自然语言处理导论

张奇 桂韬 黄萱菁

2022 年 12 月 12 日

.

数学符号

数与数组

α	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

α_i	向量 $\boldsymbol{\alpha}$ 中索引 i 处的元素
α_{-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
$\boldsymbol{A}_{ijk}$	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
$\mathbb{R}$	实数集合
$0, 1$	含 0 和 1 的二值集合
$0, 1, ..., n$	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

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

微积分

$\frac{\mathrm{d}y}{\mathrm{d}x}$	y 对 x 的导数
$\frac{\partial y}{\partial x}$	y 对 x 的偏导数
$\bigtriangledown_{\boldsymbol{x}} y$	y 对向量 $\boldsymbol{x}$ 的梯度
$\bigtriangledown_{\boldsymbol{X}} y$	y 对矩阵 $\boldsymbol{X}$ 的梯度
$\bigtriangledown_{\mathbf{X}} y$	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
Var$(f(x))$	$f(x)$ 在分布 $P(x)$ 下的方差
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

$\mathbb{X}$	数据集
$\boldsymbol{x}^{(i)}$	数据集中第 i 个样本（输入）
$\boldsymbol{y}^{(i)}$ 或 $y^{(i)}$	第 i 个样本 $\boldsymbol{x}^{(i)}$ 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
$f(\boldsymbol{x}; \boldsymbol{\theta})$	由参数 $\boldsymbol{\theta}$ 定义的关于 $\boldsymbol{x}$ 的函数（也可以直接写作 $f(\boldsymbol{x})$，省略 $\boldsymbol{\theta}$）
$\log x$	x 的自然对数函数
$\sigma(x)$	Sigmoid 函数 $\dfrac{1}{1+exp(-x)}$
$\|\|\boldsymbol{x}\|\|_p$	$\boldsymbol{x}$ 的 L^p 范数
$\|\|\boldsymbol{x}\|\|$	$\boldsymbol{x}$ 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 $x = x_1, x_2, ..., x_n$ 中第 i 个单词 x_i 的词向量 $\boldsymbol{v}_{x_i}$
- 损失函数 $\mathcal{L}$ 为负对数似然函数：$\mathcal{L}(\boldsymbol{\theta}) = -\sum_{(x,y)} \log P(y|x_1...x_n)$
- 算法的空间复杂度为 $\mathcal{O}(mn)$

目 录

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。语言模型作为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）是建模词序列 $S = \{w_1 w_2 ... w_m\}$ 的概率分布 $P(w_1 w_2 ... w_m)$，即计算给定的词序列 $w_1 w_2 ... w_m$ 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 $P(w_1 w_2 ... w_m)$ 表示，对于任意词串 $w_1 w_2 ... w_m \in V^+$，则有 $P(w_1 w_2 ... w_m) \geqslant 0$，并且对于所有词串，函数 $P(w_1 w_2 ... w_m)$ 满足归一化条件 $\sum_{w_1 w_2 ... w_m \in \mathbb{V}^+} P(w_1 w_2 ... w_m) = 1$。$P(w_1 w_2 ... w_m)$ 是定义在 $\mathbb{V}^+$ 上的概率分布。

由于联合概率 $P(w_1 w_2 ... w_m)$ 的参数量十分巨大，直接计算 $P(w_1 w_2 ... w_m)$ 非常困难。如果把 $w_1 w_2 ... w_m$ 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 7.9792×10^{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 $P(w_1 w_2 ... w_m)$ 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：

$$\begin{aligned} P(S) &= P(w_1, w_2, ..., w_m) \\ &= P(w_1)P(w_2|w_1)P(w_3|w_1w_2)\cdots P(w_m|w_1w_2...w_{n-1}) \\ &= \prod_{i=1}^{n} P(w_i|w_1w_2\cdots w_{i-1}) \end{aligned} \tag{6.1}$$

由此，$w_1w_2...w_m$ 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 $P(w_1w_2...w_m)$ 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：

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

但是，仅通过上述过程模型的参数量依然没有下降，$P(w_m|w_1w_2...w_{m-1})$ 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（n-gram Language Model）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 $P(w_i|w_{i-n+1}...w_{i-1})$ 进行估计的语言模型[1]。此后，循环神经网络[2]、卷积神经网络[3]、端到端记忆网络[4] 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。互联网的飞速发展，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[5]，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[?]，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”。本章中也将介绍采用单向、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 $w_1, w_2, ..., w_m$ 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:

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

其中,词 w_i 出现的概率受它前面的 $i-1$ 个词 $w_1, w_2, ..., w_{i-1}$ 影响,我们将这 $i-1$ 个词 $w_1, w_2, ..., w_{i-1}$ 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 $|\mathbb{V}|^{i-1}$ 种，其中 V 表示单词词表，$|\mathbb{V}|$ 表示词表的大小。最简单的根据语料库对 $P(w_i|w_1, w_2, ..., w_{i-1})$ 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。

$$P(w_i|w_1, w_2, ..., w_{i-1}) = \frac{\mathrm{C}(w_1, w_2, ..., w_{i-1}w_i)}{\mathrm{C}(w_1, w_2, ..., w_{i-1})} \tag{6.4}$$

其中，C(·) 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 $P(w_i|w_1, w_2, ..., w_{i-1})$ 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:

$$P(w_i|w_{1:i-1}) = P(w_i|w_{i-(n-1):i-1}) \tag{6.5}$$

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:

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

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：

$$\begin{aligned} P(\text{把 努力 变成 一种 习惯}) =& P(\text{把}) \times P(\text{努力}|\text{把}) \times P(\text{变成}|\text{努力}) \times \\ & P(\text{一种}|\text{变成}) \times P(\text{习惯}|\text{一种}) \end{aligned} \tag{6.7}$$

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 δ 次。以二元语法模型为例，其平滑后的条件概率为：

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

其中 $0 \leqslant \delta \leqslant 1$，$V$ 是训练语料中所有单词的集合，$C(w_{i-1}w_i)$ 代表词 w_{i-1} 和词 w_i 同时出现的频率。将公式6.8其拓展到 n 元语言模型上：

$$P(w_i|w_{i-n+1}^{i-1}) = \frac{\delta + C(w_{i-n+1}^{i})}{\delta|\mathbb{V}| + \sum\limits_{w_i} C(w_{i-n+1}^{i})} \tag{6.9}$$

当 $\delta = 1$ 时，该方法又称为加一平滑。此外，针对所有不在词表 $\mathbb{V}$ 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。尽管加一平滑足够简单，但部分学者认为，这种方法的表现通常并不能满足实际需求。

6.2.2 古德-图灵估计法

古德-图灵评估法（Good-Turing Estimate）[6] 是由古德 (I.J.Good) 基于图灵 (Turning) 的方法提出的，其思想被应用于多种平滑方法中。该方法基于一种理念：已知事件的概率能用于估计未见事

件的概率。具体来说，对于任意一个出现了 r 次的 n 元语法，都假设其出现了 r^* 次，

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

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:

$$p_r = \frac{r^*}{N} \tag{6.11}$$

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:

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

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。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[7]，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。对于一个统计数量为 r 的二元语法 w_{i-1}^i，用下列公式对其统计数量进行修正:

$$P(w_{i-1}^i) = \begin{cases} d_r \frac{c(w_{i-1}^i)}{c(w_{i-1})}, & r > 0 \\ a(w_{i-1}) P_{ML}(w_i), & r = 0 \end{cases} \tag{6.13}$$

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 $\sum_{w_i} c_{katz}(w_{i-1}^i) = \sum_{w_i} c(w_{i-1}^i)$。$a(w_{i-1})$ 的值通常如下公式估计:

$$a(w_{i-1}) = \frac{1 - \sum\limits_{w_i:c(w_{i-1}^i)>0} P_{katz}(w_i|w_{i-1})}{\sum\limits_{w_i:c(w_{i-1}^i)=0} P_{ML}(w_i)} = \frac{1 - \sum\limits_{w_i:c(w_{i-1}^i)>0} P_{katz}(w_i|w_{i-1})}{1 - \sum\limits_{w_i:c(w_{i-1}^i)>0} P_{ML}(w_i)} \tag{6.14}$$

要根据修正的计数计算概率 $P_{katz}(w_i|w_{i-1})$，只需要归一化:

$$P_{katz}(w_i|w_{i-1}) = \frac{c_{katz}(w_{i-1}^i)}{\sum\limits_{w_i} c_{katz}(w_{i-1}^i)} \tag{6.15}$$

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 $\mu, r \in \{1, 2, ..., k\}$，有公式:

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

(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:

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

上述公式6.16和公式6.17的唯一解为:

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

下面将 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:

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

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：

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

满足以下约束：

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

6.3 神经网络语言模型

基于稀疏表示的 n 元语言模型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。随着深度神经网络的发展，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①。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 $w_1, w_2, ..., w_{i-1}$，神经网络语言模型的任务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[1] 沿用了马尔可夫假设，认为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 $w_{(i-n+1)}...w_{i-1}$，输出词 w_i 在词表 $\mathcal{V}$ 上的概率分布，即估计条件概率 $P(w_i|w_{(i-n+1)}:w_{i-1})$。

前馈神经网络由三部分组成，如图6.1所示，分别为输入层、隐含层和输出层。在词向量层，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；隐含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，隐含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。

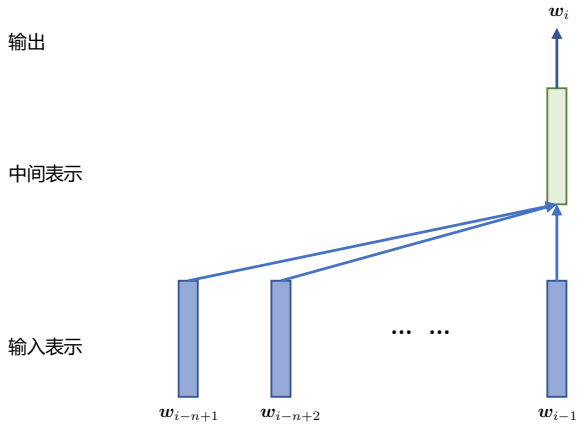

图 6.1 基于前馈神经网络的语言模型结构图

① 详见本书第 4 章分布式表示

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实值向量。在具体实现中，模型首先根据每个词在词表 $\mathcal{V}$ 中的位置，将历史词序列 $w_{(i-n+1)},...,w_{i-1}$ 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：

$$\boldsymbol{v} = [\boldsymbol{v}_{(i-n+1)},...,\boldsymbol{v}_{i-1}] \tag{6.22}$$

其中 $\boldsymbol{v}_{i-1} \in \mathbb{R}^d$ 代表词 w_{i-1} 所对应的词向量，d 代表词向量的维度，$\boldsymbol{x} \in \mathbb{R}^{(n-1)d}$ 代表将所有历史词向量拼接后的结果。

隐含层的目标是对词向量 $\boldsymbol{v}$ 进行线性变换与非线性映射。隐含层的计算可以用如下公式表示:

$$\boldsymbol{h} = f(\boldsymbol{W}^{hid}\boldsymbol{v} + \boldsymbol{b}^{hid}) \tag{6.23}$$

其中，隐含层由线性变换矩阵 $\boldsymbol{W}^{hid} \in \mathbb{R}^m \times (n-1)d$、偏置项 $\boldsymbol{b}^{hid} \in \mathbb{R}^m$ 组成，m 为隐含层维度，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含层向量 $\boldsymbol{h}$ 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：

$$\boldsymbol{y} = \text{Softmax}(\boldsymbol{W}^{out}\boldsymbol{h} + \boldsymbol{b}^{out}) \tag{6.24}$$

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

上述前馈神经网络语言模型的总参数量为 $|\mathcal{V}| \times d + m \times (n-1)d + m + |\mathcal{V}| \times m + |\mathcal{V}|$，即 $|\mathcal{V}|(d+m+1) + m((n-1)d+1)$。词向量维度 d，隐藏层维度 m 和历史词长度 $n-1$ 可以根据实际需求灵活调整。可以看出，词表大小 $|\mathcal{V}|$ 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。然而，前馈神经网络的历史词长度是固定的，而不同的句子对历史词长度的依赖往往是动态的。

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。对于一些句子只需要较短的历史词就能进行判断，前馈语言模型也足以进行准确的预测。然而，对于信息较为复杂的长文本，模型需要依赖较长的历史采用做出准确预测。

例如：与小明一起旅行游玩总是充满了惊喜，你永远不知道会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。

循环神经网络（Recurrent Neural Network, RNN）[2] 常用于处理序列结构的数据，其特点是上一时刻的模型隐含层状态会作为当前时刻模型的输入，每一时刻的隐含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含层状态来获取，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、隐含层和输出层，其结构如图6.2所示。

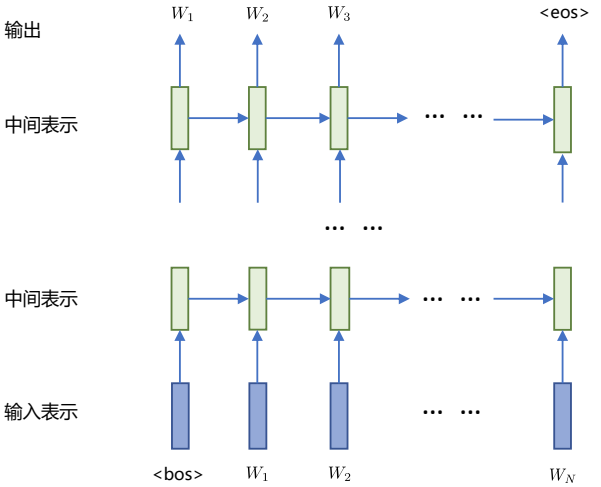

图 6.2 循环神经网络的结构

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 $w_1, ..., w_{i-1}$, 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, 一是前一个词 $\boldsymbol{w}_{i-1}$ 的词向量，二是包含所有历史词信息的 $i-1$ 时刻隐含层输出 $\boldsymbol{h}_{i-1}$

$$\boldsymbol{x}_i = [\boldsymbol{v}_{i-1}; \boldsymbol{h}_{i-1}] \tag{6.25}$$

其中 $\boldsymbol{x}_i \in \mathbb{R}^{d+m}$, $\boldsymbol{v}_{i-1} \in \mathbb{R}^d$ 代表词 w_{i-1} 所对应的词向量, d 为词向量维度 $\boldsymbol{h}_{i-1} \in \mathbb{R}^m$ 代表前一时刻模型的隐含层输出，m 为隐含层维度。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 作为初始隐含层向量。

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

$$\boldsymbol{h}_i = f(\boldsymbol{W}^{hid}\boldsymbol{x}_i + \boldsymbol{b}^{hid}) \tag{6.26}$$

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

其拆分开更能体现循环神经网络递归的特点：

$$\boldsymbol{h}_i = f(\boldsymbol{U}\boldsymbol{v}_{i-1} + \boldsymbol{V}\boldsymbol{h}_{i-1} + \boldsymbol{b}^{hid}) \tag{6.27}$$

循环神经网络语言模型输出层的目标是基于隐含层状态 $\boldsymbol{h}_i$ 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：

$$\boldsymbol{y}_i = \text{Softmax}(\boldsymbol{W}^{out}\boldsymbol{h}_i + \boldsymbol{b}^{out}) \tag{6.28}$$

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

本节只介绍了最基本的循环神经网络，隐含层的结构较为简单。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弥散或梯度爆炸，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[8] 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

在海量的无标注语料上进行自监督训练的预训练语言模型，在众多自然语言处理任务上都取得了非常好的效果。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。本节将介绍三种不同的预训练方法的模型结构以及所采用的预训练任务：单向预训练语言模型、双向预训练语言模型和序列到序列预训练语言模型。在此基础上，还将介绍如何在常见自然语言处理任务上运用预训练语言模型。

6.4.1 单向预训练语言模型

OpenAI 公司在 2018 年提出的 GPT-2 [9] 模型是经典的单向预训练语言模型之一。GPT（Generative Pre-Training）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。本节将介绍 GPT-2 的模型结构和单向语言模型的建模过程。

1. 模型结构

GPT-2 模型结构如图6.3所示，由多层 Transformer 的解码器[①]构成，解码器的结构保证了输入文本每个位置只能得到过去时刻的信息。模型结构可以分为输入层，解码层和输出层三部分。

① Transformer 解码器的具体结构请参考第 8 章机器翻译。

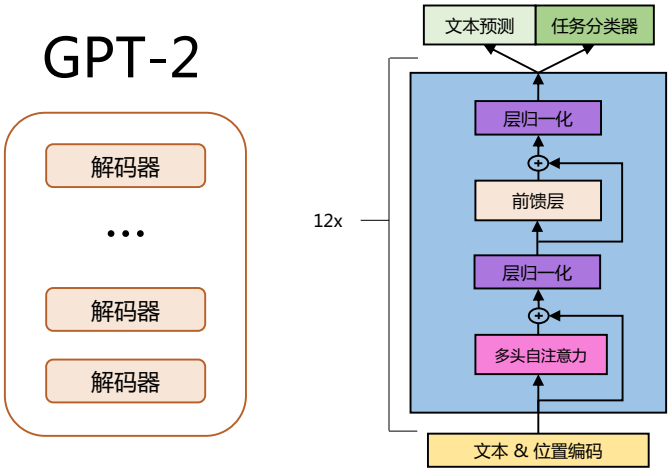

图 6.3 GPT-2 预训练语言模型结构

给定文本序列 $w_0, ..., w_n$，GPT-2 首先在输入层中将其映射为稠密的向量：

$$\boldsymbol{v}_i = \boldsymbol{v}_i^t + \boldsymbol{v}_i^p \tag{6.29}$$

其中，$\boldsymbol{v}_i$ 是词 w_i 的词向量，$\boldsymbol{v}_i^p$ 是词 w_i 的位置向量，$\boldsymbol{v}_i$ 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 $\boldsymbol{v} = \boldsymbol{v}_0 ... \boldsymbol{v}_n$，随后将 $\boldsymbol{v}$ 送入模型解码层。解码层由 L 个 Transformer 的解码器组成，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：

$$\boldsymbol{h}^{[L]} = \text{Transformer-Decoder}^{[L]}(\boldsymbol{h}_{\boldsymbol{i}}^{[\mathbf{0}]}) \tag{6.30}$$

其中 $\boldsymbol{h}^{[L]} \in \mathbb{R}^{d \times n}$ 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 $\boldsymbol{h}^{[L]}$，预测每个位置上的条件概率，其计算过程可以表示为：

$$P(w_i|w_0, ..., w_{i-1}) = \text{softmax}(\boldsymbol{W}^{out} h_i + \boldsymbol{b}^{out}) \tag{6.31}$$

其中，$\boldsymbol{W}^{out} \in \mathbb{R}^{|\mathbb{V}| \times d}$，$|\mathbb{V}|$ 为词表大小。

2. 预训练任务

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

$$\mathcal{L}(w) = -\sum_{i-1}^{n} \log P(w_i|w_0...w_{i-1};\theta) \tag{6.32}$$

其中 θ 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。GPT-2 的训练和预测是一个自回归的过程，因此更适合文本生成任务，虽然进行语言模型建模时学习到的通用语义特征让 GPT-2 在自然语言理解任务上也有不错的表现。

6.4.2 双向预训练语言模型

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。Matthew Peters 等人提出了经典的双向预训练语言模型 ELMo[5]，在当时为多种自然语言处理任务带来了显著的性能提升。本节将介绍 ELMo 的结构和双向语言模型的建模过程。

1. 模型结构

ELMo 的结构如图6.4所示，主要包含输入层，编码层和输出层三个部分。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 $c_{i_0}, c_{i_1}..., c_{i_m}$ 组成，每个字符 c_{i_j} 通过字符嵌入层转化为向量 $\boldsymbol{v}_{w_{i_j}}$：

$$\boldsymbol{v}_{w_{i_j}} = \boldsymbol{W}^{char}\boldsymbol{e}_{i_j} \tag{6.33}$$

其中，$\boldsymbol{W}^{char} \in \mathbb{R}^{d^{char} \times |\mathbb{V}^{char}|}$ 为字符嵌入矩阵、$\mathbb{V}^{char}$ 表示字符库、d^{char} 表示字符向量维度。得到词 w_i 的字符向量表示 $\boldsymbol{v}_{c_{i_0}}, \boldsymbol{v}_{c_{i_1}}..., \boldsymbol{v}_{c_{i_m}}$ 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 $\hat{\boldsymbol{v}}_i$。在得到卷积神经网络的输出 $\boldsymbol{f}_i$ 后，为了避免梯度爆炸或弥散，模型使用 Highway 网络对 $\hat{\boldsymbol{v}}_i$ 进一步转换：

$$\boldsymbol{v}_i = \boldsymbol{g} \cdot \boldsymbol{f}_i + (\boldsymbol{1} - \boldsymbol{g}) \cdot ReLU(\boldsymbol{W}\boldsymbol{f}_i + b) \tag{6.34}$$

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

$$\boldsymbol{g} = \sigma(\boldsymbol{W}^g\boldsymbol{f}_i + \boldsymbol{b}^g) \tag{6.35}$$

其中，$\boldsymbol{W}^g$ 为线性转换矩阵；$\boldsymbol{b}^g$ 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。

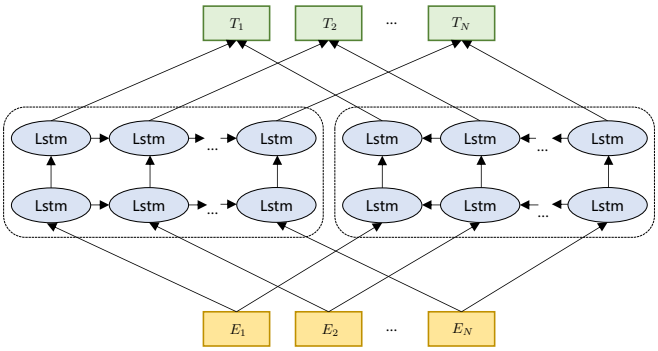

图 6.4 双向预训练语言模型 ELMo 神经网络结构[5]

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 EMLo 模型会产生 $2L+1$ 向量表示：

$$R_i = \{\boldsymbol{v}_i, \overrightarrow{\boldsymbol{h}}_{i,j}, \overleftarrow{\boldsymbol{h}}_{i,j} | j = 0, 1, ..., L\} \tag{6.36}$$

其中，$\boldsymbol{v}_i$ 代表输入层得到的上下文无关词向量，$\overrightarrow{\boldsymbol{h}}_{i,j}$ 代表第 j 层前向 LSTM 编码得到的特征，$\overleftarrow{\boldsymbol{h}}_{i,j}$ 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 $\boldsymbol{h}_{i,j}^{LM} = [\overleftarrow{\boldsymbol{h}}_{i,j}; \overrightarrow{\boldsymbol{h}}_{i,j}]$。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 $\boldsymbol{h}_{i,j}^{LM}$。因为每层 LSTM 学习到的东西不相同，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 EMLo 向量，其计算过程可以表示为：

$$\text{EMLo}_i^{\text{task}} = \gamma^{\text{task}} \sum_{j=0}^{L} s_j^{\text{task}} \boldsymbol{h}_{i,j}^{LM} \tag{6.37}$$

其中 γ 是整体的缩放系数，s^{task} 是每层的系数。在执行下游任务时，一般将 $\boldsymbol{v}_i$ 和 $\text{EMLo}_i^{\text{task}}$ 拼接起来作为词 w_i 的最终表示向量进行分类。

2. 预训练任务

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 $\overrightarrow{\boldsymbol{h}}_{i,L}$ 和 $\overleftarrow{\boldsymbol{h}}_{i,L}$ 预测下一时刻的词，其建模过程可以表示

$$
\begin{aligned}
P_{forward} &= \prod_{i=1}^{n} P(\boldsymbol{w}_i | w_{1:i-1}; \Theta) \\
P_{backward} &= \prod_{i=1}^{n} P(\boldsymbol{w}_i | w_{i+1:n}; \Theta)
\end{aligned}
\tag{6.38}
$$

其中，Θ 和 Θ 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。

6.4.3 掩码预训练语言模型

2018 年，Devlin 等人提出了掩码预训练语言模型 BERT[10]。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。本节将介绍 BERT 的结构和所采用的预训练任务。

BERT 的结构如图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。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：掩码语言建模（Masked Language Modeling，MLM）和下一句预测（Next Sentence Prediction，NSP）。

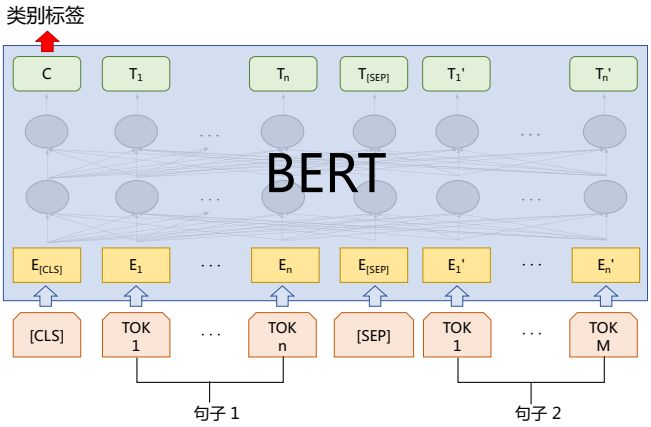

图 6.5　掩码预训练语言模型 BERT 神经网络结构[10]

1. 模型结构

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，而 BERT 则与其他预训练模型相比有较大区别。BERT 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 $\boldsymbol{v}$ 可以表示为:

$$\boldsymbol{v} = \boldsymbol{v}^t + \boldsymbol{v}^s + \boldsymbol{v}^p$$

其中，$\boldsymbol{v}^t$ 代表词嵌入；$\boldsymbol{v}^s$ 代表段嵌入；$\boldsymbol{v}^p$ 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 $\boldsymbol{e}^t \in \mathbb{R}^{|\mathbb{V}|}$，其对应的词嵌入 $\boldsymbol{v}_t$ 为：

$$\boldsymbol{v}^t = \boldsymbol{e}^t \boldsymbol{W}^t$$

其中，$\boldsymbol{W}^t \in \mathbb{R}^{|\mathbb{V}| \times e}$ 表示词嵌入矩阵；$|\mathbb{V}|$ 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 $\boldsymbol{W}^s$ 将独热段编码 $\boldsymbol{e}^s$ 转化为段嵌入 $\boldsymbol{v}^s$:

$$\boldsymbol{v}^s = \boldsymbol{e}^s \boldsymbol{W}^s$$

其中，$\boldsymbol{W}^s \in \mathbb{R}^{|\mathbb{S}| \times e}$ 表示段嵌入矩阵；$|\mathbb{S}|$ 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 $\boldsymbol{e}^p$，通过可训练的位置嵌入矩阵 $\boldsymbol{W}^p$ 即可得到位置向量 $\boldsymbol{v}^p$：

$$\boldsymbol{v}^p = \boldsymbol{e}^p \boldsymbol{W}^p$$

其中，$\boldsymbol{W}^t \in \mathbb{R}^{N \times e}$ 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。

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，单向的信息是不够的，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。

模型的预训练目标是将被掩盖位置的词还原：

$$P(x) = \text{Softmax}(\boldsymbol{h}^{[L]}\boldsymbol{W}^{t}) \tag{6.39}$$

下一句预测：通过掩码语言建模，BERT 能够有效挖掘同一段输入文本的上下文语义。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。

故名思义，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。为了完成该预训练任务，需要在输入文本的开头添加 [CLS] 字符以进行二分类，句子对之间用 [SEP] 进行分隔，最终输入文本的形式为 [CLS] A [SEP] B [SEP]。在经过模型编码后，取 [CLS] 位置的表示 $\boldsymbol{h}_{[CLS]}$，使用全连接层进行二分类。

$$P = \text{Softmax}(\boldsymbol{h}_{[CLS]}\boldsymbol{W}^{p} + \boldsymbol{b}^{o}) \tag{6.40}$$

其中，$\boldsymbol{W}^{p} \in \mathbb{R}^{d\times 2}$ 为全连接层权重；$\boldsymbol{b}^{o}$ 表示全连接层偏置。

6.4.4 序列到序列预训练语言模型

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。在本节中，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[11]。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。

1. 模型结构

BART 模型使用基于 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，因为其主要沿用了 Transformer 结构[①]，在本章中，我们不对其结构做详细介绍，而是重点介绍其预训练任务。

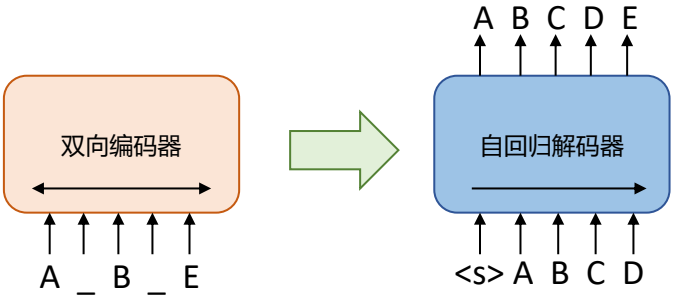

图 6.6 BART 的结构

2. 预训练任务

BART 的预训练过程是一个对含有噪声的输入文本进行去噪的过程，与 BERT 独立预测加噪位置的词不同的是，BART 通过自回归的方式对加噪词进行还原编码器对引入噪声的文本进行双向编码，再使用单向的自回归解码器对原始的文本进行还原。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：**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：**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：**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：**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：**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。

可以看出，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

① Transformer 的详细结构请参照本书第 8 章。

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。

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 $w_1, w_2, ..., w_n$，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：

$$W = [\text{CLS}], w_1, w_2, ..., w_n, [\text{SEP}] \tag{6.41}$$

$$\boldsymbol{V} = \text{InputLayer}(W) \tag{6.42}$$

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 $\boldsymbol{h}$。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 $\boldsymbol{h}_{[\boldsymbol{CLS}]}$ 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：

$$\boldsymbol{H} = \text{BERT}(\boldsymbol{V}) \tag{6.43}$$

$$P = \text{Softmax}(\boldsymbol{h}_{[CLS]}\boldsymbol{W} + \boldsymbol{b}) \tag{6.44}$$

其中，$\boldsymbol{W} \in \mathbb{R}^{d \times K}$ 和 $\boldsymbol{b} \in \mathbb{R}^K$ 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 $w_1^{(1)}, w_2^{(1)}, ..., w_n^{(1)}$ 和 $w_1^{(2)}, w_2^{(2)}, ..., w_m^{(1)}$，BERT 使用 [SEP] 作为分隔符，将两个句子拼接到一起，再输入层转换为输入表示。

$$W = [\text{CLS}], w_1^{(1)}, w_2^{(1)}, ..., w_n^{(1)}, [\text{SEP}], w_1^{(2)}, w_2^{(2)}, ..., w_m^{(1)}, [\text{SEP}] \tag{6.45}$$

$$\boldsymbol{V} = \text{InputLayer}(W) \tag{6.46}$$

其中，$\boldsymbol{V} \in \mathbb{R}^{(n+m)\times d}$，$d$ 为模型隐含层维度，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：

$$W = [\text{CLS}], w_1^{(1)}, w_2^{(1)}, ..., w_n^{(1)}, [\text{SEP}] \tag{6.47}$$

$$\boldsymbol{V} = \text{InputLayer}(W) \tag{6.48}$$

$$\boldsymbol{H} = \text{BERT}(\boldsymbol{V}) \tag{6.49}$$

其中，$\boldsymbol{V} \in \mathbb{R}^{n\times d}$ 为模型输入层的输出，$\boldsymbol{H} \in \mathbb{R}^{n\times d}$ 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。

在得到了隐含层表示后，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：

$$P = \text{Softmax}(\boldsymbol{h}_i \boldsymbol{W} + \boldsymbol{b}) \tag{6.50}$$

其中 $\boldsymbol{h}_i$ 是输入序列的第 i 个词，$i \in \{1, ..., n\}$。得到概率分布后，可以使用交叉熵学习模型参数。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 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：

$$P(s) = \prod_{i=1}^{n} P(w_i|w_{i-1}) \tag{6.51}$$

对于由句子 $(s_1, s_2..., s_n)$ 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：

$$P(T) = \prod_{i=1}^{n} P(s_i) \tag{6.52}$$

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：

$$H_p(s) = -\frac{1}{W_s} \log_2 P(s) \tag{6.53}$$

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：

$$PP_s(s) = 2^{H_p(s)} \tag{6.54}$$

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。

6.6 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器层，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）[12]，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[13–15]、命名实体识别[16, 17]、阅读理解[18, 19] 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 [20] 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合选择来大幅降低预训练模型训练时间[21]，根据领域特性进行数据选择[22]，利用领域之间关联[23]，自动优化超参数选择[24] 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[25–27]，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。

6.7 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] Good I J. The population frequencies of species and the estimation of population parameters[J]. Biometrika, 1953, 40(3-4):237-264.

[7] 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.

[8] Sundermeyer M, Schlüter R, Ney H. Lstm neural networks for language modeling[C]//Thirteenth annual conference of the international speech communication association. 2012.

[9] Radford A, Wu J, Child R, et al. Language models are unsupervised multitask learners[J]. OpenAI blog, 2019, 1(8):9.

[10] 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 Asso-

ciation for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers). 2019: 4171-4186.

[11] 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.

[12] 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.

[13] 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.

[14] 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.

[15] 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.

[16] 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.

[17] 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.

[18] 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.

[19] 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.

[20] 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.

[21] 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.

[22] 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.

[23] 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.

[24] 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.

[25] 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.

[26] 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.

[27] 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