

Transformer的一家！

Lilian 炼丹笔记 5天前

↑↑↑关注后"星标"炼丹笔记

炼丹笔记干货

作者：Lilian

Transformer Family

01 准备

Symbol	Meaning
d	The model size / hidden state dimension / positional encoding size.
h	The number of heads in multi-head attention layer.
L	The segment length of input sequence.
$\mathbf{X} \in \mathbb{R}^{L \times d}$	The input sequence where each element has been mapped into an embedding vector of shape d , same as the model size.
$\mathbf{W}^k \in \mathbb{R}^{d \times d_k}$	The key weight matrix.
$\mathbf{W}^q \in \mathbb{R}^{d \times d_k}$	The query weight matrix.
$\mathbf{W}^v \in \mathbb{R}^{d \times d_v}$	The value weight matrix. Often we have $d_k = d_v = d$.
$\mathbf{W}_i^k, \mathbf{W}_i^q \in \mathbb{R}^{d \times d_k/h}; \mathbf{W}_i^v \in \mathbb{R}^{d \times d_v/h}$	The weight matrices per head.
$\mathbf{W}^o \in \mathbb{R}^{d_v \times d}$	The output weight matrix.
$\mathbf{Q} = \mathbf{XW}^q \in \mathbb{R}^{L \times d_k}$	The query embedding inputs.
$\mathbf{K} = \mathbf{XW}^k \in \mathbb{R}^{L \times d_k}$	The key embedding inputs.
$\mathbf{V} = \mathbf{XW}^v \in \mathbb{R}^{L \times d_v}$	The value embedding inputs.
S_i	A collection of key positions for the i -th query \mathbf{q}_i to attend to.
$\mathbf{A} \in \mathbb{R}^{L \times L}$	The self-attention matrix between a input sequence of lenght L and itself. $\mathbf{A} = \text{softmax}(\mathbf{QK}^\top / \sqrt{d_k})$.
$a_{ij} \in \mathbf{A}$	The scalar attention score between query \mathbf{q}_i and key \mathbf{k}_j .
$\mathbf{P} \in \mathbb{R}^{L \times d}$	position encoding matrix, where the i -th row \mathbf{p}_i is the positional encoding for input \mathbf{x}_i .

02 Attention 以及 Self-Attention

1.Attention

- 是神经网络中的一种机制：**模型可以通过选择性地关注给定的数据集来学习做出预测**。Attention的个数是通过学习权重来量化的，输出则通常是一个加权平均值。

2.Self-Attention：

- 是一种注意机制，模型利用对同一样本观测到的其他部分来对数据样本的剩下部分进行预测。从概念上讲，它感觉非常类似于non-local的方式。还要注意的，Self-attention是置换不变的；换句话说，它是对集合的一种操作。

而关于attention和self-attention存在非常多的形式，我们之前常见的Transformer是依赖于scaled-dot-product的形式，也就是：给定query矩阵Q, key矩阵K以及value矩阵V，那么我们的输出就是值向量的加权和，其中，分配给每个值槽的权重由Query与相应Key的点积确定。

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

对于一个query以及一个key向量, $q_i, k_j \in R^d$, 我们计算下面的值:

$$a_{ij} = softmax(\frac{q_i k_j^T}{\sqrt{d_k}}) = \frac{exp(q_i k_j^T)}{\sqrt{d_k} \sum_{r \in S_i} exp(q_i k_r^T)}$$

其中, S_i 是keys的集合。

03

Multi-Head Self-Attention

multi-head self-attention是Transformer的核心组成部分，和简单的attention不同之处在于，Multihead机制将输入拆分为许多小的chunks，然后并行计算每个子空间的scaled dot product，最后我们将所有的attention输出进行拼接，

$$MultiheadAttention(X_q, X_k, X_v) = [head_1; \dots, head_h] W^o \quad head_i = Attention(X_q W_i^q, X_k W_i^k)$$

其中, $[.;.]$ 是concat操作, $W_i^q, W_i^k \in R^{d \times d_k/h}, W_i^v \in R^{d \times d_v/h}$ 是权重矩阵，它将我们的输出embeddings(L*d)的映射到query,key,value矩阵，而且 $W^o \in R^{d_v \times d}$ 是输出的线性转化，这些权重都是在训练的时候进行训练的。

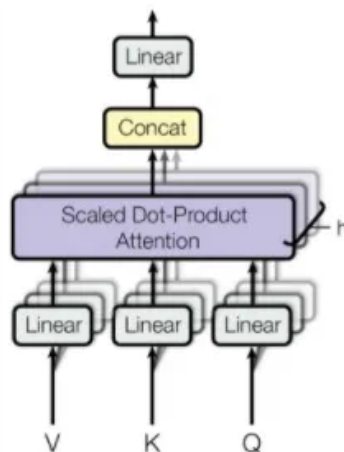


Fig. 1. Illustration of the multi-head scaled dot-product attention mechanism.
(Image source: Figure 2 in Vaswani, et al., 2017)

04

Transformer

Transformer，很多时候我们也称之为"vanilla Transformer"，它有一个encoder-decoder的结构，decoder的Transformer可以在语言建模的时候获得非常好的效果

Encoder-Decoder结构

Encoder生成一个基于attention的表示，能够从一个大的上下文中定位一个特定的信息片段。它由6个身份识别模块组成，每个模块包含两个子模块、一个multihead self-attention和一个point-wise全连接前馈网络。

按point-wise来说，这意味着它对序列中的每个元素应用相同的线性变换（具有相同的权重）。**这也可以看作是滤波器大小为1的卷积层**。每个子模块都有一个剩余连接和layer normalization。所有子模块输出相同维度 d 的数据。

Transformer的decoder功能是从encoder的表示中抽取信息。该结构与encoder非常相似，只是decoder包含两个多头注意子模块，而不是在每个相同的重复模块中包含一个。第一个多头注意子模块被屏蔽，以防止位置穿越。

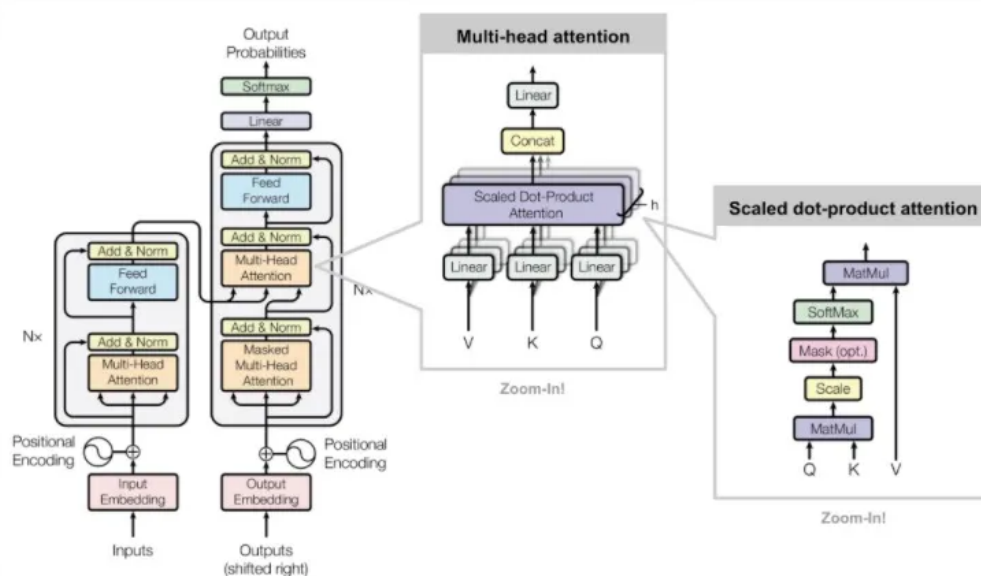


Fig. 2. The architecture of the vanilla Transformer model. (Image source: [Figure 17](#))

Positional Encoding

因为self-attention操作是permutation不变的，所以使用正确的位置编码是非常重要的，此处我们使用如下的位置编码来提供order信息，位置编码 $P \in R^{L \times d}$ ，我们可以直接将它们加入到我们的vanilla Transformer中，

- (1). Sinusoidal positional encoding, 给定token的位置 $i = 1, \dots, L$, 维度 $\delta = 1, 2, \dots, d$

$$PE(i, \delta) = \sin\left(\frac{i}{10000^{2\delta/d}}\right), \text{ if } \delta = 2\delta' \quad PE(i, \delta) = \cos\left(\frac{i}{10000^{2\delta/d}}\right), \text{ if } \delta = 2\delta' + 1$$

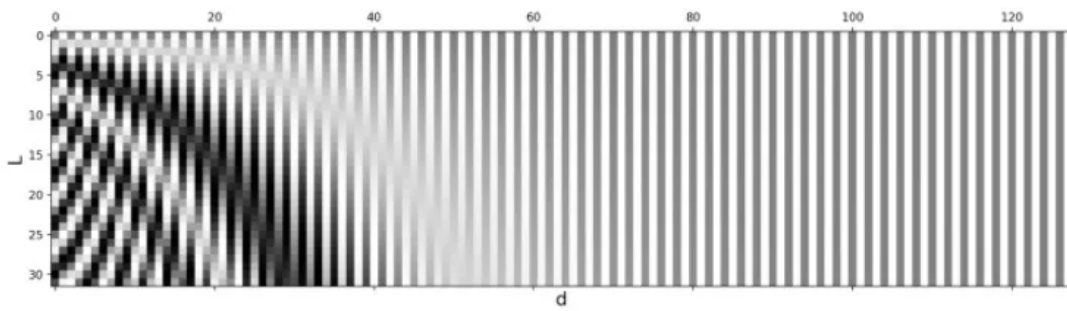


Fig. 3. Sinusoidal positional encoding with $L = 32$ and $d = 128$. The value is between -1 (black) and 1 (white) and the value 0 is in gray.

- (2). Learned positional encoding, 对每个学习得到对列向量, 对每个绝对位置进行编码。

辅助Loss

为了取得更好的效果, 我们一般会加入辅助loss,

- 除了在序列末尾只生成一个预测之外, 还要求每个immediatge位置能做出正确的预测, 迫使模型预测给定的较小上下文(例如, 上下文窗口开头的前几个tokens)。
- 每个中间Transformer也用于进行预测。随着训练的进行, 较低层的权重对总损失的贡献越来越小。
- 序列中的每个位置可以预测多个目标, 即, 对未来token的两个或多个预测。

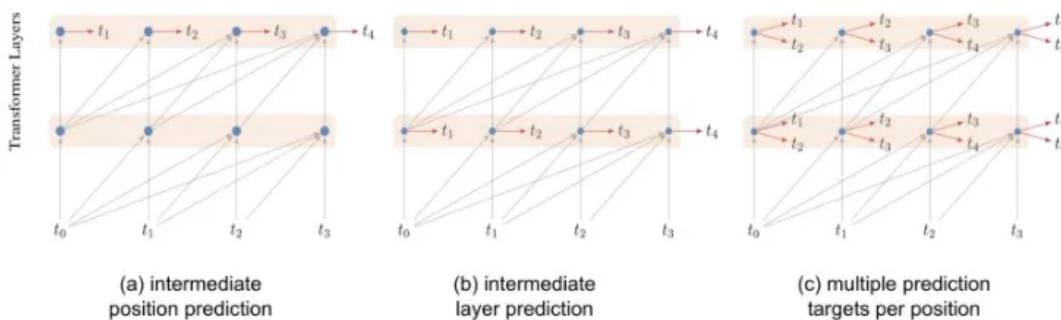


Fig. 4. Auxiliary prediction tasks used in deep Transformer for character-level language modeling. (Image source: [Al-Rfou et al. \(2018\)](#))

05

ACT(Adaptive Computation Time)

Adaptive Computation Time是一种动态决定递归神经网络需要多少计算步骤的机制。

我们有一个RNN模型 R , 它由输入的权重 W_x , 一个参数化的状态迁移函数 $S(\cdot)$ 一个输出权重 W_y 和一个输出的bias b_y 组成。给定一个输入序列 (x_1, \dots, x_L) , 输出的序列 (y_1, y_2, \dots, y_L) 是由:

$$s_t = S(s_{t-1}, W_x x_t), y_t = W_y s_t + b_y, \text{ for } t = 1, \dots, L$$

ACT使上述RNN设置能够在每个输入元素上执行数量可变的步骤。大量的计算步骤会导致中间状态序列 $(s_t^1, \dots, s_t^{N(t)})$ 并且输出 $(y_t^1, \dots, y_t^{N(t)})$, 它们都共享相同的迁移状态函数 $S(\cdot)$, 以及相同的输出权重 W_y, b_y :

$$s_t^0 = s_{t-1}^n = S(s_t^{n-1}, x_t^n) = S(s_t^{n-1}, x_t + \delta_{n,1}) \text{ for } n = 1, \dots, N(t) \quad y_t^n = W_y s_t^n + b_y$$

其中 $\delta_{n,1}$ 是一个二元的flag, 来表示是否输入步是递增的。

step的个数 $N(t)$ 是由额外的sigmoidal halting单元 h 决定的, 带有相关的权重矩阵 W_h 以及bias b_h , 对于第 t 输入元素在中间步骤 n 处输出一个中止概率 p_t^n :

$$h_t^n = \sigma(W_h s_t^n + b_h)$$

为了使计算在一个步骤后停止, ACT引入了一个小常数 ϵ (例如0.01), 因此每当累积概率超过 $1 - \epsilon$ 时, 计算就会停止。

$$N(t) = \min(\min\{n' : \sum_{n=1}^{n'} h_t^n \geq 1 - \epsilon\}, M) \quad p_t^n = h_t^n, \text{ if } n < N(t), p_t^n = R(t) = 1 - \sum_{n=1}^{N(t)-1} h_t^n$$

其中 M 为中间步骤个数的上限。

最终状态和输出的mean-field的update:

$$s_t = \sum_{n=1}^{N(t)} p_t^n s_t^n, y_t = \sum_{n=1}^{N(t)} p_t^n y_t^n$$

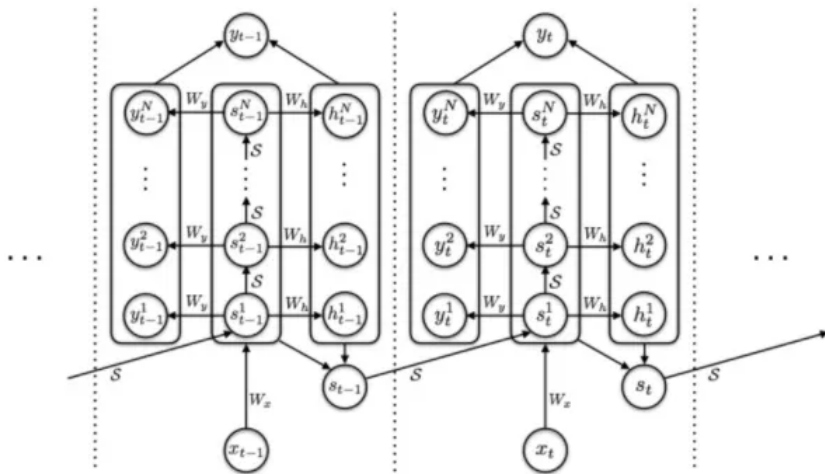


Fig. 5. The computation graph of a RNN with ACT mechanism. (Image source: Graves, 2016)

避免对每个输入进行不必要的思考, ACT增加了ponder cost

$$P(x) = \sum_{t=1}^L N(t) + R(t)$$

用此来鼓励中间计算步骤的小的次数。

06

Improved Attention Span

提高Attention Span的目的是使可用于self-attention的上下文更长、更有效、更灵活。

1. Longer Attention Span(Transformer-XL)

vanilla Transformer有一个**固定的和有限的注意广度**。在每个更新步骤中，该模型只能处理同一段中的其他元素，并且没有任何信息可以在分离的固定长度段之间流动。**也就是说层数固定不够灵活，同时对于算力需求非常大，导致其并不适合处理超长序列。**

这种context segmentation会导致几个问题：

- 模型不能捕获非常长期的依赖关系；
- 在没有上下文或上下文很薄的情况下，很难预测每个片段中的前几个tokens。
- 评估是昂贵的。每当segment右移一位时，新的segment就会从头开始重新处理，尽管有很多重叠的tokens。

Transformer-XL解决来上下文的segmentation问题：

- 对于segments之间的隐藏状态进行重复使用；
- 使用位置编码使其适用于重新使用的states；

Hidden state Reuse:

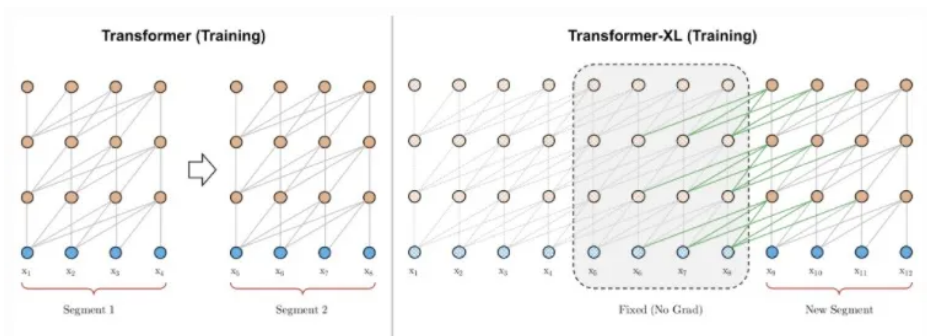


Fig. 6. A comparison between the training phrase of vanilla Transformer & Transformer-XL with a segment length 4. (Image source: left part of Figure 2 in Dai et al., 2019).

我们对第 n 层 $\tau + 1$ 的segment的隐藏状态打标签为 $h_{\tau+1}^{(n)} \in R^{L*d}$,除了对相同segment $h_{\tau+1}^{(n-1)}$ 的最后一层的英藏状态，我们还依赖于之前的segment $h_{\tau}^{(n)}$ 的相同层的隐藏状态。通过从前面隐藏状态加入信息，模型可以将attention的广度进行扩大，可以在多个segments之间发挥作用：

$$\bar{h}_{\tau+1}^{(n-1)} = [\text{stop} - \text{gradient}(h_{\tau}^{(n-1)} \odot h_{\tau+1}^{(n-1)})]Q_{\tau+1}^{(n)} = h_{\tau+1}^{(n-1)}W^qK_{\tau+1}^{(n)} = h_{\tau+1}^{(n-1)}W^kV_{\tau+1}^{(n)} = h_{\tau+1}^{(n)}$$

key和value依赖于扩展的隐藏状态，同时query仅仅只依赖于当前步的隐藏状态， $[\cdot \odot \cdot]$ 是序列长度的维度的concatenation操作。

2. Relative Positional Encoding

为了处理这种新的attention span的形式，Transformer-XL提出了一种新的位置编码。如果使用相同的方法对绝对位置进行编码，则前一段和当前段将分配相同的编码，这是不需要的。

为了保持位置信息流在各段之间的一致性，Transformer XL对相对位置进行编码，因为它足以知道位置的offset，从而做出更好的预测，即： $i - j$ ，在一个key向量 $k_{r,j}$ 以及它的query之间 $q_{r,i}$ 。

如果我们省略 $1/\sqrt{d_k}$ 并且对它们以softmax的形式进行normalize,我们可以重写在位置 i 的query和位置 j 的key之间的attention分数：

$$a_{ij} = q_i k_j^T = (x_i + p_i) W^q ((x_j + p_j) W^k)^T = x_i W^q W^{k^T} x_j^T + x_i W^q W^{k^T} p_j^T + p_i W^q W^{k^T} x_j^T$$

上面的几项又可以被表示为：

$$a_{ij}^{rel} = \underbrace{x_i W^q W_E^{k^T} x_j^T}_{\text{content-based addressing}} + \underbrace{x_i W^q W_R^{k^T} r_{j-1}^T}_{\text{content-dependent positional bias}} + \underbrace{u W_E^{k^T} x_j^T}_{\text{global content bias}} + \underbrace{v W_R^{k^T} r_{j-1}^T}_{\text{global positional bias}}$$

- 用相对位置编码 $r_{i-j} \in R^d$ 替换 p_j ;
- 用两个可训练的参数 u (针对内容)和 v (针对位置)替换 $p_i W^q$;
- 将 W^k 划分为两个矩阵, W_E^k 用于内容信息, W_R^k 用于位置信息;

3. Adaptive Attention Span

Transformer的一个关键优势是能够捕获长期依赖关系。根据上下文的不同，模型可能更愿意在某个时候比其他人更进一步地注意；或者一个attention head可能有不同于另一个attention head的注意模式。如果attention span能够灵活地调整其长度，并且只在需要时再往回看，这将有助于减少计算和内存开销，从而在模型中支持更长的最大上下文大小(这就是Adaptive Attention Span的动机)。

后来Sukhbaatar等人提出了一种self-attention机制以寻找最优的attention span，他们假设不同的attention heads可以在相同的上下文窗口中赋予不同的分数，因此最优的span可以被每个头分开训练。

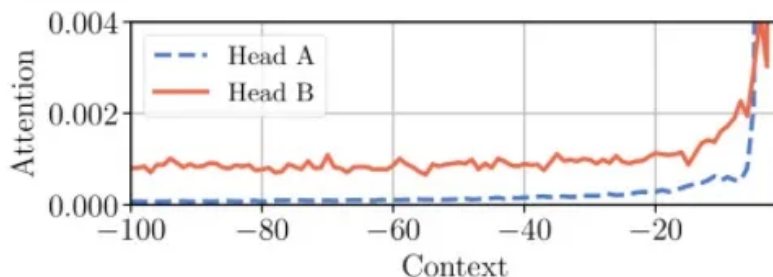


Fig. 7. Two attention heads in the same model, A & B, assign attention differently within the same context window. Head A attends more to the recent tokens, while head B look further back into the past uniformly. (Image source: [Sukhbaatar, et al. 2019](#))

给定第 i 个token,我们需要计算该token和其它在位置 $j \in S_i$ 的keys的attention权重，其中 S_i 定义了第 i 个token第上下文窗口：

$$e_{ij} = q_i k_j^T a_{ij} = \text{softmax}(e_{ij}) = \frac{\exp(e_{ij})}{\sum_{r=i-s}^{i-1} \exp(e_{ir})} y_i = \sum_{r=i-s}^{i-1} a_{ir} v_r = \sum_{r=i-s}^{i-1} a_{ir} x_r W^v$$

增加了一个soft mask函数 m_z 来控制有效的可调attention span, 它将query和key之间的距离映射成一个 $[0, 1]$ 值。 $m_z \in [0, s]$ 参数化, z 要学习:

$$m_z(x) = \text{clamp}\left(\frac{1}{R}(R + z - x), 0, 1\right)$$

其中 R 是一个超参数, 它可以定义 m_z 的softness:

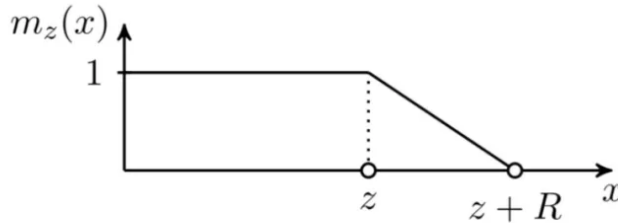


Fig. 8. The soft masking function used in the adaptive attention span. (Image source: Sukhbaatar, et al. 2019.)

soft mask函数应用于注意权重中的softmax元素:

$$a_{ij} = \frac{m_z(i-j) \exp(s_{ij})}{\sum_{r=i-s}^{i-1} m_z(i-r) \exp(s_{ir})}$$

在上面的等式, z 是可微的, 所以可以和模型的其它部分一起联合训练, 参数 $z^{(i)}$, $i = 1, 2, \dots, h$ 每个head可以分开学习, 此外, 损失函数有额外的 L_1 惩罚 $\sum_{i=1}^h z^{(i)}$.

利用Adaptive Computation Time, 该方法可以进一步增强attention span的长度, 动态地适应当前输入。attention head在时间 t 的跨度参数 z_t 是一个sigmoid函数, $z_t = S\sigma(v \cdot x_t + b)$, 其中向量 v 和偏置标量 b 与其他参数一起学习。

在具有自适应注意广度的Transformer实验中, Sukhbaatar等人发现了一个普遍趋势, 即较低层不需要很长的注意广度, 而较高层的一些attention heads可能会使用非常长的注意广度。适应性attention span有助于大大减少失败的次数, 特别是在一个有许多注意层和大上下文长度的大模型中。

4. Localized Attention Span (Image Transformer)

Transformer最初用于语言建模。文本序列是一维的, 具有明确的时间顺序, 因此attention span随着上下文大小的增加而线性增长。

然而, 如果我们想在图像上使用Transformer, 我们还不清楚如何定义上下文的范围或顺序。Image Transformer采用了一种图像生成公式, 类似于Transformer框架内的序列建模。此外, 图像Transformer将self-attention span限制在局部邻域内, 因此模型可以放大以并行处理更多的图像, 并保持可能性损失可控。

encoder-decoder架构保留用于image-conditioned生成:

- encoder生成源图像的上下文化的每像素信道表示;
- decoder自回归地生成输出图像, 每个时间步每像素一个通道。

让我们将要生成的当前像素的表示标记为查询 q 。其表示将用于计算 q 的其他位置是关键向量 k_1, k_2, \dots 它们一起形成一个内存矩阵 M 。 M 的范围定义了像素查询 q 的上下文窗口。

Image Transformer引入了两种类型的localized M , 如下所示。

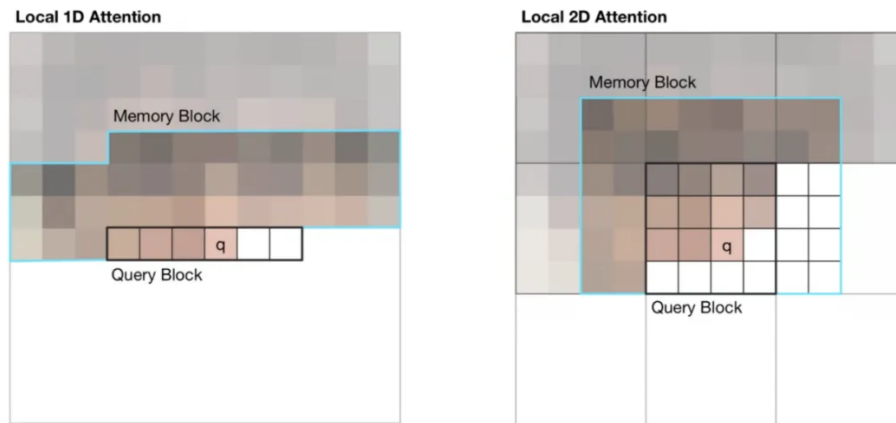


Fig. 9. Illustration of 1D and 2D attention span for visual inputs in Image Transformer. The black line marks a query block and the cyan outlines the actual attention span for pixel q . (Image source: Figure 2 in [Parmar et al, 2018](#))

- (1).1D Local Attention:输入图像按光栅扫描顺序(即从左到右、从上到下)展平。然后将线性化后的图像分割成不重叠的查询块。上下文窗口由与 q 相同的查询块中的像素和在此查询块之前生成的固定数量的附加像素组成。
- (2).2D Local Attention:图像被分割成多个不重叠的矩形查询块。查询像素可以处理相同内存块中的所有其他像素。为了确保左上角的像素也可以有一个有效的上下文窗口, 内存块将分别向上、左和右扩展一个固定的量。

07

Less Time and Memory Cost

如何减少计算和内存的消耗。

1. Sparse Attention Matrix Factorization (Sparse Transformers)

- 一般Transformer的计算和存储开销随序列长度呈二次增长, 因此很难应用于很长的序列。

Sparse Transformer

引入分解的self-attention, 通过稀疏矩阵分解, 我们可以将训练上百层的dense的attention网络, 这样序列长度就可以到达16384.

给定attention链接的模式集合 $S = \{S_1, \dots, S_n\}$, 其中 S_i 记录key位置的集合, 第 i 个query向量可以扩展为:

$$\text{Attend}(X, S) = (a(x_i, S_i))_{i \in \{1, \dots, L\}}$$

尽管 S_i 的size是不固定的, $a(x_i, S_i)$ 是size为 d_v 的, 因此, $\text{Attend}(X, S) \in R^{L \times d_v}$.

在自回归的模型中, 一个attention span被定义为 $S_i = \{j : j \leq i\}$, 它允许每个token可以处理过去的所有其它位置。

在分解的self-attention中, S_i 被分解为树的依赖, 例如对于没对 (i, j) , 其中 $j \leq i$, 存在一条路径链接 i 和 j 。

更加精确地说, 集合 S_i 被划分为 p 个non-overlapping的子集, 第 m 个子集被表示为

$A_i^{(m)} \subset S_i, m = 1, \dots, p$, 所以输出位置 i 和任意的 j 的路径有最大长度 $p + 1$, 例如, 如果 (j, a, b, c, \dots, i) 是 i 和 j 的索引路径, 我们有 $j \in A_a^{(1)}, a \in A_b^{(2)}, b \in A_c^{(3)} \dots$

Sparse Factorized Attention

Sparse Transformer提出了两类分解的attention,

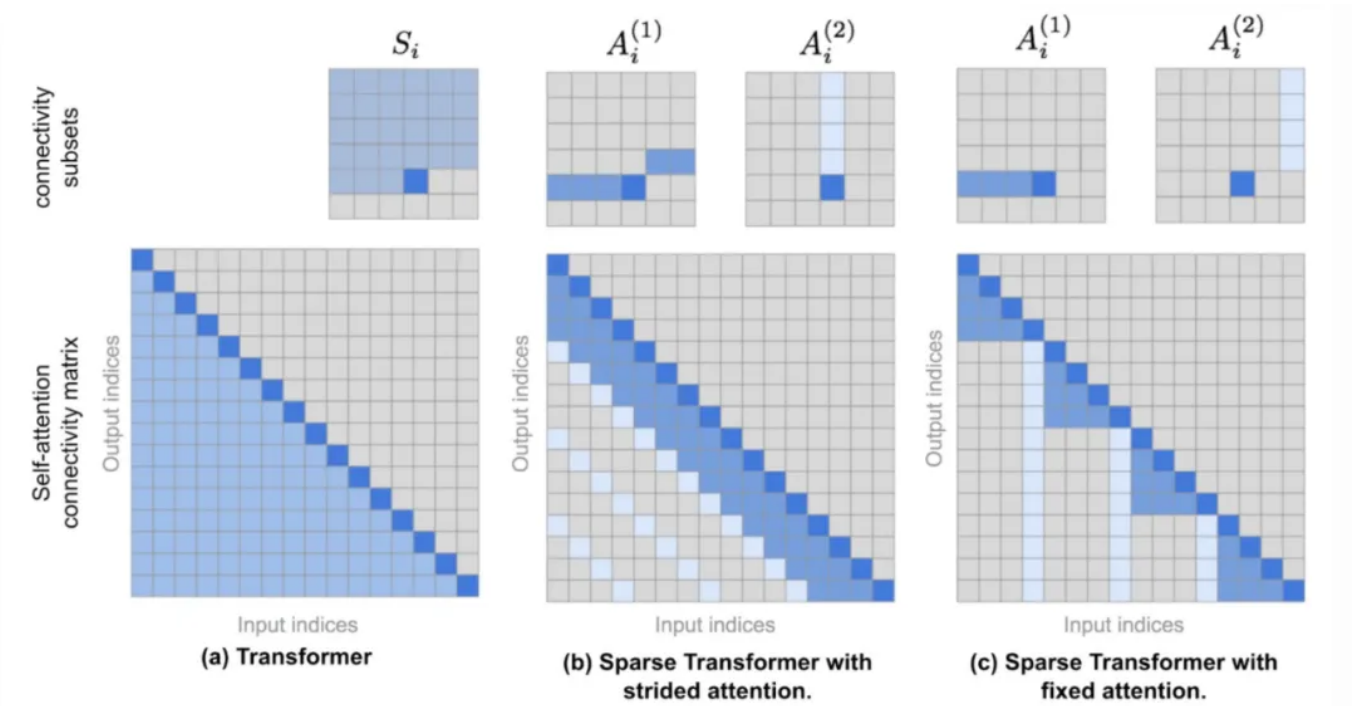


Fig. 10. The top row illustrates the attention connectivity patterns in (a) Transformer, (b) Sparse Transformer with strided attention, and (c) Sparse Transformer with fixed attention. The bottom row contains corresponding self-attention connectivity matrices. Note that the top and bottom rows are not in the same scale. (Image source: [Child et al., 2019](#) + a few of extra annotations.)

- Strided attention(stride $l = \sqrt{n}$, 在图像中, 每个像素可以链接到所有到之前 l 个像素raster scanning顺序, 然后那些像素在相同列中相互链接。

$$A_i^{(1)} = \{t, t + 1, \dots, i\}, \text{ where } t = \max(0, i - l) \quad A_i^{(2)} = \{j : (i - j) \bmod l = 0\}$$

- Fixed attention, 一个小的tokens集合总结之前的位置并且向未来的位置传递信息:

$$A_i^{(1)} = \{j : \lfloor \frac{j}{l} \rfloor = \lfloor \frac{i}{l} \rfloor\}, \text{ where } t = \max(0, i - l) A_i^{(2)} = \{j : j \bmod l \in \{l - c, \dots, l - 1\}\}$$

其中 c 是一个超参数.

Use Factorized Self-Attention in Transformer

存在三种方式使用sparse factorized attention模式的方法:

1. 每个residual block的attention type, 把它们交错起来,
 $attention(X) = Attend(X, A^{n \bmod p})W^o$, 其中 n 是当前residual模块的index;
2. 设置一个单独的head, 它负责所有分解head负责的位置,
 $attention(X) = Attend(X, \bigcup_{m=1}^p A^{(m)})W^o$;
3. 食欲哦那个一个multi-head attention机制, 但是和原始的transformer不同, 每个head可以接受上面的一种模式, 1或者2.

稀疏Transformer还提出了一套改进方案, 将Transformer训练到上百层, 包括梯度检查点、在backward pass的时候重新计算attention和FF层、混合精度训练、高效的块稀疏实现等。

2. Locality-Sensitive Hashing (Reformer)

Reformer模型旨在解决Transformer中的下面几个痛点:

- 具有 N 层的模型中的内存比单层模型中的内存大 N 倍, 因为我们需要存储反向传播的activations。
- 中间FF层通常相当大。
- 长度为 L 的序列上的注意矩阵通常在记忆和时间上都需要 $O(L^2)$ 的内存和时间;

Reformer进行了两种改变:

- 将dot-product的attention替换为locality-sensitive hashing(LSH) attention, 这将时间复杂度从 $O(L^2)$ 降低为 $O(L \log L)$;
- 将标准residual block替换为reversible residual layer, 这样在训练期间只允许存储一次激活, 而不是 N 次 (即与层数成比例)。

Locality-Sensitive Hashing Attention

在attention QK^T 中, 我们更加关注大的只, 对于每个 $q_i \in Q$, 我们在寻找 K 中于 q_i 最近的一个行向量, 为了寻找它, 我们在attention机制中加入: Locality-Sensitive Hashing (LSH)

如果它保留了数据点之间的距离信息, 我们称hashing机制 $x \rightarrow h(x)$ 是locality-sensitive的, 这么做相近的向量可以获得相似的hash, Reformer中, 给定一个固定的随机矩阵 $R \in R^{d*b/2}$, 其中 b 是超参

数, hash函数为 $h(x) = \operatorname{argmax}([xR; -xR])$

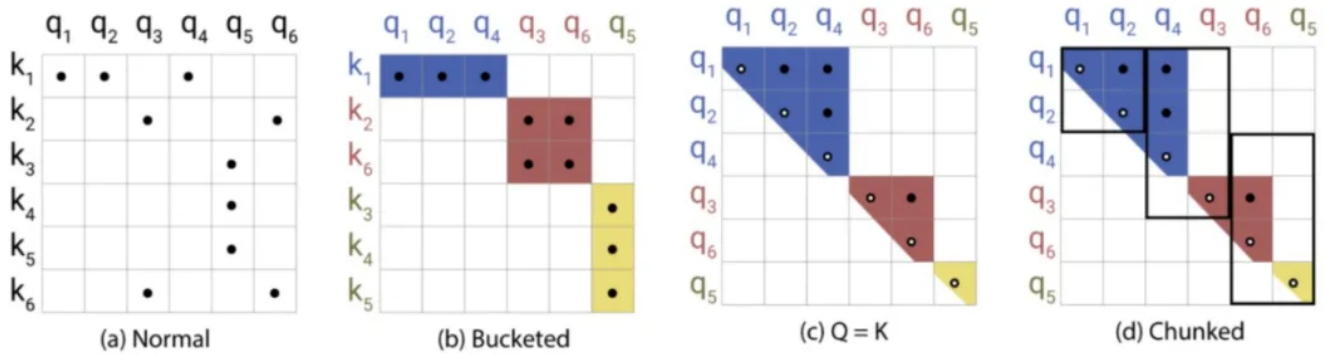


Fig. 11. Illustration of Locality-Sensitive Hashing (LSH) attention. (Image source: right part of Figure 1 in [Kitaev, et al. 2020](#)).

在LSH attention中, 一个query只可以和在相同的hashing bucket中的位置进行交互,

$$S_i = \{j : h(q_i) = h(k_j)\},$$

- attention矩阵通常是稀疏的;
- 使用LSH, 我们基于hash buckets可以对keys和queries进行排序
- 设置 $Q = K$, 这样, 一个bucket中的keys和queries相等, 更便于批处理。有趣的是, 这种“共享QK”配置并不影响Transformer的性能。
- 使用 m 个连续的group在一起的query的batching,

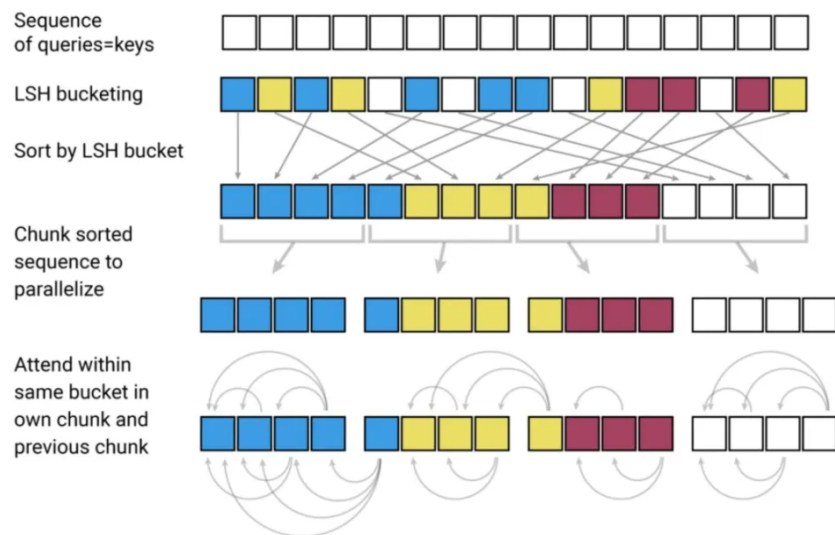


Fig. 12. The LSH attention consists of 4 steps: bucketing, sorting, chunking, and attention computation. (Image source: left part of Figure 1 in [Kitaev, et al. 2020](#)).

Reversible Residual Network

Reversible Residual Network的动机是设计一种结构, 使任何给定层的激活都可以从下一层的激活中恢复, 只需使用模型参数。因此, 我们可以通过在backprop期间重新计算激活来节省内存, 而不是存储所有激活。

给定一层 $x \rightarrow y$, 传统的residual layer都是做的 $y = x + F(x)$, 但是reversible layer将输入和输出split为 $(x_1, x_2) \rightarrow (y_1, y_2)$, 然后执行下面的操作:

$$y_1 = x_1 + F(x_2), y_2 = x_2 + F(y_1),$$

reversing就是:

$$x_2 = y_2 - G(y_1), x_1 = y_1 - F(x_2)$$

我们将相同的思想应用到Transformer中得到:

$$Y_1 = X_1 + \text{Attention}(X_2), Y_2 = X_2 + \text{FeedForward}(Y_1),$$

内存可以通过chunking 前向计算进行操作:

$$Y_2 = [Y_2^{(1)}; \dots; Y_2^{(c)}] = [X_2^{(1)} + \text{FeedForward}(Y_2^{(1)}); \dots; X_2^{(c)} + \text{FeedForward}(Y_1^{(c)})]$$

08 Make it Recurrent (Universal Transformer)

Universal Transformer将Transformer中的自我注意与RNN中的循环机制结合起来, 旨在受益于Transformer的长期全局receptive field和RNN的学习inductive偏差。

Universal Transformer使用自适应计算时间动态调整步长。如果我们固定步数, 一个通用变换器就相当于一个多层变换器, 具有跨层共享的参数。

在较高的层次上, Universal Transformer可以看作是学习每个token的隐藏状态表示的递归函数。递归函数在标记位置之间并行演化, 位置之间的信息通过self-attention进行共享。

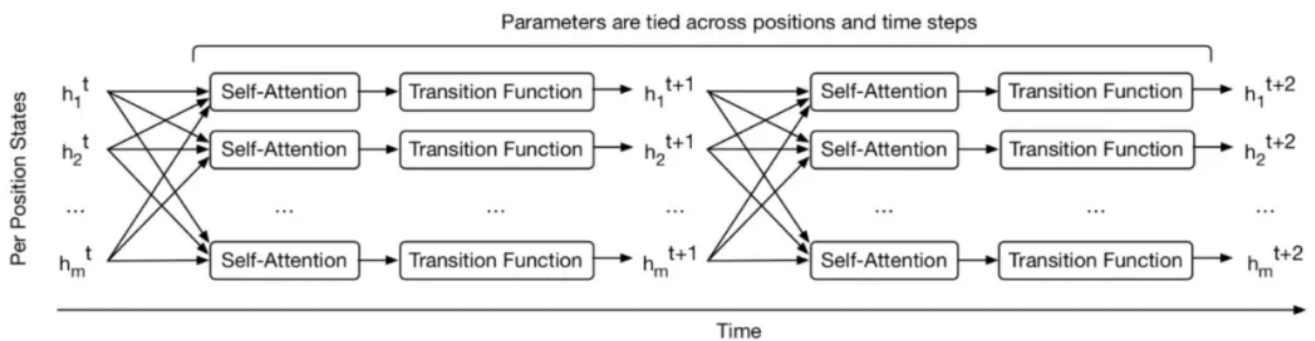


Fig. 13. How the Universal Transformer refines a set of hidden state representations repeatedly for every position in parallel. (Image source: Figure 1 in [Dehghani, et al. 2019](#)).

给定长度为 L 的序列, Universal Transformer在第 t 步迭代更新表示 $H^t \in R^{L \times d}$, 在第0步, H^0 被输出为输入embedding矩阵, 所以的位置编码在multi-head self-attention中被并行处理, 然后在经过一个recurrent transition function

$$A^t = \text{LayerNorm}(H^{t-1} + \text{MultiHeadAtt}(H^{t-1} + P^t)H^t) = \text{LayerNorm}(A^{t-1} + \text{Transi}$$

Transition(\cdot)可以是一个 separable convolution或者fully-connected neural network。

$$PE(i, t, \delta) = \begin{cases} \sin(\frac{i}{10000^{2\delta/d}}) \oplus \sin(\frac{t}{10000^{2\delta/d}}) & \text{if } \delta = 2\delta' \\ \cos(\frac{i}{10000^{2\delta/d}}) \oplus \cos(\frac{t}{10000^{2\delta/d}}) & \text{if } \delta = 2\delta' + 1 \end{cases}$$

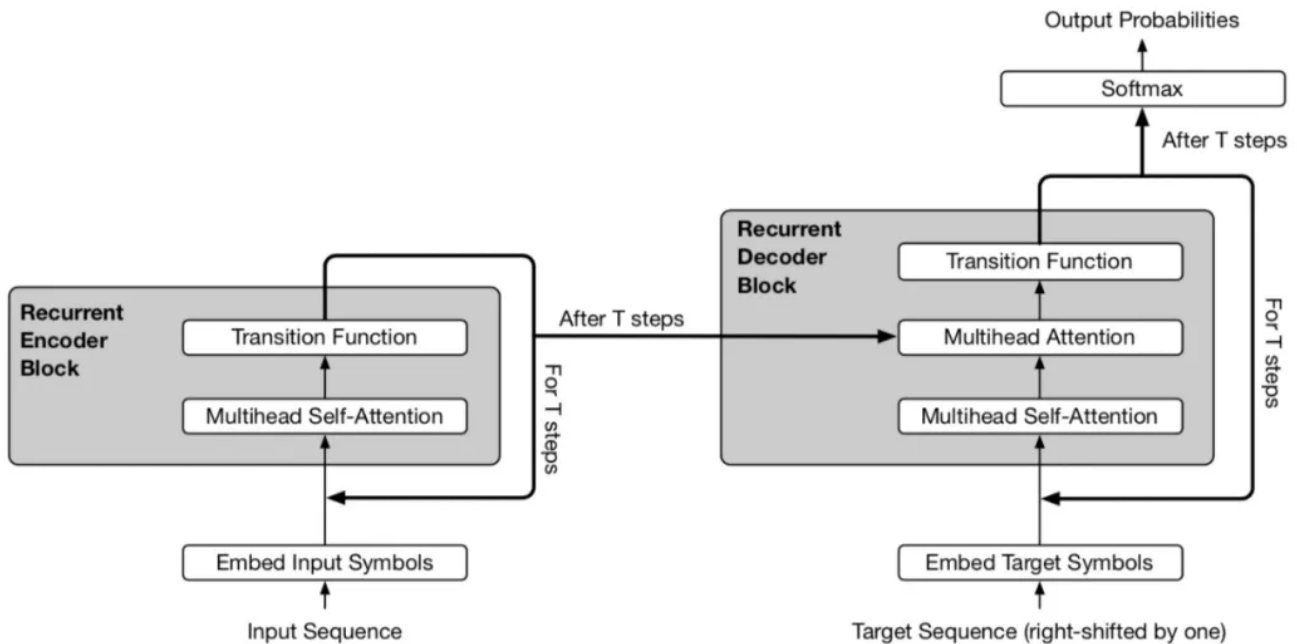


Fig. 14. A simplified illustration of Universal Transformer. The encoder and decoder share the same basic recurrent structure. But the decoder also attends to final encoder representation \mathbf{H}^T . (Image source: Figure 2 in [Dehghani, et al. 2019](#))

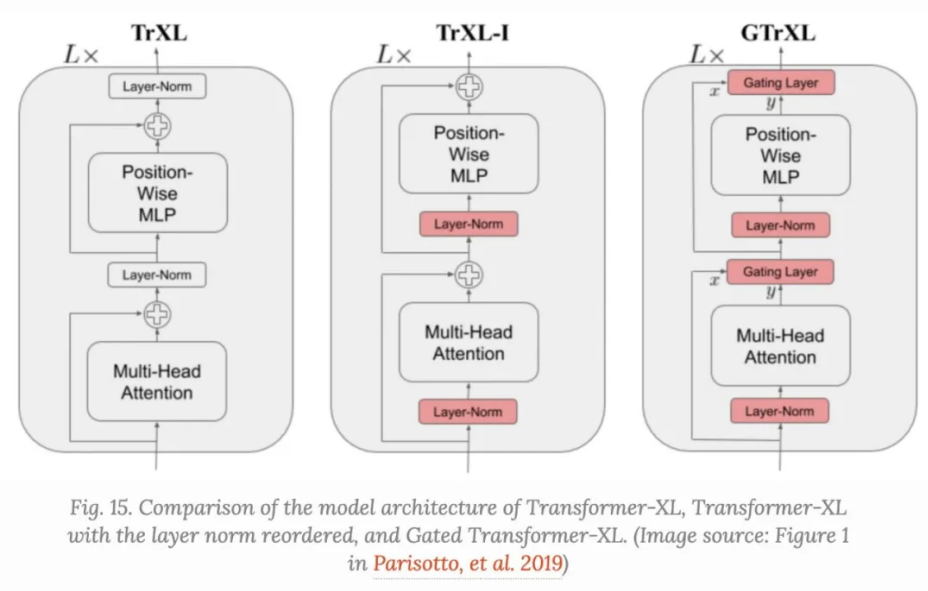
在Universal Transformer的自适应版本中，循环步数 T 由ACT动态确定。每个位置都配有一个动态停止机制。一旦令牌循环块停止，它将停止进行更多的循环更新，而只是将当前值复制到下一步，直到所有块停止或直到模型达到最大步长限制。

09 Stabilization for RL (GTrXL)

Self-attention避免了将整个过去压缩成一个固定大小的隐藏状态，并且不像RNN那样受到梯度消失或爆炸的影响。强化学习任务肯定能从这些特质中受益。然而，即使在有监督学习中，也很难训练Transformer，更不用说在RL环境中了。毕竟，稳定和训练一个LSTM代理本身可能是相当具有挑战性的。


Gated Transformer-XL (GTrXL)是使用Transformer到RL中的一次尝试，GTrXL可以在Transformer-XL上成功稳定的训练。

- layer normalization应用于residual模块中的输入流，而不应用于shortcut流。这种重新排序的一个关键好处是允许原始输入从第一层流到最后一层。
- Residual连接被GRU样式选通机制取代。




参考文献


1. <https://lilianweng.github.io/lil-log/2020/04/07/the-transformer-family.html#locality-sensitive-hashing-reformer>



炼丹笔记四




该二维码7天内(2月14日前)有效，重新进入将更新



知识星球

炼丹笔记

合伙人：九羽



长按扫码预览社群内容和星主关系更进一步