



# Attention is all you need

September 24, 2024



## 为什么选择这篇文献？

```

Step 1: 安装依赖 (CPU 的内存基础信息)
pip install torch torchvision torchaudio --index-url https://download.pytorch.org/whl/cpu

Step 2: 配置 main.py 程序依赖环境
conda create --name pytorch=0.39.0
conda activate MLP
pip install torch torchvision torchaudio --index-url https://download.pytorch.org/whl/cpu

Step 3: 激活并测试环境
conda activate MLP
python
import torch
print(torch.__version__)
2.5.1+cpu
print(torch.cuda.is_available())
True
exit()

Step 4: Transformer 模型测试的数据集
python main.py
...
"Attention is All You Need", 作者 pytorch 提供源码
官网: https://arxiv.org/abs/1609.08144v1
源码: https://github.com/lm-sys/llm-recipes/tree/main/recipes/transformer
数据集: https://github.com/lm-sys/llm-recipes/tree/main/recipes/transformer

def subsequence_mask(sizes: List) -> torch.Tensor:
    """
    生成掩码矩阵
    """
    # 生成掩码矩阵
    mask = torch.zeros(sizes[0], sizes[0])
    # 生成掩码矩阵
    for i in range(sizes[0]):
        for j in range(sizes[0]):
            if i < j:
                mask[i, j] = 1
    return mask

def subsequence_mask(sizes: List) -> torch.Tensor:
    """
    生成掩码矩阵
    """
    # 生成掩码矩阵
    mask = torch.zeros(sizes[0], sizes[0])
    # 生成掩码矩阵
    for i in range(sizes[0]):
        for j in range(sizes[0]):
            if i < j:
                mask[i, j] = 1
    return mask

def subsequence_mask(sizes: List) -> torch.Tensor:
    """
    生成掩码矩阵
    """
    # 生成掩码矩阵
    mask = torch.zeros(sizes[0], sizes[0])
    # 生成掩码矩阵
    for i in range(sizes[0]):
        for j in range(sizes[0]):
            if i < j:
                mask[i, j] = 1
    return mask

```

图 1: README.html - 2024-07-10 复现 Transformer 模型所需要的依赖环境 - <https://orzzz.net>



# 目录

研究背景

模型架构

注意力

掩码

位置编码

研究结论



# 研究背景

## 以往的 seq2seq 序列转化

- 基于 Encoder-Decoder 的 RNN 和 CNN 网络
- 时间  $t + 1$  的计算依赖于  $t$  时刻的结果  $\implies$  限制并行计算能力
- LSTM 算法的局限性  $\implies$  时序过早的信息容易被遗弃
- 整套 RNN 和 CNN 网络时空复杂度高，计算和内存开销大

## 现在的 Transformer 模型

- 通过 Attention 机制连接 Encoder 和 Decoder，摒弃 RNN 和 CNN 架构



# 模型架构

## Embedding

one-hot vector  $\xrightarrow{E}$  meaningful vector

## Positional Encoding

trigonometric function

## Encoder-Decoder $\times 6$

(Masked) Multi-Head Attention + Norm + FFN

## Generator

Linear + log\_softmax

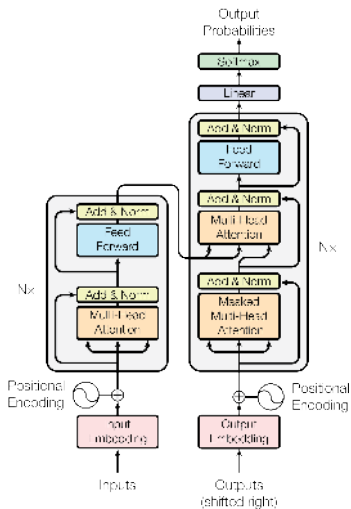


图 2: Transformer 网络



# 注意力

## 公式

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right) V$$

## 全局信息查询

数据库: 键值对字典 {"Bob": 18, "Cesus": 22, "Bob": 20, "Bylan": 19}

Q(uey): 查询所有“Bob”的平均年龄  $\implies (18 + 20) \div 2 = 19$

Q(uey): 查询首字母为“B”的平均年龄  $\implies (18 + 20 + 19) \div 3 = 19$

Q(uey): 查询“&%#@”的平均年龄



# 注意力

## 向量相似度化

数据库字典:  $\{k_1 : 18, k_2 : 22, k_3 : 20, k_4 : 19\}$

$$\begin{cases} k_1 = [1, 2, 0] & \#Bob \\ k_2 = [0, 0, 2] & \#Cesus \\ k_3 = [1, 2, 0] & \#Bob \\ k_4 = [1, 4, 0] & \#Bylan \end{cases}$$

## 注意力查询

约定:  $[1, 0, 0]$  代表首字母为“B”, Q(uey): 查询首字母为“B”的平均年龄

$$\text{softmax} \left\{ [1, 0, 0] \begin{bmatrix} 1 & 0 & 1 & 1 \\ 2 & 0 & 2 & 4 \\ 0 & 2 & 0 & 0 \end{bmatrix} \right\} [18, 22, 20, 19]^T = \left[ \frac{1}{3}, 0, \frac{1}{3}, \frac{1}{3} \right] [18, 22, 20, 19]^T = 19$$



# 注意力

## 多头注意力

卷积层  $\implies$  多个卷积核  $\implies$  多个通道特征

自注意力层  $\implies$  多个注意力  $\implies$  多头自注意力

$$head_i(E) = Attention(EQ_{W_i}, EK_{W_i}, EV_{W_i})$$

$$MultiHeadSelfAttention(E) = Contact(head_1, head_2, \dots, head_h) \odot_W$$

$$MultiHeadSelfAttention(Q, K, V) = Contact($$

$$Attention(QQ_{W_1}, KK_{W_1}, VV_{W_1}),$$

$$Attention(QQ_{W_2}, KK_{W_2}, VV_{W_2}),$$

$$\dots,$$

$$Attention(QQ_{W_h}, KK_{W_h}, VV_{W_h}),$$

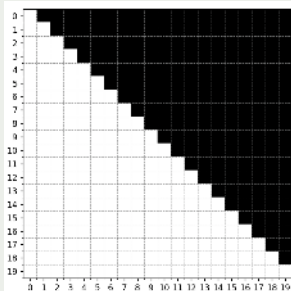
$$) \odot_W$$





# 掩码

## 掩码矩阵



$$QK^T = \begin{pmatrix} q_1 \cdot k_1 & q_1 \cdot k_2 & q_1 \cdot k_3 & q_1 \cdot k_4 \\ q_2 \cdot k_1 & q_2 \cdot k_2 & q_2 \cdot k_3 & q_2 \cdot k_4 \\ q_3 \cdot k_1 & q_3 \cdot k_2 & q_3 \cdot k_3 & q_3 \cdot k_4 \\ q_4 \cdot k_1 & q_4 \cdot k_2 & q_4 \cdot k_3 & q_4 \cdot k_4 \end{pmatrix}$$

$$QK^T \cdot M = \begin{pmatrix} q_1 \cdot k_1 & \infty & \infty & \infty \\ q_2 \cdot k_1 & q_2 \cdot k_2 & \infty & \infty \\ q_3 \cdot k_1 & q_3 \cdot k_2 & q_3 \cdot k_3 & \infty \\ q_4 \cdot k_1 & q_4 \cdot k_2 & q_4 \cdot k_3 & q_4 \cdot k_4 \end{pmatrix}$$

图 3: 黑色部分以  $-\infty$  代替, softmax 之后注意力权重为 0



# 掩码

## 掩码矩阵代码

```
def subsequentMask(size: int) -> torch.Tensor:
    """
    掩码矩阵图见 https://s21.ax1x.com/2024/07/10/pkf52UH.png

    创建掩码矩阵，屏蔽后续位置，防止解码器计算自注意力"看到"未来的词，保持序列生成的因果关系，自注意力是并行计算
    >>> subsequentMask(4)
    >>> tensor(
      [
        [
          [ True, False, False, False],
          [ True, True, False, False],
          [ True, True, True, False],
          [ True, True, True, True]
        ]
      ]
    )
    """
    attention_shape = (1, size, size)
    # torch.triu 把一个矩阵强制转化成上三角矩阵，从索引为 1 的对角线处处理。
    subsequent_mask = torch.triu(torch.ones(attention_shape), diagonal=1).type(torch.uint8)
    return subsequent_mask == 0
```

图 4: 输出  $t + 1$  个单词，阻止模型知道  $t + 1$  之后的信息，只保留  $t$  之前的信息



# 掩码

## 推理过程

decoder input:  $\langle \text{SOS} \rangle$  decoder output:  $y_1$

decoder input:  $\langle \text{SOS} \rangle y_1$  decoder output:  $y_2$

decoder input:  $\langle \text{SOS} \rangle y_1 y_2$  decoder output:  $y_3$

decoder input:  $\langle \text{SOS} \rangle y_1 y_2 y_3$  decoder output:  $y_4$

## 掩码过程

$\langle \text{SOS} \rangle y_1 y_2 y_3 y_4 \implies y_1$

$\langle \text{SOS} \rangle y_1 y_2 y_3 y_4 \implies y_2$

$\langle \text{SOS} \rangle y_1 y_2 y_3 y_4 \implies y_3$

$\langle \text{SOS} \rangle y_1 y_2 y_3 y_4 \implies y_4$



# 位置编码

## 绝对编码

把一个语句中所有单词放到一个集合里，然后从集合的第一个元素直到最后一个元素，标记为  $1, 2, \dots, n$ ，但极差会相当大，且无泛化能力

## 相对编码

在区间  $[0, 1]$  中，给每个单词分配一个浮点数，但不同长度语句中单词差异不一致

## 理想编码

- 在长度不同的语句中，任意两个单词之间的距离应保持一致

$$f(pos + 1) - f(pos) = f(pos) - f(pos - 1)$$

- 模型能泛化到更长的句子上，且数值要有限制
- 位置编码必须是确定性的，相同的单词，产生相同的编码



# 位置编码

## 二进制编码

从 0 到 15 共 4 列，每列以不同的周期依次出现 0-1

0:	0	0	0	0	8:	1	0	0	0
1:	0	0	0	1	9:	1	0	0	1
2:	0	0	1	0	10:	1	0	1	0
3:	0	0	1	1	11:	1	0	1	1
4:	0	1	0	0	12:	1	1	0	0
5:	0	1	0	1	13:	1	1	0	1
6:	0	1	1	0	14:	1	1	1	0
7:	0	1	1	1	15:	1	1	1	1

图 5:  $T_{c1} = 8$ ,  $T_{c2} = 4$ ,  $T_{c3} = 2$ ,  $T_{c4} = 1$



# 位置编码

## 三角编码公式

$$PE(pos, 2i) = \sin(pos \div 10000^{2i \div d_{model}})$$

$$PE(pos, 2i + 1) = \cos(pos \div 10000^{2i \div d_{model}})$$

Positional Encoding Matrix with  $d=4, n=100$

Sequence	Index of token, $k$	$i=0$	$i=0$	$i=1$	$i=1$
I	0	$P_{00}=\sin(0)$ = 0	$P_{01}=\cos(0)$ = 1	$P_{02}=\sin(0)$ = 0	$P_{03}=\cos(0)$ = 1
am	1	$P_{10}=\sin(1/1)$ = 0.84	$P_{11}=\cos(1/1)$ = 0.54	$P_{12}=\sin(1/10)$ = 0.10	$P_{13}=\cos(1/10)$ = 1.0
a	2	$P_{20}=\sin(2/1)$ = 0.91	$P_{21}=\cos(2/1)$ = -0.42	$P_{22}=\sin(2/10)$ = 0.20	$P_{23}=\cos(2/10)$ = 0.98
Robot	3	$P_{30}=\sin(3/1)$ = 0.14	$P_{31}=\cos(3/1)$ = -0.99	$P_{32}=\sin(3/10)$ = 0.30	$P_{33}=\cos(3/10)$ = 0.96

Positional Encoding Matrix for the sequence 'I am a robot'

图 6: Positional Encoding



# 研究结论

## Transformer 优点

- 摒弃 RNN 和 CNN 网络，新的架构设计巧妙
- 引用 Attention 运算
- 引入单词位置运算，任意两个单词距离始终是一个单位
- Transformer 模型可并行计算
- 新架构具有启示意义：编码器的 BERT，解码器的 GPT，视觉图像处理的 ViT

## Transformer 缺点

- 沿袭传统学习的套路，全连接层 + 注意力层
- 完全摒弃 RNN 和 CNN 后，缺乏捕捉局部特征的能力
- 勉强以 Positional Encoding 弥补词向量位置信息的缺失



Thanks for Listening. 😊✓