf{A}::i$	三维张量 $\mathbf{A}$ 中的一个二维切片

集合

$\mathbb{A}$	集合
$\mathbb{R}$	实数集
$\mathbb{C}$	复数集
$\{0, 1, ..., n\}$	含 0 和 $n$ 的正整数的集合
$[a, b]$	$a$ 到 $b$ 的实数闭区间
$(a, b]$	$a$ 到 $b$ 的实数左开右闭区间

## 线性代数

$\mathbf{A}^\top$	矩阵 $\mathbf{A}$ 的转置
$\mathbf{A} \odot \mathbf{B}$	矩阵 $\mathbf{A}$ 与矩阵 $\mathbf{B}$ 的 Hadamard 乘积
$\det \mathbf{A}^\top$	矩阵 $\mathbf{A}$ 的行列式
$[\mathbf{x}; \mathbf{y}]$	向量 $\mathbf{x}$ 与 $\mathbf{y}$ 的拼接
$[\mathbf{U}; \mathbf{V}]$	矩阵 $\mathbf{A}$ 与 $\mathbf{V}$ 沿行向量拼接
$\mathbf{x} \cdot \mathbf{y}$ 或 $\mathbf{x}^\top \mathbf{y}$	向量 $\mathbf{x}$ 与 $\mathbf{y}$ 的点积

## 微积分

$\frac{dy}{dx}$	$y$ 对 $x$ 的导数
$\frac{\partial y}{\partial x}$	$y$ 对 $x$ 的偏导数
$\nabla_{\mathbf{x}} y$	$y$ 对向量 $\mathbf{x}$ 的梯度
$\nabla_{\mathbf{X}} y$	$y$ 对矩阵 $\mathbf{X}$ 的梯度
$\nabla_{\mathbf{X}} y$	$y$ 对张量 $\mathbf{X}$ 的梯度

## 概率与信息论

$a \perp b$	随机变量 $a$ 与 $b$ 独立
$a \perp b \mid c$	随机变量 $a$ 与 $b$ 关于 $c$ 条件独立
$P(a)$	离散变量概率分布
$p(a)$	连续变量概率分布
$a \sim P$	随机变量 $a$ 服从分布 $P$
$\mathbb{E}_{x \sim P}(f(x))$ 或 $\mathbb{E}(f(x))$	$f(x)$ 在分布 $P(x)$ 下的期望
$\text{Var}(f(x))$	$f(x)$ 在分布 $P(x)$ 下的方差
$\text{Cov}(f(x), g(x))$	$f(x)$ 与 $g(x)$ 在分布 $P(x)$ 下的协方差
$H(f(x))$	随机变量 $x$ 的信息熵
$D_{KL}(P \parallel Q)$	概率分布 $P$ 与 $Q$ 的 KL 散度
$\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$	均值为 $\boldsymbol{\mu}$ 、协方差为 $\boldsymbol{\Sigma}$ 的高斯分布

## 数据与概率分布

$\mathbb{X}$ 或 $\mathbb{D}$	数据集
$\mathbf{x}^{(i)}$	数据集中第 $i$ 个样本（输入）
$\mathbf{y}^{(i)}$ 或 $y^{(i)}$	第 $i$ 个样本 $\mathbf{x}^{(i)}$ 的标签（输出）

## 函数

$f: \mathcal{A} \longrightarrow \mathcal{B}$	由定义域 $\mathcal{A}$ 到值域 $\mathcal{B}$ 的函数（映射） $f$
$f \circ g$	$f$ 与 $g$ 的复合函数
$f(\mathbf{x}; \boldsymbol{\theta})$	由参数 $\boldsymbol{\theta}$ 定义的关于 $\mathbf{x}$ 的函数（也可以直接写作 $f(\mathbf{x})$ ，省略 $\boldsymbol{\theta}$ ）
$\log x$	$x$ 的自然对数函数
$\sigma(x)$	Sigmoid 函数 $\frac{1}{1 + \exp(-x)}$
$\ \mathbf{x}\ _p$	$\mathbf{x}$ 的 $L^p$ 范数
$\ \mathbf{x}\ $	$\mathbf{x}$ 的 $L^2$ 范数
$\mathbf{1}^{\text{condition}}$	条件指示函数：如果 condition 为真，则值为 1；否则值为 0

## 本书中常用写法

- 给定词表  $\mathbb{V}$ ，其大小为  $|\mathbb{V}|$
- 序列  $x = x_1, x_2, \dots, x_n$  中第  $i$  个单词  $x_i$  的词向量  $\mathbf{v}_{x_i}$
- 损失函数  $\mathcal{L}$  为负对数似然函数： $\mathcal{L}(\boldsymbol{\theta}) = -\sum_{(x,y)} \log P(y|x_1 \dots x_n)$
- 算法的空间复杂度为  $\mathcal{O}(mn)$

# 目 录

6 语言模型	1
6.1 语言模型概述	1
6.2 n 元语言模型	3
6.2.1 加法平滑	4
6.2.2 古德-图灵估计法	4
6.2.3 Katz 平滑	5
6.2.4 平滑方法总结	7
6.3 神经网络语言模型	8
6.3.1 前馈神经网络语言模型	8
6.3.2 循环神经网络语言模型	9
6.4 预训练语言模型	11
6.4.1 动态词向量算法 ELMo	11
6.4.2 生成式预训练语言模型 GPT	14
6.4.3 掩码预训练语言模型 BERT	16
6.4.4 序列到序列预训练语言模型 BART	19
6.4.5 预训练语言模型的应用	21
6.5 大规模语言模型	23
6.5.1 基础大模型训练	25
6.5.2 指令微调	27
6.5.3 人类反馈	28
6.6 语言模型评价方法	30
6.7 延伸阅读	30
6.8 习题	31

## 6. 语言模型

语言模型目标是建模自然语言的概率分布，在自然语言处理研究中具有重要的作用，是机器翻译、语音识别、输入法、句法分析等任务的支撑。语言模型是自然语言处理基础任务之一，大量的研究从  $n$  元语言模型 ( $n$ -gram Language Models) 和神经语言模型 (Neural Language Models) 等不同角度开展了系列工作。由于语言模型可以为自然语言的表示学习提供天然的自监督训练目标，近年来，预训练语言模型 (Pre-trained Language Models) 做为通用的基于深度神经网络的自然语言处理算法的基础工具，受到越来越多的重视。大规模的预训练语言模型对于提升各类自然语言处理任务的效果起到了重要作用。

本章首先介绍语言模型的基本概念，在此基础上介绍  $n$  元语言模型、神经网络语言模型以及预训练语言模型。

### 6.1 语言模型概述

语言模型 (Language Model, LM) 目标是构建词序列  $w_1w_2...w_m$  的概率分布  $P(w_1w_2...w_m)$ ，即计算给定的词序列  $w_1w_2...w_m$  作为一个句子出现的可能性大小。词汇表  $\mathbb{V}$  上的语言模型由函数  $P(w_1w_2...w_m)$  表示，对于任意词串  $w_1w_2...w_m \in \mathbb{V}^+$ ，则有  $P(w_1w_2...w_m) \geq 0$ ，并且对于所有词串，函数  $P(w_1w_2...w_m)$  满足归一化条件  $\sum_{w_1w_2...w_m \in \mathbb{V}^+} P(w_1w_2...w_m) = 1$ 。  $P(w_1w_2...w_m)$  是定义在  $\mathbb{V}^+$  上的概率分布。

由于联合概率  $P(w_1w_2...w_m)$  的参数数量十分巨大，直接计算  $P(w_1w_2...w_m)$  非常困难。如果把  $w_1w_2...w_m$  看作一个变量，那么它具有  $|\mathbb{V}|^m$  种可能，其中  $m$  代表句子的长度， $|\mathbb{V}|$  表示词表中单词的数量。按照《现代汉语词典（第七版）》包含 7 万词条，句子长度按照 20 个词计算，模型参数量达到  $7.9792 \times 10^{96}$  的天文数字。中文的书面语中超过 100 个单词的句子也并不罕见，如果要将所有可能都纳入考虑，模型的复杂度还会进一步急剧增加，无法进行存储和计算。

为了减少  $P(w_1w_2...w_m)$  模型参数量，可以利用句子序列通常情况下从左至右的生成过程进

行分解，使用链式法则得到：

$$\begin{aligned} P(w_1 w_2 \dots w_m) &= P(w_1) P(w_2 | w_1) P(w_3 | w_1 w_2) \cdots P(w_m | w_1 w_2 \dots w_{m-1}) \\ &= \prod_{i=1}^m P(w_i | w_1 w_2 \cdots w_{i-1}) \end{aligned} \quad (6.1)$$

由此， $w_1 w_2 \dots w_m$  的生成过程可以看作单词逐个生成的过程。首先生成  $w_1$ ，之后根据  $w_1$  生成  $w_2$ ，再根据  $w_1$  和  $w_2$  生成  $w_3$ ，以此类推，根据前  $m-1$  个单词生成最后一个单词  $w_m$ 。通过上述过程将联合概率  $P(w_1 w_2 \dots w_m)$  转换为了多个条件概率的乘积。

例如：对于句子“把努力变成一种习惯”的概率计算，使用公式6.1可以转化为：

$$\begin{aligned} P(\text{把 努力 变成 一种 习惯}) &= P(\text{把}) \times P(\text{努力} | \text{把}) \times P(\text{变成} | \text{把 努力}) \times \\ &\quad P(\text{一种} | \text{把 努力 变成}) \times P(\text{习惯} | \text{把 努力 变成 一种}) \end{aligned} \quad (6.2)$$

但是，仅通过上述过程模型的参数量依然没有下降， $P(w_m | w_1 w_2 \dots w_{m-1})$  的参数量依然是天文数字。然而基于上述转换，我们可以进一步的对模型进行简化， $n$  元语言模型就是其中一种常见的简化方法。本章接下来的章节将对如何估计  $n$  元语言模型参数值以及模型平滑技术进行详细介绍。

由于高阶  $n$  元语言模型还是会面临十分严重的数据稀疏问题，并且单词的离散表示也忽略了单词之间的相似性。因此，基于分布式表示和神经网络的语言模型逐渐成为了研究的热点。Bengio 等人在 2000 年提出了使用前馈神经网络对  $P(w_i | w_{i-n+1} \dots w_{i-1})$  进行估计的语言模型<sup>[1]</sup>。此后，循环神经网络<sup>[2]</sup>、卷积神经网络<sup>[3]</sup>、端到端记忆网络<sup>[4]</sup> 等神经网络方法都成功应用于语言模型建模。相较于  $n$  元语言模型，神经网络方法可以在一定程度上避免数据稀疏问题，有些模型还可以避免对历史长度的限制，从而更好的建模长距离依赖关系。在本章中，我们也将对常见的基于神经网络的语言模型进行介绍。

语言模型的训练过程虽然采用的有监督方法，但是由于训练目标可以通过原始文本直接获得，从而使得模型的训练仅需要大规模无标注文本即可。语言模型也成为了典型的自监督学习（Self-supervised Learning）任务。互联网的快速发展，使得大规模无标注文本非常容易获取，因此训练超大规模的基于神经网络的语言模型成为了可能。2018 年艾伦人工智能研究所（Allen Institute for AI）Peters 等人提出了使用大规模语料，利用语言模型任务获取单词更好的表示的方法 ELMo<sup>[5]</sup>，在多个自然语言处理任务上得到了很好的效果。此后，谷歌公司 Devlin 等人在 2018 年提出了基于 Transformer 模型和掩码语言模型的方法 BERT<sup>[6]</sup>，在包括阅读理解、语义匹配等在内的多个自然语言处理任务中取得了更大幅度的提升，开启了大规模预训练语言模型研究热潮。2021 年谷歌开发的 Switch Transformer 模型参数量首次超过万亿。此后不久，北京智源研究院所就发布参数量超过 1.75 万亿的预训练模型“悟道 2.0”。本章也将介绍采用单向、双向、掩码语言模型的常见预训练方法。



## 6.2 $n$ 元语言模型

语言模型通常用于反映一个句子出现的可能性, 给定由单词序列  $w_1 w_2 \dots w_n$  组成的句子  $S$ , 可以利用语言的特性, 使用链式法分解则得到:

$$P(S) = \prod_{i=1}^n P(w_i | w_1 w_2 \dots w_{i-1}) \quad (6.3)$$

其中, 词  $w_i$  出现的概率受它前面的  $i-1$  个词  $w_1 w_2 \dots w_{i-1}$  影响, 我们将这  $i-1$  个词  $w_1 w_2 \dots w_{i-1}$  称之为词  $w_i$  的历史。如果历史单词有  $i-1$  个, 那么可能的单词组合就有  $|V|^{i-1}$  种, 其中  $V$  表示单词词表,  $|V|$  表示词表的大小。为了简化起见, 使用  $w_1^{i-1}$  表示  $w_1 w_2 \dots w_{i-1}$ 。最简单的根据语料库对  $P(w_i | w_1 w_2 \dots w_{i-1})$  进行估计的方法是基于词序列在语料中出现次数 (也称为频次) 的方法。

$$P(w_i | w_1 w_2 \dots w_{i-1}) = \frac{C(w_1 w_2 \dots w_{i-1} w_i)}{C(w_1 w_2 \dots w_{i-1})} \quad (6.4)$$

其中,  $C(\cdot)$  表示在语料库中词序列在语料库中出现次数。这种方法称为最大似然估计 (Maximum Likelihood Estimation, MLE)。随着历史单词数量的增长, 这种建模方式所需的数据量会指数级增长, 这一现象称为维数灾难 (Curse of Dimensionality)。并且, 随着历史单词数量增多, 绝大多数的历史并不会在训练数据中出现, 这也意味着  $P(w_i | w_1 w_2 \dots w_{i-1})$  就很可能为 0, 使得概率估计失去了意义。

为了解决上述问题, 可以进一步假设任意单词  $w_i$  出现的概率只与过去  $n-1$  个词相关, 即:

$$\begin{aligned} P(w_i | w_1 w_2 \dots w_{i-1}) &= P(w_i | w_{i-(n-1)} w_{i-(n-2)} \dots w_{i-1}) \\ P(w_i | w_1^{i-1}) &= P(w_i | w_{i-n+1}^{i-1}) \end{aligned} \quad (6.5)$$

满足上述条件的模型被称为  $n$  元语法或  $n$  元文法 ( $n$ -gram) 模型。其中  $n$ -gram 表示  $n$  个连续单词构成的单元, 也被称为  $n$  元语法单元。 $n$  的取值越大, 其历史信息越完整, 但参数量也会随之增大。实际应用中,  $n$  的取值通常小于等于 3。当  $n=1$  时, 每个词  $w_i$  的概率独立于历史, 称为一元语法 (Unigram)。当  $n=2$  时, 词  $w_i$  只依赖前一个词  $w_{i-1}$ , 称为二元语法 (Bigram), 又被称作一阶马尔可夫链。当  $n=3$  时, 词  $w_i$  只依赖于前两个历史词  $w_{i-1}$  和  $w_{i-2}$ , 称为三元语法 (Trigram), 又被称作二阶马尔可夫链。

以二元语法为例, 一个词的概率只依赖于前一个词, 则句子  $S$  的出现概率可以表示为:

$$P(S) = \prod_{i=1}^n P(w_i | w_{i-1}) \quad (6.6)$$

为了使  $i < 2$  时上式也成立, 通常在句子开头加上句首标识 <BOS>, 使  $w_0$  为 <BOS>。此外, 句

子结尾也会添加句尾标记 <EOS>。我们还是以计算句子“把努力变成一种习惯”的概率为例，其计算可以转化为：

$$P(<BOS> \text{ 把 努力 变成 一种 习惯 } <EOS>) = P(<BOS>|\text{把}) \times P(\text{努力}|\text{把}) \times P(\text{变成}|\text{努力}) \times P(\text{一种}|\text{变成}) \times P(\text{习惯}|\text{一种}) \quad (6.7)$$

对比公式6.2和公式6.7，可以看到语言模型计算通过  $n$  元语法假设进行了大幅度的简化。

尽管  $n$  元语言模型能缓解句子概率为 0 的问题，但语言是由人和时代创造的，具备无穷的可能性，再庞大的训练语料也无法覆盖所有的  $n$ -gram，而训练语料中的零频率并不代表零概率。因此，需要使用平滑技术（Smoothing）来解决这一问题，对所有可能出现的字符串都分配一个非零的概率值，从而避免零概率问题。平滑是指为了产生更合理的概率，对最大似然估计进行调整的一类方法，也称为数据平滑（Data Smoothing）。平滑处理的基本思想是提高低概率，降低高概率，使整体的概率分布趋于均匀。本节将介绍三种常用的平滑技术。

### 6.2.1 加法平滑

G.J.Lidstone, W.E.Johnson 和 H.Jeffrey 提出的加法平滑（Additive Smoothing）是实际运用中最常用的平滑技术之一。其思想是假设事件出现的次数比实际出现的次数多  $\delta$  次。以二元语法模型为例，其平滑后的条件概率为：

$$P(w_i|w_{i-1}) = \frac{\delta + C(w_{i-1}w_i)}{\sum_{w_i} \delta + C(w_{i-1}w_i)} = \frac{\delta + C(w_{i-1}w_i)}{\delta|V| + C(w_{i-1})} \quad (6.8)$$

其中  $0 \leq \delta \leq 1$ ， $V$  是训练语料中所有单词的集合， $C(w_i)$  表示单词  $w_i$  出现的次数， $C(w_{i-1}w_i)$  代表词  $w_{i-1}$  和词  $w_i$  同时出现次数。可以进一步将公式6.8拓展到  $n$  元语言模型上：

$$P(w_i|w_{i-n+1}^{i-1}) = \frac{\delta + C(w_{i-n+1}^i)}{\delta|V| + \sum_{w_i} C(w_{i-n+1}^i)} \quad (6.9)$$

当  $\delta = 1$  时，该方法又称为加一平滑。此外，针对所有不在词表  $V$  中的单词，可以统一映射为一个特定的词汇，从而保证所有情况下都不存在非零概率。

### 6.2.2 古德-图灵估计法

古德-图灵估计法（Good-Turing Estimate）<sup>[7]</sup> 是 1953 年由 I.J.Good 基于图灵 (Turning) 的方法提出的，是多种平滑技术的核心。该方法基于的核心思想是将一部分已知事件的概率分配给未见的事件。对于  $n$  元语言模型来说，降低出现次数多的  $n$ -gram，同时将剩余概率分配给未出现的  $n$ -gram。

具体来说, 对于任意一个出现了  $r$  次的  $n$  元语法, 按照如下公式修改为出现了  $r^*$  次:

$$r^* = (r + 1) \frac{n_{r+1}}{n_r} \quad (6.10)$$

其中,  $n_r$  代表有  $n_r$  个  $n$ -gram 在训练语料中出现了  $r$  次。对其进行归一化后, 即可得到出现  $r$  次的  $n$  元语法概率:

$$p_r = \frac{r^*}{N} \quad (6.11)$$

其中,  $N = \sum_{r=0}^{\infty} n_r r^*$ , 即  $N$  为分布中最初的计数, 样本中所有事件的概率之和为:

$$\sum_{r>0} n_r p_r = 1 - \frac{n_1}{N} < 1 \quad (6.12)$$

对于  $r = 0$  的未见事件, 有  $\frac{n_1}{N}$  的概率余量可以用于分配。

表6.1给出了一个使用 Good-Turing 方法进行估计的样例。通过语料库统计 Bigram 的出现次数, 并通过公式6.10修正后得到的  $r^*$  以及通过公式6.11得到修正后的概率  $p_r$ 。对于没有出现过的 Bigram 的概率总和为:  $p_0 = \frac{n_1}{N}$ 。可以根据词表  $\mathbb{V}$ , 估计未出现的 Bigram 的总数  $n_0 = |\mathbb{V}|^2 - \sum_{r>0} n_r$ 。将  $p_0$  根据  $n_0$  均分, 可以得到未出现过得 Bigram 的概率值。

表 6.1 Good-Turing 估计方法样例

$r$	$n_r$	$r^*$	$p_r$
1	3286	0.448	$5.220 \times 10^{-5}$
2	736	1.25	$1.454 \times 10^{-4}$
3	306	2.25	$2.620 \times 10^{-4}$
4	172	3.17	$3.693 \times 10^{-4}$
5	109	4.18	$4.875 \times 10^{-4}$
6	76	5.53	$6.440 \times 10^{-4}$
7	60	5.73	$6.681 \times 10^{-4}$
8	43	5.86	$6.830 \times 10^{-4}$
9	28	7.14	$8.324 \times 10^{-4}$
10	20	—	—

古德-图灵估计法的缺点是其无法用于估计  $n_r = 0$  的  $n$  元语法概率, 并且其不能用于高阶语言模型和低阶语言模型的结合, 而高阶与低阶模型的结合通常能带来更好的平滑效果。但古德-图灵方法的思想简单普适, 因此其往往是作为一种基本方法与其他的平滑方法结合。

### 6.2.3 Katz 平滑

Katz 平滑是 1987 年由 S. M. Katz 所提出的后备 (back-off) 平滑方法<sup>[8]</sup>, 其在古德-图灵估计法

的基础上引入了高阶模型与低阶模型的结合。Katz 平滑法的基本思想是将因减值获得的概率余量根据低阶模型的分布分配给未见事件，而不是进行平均分配，从而令低概率事件有更合理的概率分布。Katz 平滑法的做法是，当事件在样本中出现的频次大于某一数值  $k$  时，运用最大似然估计法，通过减值来估计其概率值；而当事件的频次小于  $k$  值时，使用低阶的语法模型作为代替高阶语法模型的后备。

下面以二元语法模型为例说明 Katz 平滑方法的实现方法。对于一个出现次数为  $r$  的二元语法  $w_{i-1}^i$ ，用下列公式对其次数进行修正：

$$C_{katz}(w_{i-1}^i) = \begin{cases} d_r \frac{C(w_{i-1}w_i)}{C(w_{i-1})}, & r > 0 \\ a(w_{i-1})P_{ML}(w_i), & r = 0 \end{cases} \quad (6.13)$$

其中  $d_r \approx \frac{r^*}{r}$  是由古德-图灵估计法预测的折扣率。可以看出，所有具有非零计数  $r$  的二元语法都根据折扣率  $d_r$  被减值了。从非零计数中减去的计数量，根据低一阶的分布，即一元语法模型，被分配给了计数为零的二元语法。式中  $P_{ML}(w_i)$  为  $w_i$  的最大似然估计概率， $a(w_{i-1})$  使分布中总计数保持不变，即  $\sum_{w_i} C_{katz}(w_{i-1}^i) = \sum_{w_i} C(w_{i-1}w_i)$ 。  $a(w_{i-1})$  的值通常按照如下公式估计：

$$a(w_{i-1}) = \frac{1 - \sum_{w_i: C(w_{i-1}w_i) > 0} P_{katz}(w_i|w_{i-1})}{\sum_{w_i: C(w_{i-1}^i) = 0} P_{ML}(w_i)} = \frac{1 - \sum_{w_i: C(w_{i-1}^i) > 0} P_{katz}(w_i|w_{i-1})}{1 - \sum_{w_i: C(w_{i-1}^i) > 0} P_{ML}(w_i)} \quad (6.14)$$

根据修正的计数计算概率  $P_{katz}(w_i|w_{i-1})$ ，需要按照如下公式进行归一化：

$$P_{katz}(w_i|w_{i-1}) = \frac{C_{katz}(w_{i-1}^i)}{\sum_{w_i} C_{katz}(w_{i-1}^i)} \quad (6.15)$$

折扣率  $d_r$  需要满足两个约束条件：(1) 保证总折扣量和古德-图灵估计得到的减值量成比例，即保证对于常数  $\mu, r \in \{1, 2, \dots, k\}$ ，如以下公式所示：

$$1 - d_r = \mu(1 - \frac{r^*}{r}) \quad (6.16)$$

(2) 保证二元语法分布中被折扣的计数总量等于古德-图灵估计得到的次数为零的 Bi-gram 总数  $n_0 0^* = n_0 \frac{n_1}{n_0} = n_1$ ，相当于：

$$\sum_{r=1}^k n_r(1 - d_r)r = n_1 \quad (6.17)$$

上述公式6.16和公式6.17的唯一解为：

$$d_r = \frac{\frac{r^*}{r} - \frac{(k+1)n_{k+1}}{n_1}}{1 - \frac{(k+1)n_{k+1}}{n_1}} \quad (6.18)$$

在二元语法的基础上可以将 Katz 平滑算法拓展到高阶  $n$  元语法模型。类似于公式6.13，可以根据一元语法模型定义二元语法模型，Katz 的  $n$  元语法模型可以根据  $n-1$  元语法模型定义：

$$P(w_i|w_{i-n+1}^{i-1}) = \begin{cases} P_{GT}(w_i|w_{i-n+1}^{i-1}), & C(w_{i-n+1}^i) > 0 \\ a(w_{i-n+1}^{i-1})P_{GT}(w_i|w_{i-n+2}^{i-1}), & C(w_{i-n+1}^i) = 0 \text{ 并且 } C(w_{i-n+1}^{i-1}) > 0 \\ P_{BF}(w_i|w_{i-n+2}^{i-1}), & C(w_{i-n+1}^i) = 0 \end{cases} \quad (6.19)$$

其中， $P_{BF}$  和  $P_{GT}$  分别代表后备法和古德-图灵估计法计算得到的概率值。 $a(w_{i-n+1}^{i-1})$  定义为：

$$\begin{aligned} a(w_{i-n+1}^{i-1}) &= \frac{1 - \sum_{w_i: C(w_{i-n+1}^i) > 0} P_{GT}(w_i|w_{i-n+1}^{i-1})}{\sum_{w_i: \{C(w_{i-n+1}^i)=0 \& C(w_{i-n+1}^{i-1}) > 0\}} P_{GT}(w_i|w_{i-n+2}^{i-1})} \\ &= \frac{1 - \sum_{w_i: C(w_{i-n+1}^i) > 0} P_{GT}(w_i|w_{i-n+1}^{i-1})}{1 - \sum_{w_i: C(w_{i-n+1}^i) > 0} P_{GT}(w_i|w_{i-n+2}^{i-1})} \end{aligned} \quad (6.20)$$

满足以下约束：

$$\sum_{w_i: \{C(w_{i-n+1}^i)=0 \& C(w_{i-n+1}^{i-1}) > 0\}} P_{BF}(w_i|w_{i-n+1}^{i-1}) + \sum_{w_i: C(w_{i-n+1}^i) > 0} P_{BF}(w_i|w_{i-n+1}^{i-1}) = 1 \quad (6.21)$$

## 6.2.4 平滑方法总结

除了我们在上述章节介绍的平滑算法之外，研究人员们提出了有很多针对语言模型的平滑算法，包括 Jelinek Mercer 平滑算法<sup>[9]</sup>、Witten-Bell 平滑算法<sup>[10]</sup>、Kneser-Ney 平滑算法<sup>[11]</sup> 等。这些方法的核心思想大都可以归纳为：如果  $n$ -gram 存在则使用其本身计数，如果不存在再退后到低阶分布。可以用如下公式表示：

$$P_{\text{smooth}}(w_i|w_{i-n+1:i-1}) = \begin{cases} \alpha(w_i|w_{i-n+1}^{i-1}), & C(w_{i-n+1}^i) > 0 \\ \gamma(w_{i-n+1}^{i-1})P_{\text{smooth}}(w_i|w_{i-n+2}^{i-1}), & C(w_{i-n+1}^i) = 0 \end{cases} \quad (6.22)$$

在此基础上，一些平滑算法采用高阶和低阶  $n$  元语法模型的线性插值的方法，融合高阶和低

阶语法的估计，如以下公式所示：

$$P_{\text{smooth}}(w_i|w_{i-n+1}^{i-1}) = \lambda P_{\text{ML}}(w_i|w_{i-n+1}^{i-1}) + (1 - \lambda)P_{\text{smooth}}(w_i|w_{i-n+2}^{i-1}) \quad (6.23)$$

$n$  语法模型整体上来看与训练语料规模和模型的阶数有较大的关系，不同的平滑算法在不同情况下的表现有较大的差距。平滑算法虽然较好的解决了零概率问题，但是基于稀疏表示的  $n$  元语言模型仍然有三个较为明显的缺点：（1）无法建模长度超过  $n$  的上下文；（2）依赖人工设计规则的平滑技术；（3）当  $n$  增大时，数据的稀疏性随之增大，模型的参数量更是指数级增加，并且模型受到数据稀疏问题的影响，其参数难以被准确的学习。

## 6.3 神经网络语言模型

随着神经网络的发展，利用神经网络的语言模型展现出了比  $n$  元语言模型更强学习能力。神经网络先进的结构使其能有效的建模长距离上下文依赖，以词向量（Word Embedding）为代表的分布式表示的语言模型深刻地影响了自然语言处理领域的其他模型与应用的变革<sup>①</sup>。因此， $n$  元语言模型几乎已经被神经网络的语言模型所替代。本节将介绍如何使用经典的前馈神经网络和循环神经网络来建模语言模型。

### 6.3.1 前馈神经网络语言模型

给定历史单词序列  $w_1 w_2 \dots w_{i-1}$ ，神经网络语言模型的目标是根据历史单词对下一时刻词进行预测。与传统  $n$  元语言模型类似，前馈神经网络语言模型<sup>[1]</sup>沿用了马尔可夫假设，即下一时刻的词只与过去  $n-1$  个词相关，其目标可以表示为输入历史单词  $w_{(i-n+1)} \dots w_{i-1}$ ，输出词  $w_i$  在词表  $\mathbb{V}$  上的概率分布，即估计条件概率  $P(w_i|w_{(i-n+1)}^{i-1})$ 。

前馈神经网络由三部分组成，如图6.1所示，分别为输入层、隐藏层和输出层。历史词序列首先经过输入层被转换为离散的独热编码，随后每个词的独热编码被映射为一个低维稠密的实数向量；隐藏层对词向量层的输出进行编码，进行多次线性变换与非线性映射；最后，隐藏层向量经过输出层被映射到词表空间，再利用 Softmax 函数得到其词表上的概率分布。

**输入层**的目标是将由文本组成的词序列转化为模型可接受的低维稠密向量。在具体实现中，模型首先根据每个词在词表  $\mathbb{V}$  中的位置，将历史词序列  $w_{(i-n+1)}, \dots, w_{i-1}$  转化为对应的独热编码 (One-Hot Encoding)，再将每个词的独热编码映射到一个低维稠密的实数向量。该映射可以视作是根据一个查找表，获取每个词特有的词向量的过程：

$$\mathbf{v} = [\mathbf{v}_{(i-n+1)}, \dots, \mathbf{v}_{i-1}] \quad (6.24)$$

其中  $\mathbf{v}_{i-1} \in \mathbb{R}^d$  代表词  $w_{i-1}$  所对应的词向量， $d$  代表词向量的维度， $\mathbf{v} \in \mathbb{R}^{(n-1) \times d}$  代表将所有历

<sup>①</sup> 详见本书第4章分布式表示

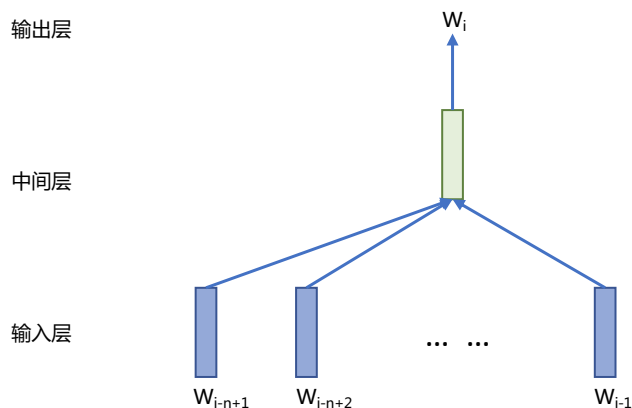


图 6.1 基于前馈神经网络的语言模型结构图

史词向量拼接后的结果。

**隐藏层**的目标是对词向量  $v$  进行线性变换与非线性映射。具体计算过程可以使用如下公式表示：

$$\mathbf{h} = f(\mathbf{W}^{hid}\mathbf{v} + \mathbf{b}^{hid}) \quad (6.25)$$

其中，隐藏层由线性变换矩阵  $\mathbf{W}^{hid} \in \mathbb{R}^{m \times (n-1)d}$ 、偏置项  $\mathbf{b}^{hid} \in \mathbb{R}^m$  组成， $m$  为隐藏层维度， $f$  为非线性激活函数，常见激活函数的有 Sigmoid、tanh 和 ReLU 等。

**输出层**的目标是基于隐藏层向量  $\mathbf{h}$  得到词表空间  $\mathbb{V}$  上的概率分布。输出层的计算可以用如下公式表示：

$$\mathbf{y} = \text{Softmax}(\mathbf{W}^{out}\mathbf{h} + \mathbf{b}^{out}) \quad (6.26)$$

其中， $\mathbf{W}^{out} \in \mathbb{R}^{|\mathbb{V}| \times m}$  是输出层的线性变换矩阵， $\mathbf{b}^{out}$  为偏置项， $|\mathbb{V}|$  为词表大小。

上述前馈神经网络语言模型的总参数量为  $|\mathbb{V}| \times d + m \times (n-1)d + m + |\mathbb{V}| \times m + |\mathbb{V}|$ ，即  $|\mathbb{V}|(d + m + 1) + m((n-1)d + 1)$ 。词向量维度  $d$ ，隐藏层维度  $m$  和历史词长度  $n-1$  可以根据实际需求进行调整。可以看出，词表大小  $|\mathbb{V}|$  和历史词长度  $n-1$  的增大并不会显著增加前馈神经网络语言模型的总参数量，而是维持着线性增长的关系，这也是神经网络模型优于  $n$  元语言模型重要方面。

### 6.3.2 循环神经网络语言模型

在实际场景下，固定长度的历史词并不是总能提供充分的信息，对于信息较为复杂的长文本，模型需要依赖较长的历史才能做出准确预测。

例如：与小明一起旅行游玩总是充满了惊喜，你永远不会知道他将要带你到哪里去。  
模型需要获取“小明”这一信息才能对“他”进行准确的预测。而这两个词语之间的距离接近 15 个单

词，如果采用前馈神经网络，需要的历史词长度过长。

循环神经网络（Recurrent Neural Network, RNN）<sup>[2]</sup> 常用于处理序列结构的数据，其特点是上一时刻的模型隐藏层状态会作为当前时刻模型的输入，每一时刻的隐藏层状态都会维护所有过去词的信息。循环神经网络语言模型不再基于马尔可夫假设，每个时刻的单词都会考虑到过去所有时刻的单词，词之间的依赖通过隐藏层状态来获取，这刚好解决了语言模型需要动态依赖的问题。与前馈神经网络语言模型类似，循环神经网络语言模型由三部分组成：输入层、隐藏层和输出层，其结构如图6.2所示。

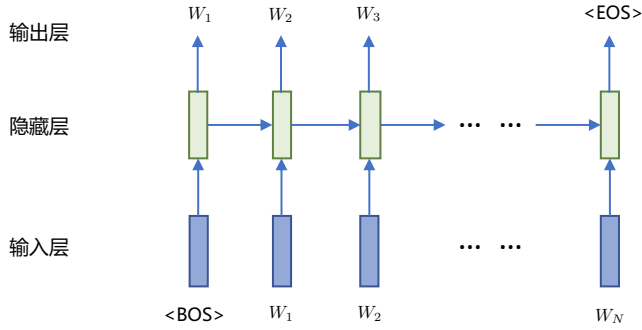


图 6.2 循环神经网络的结构

循环神经网络语言模型不限制历史词长度,而是使用整个历史序列。给定历史序列  $w_1, \dots, w_{i-1}$ , 第  $i$  时刻语言模型的目标是预测第  $i$  个词  $w_i$ , 此时循环神经网络语言模型的输入由两部分组成, 前一个词  $w_{i-1}$  的词向量以及包含所有历史词信息的  $i-1$  时刻隐藏层输出  $h_{i-1}$ :

$$\mathbf{x}_i = [\mathbf{v}_{i-1}; \mathbf{h}_{i-1}] \quad (6.27)$$

其中  $\mathbf{x}_i \in \mathbb{R}^{d+m}$ ,  $\mathbf{v}_{i-1} \in \mathbb{R}^d$  代表词  $w_{i-1}$  所对应的词向量,  $d$  为词向量维度  $\mathbf{h}_{i-1} \in \mathbb{R}^m$  代表前一时刻模型的隐藏层输出,  $m$  为隐藏层维度。特别的, 对于第 1 个词  $w_1$ , 由于其没有历史时刻信息, 通常使用一个随机初始化向量或  $\mathbf{0}$  向量  $h_0$  作为初始隐藏层向量。

循环神经网络语言模型隐藏层的目标是进行线性变化与非线性激活, 隐藏层的计算可以用如下公式表示:

$$\mathbf{h}_i = f(\mathbf{W}^{hid} \mathbf{x}_i + \mathbf{b}^{hid}) \quad (6.28)$$

其中,  $\mathbf{W}^{hid} \in \mathbb{R}^{m \times (d+m)}$ ,  $\mathbf{b}^{hid} \in \mathbb{R}^m$ 。其中, 因为  $\mathbf{x}_i$  可以被分解为两部分,  $\mathbf{W}^{hid}$  也可以分解为  $\mathbf{W}^{hid} = [\mathbf{U}; \mathbf{V}]$ ,  $\mathbf{U} \in \mathbb{R}^{m \times d}$  是词向量  $w_{i-1}$  的权重,  $\mathbf{V} \in \mathbb{R}^{m \times m}$  是隐藏层输出  $h_{i-1}$  的权重。将



其拆分开更能体现循环神经网络递归的特点：

$$\mathbf{h}_i = f(\mathbf{U}\mathbf{v}_{i-1} + \mathbf{V}\mathbf{h}_{i-1} + \mathbf{b}^{hid}) \quad (6.29)$$

循环神经网络语言模型输出层的目标是基于隐藏层状态  $\mathbf{h}_i$  预测词表  $\mathbb{V}$  上的概率分布，输出层的计算可以用如下公式表示：

$$\mathbf{y}_i = \text{Softmax}(\mathbf{W}^{out}\mathbf{h}_i + \mathbf{b}^{out}) \quad (6.30)$$

其中， $\mathbf{W}^{out} \in \mathbb{R}^{|\mathbb{V}| \times m}$ 。

本节只介绍了最基本的循环神经网络，隐藏层的结构较为简单。在处理长序列时，训练这样的循环神经网络可能会遇到梯度消失或梯度爆炸问题，导致无法进行有效的训练。一种解决方案是在反向传播的过程中按长度对梯度进行截断，但这一做法会损害模型建模长距离依赖的能力。另一种做法是使用如 LSTM<sup>[12]</sup> 等具备门控机制的循环神经网络，这类循环神经网络语言模型往往能实现更稳定的训练和更好的性能。

## 6.4 预训练语言模型

受到计算机视觉领域采用 ImageNet<sup>[13]</sup> 对模型进行一次预选训练，使得模型可以通过海量图像充分学习如何提取特征，然后再根据任务目标进行模型精调的范式影响，自然语言处理领域基于预训练语言模型的方法也逐渐成为主流。以 ELMo<sup>[5]</sup> 为代表的动态词向量模型开启了语言模型预训练的大门，此后以 GPT<sup>[14]</sup> 和 BERT<sup>[6]</sup> 为代表的基于 Transformer 的大规模预训练语言模型的出现，使得自然语言处理全面进入了预训练微调范式新时代。利用丰富的训练语料、自监督的预训练任务以及 Transformer 等深度神经网络结构，使预训练语言模型具备了通用且强大的自然语言表示能力，能够有效地学习到词汇、语法和语义信息。将预训练模型应用于下游任务时，不需要了解太多的任务细节，不需要设计特定的神经网络结构，只需要“微调”预训练模型，即使用具体任务的标注数据在预训练语言模型上进行监督训练，就可以取得显著的性能提升。

本节中，我们将首先介绍以 ELMo 为代表的动态词向量方法，在此基础上介绍基于 Transformer 结构的 BERT 预训练语言模型和以 GPT 和 BART 为代表的生成式预训练模型。

### 6.4.1 动态词向量算法 ELMo

如第 4 章所介绍的单词分布式表示所述，词向量主要利用语料库中词之间的共现信息，学习词语的向量表示。因此，根据给定的语料库所学习的到的词向量是恒定不变的，可以认为是“静态”的，不跟随上下文发生变化。然而，自然语言中词语往往具有多种语义，在不同的上下文或语境下会具有不同的语义。针对该问题，研究人员们提出了动态词向量（Dynamic Word Embedding），也称为上下文相关的词向量（Contextualized Word Embedding）方法，一个词语的向量通过其所在的上下

文计算获得，跟随上下文动态变化。

文献[5]提出了深度上下文相关词向量并介绍了双向预训练语言模型 ELMo (Embeddings from Language Models)。双向语言模型是从两个方向进行语言模型建模：从左到右前向建模和从右到左后向建模。双向建模带来了更好的上下文表示，文本中的每个词能同时利用其左右两侧文本的信息。ELMo 的神经网络结构如图 6.3 所示，主要包含输入层，编码层和输出层三个部分。

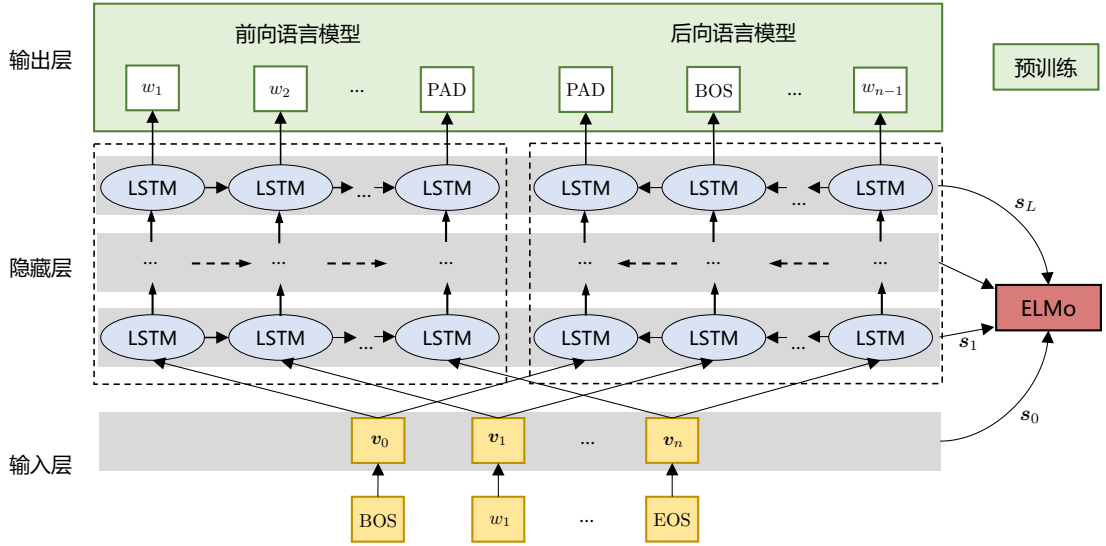


图 6.3 双向预训练语言模型 ELMo 神经网络结构<sup>[5]</sup>

输入层为了减少词不在词表中 (Out-of-Vocabulary) 的情况，对输入文本进行字符级别的编码。具体来说，输入文本中的每个词  $w_i$  视作由字符序列  $c_{i_1}c_{i_2}\dots c_{i_m}$  组成，每个字符  $c_{i_j}$  通过字符嵌入层转化为向量  $v_{c_{i_j}}$ ：

$$v_{c_{i_j}} = \mathbf{W}^{\text{char}} e_{c_{i_j}} \quad (6.31)$$

其中， $\mathbf{W}^{\text{char}} \in \mathbb{R}^{d^{\text{char}} \times |\mathbb{V}^{\text{char}}|}$  为字符嵌入矩阵、 $\mathbb{V}^{\text{char}}$  表示字符库、 $d^{\text{char}}$  表示字符向量维度、 $e_{c_{i_j}}$  表示字符  $c_{i_j}$  的独热向量。得到词  $w_i$  的字符向量表示  $v_{c_{i_1}}, v_{c_{i_2}}, \dots, v_{c_{i_m}}$  后，ELMo 模型使用卷积神经网络对字符级的表示进行语义组合，通过调整卷积神经网络的卷积核与通道数，可以得到不同粒度的字符级上下文信息。随后，在每个位置的卷积输出上使用池化层，得到词  $w_i$  的词级别表示  $\hat{v}_i$ 。在得到卷积神经网络的输出  $\hat{v}_i$  后，为了避免梯度消失或爆炸，模型使用 Highway 网络对  $\hat{v}_i$  进一步转换：

$$v_i = g \cdot \hat{v}_i + (1 - g) \cdot \text{ReLU}(\mathbf{W}\hat{v}_i + b) \quad (6.32)$$

其中,  $g$  为门控向量, 以卷积神经网络输出  $\hat{v}_i$  为输入:

$$g = \sigma(\mathbf{W}^g \hat{v}_i + \mathbf{b}^g) \quad (6.33)$$

其中,  $\mathbf{W}^g$  为线性转换矩阵、 $\mathbf{b}^g$  为偏置。得到了每个词上下文无关的词向量后, 接下来 ELMo 的编码层将从两个方向对词向量进一步编码。

ELMo 使用了两个独立的编码器分别对前向和后向进行语言模型建模, 在进行预训练时, 分别取最高层的正向和反向 LSTM 输出  $\vec{h}_{i,L}$  和  $\overleftarrow{h}_{i,L}$  预测下一时刻的词。对于给定的一段文本  $w_1 w_2 \dots w_n$  而言, 前向语言模型在  $t$  时刻的目标词为  $w_{t+1}$ , 而后向语言模型的目标词则为  $w_{t-1}$ 。采用前向和后向语言模型的建模过程可以表示为:

$$\begin{aligned} P_{forward}(w_1 w_2 \dots w_n) &= \prod_{i=1}^n P(w_i | w_{1:i-1}; \theta_f) \\ P_{backward}(w_1 w_2 \dots w_n) &= \prod_{i=1}^n P(w_i | w_{i+1:n}; \theta_b) \end{aligned} \quad (6.34)$$

其中,  $\theta_f$  和  $\theta_b$  分别代表了代表前向和后向 LSTM 模型的参数。特别需要注意的是双向模型共享输出层的参数。

ELMo 算法的编码层采用了多层双向 LSTM 结构, 通常认为, 模型低层能捕捉语法等基础特征, 高层能捕捉语义语境等更深层次的语言特征, 双向的 LSTM 能保证在编码过程中每个位置都能获得该位置过去和未来位置的词信息。对于词  $w_i$  来说, 一个  $L$  层的 ELMo 模型会产生  $2L + 1$  向量表示:

$$R_i = \{v_i, \vec{h}_{i,j}, \overleftarrow{h}_{i,j} | j = 1, \dots, L\} \quad (6.35)$$

其中,  $v_i$  代表输入层得到的上下文无关词向量,  $\vec{h}_{i,j}$  代表第  $j$  层前向 LSTM 编码得到的特征,  $\overleftarrow{h}_{i,j}$  代表第  $j$  层后向 LSTM 编码得到的特征。对于每层得到的两个方向的特征, ELMo 将其拼接起来得到  $\mathbf{h}_{i,j}^{LM} = [\vec{h}_{i,j}; \overleftarrow{h}_{i,j}]$ 。在进行下游任务时, ELMo 将  $R_i$  中的所有向量整合成一个向量, 整合的方式由任务而定, 最简单的情况是直接使用最后一层的表示  $\mathbf{h}_{i,j}^{LM}$ 。因为每层 LSTM 学习到的信息不相同, 对于不同任务来说, 每层特征的重要性也不尽相同, 因此更普遍的做法是根据任务所需信息, 对每层的特征进行加权得到词  $w_i$  的对应的 ELMo 向量, 其计算过程可以表示为:

$$\text{ELMo}_i^{\text{task}} = \gamma^{\text{task}} \sum_{j=0}^L s_j^{\text{task}} \mathbf{h}_{i,j}^{LM} \quad (6.36)$$

其中  $\gamma^{\text{task}}$  是整体的缩放系数,  $s^{\text{task}}$  是每层的权重系数, 反映每一层向量对于目标任务的重要性。在执行下游任务时, 一般将  $v_i$  和  $\text{ELMo}_i^{\text{task}}$  拼接起来作为词  $w_i$  的最终表示向量进行分类, 使用  $\gamma^{\text{task}}$  对 ELMo 向量进行适当缩放。  $s^{\text{task}}$  则通常在下流任务的训练过程中学习得到。而 ELMo 模型的中

编码器参数在下游任务训练时则被“冻结”，不参与更新。

### 6.4.2 生成式预训练语言模型 GPT

OpenAI 公司在 2018 年提出的 GPT (Generative Pre-Training) [14] 模型是典型的生成式预训练语言模型之一。GPT-2 模型结构如图 6.4 所示，由多层 Transformer 组成的单向语言模型，主要可以分为输入层，编码层和输出层三部分。本节将介绍 GPT-2 模型结构以及单向语言模型的预训练过程和判别式任务精调。

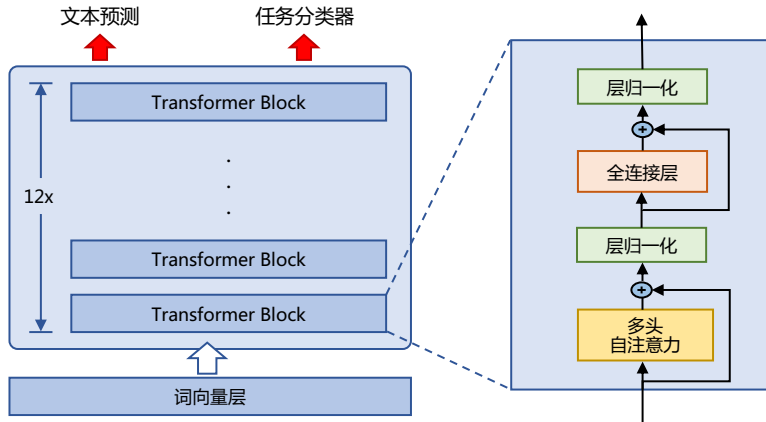


图 6.4 GPT-2 预训练语言模型结构

#### 1. 无监督预训练

GPT 采用生成式预训练方法，单向意味着模型只能从左到右或从右到左对文本序列建模，所采用的 Transformer 结构<sup>①</sup>保证了输入文本每个位置只能依赖过去时刻的信息。

给定文本序列  $w = w_1 w_2 \dots w_n$ ，GPT-2 首先在输入层中将其映射为稠密的向量：

$$\mathbf{v}_i = \mathbf{v}_i^t + \mathbf{v}_i^p \quad (6.37)$$

其中， $\mathbf{v}_i^t$  是词  $w_i$  的词向量， $\mathbf{v}_i^p$  是词  $w_i$  的位置向量， $\mathbf{v}_i$  为第  $i$  个位置的单词经过模型输入层（第 0 层）后的输出。GPT-2 模型的输入层与前文中介绍的神经网络语言模型的不同之处在于其需要添加位置向量，这是 Transformer 结构自身无法感知位置导致的，因此需要来自输入层的额外位置信息。

经过输入层编码，模型得到表示向量序列  $\mathbf{v} = \mathbf{v}_1 \dots \mathbf{v}_n$ ，随后将  $\mathbf{v}$  送入模型编码层。编码层由

<sup>①</sup> Transformer 解码器的具体结构请参考第 8 章??节。

$L$  个 Transformer 模块组成，在自注意力机制的作用下，每一层的每个表示向量都会包含之前位置表示向量的信息，使每个表示向量都具备丰富的上下文信息，并且经过多层解码后，GPT-2 能得到每个单词层次化的组合式表示，其计算过程表示如下：

$$\mathbf{h}^{(L)} = \text{Transformer-Block}^{(L)}(\mathbf{h}_i^{(0)}) \quad (6.38)$$

其中  $\mathbf{h}^{(L)} \in \mathbb{R}^{d \times n}$  表示第  $L$  层的表示向量序列， $n$  为序列长度， $d$  为模型隐藏层维度， $L$  为模型总层数。

GPT-2 模型的输出层基于最后一层的表示  $\mathbf{h}^{(L)}$ ，预测每个位置上的条件概率，其计算过程可以表示为：

$$P(w_i | w_1, \dots, w_{i-1}) = \text{Softmax}(\mathbf{W}^e \mathbf{h}_i^{(L)} + \mathbf{b}^{out}) \quad (6.39)$$

其中， $\mathbf{W}^e \in \mathbb{R}^{|\mathbb{V}| \times d}$  为词向量矩阵， $|\mathbb{V}|$  为词表大小。

单向语言模型是按照阅读顺序输入文本序列  $w$ ，用常规语言模型目标优化  $w$  的最大似然估计，使之能根据输入历史序列对当前词能做出准确的预测：

$$\mathcal{L}^{\text{PT}}(w) = - \sum_{i=1}^n \log P(w_i | w_0 \dots w_{i-1}; \theta) \quad (6.40)$$

其中  $\theta$  代表模型参数。也可以基于马尔可夫假设，只使用部分过去词进行训练。预训练时通常使用随机梯度下降法进行反向传播优化该似然函数。

## 2. 有监督下游任务精调

通过无监督语言模型预训练，使得 GPT 模型具备了一定的通用语义表示能力。根据下游任务精调（Fine-tuning）的目的是在通用语义表示基础上，根据下游任务的特性进行适配。下游任务通常需要利用有标注数据集进行训练，数据集使用  $\mathbb{D}$  进行表示，每个样例输入长度为  $n$  的文本序列  $x = x_1 x_2 \dots x_n$  和对应的标签  $y$  构成。

首先将文本序列  $x$  输入 GPT 模型，获得最后一层的最后一个词所对应的隐藏层输出  $\mathbf{h}_n^{(L)}$ ，在此基础上通过全连接层变换结合 Softmax 函数，得到标签预测结果。

$$P(y | x_1 \dots x_n) = \text{Softmax}(\mathbf{h}_n^{(L)} \mathbf{W}^y) \quad (6.41)$$

其中  $\mathbf{W}^y \in \mathbb{R}^{d \times k}$  为全连接层参数， $k$  为标签个数。通过对整个标注数据集  $\mathbb{D}$  优化如下损失函数精调下游任务：

$$\mathcal{L}^{\text{FT}}(\mathbb{D}) = \sum_{(x,y)} \log P(y | x_1 \dots x_n) \quad (6.42)$$

下游任务在精调过程中，针对任务目标进行优化，很容易使得模型对预训练阶段所学习到的通

用语义知识表示遗忘,从而损失模型的通用性和泛化能力,造成灾难性遗忘(Catastrophic Forgetting)问题。因此,通常会采用混合预训练任务损失和下游精调损失的方法来缓解上述问题。在实际应用中,通常采用如下公式进行下游任务精调:

$$\mathcal{L} = \mathcal{L}^{\text{FT}}(\mathbb{D}) + \lambda \mathcal{L}^{\text{PT}}(\mathbb{D}) \quad (6.43)$$

其中  $\lambda$  取值为  $[0,1]$ , 用于调节预训练任务损失占比。

### 6.4.3 掩码预训练语言模型 BERT

2018 年, Devlin 等人提出了掩码预训练语言模型 BERT<sup>[6]</sup> (Bidirectional Encoder Representation from Transformers)。BERT 利用掩码机制构造了基于上下文预测中间词的预训练任务, 相较于传统的语言模型建模方法, BERT 能进一步挖掘上下文所带来的丰富语义。BERT 所采用的神经结构如图6.5所示, 其由多层 Transformer 编码器组成, 这意味着在编码过程中, 每个位置都能获得所有位置的信息, 而不仅仅是历史位置的信息。BERT 同样由输入层, 编码层和输出层三部分组成。编码层由多层 Transformer 编码器组成。在预训练时, 模型的最后有两个输出层 MLM 和 NSP, 分别对应了两个不同的预训练任务: 掩码语言模型 (Masked Language Modeling, MLM) 和下一句预测 (Next Sentence Prediction, NSP)。

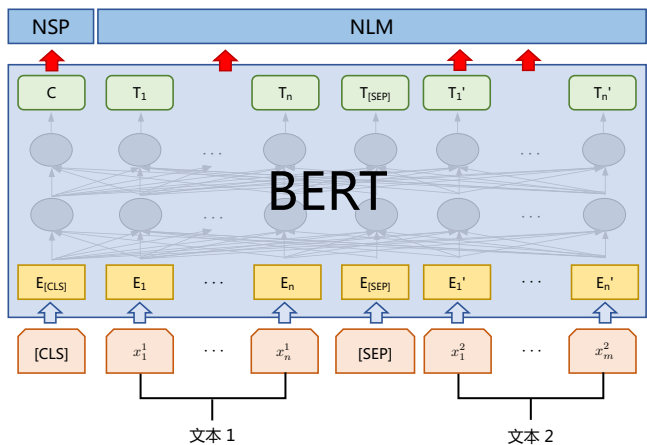


图 6.5 掩码预训练语言模型 BERT 神经网络结构<sup>[6]</sup>

需要注意的是, 掩码语言模型的训练对于输入形式没有要求, 可以是一句话也可以一段文本, 甚至可以是整个篇章, 但是下一句预测则需要输入为两个句子, 因此 BERT 在预训练阶段的输入形式统一为两段文字的拼接, 这与其他预训练模型相比有较大区别。

## 1. 模型结构

BERT 输入层采用了 WordPiece 分词，根据词频，决定是否将一个完整的词切分为多个子词（例如：单词 highest 可以被切分为 high 和 ##est 两个子词）以缓解 OOV 问题。对输入文本进行分词后，BERT 的输入表示由三部分组成：词嵌入（Token Embedding）、段嵌入（Segment Embedding）和位置嵌入（Position Embedding）。每个词的输入表示  $v$  可以表示为：

$$v = v^t + v^s + v^p$$

其中， $v^t$  代表词嵌入； $v^s$  代表段嵌入； $v^p$  代表位置嵌入；三种嵌入维度均为  $e$ 。

**词嵌入**用来将词转换为实值向量表示。完成分词后，切分完的子词通过词嵌入矩阵转化为词嵌入表示，假设子词对应的独热向量表示为  $e^t \in \mathbb{R}^{N \times |\mathcal{V}|}$ ，其对应的词嵌入  $v_t$  为：

$$v^t = e^t W^t \quad (6.44)$$

其中， $W^t \in \mathbb{R}^{|\mathcal{V}| \times e}$  表示词嵌入矩阵； $|\mathcal{V}|$  表示词表大小； $e$  表示词嵌入维度。

**段嵌入**用于区分不同词所属的段落（Segment），同一个段落中所有词的段嵌入相同。每个段落有其特有的段编码（Segment Encoding），段编码从 0 开始计数。通过段嵌入矩阵  $W^s$  将独热段编码  $e^s$  转化为段嵌入  $v^s$ ：

$$v^s = e^s W^s \quad (6.45)$$

其中， $W^s \in \mathbb{R}^{|\mathcal{S}| \times e}$  表示段嵌入矩阵； $|\mathcal{S}|$  表示段落数量； $e$  表示段嵌入维度。

**位置嵌入**用于表示不同词的绝对位置。将输入序列中每个词从左到右编号后，每个词都获得位置独热编码  $e^p$ ，通过可训练的位置嵌入矩阵  $W^p$  即可得到位置向量  $v^p$ ：

$$v^p = e^p W^p \quad (6.46)$$

其中， $W^p \in \mathbb{R}^{N \times e}$  表示位置嵌入矩阵； $N$  表示位置长度上限； $e$  表示位置嵌入维度。

BERT 的编码层采用多层 Transformer 结构，使用  $L$  表示所采用的层数， $H$  表示每层的隐藏单元数， $A$  是指自注意力头数量。在文献 [6] 给出了两种不同的参数设置，BERT<sub>BASE</sub> 使用  $L = 12$ ， $H = 768$ ， $A = 12$ ，总参数量为 110M，BERT<sub>LARGE</sub> 使用  $L = 24$ ， $H = 1024$ ， $A = 16$ ，总参数量为 340M。需要注意的是，与 GPT 中 Transformer 结构所采用的约束自注意力（Constrained Self-Attention）仅关注当前单元左侧上下文不同，BERT 采用的 Transformer 结构使用了双向多头自注意机制，不仅关注当前单元左侧上下文情况，也会关注右侧上下文。

## 2. 预训练任务

不同于传统的自回归语言建模方法，BERT 使用去噪自编码（Auto-Encoding）的方法进行预训练。接下来将详细介绍 BERT 所采用预训练任务。



**掩码语言建模：**传统的语言模型只能顺序或逆序进行建模，这意味着除了当前词本身外，每个词的表示只能利用词左侧（顺序）或右侧（逆序）的词信息。但对于大部分下游任务来说，单方向的信息是不充分的，因此同时利用两个方向的信息能带来更好的词表示，双向语言模型 ELMo 使用了顺序和逆序两个语言模型来解决这一问题。为了更好的利用上下文信息，让当前时刻的词表示同时编码“过去”和“未来”的文本，BERT 采用了一种类似于完形填空的任务，即掩码语言建模。在预训练时，随机将输入文本的部分单词掩盖（Mask），让模型预测被掩盖的单词，从而让模型具备根据上下文还原被掩盖的词的能力。

在 BERT 的预训练过程中，输入文本中 15% 的子词会被掩盖。具体来说，模型将被掩盖位置的词替换为特殊字符“[MASK]”，代表模型需要还原该位置的词。但在执行下游任务时，[MASK] 字符并不会出现，这导致预训练任务和下游任务不一致。因此，在进行掩盖时，并不总是直接将词替换为 [MASK]，而是根据概率从三种操作中选择一种：（1）80% 的概率替换为 [MASK]；（2）10% 的概率替换为词表中任意词；（3）10% 的概率不进行替换。

针对该掩码语言模型任务，使用  $x_1x_2...x_n$  表示原始文本，在经过上述掩码替换后得到输入为  $x'_1x'_2...x'_n$ 。对掩码替换后的输入按照 BERT 框架输入层处理后，得到 BERT 的输入表示  $v$ ：

$$X = [\text{CLS}]x'_1x'_2...x'_n[\text{SEP}] \quad (6.47)$$

$$v = \text{InputRepresentation}(X) \quad (6.48)$$

在编码层，对于输入表示  $v$  经过  $L$  层 Transformer，根据双向自注意力机制充分学习到文本中词语之间的联系，可以得到每个隐藏层输出以及最后的输出：

$$h^{(l)} = \text{Transformer-Block}(h^{(l-1)}) l \in 1, 2, \dots, L \quad (6.49)$$

其中  $h^{(l)} \in \mathbb{R}^{N \times d}$  表示第  $l$  层 Transformer 的隐藏层输出， $d$  表示隐藏层维度， $N$  为输入的最大序列长度， $h^{(0)} = v$  表示输入。为了简化标记，可以还可以省略中间层，使用如下公式表示最终输出：

$$h = \text{Transformer}(v) \quad (6.50)$$

其中  $h = h^{(L)}$ ，即模型最后一层的输出，得到最终上下文语义表示  $h \in \mathbb{R}^{N \times d}$ 。

根据对于原始文本进行的掩码情况，得到掩盖位置的下标集合  $\mathbb{M} = \{m_1, m_2, \dots, m_k\}$ ， $k$  表示掩码数量。BERT 模型输出层，首先根据集合  $\mathbb{M}$  中元素下标，从隐藏层得到的上下文语义表示  $h$  中抽取对应的表示  $h_{m_i}$ 。在此基础上，利用公式 6.44 中所给出的词向量矩阵  $W^t \in \mathbb{R}^{\mathbb{V} \times e}$  将其映射到词空间表示，并通过如下公式计算对应词表上的概率分布  $P_i$ ：

$$P_i = \text{Softmax}(h_{m_i} W^t{}^\top + b^0) \quad (6.51)$$



其中  $\mathbf{b}^0 \in \mathbb{R}^V$  表示全连接层偏置。最后利用  $P_i$  与原始单词独热向量表示之间的交叉熵损失学习模型参数。

**下一句预测：**通过掩码语言建模，BERT 能够根据上下文还原掩码单词，从而具备构建对文本的语义表示能力。然而，对于阅读理解、语言推断等需要输入两段文本的任务来说，模型尚不具备判断两段文本关系的能力。因此，为了学习到两段文本间的关联，BERT 引入了第二个预训练任务：下一句预测（NSP）。

顾名思义，下一句预测的任务目标是预测两段文本是否构成上下句的关系。具体来说，对于句子 A 和句子 B，若语料中这两个句子相邻，则构成正样本，若不相邻，则构成负样本。在预训练时，一个给定的句子对，有 50% 的概率将其中一句替换成来自其他段落的句子。这样可以将训练样本的正负例比例控制在 1:1。

该预训练任务与掩码语言模型任务非常类似，主要区别在于输出层。在输入层，对于给定的经过掩码处理的句子对  $x^{(1)} = x_1^{(1)} x_2^{(1)} \dots x_n^{(1)}$  和  $x^{(2)} = x_1^{(2)} x_2^{(2)} \dots x_m^{(2)}$ ，经过如下处理得到 BERT 的输入表示  $\mathbf{v}$ ：

$$X = [\text{CLS}]x_1^{(1)}x_2^{(1)}\dots x_n^{(1)}[\text{SEP}]x_1^{(2)}x_2^{(2)}\dots x_m^{(2)}[\text{SEP}] \quad (6.52)$$

$$\mathbf{v} = \text{InputRepresentation}(X) \quad (6.53)$$

在 BERT 编码层，与掩码语言模型一样，通过  $L$  层 Transformer 编码，可以充分学习文本每个单词之间的关联，并最终得到文本语义表示：

$$\mathbf{h} = \text{Transformer}(\mathbf{v}) \quad (6.54)$$

下一句预测任务的输出层目标是判断输入文本  $x^{(2)}$  是否是  $x^{(1)}$  的下一个句子，可以转化为二分类问题。在该任务中，BERT 使用输入文本的开头添加 [CLS] 所对应的表示  $\mathbf{h}_{[\text{CLS}]}$  进行分类预测。使用全连接层预测输入文本的分类概率  $P \in \mathbb{R}^2$ ：

$$P = \text{Softmax}(\mathbf{h}_{[\text{CLS}]} \mathbf{W}^P + \mathbf{b}^o) \quad (6.55)$$

其中， $\mathbf{W}^P \in \mathbb{R}^{d \times 2}$  为全连接层权重； $\mathbf{b}^o$  表示全连接层偏置。根据分类概率  $P$  与真实分类标签之间的交叉熵损失，学习模型参数。

#### 6.4.4 序列到序列预训练语言模型 BART

在之前的章节中，我们介绍了适合自然语言生成的自回归式单向预训练语言模型 GPT 和适合自然语言理解任务的掩码预训练语言模型 BERT，自回归的 GPT 缺乏了上下文语境信息，BERT 虽然能利用上下文信息，但其预训练任务使其在自然语言生成任务上表现不佳。本节中，我们介绍一种符合自然语言生成任务需求的预训练模型 BART (Bidirectional and Auto-Regressive Transformers)

[15]。BART 兼具上下文语境信息的编码器和自回归特性的解码器，配合上针对自然语言生成制定的预训练任务，使其格外契合生成任务的场景。

BART 模型也是使用基于 Transformer 的序列到序列结构, 相较于标准的 Transformer, BART 选择了 GeLU 而不是 ReLU 作为激活函数, 并且使用了正态分布  $N(0, 0.02)$  进行初始化。Transformer 编码器具备双向编码上下文信息的能力, 单向的 Transformer 解码器又满足生成任务的需求。BART 模型的基本结构如图 6.6 所示, 结合了双向 Transformer 编码器以及单向的自回归解码器。

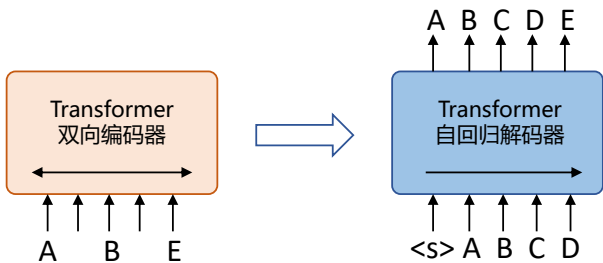


图 6.6 BART 的神经网络结构<sup>[15]</sup>

### 1. 预训练任务

BART 的预训练过程采用的是对含有噪声的输入文本进行去噪重构方法, 属于去噪自编码器 (Denoising Autoencoder)。BART 使用双向编码对引入噪声的文本进行编码。然后, 单向的自回归解码器通过自回归方式顺序重构原始文本。编码器最后一层隐藏层表示参与解码器每一层的计算。BART 的预测过程与 BERT 独立预测掩码位置的词有很大不同。因此, BART 的预训练任务主要关注如何引入噪声。BART 模型使用了五种方式在输入文本上引入噪音:

- **单词掩码 (Token Masking)**: 随机从输入文本中选择一些单词, 将其替换为掩码 ([MASK]) 标记, 类似于 BERT。该噪声需要模型具备预测单个单词的能力。
- **单词删除 (Token Deletion)**: 随机从输入文本中删除一部分单词。该噪声除了需要模型预测单个单词的能力, 还需要模型能定位缺失单词的位置。
- **文本填充 (Text Infilling)**: 随机将输入文本中多处连续的单词 (称作文本片段) 替换为一个掩码标记。文本片段的长度服从  $\lambda = 3$  的泊松分布。当文本片段长度为 0 时, 相当于插入一个掩码标记。该噪音需要模型能识别一个文本片段有多长, 并具备预测缺失片段的能力。
- **句子排列变换 (Sentence Permutation)**: 对于一个完整的句子, 根据句号将其分割为多个子句, 随机打乱子句的顺序。该噪音需要模型能一定程度上理解输入文本的语义, 具备推理前后句关系的能力。
- **文档旋转变换 (Document Rotation)**: 随机选择输入文本中的一个单词, 以该单词作为文档

的开头，并旋转文档。该噪音需要模型具备找到原始文本开头的能力。图6.7给出了各种加噪方案的示例，输入加噪过程可以对这些方式进行组合使用。

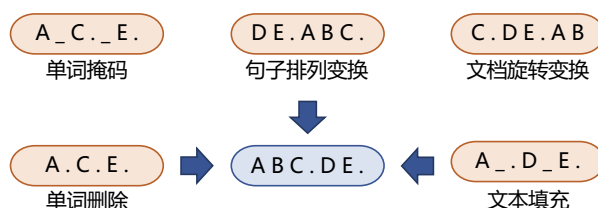


图 6.7 BART 各类型加噪方式示例

可以看到，BART 的预训练时包含单词、句子和文档多种级别的任务，除了上述噪声之外，其他任意形式的文本噪声也是适用的。实验表明，使用文本填充任务能在下游任务上普遍取得性能提升，在文本填充噪音的基础上添加句子级别的去噪任务还能带来小幅提升。另外，尽管 BART 的预训练任务主要是为自然语言生成任务设计，但是它在一些自然语言理解任务上也展现出了不错的性能。

## 2. 模型精调

BART 预训练模型具备文本表示和生成能力，因此不仅适用于文本理解任务，也适用于文本生成任务，但是用于不同类型任务时，其精调方式有所不同。

对于序列分类任务，BART 模型的编码器和解码器的输入相同，但是将解码器最终时刻的隐藏层状态作为输入文本的语义向量表示，并利用线性分类器进行标签预测。利用标注数据和模型输出结果对模型参数进行调整。整个过程与 BERT 模型类似，在句子末尾添加特殊标记，利用该位置所对应的隐藏层状态表示文本。

对于生成式的任务，比如生成式文本摘要 (Abstractive Summarization)、生成式问答 (Abstractive Question Answering) 等任务，精调时模型输入为任务所给定的输入文本，解码器所产生的文本与任务的目标文本构成学习目标。

对于机器翻译任务，由于其输入文本和输出文本是两种不同的语言，使用的不同词汇集合，因此不能采用与生成式任务相同的方法。为了解决上述问题，研究人员们提出了将 BART 模型的输入层前增加小型 Transformer 编码器，将源语言文本映射到目标语言的输入表示空间。同时，为了解决两段模型训练过程不匹配的问题，采取分阶段的训练方法。详细过程可以参见文献 [15]。

## 6.4.5 预训练语言模型的应用

在预训练阶段，大规模的数据使预训练语言模型有效地学习到了语言的通用语义表示，微调 (Fine-tuning) 则是利用预训练语言模型的主要范式，其目的是基于学习到的通用表示，针对目标领域的特性对模型进行调整，使其更适合下游任务。相较于深入了解下游任务的特有知识，为其

精心设计特别的模型，预训练模型只用转换下游任务的输入输出形式后进行微调，即可获得相当有竞争力的性能。本节将以 BERT 为例，针对三种经典的自然语言处理下游任务介绍如何微调预训练模型。

### 1. 单句文本分类

单句文本分类是自然语言处理中最为常见任务之一，其目的是判断一段文本所属的类别。例如，判断一段电影评价的情感倾向是正面还是负面，判断一篇新闻所属的类别。使用 BERT 进行单句文本分类任务时，对于将要进行单句文本分类的句子，BERT 首先使用 WordPiece 进行分词，得到分词后的句子  $w_1 w_2 \dots w_n$ ，分别添加特殊字符 [CLS] 和 [SEP] 到句首和句尾，再经过输入层将其转换为 BERT 编码层所需的输入表示，其过程可以表述如下：

$$w = [\text{CLS}]w_1 w_2 \dots w_n, [\text{SEP}] \quad (6.56)$$

$$v = \text{InputLayer}(w) \quad (6.57)$$

随后，输入表示进入编码层，经过多层 Transformer 编码，每个位置的表示都通过自注意力机制进行充分交互，在最后一层得到具备丰富上下文信息的表示  $h$ 。和预训练阶段时使用 [CLS] 进行 NSP 任务类似，在进行文本分类时，模型使用 [CLS] 位置的隐藏层表示  $h_{[\text{CLS}]}$  进行预测。在编码层之后，模型通过一个全连接层预测输入文本对应的类别。其过程可以表述如下：

$$H = \text{BERT}(V) \quad (6.58)$$

$$P = \text{Softmax}(h_{[\text{CLS}]}W + b) \quad (6.59)$$

其中， $W \in \mathbb{R}^{d \times K}$  和  $b \in \mathbb{R}^K$  分别为全连接层的权重和偏置， $K$  为类别总数。

在得到概率  $P$  后，若为训练阶段，则可以计算  $P$  与真实标签间的交叉熵对模型参数进行训练。若为预测阶段，则可以取概率最高的一项作为输入文本的类别。

### 2. 句子对分类

句子对分类需要预测一对有关联的句子的类别，例如判断一个句子的意思是否蕴含在另一个句子之中。句子对分类与单句分类的区别在于处理的输入不同，BERT 处理这两个任务时，也主要在输入上有所区别。对于分词后的两个句子  $w_1^{(1)}, w_2^{(1)}, \dots, w_n^{(1)}$  和  $w_1^{(2)}, w_2^{(2)}, \dots, w_m^{(2)}$ ，BERT 使用 [SEP] 作为分隔符，将两个句子拼接到一起，在输入层转换为输入表示。

$$w = [\text{CLS}]w_1^{(1)}w_2^{(1)} \dots w_n^{(1)}[\text{SEP}]w_1^{(2)}w_2^{(2)} \dots w_m^{(2)}[\text{SEP}] \quad (6.60)$$

$$v = \text{InputLayer}(w) \quad (6.61)$$

其中,  $\mathbf{v} \in \mathbb{R}^{(n+m+3) \times d}$ ,  $d$  为模型隐藏层维度,  $n$  和  $m$  分别代表第一个句子和第二个句子的长度。得到输入表示后, 剩下的流程与单句文本分类一致, 此处不再赘述。

### 3. 序列标注

序列标注任务需要解决的是字符级别的分类问题, 其应用范围非常广泛, 可用于分词, 词性标注和命名实体识别等自然语言处理基础任务。以命名实体识别为例, 在用序列标注的形式完成该任务时, 需要对输入文本中的每一个词预测一个相应的标签, 再根据整个序列的标签抽取出句子中的实体词。

传统的序列标注方法通常以词为输入的最小粒度, 而在使用 BERT 等预训练模型时, 通常会使用分词器将词分割为更小粒度的子词, 这会破坏序列标注中词和标签一对一的关系。为了处理这种情况, 可以让一个词的所有子词都保持原标签, 或者只让第一个子词参与训练, 预测时也只考虑第一个子词的预测结果。在完成分词后, 将输入序列送入输入层转化为词向量, 再将词向量送入预训练模型得到最终的隐藏层表示, 其过程可以表示如下:

$$\mathbf{w} = [\text{CLS}]w_1^{(1)}w_2^{(1)}\dots w_n^{(1)}[\text{SEP}] \quad (6.62)$$

$$\mathbf{v} = \text{InputLayer}(\mathbf{w}) \quad (6.63)$$

$$\mathbf{h} = \text{BERT}(\mathbf{v}) \quad (6.64)$$

其中,  $\mathbf{v} \in \mathbb{R}^{(n+2) \times d}$  为模型输入层的输出,  $\mathbf{h} \in \mathbb{R}^{(n+2) \times d}$  为预训练模型最后一层的隐藏层表示,  $n$  为分词后的序列长度,  $d$  为模型隐藏层维度。

在得到了隐藏层表示后, 需要使用一个分类器对预测每个词在标签集上的概率分布:

$$\mathbf{P} = \text{Softmax}(\mathbf{h}_i \mathbf{W} + \mathbf{b}) \quad (6.65)$$

其中  $\mathbf{h}_i$  是隐藏层表示  $\mathbf{h}$  在第  $i$  时刻的分量,  $i \in \{1, \dots, n\}$ 。得到概率分布后, 可以使用交叉熵损失学习模型参数。除此以外, 还可以使用条件随机场等方法进一步提升序列标注性能, 感兴趣的读者可以参考第 7 章信息抽取对序列标注任务作进一步了解。

## 6.5 大规模语言模型

自 2020 年 Open AI 发布了包含 1750 亿参数的生成式大规模预训练语言模型 Generative Pre-trained Transformer 3 (GPT 3) [16] 以来, 包含 Google、Meta、百度、智源等公司和研究机构都纷纷发布了包括 PaLM[17]、LaMDA[18]、T0[19] 等为代表的不同大规模语言模型 (Large Language Model, LLM), 也称大模型。大模型在文本生成、少样本学习、零样本学习、推理任务等方面取得了非常大的进展。表 6.2 给出了截止 2023 年 1 月典型大规模语言模型的基本情况。我们可以看到从 2022 年开始大模型呈现爆发式的增长, 各大公司和研究机构都在发布各种不同类型的大模型。

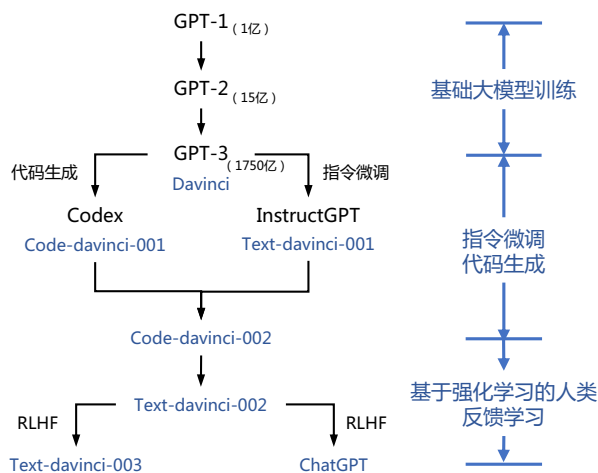
表 6.2 典型大规模语言模型汇总

模型名称	参数量	训练单词数	研发机构	发布时间
ChatGPT	1750 亿	3000 亿	OpenAI	2022 年 11 月
Galactica	1200 亿	4500 亿	Meta AI	2022 年 11 月
BLOOMZ	1760 亿	3660 亿	BigScience	2022 年 11 月
U-PaLM	5400 亿	7800 亿	Google Research	2022 年 10 月
CodeGeeX	130 亿	8500 亿	清华大学	2022 年 9 月
PaLM	5400 亿	7800 亿	Google Research	2022 年 4 月
ERNIE 3.0 Titan	2600 亿	–	Baidu	2021 年 12 月
FLAN	1370 亿	–	Google	2021 年 9 月
GPT-3	1750 亿	3000 亿	OpenAI	2020 年 5 月
T5	110 亿	340 亿	Google	2019 年 10 月
RoBERTa	3.55 亿	22000 亿	Meta AI	2019 年 7 月
GPT-2	15 亿	100 亿	OpenAI	2019 年 2 月
BERT	3 亿	1370 亿	Google	2018 年 10 月
GPT-1	1 亿	–	OpenAI	2018 年 6 月

2022 年 11 月 ChatGPT（Chat Generative Pre-trained Transformer）自发布起就引起了极大的关注，5 天内注册用户超 100 万，在系统推出仅两个月后，月活跃用户估计已达 1 亿，并与 Bing 深度搜索结合构造了对话式搜索新范式。ChatGPT 允许用户使用自然语言与系统交互，便可实现包括问答、分类、摘要、翻译、聊天等从理解到生成的各种任务。在很多自然语言理解的开放领域识别结果上都达到了非常好的效果，甚至在一些任务上超过了针对特定任务设计并且使用有监督数据进行训练的模型。ChatGPT 的生成能力也非常优秀，针对用户提出的各种各样的问题，大多数情况下都可以生成出语言通畅、有一定逻辑并且多样化的长文本。ChatGPT 的整个发展和技术演进过程如图6.8所示。图中黑色字表示论文相关介绍中的名字，蓝色字表示 Open AI 的 API 中的模型名称。整个过程我们可以看到大体上可以分为三个主要阶段：第一个阶段是基础大模型训练阶段，该阶段主要完成长距离语言模型的预训练；第二阶段是指令微调（Instruct Tuning）和代码生成训练阶段，通过给定指令进行微调的方式使得模型具备完成各类任务的能力，通过代码预训练使得模型具备代码生成的能力；第三个阶段是加入更多人工提示词，并利用基于强化学习的方式，使得模型输出更贴合人类需求。

本章中，我们以 ChatGPT 为例，介绍大模型训练三个基本阶段：基础大模型训练、指令微调以及人类反馈。需要特别说明的是，由于在本书写作阶段，包括 ChatGPT 在内的绝大部分大模型的技术细节还没有完全公开，一些已经公开的研究内容和方法也仍然需要更多时间进行验证，阅读该部分内容需要大家更多的独立思考和批判精神，并结合当前的研究进行理解。



图 6.8 ChatGPT 发展历程<sup>[20]</sup>

### 6.5.1 基础大模型训练

文献 [16] 介绍了 GPT-3 模型的训练过程, 包括模型架构、训练数据组成、训练过程以及评估方法。由于 GPT-3 并没有开放源代码, 根据论文直接重现整个训练过程并不容易, 因此文献 [21] 介绍了根据 GPT-3 的描述复现的过程, 并构造开源了系统 OPT (Open Pre-trained Transformer Language Models)。

在模型架构方面不论是 GPT-3 还是 OPT 所采用的模型结构都与我们在本章第6.4.2所介绍的 GPT-2 模型一样, 都采用由多层 Transformer 组成的单向语言模型, 采用自回归方式从左到右对文本序列建模。但是针对不同的规模的参数量要求, 其所使用的层数、自注意力头数、嵌入表示维度大小等具体参数各不相同。OPT 给出了 8 种模型参数的细节, 如表6.3所示。采用 AdamW 优化器进行优化, 其参数  $(\beta_1, \beta_2)$  设置为  $(0.9, 0.95)$ 。其他参数细节可以参考文献 [21]。

在预训练语料集方面, 根据文献 [16] 中的报道, GPT-3 中通过主要包含经过过滤的 Common Crawl 数据集<sup>[22]</sup>、WebText2、Books1、Books2 以及英文 Wikipedia 等数据集合。其中 CommonCrawl 的原始数据有 45TB, 进行过滤后仅保留了 570GB 的数据。通过子词方式对上述语料进行切分, 大于一共包含 5000 亿子词。为了保证模型使用更多高质量数据进行训练, 在 GPT-3 训练时, 赋予不同的语料来源的不通过的权重采样。在完成 3000 亿子词训练时, 英文 Wikipedia 的语料平均训练轮数为 3.4 次, 而 Common Crawl 和 Books 2 仅有 0.44 次和 0.43 次。由于 Common Crawl 数据集合的过滤过程繁琐复杂, OPT 则采用了混合 RoBERTa<sup>[23]</sup>、Pile<sup>[24]</sup> 和 PushShift.io Redit<sup>[25]</sup> 数据的方法。由于这些数据集合中包含的绝大部分都是英文数据, 因此 OPT 也从 Common Crawl 数据集中抽取了部分非英文数据加入训练语料。

表 6.3 OPT 不同模型规模下的具体参数细节<sup>[21]</sup>

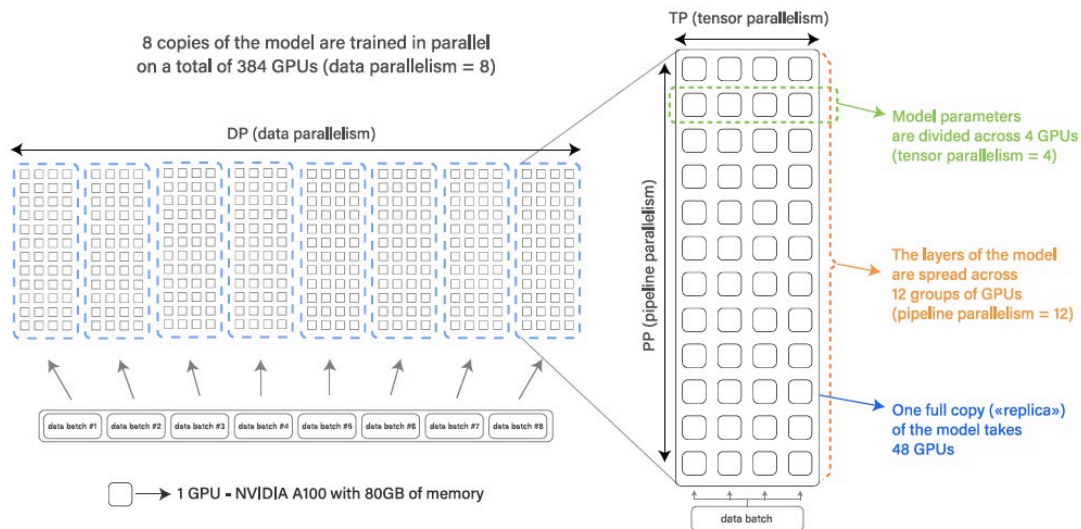
参数规模	层数	自注意力头数	嵌入向量维度	学习率	全局批次大小
125M	12	12	768	6.0e-4	50 万
350M	24	16	1024	3.0e-4	50 万
1.3B	24	32	2048	2.0e-4	100 万
2.7B	32	32	2560	1.6e-4	100 万
6.7B	32	32	4096	1.2e-4	200 万
13B	40	40	5120	1.0e-4	400 万
30B	48	56	7168	1.0e-4	400 万
66B	64	72	9216	0.8e-4	200 万
175B	96	96	12288	1.2e-4	200 万

由于模型参数量和所使用的数据量都非常巨大，普通的服务器单机无法完成训练过程，因此通常采用分布式架构完成训练。GPT-3 和 OPT 中没有对这个部分给出详细的描述。文献 [16]GPT-3 仅介绍了训练过程全部使用 NVIDIA V100 GPU，文献 [21] 介绍了 OPT 使用了 992 块 NVIDIA A100 80G GPU，采用全分片数据并行（Fully Shared Data Parallel）<sup>[26]</sup> 以及 Megatron-LM 张量并行（Tensor Parallelism）<sup>[27]</sup>，整体训练时间将近 2 个月。BLOOM<sup>[28]</sup> 则公开了更多在硬件和所采用的系统架构方面的细节。该模型的训练一共花费 3.5 个月，使用 48 个计算节点，每个节点包含 8 块 NVIDIA A100 80G GPU（总计 384GPU）。节点内容包含 4 NVLink 用于节点内部 GPU 之间通信，节点之间采用四个 Omni-Path 100 Gbps 网卡构建的增强 8 维超立方体全局拓扑网络通信。

BLOOM 使用 Megatron-DeepSpeed<sup>[29]</sup> 框架进行训练，主要包含两个部分：Megatron-LM 提供张量并行能力和数据加载原语；DeepSpeed<sup>[30]</sup> 提供 ZeRO 优化器、模型流水线以及常规的分布式训练组件。通过这种方式可以实现数据、张量和流水线三维并行，如图6.9所示。数据并行（Data Parallelism）将模型构建多个副本，每个副本放置在不同的设备上，并分别针对一部分数据并行进行训练，在每个训练步结束时同步副本间数据。张量并行（Tensor Parallelism）将模型的单个层划分到不同设备中，这样可以避免将所有激活或梯度张量都放置在一个 GPU 上，这种方法也称为水平并行或层内模型并行。流水线并行（Pipeline Parallelism）将模型不同层放置在多个 GPU 中，每个 GPU 中仅包含部分的层，这种方法也称为垂直并行。ZeRO（Zero Redundancy Optimizer）优化器<sup>[31]</sup> 允许不同的进程只保存一小部分数据（训练步骤所需的参数、梯度和优化器状态）。通过上述四个步骤可以实现数百个 GPU 的高效并行计算。

基础大模型构建了长文本的建模能力,使得模型具有语言生成能力,根据输入的提示词(Prompt),模型可以生成文本补全句子。也有部分研究人员认为，语言模型建模过程中也隐含的构建了包括事实性知识 (Factual Knowledge) 和常识知识 (Commonsense) 在内的世界知识 (World Knowledge)。



图 6.9 BLOOM 并行结构<sup>[28]</sup>

### 6.5.2 指令微调

以 BERT 为代表的预训练语言模型需要根据任务数据进行微调 (Fine-tuning)，这种范式可以应用于参数量在几百万到几亿规模的预训练模型。但是针对数十亿甚至是数百亿规模的大模型，针对每个任务都进行微调的计算开销和时间成本几乎都是不可接受的。因此，研究人员们提出了指令微调 (Instruction Finetuning)<sup>[32]</sup> 方案，将大量各类型任务，统一为生成式自然语言理解框架，并构造训练语料进行微调。

例如，可以将情感倾向分析任务，通过如下指令，将贬义和褒义的分类问题转换到生成式自然语言理解框架：

For each snippet of text, label the sentiment of the text as positive or negative.

Text: this film seems thirsty for reflection, itself taking on adolescent qualities.

Label: [positive / negative]

利用有标注数据集，再结合上述指令模板，就可以生成大量用于微调的训练数据。利用这些训练数据，就可以在一个大模型中同时训练大量不同的任务。当前的研究工作表明，这种训练方法可以使得模型具有很好的任务泛化能力<sup>[33]</sup>，很多没有出现在指令微调训练语料中的任务也可以很好的完成，在零样本和少样本的情况下获得非常好的任务性能。

FLAN-T5<sup>[32]</sup> 中通过混合之前的 Muffin<sup>[33]</sup>、T0-SF<sup>[19]</sup>、NIV2<sup>[34]</sup> 以及思维链 (Chain-of-thought, CoT) 混合微调等工作，使用 473 个数据集，针对 146 个任务类型，构造了 1836 个任务。数据

集合包括 SQuAD、MNLI、CoNLL2003 等。任务类型包括阅读理解、问题生成、常识推理、对话上下文生成、完型填空、命名实体识别、文本分类等。任务是指 <数据集, 任务类型> 的组合, 比如 SQuAD 既可以构造阅读理解任务, 也可以用来构造问题生成任务。Muffin、T0-SF、NIV2 是之前类似研究中所构造的任务, FLAN-T5 中直接进行了使用。CoT 混合微调模型任务, 则与前面三种不同, 通过使用之前的任务, 但是在任务的指令中增加思维链, 以及在目标结果中增加思维链, 构造新的任务, 试图用于提升模型的在未知任务上的推理能力。FLAN-T5 中使用构造 9 个数据集合用于思维链任务, 包括数学推理、多跳推理以及自然语言推断等。图6.10给出了指令和训练目标中添加思维链的样例。

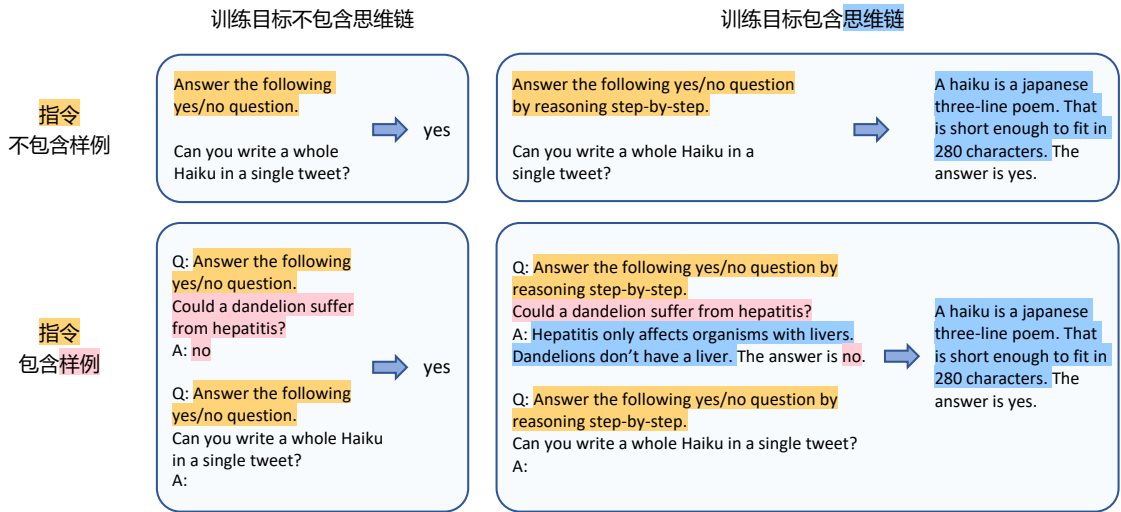


图 6.10 指令和训练目标中思维链示例<sup>[32]</sup>

通过指令微调, 大模型学习到了如何响应人类指令, 而是可以根据指令直接能够生成合理的答案。基础大模型 GPT-3 在处理任务上通常仅能生成一些句子, 需要后续模块根据句子内容再提取答案。由于指令微调阶段训练了非常多的任务, 大模型任务能力可以泛化到之前没有见过的任务上, 这使得模型初步具备了回答人们提出的任何指令的可能。这种能力对于大模型来说至关重要, 使其可以在开放领域有很好的表现。思维链以及代码生成的引入, 又在一定程度上提升了大模型的推理能力。指令微调使得大模型在处理任务的能力上有了质的飞跃。

### 6.5.3 人类反馈

经过指令微调后的模型, 虽然在开放领域任务能力表现优异, 但是模型输出的结果通常是简单答案, 这与人类的回答相差很大。因此需要进一步优化模型, 使其可以生成更加贴近人类习惯的

文本内容。但是，由于自然语言处理语料集中所包含的答案，通常都是简短的标签，基本没有类似人类回答的有监督数据。如果全部使用人工，将自然语言处理任务数据集进行改造，构造类似人工回答的训练语料，所需要的规模过于庞大，时间成本和人工成本都过于高昂。因此，Open AI 提出了使用基于人类反馈的强化学习方法（Reinforcement Learning from Human Feedback, RLHF），从而大幅度降低了数据集构建成本，但是达到了非常好的效果。

在文献 [35] 中，使用的 RLHF 方法与风格续写<sup>[36]</sup> 以及文本摘要<sup>[37]</sup> 中所使用的方法非常类似。首先，仍然需要收集一定数量的用户输入以及期望的系统输出。Open AI 团队针对初始数据收集，制定了严格的标准和规范，雇佣了 40 人的团队完成该项工作。再结合通过线上 API 所收集到的高质量数据，一共构造了约 11.28 万标注集合。基于上述标注数据和初始大模型，整个 RLHF 算法训练过程如图 6.11 所示。

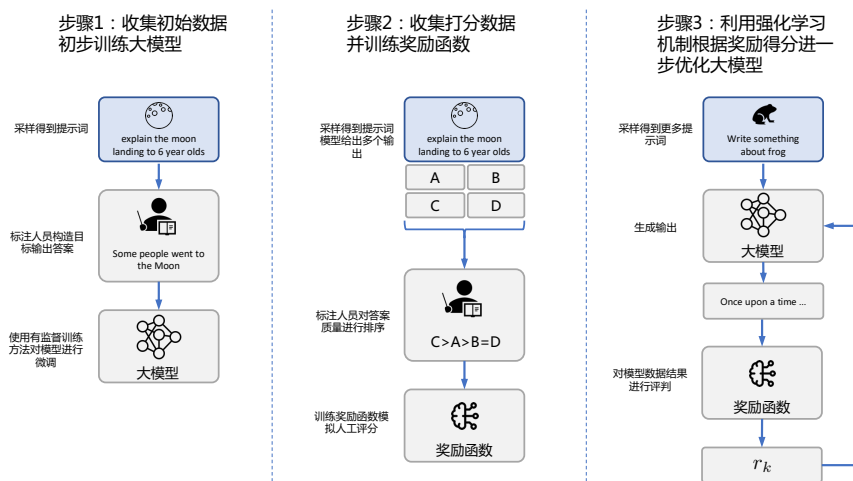


图 6.11 基于人类反馈的强化学习方法基本步骤图<sup>[35]</sup>

RLHF 算法主要分为如下三个步骤：

- (1) 收集初始数据初步训练大模型，从得的指令集中采样部分数据，对初始的大模型进行有监督微调；
- (2) 收集打分数据并训练奖励（Reward）函数，使用无监督数据，收集模型的多个输出结果，评价人员根据模型的输出结果进行对比评价，确定模型输出的结果排序，并利用该评分训练奖励函数，使其未来可以对模型输出的优劣进行判断；
- (3) 使用更多的指令数据，利用奖励函数输出的得分，利用强化学习机制根据奖励得分进一步优化大模型。

## 6.6 语言模型评价方法

语言模型最直接的测评方法就是使用模型计算测试集的概率，或者利用交叉熵（Cross-entropy）和困惑度（Perplexity）等派生测度。

对于一个平滑过的概率  $P(w_i|w_{i-n+1}^{i-1})$  的  $n$  元语法模型，可以用下列公式计算句子  $P(s)$  的概率：

$$P(s) = \prod_{i=1}^n P(w_i|w_{i-n+1}^{i-1}) \quad (6.66)$$

对于由句子  $(s_1, s_2, \dots, s_n)$  组成的测试集  $T$ ，可以通过计算  $T$  中所有句子概率的乘积来得到整个测试集的概率：

$$P(T) = \prod_{i=1}^n P(s_i) \quad (6.67)$$

交叉熵的测度则是利用预测和压缩的关系进行计算。对于  $n$  元模型  $P(w_i|w_{i-n+1}^{i-1})$ ，文本  $s$  的概率为  $P(s)$ ，在数据  $s$  上  $n$  元模型  $P(w_i|w_{i-n+1}^{i-1})$  的交叉熵为：

$$H_p(s) = -\frac{1}{W_s} \log_2 P(s) \quad (6.68)$$

其中， $W_s$  为文本  $s$  的长度，该公式可以解释为：利用压缩算法对  $s$  中的  $W_s$  个词进行编码，每一个编码所需要的平均比特位数。

困惑度的计算可以视为模型分配给测试集中每一个词汇的概率的几何平均值的倒数，它和交叉熵的关系为：

$$PP_s(s) = 2^{H_p(s)} \quad (6.69)$$

交叉熵和困惑度越小，语言模型性能就越好。不同的文本类型其合理的指标范围是不同的，对于英文来说， $n$  元语言模型的困惑度约在 50 到 1000 之间，相应的，交叉熵在 6 到 10 之间。

## 6.7 延伸阅读

随着深度学习的发展，预训练语言模型正逐渐成为自然语言处理的基础模型。预训练语言模型通过设计特定的自监督训练目标，有效地从大量标记和未标记数据中获取知识，并存储到巨大的参数中。通过在特定任务上进行微调，隐藏在巨大参数中的丰富知识可以使各种下游任务受益。尽管预训练-微调范式取得了巨大的成功，但在实际场景中应用该范式依旧面临许多困难与挑战，在本节中，我们探讨限制预训练模型在真实场景中应用的三点困难，并简单介绍一些前沿的解决方案。

(1) 更好的预训练语言模型迁移范式。在下游任务上微调预训练模型时，通常会在预训练模型最后添加任务特有的分类层，模型的优化目标是基于下游任务的分类任务，而预训练阶段的进

行的是语言模型建模任务。预训练和微调阶段不一致的优化目标为预训练模型的迁移带来了隐形的阻碍，导致其需要更多的训练样本才能使预训练模型“适配”到下游任务上。受到 GPT-3 启发，一种新的预训练模型迁移范式，提示学习（Prompt Learning）<sup>[38]</sup>，逐渐走入研究者的视野。如果说微调是让预训练模型“迁就”下游任务，提示学习则可以看作是让下游任务“迁就”预训练模型。具体来说，提示学习需要将下游任务转化为预训练任务的形式，举一个在掩码预训练语言模型 BERT 上运用提示学习的例子，若对“这部电影剧情拖沓，演员演技差，我很不喜欢。”进行电影情感分类，提示学习会在待分类的句子后面添加模版“这是部 [MASK] 电影。”，让预训练语言模型在 [MASK] 位置预测标签相关词，若预测结果为坏、差、烂等负面情感的词，则这句话的情感为负面。通过将下游任务转化为预训练任务，提示学习减少了与训练阶段和下游任务阶段的差距，让模型能快速完成迁移，因此提示学习在少样本场景下表现出色，能高效地利用预训练模型。提示学习在文本分类<sup>[39-41]</sup>、命名实体识别<sup>[42, 43]</sup>、阅读理解<sup>[44, 45]</sup> 等任务的小规模语料学习上都取得了一定的效果。

（2）绿色低碳预训练。预训练阶段需要在大量无监督语料上对大模型进行语言模型建模，这一阶段需要大量计算资源支撑。例如，BERT-base 需要在 64 块 TPU 上进行 4 天预训练，对于大部分科研人员和企业来说，训练一个自己的预训练模型所需的代价是非常高昂的，并且模型训练过程中会耗费大量的能源，对经济和环保带来额外的负担。因此，减少预训练阶段的成本是一项重大挑战。文献 [46] 提出了 ELECTRA 算法，基于对抗的思想，使用替换词检测作为预训练任务，只需要 1/4 的预训练计算量，就可以实现和其他预训练模型相似的性能。除此之外，还有一些模型通过数据集选择来大幅降低预训练模型训练时间<sup>[47]</sup>，根据领域特性进行数据选择<sup>[48]</sup>，利用领域之间关联<sup>[49]</sup>，自动优化超参数选择<sup>[50]</sup> 等方法降低预训练模型的计算消耗。

（3）高效微调方法。现有的微调范式需要更新预训练模型的所有参数，这意味着对于每个下游任务来说，每次微调都会得到一个不同的模型。而预训练模型的参数量越来越大，微调所有参数需要耗费大量计算资源，存储微调后的模型需要占用大量存储资源，在工业界中，模型是需要部署在服务器上供用户调用的，为每个任务都部署微调后的模型需要占用大量显存资源。为了解决传统微调方法耗费资源过多，使用成本过高的问题，大量研究者开始探究如何进行高效微调<sup>[51-53]</sup>，即冻结模型大部分参数，只更新部分参数来完成下游任务。最近的研究表明，只需要更新预训练模型的 1% 的参数，就可以实现和微调所有参数相似的性能，大大降低了预训练模型的使用成本。

## 6.8 习题

- （1）试比较不同平滑方法的优缺点。
- （2）预训练语言模型中常用的子词（Subword）是为了解决什么问题？
- （3）常见的预训练任务有哪些？这些预训练任务的目的是什么？
- （4）预训练-微调范式可能存在哪些问题？

## 参考文献

- [1] Bengio Y, Ducharme R, Vincent P. A neural probabilistic language model[J]. Advances in neural information processing systems, 2000, 13.
- [2] Mikolov T, Karafiát M, Burget L, et al. Recurrent neural network based language model.[C]// Interspeech: volume 2. Makuhari, 2010: 1045-1048.
- [3] Pham N Q, Kruszewski G, Boleda G. Convolutional neural network language models[C]// Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing. 2016: 1153-1162.
- [4] Sukhbaatar S, Weston J, Fergus R, et al. End-to-end memory networks[C]//Advances in neural information processing systems. 2015: 2440-2448.
- [5] Peters M, Neumann M, Iyyer M, et al. Deep contextualized word representations[C]//Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers): volume 1. 2018: 2227-2237.
- [6] Devlin J, Chang M W, Lee K, et al. Bert: Pre-training of deep bidirectional transformers for language understanding[C]//Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers). 2019: 4171-4186.
- [7] Good I J. The population frequencies of species and the estimation of population parameters[J]. Biometrika, 1953, 40(3-4):237-264.
- [8] Katz S. Estimation of probabilities from sparse data for the language model component of a speech recognizer[J]. IEEE transactions on acoustics, speech, and signal processing, 1987, 35(3):400-401.
- [9] Jelinek F. Interpolated estimation of markov source parameters from sparse data[C]//Proc. Workshop on Pattern Recognition in Practice, 1980. 1980.



- [10] WITTEN I, BELL T. The zero-frequency problem: estimating the probabilities of novel events in adaptive text compression[J]. IEEE transactions on information theory, 1991, 37(4):1085-1094.
- [11] Kneser R, Ney H. Improved backing-off for m-gram language modeling[C]//1995 international conference on acoustics, speech, and signal processing: volume 1. IEEE, 1995: 181-184.
- [12] Sundermeyer M, Schlüter R, Ney H. Lstm neural networks for language modeling[C]//Thirteenth annual conference of the international speech communication association. 2012.
- [13] Deng J, Dong W, Socher R, et al. Imagenet: A large-scale hierarchical image database[C]//2009 IEEE conference on computer vision and pattern recognition. Ieee, 2009: 248-255.
- [14] Radford A, Wu J, Child R, et al. Language models are unsupervised multitask learners[J]. OpenAI blog, 2019, 1(8):9.
- [15] Lewis M, Liu Y, Goyal N, et al. Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension[C]//Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics. 2020: 7871-7880.
- [16] Brown T, Mann B, Ryder N, et al. Language models are few-shot learners[J]. Advances in neural information processing systems, 2020, 33:1877-1901.
- [17] Chowdhery A, Narang S, Devlin J, et al. Palm: Scaling language modeling with pathways[J]. arXiv preprint arXiv:2204.02311, 2022.
- [18] Thoppilan R, De Freitas D, Hall J, et al. Lamda: Language models for dialog applications[J]. arXiv preprint arXiv:2201.08239, 2022.
- [19] Sanh V, Webson A, Raffel C, et al. Multitask prompted training enables zero-shot task generalization [J]. arXiv preprint arXiv:2110.08207, 2021.
- [20] Fu H, Yao; Peng, Khot T. How does gpt obtain its ability? tracing emergent abilities of language models to their sources[J/OL]. Yao Fu' s Notion, 2022. <https://yaofu.notion.site/How-does-GPT-Obtain-its-Ability-Tracing-Emergent-Abilities-of-Language-Models-to-their-Sources-b9a57>
- [21] Zhang S, Roller S, Goyal N, et al. Opt: Open pre-trained transformer language models[J]. arXiv preprint arXiv:2205.01068, 2022.
- [22] Raffel C, Shazeer N, Roberts A, et al. Exploring the limits of transfer learning with a unified text-to-text transformer[J]. The Journal of Machine Learning Research, 2020, 21(1):5485-5551.

- [23] Liu Y, Ott M, Goyal N, et al. Roberta: A robustly optimized bert pretraining approach[J]. arXiv preprint arXiv:1907.11692, 2019.
- [24] Gao L, Biderman S, Black S, et al. The pile: An 800gb dataset of diverse text for language modeling [J]. arXiv preprint arXiv:2101.00027, 2020.
- [25] Baumgartner J, Zannettou S, Keegan B, et al. The pushshift reddit dataset[C]//Proceedings of the international AAAI conference on web and social media: volume 14. 2020: 830-839.
- [26] Artetxe M, Bhosale S, Goyal N, et al. Efficient large scale language modeling with mixtures of experts[J]. arXiv preprint arXiv:2112.10684, 2021.
- [27] Shoenberger M, Patwary M, Puri R, et al. Megatron-lm: Training multi-billion parameter language models using model parallelism[J]. arXiv preprint arXiv:1909.08053, 2019.
- [28] Scao T L, Fan A, Akiki C, et al. Bloom: A 176b-parameter open-access multilingual language model [J]. arXiv preprint arXiv:2211.05100, 2022.
- [29] Smith S, Patwary M, Norick B, et al. Using deepspeed and megatron to train megatron-turing nl-g 530b, a large-scale generative language model[J]. arXiv preprint arXiv:2201.11990, 2022.
- [30] Rasley J, Rajbhandari S, Ruwase O, et al. Deepspeed: System optimizations enable training deep learning models with over 100 billion parameters[C]//Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining. 2020: 3505-3506.
- [31] Rajbhandari S, Rasley J, Ruwase O, et al. Zero: Memory optimizations toward training trillion parameter models[C]//SC20: International Conference for High Performance Computing, Networking, Storage and Analysis. IEEE, 2020: 1-16.
- [32] Chung H W, Hou L, Longpre S, et al. Scaling instruction-finetuned language models[J]. arXiv preprint arXiv:2210.11416, 2022.
- [33] Wei J, Bosma M, Zhao V, et al. Finetuned language models are zero-shot learners[C]//International Conference on Learning Representations. 2022.
- [34] Wang Y, Mishra S, Alipoormolabashi P, et al. Benchmarking generalization via in-context instructions on 1,600+ language tasks[J]. arXiv preprint arXiv:2204.07705, 2022.
- [35] Ouyang L, Wu J, Jiang X, et al. Training language models to follow instructions with human feedback [J]. arXiv preprint arXiv:2203.02155, 2022.



- [36] Ziegler D M, Stiennon N, Wu J, et al. Fine-tuning language models from human preferences[J]. arXiv preprint arXiv:1909.08593, 2019.
- [37] Stiennon N, Ouyang L, Wu J, et al. Learning to summarize with human feedback[J]. Advances in Neural Information Processing Systems, 2020, 33:3008-3021.
- [38] Liu P, Yuan W, Fu J, et al. Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing[J]. arXiv preprint arXiv:2107.13586, 2021.
- [39] Schick T, Schütze H. Exploiting cloze-questions for few-shot text classification and natural language inference[C]//Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume. 2021: 255-269.
- [40] Lin J, Nogueira R, Yates A. Pretrained transformers for text ranking: Bert and beyond[J]. Synthesis Lectures on Human Language Technologies, 2021, 14(4):1-325.
- [41] Bragg J, Cohan A, Lo K, et al. Flex: Unifying evaluation for few-shot nlp[J]. Advances in Neural Information Processing Systems, 2021, 34:15787-15800.
- [42] Cui L, Wu Y, Liu J, et al. Template-based named entity recognition using bart[C]//Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021. 2021: 1835-1845.
- [43] Ma R, Zhou X, Gui T, et al. Template-free prompt tuning for few-shot NER[C/OL]//Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies. Seattle, United States: Association for Computational Linguistics, 2022: 5721-5732. <https://aclanthology.org/2022.naacl-main.420>. DOI: 10.18653/v1/2022.naacl-main.420.
- [44] Ram O, Kirstain Y, Berant J, et al. Few-shot question answering by pretraining span selection[C]//Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers). 2021: 3066-3079.
- [45] Zhong W, Gao Y, Ding N, et al. Proqa: Structural prompt-based pre-training for unified question answering[J]. arXiv preprint arXiv:2205.04040, 2022.
- [46] Clark K, Luong M T, Le Q V, et al. Electra: Pre-training text encoders as discriminators rather than generators[J]. arXiv preprint arXiv:2003.10555, 2020.

- [47] Yao X, Zheng Y, Yang X, et al. Nlp from scratch without large-scale pretraining: A simple and efficient framework[C]//International Conference on Machine Learning. PMLR, 2022: 25438-25451.
- [48] Zhang X, Jiang Y, Wang X, et al. Domain-specific ner via retrieving correlated samples[C]//Proceedings of the 29th International Conference on Computational Linguistics. 2022: 2398-2404.
- [49] Hu D, Hou X, Du X, et al. Varmae: Pre-training of variational masked autoencoder for domain-adaptive language understanding[J]. arXiv preprint arXiv:2211.00430, 2022.
- [50] Yin Y, Chen C, Shang L, et al. Autotinybert: Automatic hyper-parameter optimization for efficient pre-trained language models[C]//Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers). 2021: 5146-5157.
- [51] Ding N, Qin Y, Yang G, et al. Delta tuning: A comprehensive study of parameter efficient methods for pre-trained language models[J]. arXiv preprint arXiv:2203.06904, 2022.
- [52] Zhou X, Ma R, Zou Y, et al. Making parameter-efficient tuning more efficient: A unified framework for classification tasks[C]//Proceedings of the 29th International Conference on Computational Linguistics. 2022: 7053-7064.
- [53] Shi H, Zhang R, Wang J, et al. Layerconnect: Hypernetwork-assisted inter-layer connector to enhance parameter efficiency[C]//Proceedings of the 29th International Conference on Computational Linguistics. 2022: 3120-3126.

## 索引

$n$  元文法, 3  
 $n$  元语法, 3  
 $n$  元语法单元, 3

Additive Smoothing, 4

Contextualized Word Embedding, 11

Cross-entropy, 30

Dynamic Word Embedding, 11

Language Model, LM, 1

Perplexity, 30

上下文相关的词向量, 11

交叉熵, 30

加一平滑, 4  
加法平滑, 4  
动态词向量, 11  
单向语言模型, 15  
古德-图灵估计法, 4  
困惑度, 30  
平滑, 4

语言模型, 1