**体验大模型**

对于attention机制，我用一下提示词“请你写一段html，可视化地解释attention机制”分别给了不同的AI，给出的结果有些差异。

对于attention机制的可视化解释，Deepseek给出的代码为“attention\_by\_deepseek.html”运行后，发现页面中显示的内容难以被用户通俗地理解，虽然可以交互，但是所传达的信息并没有那么明确，看完之后还是对attention机制没有一个明确的认识。

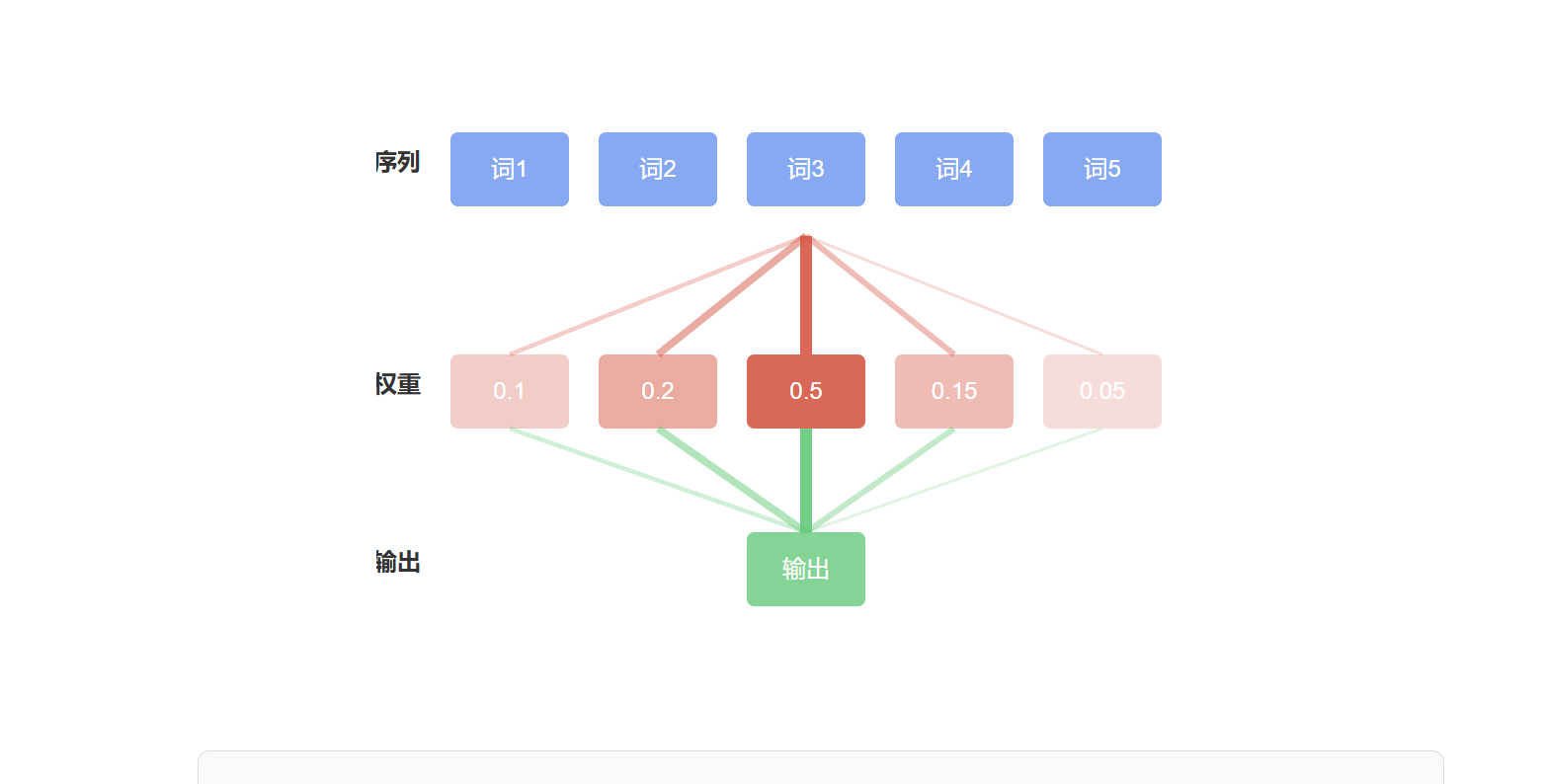


ChatGPT给出的代码为“attention\_by\_gpt.html”，所生成的页面不能与用户交互，仅仅包含简单的文字，并不能让用户理解attention机制。



Grok3给出的代码为“attention\_by\_grok3.html”，运行之后页面包含的信息量要多于其他两个AI所生成的页面，同时也包含用户交互模块。相比较与前两个AI所给出的页面，grok3完成的相对不错。







从以上三个大模型对于网页生成的任务体验来看，大模型之间有差异，想要让大模型高效地运作，产出符合用户需求的内容，对于用户的输入也有一些要求。在使用过程中，用户的输入也影响着生成的质量。

在这之后，我使用deepseek去学习attention机制。首先我先输入“我现在正在学习attention机制，请你现为我通俗地讲解一下，“，等待deepseek输出后，又输入”现在有些明白了，请你详细解释一下技术细节“ 在使用过程中，通过ai用循序渐进、先通俗解释，后复杂解释的方式来理解学习新知识会比较适合。

以下是我使用deepseek学习attention机制的过程。

我：我现在正在学习attention机制，请你现为我通俗地讲解一下，

Deepseek：好的，我来用最通俗的方式解释「注意力机制」（Attention Mechanism），就像给朋友讲故事一样！

---

### 举个现实中的例子 🌟

想象你在读一本外文书，遇到一个复杂的长句子，里面有5个生词。你的大脑不会平均关注每一个词，而是会：

- \*\*聚焦重点\*\*：先找到最关键的2个词（比如主语和动词）

- \*\*忽略次要\*\*：快速扫过其他不重要的词（比如形容词或副词）

- \*\*动态调整\*\*：翻译到后半句时，回头再看某个之前忽略的词

这就是\*\*人脑的注意力\*\*——把有限的精力集中在最有用的信息上。

---

### 传统神经网络的缺陷 🤖

在深度学习里，早期的模型（如RNN）处理信息时有个问题：

- \*\*记忆力差\*\*：就像背课文时只能逐字硬背，遇到长句子会忘记开头

- \*\*平均主义\*\*：处理输入时对所有内容一视同仁（比如把"不重要的词"和"关键词"同等对待）

- \*\*结果\*\*：翻译长句子时容易出错，比如漏掉主语或搞错动词

---

### 注意力机制怎么解决？💡

\*\*核心思想\*\*：让AI学会「选择性关注」——像人类一样动态调整重点

#### 工作流程（类比手电筒）🔦

1. \*\*输入一句话\*\*：比如英文 "The cat sat on the mat"

2. \*\*准备手电筒\*\*：为要生成的每个中文词（如「猫」「坐在」「垫子」）准备一个专用手电筒

3. \*\*照亮关键区域\*\*：

- 生成「猫」时，手电筒照亮英文词 "The cat"

- 生成「垫子」时，手电筒移到 "on the mat"

4. \*\*动态调整亮度\*\*：重要区域高亮（权重0.9），次要区域调暗（权重0.05）

---

### 技术实现的比喻 🛠️

想象你在网购时对比三款手机：

1. \*\*查询（Query）\*\*：你的需求（比如「续航强+便宜」）

2. \*\*键（Key）\*\*：每款手机的属性标签（「续航」「价格」「拍照」）

3. \*\*值（Value）\*\*：手机的具体参数

4. \*\*计算匹配度\*\*：找出和你需求最匹配的手机（比如续航权重0.7，价格权重0.3）

注意力机制就是帮AI完成这个「匹配-加权」的过程！

---

### 实际应用场景 🚀

1. \*\*机器翻译\*\*：生成「垫子」时关注英文"mat"而不是"cat"

2. \*\*图片描述\*\*：描述「鸟」时聚焦图片中的翅膀区域

3. \*\*语音识别\*\*：识别「发音模糊的单词」时参考上下文

4. \*\*推荐系统\*\*：根据你最近的行为动态调整推荐权重

---

### 为什么这么火？ 🔥

1. \*\*解决长距离依赖\*\*：轻松记住「50个词前的关键名词」

2. \*\*动态灵活\*\*：每个输出步骤都能重新调整关注点

3. \*\*可解释性强\*\*：通过权重分布看到AI的关注重点（比如医疗AI诊断时关注了CT片的哪个区域）

---

希望这个解释能让你对注意力机制有直观感受！如果想了解技术细节（比如QKV矩阵计算），可以随时问我哦~ 😊

我：现在有些明白了，请你详细解释一下技术细节

Deepseek：好的！我们来拆解注意力机制（尤其是Transformer中的Self-Attention）的技术细节，尽量用直观的数学和代码逻辑解释。以下是分步骤的详细说明：

---

### \*\*1. 核心组件：Query, Key, Value (Q, K, V)\*\*

注意力机制的核心是三个向量：\*\*Query（查询）\*\*、\*\*Key（键）\*\*、\*\*Value（值）\*\*。它们由输入序列通过线性变换生成。

- \*\*输入\*\*：一个序列的向量（例如词嵌入），形状为 `[序列长度, 嵌入维度]`（如 `[n, d\_model]`）。

- \*\*线性变换\*\*：通过三个独立的权重矩阵 \( W^Q \)、\( W^K \)、\( W^V \)，将输入转换为 Q、K、V：

\[

Q = X \cdot W^Q, \quad K = X \cdot W^K, \quad V = X \cdot W^V

\]

- 其中 \( W^Q, W^K, W^V \) 是可学习的参数矩阵。

#### \*\*直观理解\*\*：

- \*\*Query\*\*：当前要关注的位置（比如生成中文词「猫」时，需要查询英文中的哪个词）。

- \*\*Key\*\*：所有位置的“标签”，用于和 Query 匹配。

- \*\*Value\*\*：实际携带信息的向量，加权后得到输出。

---

### \*\*2. 计算注意力得分（Attention Scores）\*\*

注意力得分衡量 \*\*Query\*\* 和 \*\*Key\*\* 之间的相关性。

- \*\*点积计算\*\*：对每个 Query 和所有 Key 做点积（内积），得到一个分数矩阵：

\[

\text{Scores} = Q \cdot K^T

\]

- 形状为 `[n, n]`（n 是序列长度）。

- \*\*缩放（Scaling）\*\*：为防止点积结果过大导致梯度消失，除以 \( \sqrt{d\_k} \)（\( d\_k \) 是 Key 的维度）：

\[

\text{Scores} = \frac{Q \cdot K^T}{\sqrt{d\_k}}

\]

#### \*\*直观理解\*\*：

- 点积越大，表示 Query 和 Key 的相似度越高。

- 缩放是为了让梯度更稳定（尤其在维度较高时）。

---

### \*\*3. Softmax 归一化权重\*\*

将得分转化为概率分布（权重和为1）：

\[

\text{Attention Weights} = \text{Softmax}(\text{Scores})

\]

- Softmax 沿着最后一个维度（每行的权重和为1）。

#### \*\*直观理解\*\*：

- 如果生成中文词「猫」时，英文词 "cat" 的权重接近1，其他词权重接近0。

---

### \*\*4. 加权求和得到输出\*\*

用权重对 Value 向量加权求和：

\[

\text{Output} = \text{Attention Weights} \cdot V

\]

- 形状为 `[n, d\_v]`（与 Value 的维度相同）。

#### \*\*直观理解\*\*：

- 输出是当前步骤最需要关注的信息的聚合。

---

### \*\*5. 多头注意力（Multi-Head Attention）\*\*

为了捕捉不同角度的信息，使用多个“注意力头”：

1. \*\*拆分 Q, K, V\*\*：将 Q、K、V 沿嵌入维度切分为 `h` 个头（如 `h=8`）。

- 例如，原维度 `d\_model=512` → 每个头维度 `d\_k=64`。

2. \*\*每个头独立计算注意力\*\*：

\[

\text{head}\_i = \text{Attention}(Q\_i, K\_i, V\_i)

\]

3. \*\*合并结果\*\*：将所有头的输出拼接，再通过线性层融合：

\[

\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}\_1, ..., \text{head}\_h) \cdot W^O

\]

- \( W^O \) 是输出的可学习参数矩阵。

#### \*\*直观理解\*\*：

- 不同的头可以关注不同模式（例如一个头关注句法，另一个头关注语义）。

---

### \*\*6. 位置编码（Positional Encoding）\*\*

自注意力（Self-Attention）本身不考虑序列顺序，因此需要显式添加位置信息：

- \*\*正弦和余弦函数编码位置\*\*：

\[

PE\_{(pos, 2i)} = \sin\left(\frac{pos}{10000^{2i/d\_{\text{model}}}}\right)

\]

\[

PE\_{(pos, 2i+1)} = \cos\left(\frac{pos}{10000^{2i/d\_{\text{model}}}}\right)

\]

- \*\*输入增强\*\*：将位置编码加到输入嵌入中：

\[

X = X\_{\text{embed}} + PE

\]

#### \*\*直观理解\*\*：

- 位置编码让模型知道每个词的位置（例如第一个词 vs 第五个词）。

---

### \*\*完整流程总结（Self-Attention）\*\*

1. \*\*输入嵌入\*\*：将词转换为向量，并添加位置编码。

2. \*\*生成 Q, K, V\*\*：通过线性变换得到三个矩阵。

3. \*\*计算注意力得分\*\*：Q 和 K 的点积，缩放后 Softmax。

4. \*\*加权求和\*\*：用权重对 V 加权，得到输出。

5. \*\*多头扩展\*\*：重复多次后合并结果。

---

### \*\*代码示例（PyTorch风格伪代码）\*\*

```python

import torch

import torch.nn.functional as F

def scaled\_dot\_product\_attention(Q, K, V, mask=None):

d\_k = Q.size(-1)

scores = torch.matmul(Q, K.transpose(-2, -1)) / torch.sqrt(torch.tensor(d\_k))

if mask is not None:

scores = scores.masked\_fill(mask == 0, -1e9) # 处理填充（padding）

weights = F.softmax(scores, dim=-1)

output = torch.matmul(weights, V)

return output, weights

# 多头注意力

class MultiHeadAttention(torch.nn.Module):

def \_\_init\_\_(self, d\_model, num\_heads):

super().\_\_init\_\_()

self.d\_model = d\_model

self.num\_heads = num\_heads

self.d\_k = d\_model // num\_heads

# 定义线性变换层

self.W\_Q = torch.nn.Linear(d\_model, d\_model)

self.W\_K = torch.nn.Linear(d\_model, d\_model)

self.W\_V = torch.nn.Linear(d\_model, d\_model)

self.W\_O = torch.nn.Linear(d\_model, d\_model)

def forward(self, Q, K, V, mask=None):

# 分头处理

Q = self.W\_Q(Q).view(batch\_size, -1, self.num\_heads, self.d\_k).transpose(1, 2)

K = self.W\_K(K).view(batch\_size, -1, self.num\_heads, self.d\_k).transpose(1, 2)

V = self.W\_V(V).view(batch\_size, -1, self.num\_heads, self.d\_k).transpose(1, 2)

# 计算注意力

attn\_output, attn\_weights = scaled\_dot\_product\_attention(Q, K, V, mask)

# 合并多头输出

attn\_output = attn\_output.transpose(1, 2).contiguous().view(batch\_size, -1, self.d\_model)

output = self.W\_O(attn\_output)

return output, attn\_weights

```

---

### \*\*关键点总结\*\*

1. \*\*QKV的本质\*\*：通过线性变换从输入中提取不同角色。

2. \*\*缩放点积\*\*：防止梯度消失，提高训练稳定性。

3. \*\*Softmax权重\*\*：动态聚焦重要部分。

4. \*\*多头机制\*\*：捕捉多样化的依赖关系。

5. \*\*位置编码\*\*：弥补自注意力缺乏位置感知的缺陷。

如果有任何步骤不清楚，或者想了解某个部分的具体数学推导，可以随时问我！ 😊

我：能不能举个示例来示范线性变化中的计算过程

Deepseek：好的！我来用一个\*\*极简的数值例子\*\*，手动计算线性变换生成 Q、K、V 的过程。假设我们的输入非常简单，以便你能直观看到每一步的数学操作。

---

### \*\*示例设定\*\*

假设：

- \*\*输入序列\*\*：两个词（比如 "I" 和 "love"），每个词的嵌入向量维度是3。

- \*\*输入矩阵 X\*\*：

\[

X = \begin{bmatrix}

1 & 0.5 & 2 \\ # "I" 的嵌入向量

0.8 & 1.2 & 0.3 # "love" 的嵌入向量

\end{bmatrix}

\]

形状为 `[2, 3]`（2个词，每个词3维）。

- \*\*权重矩阵\*\*（人为设定数值，实际中由模型学习）：

- \( W^Q = \begin{bmatrix} 0.1 & 0.2 \\ 0.3 & 0.4 \\ 0.5 & 0.6 \end{bmatrix} \) （形状 `[3, 2]`）

- \( W^K = \begin{bmatrix} 0.7 & 0.8 \\ 0.9 & 1.0 \\ 1.1 & 1.2 \end{bmatrix} \) （形状 `[3, 2]`）

- \( W^V = \begin{bmatrix} 1.3 & 1.4 \\ 1.5 & 1.6 \\ 1.7 & 1.8 \end{bmatrix} \) （形状 `[3, 2]`）

---

### \*\*Step 1: 计算 Query (Q)\*\*

\[

Q = X \cdot W^Q

\]

具体计算：

1. \*\*第一个词 "I" 的 Query\*\*（X的第一行）：

\[

Q\_1 = [1, 0.5, 2] \cdot \begin{bmatrix} 0.1 & 0.2 \\ 0.3 & 0.4 \\ 0.5 & 0.6 \end{bmatrix}

\]

\[

Q\_1 = (1×0.1 + 0.5×0.3 + 2×0.5, \quad 1×0.2 + 0.5×0.4 + 2×0.6)

\]

\[

Q\_1 = (0.1 + 0.15 + 1.0, \quad 0.2 + 0.2 + 1.2) = [1.25, 1.6]

\]

2. \*\*第二个词 "love" 的 Query\*\*（X的第二行）：

\[

Q\_2 = [0.8, 1.2, 0.3] \cdot \begin{bmatrix} 0.1 & 0.2 \\ 0.3 & 0.4 \\ 0.5 & 0.6 \end{bmatrix}

\]

\[

Q\_2 = (0.8×0.1 + 1.2×0.3 + 0.3×0.5, \quad 0.8×0.2 + 1.2×0.4 + 0.3×0.6)

\]

\[

Q\_2 = (0.08 + 0.36 + 0.15, \quad 0.16 + 0.48 + 0.18) = [0.59, 0.82]

\]

\*\*最终 Q 矩阵\*\*：

\[

Q = \begin{bmatrix} 1.25 & 1.6 \\ 0.59 & 0.82 \end{bmatrix}

\]

---

### \*\*Step 2: 计算 Key (K)\*\*

\[

K = X \cdot W^K

\]

计算过程类似，直接给出结果：

1. \*\*第一个词 "I" 的 Key\*\*：

\[

K\_1 = [1, 0.5, 2] \cdot \begin{bmatrix} 0.7 & 0.8 \\ 0.9 & 1.0 \\ 1.1 & 1.2 \end{bmatrix}

\]

\[

K\_1 = (1×0.7 + 0.5×0.9 + 2×1.1, \quad 1×0.8 + 0.5×1.0 + 2×1.2)

\]

\[

K\_1 = (0.7 + 0.45 + 2.2, \quad 0.8 + 0.5 + 2.4) = [3.35, 3.7]

\]

2. \*\*第二个词 "love" 的 Key\*\*：

\[

K\_2 = [0.8, 1.2, 0.3] \cdot \begin{bmatrix} 0.7 & 0.8 \\ 0.9 & 1.0 \\ 1.1 & 1.2 \end{bmatrix}

\]

\[

K\_2 = (0.8×0.7 + 1.2×0.9 + 0.3×1.1, \quad 0.8×0.8 + 1.2×1.0 + 0.3×1.2)

\]

\[

K\_2 = (0.56 + 1.08 + 0.33, \quad 0.64 + 1.2 + 0.36) = [1.97, 2.2]

\]

\*\*最终 K 矩阵\*\*：

\[

K = \begin{bmatrix} 3.35 & 3.7 \\ 1.97 & 2.2 \end{bmatrix}

\]

---

### \*\*Step 3: 计算 Value (V)\*\*

\[

V = X \cdot W^V

\]

同样直接计算：

1. \*\*第一个词 "I" 的 Value\*\*：

\[

V\_1 = [1, 0.5, 2] \cdot \begin{bmatrix} 1.3 & 1.4 \\ 1.5 & 1.6 \\ 1.7 & 1.8 \end{bmatrix}

\]

\[

V\_1 = (1×1.3 + 0.5×1.5 + 2×1.7, \quad 1×1.4 + 0.5×1.6 + 2×1.8)

\]

\[

V\_1 = (1.3 + 0.75 + 3.4, \quad 1.4 + 0.8 + 3.6) = [5.45, 5.8]

\]

2. \*\*第二个词 "love" 的 Value\*\*：

\[

V\_2 = [0.8, 1.2, 0.3] \cdot \begin{bmatrix} 1.3 & 1.4 \\ 1.5 & 1.6 \\ 1.7 & 1.8 \end{bmatrix}

\]

\[

V\_2 = (0.8×1.3 + 1.2×1.5 + 0.3×1.7, \quad 0.8×1.4 + 1.2×1.6 + 0.3×1.8)

\]

\[

V\_2 = (1.04 + 1.8 + 0.51, \quad 1.12 + 1.92 + 0.54) = [3.35, 3.58]

\]

\*\*最终 V 矩阵\*\*：

\[

V = \begin{bmatrix} 5.45 & 5.8 \\ 3.35 & 3.58 \end{bmatrix}

\]

---

### \*\*总结\*\*

- \*\*输入 X\*\*：2个词，每个词3维。

- \*\*线性变换后\*\*：

- Q: `[2, 2]`

- K: `[2, 2]`

- V: `[2, 2]`

通过不同的权重矩阵 \( W^Q, W^K, W^V \)，输入 X 被投影到不同的空间，分别承担“查询”、“键”、“值”的角色。模型通过训练学习这些权重矩阵，使 Q、K、V 能有效捕捉输入之间的关系。

---

### \*\*代码验证\*\*

用 PyTorch 验证上述计算：

```python

import torch

# 输入 X

X = torch.tensor([[1.0, 0.5, 2.0], [0.8, 1.2, 0.3]])

# 定义权重矩阵

WQ = torch.tensor([[0.1, 0.2], [0.3, 0.4], [0.5, 0.6]])

WK = torch.tensor([[0.7, 0.8], [0.9, 1.0], [1.1, 1.2]])

WV = torch.tensor([[1.3, 1.4], [1.5, 1.6], [1.7, 1.8]])

# 计算 Q, K, V

Q = torch.matmul(X, WQ)

K = torch.matmul(X, WK)

V = torch.matmul(X, WV)

print("Q:\n", Q)

print("K:\n", K)

print("V:\n", V)

```

输出结果应与手动计算一致：

```

Q:

tensor([[1.2500, 1.6000],

[0.5900, 0.8200]])

K:

tensor([[3.3500, 3.7000],

[1.9700, 2.2000]])

V:

tensor([[5.4500, 5.8000],

[3.3500, 3.5800]])

```

这个例子清晰地展示了线性变换如何生成 Q、K、V。实际中，这些权重矩阵会通过训练自动优化，使模型学会关注重要的信息。

**Transformer**

Transformer的核心为自注意力机制，通过自注意力机制，计算序列中所有词的关系权重。

核心组件包括embedding，positional encoding，self-attention，free-forward network，residual connection ，LayerNorm等。词嵌入+位置编码将离散符号转换为向量并添加位置信息。编码器层提取输入序列的全局上下文特征。解码器层基于编码器输出和已生成内容预测下一个词，输出层生成词表概率分布。

对于transformer，主要应用于生成文本，翻译，文本分类，问答，或者是跨模态的图像处理与语音识别。

**Attention机制**

Attention机制模拟了人类注意力的分配方式，会给不同的词不同的权重，自动发现不同信息片段之间的相关性。它会计算query与key的匹配程度，通过softmax函数转化为概率分布，并且放大差异。用权重对Value加权求和，得到输出。对于多头注意力机制，将Q，K，V分成多个头，每个头独立计算注意力，最后将所有头的输出拼接，融合。