Deep Learning I

11. Transformer

WU Xiaokun 吴晓堃

xkun.wu [at] gmail

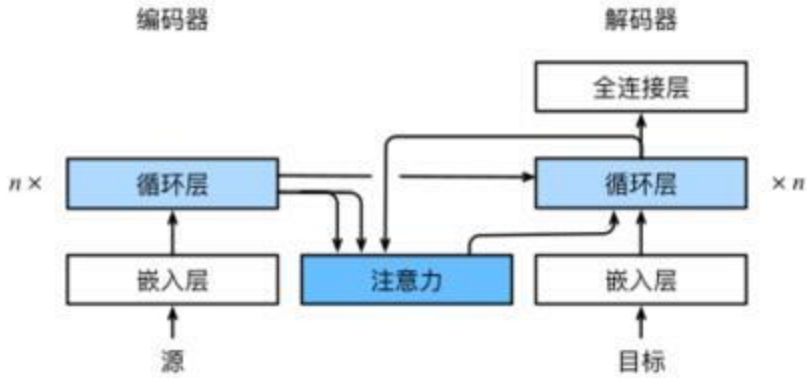
2022/02/28

多头注意力

回顾：注意力架构

Bahdanau 注意力：编码器-解码器架构

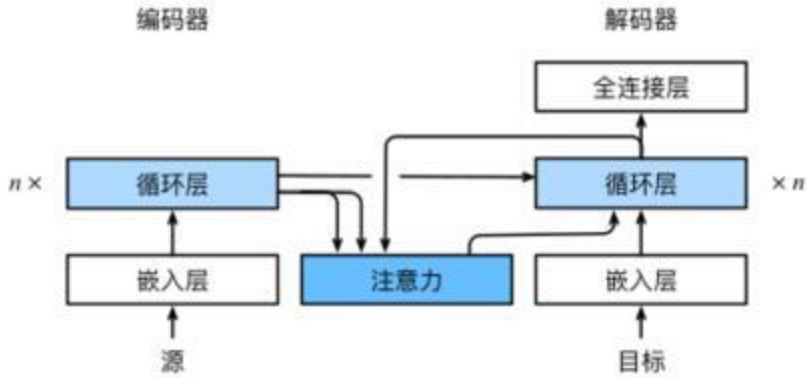
- 解码器上一时间步的隐状态用于**查询**
- 编码器所有时间步的隐状态用作**键、值**



回顾：注意力架构

Bahdanau 注意力：编码器-解码器架构

- 解码器上一时间步的隐状态用于**查询**
- 编码器所有时间步的隐状态用作**键、值**

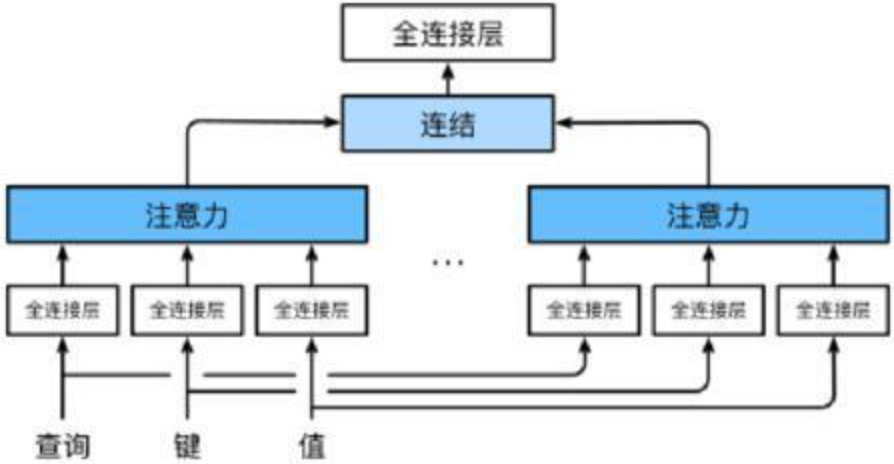


只有一个注意力模块：单一架构只能学习**固定模式**

- 表达能力有限，例如长、短距离依赖的学习模式：理论上应该不同

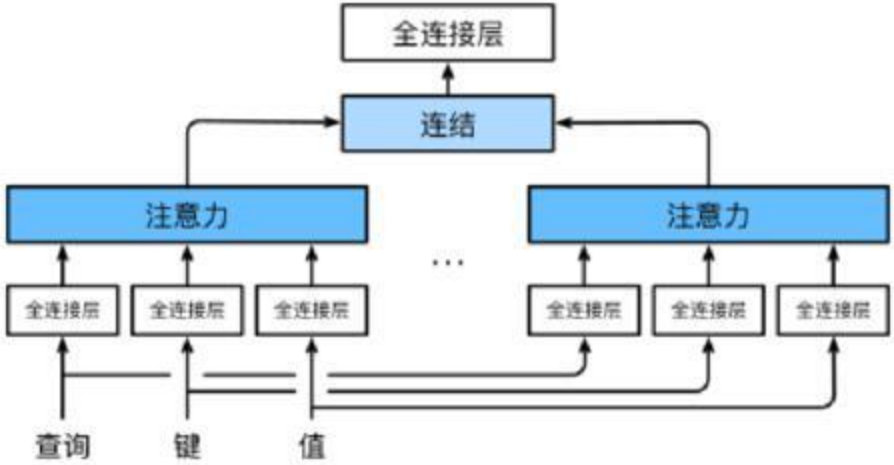
多头注意力：架构

[Vaswani 2017] **独立学习**多个注意力池化，称为“头”



多头注意力：架构

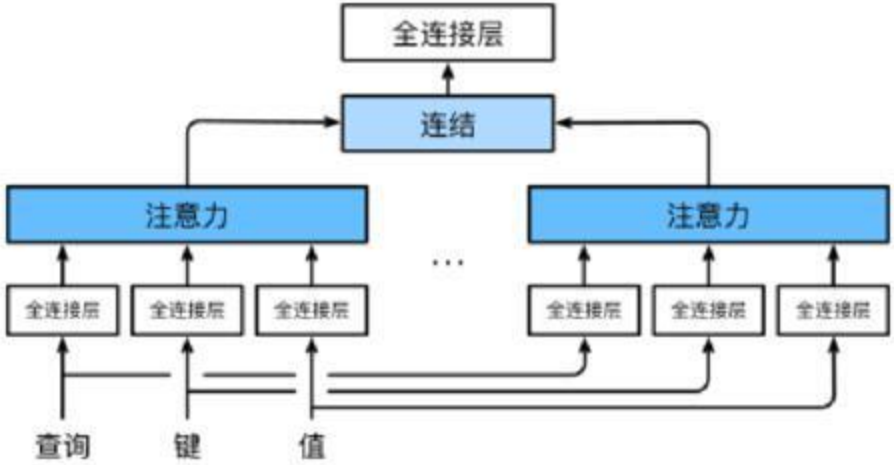
[Vaswani 2017] **独立学习**多个注意力池化，称为“头”



- 每个头都是**线性投影（全连接）**
  - 变换相同的查询、键、值：不同的子空间表示
  - 并行计算、输送到注意力池化

多头注意力：架构

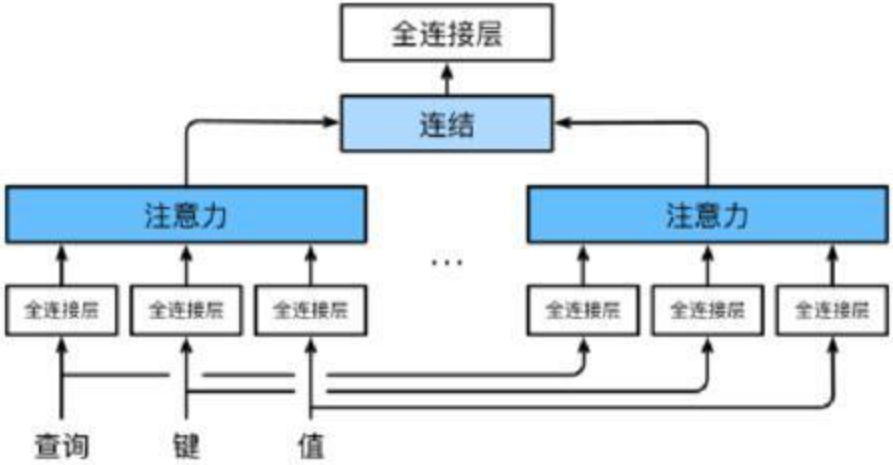
[Vaswani 2017] **独立学习**多个注意力池化，称为“头”



- 每个头都是**线性投影（全连接）**
  - 变换相同的查询、键、值：不同的子空间表示
  - 并行计算、输送到注意力池化
- 拼接注意力池化的输出，再线性投影（全连接）

多头注意力：模型计算

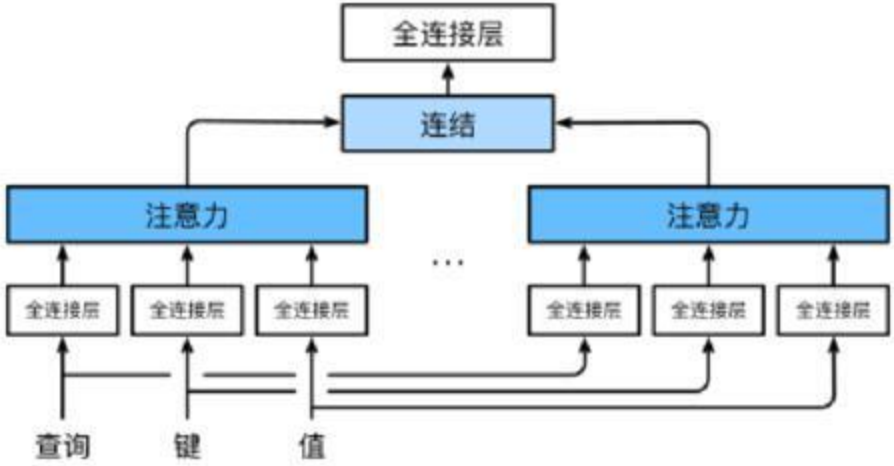
[Vaswani 2017] **独立学习**多个注意力池化，称为“头”



- 每个注意力头：$\mathbf{h}_i = f(\mathbf{W}_i^{(q)}\mathbf{q}, \mathbf{W}_i^{(k)}\mathbf{k}, \mathbf{W}_i^{(v)}\mathbf{v})$

多头注意力：模型计算

[Vaswani 2017] **独立学习**多个注意力池化，称为“头”



- 每个注意力头：$\mathbf{h}_i = f(\mathbf{W}_i^{(q)}\mathbf{q}, \mathbf{W}_i^{(k)}\mathbf{k}, \mathbf{W}_i^{(v)}\mathbf{v})$
- 输出转换：$\mathbf{W}_o[\mathbf{h}_1, .., \mathbf{h}_h]^T$

实验：多头注意力

小结：多头注意力

- 多头注意力：融合多个注意力池化的不同知识
  - 知识来源于相同的查询、键、值
  - 提取不同的子空间表示，学习不同长度的依赖关系

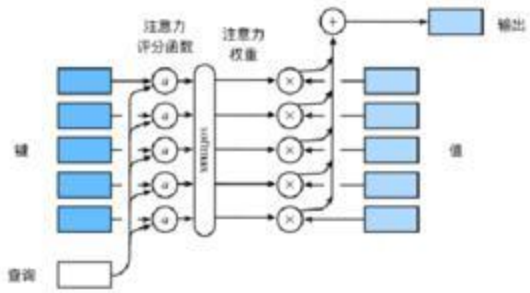
自注意力和位置编码

回顾：注意力池化

注意力池化：$f(x) = \sum_i \alpha(x, x_i) y_i$

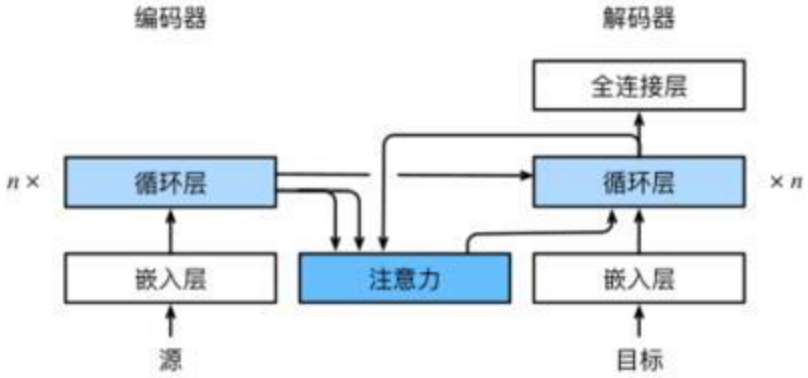
查询$\mathbf{q}$和m个“键－值”对$(\mathbf{k}_m, \mathbf{v}_m)$

$$f(\mathbf{q}, (\mathbf{k}_1, \mathbf{v}_1), .., (\mathbf{k}_m, \mathbf{v}_m)) = \sum_i \alpha(\mathbf{q}, \mathbf{k}_i)\mathbf{v}_i$$



回顾：编码器

机器翻译：对源序列编码，输出三条路径



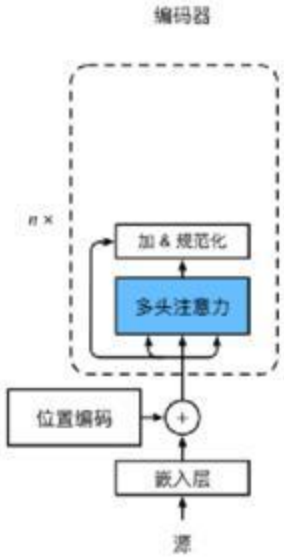
- 两条直接传入注意力池化：键$\mathbf{k}$、值$\mathbf{v}$
- 一条经过解码器：查询$\mathbf{q}$

自注意力

自注意力：查询、键、值来自同一组输入

$$\mathbf{y}_i = f(\mathbf{x}_i, (\mathbf{x}_1, \mathbf{x}_1), .., (\mathbf{x}_n, \mathbf{x}_n)) = \sum_j \alpha(\mathbf{x}_i, \mathbf{x}_j):$$

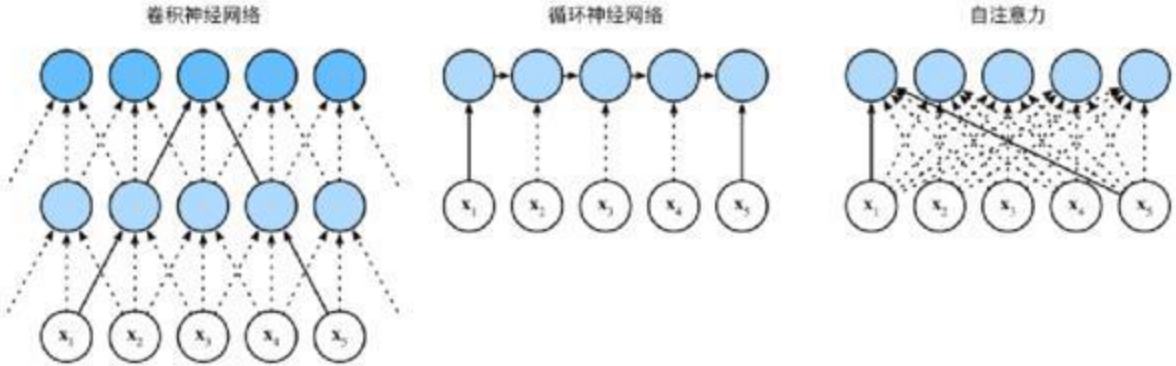
- 查询$\mathbf{q}$、键$\mathbf{k}$、值$\mathbf{v}$：都用$\mathbf{x}$替代



架构对比

序列任务：复杂性、顺序操作（妨碍并行计算）、最大路径长度

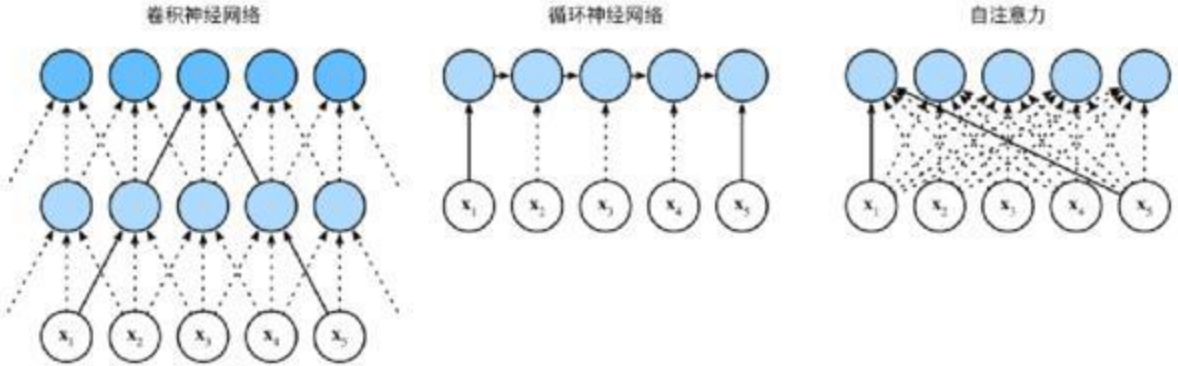
- [Hochreiter 2001] 任意序列位置组合间路径越短，越容易学习远距离依赖关系



架构对比：CNN

序列任务：复杂性、顺序操作（妨碍并行计算）、最大路径长度

- [Hochreiter 2001] 任意序列位置组合间路径越短，越容易学习远距离依赖关系

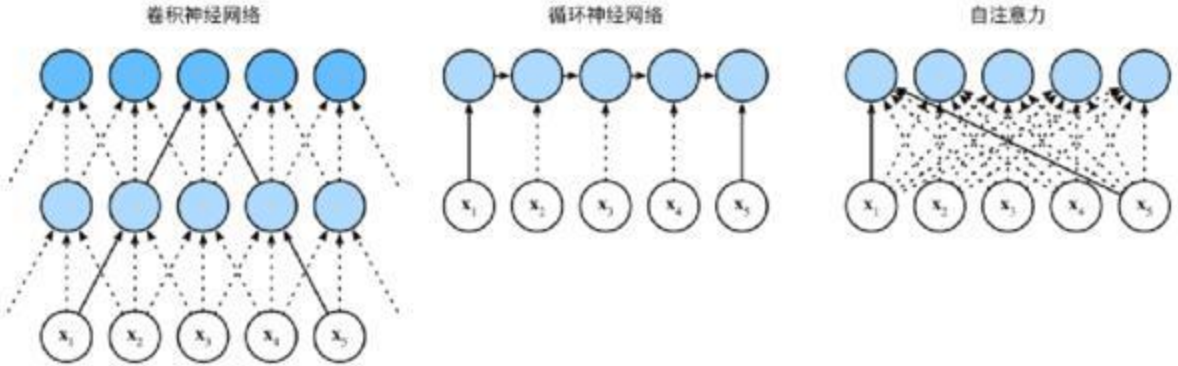


- 计算复杂度$O(knd^2)$：卷积核大小，序列长度，输入、输出通道数
- 并行度$O(n)$：同层元素间可并行
- 最长路径$O(n/k)$：层越深，感受野越大

架构对比：RNN

序列任务：复杂性、顺序操作（妨碍并行计算）、最大路径长度

- [Hochreiter 2001] 任意序列位置组合间路径越短，越容易学习远距离依赖关系

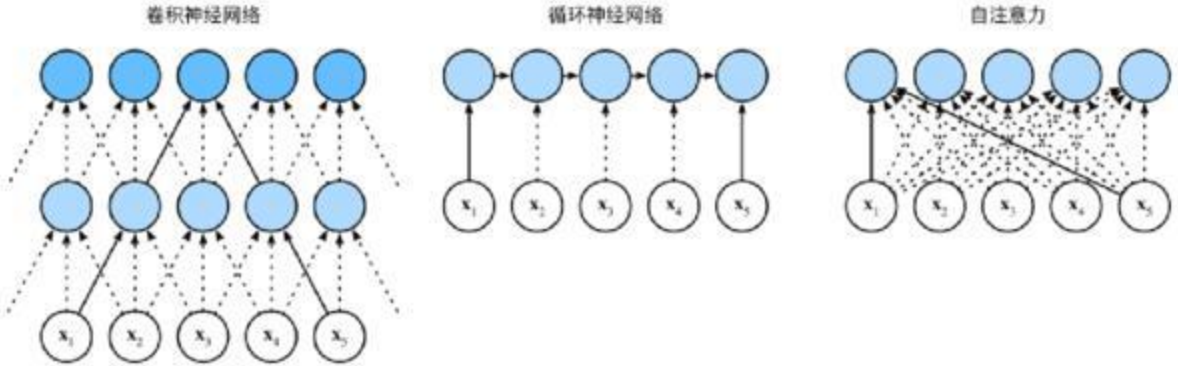


- 计算复杂度$O(nd^2)$：序列长度，权重矩阵$d \times d$、隐状态d相乘
- 并行度$O(1)$：无法并行
- 最长路径$O(n)$：只有相邻连接

架构对比：自注意力

序列任务：复杂性、顺序操作（妨碍并行计算）、最大路径长度

- [Hochreiter 2001] 任意序列位置组合间路径越短，越容易学习远距离依赖关系

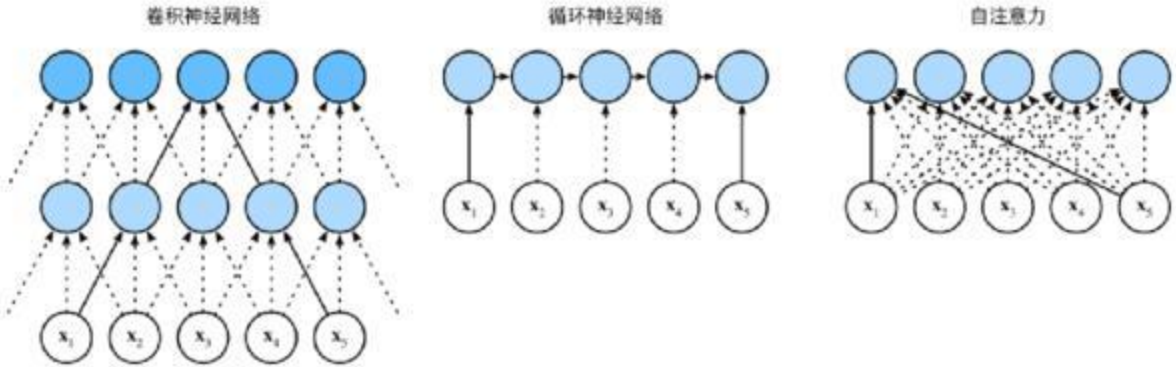


- 计算复杂度$O(n^2d)$：点积注意力（nxd乘dxn），输出（nxn乘nxd）
- 并行度$O(n)$：多头注意力可并行
- 最长路径$O(1)$：元素间通过注意力直接连接

架构对比：总结

序列任务：复杂性、顺序操作（妨碍并行计算）、最大路径长度

- [Hochreiter 2001] 任意序列位置组合间路径越短，越容易学习远距离依赖关系

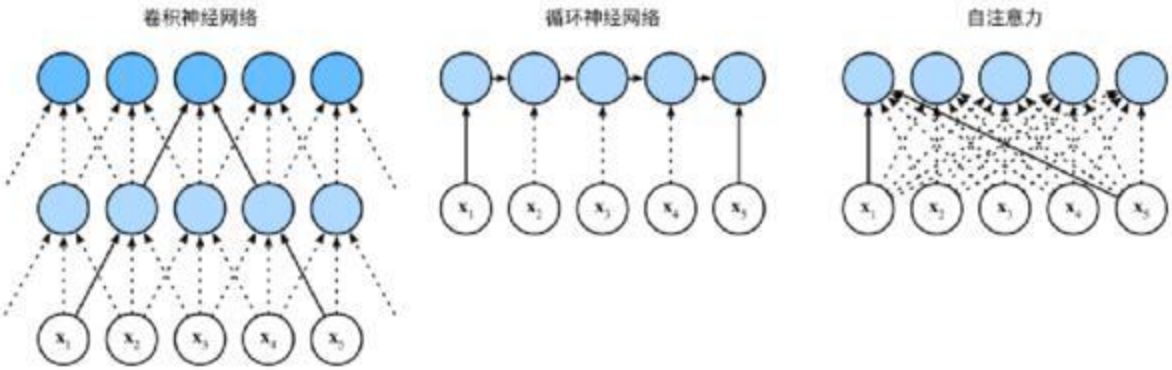


	CNN	RNN	自注意力
计算复杂度	$O(knd^2)$	$O(nd^2)$	$O(n^2d)$

- CNN、自注意力：并行计算优势
- 自注意力：善于处理远距离依赖
  - 计算速度非常慢

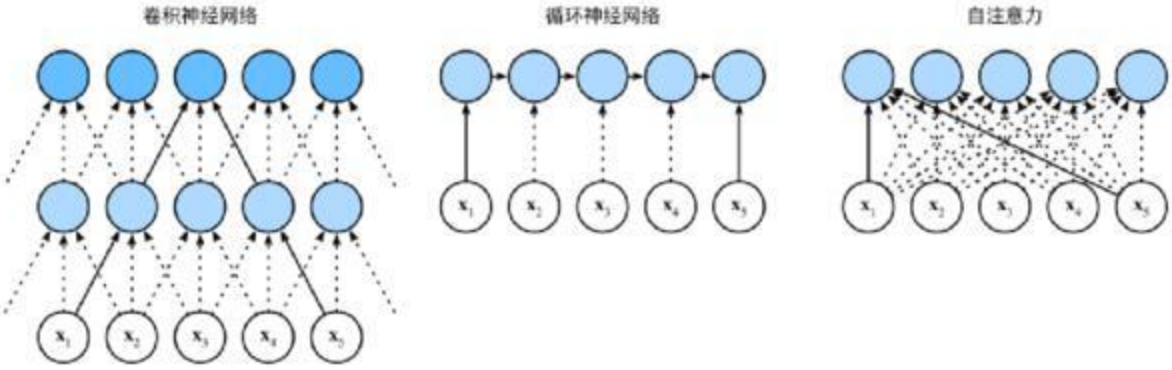
位置编码

自注意力的问题：为了并行计算，**丢失顺序信息**



位置编码

自注意力的问题：为了并行计算，**丢失顺序信息**



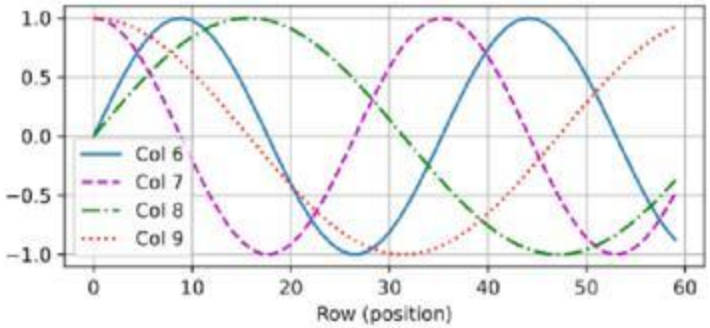
位置编码：给输入注入绝对或相对位置信息，可学习或固定

- [Vaswani 2017] 固定位置编码：嵌入矩阵$\mathbf{P} \in \mathbb{R}^{n \times d}$，$n$个词元的$d$维嵌入

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

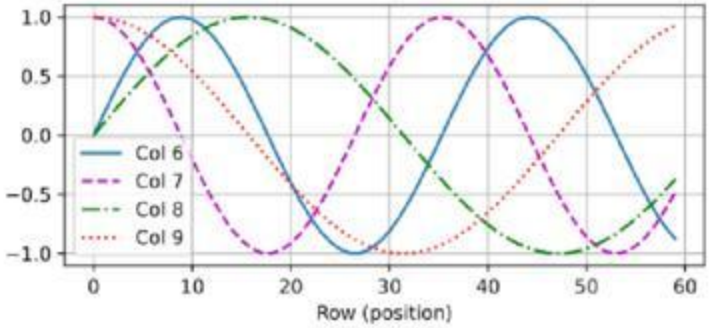
位置编码：绝对位置信息

三角函数族：频率不同，导致相同位置上值不同



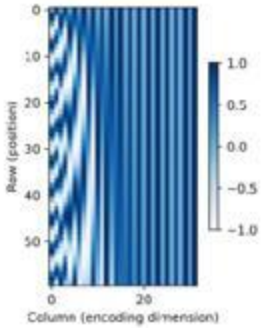
位置编码：绝对位置信息

三角函数族：频率不同，导致相同位置上值不同



词元序列 + 位置编码：$\mathbf{X} + \mathbf{P}$

- 相当于给每个输入注入位置信息



实验：自注意力和位置编码

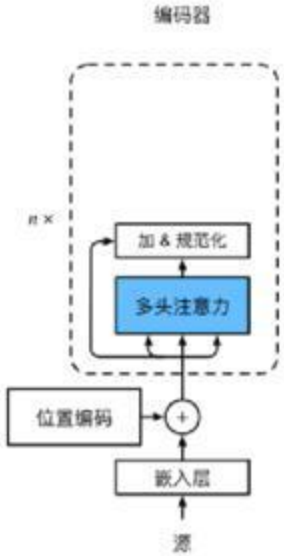
小结：自注意力和位置编码

- 自注意力：查询、键、值来自同一组输入
  - 完全并行，最长序列为1，计算复杂度高
- 位置编码：给每个输入注入位置信息

Transformer

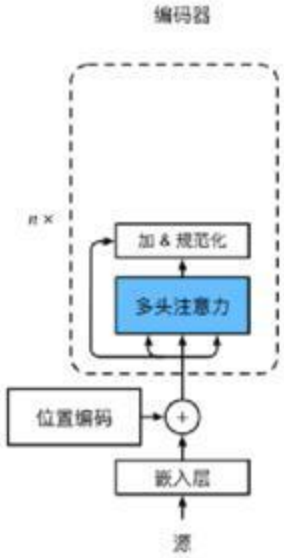
回顾：自注意力和位置编码

- **自注意力**：查询、键、值来自同一组输入
- **位置编码**：给每个输入注入位置信息



回顾：自注意力和位置编码

- **自注意力**：查询、键、值来自同一组输入
- **位置编码**：给每个输入注入位置信息



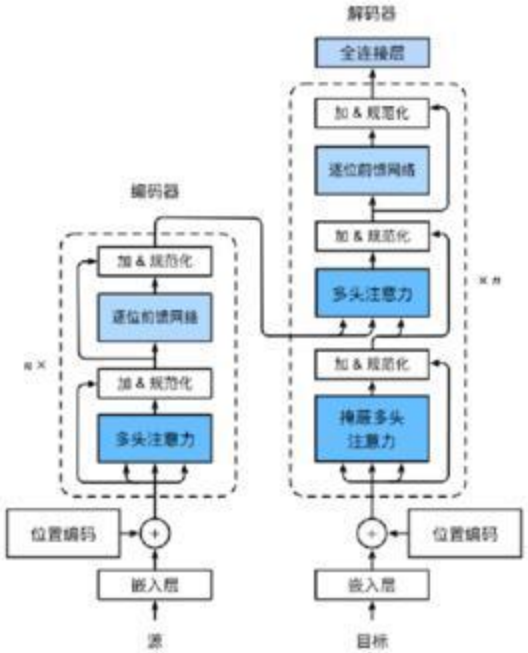
自注意力同时具有两个优势：完全并行、元素直连

- [Vaswani 2017] transformer：**完全基于注意力机制**，没有CNN/RNN
  - 最早用于文本序列，现已推广至其他各种领域

transformer：架构

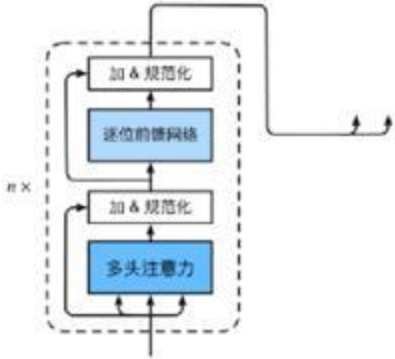
transformer：基于编码器-解码器

- 多头注意力、自注意力、位置编码
- 残差连接、**层规范化**
  - 层规范化：针对特征（编码）维度
    - 输入通常是变长序列，按长度规范
- **基于位置的前馈网络**
  - 而不是批量、时间步



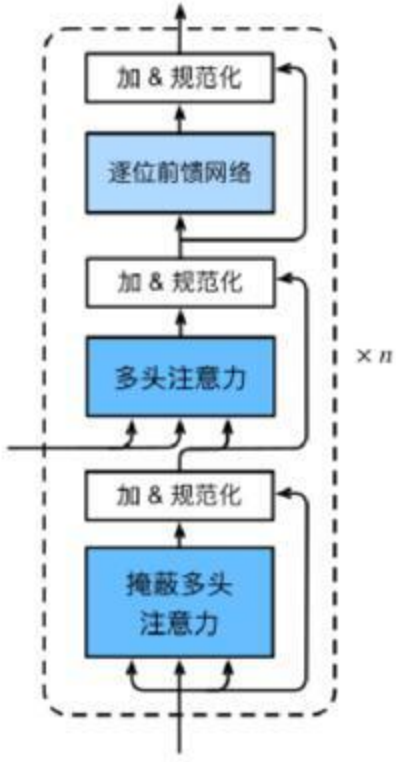
transformer：编码器模块

- 多头（自）注意力：提取不同距离依赖关系
- **基于位置的前馈网络**：全连接或1x1卷积
  - 合并批量、时间步通道，然后批量处理
    - (b, n, d)变形成(bn, d)
- **层规范化**：针对特征（编码）维度
  - 输入通常是变长序列，按长度规范
- **输出两条路径**：“键”、“值”
  - “查询”来自目标序列



transformer：解码器模块

- 多头（自）注意力**掩码**：预测时不能“偷看”未来
- **预测**$t+1$时间步：解码器输入前t个预测值
  - 前t个预测值：“键”、“值”
  - 第t个预测值：同时用于“查询”



实验：Transformer

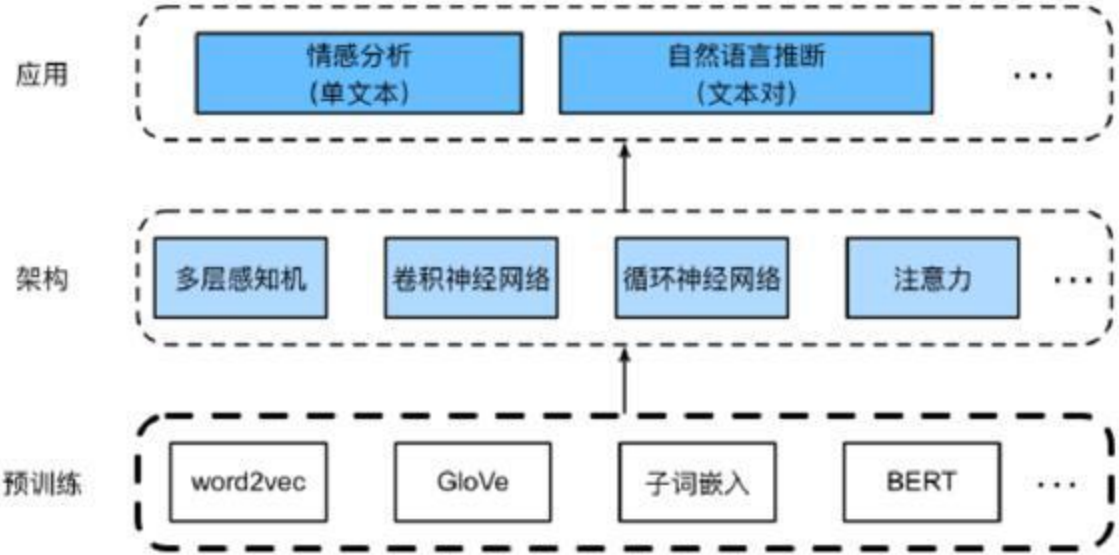
小结：Transformer

- transformer：完全基于注意力机制的编码器-解码器
- 编码器、解码器都由transformer块串联而成
- transformer块：多头（自）注意力、基于位置的前馈网络、层归一化

词嵌入（word2vec）

自然语言处理：预训练

文本处理：从文本自身的上下文获取监督信息，即**自监督学习**



- 预训练：学习词元的向量表示，即**词嵌入**

实验：Penn Tree Bank（PTB）

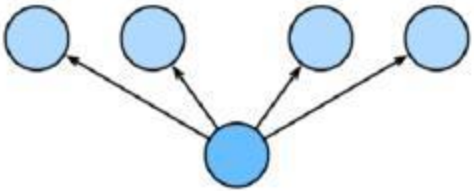
实验：skip-gram

连续词袋

[Mikolov 2013] **连续词袋 CBOW**模型类似于skip-gram

skip-gram

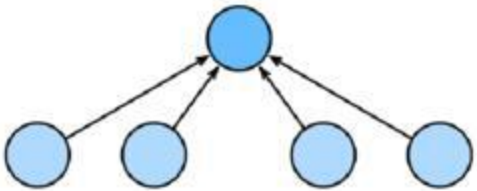
- 假设中心词生成上下文



- $P(今,人,为,刀|方) = P(今|方)P(人|方)P(为|方)P(刀|方$

CBOW

- 假设上下文生成中心词



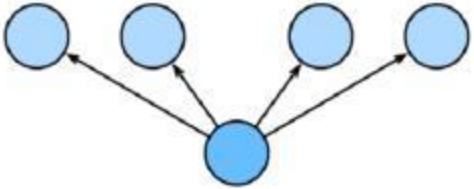
- $P(方|今,人,为,刀)$

连续词袋

[Mikolov 2013] **连续词袋 CBOW**模型类似于skip-gram

skip-gram

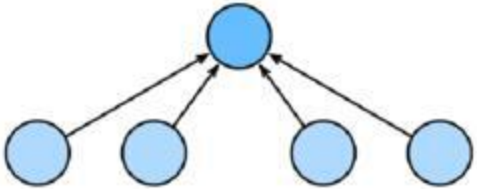
- 假设中心词生成上下文



- $P(今,人,为,刀|方) = P(今|方)P(人|方)P(为|方)P(刀|方$

CBOW

- 假设上下文生成中心词



- $P(方|今,人,为,刀)$

两者训练方法相同，仅公式计算有区别

全局向量的词嵌入

[Pennington 2014] **全局向量的词嵌入 GloVe**：预先提取语料集的全局统计信息

- 以 w_i 为中心词的上下文可能有多个
- **重数** x_{ij} ： 上下文中的词 w_j 与 w_i 共现的全局计数

全局向量的词嵌入

[Pennington 2014] **全局向量的词嵌入 GloVe**：预先提取语料集的全局统计信息

- 以 w_i 为中心词的上下文可能有多个
- **重数** x_{ij} ： 上下文中的词 w_j 与 w_i 共现的全局计数

（带全局语料统计的）skip-gram损失函数：$-\sum_{ij} x_{ij} \log P(w_j|w_i)$

全局向量的词嵌入

[Pennington 2014] **全局向量的词嵌入 GloVe**：预先提取语料集的全局统计信息

- 以 w_i 为中心词的上下文可能有多个
- **重数** x_{ij} ： 上下文中的词 w_j 与 w_i 共现的全局计数

（带全局语料统计的）skip-gram损失函数：$-\sum_{ij} x_{ij} \log P(w_j|w_i)$

GloVe损失函数：

$$\sum_{ij} h(x_{ij}) \left(w_j \cdot w_i + b_i + c_j - \log x_{ij}\right)^2$$

- 平方损失
- 中心词偏置 b_i 和上下文词偏置 c_i
- 权重函数 $h(x_{ij})$： $h(x)$ 在$[0, 1]$递增

实验：词的相似性、类比任务

fastText模型

[Bojanowski 2017] fastText模型是一种**子词嵌入**方法

- 子词是基于单字符的N元语法
  - 可以被认为是子词级skip-gram

fastText模型

[Bojanowski 2017] fastText模型是一种**子词嵌入**方法

- 子词是基于单字符的N元语法
  - 可以被认为是子词级skip-gram

例如单词“where”

```
“<wh”、“whe”、“her”、“ere”、“re>”、“<where>”
```

fastText模型

[Bojanowski 2017] fastText模型是一种**子词嵌入**方法

- 子词是基于单字符的N元语法
  - 可以被认为是子词级skip-gram

例如单词“where”

```
“<wh”、“whe”、“her”、“ere”、“re>”、“<where>”
```

开源：https://fasttext.cc

- 解决测试集中未知词<unk>的问题
- 解决词变形很多、在文本中罕见的问题

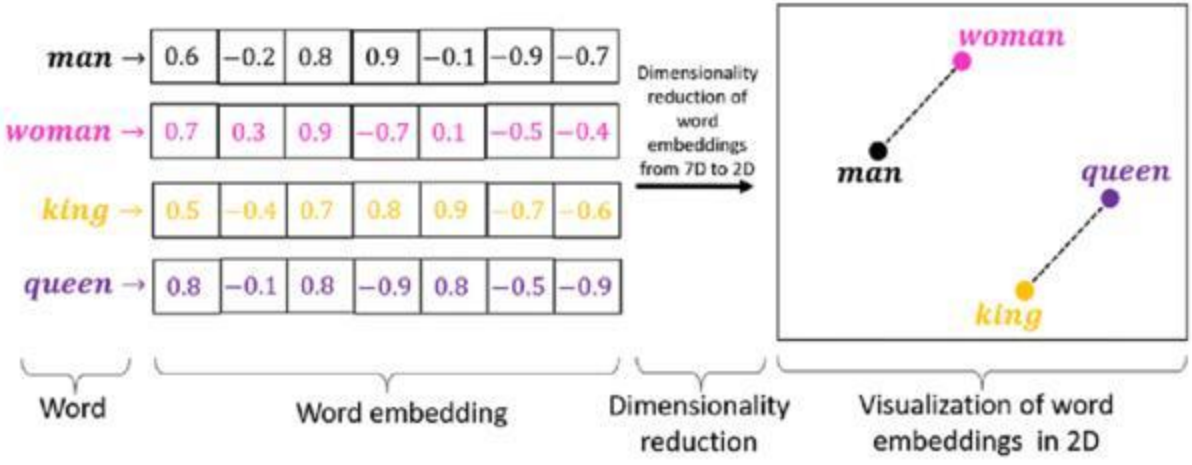
实验：BPE 分词器

来自Transformers的双向编码器表示（BERT）

回顾：词嵌入模型

词嵌入模型：输出**嵌入矩阵**

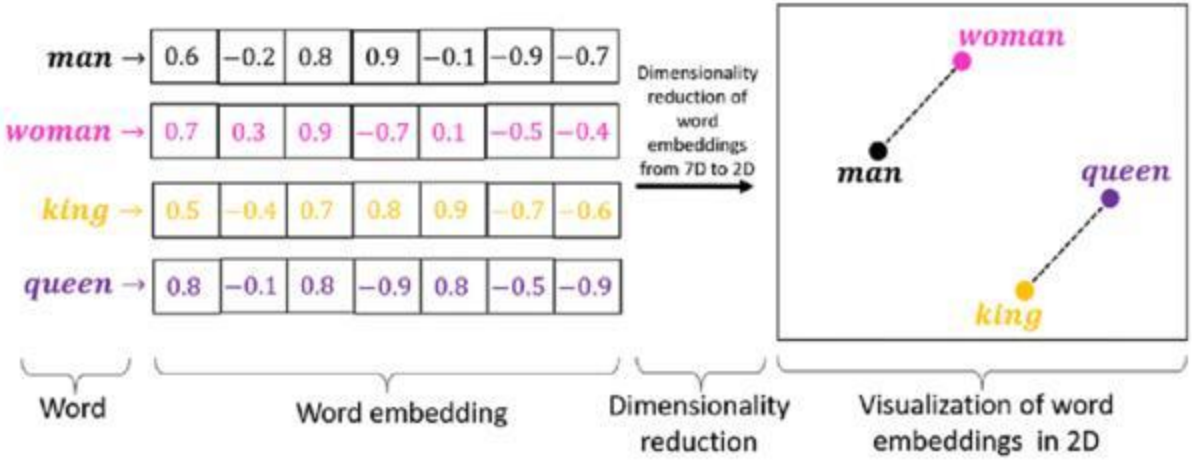
- 行数与词表大小相同，每行一个词嵌入向量



回顾：词嵌入模型

词嵌入模型：输出**嵌入矩阵**

- 行数与词表大小相同，每行一个词嵌入向量



注意：（静态）词嵌入是**上下文无关**的

- 例如word2vec和GloVe：预训练后，将词嵌入**固定分配**给词

- 但相同词目可以有不同含义，例如mouse：老鼠；鼠标

从上下文无关到上下文敏感

词元的**上下文敏感**表示：$f(x, c(x))$

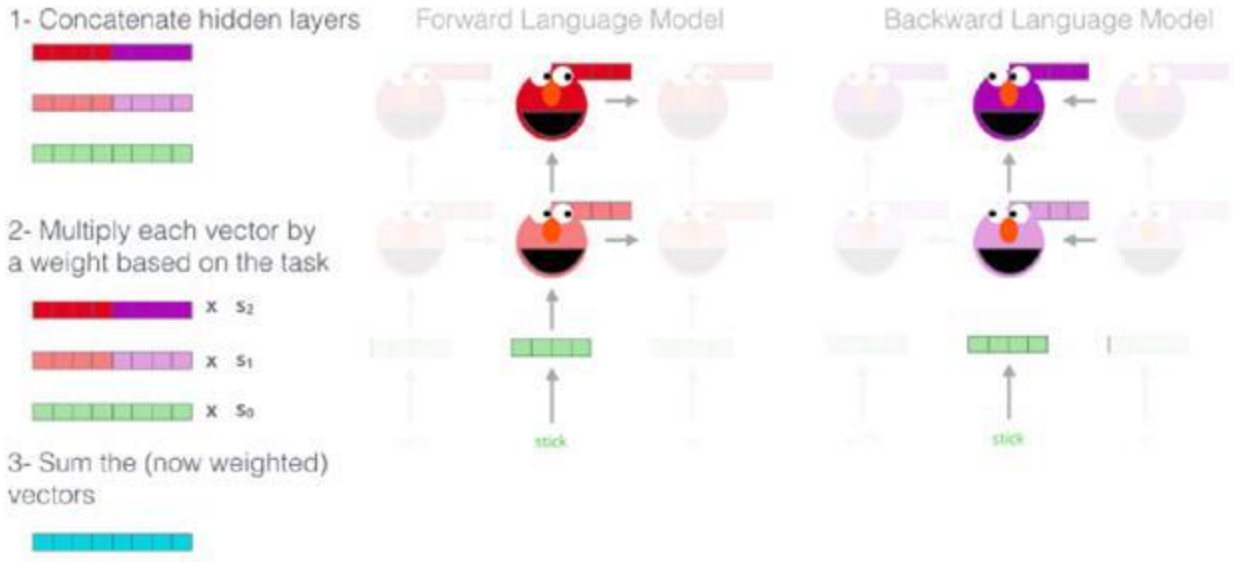
- 取决于词x、上下文$c(x)$

从上下文无关到上下文敏感

词元的**上下文敏感**表示：$f(x, c(x))$

- 取决于词x、上下文$c(x)$

[Peters 2018] ELMo（Embeddings from LMs，来自语言模型的嵌入）

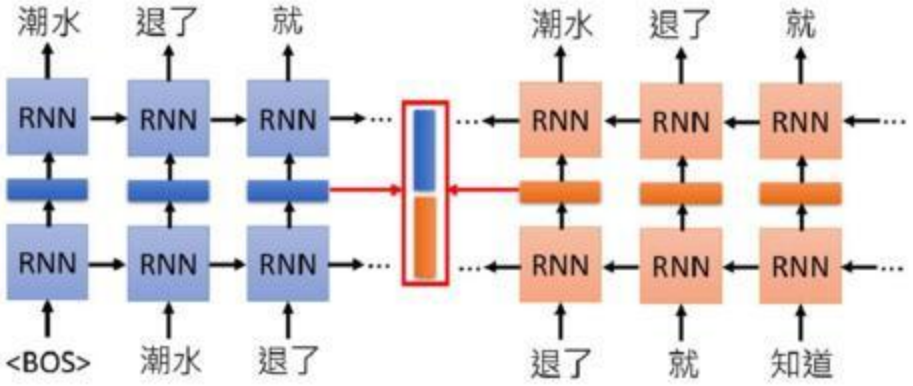


ELMo：示例

Embeddings from Language Model (ELMO) https://arxiv.org/abs/1802.05365

- RNN-based language models (trained from lots of sentences)

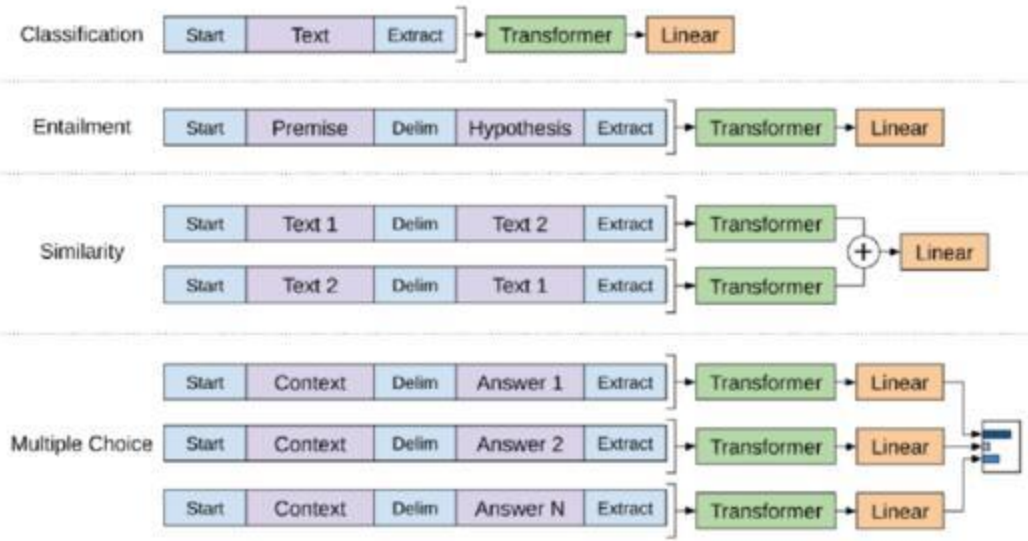
e.g. given "潮水 退了 就 知道 誰 沒穿 褲子"



从特定任务到不可知任务

[Radford 2018] GPT（Generative Pre Training，生成式预训练）

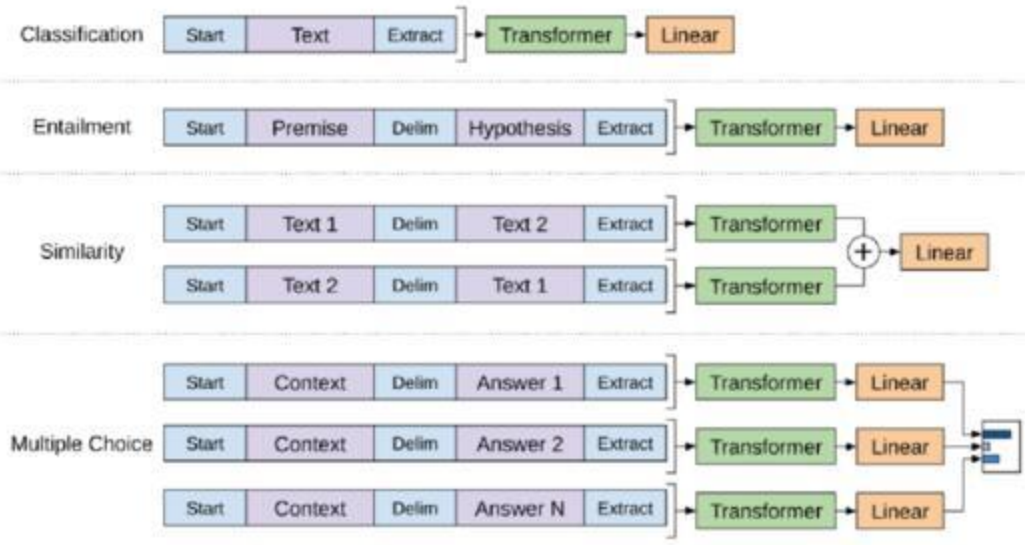
- **任务无关**：对预训练Transformer解码器中所有参数微调



从特定任务到不可知任务

[Radford 2018] GPT（Generative Pre Training，生成式预训练）

- **任务无关**：对预训练Transformer解码器中所有参数微调

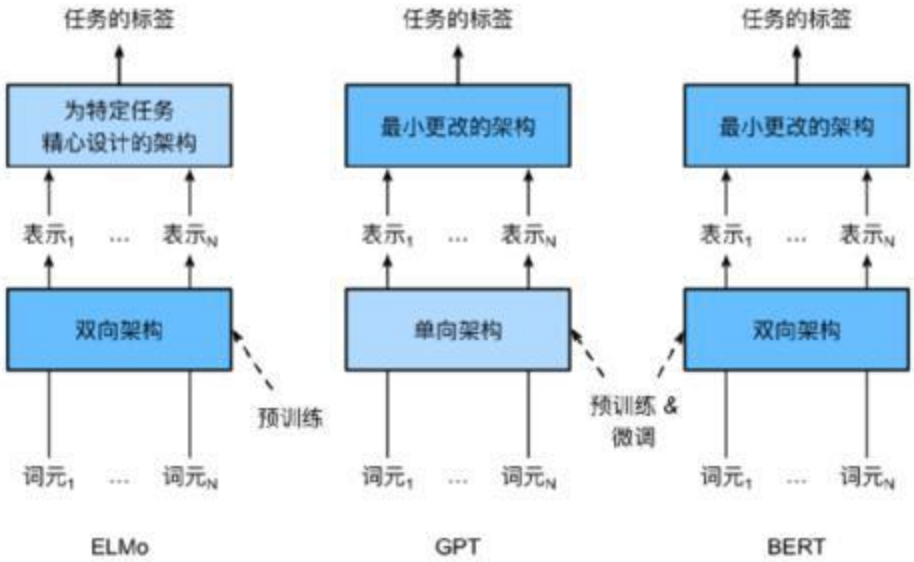


注意：GPT 只用到 **Transformer 的解码器**

BERT：结合两者优点

[Devlin 2018] BERT（来自Transformers的双向编码器表示）

- 对上下文双向编码：**上下文敏感表示**
- 只训练任务相关的额外输出层：**极少架构修改**



BERT：架构

BERT 本质上是**只有编码器的 Transformer**

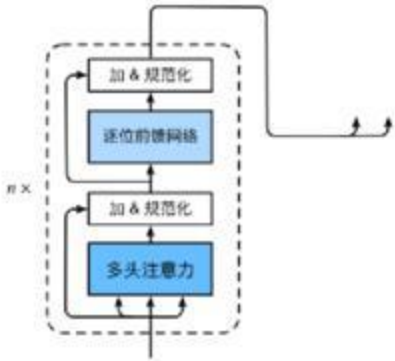
基本模型：12层Transformer编码模块

- 768个隐藏单元，12个自注意头
  - **1.1亿**个参数

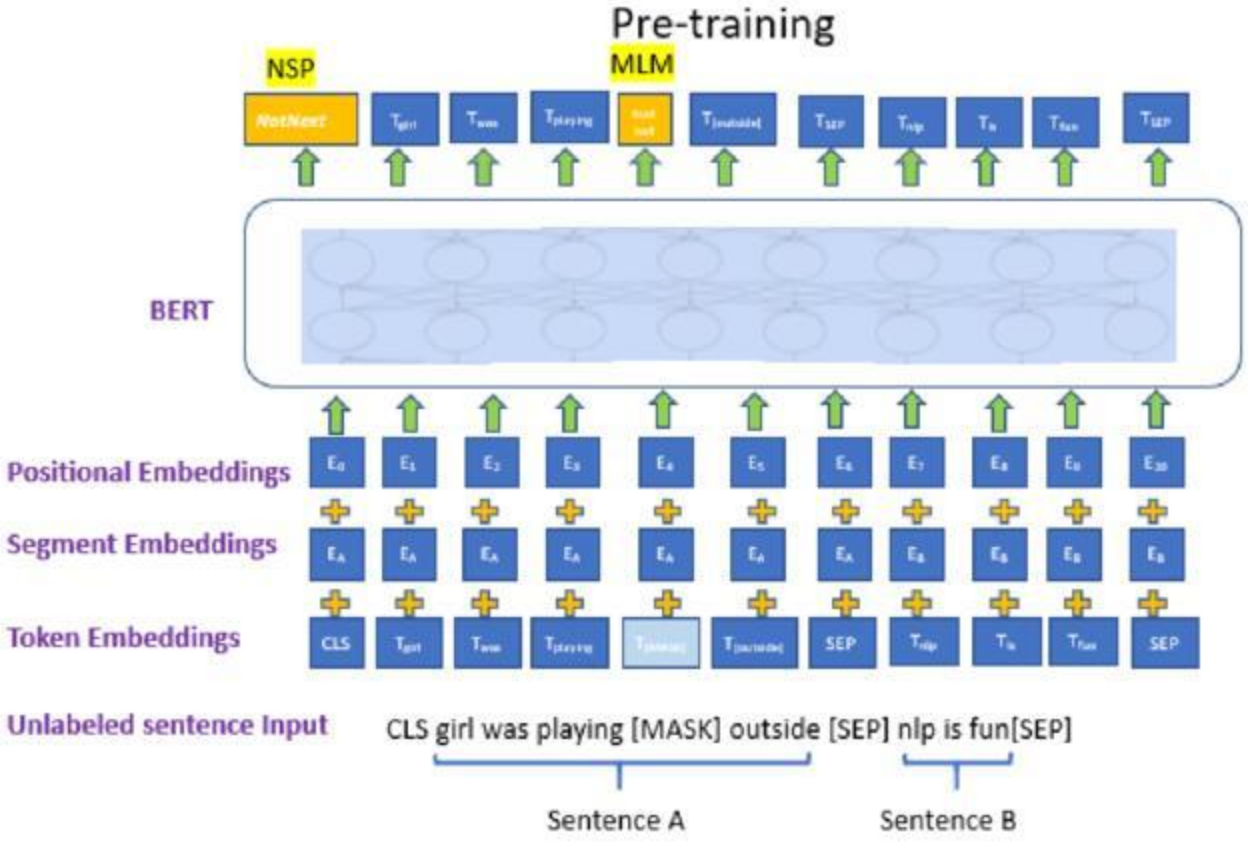
大型模型：24层Transformer编码模块

- 1024个隐藏单元，16个自注意头
  - **3.4亿**个参数

大规模数据训练：**> 30 亿**词



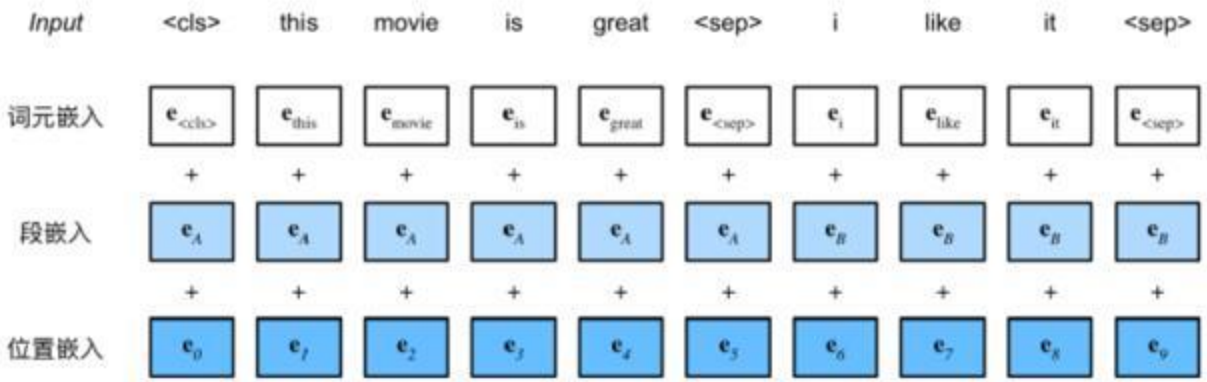
BERT：预训练架构图


Pre-training
NSP
MLM
BERT
Positional Embeddings
Segment Embeddings
Token Embeddings
CLS
SEP
Unlabeled sentence Input
CLS girl was playing [MASK] outside [SEP] nlp is fun[SEP]
Sentence A
Sentence B

BERT：输入表示

BERT任务无关，需要**明确区分输入序列**

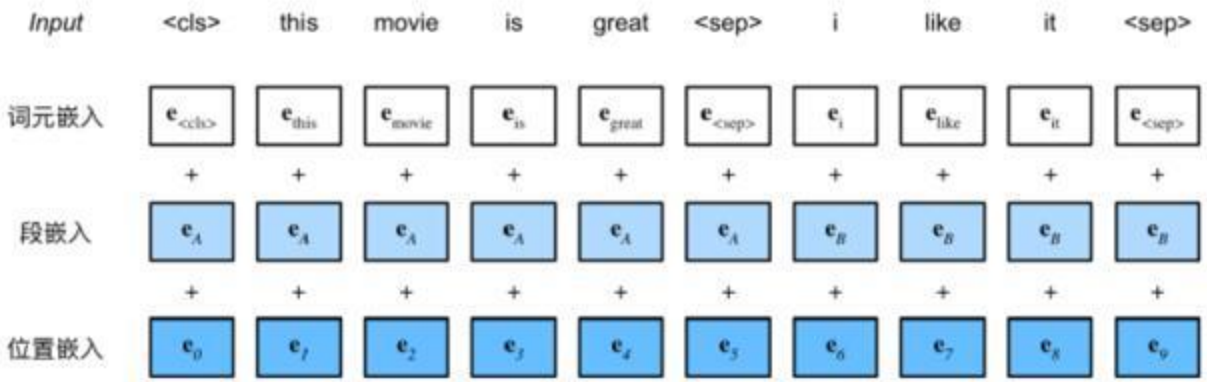
- 单个文本：特殊类别词元“<cls>”、文本序列的标记、特殊分隔词元“<sep>”
- 文本对：“<cls>”、第一个文本序列、“<sep>”、第二个文本序列、“<sep>”



BERT：输入表示

BERT任务无关，需要**明确区分输入序列**

- 单个文本：特殊类别词元“<cls>”、文本序列的标记、特殊分隔词元“<sep>”
- 文本对：“<cls>”、第一个文本序列、“<sep>”、第二个文本序列、“<sep>”



- BERT使用**可学习的位置嵌入**

BERT：迁移特征编码

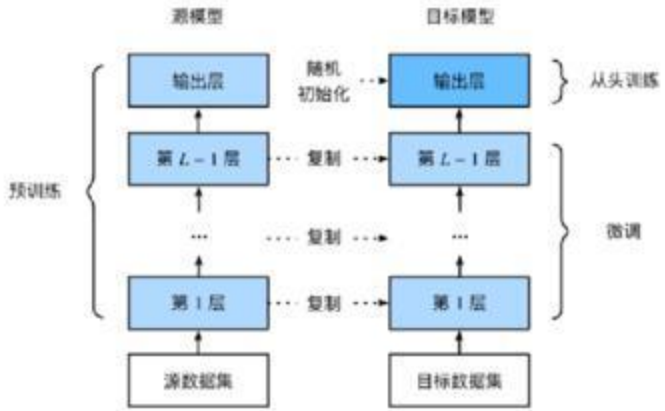
BERT 如何实现“极少架构修改”？**微调预训练模型**

BERT 作为编码器：提取**语言特征**

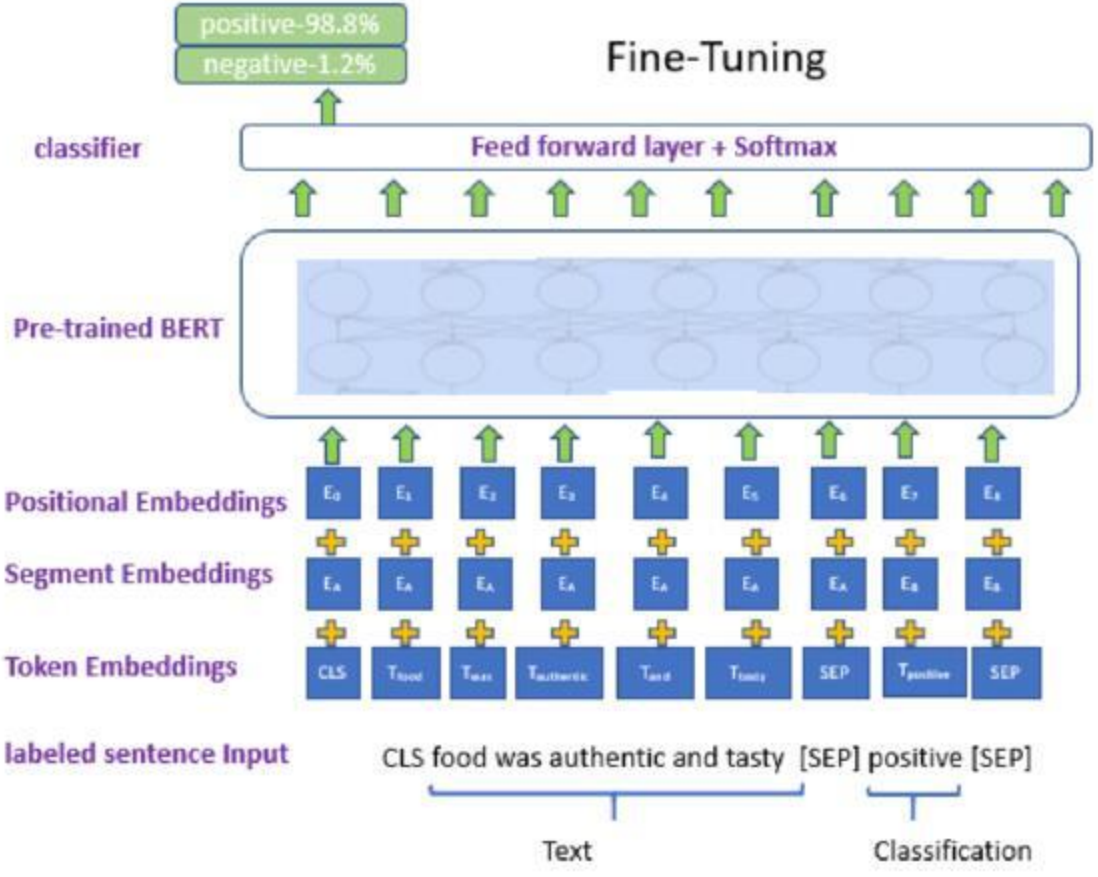
- 输出编码：词义、语义向量

预训练的模型如何应用于新任务？

- **微调**：固定特征参数
  - 只需训练输出层



BERT：微调架构图


positive-98.8%
negative-1.2%
Fine-Tuning
classifier
Feed forward layer + Softmax
Pre-trained BERT
Positional Embeddings
Segment Embeddings
Token Embeddings
CLS
SEP
SEP
labeled sentence Input
CLS food was authentic and tasty [SEP] positive [SEP]
Text
Classification

实验：BERT模型

BERT：预训练任务

完形填空 Masked LM：使用双向上下文，自监督预测

- 自监督：随机选取15%词元作为（预测）目标

BERT：预训练任务

完形填空 Masked LM：使用双向上下文，自监督预测

- 自监督：随机选取15%词元作为（预测）目标
- 掩码替换：例如在“this movie is great”中掩蔽、预测“great”
  - 80%概率**替换为特殊词元**“<mask>”
    - <cls>this movie is <mask><sep><sep>

BERT：预训练任务

完形填空 Masked LM：使用双向上下文，自监督预测

- 自监督：随机选取15%词元作为（预测）目标
- 掩码替换：例如在“this movie is great”中掩蔽、预测“great”
  - 80%概率**替换为特殊词元**“<mask>”
    - <cls>this movie is <mask><sep><sep>
- 10%概率替换为随机词元：引入**偶然噪音**
  - <cls>this movie is drink<sep><sep>
- 10%概率不变：正例总数还是多一些
  - <cls>this movie is great<sep><sep>

BERT：预训练任务

完形填空 Masked LM：使用双向上下文，自监督预测

- 自监督：随机选取15%词元作为（预测）目标
- 掩码替换：例如在“this movie is great”中掩蔽、预测“great”
  - 80%概率**替换为特殊词元**“<mask>”
    - <cls>this movie is <mask><sep><sep>
- 10%概率替换为随机词元：引入**偶然噪音**
  - <cls>this movie is drink<sep><sep>
- 10%概率不变：正例总数还是多一些
  - <cls>this movie is great<sep><sep>

下一句预测 NSP：建模文本对间逻辑关系

- 50%概率将第二个序列替换成随机句子
  - <cls>this movie is great<sep>hello world<sep>

实验：用于预训练BERT的数据集

实验：预训练BERT

小结：BERT

- word2vec、GloVe等静态词嵌入模型：上下文无关
- ELMo、GPT构建上下文敏感词表示
  - ELMo对上下文双向编码，但使用特定于任务的架构
  - GPT是任务无关的，但单向从左到右编码上下文
- BERT结合两方面优点：对上下文双向编码；仅需最小的架构更改
  - 输入序列的嵌入：词元嵌入、片段嵌入和位置嵌入的和
  - 预训练包括两个任务：掩蔽语言模型、下一句预测

Review

本章内容

多头注意力。自注意力和位置编码。Transformer。词嵌入。来自Transformers的双向编码器表示（BERT）。

重点：多头注意力；自注意力，位置编码；Transformer；词嵌入；BERT。

难点：注意力模型的实现。

学习目标

- 理解多头注意力的原理（融合多个注意力池化的不同知识）
- 理解自注意力的特点（查询、键、值来自同一组输入）与技术优势（高度并行，善于处理远距离依赖）；了解位置编码的原理（给输入注入绝对或相对位置信息）
- 理解 Transformer 模型架构
- 理解词嵌入方法 word2vec（包括skip-gram、CBOW）、GloVe、BPE 的原理
- 理解 ELMo、GPT、BERT 模型的原理

问题

简述多头注意力的原理。

简述自注意力的特点与技术优势。

简述 Transformer 模型架构。

简述词嵌入方法 word2vec（包括skip-gram、CBOW）、GloVe、BPE 的原理。

简述 ELMo、GPT、BERT 模型的原理。