24 注意力机制

传统编码器的困境：一个向量承载一切

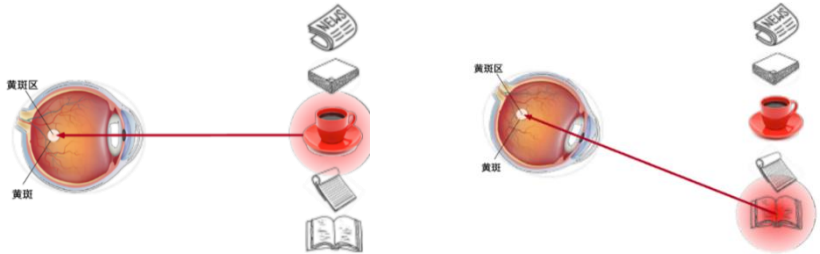
- 在早期的Seq2Seq模型（如RNN/LSTM）中，编码器（Encoder）必须将整个输入序列的所有信息压缩成一个固定长度的上下文向量 (Context Vector)
- 根本困境：
  - 信息丢失： 无论输入序列多长，都必须被压缩进同一个尺寸的向量，长序列的信息不可避免地会大量丢失
  - 无差别对待： 解码器（Decoder）在生成每个词时，都使用完全相同的上下文向量，无法动态关注输入序列的不同部分

- 我们需要一种机制，允许解码器在生成不同输出时，能够“有选择地”、“动态地”关注输入序列中最相关的部分

注意力机制

直觉来源：人类的视觉注意力

- 人类的注意力系统使我们能从复杂的视觉场景中高效地筛选出关键信息
- 两种线索驱动注意力
  - 不随意线索 (Bottom-up): 环境中某些物体因其自身特性（如颜色、形状）而“脱颖而出”，被动地吸引我们的注意
    - 输入数据中某些部分本身就具有较高的重要性
  - 随意线索 (Top-down): 基于我们的主观意图或任务目标，主动地将注意力集中在特定物体上
    - 我们根据当前的任务需求，去寻找特定的信息
- 选择机制例子
  - 咖啡杯，在视觉环境中突出和显眼，不由自主地引起人们的注意。所以视力放到最敏锐的地方咖啡上【不随意，非自主性提示 】
  - 喝咖啡后，会变得兴奋并想读书。所以聚焦眼睛选择看书。此时选择书是受到了认知和意识的控制，因此注意力在【随意，自主性提示 】去辅助选择

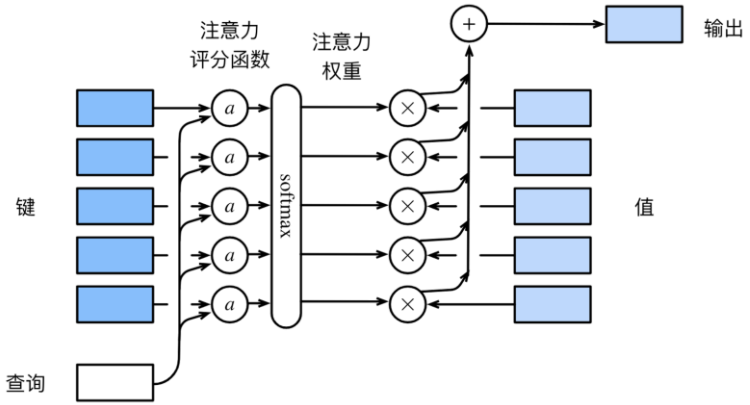


将直觉形式化：查询、键、值

- 注意力机制的核心可以被抽象为对信息的一次查询过程。三大核心组件
  - 查询 (Query, Q): 代表当前的任务或意图，即“我想找什么信息？” (对应随意线索)
  - 键 (Key, K): 代表信息源的“可供查询的标签或特征”，用于与Query进行匹配度计算 (对应不随意线索)
  - 值 (Value, V): 代表信息源的实际内容。Key和Value通常成对出现
- 工作流程
  - 计算相似度： 用Query和每个Key计算一个注意力分数（Attention Score）
  - 权重归一化： 将分数通过Softmax函数转换成概率分布，得到注意力权重（Attention Weights）

$$\alpha(\boldsymbol{q}, \boldsymbol{k}_i) = \mathrm{softmax}\big(a(\boldsymbol{q}, \boldsymbol{k}_i)\big) = \frac{\exp\big(a(\boldsymbol{q}, \boldsymbol{k}_i)\big)}{\sum_{j=1}^{m} \exp\Big(a\big(\boldsymbol{q}, \boldsymbol{k}_j\big)\Big)} \in \mathbb{R}$$

  - 加权求和： 用权重对所有的Value进行加权求和，得到最终的输出



注意力汇聚的数学表达

- 给定一个查询 q 和 n 个键值对 $(k_1, v_1), \ldots, (k_n, v_n)$，注意力汇聚函数 f 被定义为一个加权和

$$f(q, (k_1, v_1), \ldots, (k_n, v_n)) = \sum_{i=1}^{n} \alpha\,(q, k_i) v_i$$

- 注意力权重 $\alpha(q, k_i)$ 的计算
  - 评分函数 (Scoring Function) s： 计算查询 q 和键 k_i 的相关性得分
  - Softmax归一化： 保证所有权重之和为1，形成一个概率分布

$$\alpha(q, k_i) = \text{softmax}(s(q, k_i)) = \frac{\exp(s(q, k_i))}{\sum_{j=1}^{n} \exp\,(s(q, k_j))}$$

- 不同的注意力模型，其主要区别就在于评分函数 s 的设计

早期探索：非参数注意力

- 一个早期的、非参数化的注意力机制实例。它假设 $k_i = x_i$ (输入)，$v_i = y_i$ (输出)
- 使用高斯核 (Gaussian Kernel) 作为评分函数

$$s(q, k_i) = -\frac{1}{2}(q - k_i)^2$$

- 查询 q 与键 k_i 的距离越近，得分越高
- 完整的N-W回归公式：

$$f(q) = \sum_{i=1}^{n} \frac{\exp(-\frac{1}{2}(q - x_i)^2)}{\sum_{j=1}^{n} \exp\left(-\frac{1}{2}(q - x_j)^2\right)} y_i$$

  - 简单，无需学习参数
  - 权重完全由输入数据的位置决定，模型无法学习到更复杂的、与任务相关的注意力模式

早期探索： Nadaraya-Watson核回归

- 使用高斯核 $K(u) = \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{u^2}{2}\right)$
- 那么

$$
\begin{aligned}
f(x) &= \sum_{i=1}^{n} \frac{\exp\left(-\frac{1}{2}(x - x_i)^2\right)}{\sum_{j=1}^{n} \exp\left(-\frac{1}{2}(x - x_j)^2\right)} y_i \\
&= \sum_{i=1}^{n} \operatorname{softmax}\left(-\frac{1}{2}(x - x_i)^2\right) y_i
\end{aligned}
$$

  - 没有需要学习的东西
- softmax形式的注意力权重
  - 权重和为1

早期探索：参数化的注意力机制

- 带参数的注意力权重
- 高斯核引入可以学习的 w

$$f(x) = \sum_{i=1}^{n} \text{softmax}\left(-\frac{1}{2}\left((x - x_i)w\right)^2\right) y_i$$

- 如果一个键x_i越是接近给定的查询x，那么分配给这个键对应值y_i的注意力权重就会越大，也就“获得了更多的注意力”，得到更精确的预测函数
- 非参数的注意力汇聚（non- parametric attention pooling）模型

关键一步：可学习的参数化注意力

- 固定的评分函数限制了模型的表达能力。我们希望模型能根据训练数据，自动学习出如何计算注意力分数。两种主流的参数化评分函数：
  - 加性注意力 (Additive Attention / Bahdanau Attention):

$$s(q, k_i) = w_v^T \tanh(W_q q + W_k k_i)$$

    - W_q, W_k, w_v 都是可学习的参数矩阵/向量。
    - 将 q 和 k_i 投影到同一空间后相加，通过一个单层MLP计算分数。灵活，尤其在Q和K维度不同时
  - 缩放点积注意力 (Scaled Dot-Product Attention):

$$s(q, k_i) = \frac{q^T k_i}{\sqrt{d_k}}$$

    - d_k 是键向量的维度
    - 直接计算 q 和 k_i的点积。计算效率极高，但要求Q和K维度相同。除以 $\sqrt{d_k}$是为了防止点积结果过大导致Softmax梯度消失

第一次飞跃：注意力机制在Seq2seq中的应用

- 传统Seq2seq模型依赖一个固定的上下文向量 c 来连接编码器和解码器，导致信息瓶颈
- 新的解决方案：在解码的每一步，都动态地生成一个专属的上下文向量 c_t
  - 1．在解码第 t 步时，解码器当前的隐状态 h_t^{dec}（它包含了已生成序列的信息）被用作Query
  - 2．这个Query会去和编码器所有时间步的隐状态 $\{h_1^{\text{enc}}, h_2^{\text{enc}}, \ldots, h_n^{\text{enc}}\}$（它们被同时用作Key和Value）进行匹配
  - 3．通常使用加性注意力来计算注意力权重 α_{ti}，即 h_t^{dec} 对每个 h_i^{enc} 的关注程度。
  - 4．最终的上下文向量 c_t 是编码器所有隐状态的加权和：$c_t = \sum_{i=1}^{n} \alpha_{ti} h_i^{\text{enc}}$
  - 5．这个动态的 c_t 与解码器隐状态 h_t^{dec} 一起，被用来预测下一个词元 y_t
- 核心价值
  - 解码器可以直接＂看到＂并访问输入序列的每一个部分，不再受限于固定长度的向量
  - 动态聚焦：在翻译句子的不同部分时，模型可以自动将注意力集中在源句的不同关键词上，实现了类似人类翻译的＂对齐＂过程
  - 异源注意力（Cross-Attention）的经典应用，Query和Key/Value来自不同的源头（解码器/编码器）

范式转变：自注意力机制 (Self-Attention)

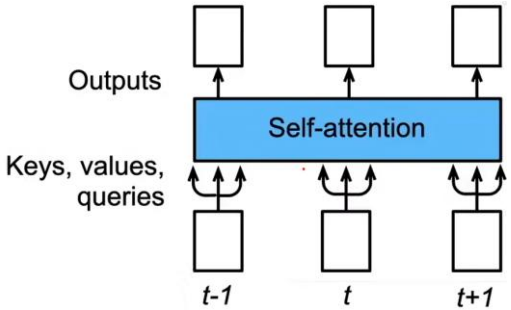
- 默认的Encoder-Decoder场景：Query来自解码器，Key和Value来自编码器
- 如果Query, Key, Value都源自输入序列自身，会发生什么？
  - $Q = XW_Q, K = XW_K, V = XW_V$
  - 其中 X 是输入序列的嵌入矩阵，W_Q, W_K, W_V 是可学习的投影矩阵
- 注意力不再是连接Encoder和Decoder，而是建模输入序列内部的依赖关系
  - 序列中的每个词元（token）都会生成一个Query，去和序列中所有其他词元的Key计算相关性，然后汇聚所有词元的Value
  - 这使得模型可以直接捕捉序列中任意两个位置之间的长距离依赖
- 优势：
  - 并行计算： 对序列中所有位置的计算可以完全并行，相比RNN的顺序计算，效率极大提升
  - 最短路径： 任意两个位置之间的信息交互路径长度为 $O(1)$，RNN的 $O(n)$ 和CNN的 $O(\log n)$

自注意力

- 给定序列 $\mathbf{x}_1, \ldots, \mathbf{x}_n, \forall \mathbf{x}_i \in \mathbb{R}^d$，自注意力池化层将 $\mathbf{x}_i$ 当做key, value, query来对序列抽取特征得到 $\mathbf{y}_1, \ldots, \mathbf{y}_n$, 这里

$$\mathbf{y}_i = f\left(\mathbf{x}_i, (\mathbf{x}_1, \mathbf{x}_1), \ldots, (\mathbf{x}_n, \mathbf{x}_n)\right) \in \mathbb{R}^d$$

- 只从自身数据中提取特征【而不依赖其它的随意线索】
  - 注意，(Q,K)定义需要预先给出
- 使用自注意力进行序列编码，不再需要Encoder-Decoder



位置编码

- 自注意力的“缺陷”： 自注意力机制本身是置换不变的 (Permutation Invariant)。即，打乱输入序列的顺序，输出结果完全相同。它无法感知词元的绝对或相对位置
- 在输入嵌入中注入位置信息

$$\text{InputEmbedding}_{\text{final}} = \text{TokenEmbedding} + \text{PositionalEncoding}$$

- 输入表示 $\mathbf{X} \in \mathbb{R}^{n \times d}$，一个序列中$n$个词元的$d$维嵌入表示；位置编码相同形状的位置嵌入矩阵$\mathbf{P} \in \mathbb{R}^{n \times d}$，矩阵第$i$行、第$2j$列和$2j+1$列元素

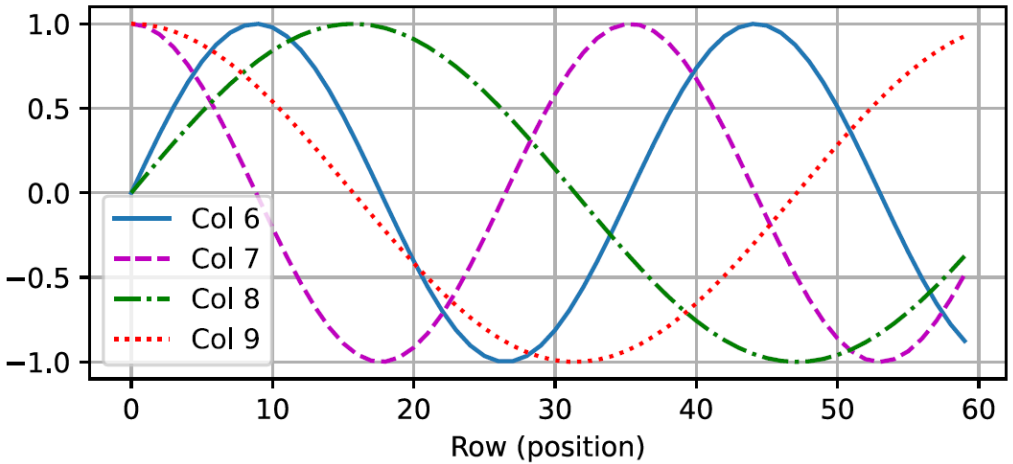
$$p_{i,2j} = \sin\left(\frac{i}{10000^{2j/d}}\right)$$

$$p_{i,2j+1} = \cos\left(\frac{i}{10000^{2j/d}}\right)$$

- 位置编码通过使用三角函数在编码维度上降低频率
  - 由于输出是浮点数，因此此类连续表示比二进制表示法更节省空间
- 几维就几个三角函数！

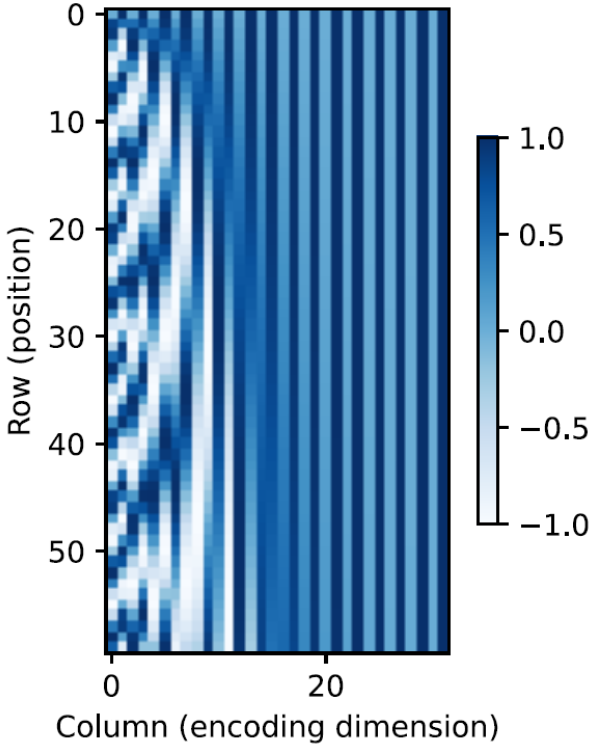
位置编码矩阵

- $\mathbf{P} \in \mathbb{R}^{n \times d}$, $p_{i,2j} = \sin\left(\frac{i}{10000^{2j/d}}\right)$, $p_{i,2j+1} = \cos\left(\frac{i}{10000^{2j/d}}\right)$



位置编码矩阵

- 每一列【不同周期】
  - 列越大，变化越平缓
- 每一行
  - 没有直观意义



相对位置信息

- 位置于 $i+\delta$ 处的位置编码可以线性投影位置 i 处的位置编码来表示
- 记 $\omega_j = 1/10000^{2j/d}$, 那么

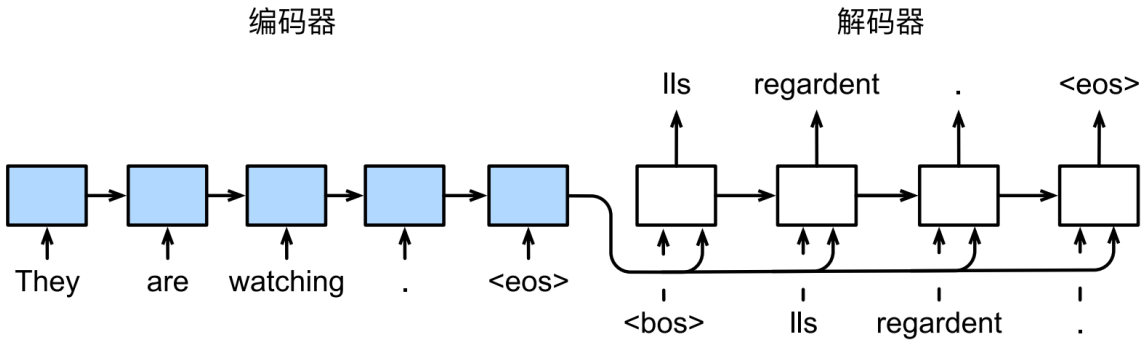
$$\begin{bmatrix} \cos(\delta\omega_j) & \sin(\delta\omega_j) \\ -\sin(\delta\omega_j) & \cos(\delta\omega_j) \end{bmatrix} \begin{bmatrix} p_{i,2j} \\ p_{i,2j+1} \end{bmatrix} = \begin{bmatrix} p_{i+\delta,2j} \\ p_{i+\delta,2j+1} \end{bmatrix}$$

- 编码的是相对位置： 2×2 投影矩阵不依赖于任何位置的索引i
- 对于任何确定的位置偏移 δ, 任何一对 $(p_{i,2j}, p_{i,2j+1})$ 都可以线性投影到 $(p_{i+\delta,2j}, p_{i+\delta,2j+1})$

结合注意力机制的Seq2seq

传统RNN Seq2seq模型

- 循环神经网络编码器将长度可变的序列转换为固定形状的上下文变量，然后循环神经网络解码器根据生成的词元和上下文变量按词元生成输出（目标）序列词元
- 即使并非所有输入（源）词元都对解码某个词元都有用，在每个解码步骤中仍使用编码相同的上下文变量
  - 如何改变上下文变量？



编码器解码器

- 编码器输出的上下文变量c 对整个输入序列 $x_1, \ldots, x_T$ 进行编码
- 来自训练数据集的输出序列$y_1, y_2, \ldots, y_{T'}$
- 对于每个时间步 t' (与输入序列或编码器的时间步 t 不同)，解码器输出 $y_{t'}$ 的概率取决于先前的输出子序列 $y_1, \ldots, y_{t'-1}$ 和上下文变量 $\mathbf{c}$ ，即$P(y_{t'} \mid y_1, \ldots, y_{t'-1}, \mathbf{c})$
- 循环神经网络将来自上一时间步的输出 $y_{t'-1}$ 和上下文变量 $\mathbf{c}$ 作为其输入，然后在当前时间步将它们 和上一隐状态 $\mathbf{s}_{t'-1}$ 转换为隐状态 $\mathbf{s}_{t'}$

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

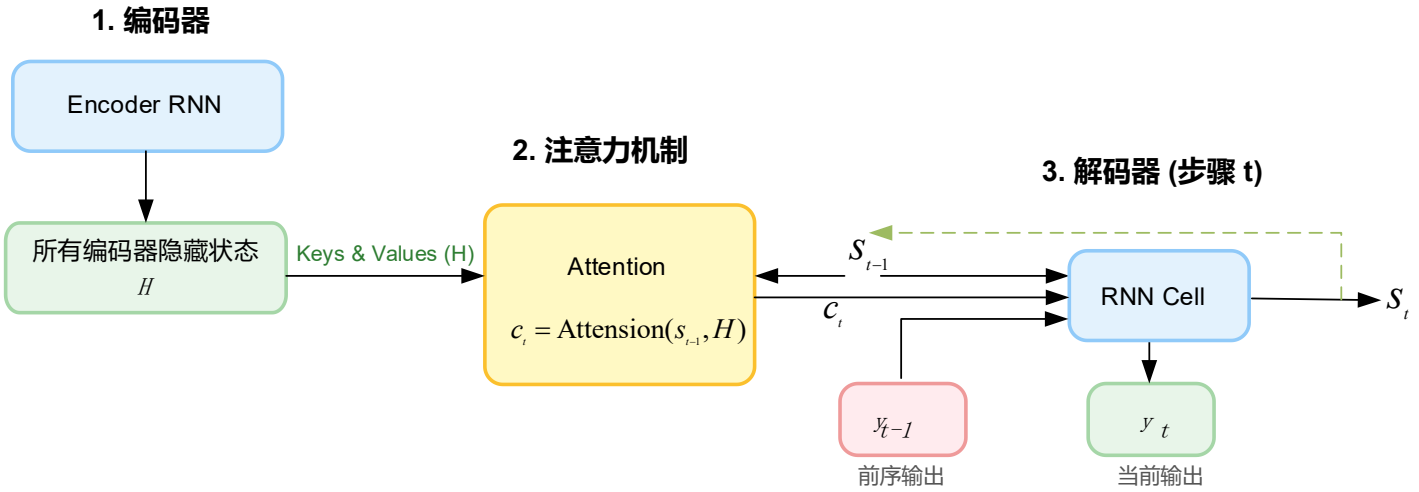
- 在获得解码器的隐状态之后，可以使用输出层和 softmax操作来计算在时间步 t' 时输出 $y_{t'}$ 的条件概率分 布 $P(y_{t'} \mid y_1, \ldots, y_{t'-1}, \mathbf{c})$

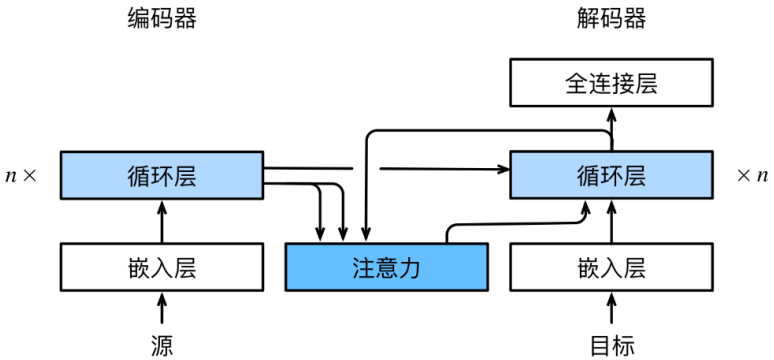
结合注意力的seq2seq

- 编码器（Encoder）
  - 编码器（通常是一个双向RNN/LSTM/GRU）处理整个输入序列，且保留编码器在每个时间步的所有隐藏状态。这些隐藏状态集合 H 包含了输入序列在不同位置的丰富局部信息，而不仅仅是最后一个状态
- 解码器与注意力（Decoder with Attention）
  - 解码器在生成输出序列 的每一步t，都会执行一个“注意-生成”循环：
    - a. 计算注意力权重: 基于解码器前一时刻的隐藏状态s_{t-1}，去和编码器的所有隐藏状态 H 进行比较，计算出一个“对齐分数”或“相关性分数”
    - b. 归一化: 将这些分数通过Softmax函数，转换成一组权重，即注意力分布
    - c. 计算上下文向量: 使用这组权重对编码器的所有隐藏状态 H 进行加权求和，得到该时刻 t 专属的上下文向量c_t
    - d. 生成输出: 将 c_t与解码器当前状态 s_{t-1}和前一时刻的输出 y_{t-1}结合，共同预测当前时刻的输出 y_t ，并更新解码器状态为 s_t

结合注意力的seq2seq

- 工作流





构建模块：Transformer 编码器块

Transformer 编码器块

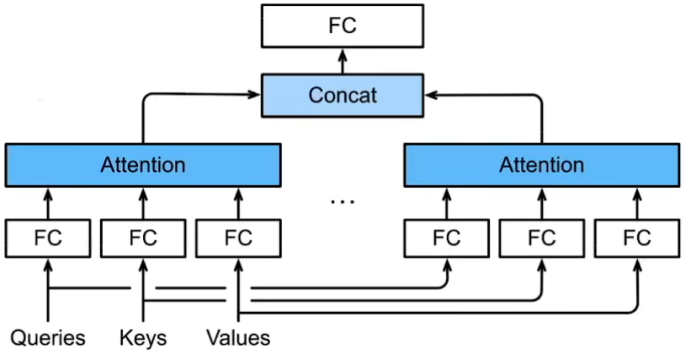
- Transformer的由多个相同的“块”(Block)堆叠而成。一个标准的编码器块包含两个核心子层
  - 多头自注意力层 (Multi-Head Self-Attention Layer)
  - 逐位置前馈网络 (Position-wise Feed-Forward Network)
- 连接方式： 每个子层都采用了残差连接 (Residual Connection) 和 层归一化 (Layer Normalization)
  - SublayerOutput = LayerNorm(x + Sublayer(x))
- 作用：
  - 残差连接： 缓解深度网络中的梯度消失问题，让信息和梯度能更顺畅地跨层流动。
  - 层归一化： 稳定每层输入的分布，加速模型训练。它对每个样本在特征维度上进行归一化，与批次大小无关，非常适合NLP任务

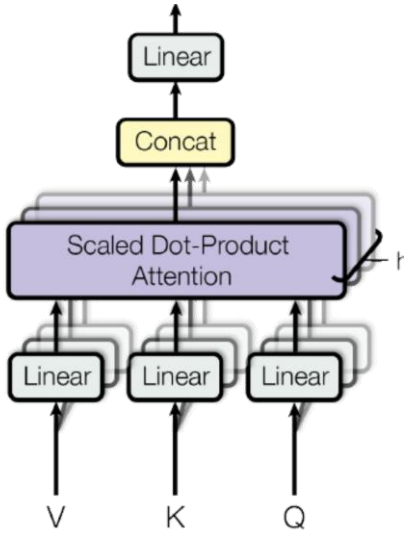
多头注意力机制(Multi-head Attention)

- 与其用一个大的注意力头（d_{model}维度）一次性计算注意力，不如将Q, K, V线性投影到多个低维子空间，在每个子空间上并行计算注意力，最后再将结果拼接起来
  - 投影： 将Q, K, V分别通过 h 个不同的线性层，投影成 h 组低维的 q_i, k_i, v_i
  - 并行注意力： 对每一组 (q_i, k_i, v_i) 并行执行缩放点积注意力，得到 h 个输出 $head_i$

$$head_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

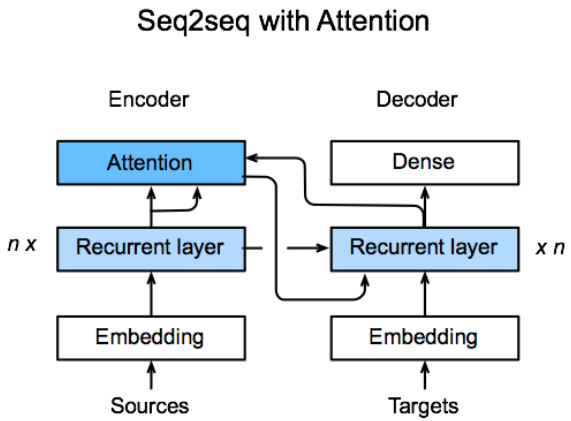
  - 拼接与再投影： 将 h 个头的输出拼接起来，再通过一个线性层进行融合
- 允许模型在不同表示子空间中共同关注来自不同位置的信息

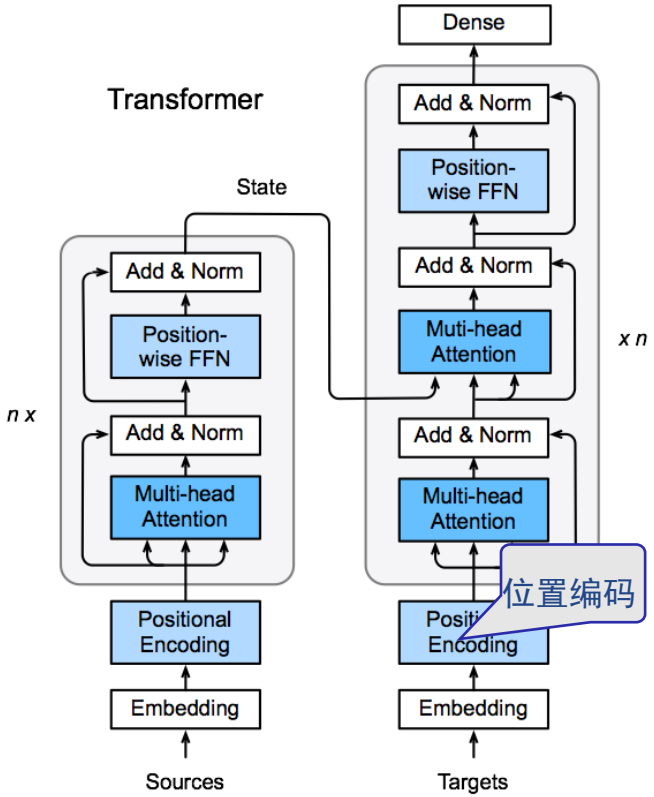




Transformer模型架构

- 编码器 - 解码器架构
- 与seq2seq不同，注意力集中在3个地方



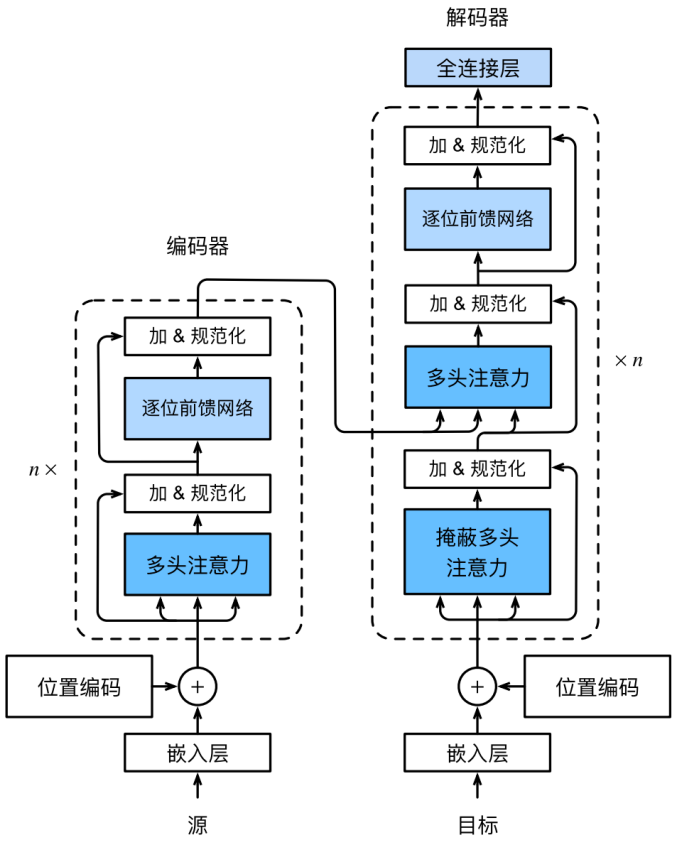


Transformer模型架构

- Transformer模型遵循经典的编码器-解码器架构，但内部完全由注意力机制构成
- 编码器 (Encoder)
  - 由N个相同的编码器块堆叠而成
  - 负责将输入序列 $(x_1, \ldots, x_n)$ 编码成一系列上下文表示 $(z_1, \ldots, z_n)$
  - 内部使用多头自注意力
- 解码器 (Decoder):
  - 由N个相同的解码器块堆叠而成
  - 在每个时间步，根据编码器的输出和已生成的部分序列，预测下一个词元
  - 内部使用两种注意力
    - 带掩码的多头自注意力 (Masked Multi-Head Self-Attention): 在解码时，当前位置只能关注到它之前的位置，不能“看到”未来的词元
    - 编码器-解码器注意力 (Encoder-Decoder Attention): Query来自解码器，而Key和Value来自编码器的最终输出。这是连接编码器和解码器的桥梁

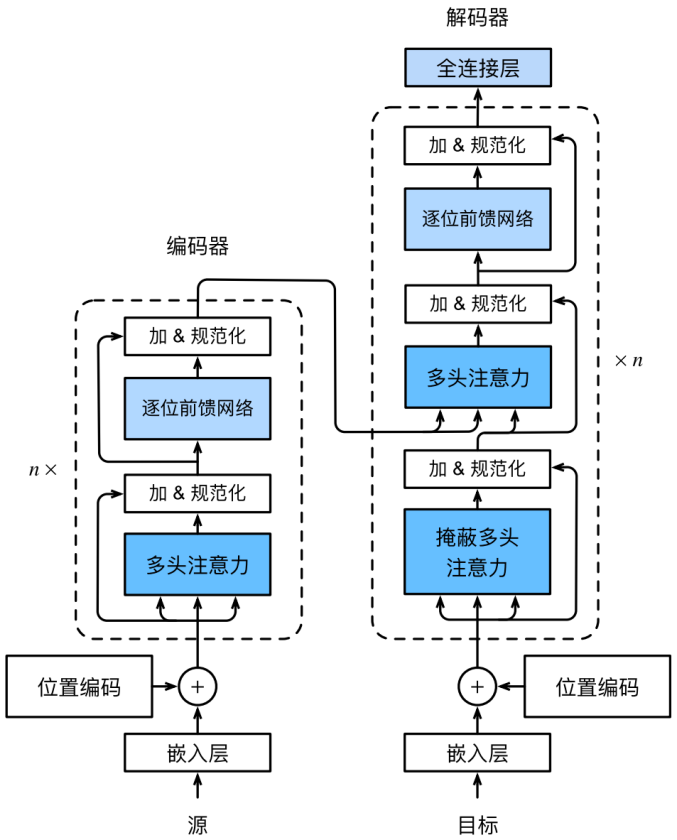
有掩码的多头注意力

- 解码器对序列中一个元素输出时，不应该考虑该元素之后的元素
- 可以通过掩码来实现
  - 计算 $\mathbf{x}_i$ 输出时，当前序列长度设置为 i
  - 过滤掉超出指定范围的位置



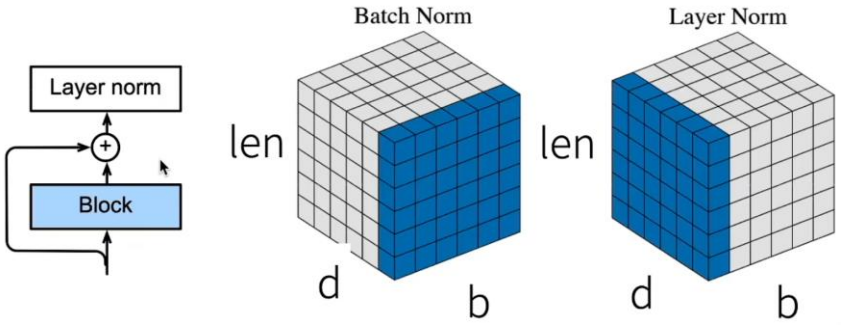
基于位置的前馈网络

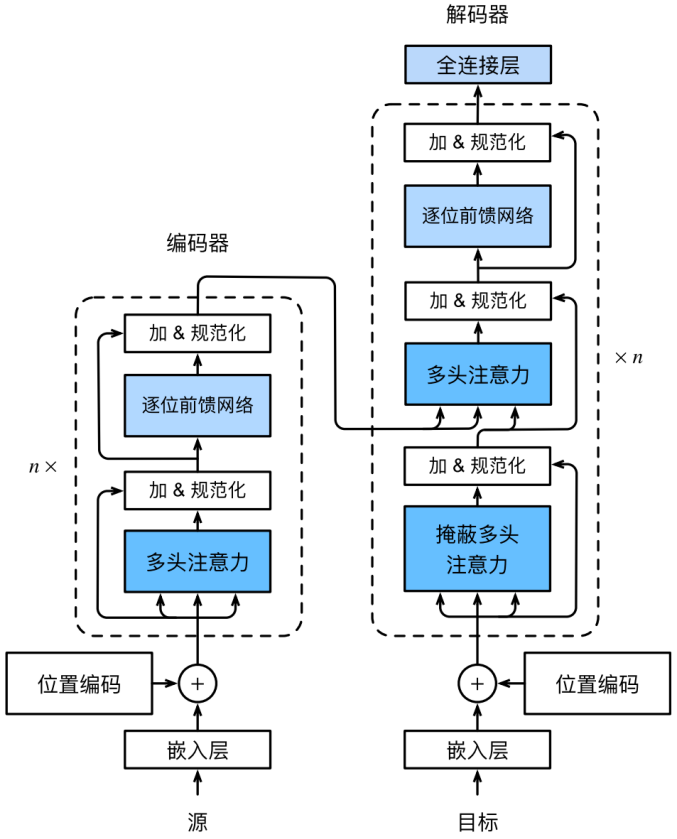
- 将将输入形状由 (b, n, d) 变换成 (bn, d)
  - 输入（批量大小，时间步数或序列长度，隐单元数或特征维度）
  - 整形为(批量大小 *序列长度，特征维度)
- 用两层 MLP
- 输出形状由 (bn, d) 变化回 (b, n, d)
  - 批量大小，时间步数， ffn_num_outputs
- Position-wise Feed-Forward Networks
  - FFN
  - 基于位置的前馈网络



添加层归一化

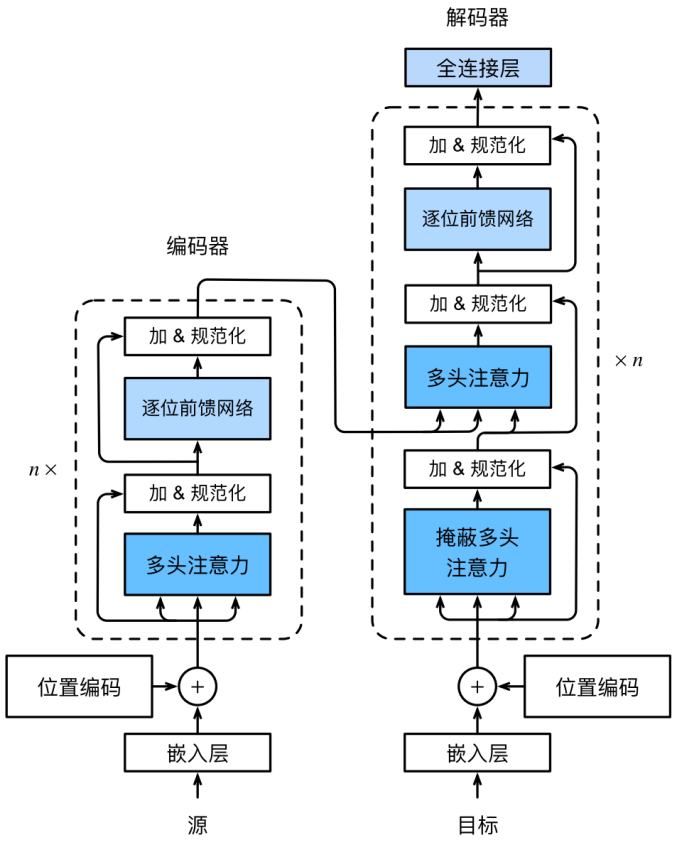
- 批量归一化
  - 对每个特征/通道里元素进行归一化
  - 不适合序列长度会变的NLP应用
- 层归一化
  - 对网络每一层内部的数据进行标准化
  - 对于每个神经元的输入值，减去该层的均值并除以其标准差





信息传递

- 编码器中的输出 $\mathbf{y}_1, \ldots, \mathbf{y}_n$
- 将其作为解码中第 i 个Transformer块中多头注意力的key和value
  - query来自前一个解码器层的输出
- 编码器和解码器中块的个数和输出维度都是一样的



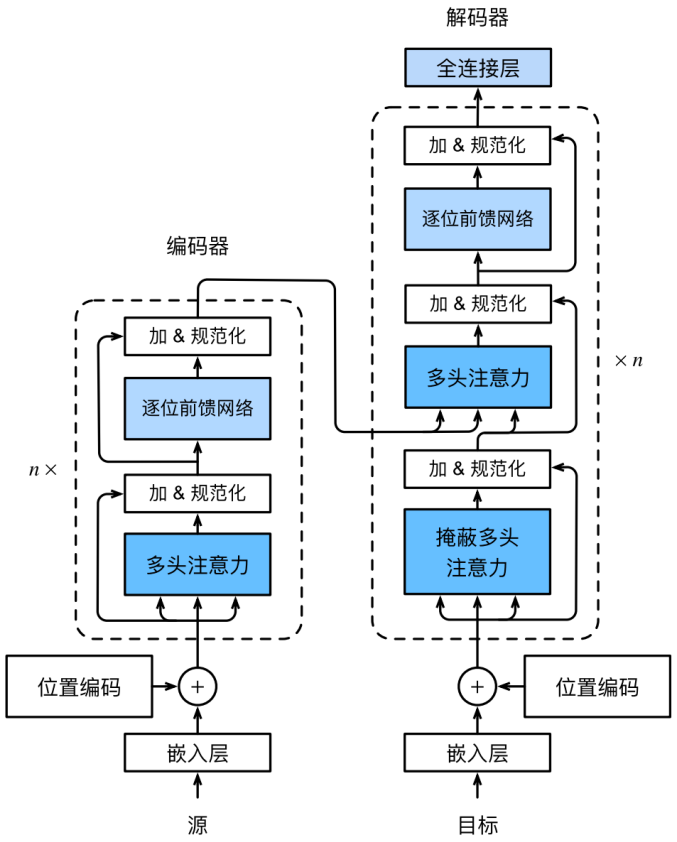
位置编码

- 假设嵌入 $X \in \mathbb{R}^{l \times d}$ 输出的形状(序列长度，嵌入维度)
- 创建 $P \in \mathbb{R}^{l \times d}$

$$P_{i,2j} = \sin(\frac{i}{10000^{2j/d}})$$

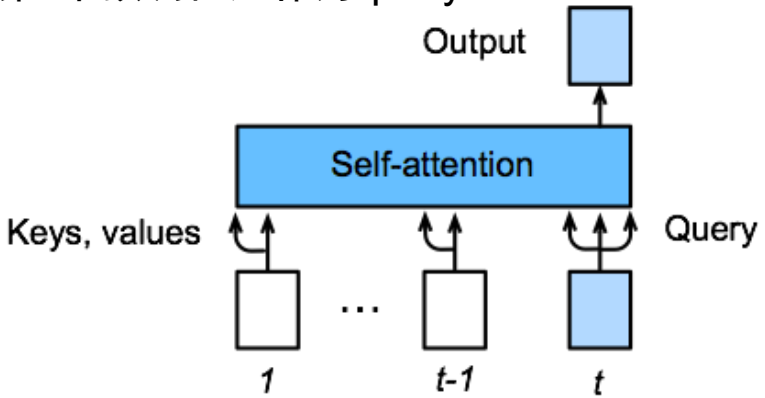
$$P_{i,2j+1} = \cos(\frac{i}{10000^{2j/d}})$$

- 输出$X + P$



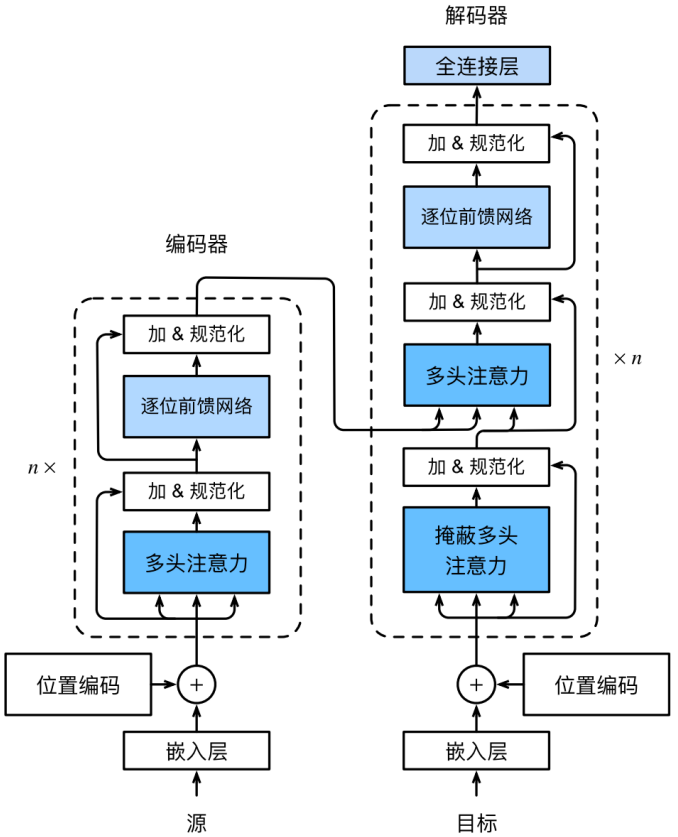
预测

- 预测第 $t+1$ 个输出时
- 解码器中输入前t个预测值
  - 在自注意力中, 前 t 个预测值作为key和value，第t个预测值还作为query



总结

- Transformer是一个纯使用注意力的编码-解码器
- 编码器和解码器都有n个 transformer块
- 每个块使用多头（自）注意力, 基于位置的前馈



BERT: Bidirectional Encoder Representations from Transformers

从零训练到“预训练-微调”范式

- 像Word2Vec/GloVe这样的词嵌入是上下文无关的。ELMo和GPT等模型开启了“上下文敏感”的预训练新时代
  - ELMo: 使用双向LSTM，但架构是任务特定的
  - GPT: 使用Transformer解码器，是任务无关的，但其自回归特性使其只能看到单向（从左到右）的上下文

- 能否创建一个模型，既是任务无关的，又能编码深度双向的上下文？
  - 仅使用Transformer的编码器部分，并通过巧妙的预训练任务来强制模型学习双向表示

BERT - 基于Transformer编码器的双向语言表示

- 一个堆叠了多层Transformer编码器块的深度网络。
- 关键输入表示：
  - 特殊词元： [CLS] 用于句子（对）级别的分类任务，[SEP] 用于分隔两个句子
  - 三种嵌入相加：
    - 词元嵌入 (Token Embeddings): 词本身的表示
    - 片段嵌入 (Segment Embeddings): 用于区分句子A和句子B
    - 位置嵌入 (Position Embeddings): 与Transformer不同，BERT使用可学习的位置嵌入
- 核心产出： 对于输入的每个词元，BERT都能输出一个融合了其左右双向、深度上下文信息的向量表示

如何让模型“理解”语言？

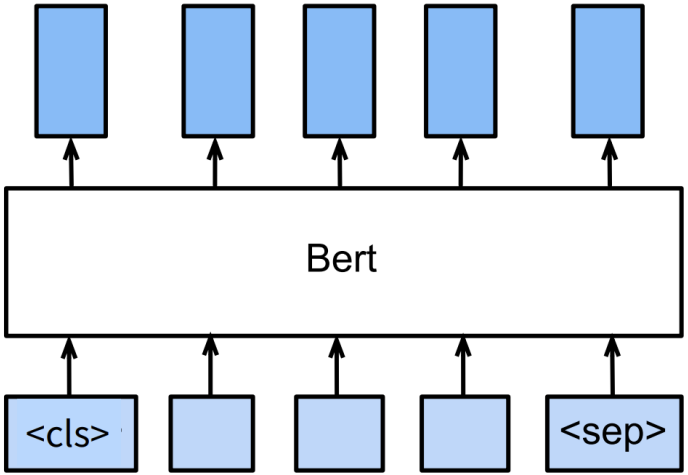
- 掩码语言模型 (Masked Language Model, MLM)
  - 随机遮盖（mask）输入序列中15%的词元，然后让模型预测这些被遮盖的词元是什么
  - 迫使模型必须依赖被遮盖词元的双向上下文来进行预测，从而学习到深度的双向语境表示。这打破了传统语言模型只能单向预测的限制
  - 细节： 80%时间用[MASK]替换，10%用随机词替换，10%保持不变，以减少预训练和微调阶段的不匹配
- 下一句预测 (Next Sentence Prediction, NSP)
  - 给定句子对(A, B)，判断B是否是A的真实下一句。这是一个二分类任务
  - 让模型学习句子间的关系，这对于问答（QA）、自然语言推理（NLI）等任务至关重要
  - 实现： 将[CLS]词元对应的输出向量送入一个分类器进行预测

BERT的应用：强大的下游任务微调

- 预训练好的BERT模型可以通过在其之上添加一个简单的输出层，来适应各种下游任务，并只需对整个模型进行少量轮次的微调（Fine-tuning）
- 典型应用示例
  - 句子（对）分类
    - 任务： 情感分析、自然语言推理
    - 方法： 将[CLS]词元对应的输出向量送入一个全连接分类器
  - 序列标注
    - 任务： 命名实体识别（NER）
    - 方法： 将每个词元对应的输出向量分别送入一个分类器，预测其标签（如PER, LOC, ORG）
  - 问答任务
    - 任务： 在给定段落中找到问题的答案（SQuAD）
    - 方法： 对段落中的每个词元，预测它是答案“起始位置”和“结束位置”的概率

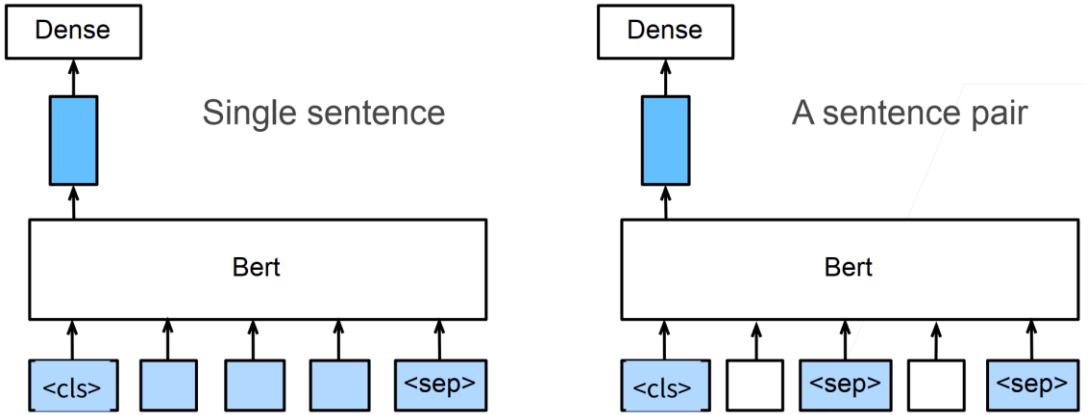
用BERT微调

- BERT为每一个词元返回抽取了上下文信息的特征向量
- 不同的微调任务使用不同的特征



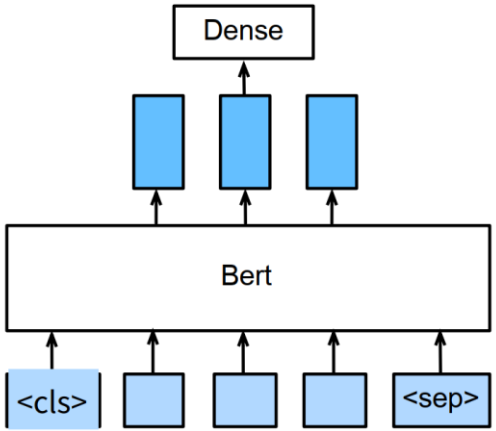
句子分类

- 如单一句子的分类
  - 将<cls>对应的向量，输入到全连接层分类



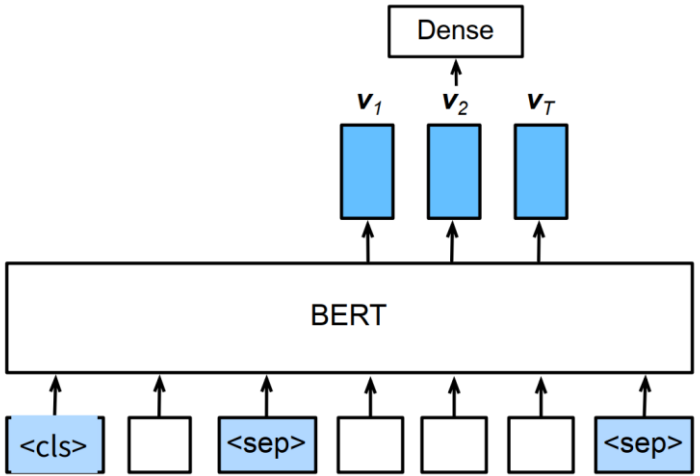
命名实体识别

- 识别一个词元是否为命名实体，分类
  - 如人员，组织和位置等
- 将非特殊词元放进全连接层分类
  - 如去掉<cls><sep>

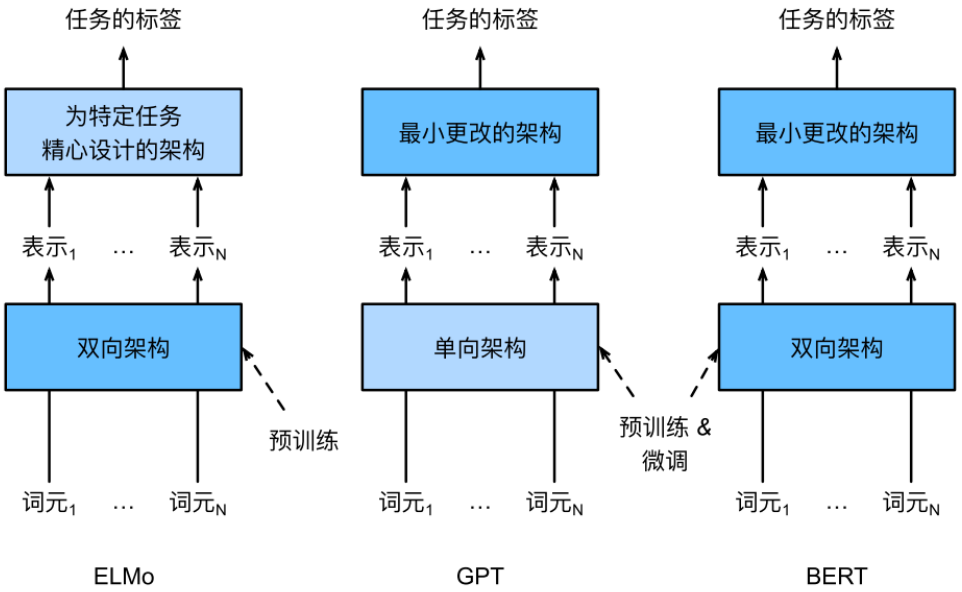


问题回答

- 给定问题和描述文字，找出一个片段作为回答
- 对片段中的每个词元预测它是否为回答的开头或结束
  - 三分类：开头，结尾，其它



ELMo、 GPT和BERT之间的差异



总结

- BERT针对微调设计
- 基于Transformer的编码器做了如下修改
  - 模型更大, 训练数据更多
  - 输入句子对, 片段嵌入, 可学习的位置编码
  - 训练时使用两个任务
    - 带掩码的语言模型
    - 下一个句子预测

注意力机制：从辅助工具到核心引擎

- 演进回顾：
  - 注意力机制最初是为了解决Seq2Seq模型的信息瓶颈而生。
  - 自注意力的提出，使其从一个连接模块变成了强大的序列内部关系建模工具，并因其并行性和短路径优势而脱颖而出。
  - Transformer将自注意力机制的能力发挥到极致，成为现代NLP架构的基石。
  - BERT等预训练模型，利用注意力机制学习通用的、深度的双向语言表示，彻底改变了NLP领域的研究和应用范式。
- 核心洞见： 注意力机制的本质是一种动态、可微的寻址和信息加权机制。正是这种灵活性和强大的表示能力，使其成为当前深度学习，尤其是大语言模型（LLMs）中不可或缺的核心组件。