南开大学
Nankai University

南　　开　　大　　学

计算机与网络空间安全学院

并行程序设计实验报告

期末研究报告——开题报告

2011763 黄天昊 2012527 王旭尧

年级：2020 级

指导教师：王刚

2023 年 3 月 26 日

摘要

近年来，随着深度学习的快速发展和广泛应用，越来越多的深度学习模型被提出和应用到各种领域。作为一种新的深度学习算法框架, Transformer 引起了越来越多研究人员的关注, 成为当下的研究热点之一。其中，自注意力机制是 Transformer 的一个重要特点，它能够对每个单词分配相应的“注意力”，这一性质可以迁移到图形领域，用于对图像的某一特征进行“注意力”分配。因此, Transformer 逐渐被引入到视觉领域, 应用于图像处理和计算机视觉任务中。

关键字：Transformer; Self-attention

目录

一、 Transfomer 框架简介

Transformer 是一种基于自注意力机制的序列到序列学习模型，由 Google 于 2017 年提出 [2]。Transformer 的设计目的是解决循环神经网络（RNN）和卷积神经网络（CNN）的一些问题，如处理长序列时的信息丢失、训练时间长等。目前，Transformer 已经应用于各个领域，如计算机视觉、音频处理等，在自然语言处理领域，Transformer 已成为一种重要的序列到序列学习模型。chatGPT 也正是基于这一模型发展出的强大人工智能模型，足见该框架的巨大潜力。

(一) 词嵌入

词嵌入是 NLP 中一种语言模型与表征学习技术，通过将每个单词或短语映射为一个连续向量空间中的实数向量，从而将高维空间嵌入到一个维数更低的向量空间中。Word2Vec 和随机初始化是实现词嵌入的两种常见方法，当数据量很大时，它们的效果相当。在实际应用中，向量的维度通常默认为 512 维，并且取决于最长句子的单词数。

嵌入仅发生在最底层的编码器中。所有编码器都会接收到一个大小为 512 的向量列表——底部编码器接收的是词嵌入向量，其他编码器接收的是上一个编码器的输出。这个列表大小是我们可以设置的超参数——基本上这个参数就是训练数据集中最长句子的长度。对输入序列完成嵌入操作后，每个词都会流经编码器的两层。

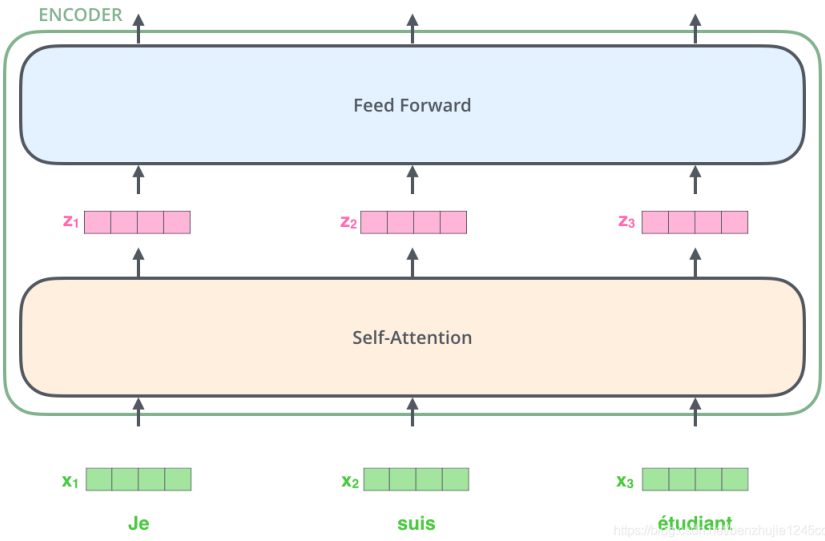

图 1: 词嵌入操作

(二) 位置编码

在 NLP 领域，位置编码也是一项很重要的技术。原因有二：

1. 对于任何一门语言，单词在句子中的位置以及排列顺序是非常重要的，它们不仅是一个句子的语法结构的组成部分，更是表达语义的重要概念。一个单词在句子的位置或排列顺序不同，可能整个句子的意思就发生了偏差。如：

   - I do not like the story of the movie, but I do like the cast.
   - I do like the story of the movie, but I do not like the cast.

上面两句话所使用的的单词完全一样，但是所表达的句意却截然相反。那么，引入词序信息有助于区别这两句话的意思。

2. Transformer 模型抛弃了 RNN、CNN 作为序列学习的基本模型。我们知道，循环神经网络本身就是一种顺序结构，天生就包含了词在序列中的位置信息。当抛弃循环神经网络结构，完全采用 Attention 取而代之，这些词序信息就会丢失，模型就没有办法知道每个词在句子中的相对和绝对的位置信息。因此，有必要把词序信号加到词向量上帮助模型学习这些信息。

位置编码就是用来解决这种问题的方法。位置编码（Positional Encoding）是一种用词的位置信息对序列中的每个词进行二次表示的方法。正如前文所述，Transformer 模型本身不具备像 RNN 那样的学习词序信息的能力，需要主动将词序信息喂给模型。那么，模型原先的输入是不含词序信息的词向量，位置编码需要将词序信息和词向量结合起来形成一种新的表示输入给模型，这样模型就具备了学习词序信息的能力。

一种好的位置编码方案需要满足以下几条要求：

- 它能为每个时间步输出一个独一无二的编码；
- 不同长度的句子之间，任何两个时间步之间的距离应该保持一致；
- 模型应该能毫不费力地泛化到更长的句子。它的值应该是有界的；
- 它必须是确定性的。

Transformer 的作者们提出了一个简单但非常创新的位置编码方法，能够满足上述所有的要求。首先，这种编码不是单一的一个数值，而是包含句子中特定位置信息的 d 维向量（非常像词向量）。第二，这种编码没有整合进模型，而是用这个向量让每个词具有它在句子中的位置的信息。换句话说，通过注入词的顺序信息来增强模型输入。

给定一个长度为 n 的输入序列，让 t 表示词在序列中的位置，$\overrightarrow{p_t} \in \mathbb{R}^d$ 表示 t 位置对应的向量，d 是向量的维度。$f : \mathbb{N} \to \mathbb{R}^d$ 是生成位置向量 $\overrightarrow{p_t}$ 的函数，定义如下：

$$\overrightarrow{p_t}^{(i)} = f(t)^{(i)} := \begin{cases} \sin(\omega_k \cdot t), & \text{if } i = 2k \\ \cos(\omega_k \cdot t), & \text{if } i = 2k+1 \end{cases}$$

其中，频率 w_k 定义如下：

$$\omega_k = \frac{1}{10000^{2k/d}}$$

从函数定义中可以得出，频率沿向量维度减小。因此，它在波长上形成从 2π 到 $1000 \cdot 2\pi$ 几何级数。也可以认为，位置编码 $\overrightarrow{p_t}$ 是一个包含每个频率的正弦和余弦对（注意 d 是能被 2 整除的）。

$$\overrightarrow{p_t} = \begin{bmatrix} \sin(\omega_1 \cdot t) \\ \cos(\omega_1 \cdot t) \\ \sin(\omega_2 \cdot t) \\ \cos(\omega_2 \cdot t) \\ \vdots \\ \sin(\omega_{d/2} \cdot t) \\ \cos(\omega_{d/2} \cdot t) \end{bmatrix}$$

位置编码方法已经有了，那如何让每个词具有它们的位置信息？原始论文将位置编码加到模型输入之上。也就是，对于句子里的每个词 w_t，计算其对应的词嵌入 $\psi(w_t)$，然后按照下面的方法喂给模型：

$$\psi'(w_t) = \psi(w_t) + \overrightarrow{p_t}$$

为了保证这种相加操作正确，让位置向量 (PE) 的维度等于词向量 (WE) 的维度，即 $d_{\mathrm{WE}} = d_{\mathrm{PE}}$

(三) 多头注意力机制

多头注意力机制利用多个独立的注意力头来计算自注意力，以进一步增强注意力层的表现。自注意力是指算法在句子中建立不同词之间相互联系的能力。为了计算自注意力，需要使用向量表示每个单词，例如在单词组”Thinking Machines” 中，单词”Thinking” 可以用一个向量来表示。

1. QKV 向量的生成

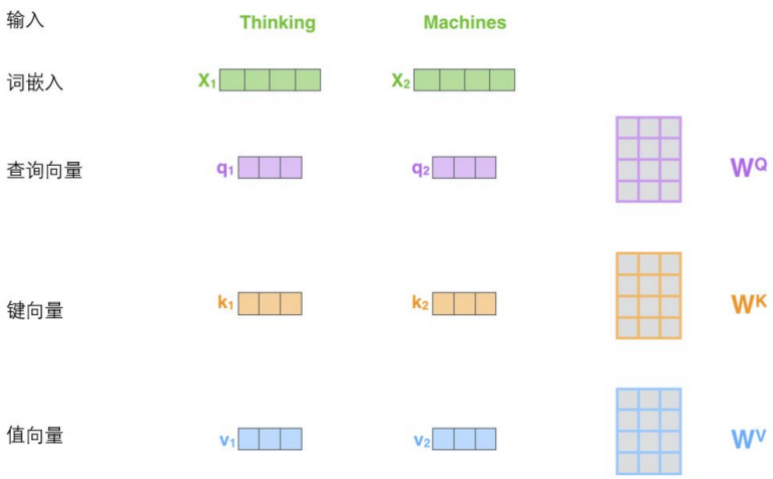

图 2: Q,K,V 的计算

如上图所示，针对每个编码器的输入向量（即每个单词的词向量），我们生成三个向量，即查询向量、键向量和值向量。具体而言，对于每个单词，我们使用词嵌入和三个权重矩阵（即右侧的 WQ、WK、WV 矩阵）相乘来创建这三个向量。以单词”Thinking” 为例，其词嵌入结果 X1 与 WQ 权重矩阵相乘得到 q1，这就是与该单词相关的查询向量。类似地，我们使用相同的方式为输入序列中的每个单词创建一个查询向量 q、一个键向量 k 和一个值向量 v。

2. 计算注意力权重

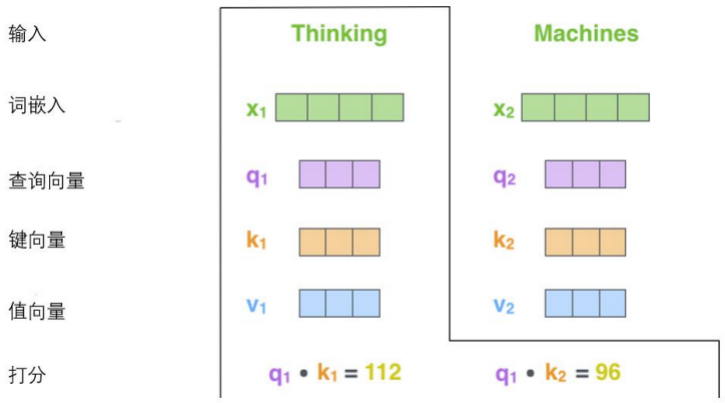

图 3: 计算注意力权重

我们可以使用输入句子中的每个单词来对单词"Thinking" 进权重计算，这些结果将决定在编码单词"Thinking" 的过程中，应该如何考虑输入句子中的其他部分。具体而言，这些结果是通过计算单词"Thinking" 的查询向量 q 与所有输入句子单词的键向量 k 作点积得到的。

3. normalization

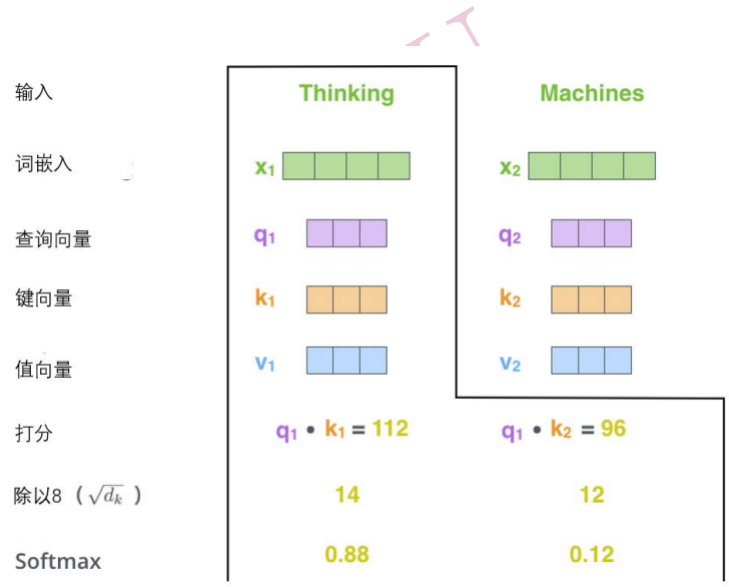

图 4: normalization

在计算完单词"Thinking" 与所有输入句子单词的点积得分后，我们将这些分数除以 8（8 是使用 k 的维数 64 的平方根），以使梯度更加稳定。然后，我们对这些得分进行 softmax 运算，以将所有单词的得分归一化，使得它们的总和为 1 且均为正数。这个 softmax 得分将决定每个单词对于编码器当前位置（即"Thinking"）的贡献。显然，处于该位置上的单词将获得最高的 softmax 分数，但有时也有必要关注与当前单词相关的其他单词。

4. 语义处理与输出

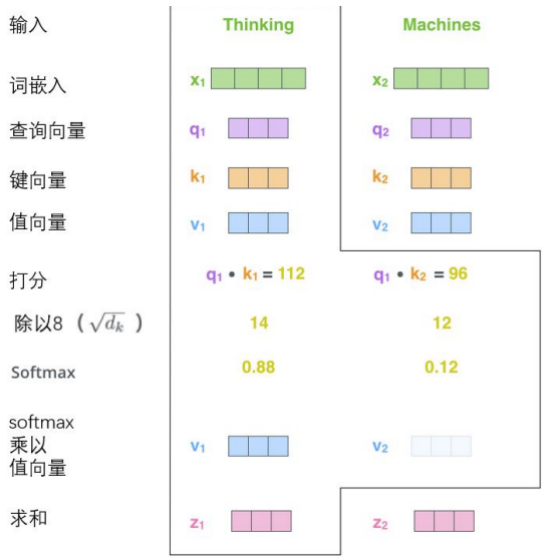

图 5: 加权求和

通过将每个值向量乘以对应的 softmax 分数，并对它们进行加权求和，我们可以得到自注意力层在”Thinking” 位置的输出向量，该向量可以传递给前馈神经网络进行下一步处理。可以通过矩阵运算来简化上述步骤的实现，同时可以使用 C++ 语言来优化矩阵运算的性能。下面是矩阵运算的表达式：

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

其中 Q、K、V 分别是查询向量、键向量和值向量的矩阵，^{T} 表示转置，d_k 表示键向量的维度，$softmax$ 表示 $softmax$ 函数。

二、 Transfomer 相关并行计算研究

Transformer 的优点之一便是处理长序列时表现良好，可以并行计算，且无需像 RNN 那样按时间步展开，因此训练速度更快。

Transformer 的核心是自注意力机制，它能够捕捉输入序列中的上下文关系，并且处理长序列时表现良好。自注意力机制是通过对输入序列中每个位置的表示进行线性变换，计算每个位置与其他位置的相似度，并加权求和得到该位置的新表示来实现的。多头自注意力机制则将输入序列划分为多个头，每个头可以独立地寻找相关信息，然后将结果合并起来作为该位置的表示。这个过程可以并行执行，因为每个头都可以独立地寻找相关信息。

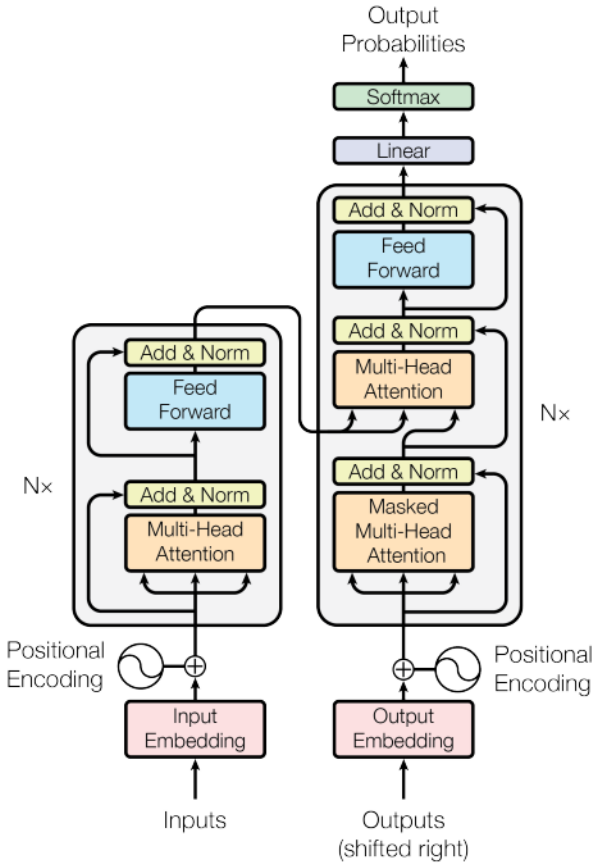

图 6: Transformer 模型结构

Transformer 模型的并行计算包括两个方面：层间并行和头间并行。

层间并行是指在 Transformer 中，不同层之间的计算可以并行进行。具体地，Transformer 中每个编码器和解码器层的计算都可以独立地进行，因此在训练过程中可以同时对多个层进行计算，从而加快了计算速度。在推理过程中，也可以将不同层的计算分配到不同的计算设备上进行加速。

头间并行是指在多头自注意力机制中，每个头的计算可以独立地进行，从而实现并行计算。具体地，多头自注意力机制将输入序列划分为多个头，每个头都可以独立地寻找相关信息，然后将结果合并起来作为该位置的表示。这个过程可以并行执行，因为每个头都可以独立地寻找相关信息。

除了层间并行和头间并行，还有一些其他的并行优化方法可以应用于 Transformer 模型。例如，可以对输入序列进行分块，然后将每个块分配到不同的计算设备上进行计算，从而进一步加快计算速度。此外，还可以使用异步并行计算来优化模型的训练速度。

总的来说，Transformer 模型具有良好的并行计算性能，可以利用多个计算设备并行计算不同的任务，从而加快计算速度，提高模型的训练和推理效率。

(一) Gshard

Gshard 是 Google 在 2020 年提出的一种针对大规模 Transformer 模型的并行计算框架 [1]，旨在解决传统 Transformer 在大规模任务中遇到的计算和通信瓶颈问题。

传统 Transformer 模型在大规模任务中遇到的主要问题是，随着模型规模的增大，计算和通

信开销呈指数级增长，导致模型的训练和推理速度变慢，甚至无法完成训练。Gshard 框架通过对模型进行分片，将模型的计算和通信分配到多个设备上并行执行，从而减少了计算和通信的开销，加速了模型的训练和推理速度。

Gshard 框架包括两个核心组件：Shard 和 Pipeline。Shard 将模型分成多个片段，并将每个片段分配到不同的设备上进行计算和通信。Pipeline 则通过将计算和通信分为多个阶段，将不同设备上的计算和通信串联起来，从而实现高效的并行计算。

具体地，Gshard 采用了两级分片的策略，将模型首先分成若干个模块，然后将每个模块进一步分成若干个片段，每个片段都可以分配到不同的设备上进行计算和通信。这种分片策略使得计算和通信的开销可以线性缩小，从而适用于大规模模型的训练和推理。

在 Gshard 框架中，Pipeline 用于将不同设备上的计算和通信串联起来，形成一个完整的计算流程。具体地，Gshard 将模型分成多个计算阶段，每个阶段包括若干个模块，每个模块包括若干个片段。通过将每个阶段分配到不同的设备上进行计算和通信，可以实现高效的并行计算。

Gshard 框架的实验结果表明,与传统的单设备训练相比,Gshard 能够显著提高大规模 Transformer 模型的训练和推理速度，同时保持较低的通信和计算开销。该框架已经在 Google 的多个应用中得到应用，并为 Google 的机器翻译和语言建模等任务提供了重要支持。

(二) Megatron

Megatron 是 NVIDIA 研发的一个用于训练大规模 Transformer 模型的框架，它能够训练拥有数千亿个参数的模型，并在多个 GPU 或多个机器上进行并行训练。该框架已经被应用于多项自然语言处理任务中，例如语言建模、翻译和问答等。

Megatron 采用了数据并行和模型并行相结合的方法，将数据分成多份并分配到多个 GPU 或机器上进行并行计算，同时还将模型分成多个部分，每个部分分配到不同的 GPU 或机器上进行并行计算。这种方法能够提高模型训练的速度和效率，并且具有较好的可扩展性。

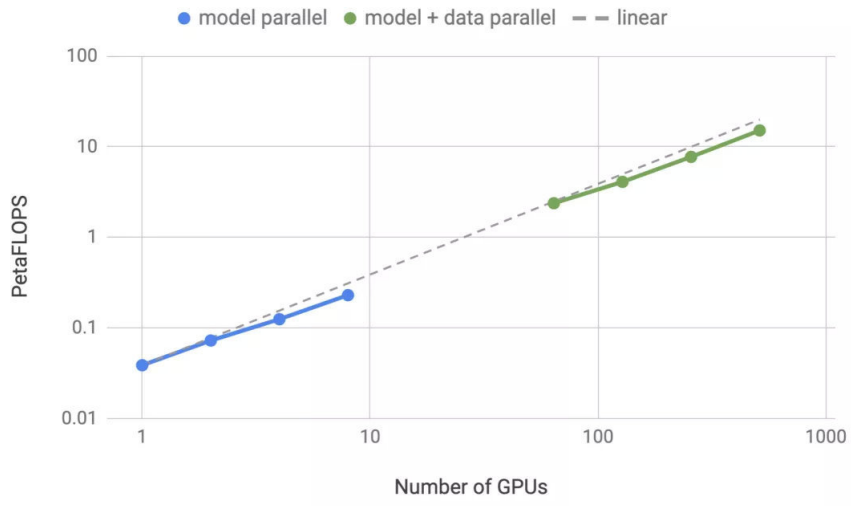

图 7: Megatron 模型的并行性能随 GPU 数量变化的函数

Megatron 还采用了一些其他的技术来进一步提高训练效率，例如混合精度计算、梯度累积和动态调整学习率等。混合精度计算可以将模型中的某些操作用低精度的数值表示，从而减少存储和计算开销。梯度累积可以减少计算中的内存使用，同时也有助于稳定训练。动态调整学习率则可以根据训练过程中的表现来自适应地调整学习率，以提高训练效率。

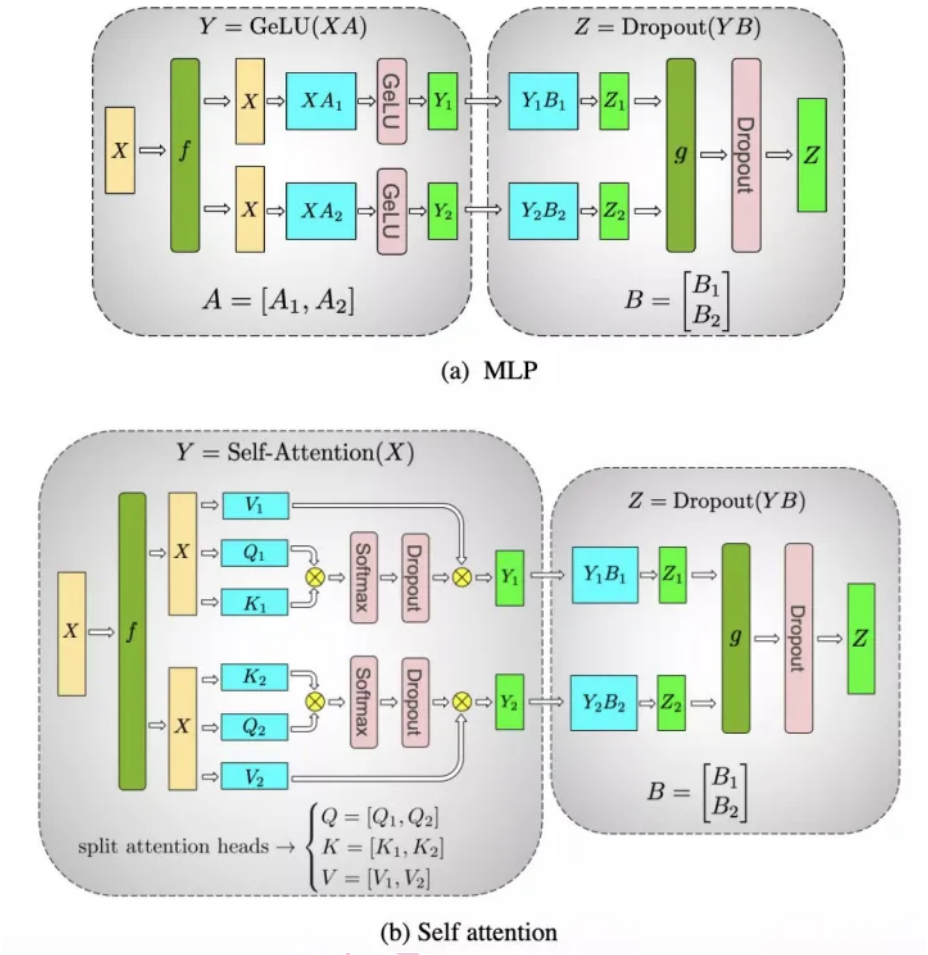

图 8: Megatron 利用 multihead attention operation 进行并行计算

三、 研究计划

我们的研究计划分为三部分，第一部分是实现基础的 Tensor 数据结构和反向求导算法，第二部分是搭建 Transformer 模型并实现串行训练，第三部分是针对多头注意力机制进行并行化训练

(一) C++ 实现 Tensor

在 Tensor 实现部分，我们预期实现两部分的功能。

1. 基础 Tensor 运算

我们的 Tensor 支持的运算包括：

1. 线性运算，包括对应元素加法，对应元素乘法
2. 对应元素的除法运算
3. 矩阵乘法运算
4. 向量点积运算

5. 以 e 为底数的指数运算

6. ln 运算

2. 反向求导

为了实现自动反向求导功能，我们打算实现一个计算图。根据我们已经实现了的 Tensor 运算制定对应的求导操作。进而实现自动反向求导。

(二) 搭建 Transformer 并实现串行化训练

在实现 Tensor 的基础上，我们计划首先实现 Transformer 的基本组件，包括自注意力层、前馈神经网络、残差连接和层归一化等。然后，我们可以进而构建 Transformer 的编码器和解码器模型，并使用序列到序列的模型架构将它们连接起来。最后，我们在 CPU 上训练这个简单的模型，并使用训练数据集来检查模型的正确性。如果串行训练能够正常工作，那么可以考虑并行化训练以加快模型训练速度。

(三) 将串行 Transformer 进行并行化处理

Transformer 的并行化处理主要包括两个部分：头并行（head parallelism）与层并行（layer parallelism）。

1. head parallelism

Transformer 的头并行（head parallelism）是指在 Transformer 模型的多头注意力机制中，将多个头同时计算并行处理，以提高模型的计算效率和训练速度。

在标准的 Transformer 模型中，多头注意力机制包括多个独立的注意力头，每个头都负责学习不同的注意力模式，并产生一个注意力向量。这些向量最终会被连接起来并输入到下一层。在头并行的 Transformer 中，这些独立的注意力头可以被分配到不同的 GPU 或处理器上并行计算，因此多个头可以同时进行计算，从而提高了计算速度。

2. layer parallelism

Transformer 的层并行（layer parallelism）是指在 Transformer 模型中，将不同的层分配到不同的 GPU 或处理器上并行计算，以提高模型的计算效率和训练速度。

在标准的 Transformer 模型中，每个层都包括多个子层，例如自注意力层和前馈神经网络层。在层并行的 Transformer 中，这些子层可以被分配到不同的 GPU 或处理器上并行计算，从而使不同层之间的计算可以同时进行。这种并行化方法可以显著减少模型的训练时间和计算资源的使用。

层并行通常用于训练大型 Transformer 模型，因为它们具有多个层，需要大量的计算资源来训练。通过层并行，模型的各个层可以同时计算，加快了模型的训练速度。此外，层并行还可以在多个计算节点之间进行分布式计算，从而更有效地使用计算资源。

四、 可行性分析

(一) Transformer 为什么可以做并行计算?

NLP 中的 RNN 之所以不能并行化，是因为其是一个马尔可夫过程，即当前状态只与前一个状态有关, 而与再之前的所有状态无关。它天生是个时序结构，t 时刻依赖 t-1 时刻的输出，而 t-1 时刻又依赖 t-2 时刻，如此循环往前，我们可以说 t 时刻依赖了前 t 时刻所有的信息。

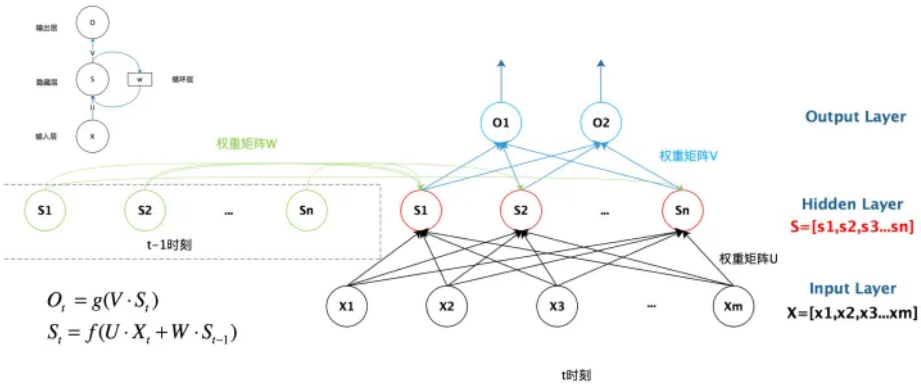

图 9: RNN 模型结构图

Transformer 可以并行化的重要部分体现在两个方面：Encoder 和 Decoder。Transformer 的核心之一是 self-Attention 自注意力机制, 其中, 自注意力机制就是利用两个输入之间两两相关性作为权重的一种加权平均, 将每一个输入映射到输出上。所以从这个层面上来说，Transformer 的 Encoder 部分里，输出与之前所有的输入都有关，并不是只依赖上一个输入，因此，Transformer 的 Encoder 可以并行化计算所有的自注意力机制。

Transformer 的 Decoder 部分，引入了一种“teacher force”的概念，就是每个时刻的输入不依赖上一时刻的输出，而是依赖之前所有正确的样本，而正确的样本在训练集中已经全部提供了。正是这种“teacher force”的思想，才可以在 Transformer 的 Decoder 部分进行并行化计算。我们以下面的一个例子来阐述 teacher force 的具体工作流程：

假设目标语句为：“I love China”这样一个长度为 3 的句子。首先我们给句子加入一个起始符号：“<start> I love China”。对于解码器模块，我们先来看一个正常的处理思路。

1. 第一轮：给解码器模块输入“<start>”和编码器的输出结果，解码器输出“I”
2. 第二轮：给解码器模块输入“<start> I”和编码器的输出结果，解码器输出“Iove”
3. 第三轮：给解码器模块输入“<start> I love”和编码器的输出结果，解码器输出“China”
4. 第四轮：给解码器模块输入“<start> I love China”和编码器的输出结果，解码器输出“<end>”，至此完成。

这就是一个“完美”解码器按时序处理的基本流程。之所以说完美，是因为解码器每次都成功的预测出了正确的单词，如果编码器在某一轮预测错了，那么给下一轮解码器的输入中就包含了错误的信息，然后继续解码。

如果在第二轮时，给解码器模块输入“<start> I”和编码器的输出结果，解码器没有正确预测出“Iove”，而是得到了“want”。如果没有采用 teacher force，在第三轮时，解码器模块输入的就是“<start> I want”。如果采用了 teacher force，第三轮时，解码器模块输入的仍然是“<start> I love”。通过这样的方法可以有效的避免因中间预测错误而对后续序列的预测，从而

加快训练速度。而 Transformer 采用这个方法，为并行化训练提供了可能，因为每个时刻的输入不再依赖上一时刻的输出，而是依赖正确的样本，而正确的样本在训练集中已经全量提供了。

(二) SIMD 对于深度学习并行计算的支持

SIMD（Single Instruction Multiple Data）是一种计算机指令集架构，它可以同时对多个数据元素执行相同的操作。在深度学习中，矩阵运算是常见的操作，如矩阵乘法、加法和卷积等。使用 SIMD 指令可以加速这些矩阵运算，从而提高深度学习算法的效率和性能。

具体而言，SIMD 可以将矩阵运算拆分成多个相同的操作，同时在同一时钟周期内对多个数据元素执行相同的操作。这使得 SIMD 指令集可以快速处理大量数据，大大提高了深度学习算法的运算速度。另外，由于 SIMD 指令可以一次执行多个操作，因此可以减少数据传输和内存访问的次数，降低了算法的功耗和内存带宽的消耗。

五、 分工

本实验由王旭尧（2012527）和黄天昊（2011763）共同实现。分工如下：王旭尧同学负责 Tensor 的实现，黄天昊同学负责 Transformer 的串行化搭建与检验。并行化处理由王旭尧同学和黄天昊同学共同研究完成。

参考文献

[1] Dmitry Lepikhin, HyoukJoong Lee, Yuanzhong Xu, Dehao Chen, Orhan Firat, Yanping Huang, Maxim Krikun, Noam Shazeer, and Zhifeng Chen. Gshard: Scaling giant models with conditional computation and automatic sharding. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021.

[2] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.