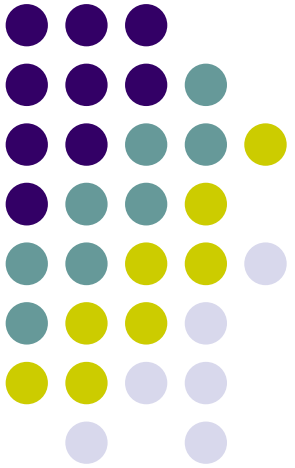
预训练模型

熟读唐诗三百首，不会做诗也会吟。——清·孙洙



哈尔滨工业大学计算学部

刘远超

预训练模型

本章内容简介

- **什么是预训练**
  - 预训练与迁移学习
  - 计算机视觉领域的预训练
  - 自然语言处理领域的预训练
- **语言模型与预训练**
- **上下文词嵌入模型**

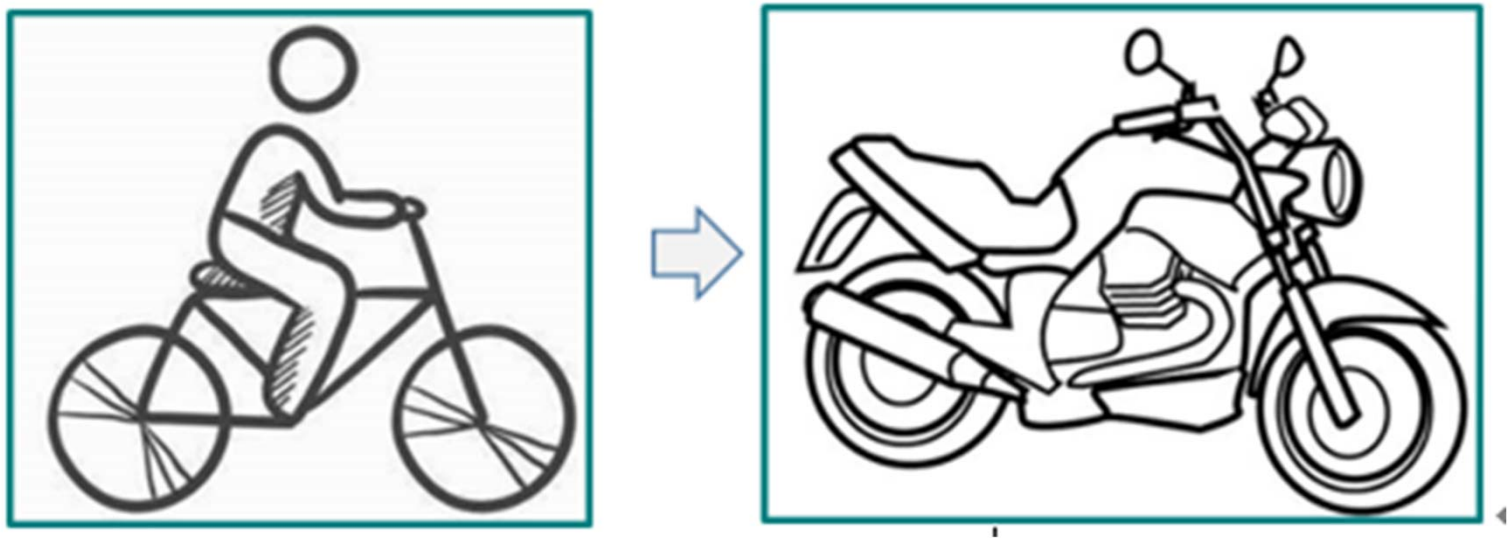
预训练模型

本章内容简介

- 什么是预训练
  - 预训练与迁移学习
  - 计算机视觉领域的预训练
  - 自然语言处理领域的预训练
- 语言模型与预训练
- 上下文词嵌入模型

预训练与迁移学习

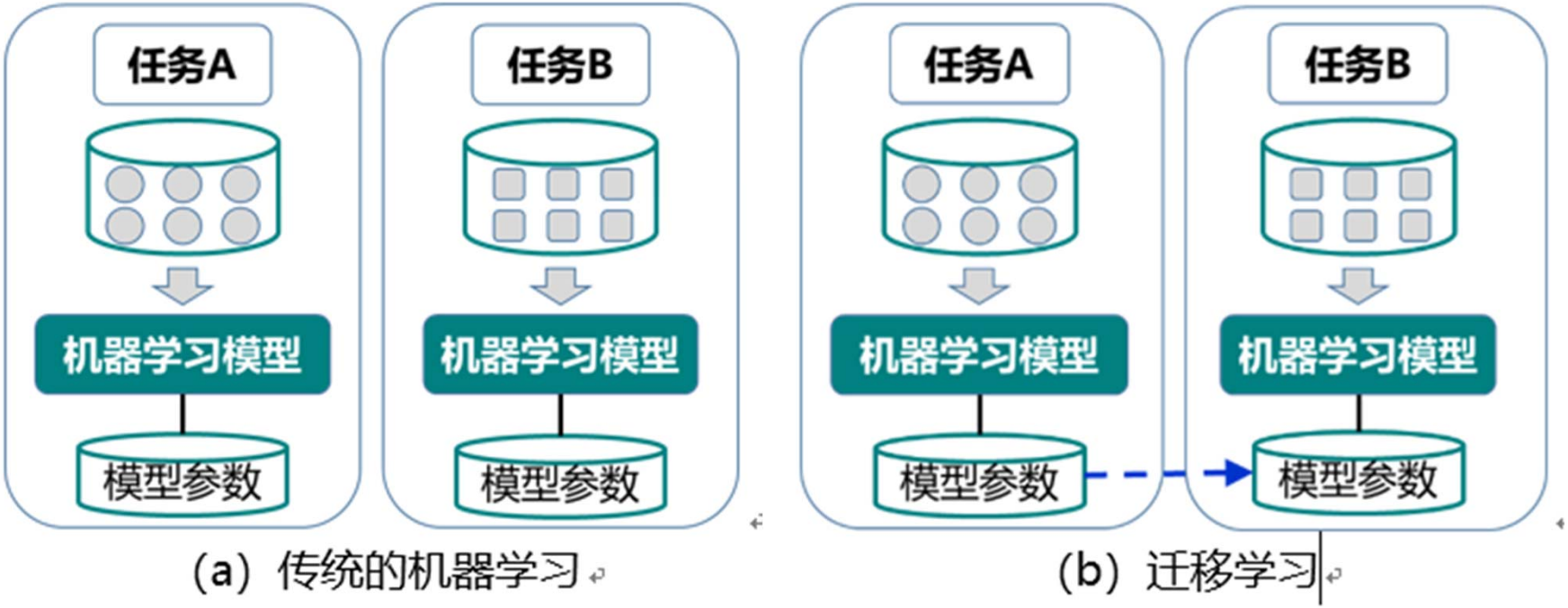
迁移学习的直观认识



学习骑普通自行车的过程中得到的知识可以很容易地迁移到新的问题，例如如何学习骑摩托车的问题上

预训练与迁移学习

什么是迁移学习?



(a) 传统的机器学习　　(b) 迁移学习

- **迁移学习是指**将某个研究任务上学习到的知识或模式应用到其它不同但密切相关的任务中。通俗地讲，迁移学习可以使我们得以利用前人在相近任务或者问题上训练得到的模型（及其参数取值），在其基础上再次训练对参数取值小幅度改动形成新模型，进而应用到自己所关注的任务或问题（也常称为下游任务或者问题）中。

预训练与迁移学习

什么是预训练?

- 迁移学习的研究主要兴起于 1995 年。随着深度学习的崛起，在计算机视觉和自然语言处理等领域中的迁移学习往往通过**预训练**实现。**预训练模型**是在大型基准数据集上训练的模型，用于解决相似的任务或者下游相关任务。
- 因此，对标迁移学习的基本思想，基于预训练的学习也有两大阶段：

A. **预训练阶段**。在面向通用任务或者与目标任务相关的**大型基准数据集**上利用某种神经网络训练，得到预训练的模型，此时模型参数的取值得到了很好的初步优化；

B. **微调阶段**：微调阶段的参数通常包括两大部分：一部分是**延续使用**预训练阶段的模型参数（及其初步优化取值）、另外一部分是微调阶段**可能额外增加的模型参数**（取值通常进行随机初始化）。这些参数取值都将在微调阶段在目标任务模型上利用**新数据集（通常规模要小）**做进一步的定制训练和微调。这里的“微调”主要是强调对预训练阶段得到的模型参数的**小幅度调整**。

预训练与迁移学习

预训练的优点

“预训练+微调”的这种机制，有时也可以简称预训练模型，其有如下优势：

1) **有助于提升目标任务或下游任务的预测水平**。这是因为预训练阶段的任务与目标任务有很好的关联性，借助预训练的大规模数据集可以取得目标任务模型参数很好的初始值，从而有助于指导微调阶段的训练；

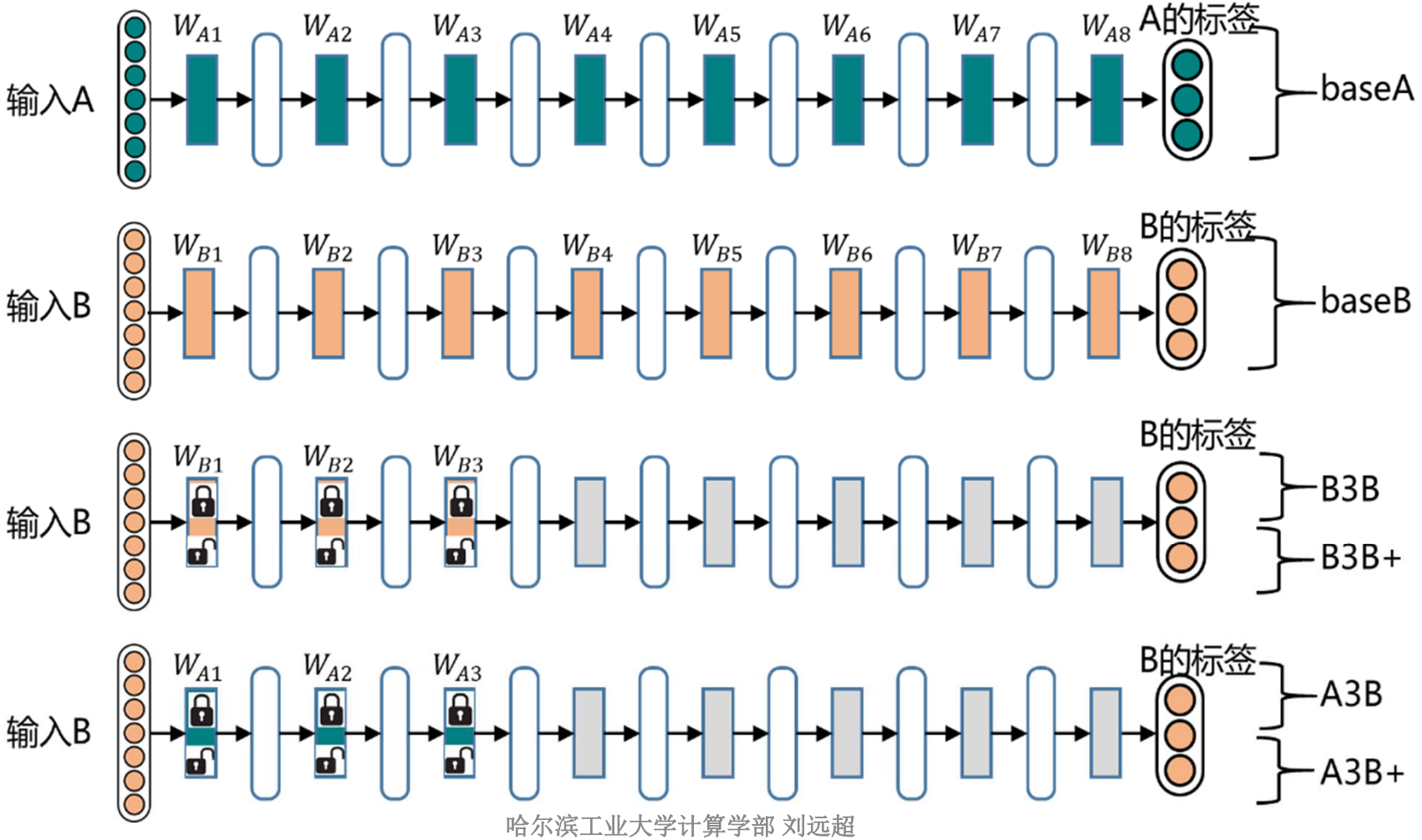
2) **降低目标任务的训练计算成本**。有了预训练阶段得到的初始模型参数，微调阶段的计算开销就可以大幅降低，取得更快的收敛速度。尽管预训练阶段由于数据集巨大，往往需要消耗较长的训练时间，但一旦训练完成，得到的模型可以被反复使用。例如，目前很多训练好的模型如BERT、GPT等可以在情感分析、自动问答等很多其它自然语言处理任务中直接使用。

本章内容简介

- **什么是预训练**
  - 预训练与迁移学习
  - **计算机视觉领域的预训练**
  - 自然语言处理领域的预训练
- **语言模型与预训练**
- **上下文词嵌入模型**

计算机视觉领域的预训练

在ImageNet图像数据集上的预训练对比实验概图（基于CNN的多层网络结构）



本章内容简介

- **什么是预训练**
  - 预训练与迁移学习
  - 计算机视觉领域的预训练
  - **自然语言处理领域的预训练**
- **语言模型与预训练**
- **上下文词嵌入模型**

自然语言处理领域的预训练

计算机视觉领域的迁移学习思想可以移植到自然语言处理领域

- 事实上，与之前介绍的将同一图像数据集ImageNet拆分得到预训练数据集和微调阶段训练数据集的迁移学习设置不同的是，在**自然语言处理任务中，预训练阶段和微调阶段不一定采用完全相同类型的数据集**。
- 此时预训练学习思想的本质是，将训练任务拆解成共性学习和特性学习两个阶段：
  - A. **共性学习阶段**：在通用任务上进行预训练，得到初始参数。这里的“通用任务”一般是指在无标注数据集上的语言模型预测任务;
  - B. **特性学习阶段**：对共性阶段学到的参数在下游自然语言处理任务上微调，从而获得任务相关的优化参数。
- 在自然语言处理领域中，基于**无标注数据集上语言模型预测任务**的预训练方法在**多项下游任务**，如自然语言推断、文本蕴含、情感分析等上都获得了很好的效果提升。

本章内容简介

- **什么是预训练**
- **语言模型与预训练**
  - 什么是语言模型
  - 语言模型的实现方法：（一）N-gram语言模型
  - 语言模型的实现方法：（二）神经网络语言模型(NNLM)
  - 什么是词嵌入
  - Word2Vec之CBOW和Skip-gram
  - Word2vec的优化策略之一：Hierarchical Softmax
  - Word2vec的优化策略之二：Negative Sampling（负采样）
- **上下文词嵌入模型**

本章内容简介

- 什么是预训练
- 语言模型与预训练
  - 什么是语言模型
  - 语言模型的实现方法：（一）N-gram语言模型
  - 语言模型的实现方法：（二）神经网络语言模型(NNLM)
  - 什么是词嵌入
  - Word2Vec之CBOW和Skip-gram
  - Word2vec的优化策略之一：Hierarchical Softmax
  - Word2vec的优化策略之二：Negative Sampling（负采样）
- 上下文词嵌入模型

什么是语言模型

这里举个例子来辅助介绍。

例1: 假设有下面3个句子，其中哪个更像一个合理的句子?

- S1：我 | 想 | 要 | 吃 | 一个 | 苹果
- S2：苹果 | 我 | 吃 | 想 | 要 | 一个
- S3：一个 | 想 | 要 | 我 | 苹果 | 吃

语言模型与预训练

什么是语言模型

- 为了判断句子的合理性，一种办法是计算句子在一个足够大的语料中的出现概率（即存在的可能性）。
- 假设句子S由n个词$w_1, w_2, \dots, w_n$按顺序组成，则S的概率可以计算为这些词的联合概率：

$$\boldsymbol{P(S) = P(w_1, w_2, \dots, w_n)}$$

$$\boldsymbol{= P(w_1)P(w_2|w_1)P(w_3|w_1, w_2) \cdots P(w_n|w_1, w_2, \dots, w_{n-1})}$$

这里根据贝叶斯公式，将联合概率分解为若干条件概率的乘积，其通项$P(w_t|w_1, w_2, \dots, w_{t-1})$表示如果前导词为$w_1, w_2, \dots, w_{t-1}$，则当前词为$w_t$的概率。

语言模型与预训练

什么是语言模型

- 例如，对于句子S“***我|想 |要| 吃|一个|苹果***”，则

$$P(S) = P(w_1, w_2, \dots, w_n) = P(\text{“我”}) \cdot P(\text{“想”}|\text{“我”}) \cdot P(\text{“要”}|\text{“我想”}) \cdot P(\text{“吃”}|\text{“我想要”}) \cdot P(\text{“一个”}|\text{“我想要吃”}) \cdot P(\text{“苹果”}|\text{“我想要吃一个”})$$

- **语言模型**通常构建为字符串S的概率分布P(S)，这里的P(S)实际上反映的是S作为一个句子出现的概率。因此某句子出现的概率越大，则表明该句子越合理。

- 等式右边的式子中各项条件概率就是该语言模型（LM）的参数。如果得到了这些参数，就可以计算得到句子出现的概率。

语言模型与预训练

如何求解语言模型的参数?

简单方法的就是直接基于频率，即

$$P(w_t|w_1, w_2, \dots, w_{t-1}) = \frac{count(w_1 w_2 \dots w_{t-1} w_t)}{count(w_1 w_2 \dots w_{t-1})}$$

- 例如，为了计算P(“一个”|“我想要吃”)，可以分别计算出 “我想要吃一个” 和 “我想要吃” 在语料库中的出现频率，然后相除，即

$$P(\text{“一个”}|\text{“我想要吃”}) = \frac{count(\text{“我想要吃一个”})}{count(\text{"我想要吃"})}$$

- 但这种方法**存在两个问题**:
  - A. 模型中**自由参数的数目巨大**，且随着句子（字符串）长度的增加呈现指数级增长。
  - **B. 数据稀疏**。某些组合在训练语料中频率较低甚至为0，则根据公式，导致整个句子的概率为0。

为了克服这两个问题，目前计算语言模型参数的常见方法有两类，即**N-gram和神经网络语言模型NNLM**。下面将分别介绍。

本章内容简介

- **什么是预训练**
- **语言模型与预训练**
  - 什么是语言模型
  - **语言模型的实现方法：（一）N-gram语言模型**
  - 语言模型的实现方法：（二）神经网络语言模型(NNLM)
  - 什么是词嵌入
  - Word2Vec之CBOW和Skip-gram
  - Word2vec的优化策略之一：Hierarchical Softmax
  - Word2vec的优化策略之二：Negative Sampling（负采样）
- **上下文词嵌入模型**

语言模型的实现方法：（一）N-gram语言模型

N-gram语言模型引入了马尔科夫假设

- 为解决自由参数数目过多的问题，N-gram语言模型引入了马尔科夫假设，即**假设一个词出现的概率只与它前面的n个词有关**。
  - A. 当n=1时，假设一个词与周围的词独立，称为Uni-gram，也就是一元语言模型，此时自由参数的数量级是词典大小$|V|$。
  - B. 当n=2时，假设一个词的出现仅与它前面的一个词有关，称为Bi-gram，也叫一阶马尔科夫链，此时自由参数的数量级是$|V|^2$。
  - C. 当n=3时，假设一个词的出现仅与它前面的两个词有关，称为Tri-gram，也叫二阶马尔科夫链，此时自由参数的数量级是$|V|^3$。
- 如何确定n的取值？通常情况下，n的取值不能够太大，否则自由参数的数量将仍然很多，因此一般情况下只使用上述取值，即n=1、2或3。
- 为了克服数据稀疏问题，N-gram语言模型通常采用**平滑化**的方法。例如可采用加1的方法（也称为拉普拉斯平滑）

预训练模型

本章内容简介

- **什么是预训练**
- **语言模型与预训练**
  - 什么是语言模型
  - 语言模型的实现方法：（一）N-gram语言模型
  - **语言模型的实现方法：（二）神经网络语言模型(NNLM)**
  - 什么是词嵌入
  - Word2Vec之CBOW和Skip-gram
  - Word2vec的优化策略之一：Hierarchical Softmax
  - Word2vec的优化策略之二：Negative Sampling（负采样）
- **上下文词嵌入模型**

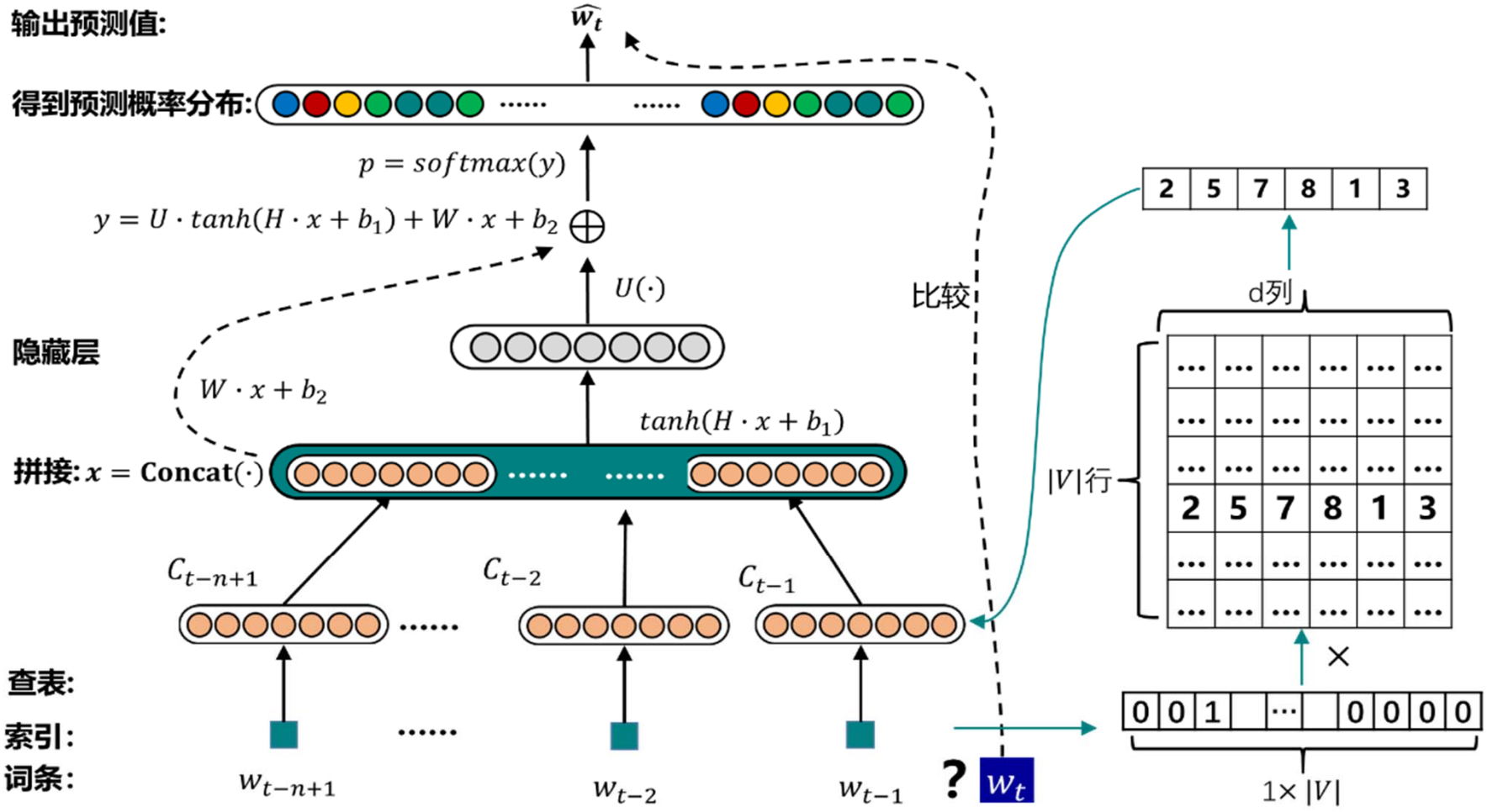
语言模型的实现方法：（二）神经网络语言模型(NNLM)

用神经网络中的参数θ代替了N-gram语言模型中的自由参数

- 神经网络语言模型的学习任务也是根据语句中前$n-1$个词来预测下一个词w_n，其训练的目标是最大化似然函数：$P(w_n|w_1, w_2, \dots, w_{n-1}; \theta)$。其与上一小节介绍的N-gram语言模型最大的不同是**其使用神经网络中的参数θ（θ通常包括神经网络常见的权重系数以及词嵌入矩阵等）代替了N-gram语言模型中的指数级自由参数**。
- 另外，对于N-gram语言模型中容易出现的数据稀疏问题，神经网络语言模型由于最后在**输出预测概率时使用softmax进行归一化**，所以求解的概率是平滑的。
- 因此，现在的问题是：**如何设计一个神经网络**，并利用大量的语料训练，来完成上述语言模型建模的任务。

语言模型的实现方法：（二）神经网络语言模型(NNLM)

神经网络语言模型（NNLM）（Bengio，2003）



NNLM正向传播的基本过程

输入：当前词w_t的前n-1个词（用其在词汇表中的索引表示）

输出：当前词w_t 的概率分布p（含|V|个元素，分别表示其是词汇表中每个词的概率），并根据概率最大值找到对当前词的预测结果$\widehat{w_t}$。

步骤：

自底向上看：

1) 查表。对于当前词w_t所在句子中的前$n-1$个词w_{t-n+1}，…，w_{t-2}，w_{t-1}，到词汇表嵌入矩阵C中查询，得到返回的向量$C(w_{t-n+1})$、…、$C(w_{t-2})$；$C(w_{t-1})$

2) 上下文向量的拼接。将得到的向量首尾拼接，记为x，即

$$x = [C(w_{t-n+1}); \ldots ; C(w_{t-2}) ; C(w_{t-1})] \qquad (11.3)$$

因此，$x \in \mathbb{R}^{(n-1)d \times 1}$。

3) 隐藏层计算：在隐藏层中，对x进行特征变换处理，得到y：

$$y = U \cdot \tanh(H \cdot x + b_1) + W \cdot x + b_2 \qquad (11.4)$$

其中，$U \in \mathbb{R}^{|V| \times h}$，$H \in \mathbb{R}^{h \times (n-1)d}$，$b_1 \in \mathbb{R}^{h \times 1}$。h为隐藏层神经元数。$W \in \mathbb{R}^{|V| \times (n-1)d}$，$b_2 \in \mathbb{R}^{|V| \times 1}$。公式（11.3）中$W \cdot x + b_2$为从输入层到输出层的直连线性变换，其是一个可选的处理，如图11.4中虚线所示。

4) 输出预测概率：对y进行归一化处理，得到输出概率分布p：

$$p = \mathrm{softmax}(y) \qquad (11.5)$$

5) 得到预测的当前词。从p中找到概率值最大处的索引，查表找到当前词的预测

语言模型的实现方法：（二）神经网络语言模型(NNLM)

NNLM总结

- 可见，上述神经网络语言模型也借鉴了N-gram模型的思想，即根据前n-1个词语预测下一个词语的概率。主要区别在于其用神经网络中的参数取代了N-gram语言模型中的指数级自由参数。另外，由于模型是使用当前词所在的上下文词训练学习得到，因而也是一个**无监督的过程。**

- 上述建模语言模型的神经网络实际上有两个用途：

  A. **语言模型**，即能够根据上文预测后接单词是什么；

  B. **同时学习得到一个副产品即矩阵C**，其包含了每个词的向量表示（即词嵌入）。该词嵌入也可以在后续的其它下游自然语言处理任务中用于词嵌入矩阵参数的初始化，进而加速下游任务的训练。

本章内容简介

- **什么是预训练**
- **语言模型与预训练**
  - 什么是语言模型
  - 语言模型的实现方法：（一）N-gram语言模型
  - 语言模型的实现方法：（二）神经网络语言模型(NNLM)
  - 什么是词嵌入
  - Word2Vec之CBOW和Skip-gram
  - Word2vec的优化策略之一：Hierarchical Softmax
  - Word2vec的优化策略之二：Negative Sampling（负采样）
- **上下文词嵌入模型**

本章内容简介

- 什么是预训练
- 语言模型与预训练
  - 什么是语言模型
  - 语言模型的实现方法：（一）N-gram语言模型
  - 语言模型的实现方法：（二）神经网络语言模型(NNLM)
  - 什么是词嵌入
  - Word2Vec之CBOW和Skip-gram
  - Word2vec的优化策略之一：Hierarchical Softmax
  - Word2vec的优化策略之二：Negative Sampling（负采样）
- 上下文词嵌入模型

什么是词嵌入

词嵌入概述

维度	1	2	3	4	5	6	7	8	9	10
not:	-0.022975	0.087888	-0.24248	-0.074691	0.02824	0.22998	-4.5327	0.5373	-0.12518	-0.66138
very:	0.24487	-0.15216	-0.3001	-0.22925	0.071235	-0.37077	-4.2901	0.068465	-0.31345	-0.7891
good:	-0.069254	0.37668	-0.16958	-0.27482	0.25667	-0.20293	-4.1122	0.02595	-0.27085	-0.87003

- 之前提到，深度学习预训练的**一个副产品是词嵌入**（word embedding）。
- 词嵌入是一种**低维实数向量**，也被称之为分布式表征（Distributed representation）。
- **词嵌入的维度**一般为 50 维、 100 维、200维或者300维等，且每个元素的取值范围一般为[-1, +1]。

什么是词嵌入

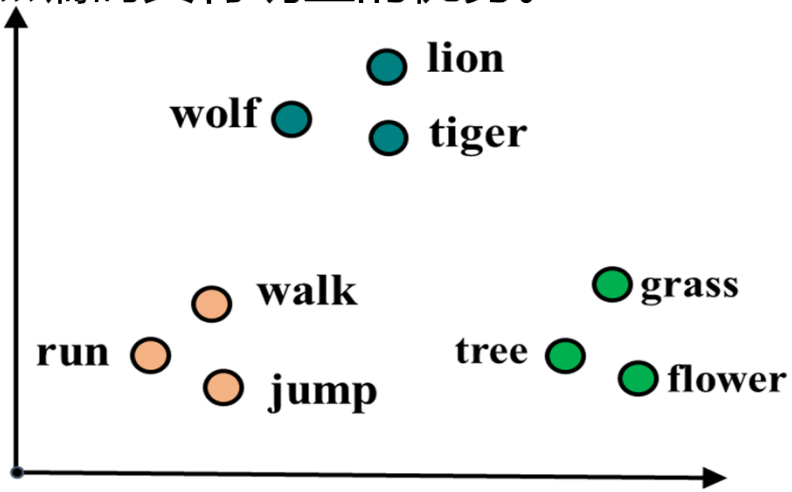
词嵌入有助于克服“语义鸿沟”

- 在传统的独热编码表示方法中，存在的问题是“**词汇鸿沟**”现象。 例如“菠萝”和“凤梨”在独热编码体系下，二者之间的相似性难以有效表达：

  “菠萝”： [0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 ...]

  “凤梨”： [0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 ...]

- 而**词嵌入使得在语义上相似或相关的词，在距离上更接近**。这种表达模式相对于独热编码具有明显的优势。



本章内容简介

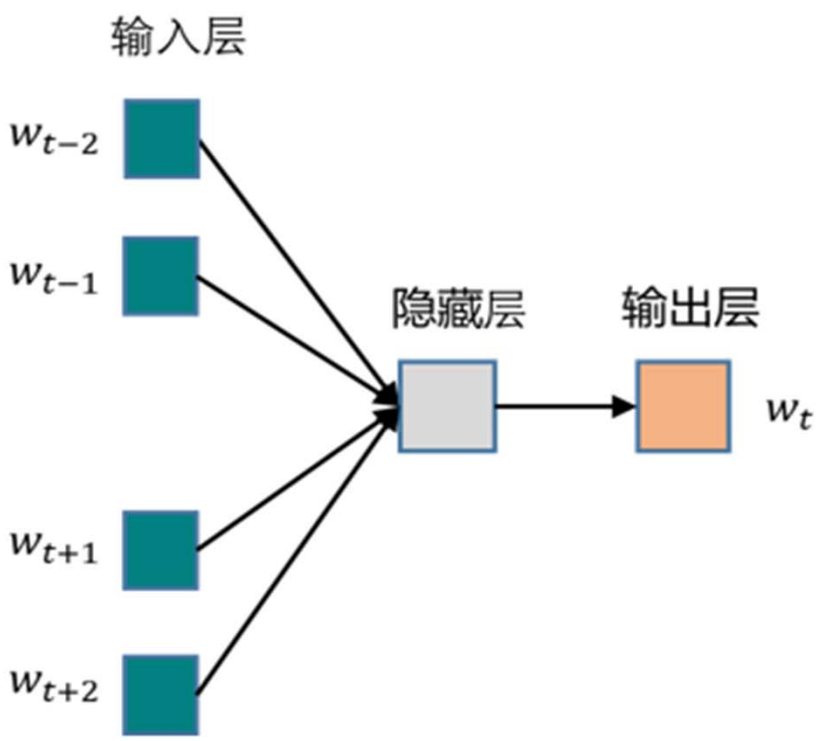
- **什么是预训练**
- **语言模型与预训练**
  - 什么是语言模型
  - 语言模型的实现方法：（一）N-gram语言模型
  - 语言模型的实现方法：（二）神经网络语言模型(NNLM)
  - 什么是词嵌入
  - **Word2Vec之CBOW和Skip-gram**
  - Word2vec的优化策略之一：Hierarchical Softmax
  - Word2vec的优化策略之二：Negative Sampling（负采样）
- **上下文词嵌入模型**

Word2vec之CBOW和Skip-gram

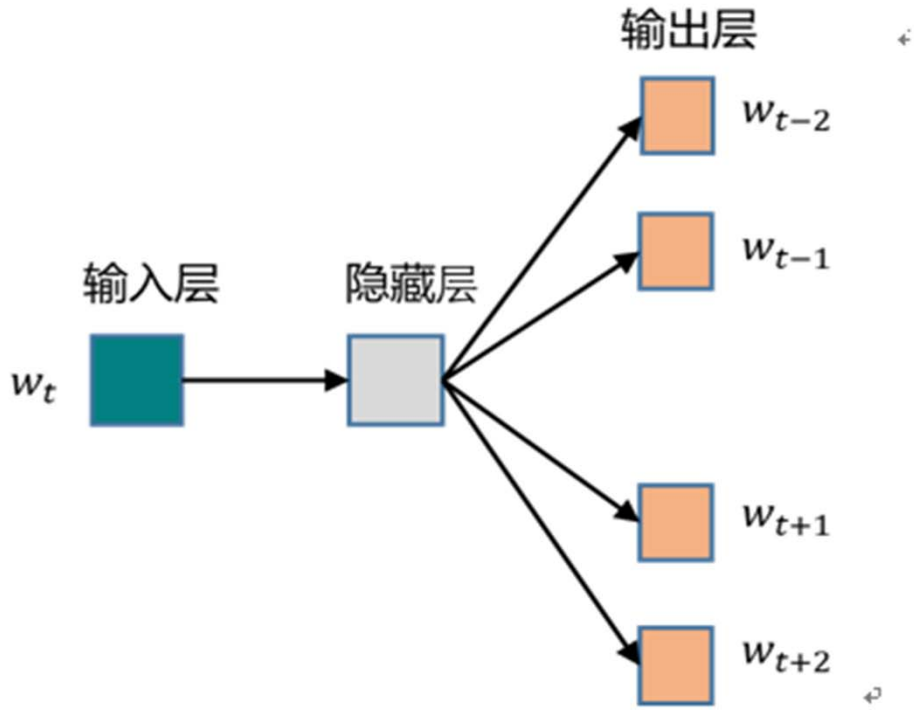
Word2vec概述

- 前文提到NNLM可以学习得到一个副产品，即可以训练得到每个词的向量表示（即词嵌入）。实际上，**后续有很多基于神经网络的办法得到类似这样的词嵌入向量，如这里要介绍的Word2vec**。
- Word2vec是谷歌公司推出的一款可以获得词嵌入的工具，**其实现了2013年由Tomas Mikolov提出的CBOW (Continuous Bag-of-Words Model)和Skip-gram模型**。另外，Word2vec使用了Hierarchical Softmax、Negative Sampling等训练优化方法。

Word2vec之CBOW和Skip-gram



(a) CBOW



(b) Skip-gram

■ CBOW 利用上下文词来预测中心词是什么，即此时$w(t)$是被预测的词；

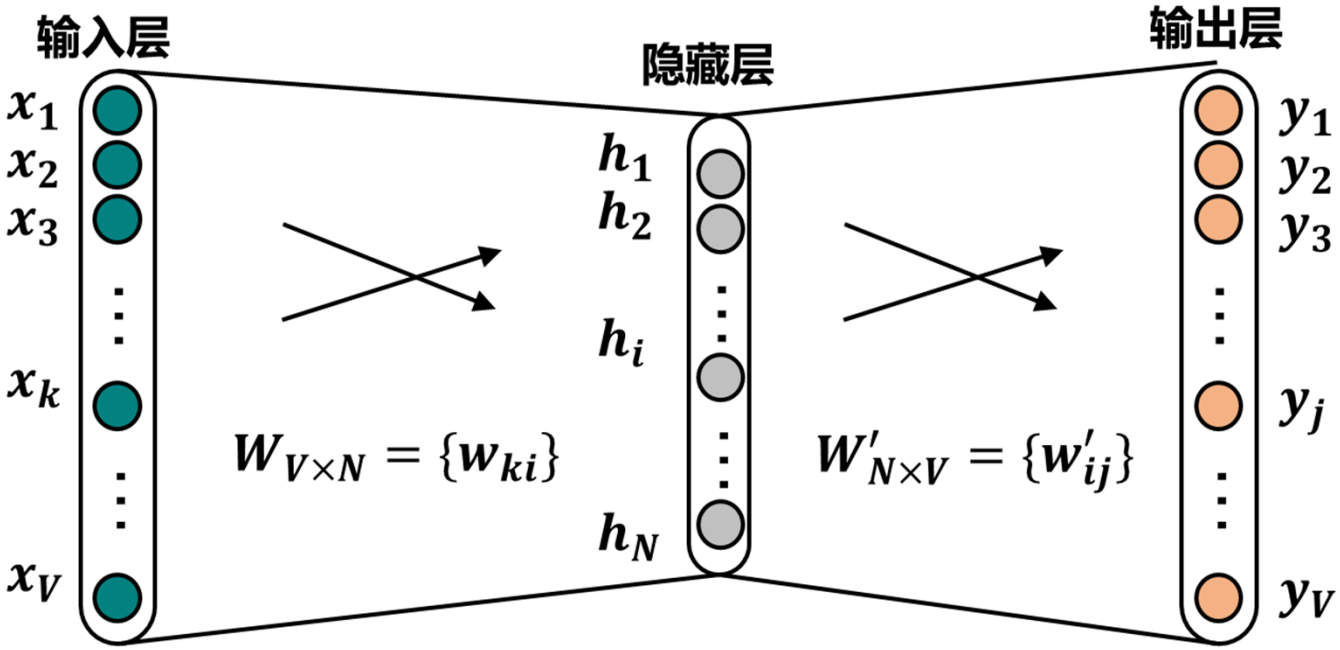
■ **CBOW** 的 计算复杂度为 O(V)。因为其使用上下文词预测中心词，每个窗口只预测一次。

■ **Skip-gram** 模型利用中心词$w(t)$来预测上下文词是什么；

■ **Skip-gram** 的计算复杂度为 O(KV)。因为其使用中心词预测周围 K 个词，每个窗口要预测 K 次。

Word2vec之CBOW和Skip-gram

隐藏层到输出层的softmax计算开销很大



- 但尽管如此，Word2vec（无论是CBOW模型还是Skip-gram模型）中从隐藏层到输出层的softmax计算仍存在计算开销很大的问题。
- 因此，为了提高Softmax计算的效率，Word2vec进一步采用了两种优化策略：即Hierarchical Softmax和Negative Sampling。这两种策略的共同点是，不再使用全连接计算。下面两节中将分别予以介绍。

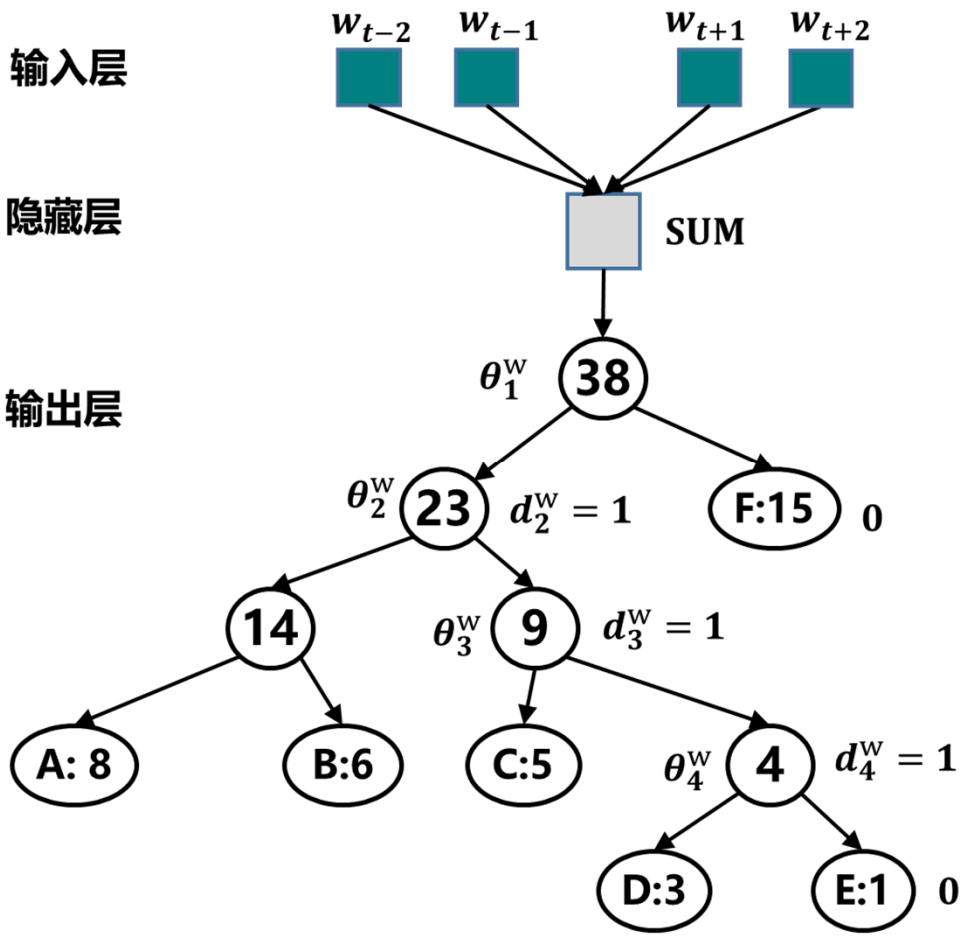
预训练模型

本章内容简介

- 什么是预训练
- 语言模型与预训练
  - 什么是语言模型
  - 语言模型的实现方法：（一）N-gram语言模型
  - 语言模型的实现方法：（二）神经网络语言模型(NNLM)
  - 什么是词嵌入
  - Word2Vec之CBOW和Skip-gram
  - **Word2vec的优化策略之一：Hierarchical Softmax**
  - Word2vec的优化策略之二：Negative Sampling（负采样）
- 上下文词嵌入模型

Word2vec的优化策略之一：Hierarchical Softmax

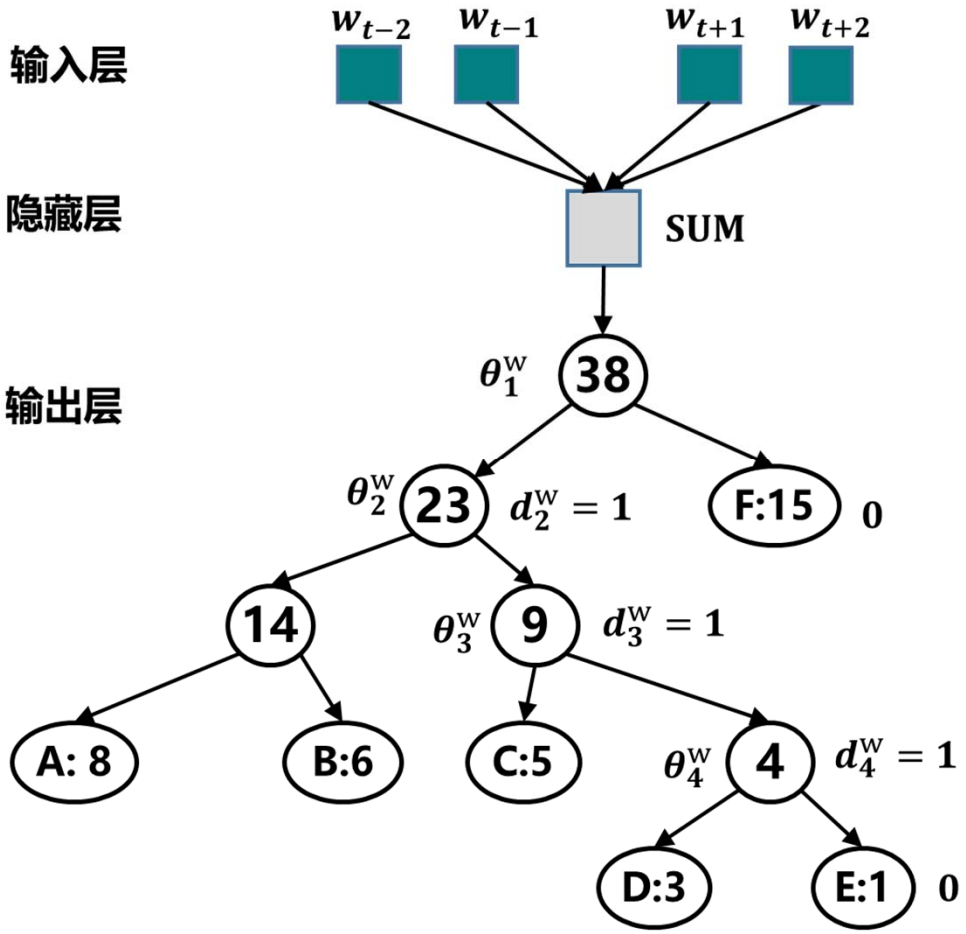
（一）Hierarchical Softmax的基本原理



- Hierarchical Softmax的基本原理是，通过构建哈夫曼树（一种二叉树）来取代隐藏层到输出层的全连接计算，从而可以快速找到概率最大的输出词。
- 如图所示，在Hierarchical Softmax机制中，隐藏层（投射层）直接与哈夫曼树的根节点相连。哈夫曼树的叶子节点表示词汇表中的每个词（共计|V| 个）。
- 从根节点出发到每个叶子节点都有一条唯一的经过非叶节点的路径。每个非叶节点都有一个向量。

Word2vec的优化策略之一：Hierarchical Softmax

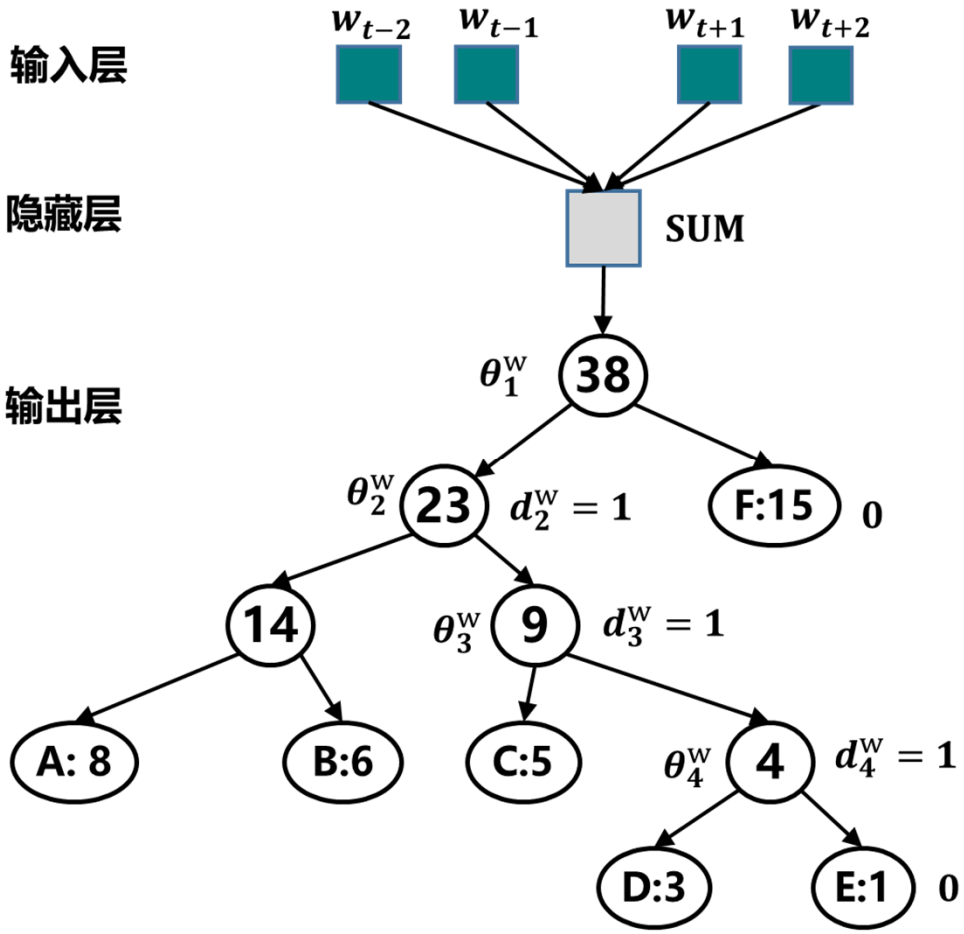
（一）Hierarchical Softmax的基本原理



- 从图可见，Hierarchical Softmax优化策略实际上是利用二叉树，将输出层上的多类分类问题转化为多个二分类问题。
- 每个二分类问题实际上回答的是：当前预测的结果是某个词w的概率是否大于不是该词的概率，如果前者较大，则可以直接输出该词，不必遍历该树上的其它节点；否则继续沿着该分支继续向下探索。
- 哈夫曼树上的高频词的节点比较靠近根节点，因此会优先探索。

Word2vec的优化策略之一：Hierarchical Softmax

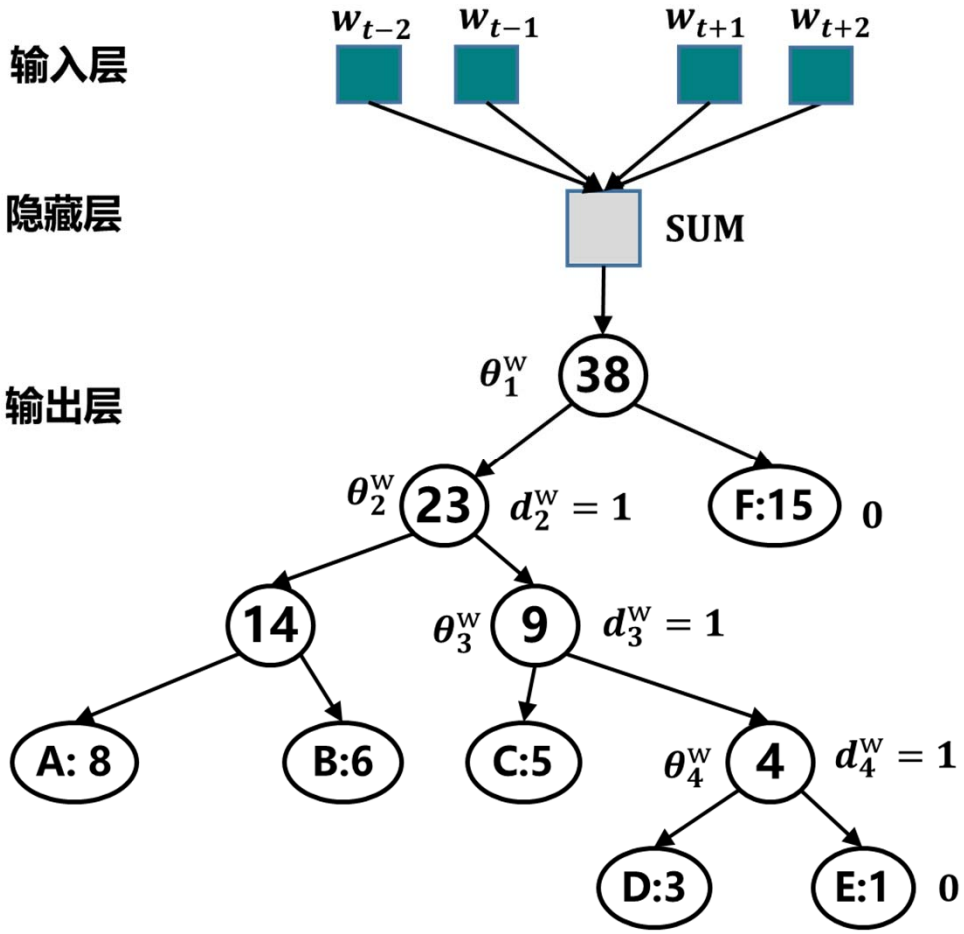
（二）Hierarchical Softmax中哈夫曼树的构建



- Hierarchical Softmax中，主要利用标签（如单词）的出现频率构建哈夫曼树。标签的频率越高，则其从根节点到该标签的路径越短，因此总体上会取得较好的效率提升。
- 图中，假设叶节点上有（A，B，C，D，E，F）6个单词，且其权重（本例中假设为词频）分为(8，6，5，3，1，15)。则哈夫曼树构建的过程是对上述节点进行多次合并以最终形成树结构。
- 每次合并是将根节点权重最小的两棵树合并，以得到一颗新树，新树的根节点权重为原来参与合并的两颗子树根节点的权重之和。树。

Word2vec的优化策略之一：Hierarchical Softmax

（二）Hierarchical Softmax中哈夫曼树的构建

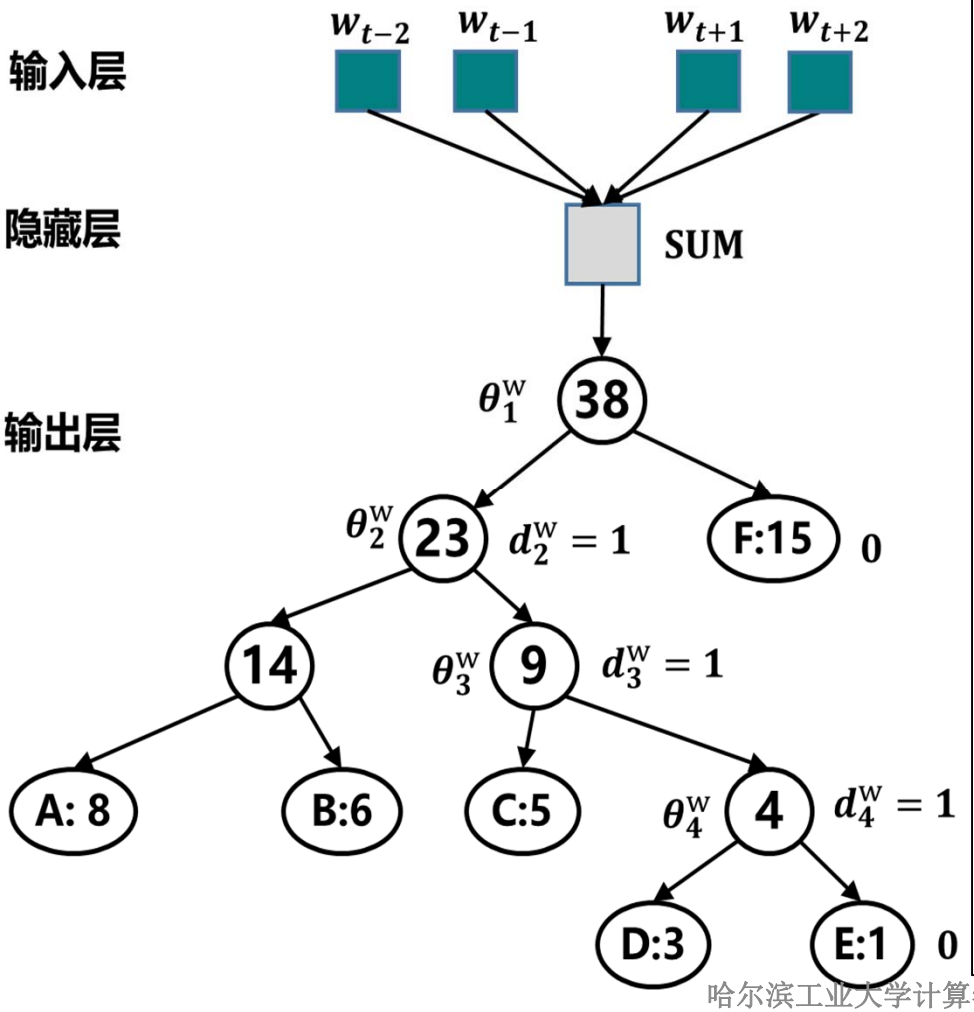


哈夫曼树中叶子节点（即单词）的编码：

- 对于除根节点以外的每个节点，可以约定其到左侧孩子节点的边编码为0，其到右侧孩子节点的边编码为1。则可以得到每个叶节点（单词）的编码。
- 例如A的编码000，B的编码是001，等等。可见，由于词频较高的常用词节点更靠近根节点，因此其编码长度较短。

Word2vec的优化策略之一：Hierarchical Softmax

（三）Hierarchical Softmax的预测计算过程



- 结合左图可见，借助于构建好的哈夫曼树，隐藏层到输出层的softmax映射计算可以沿着哈夫曼树从根节点开始探索其中的节点。
- 每个节点的判断实际上一种二元逻辑回归，例如规定左子树为负类，右子树为正类，则有

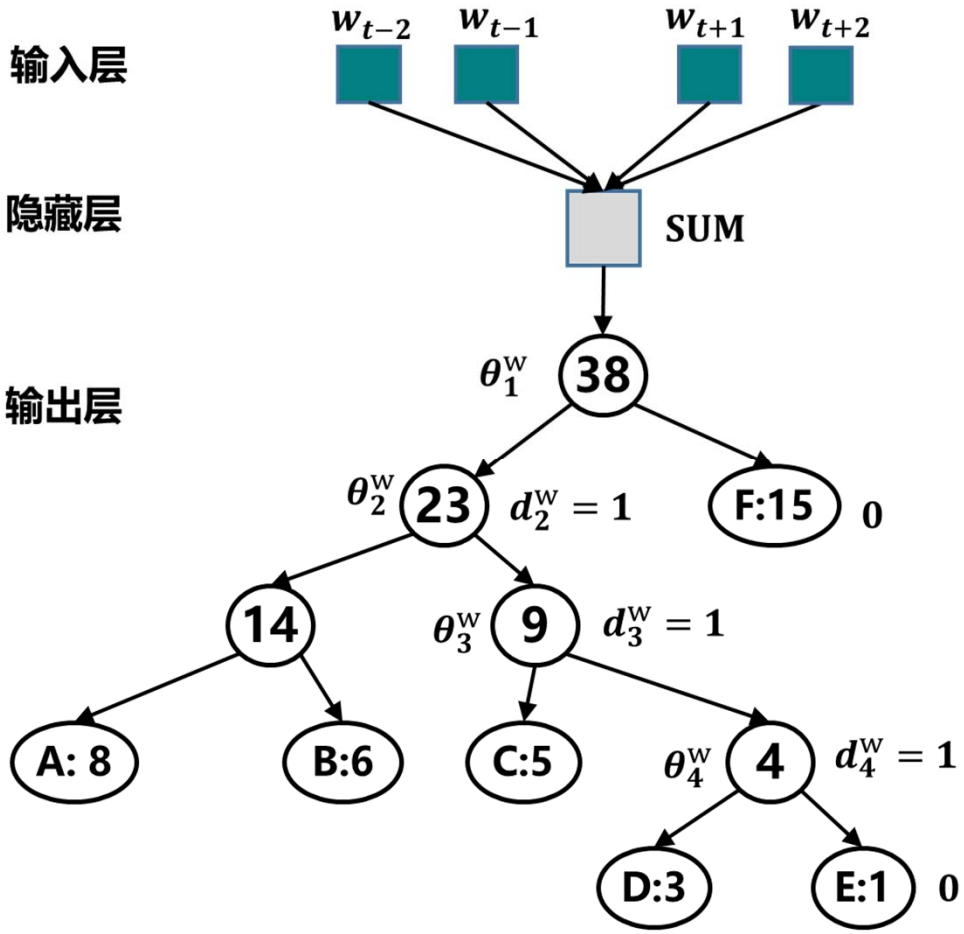
$$P(+) = \sigma(x_w^T\theta) = \frac{1}{1 + e^{-x_w^T\theta}}$$

其中，x_w为当前节点的词向量，θ为模型参数。

- 如果某节点的某个子树为叶节点，且该其概率较大，则可以不必探索其它节点，直接得到预测结果。

预训练与迁移学习

（三）Hierarchical Softmax的预测计算过程



- 由于高频词的探索路径比较短，因而通过这种机制，隐藏层到输出层的softmax映射计算复杂度得以降低：
- 如果采用全连接层，计算复杂度为$\mathcal{O}(\mathrm{kh})$，其中k为类别数量（即词汇表大小），h为文本特征嵌入维度。而Hierarchical Softmax中借助于哈夫曼树，其计算复杂度为$\mathcal{O}(\mathrm{h}\log_2 k)$。

本章内容简介

- **什么是预训练**
- **语言模型与预训练**
  - 什么是语言模型
  - 语言模型的实现方法：（一）N-gram语言模型
  - 语言模型的实现方法：（二）神经网络语言模型(NNLM)
  - 什么是词嵌入
  - Word2Vec之CBOW和Skip-gram
  - Word2vec的优化策略之一：Hierarchical Softmax
  - **Word2vec的优化策略之二：Negative Sampling（负采样）**
- **上下文词嵌入模型**

Word2vec的优化策略之二：Negative Sampling（负采样）

负采样技术的使用背景

- **为什么要负采样?**
  - 使用哈夫曼树来代替传统的全连接神经网络， Hierarchical Softmax优化策略可以使Word2vec模型获得训练效率的提升。但如果训练集的中心词w是相对低频的非常见词，则在哈夫曼树中将会从根节点向下的判断路径会较长，其训练时间开销将仍然很大。
  - Negative Sampling是Word2vec模型的另一种优化策略，其没有采用哈夫曼树，而是使用负采样的方法用相对较低的成本获得大量的二类分类训练样本。

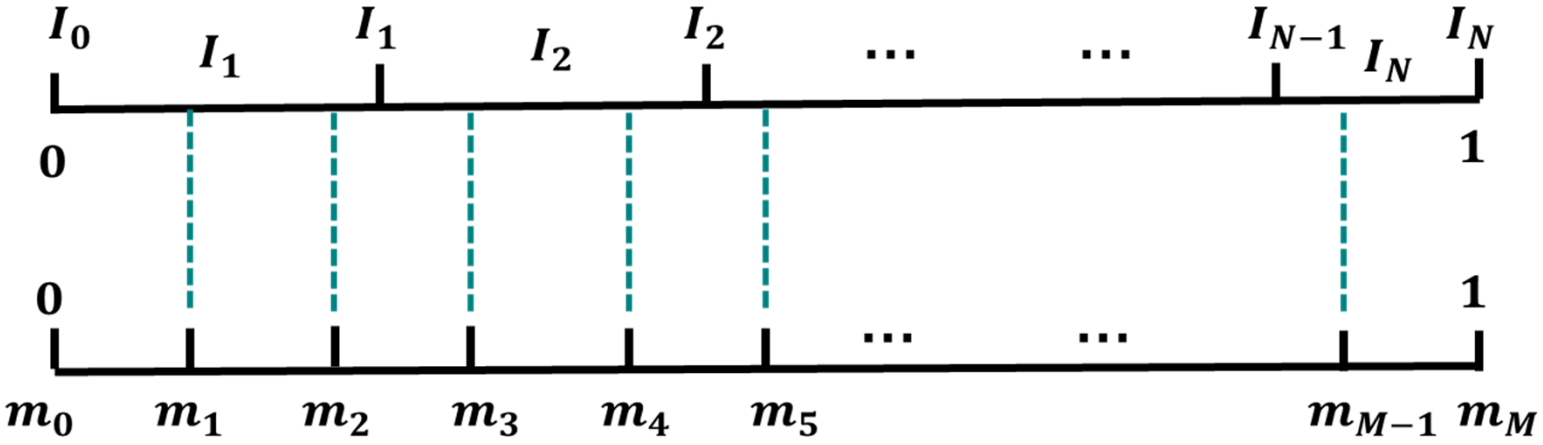
Word2vec的优化策略之二：Negative Sampling（负采样）

什么是负采样?

- 对于某个训练样本，假设其中心词为w，上下文共有2c个词，记为context(w)。由于w的确是上下文context(w)的中心词，因此w（可以视为标签）和context(w)可以构成一个训练正例（正类）。**现在的问题是如何构建训练负例（负类）？**
- 所谓负采样**Negative Sampling**就是指通过采样的方法，从词汇表中取出neg个不同于w的词$w_i, i = 1,2,..neg$，将w_i与context(w) 组成负例。每个负例的中心词和上下文在真实语料中未必真实存在。
- 现在，对每个中心词，既有正例又有负例样本，则可以采用二元逻辑回归的方法进行训练语言模型，得到每个词w_i的模型参数θ_i和词向量。

Word2vec的优化策略之二：Negative Sampling（负采样）

如何负采样?



- **如何获得负例样本。**Word2vec中的负采样考虑到了词频，即高频词更有可能被选中。假设词汇表大小为|V|，则每个词w被选中的概率为：

$$len(w) = \frac{count(w)^{3/4}}{\sum_{u \in vocab} count(u)^{3/4}}$$

- 可以通过上图来理解，假设词汇表尺寸所对应的线段长度为1，则可以将其划分为M等份（M>>|V|），则从M个片段中采样出neg个即可，每个片段所对应的词即为选中的负例词。

本章内容简介

- **什么是预训练**
- **语言模型与预训练**
- **上下文词嵌入模型**
  - 静态词嵌入与上下文词嵌入
  - 典型的预训练模型：OpenAI GPT模型
  - 典型的预训练模型：BERT模型

本章内容简介

- 什么是预训练
- 语言模型与预训练
- 上下文词嵌入模型
  - 静态词嵌入与上下文词嵌入
  - 典型的预训练模型：OpenAI GPT模型
  - 典型的预训练模型：BERT模型

上下文词嵌入

静态词嵌入与上下文词嵌入

- **什么是静态词嵌入？**Word2vec存在的问题是，其训练得到的词嵌入本质上是静态的，即训练完毕后，单词（或者字符）的嵌入式表示就确定了，即所谓的静态嵌入。

- **静态嵌入容易带来一些问题**，如难以应对一词多义问题。例如，英文单词“Bank”在不同语境（上下文）中有“银行”、“河堤”等不同含义。而采用Word2vec得到的词嵌入表示可能将这些不同语义混在一起。

上下文词嵌入

静态词嵌入与上下文词嵌入

- **什么是上下文化的词嵌入（Contextualized word embedding）？**因此，理想状态是，对于不同的上下文语境，同一个单词能够得到不同的嵌入，即所谓的上下文化的词嵌入。
- **上下文化的词嵌入基本思路是**：
  - 首先，用语言模型学习得到词嵌入，虽然此时的词嵌入仍然无法区分多义词，**是一种比较通用的混合表征**。
  - 然后，在**具体的下游任务上下文中，根据上下文单词的语义对单词的嵌入表示做相应调整**，以使得调整后的词嵌入能充分表达该词在下游任务上下文语境中的具体含义。
- 因此**概括起来**，上下文嵌入或称动态词嵌入，是先用语言模型学习得到单词的通用嵌入表达，然后根据特定任务上下文对嵌入做动态调整的思路。
- 常见的重要预训练模型：包括OpenAI GPT模型、BERT模型等。

本章内容简介

- 什么是预训练
- 语言模型与预训练
- 上下文词嵌入模型
  - 静态词嵌入与上下文词嵌入
  - **典型的预训练模型：OpenAI GPT模型**
  - 典型的预训练模型：BERT模型

典型的预训练模型：OpenAI GPT模型

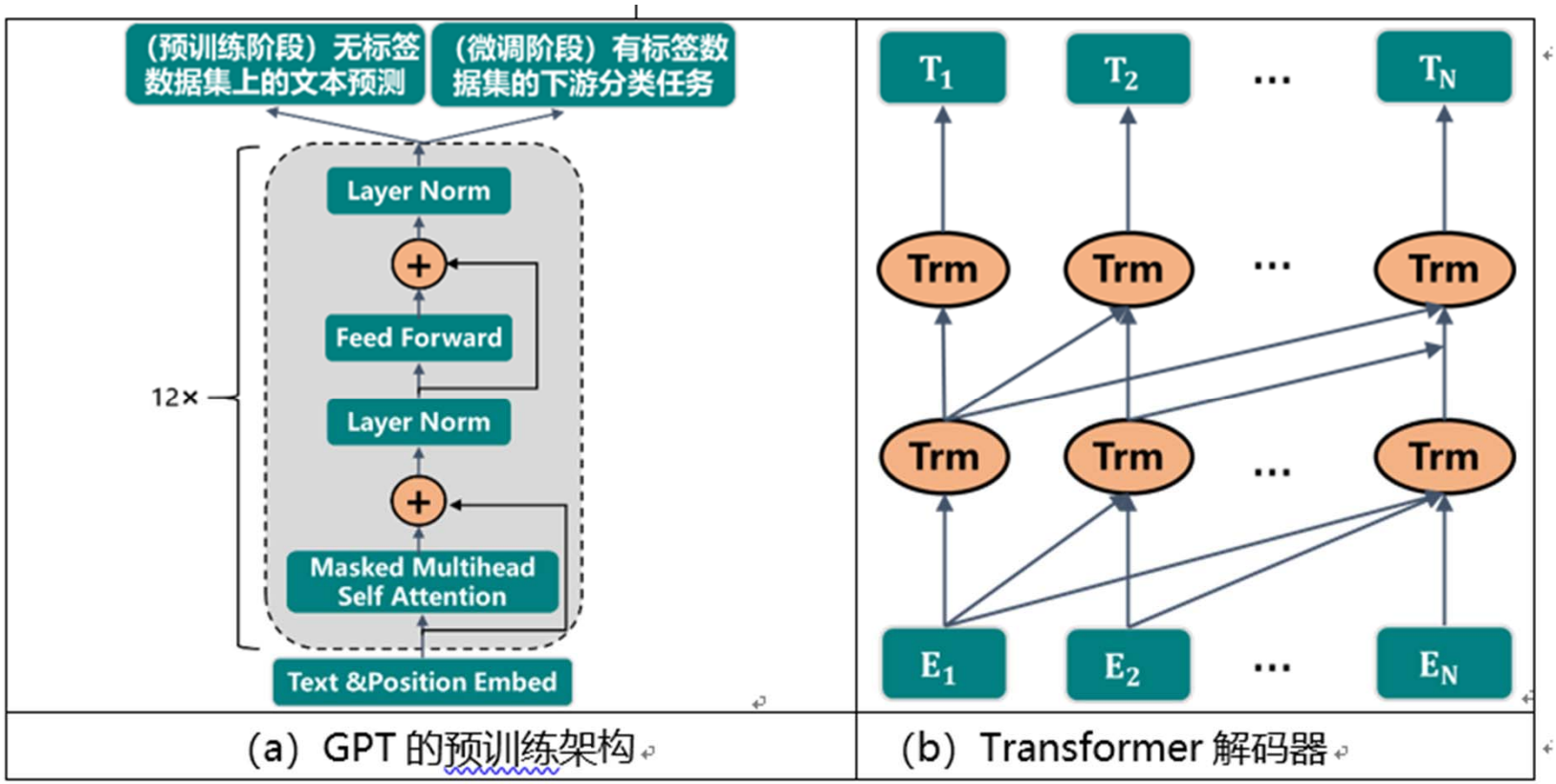
GPT采用语言模型在无标签数据集上训练

- 从相关模型来看，NNLM、Word2vec使用的是无标签数据，**CoVe模型和CVT模型尝试使用有标签数据**做预训练（后者实际上使用的是有标签和无标签数据的混合）。但有标签数据的获得毕竟成本较高。
- 因此后来出现的预训练模型，如GPT模型在预训练阶段仍然使用数量更为丰富的无标签数据，并以语言模型预测作为预训练的任务。
- GPT采用了传统的语言模型进行训练，即基于上文预测下文。与其它预训练学习任务类似，GPT的预训练学习也包括两个阶段：
  - **A. 预训练阶段**
  - **B. 微调阶段**

Alec Radford et al. Improving Language Understanding by Generative Pre-Training. OPENAI https://openai.com/blog/language-unsupervised/. 2018

典型的预训练模型：OpenAI GPT模型

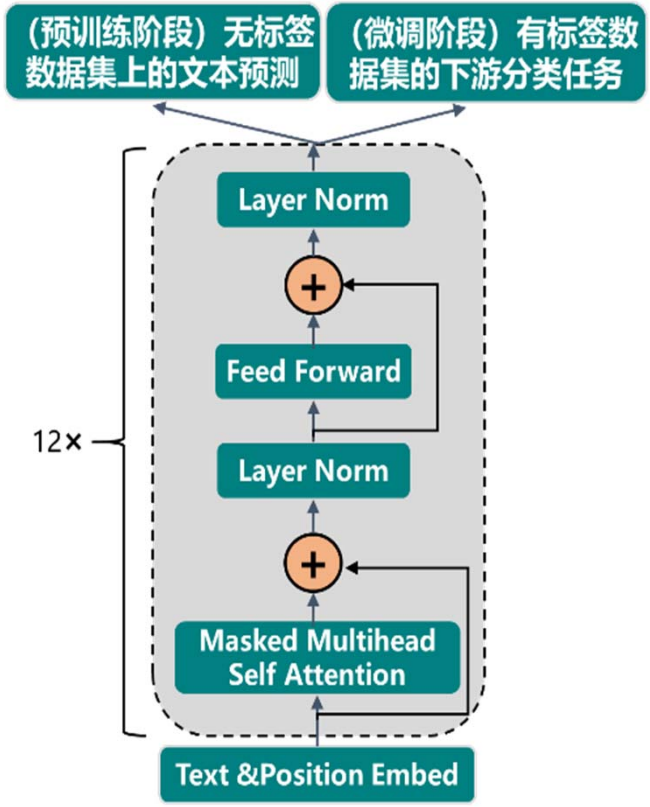
GPT预训练模型使用多层Transformer解码器作为预训练架构



(a) GPT 的预训练架构　(b) Transformer 解码器

典型的预训练模型：OpenAI GPT模型

（一） GPT的预训练阶段



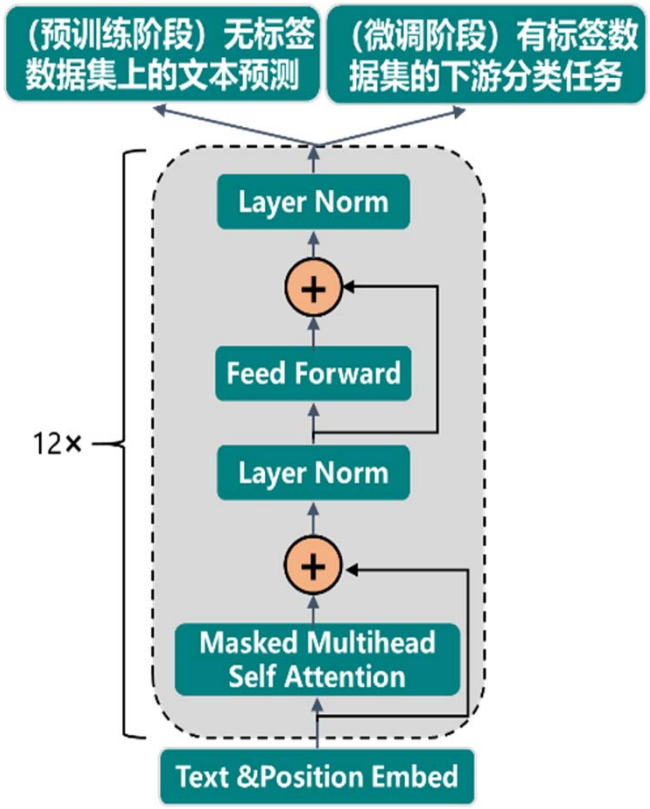
- GPT模型预训练阶段用到的数据集是无标签语料$u = \{u_1, \dots, u_n\}$。**预训练的目的**是利用图中左侧架构学习得到多层Transformer的解码器（参数）。
- 其**具体过程**可以描述如下：

1) 使用多层Transformer的解码器利用上文词预测目标词u的分布$P(u)$（左图自底向上看）：

   1. 第0层：$h_0 = UW_e + W_p$。其中W_e是词语的嵌入矩阵，W_p是位置嵌入矩阵，$U = (u_{-k}, \dots, u_{-1})$是目标词$u$的上文词语向量。
   2. 隐藏层的逐层递归计算方法：$h_l = transformer\_block(h_{l-1}), \forall i \in [1, n]$。$n$是总层数。
   3. 目标词语的概率分布：$P(u) = softmax(h_n W_e^T)$。

典型的预训练模型：OpenAI GPT模型

（一） GPT的预训练阶段



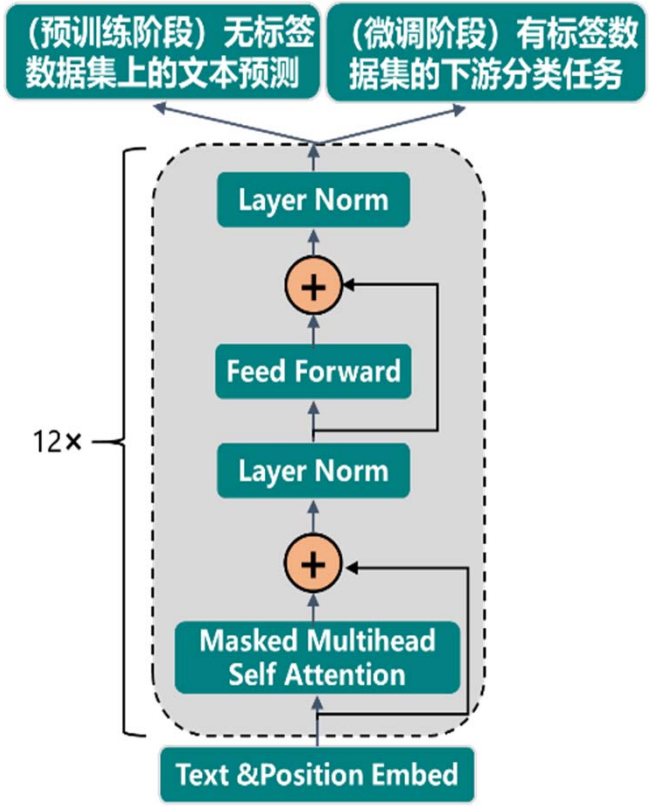
2）优化方法：基于上述预测结果，使用标准语言建模目标来最大化似然函数，从而学习得到Transformer的解码器参数：

$$L_1(u) = \sum_i log\ P(u_i|u_{i-k}, \dots, u_{i-1}; \Theta)$$

- 其中，k是上下文窗口的大小，条件概率P使用参数为Θ的神经网络建模。这里，语言模型的参数Θ实质上就是多层Transformer的解码器中的参数。
- 使用随机梯度下降法对这些参数进行训练。这里之所以命名为$L_1(u)$是因为后续还有在有标注数据集上的其它优化函数。

典型的预训练模型：OpenAI GPT模型

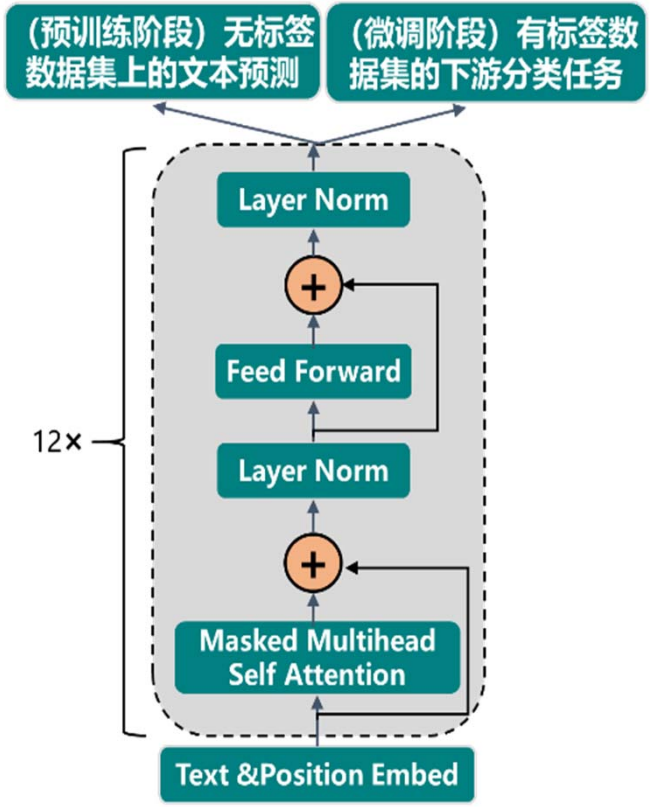
（二）GPT的微调阶段



- **GPT微调阶段**使用预训练阶段的整体框架（图左侧）及将其学习到的模型参数取值作为初始值，并在输出层上将优化目标从“**Text Prediction**”替换为“**Task Classifier**”。
- 微调阶段主要工作是调整模型参数**以适应带有标签数据的不同有监督学习下游NLP任务**，如分类、文本蕴含、相似度计算、多选判断等。

典型的预训练模型：OpenAI GPT模型

（二）GPT的微调阶段



微调阶段的基本过程：

- 对于微调阶段的每个输入样本，其包括输入标记序列$x^1,\ldots,x^m$以及对应的标签y，将其传递给图中左侧的多层Transformer的解码器，从而获得transformer block最终的激活h_l^m，然后**被送入额外增加的参数为W_y的线性输出层**（即图中的“task classification”）来预测标签y：

$$P(y|x^1,\ldots,x^m)=softmax(h_l^m W_y)$$

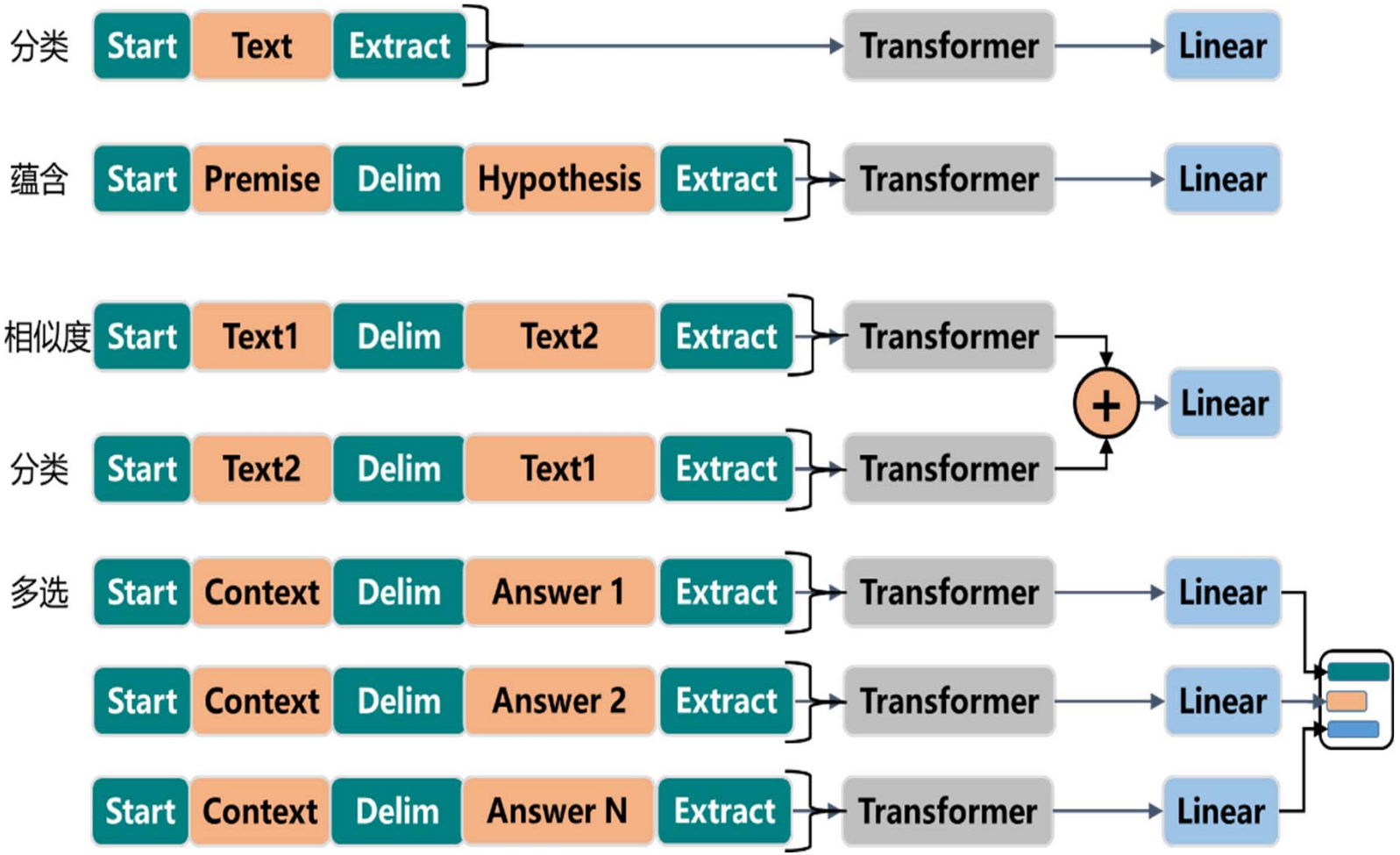
- 因此**微调阶段的训练目标**变为最大化如下函数：

$$L_2(C)=\sum_{(x,y)} logP(y|x^1,\ldots,x^m)$$

- 在微调过程中需要的额外参数除了W_y以外，还有分隔符（delimiter）嵌入。

典型的预训练模型：OpenAI GPT模型

GPT微调阶段的下游任务类型及相应的输入转换



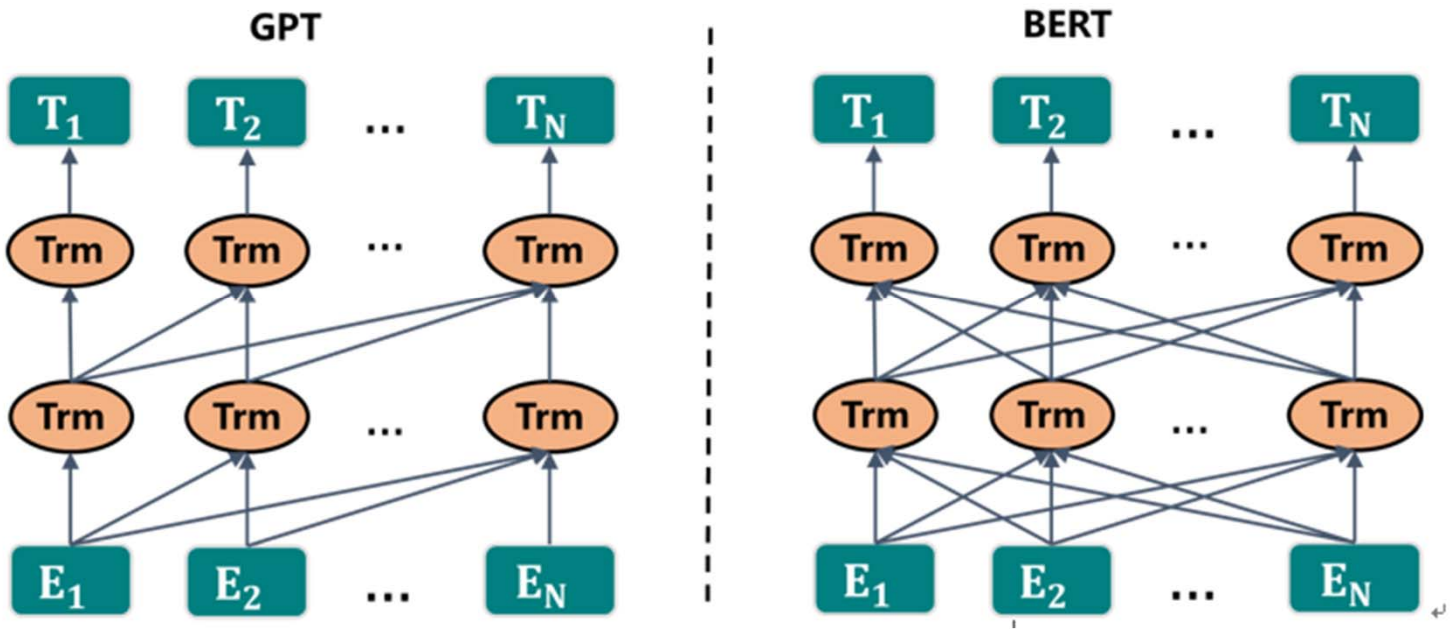
本章内容简介

- **什么是预训练**
- **语言模型与预训练**
- **上下文词嵌入模型**
  - 静态词嵌入与上下文词嵌入
  - 典型的预训练模型：OpenAI GPT模型
  - **典型的预训练模型：BERT模型**

典型的预训练模型：BERT模型

BERT与GPT的区别

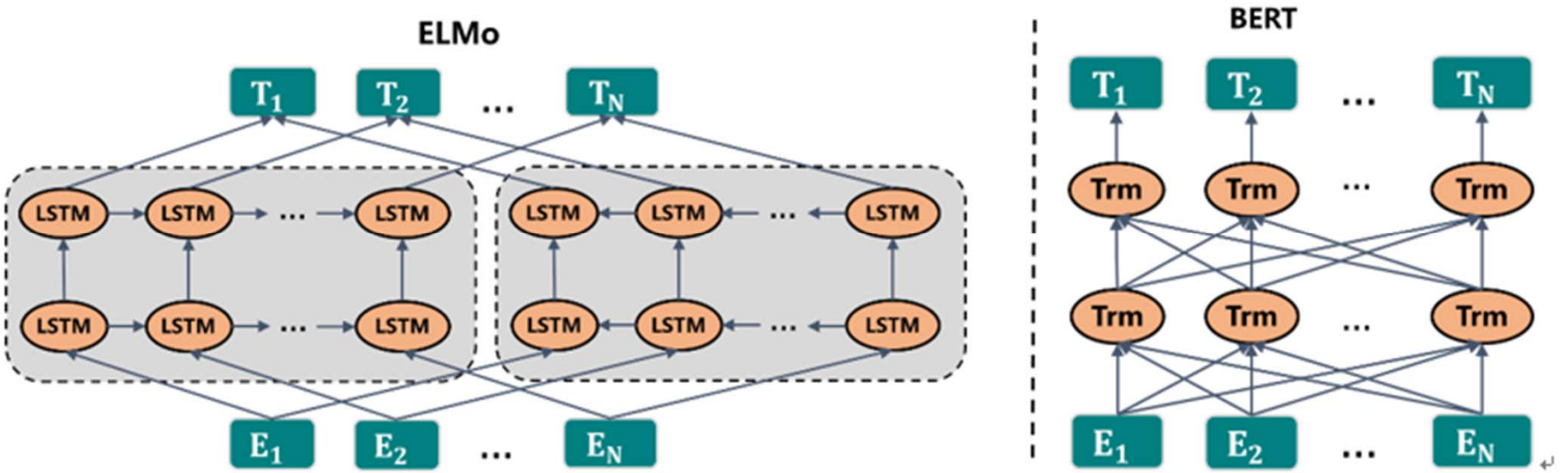
- BERT，全称为Bidirectional Encoder Representation from Transformers，是Google于2018年提出的预训练模型。
- **BERT的模型结构是一种多层双向的Transformer 编码器。**



Jacob Devlin Ming-Wei Chang Kenton Lee Kristina Toutanova. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. 2018. https://arxiv.org/abs/1810.04805

典型的预训练模型：BERT模型

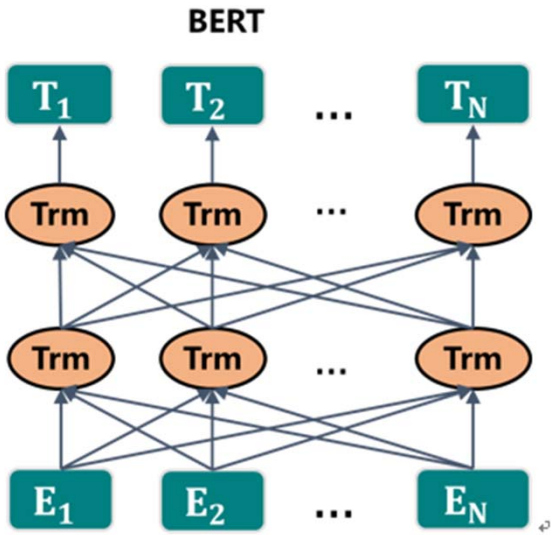
BERT与ELMO的区别



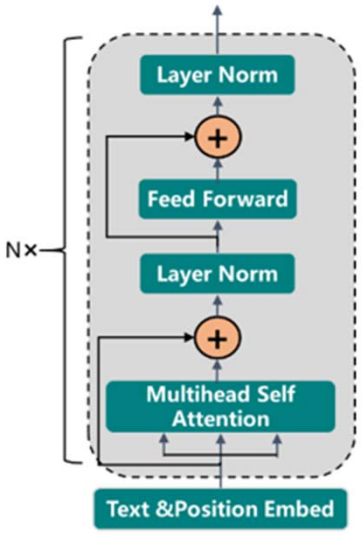
- ELMO（全称为Embedding from Language Models），其也是一种两阶段的预训练学习方法，且训练阶段使用语言模型。
  - BERT与ELMO都是双向语言模型，但ELMO的预训练采用的是双层双向LSTM对上下文进行编码，对每层LSTM将上文向量与下文向量进行拼接作为当前向量；而BERT使用的是transformer。
  - ELMO预训练方法给下游任务提供的是词条的特征形式，而BERT预训练方法则是基于微调（Fine-tuning）的模式。

典型的预训练模型：BERT模型

BERT的预训练架构



(a) BERT 的整体框架



(b) Transformer 编码器结构

- **BERT的输入**，即图中的E1，E2等可以是字符或者词条。中间层的Trm等表示Transformer的编码器组件，每个输入元素所对应的Trm都可以接受来自其它元素的输入信息，因为这是一个全连接。
- **BERT的输出**即T1，T2等，表示隐藏层向量信息。

- **BERT网络的超参数**包括：
  - A. 层数L，即竖直方向Transformer 编码器组件的个数;
  - B. 隐藏层维度尺寸（the hidden size）H;
  - C. 自注意力头的个数（the number of self-attention heads）A。基于这些超参数取值的不同，BERT有两种常用的模型尺寸：1）BERT_BASE: L=12, H=768, A=12； 2) BERT_LARGE: L=24, H=1024, A=16。

典型的预训练模型：BERT模型

（一）BERT的预训练阶段

- 和GPT相同的是，BERT的预训练也是利用无指导的语言模型。不同的是，GPT采用传统语言模型来预训练，而**BERT的预训练则采用了两个如下新的无指导预测任务**：

1) **遮盖语言模型任务（Masked LM）**。BERT的预训练阶段随机遮盖(mask)句子中一定比例（默认值为15%）的词，然后基于其上下文得到遮盖词的隐藏向量后，送到softmax层预测被遮住的词是什么。

例如：将语句“my dog is hairy”变为“my dog is [MASK]”，然后预测[MASK]所在位置是哪个词。

如果被遮盖的词数量较高，可能导致在后续的微调阶段从未见过某些被遮盖的词。为此BERT对这15%进一步分成如下三种处理策略：

- 80%的比例将其替换为[MASK]：例如my dog is hairy -> my dog is [MASK]
- 10%的比例用另一个随机词代替：例如my dog is hairy -> my dog is apple
- 其余10%的比例原词保持不变：例如my dog is hairy -> my dog is hairy

可见，这里的8:1:1是指对语句中15%被遮盖的词语的进一步划分。

Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. 2018. Improving language understanding with unsupervised learning. Technical report, OpenAI.

典型的预训练模型：BERT模型

（一）BERT的预训练阶段

2）下一语句预测任务 (Next Sentence Prediction)。

- 给定两个语句，判定二者在原文中是否为相邻关系，这是一个二分类预测任务。
- 下一语句预测预训练任务的具体实现方法是，选择若干组句子A与句子B构成的配对：其中50%情况下，B是A在原始语料中的下句；而另外50%情况下，B是从语料库中随机选择的语句。然后训练模型来预测二者之间的关系。

例如，下面两个配对分别为两种不同类型（IsNext、NotNext）的标签。

配对1：

Input = [CLS] the man went to [MASK] store [SEP] he bought a gallon [MASK] milk [SEP]

Label = *IsNext*

配对2：

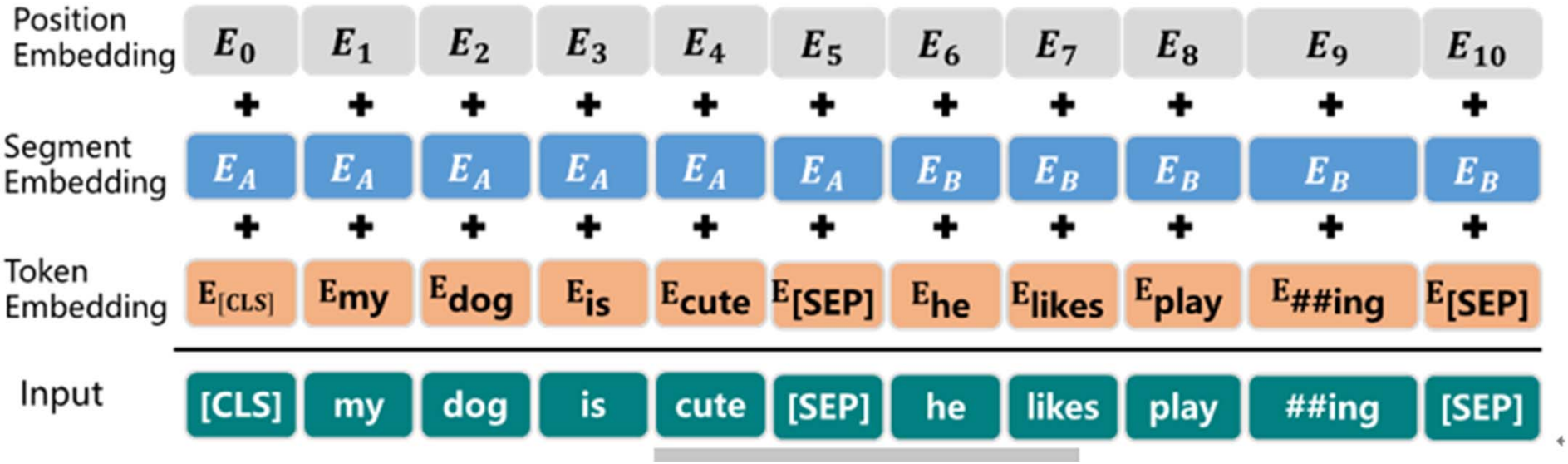
Input = [CLS] the man [MASK] to the store [SEP] penguin [MASK] are flight ##less birds [SEP]

Label = *NotNext*

BERT的预训练损失为上述遮盖语言模型任务和下一语句预测任务的损失的和。

典型的预训练模型：BERT模型

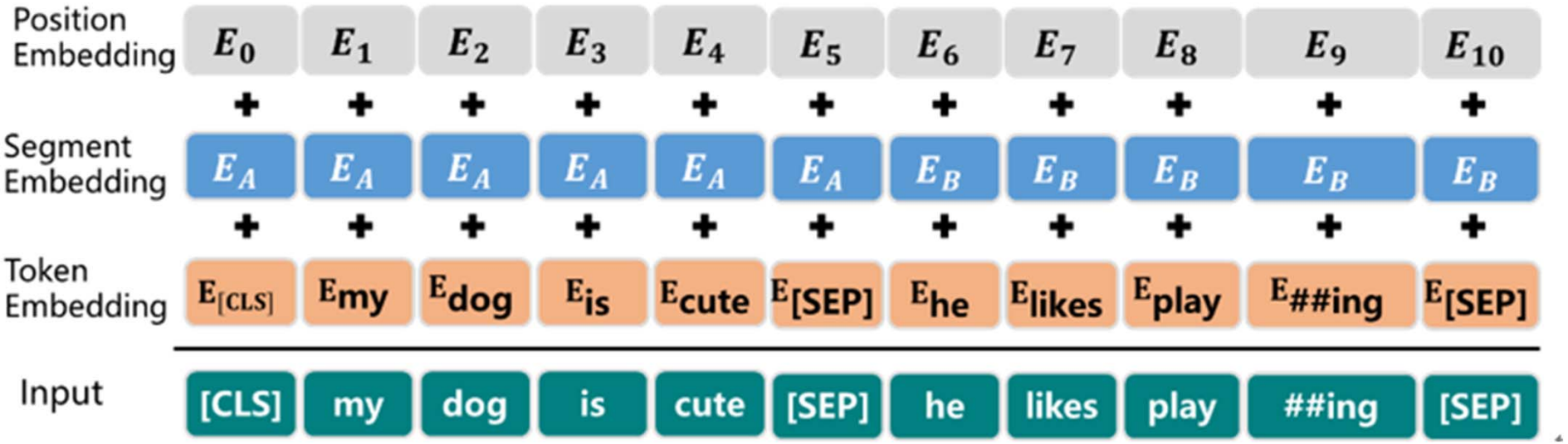
BERT中输入表示的构成



- 如果下游任务的输入（Input）是两个句子，例如[Question，Answer]，则需要使用分隔符“[SEP]”分隔后再连接在一起。
- 如果下游任务的输入只有一个句子，则不需要该分隔符。

典型的预训练模型：BERT模型

BERT中输入表示的构成

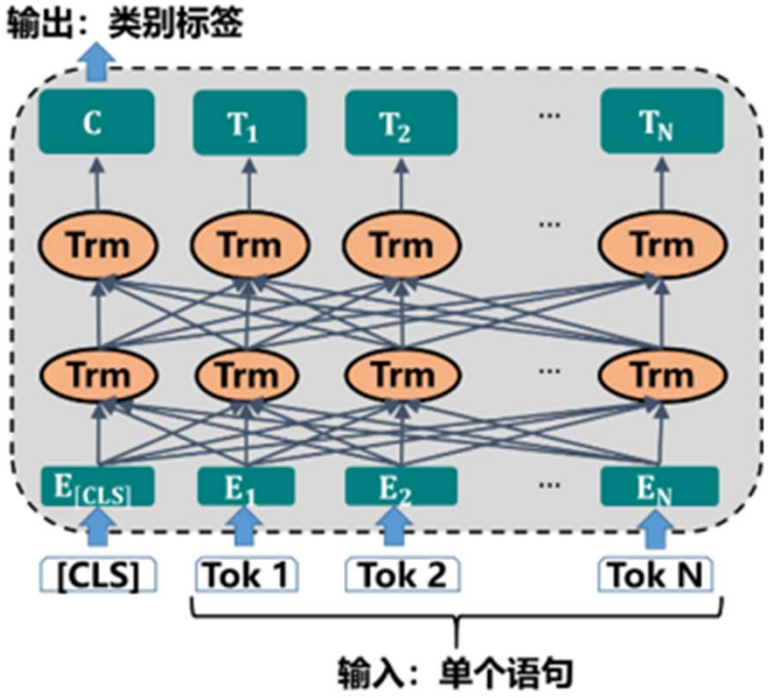


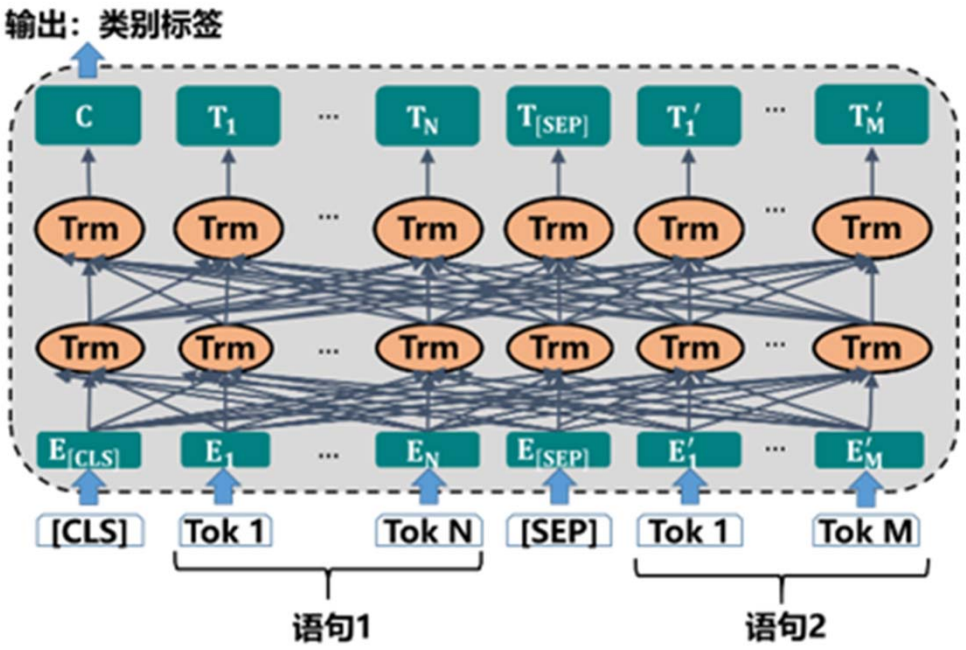
BERT在微调阶段，下游任务中输入的每个词语(token), **其输入嵌入是三部分的和**:

1) **词语嵌入(Token Embeddings)**。即每个输入词语的向量表示，包括两个特殊的词语符号，即开头的“CLS”标记、以及分隔符“[SEP]”；
2) **分段嵌入(Segmentation Embeddings)**。其用于区分词语的来源。
3) **位置嵌入(Position Embedding)**。每个词语向量加上不同的位置嵌入，以反映词语之间的不同位置和语序关系，BERT的位置嵌入向量是通过训练学习得到的。

典型的预训练模型：BERT模型

BERT微调阶段的下游任务类型及处理策略（序列级）





典型的预训练模型：BERT模型

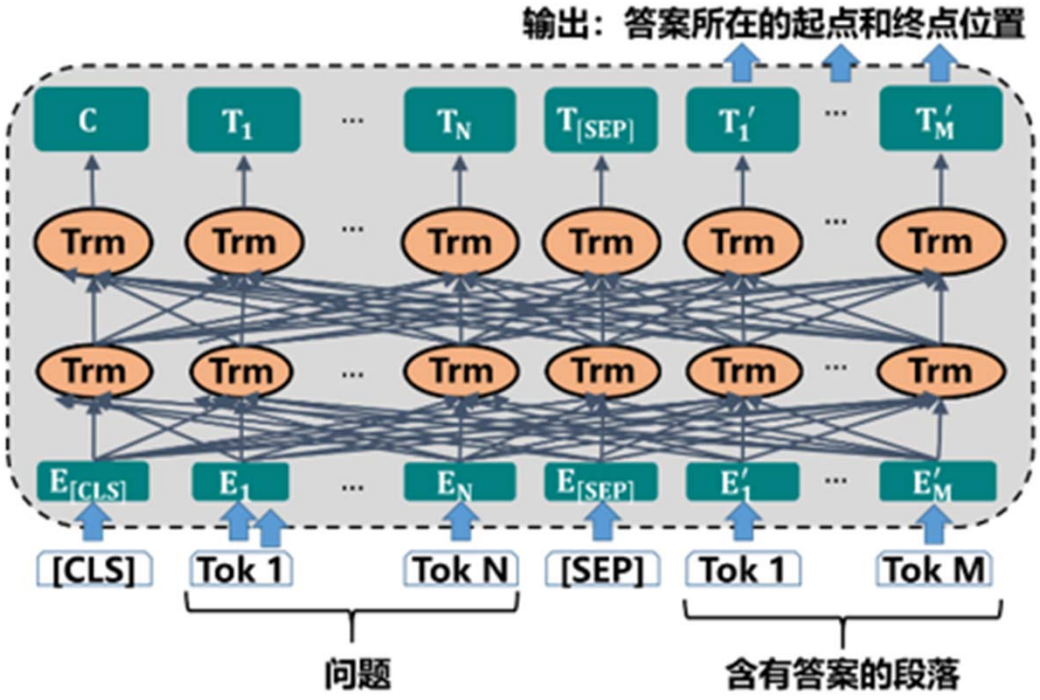
序列级分类任务的微调

- 对于序列级分类任务的微调，无论是单句分类，还是句对的分类，由于**都被视为一个字符串的分类问题**，二者的分类方法基本相同。
- 序列级分类任务微调训练阶段具体分类过程描述如下：

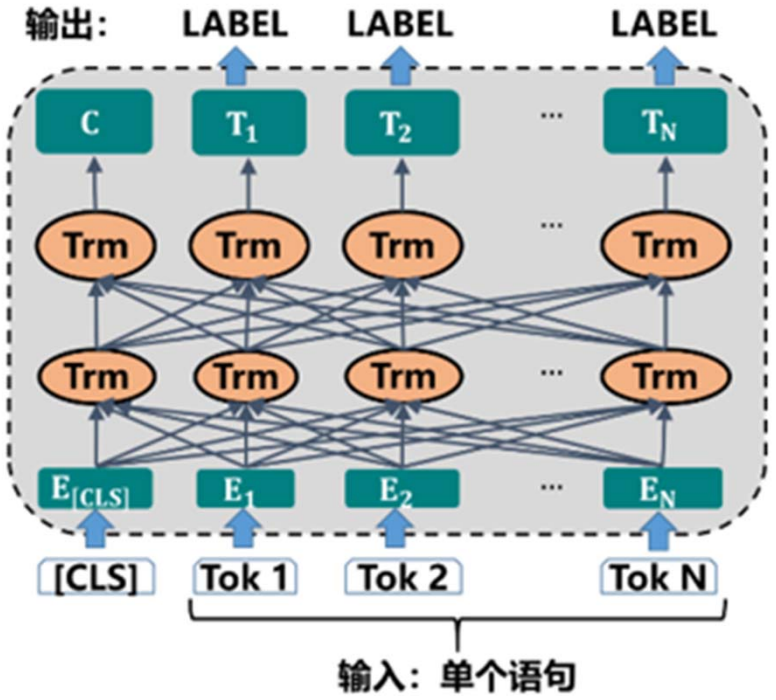
1) **输入层和中间各隐藏层使用预训练阶段的模型结构及预训练得到的网络参数取值**；
2) 设E表示微调阶段输入词语序列的嵌入，输入序列词语经过transformer多层变换得到最终的隐藏向量表示T_x。特别地，**设其中分类标记“[CLS]”得到的最终隐藏向量为C**；
3) **在隐藏向量C对应的单元上增加线性分类层。**因此在微调阶段唯一需要加的参数是就是该分类层的参数$W \in R^{K \times H}$，其中K是分类标签的数量。然后使用公式$P = softmax(CW^T)$计算输入序列的分类标签概率分布$P \in R^K$。
4) 此时，BERT的所有参数（即之前预训练阶段的参数）和W被联合起来微调训练，以便最大化正确标签的概率。

典型的预训练模型：BERT模型

BERT微调阶段的下游任务类型及处理策略（单个词条或字符级）



(a) 文本片段提取（例如 QA 任务）



(b) 词语打标签（例如 NER 任务）

典型的预训练模型：BERT模型

词语级分类任务的微调训练-文本片段提取任务描述

相对于序列级分类任务判断整个语句的类型，词语级分类任务的处理粒度相对较细。又可以进一步分为如下两种类型：

1）**文本片段提取。**其典型任务通常是问答系统，即给定问题预测输入段落中答案所在的文本片段，因此需要给出答案在段落中的开始和结束位置。例如，对于下面来自SQuAD数据集的例子。

Input Question: Where do water droplets collide with ice crystals to form precipitation?（水滴在哪里与冰晶碰撞形成降水？）

Input Paragraph: ... Precipitation forms as smaller droplets coalesce via collision with other rain drops or ice crystals within a cloud. ...（当较小的雨滴通过与云中的其他雨滴或冰晶碰撞而聚结时，就会形成降水）

Output Answer: within a cloud

典型的预训练模型：BERT模型

词语级分类任务的微调训练---文本片段提取任务的具体过程

结合上图，**BERT微调阶段对于文本片段提取任务**的具体过程描述如下：

① 初始化开始向量参数为$S \in R^H$，结束向量参数为$E \in R^H$。然后遍历输入段落（Input Paragraph）中的每个词语 i：

a) **计算第 i 个词语是答案开始的概率。**假设第 i 个词语的最终隐藏向量为$T_i \in R^H$。则该词语是答案开始的概率可以计算为 T_i 和 S 之间的点乘并做softmax处理：$\boldsymbol{P_i = \frac{e^{S \cdot T_i}}{\sum_j e^{S \cdot T_j}}}$。

b) **计算第 i 个词语是答案结束的概率。**计算原理同上。

② 根据上述结果计算最大得分跨度(the maximum scoring span)，进而得到预测的结果。

上述预测过程的训练目标是正确的开始和结束位置的对数似然。在推理时添加约束，即结束位置必须在开始位置之后。

典型的预训练模型：BERT模型

（二）词语级分类任务的微调训练--词语打标签任务

2）词语打标签任务（token tagging task）。

其比较典型的任务是命名实体识别（例如CoNLL 2003 NER）。如图（b）所示，其预测原理与前者类似，即输入序列中的每个词语 i 经过transformer多层变换得到最终的隐藏向量表示$T_i \in R^H$，并将其送给外接的分类层进行分类。

Alex Wang, Amapreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R Bowman. 2018. Glue: A multi-task benchmark and analysis platform for natural language understanding. arXiv preprint arXiv:1804.07461.

预训练模型

本章内容小结

- **什么是预训练**
  - 预训练与迁移学习
  - 计算机视觉领域的预训练
  - 自然语言处理领域的预训练
- **语言模型与预训练**
  - 什么是语言模型
  - 语言模型的实现方法：（一）N-gram语言模型
  - 语言模型的实现方法：（二）神经网络语言模型(NNLM)
  - 什么是词嵌入
  - Word2Vec之CBOW和Skip-gram
  - Word2vec的优化策略之一：Hierarchical Softmax
  - Word2vec的优化策略之二：Negative Sampling（负采样）
- **上下文词嵌入模型**
  - 静态词嵌入与上下文词嵌入
  - 典型的预训练模型：OpenAI GPT模型
  - 典型的预训练模型：BERT模型