中国科学技术大学

学士学位论文

基于预训练的长文本建模

作者姓名： 李维晟

学科专业： 计算机科学与技术

导师姓名： 周文罡 教授 宫叶云 研究员

完成时间： 二〇二一年五月二十日

University of Science and Technology of China

A dissertation for bachelor's degree

Long text modeling based on pretraining

Author: Li Weisheng

Speciality: Computer Science and Technology

Supervisors: Prof. Zhou Wengang, Researcher Gong Yeyun

Finished time: May 20, 2021

中文内容摘要

本科生需要手动将摘要置于目录后。

基于 Transformers 的模型在面对长文本任务时往往会因为其序列长度的平方复杂度而带来庞大的计算开销。为了解决这个问题，本论文提出了一种具有双层稀疏注意力机制的改进模型 Poolingformer。其第一层注意力采用了滑动窗口注意力机制，每个 token 只和全局 token 与其临近的接收窗口内的 token 计算注意力。而在额外的第二层注意力中，在接收窗口比第一层更大的同时，模型还通过池化操作压缩了序列长度，所以模型在获得更多信息的同时减少了计算量。除此之外，本文还尝试将 Poolingformer 与 Deberta 模型的相对位置编码机制进行融合。本文通过长文本摘要任务的 arXiv 与 PubMed 数据集检验了 Poolingformer，并取得了 SoTA 的结果。这充分说明了 Poolingformer 高效利用、发掘信息的能力。

关键词：Poolingformer；Transformers ；注意力；长文本；摘要；相对位置编码

Abstract

For models based on Transformers, their computational and memory complexity tend to grow quadratically with respect to sequence length. To solve this problem, we introduce Poolingformer which revises full attention schema to a two-level sparse attention schema. Its first layer adopts a sliding window attention in which every token attend to their neighbors in the receptive window. In the additional second layer of attention computing, model applies pooling to compress sequence length as long as the larger window size. By doing so, model can achieve more information gain at a relatively low computation cost. Beside of above, we tried to apply Deberta's relative attention to Poolingformer to create a combined model.For experiments, we evaluate Poolingformer on long sequence summarization task, and achieved SoTA on both arXiv and PubMed dataset. This fully demonstrates Poolingformer's outtanding ability to utilize the information.

Key Words: Poolingformer; Transformers ; Attention; Long Document; Summarization; Relative Position Embedding

目　录

第一章　绪论

Transformer[1] 是一种利用自注意力机制处理序列数据的文本建模模型。自从 BERT[2] 开始，各种基于 Transformer 的模型与变种模型已经在各种自然语言处理任务上取得了一波又一波的冲击，包括语言模型[3-4]，翻译[5]，摘要[6]，文本分类[7]，QA[8-10] 等等，都有基于 Transformer 的模型取得了 SOTA 的结果。

Transformer 的成功，可部分归功于其自注意力机制，这种机制在序列, 即句子的所有 token 两两之间计算一个分数，以更好地在整个序列中传递、汇总信息。因此，时间和空间复杂度与序列长度的平方成正比。在文本较短时尚可接受，但对于长文本任务来说，自注意力机制就会带来巨大的时间与内存开销。因此，BERT 和很多基于 Transformer 的模型都将一次计算的序列长度限制在 512 个 token[11]。而对于文本远长于 512 的数据集（比如 arXiv，见表 3.1)，这种限制无疑会降低模型的表现。而这种限制也体现在 QA，长文本分类等其他任务上。为了把长文本的输入长度限制在 512，可以采取分割成长度为 512 的片段的办法，但这样会丢失片段间的关联，为了弥补这种丢失又会使模型更加复杂。

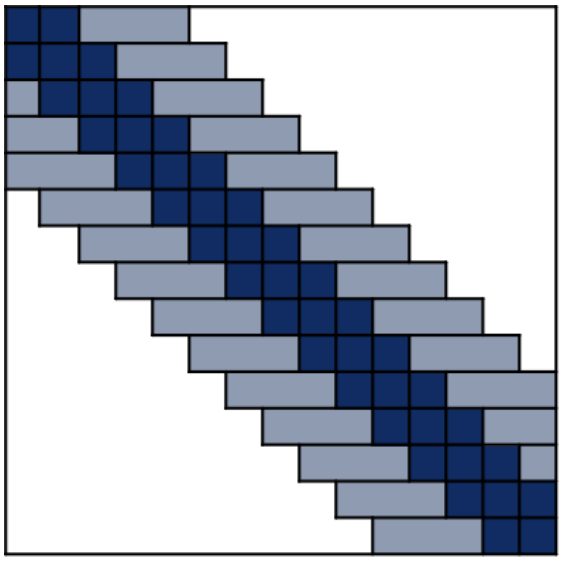

图 1.1　Poolingformer 的双层注意力机制示意图。深色为第一层注意力的接收窗口，浅色为第二层注意力的接收窗口，相连的格子表示对应的 token 序列经过池化操作，被压缩为一个 token。

为了解决这个问题，在现有工作[9,12-13] 的基础上，本文提出了 Poolingformer，它将 Transformer 的完全的自注意力机制改为了一种双层的稀疏注意力机制。在第一层里，token 只和部分 token 计算自注意力，这部分 token 包括所有的全局 token 和自己相邻的一定长度的窗口内的的 token。而全局 token 和全部的 token

计算自注意力，总体上是一种滑动窗口的自注意力。这一层与 longformer 的自注意力是一样的。在此之上的第二层里，每个 token 关注范围是更大的相邻窗口，为了减少关注的 token 数量，将窗口对应的 token 序列进行 pooling 操作，每个 token 与 pooling 操作后的序列计算注意力。图 1.1形象展示了 Poolingformer 的双层注意力机制。与只有第一层自注意力的模型[9,12] 相比，Poolingformer 的第二层注意力能让 token 接受到更大范围的 token，这就意味着更加丰富的信息。和 Transformer 相比，Poolingformer 的双层稀疏注意力在分层利用信息上做的更好，将算力按照信息的重要程度分配，最关键的信息能够获得最密集的算力，而非 Transformer 的完全的自注意力机制那样将算力平均分配，在长文本的情景下能够大量的节约运算力资源。Poolingformer 的时间和空间复杂度只与序列长度成线性关系。

与此同时，本文还将 Poolingformer 与 Deberta——另外一个基于 Transformer 的模型相结合。Deberta[7] 改进了 token 位置信息的融入方式，采用了一种新的相对位置编码，在短文本理解的任务上取得了 SOTA 的结果。由于两种模型修改了 Transformer 的两个不同的要素，因此将 Poolingformer 的注意力计算模式，与 Deberta 的自注意力分数计算，两者结合起来，用于长文本理解的任务，应该会得到一个更好的结果。

第二章　模型

这一章会详细介绍所涉及模型的架构，包括创新点以及各模型之间的不同之处。在第一节里，会对标准的 Transformer[1] 与其自注意力机制做简要介绍，然后在第二节里会详细叙述 Poolingformer 模型与其双层注意力机制。在第三节里，会介绍 Deberta[7] 模型的相对位置编码，并在第四节详细阐述将 Poolingformer 与 Deberta 两个模型结合的细节。

第一节　Transformer 模型

根据任务需不需要生成文本的不同，Transformer 可以只包含 Encoder 编码器，又或者体现为一个 Encoder-Decoder 架构，包括编码器与解码器，实现信息的理解并用于文本的生成。无论是编码器还是解码器，Transformer 都是由多个结构完全相同的层串联在一起形成的，每一层的输出即作为下一层的输入。当然，编码器与解码器的层有一些不同，但其基本组件都是一致的，包括多头自注意力部件，双层前馈神经网络，层正则化[14] 与其之后的残差连接，具体可以参考图 2.1[15]。

对于长度为 N 的输入序列，首先将其嵌入到高维空间，用长度为 d 的向量来表示一个 token，此时序列的形状为 $\mathbb{R}^N \times \mathbb{R}^d$。在输入第一层之前，还要加上一个相同形状的位置编码，使得模型可以识别每个 token 的位置。这个位置编码可以通过训练得到，也可以是用三角函数构造的彼此正交的向量集合。

以编码器为例，Transformer 的每一层中，数据又会先后通过多头自注意力与双层前馈神经网络两大部件，且在经过这两个部件时都要进行正则化与残差连接。每一层的计算可以表示为先

$$X = \text{LayerNorm}(\text{Self-Attention}(X)) + X \tag{2.1}$$

再

$$X = \text{LayerNorm}(\text{FFN}(X)) + X \tag{2.2}$$

解码器的一层比编码器多一个多头交叉注意力组件，将要生成的 token 与编码器输出的 token 计算注意力，以融合来自编码器的信息。

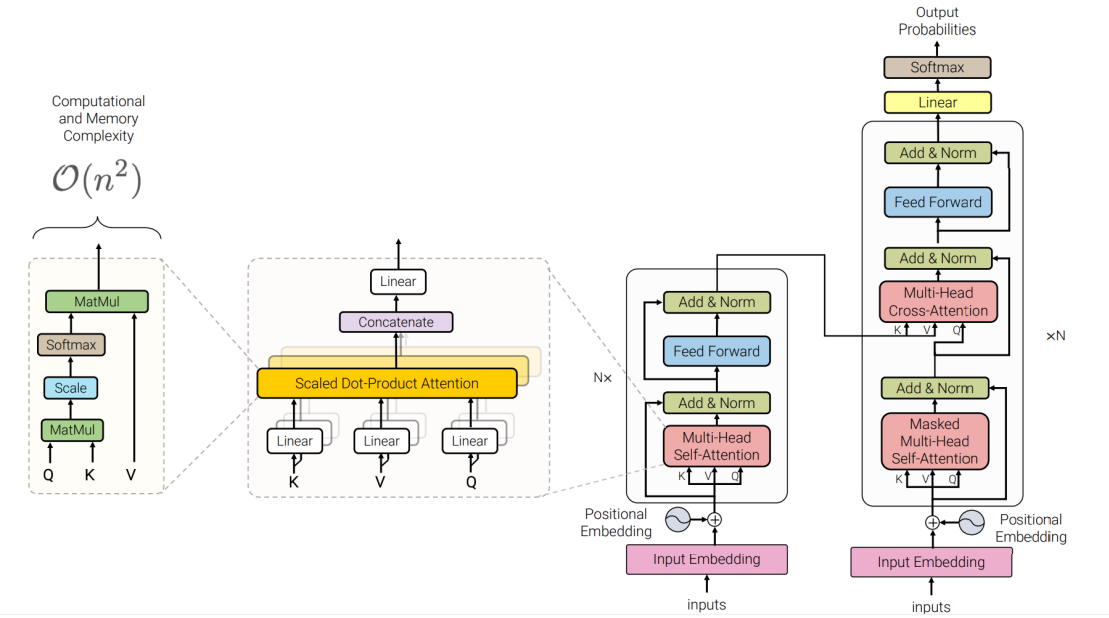

图 2.1　标准 Transformer 架构

一、多头自注意力

自注意力机制无疑是 Transformer 模型最重要的部分。对于一个头而言，计算可以表示为：

$$\mathbf{A}_h = \text{Softmax}(\alpha \mathbf{Q}_h \mathbf{K}_h^\top)\mathbf{V}_h \tag{2.3}$$

其中 $\mathbf{Q}_h = \mathbf{W}_q\mathbf{X}$，$\mathbf{K}_h = \mathbf{W}_k\mathbf{X}$，$\mathbf{V}_h = \mathbf{W}_v\mathbf{X}$ 是经过了三个不同的映射变换的输入序列，分别为 query,key,value 序列，而 $\mathbf{W}_q$，$\mathbf{W}_k$，$\mathbf{W}_v \in \mathbb{R}^{d\times d_h}$ 则是三个不同的映射矩阵。$\mathbf{X}$ 是输入序列，而 N_h 表示头的个数，d_h 表示每个头的嵌入长度。一般来说 $d = N_h \times d_h$，所以可以视作每个头都分配了嵌入的一段，进行计算。α 是正则化因子，一般设置为 $\frac{1}{\sqrt{d_h}}$。

用 $\mathbf{A}_1 \cdots \mathbf{A}_{N_h}$ 表示每个头的输出，则自注意力计算的最后一步是：

$$\mathbf{Y} = \mathbf{W}_o[\mathbf{A}_1 \cdots \mathbf{A}_{N_h}] \tag{2.4}$$

$\mathbf{Y}$ 为自注意力输出，而 $\mathbf{W}_o \in \mathbb{R}^{d\times d}$ 是输出映射矩阵。

通过注意力分数矩阵 $\mathbf{A} = \mathbf{Q}\mathbf{K}^\top$ 的计算，自注意力机制让每个 token 都能从所有 token 获得信息。与此同时，这步操作也带来了 $\mathcal{O}(n^2)$ 的时间和空间复杂度，这大大限制了输入序列的长度，以及 Transformer 在长文本上的应用。

第二节　Poolingformer：双层注意力机制

由于模型对 Transformer 的修改仅限于注意力部分，本小节只详细介绍 Poolingformer 的双层注意力机制。

一、第一层注意力：滑动窗口注意力

第一层注意力采用了滑动窗口的自注意力机制。在没有全局 token 的情况下，每个 token 之和其前后一定范围内的 token，即一个接收窗口内的 token 计算注意力分数。设 w_1 为窗口长度，则第 i 个 token 的接收窗口 $\mathcal{N}(i, w_1)$ 可以表示为：

$$\mathcal{N}(i, w_1) = \left\{ i - w_1, ..., i, ..., i + w_1 \right\} \tag{2.5}$$

用 $\mathbf{K}_{\mathcal{N}(i,w_1)}$ 和 $\mathbf{V}_{\mathcal{N}(i,w_1)}$ 来表示 $\mathbf{K}$ 和 $\mathbf{V}$ 的对应列，则第一层注意力的输出 $\mathbf{Y}$ 的第 i 个 token y_i 为：

$$y_i = \text{Softmax}\left(\alpha q_i \mathbf{K}^{\top}_{\mathcal{N}(i,w_1)} \right) \mathbf{V}_{\mathcal{N}(i,w_1)} \tag{2.6}$$

可见每个 token 只和窗口范围 $\mathcal{N}(i, w_1)$ 内的 token 计算了注意力，体现了稀疏自注意力的特点。

有一部分任务需要设置一些特殊的 token，而这些 token 往往需要和整个序列的全部 token 交换信息。以 QA 任务为例，在序列中会有作为问题的 token，这些 token 显然是要关注整篇文章，反之整篇文章也要关注这些 token。这些 token 被称为全局 token。如果输入数据指定了全局 token，那非全局 token 的普通 token 的接受范围除了接收窗口以外，还要包括所有的全局 token。而全局 token 要和整个序列全部 token 计算注意力。以 $\mathcal{G} = \{q_1, ..., q_l\}$ 表示全局 token 集合，则第 i 个 token 的接受窗口 $\mathcal{N}_{\mathcal{G}}(i, w_1)$ 表示为：

$$\mathcal{N}_{\mathcal{G}}(i, w_1) = \begin{cases} \mathcal{N}(i, w_1) \cup \mathcal{G} & i \notin \mathcal{G} \\ [\![1, ..., n]\!] & i \in \mathcal{G} \end{cases} \tag{2.7}$$

注意力计算修改为：

$$y_i = \text{Softmax}\left(\alpha q_i \mathbf{K}^{\top}_{\mathcal{N}_{\mathcal{G}}(i,w_1)} \right) \mathbf{V}_{\mathcal{N}_{\mathcal{G}}(i,w_1)} \tag{2.8}$$

二、第二层注意力：池化

第二层注意力是一种基于池化（pooling）的注意力机制。对第一层注意力的输出 $\mathbf{Y}$ 进行映射

$$\tilde{\mathbf{Q}} = \mathbf{W}_{\tilde{q}}\mathbf{Y} \tag{2.9}$$

$$\tilde{\mathbf{K}} = \mathbf{W}_{\tilde{k}}\mathbf{Y} \tag{2.10}$$

$$\tilde{\mathbf{V}} = \mathbf{W}_{\tilde{v}}\mathbf{Y} \tag{2.11}$$

得到变换后的 query，key 与 value 序列 $\tilde{\mathbf{Q}}, \tilde{\mathbf{K}}, \tilde{\mathbf{V}}$，$\mathbf{W}_{\tilde{q}}, \mathbf{W}_{\tilde{k}}, \mathbf{W}_{\tilde{v}}$ 是不同于第一层注意力的新的映射矩阵。每个 token 的关注范围仍然是之和其前后一定范围内的 token，而窗口长度 w_2 要比第一层的窗口长度 w_1 要大，从而 token 能够看到更大范围的内容，极端情况下，w_2 也可以等于序列长度，这样 token 的接受窗口就都是整个序列了。

在扩大信息接受面的同时，为了达到节省计算资源的目的，Poolingformer 会对接受窗口（以第 i 个 token 为例）$\mathcal{N}(i, w_2)$ 所对应的序列 $\widetilde{\mathbf{K}}_{\mathcal{N}(i,w_2)}$ 与 $\widetilde{\mathbf{K}}_{\mathcal{N}(i,w_2)}$ 进行池化操作：

$$\bar{\mathbf{K}}_i = \text{Pooling}(\widetilde{\mathbf{K}}_{\mathcal{N}(i,w_2)}; \kappa, \xi) \tag{2.12}$$

$$\bar{\mathbf{V}}_i = \text{Pooling}(\widetilde{\mathbf{V}}_{\mathcal{N}(i,w_2)}; \kappa, \xi) \tag{2.13}$$

$\bar{\mathbf{K}}_i$，$\bar{\mathbf{V}}_i$ 表示经过池化的 key,value 序列。κ，ξ 表示池化的核大小与步长。在池化操作中，每隔 ξ 个 token，就会由相邻的 κ 个 token 计算得到一个 token。因此，经过池化后，$\bar{\mathbf{K}}_i$ 与 $\bar{\mathbf{V}}_i$ 的长度比 $\widetilde{\mathbf{K}}_{\mathcal{N}(i,w_2)}$，$\widetilde{\mathbf{V}}_{\mathcal{N}(i,w_2)}$ 短了 κ 倍，通过池化操作，达到了缩短序列长度，压缩信息的目的，实际的窗口长度只有 $\frac{w_2}{\kappa}$，节省了计算资源。

所以第二层注意力输出 $\mathbf{Z}$ 的第 i 个 token z_i 为：

$$z_i = \text{Softmax}\left(\alpha \tilde{q}_i \bar{\mathbf{K}}_i^{\top}\right)\bar{\mathbf{V}}_i \tag{2.14}$$

Poolingformer 以残差连接的形式串联了第一层与第二层注意力，经过两层注意力计算后的输出 $\mathbf{X}$ 可以表示为：

$$\mathbf{X} = \mathbf{W}_o(\mathbf{Y} + \mathbf{Z}) \tag{2.15}$$

$\mathbf{W}_o \in \mathbb{R}^{d\times d}$ 是输出映射矩阵。

池化，压缩操作的种类很多，本文选择了均值，最大值，卷积三种进行实验对比，并将实验结果记录于（）节中。其中，卷积池化采用的是由[16]介绍的一

种需要训练的轻量级动态卷积池化机制。本实验中，以 $(v_1, ..., v_\kappa)$ 作为一个卷积核的 token 序列片段，卷积结果为：

$$\text{LDConv}(v_1, ..., v_\kappa) = \sum_{i=1}^{\kappa} \delta_i \cdot v_i \tag{2.16}$$

其中 $(\delta_1, ..., \delta_\kappa)$ 为动态卷积权重，由片段的中心 token$v_{\lceil \frac{1+\kappa}{2} \rceil}$ 通过一个可训练映射矩阵 $\mathbf{W}_p$ 进行映射得到，即

$$(\delta_1, ..., \delta_\kappa)^T = \text{Softmax}(\mathbf{W}_p v_i) \tag{2.17}$$

三、复杂度分析

在第一层注意力中，如果没有全局 token，每个 token 只要和接收窗口内的 $2w_1$ 个 token 计算注意力，所以复杂度为 $\mathcal{O}(w_1 n)$。若有 g 个全局 token，每个 token 则多和这 g 个 token 计算注意力，增加的复杂度为 $\mathcal{O}(gn)$。在第二层注意力中，池化后的接收窗口序列长为 $\frac{w_2}{\kappa}$，所以其复杂度为 $\mathcal{O}(\frac{w_2}{\kappa} n)$。$w_1$，$g$，$w_2$ 都是可以设置的参数，相对较小且与序列长度无关，所以对整个 Poolingformer 模型来说，时空复杂度为 $\mathcal{O}(n)$。这意味着资源开销只随着序列长度线性增长，解决了 Transformer 模型 $\mathcal{O}(n^2)$ 复杂度，面对长文本任务时资源开销过大的问题。

第三节　Deberta 的相对位置编码机制

Deberta[7] 的相对位置编码的核心思想便是表征两个 token 之间的相对位置关系。在 Transformer 的自注意力计算中，计算两个 token 的注意力分数时，只需要进行一次点乘 $QK^\top$，query 与 key 都主要携带着 token 内容相关的信息。而 Deberta 在计算分数时需要三次点乘；以第 i 与第 j 个 token 为例，除了两者的内容分数 $QK^\top$ 外，还要将 query 与 key 分别与两者相对位置的表征 $P_{i|j}$ 与 $P_{j|i}$ 进行点乘，再将三者的结果相加作为两个 token 的注意力分数。可以说，Deberta 的注意力分数是由内容-内容，内容-位置，位置-内容三部分组成的。以 k 表示最大相对位置，则第 i 相对于第 j 个 token 的位置 $\delta(i, j)$ 表示为：

$$\delta(i,j) = \begin{cases} 0 & i - j \leqslant -k \\ 2k-1 & i - j \geqslant k \\ i - j + k & -k < i - j < k \end{cases} \tag{2.18}$$

自注意力计算可以表示为：

$$\widetilde{A}_{i,j} = \mathbf{Q}_i^c \mathbf{K}_j^{c\top} + \mathbf{Q}_i^c \mathbf{K}_{\delta(i,j)}^{r\top} + \mathbf{K}_j^c \mathbf{Q}_{\delta(j,i)}^{r\top} \tag{2.19}$$

$$\mathbf{Y} = \mathrm{Softmax}(\alpha\tilde{\mathbf{A}})\mathbf{V}_c \tag{2.20}$$

$\mathbf{Q}_c = \mathbf{W}_{q,c}\mathbf{X}, \mathbf{K}_c = \mathbf{W}_{k,c}\mathbf{X}, \mathbf{V}_c = \mathbf{W}_{v,c}\mathbf{X}$ 是输入序列映射后得到的 query，key，value 内容序列，而 $\mathbf{W}_{q,c}, \mathbf{W}_{k,c}, \mathbf{W}_{v,c} \in \mathbb{R}^{d\times d}$ 为对应内容映射矩阵。而 $\mathbf{P} \in \mathbb{R}^{2k\times d}$ 为相对位置编码的嵌入矩阵，Deberta 最多可以表征 $2k$ 种相对位置，$\mathbf{P}$ 的一行就对应了一种相对位置，而且 $\mathbf{P}$ 是模型的全部层共用的。在每一层里，各自不同的 $\mathbf{W}_{q,r}, \mathbf{W}_{k,r} \in \mathbb{R}^{d\times d}$ 位置映射矩阵对 $\mathbf{P}$ 映射后，生成 $\mathbf{Q}_r = \mathbf{W}_{q,r}\mathbf{P}$ 与 $\mathbf{K}_r = \mathbf{W}_{k,r}\mathbf{P}$ 为两个经过映射的相对位置编码矩阵。

因为 $\mathbf{A}$ 中的分数是由三项相加得来，正则化因子也改为了 $\frac{1}{\sqrt{3d}}$，这能使训练更加稳定。

由于 Transformer 的序列长度限制一般为 512，Deberta 将最大相对位置 k 也设为了 512。已经训练好的相对位置编码不能再准确表征距离更远的相对位置，这也为 Deberta 的模型参数用于处理文本更长的任务带来了不便。

第四节　Poolingformer 与 Deberta 的融合模型

Deberta 的相对位置编码可以更准确地编码 token 的位置信息。因此，将这种相对位置编码引入 Pooingformer，引入在第一层的滑动窗口自注意力中，也可以让模型更好的获得位置信息。

在这个融合模型的第一层注意力中，token 只和接受窗口里的 token 计算注意力分数，这一点和 Poolingformer 一致。而计算分数时，由内容-内容，内容-位置，位置-内容三部分分数再相加得到，这一点和 Deberta 是一样的。设 w_1 为窗口长度，窗口 $\mathcal{N}(i, w_1)$ 为：

$$\mathcal{N}(i, w_1) = \{i - w_1, ..., i, ..., i + w_1\} \tag{2.21}$$

注意力计算公式为：

$$\widetilde{A}_{i,j} = \mathbf{Q}_i^c \mathbf{K}_j^{c\top} + \mathbf{Q}_i^c \mathbf{K}_{\delta(i,j)}^{r\top} + \mathbf{K}_j^c \mathbf{Q}_{\delta(j,i)}^{r\top} \tag{2.22}$$

$$\mathbf{Y} = \mathrm{Softmax}(\alpha\tilde{\mathbf{A}})\mathbf{V}_c \tag{2.23}$$

因为只和接收窗口里的 token 计算注意力，所以 $j \in \mathcal{N}(i, w_1)$。

值得注意的是，在没有全局 token 时，在滑动窗口自注意力的限制下，最远的相对位置的距离绝对值便是窗口长度 w_1。而相对位置编码 $\mathbf{P}$ 只能表征最长 k 的相对位置，所以必然有

$$w_1 \leqslant k \tag{2.24}$$

不然就会出现相对位置编码不够，或者说越界的情况。幸运的是，消融实验得到的结论是最优 $w_1 = 128$，这也是主实验所使用的设置，满足小于 $k = 512$ 的要求，当然这也是 Poolingformer 能够高效利用距离 token 最近的这些最有效的信息的证明。

一、全局 token 与相对位置编码

在存在全局 token 的情况下，如果不加以特殊处理，相对位置编码就一定会越界。因为全局 token 要和整个序列进行注意力计算，而且全局 token 一般置于整个序列的起始位置，所以最远的相对位置的距离绝对值为序列长度 N。在长文本的情况下，N 可以为 4096 乃至 16384，远远大于 $k = 512$。

为了解决这个问题，在计算全局 token 与其他 token 的注意力时，必须有额外的相对位置编码以供使用。因此，融合模型设置了额外的全局相对位置编码表征 $\mathbf{P}_g \in \mathbb{R}^{2k_g} \times d$，$k_g$ 为额外设置的表征对的数量，每一对表征包括相对位置为正、负的情况各一。当计算全局 token 与其他 token 的注意力时，注意力分数 A 的计算公式改为：

$$\widetilde{A}_{i,j} = \mathbf{Q}_i^c \mathbf{K}_j^{c\top} + \mathbf{Q}_i^c \mathbf{K}_{\mathcal{F}(\delta(i,j))}^{r,g\top} + \mathbf{K}_j^c \mathbf{Q}_{\mathcal{F}(\delta(i,j))}^{r,g\top} \tag{2.25}$$

$\mathbf{Q}_{r,g}, \mathbf{K}_{r,g}$ 为经过映射的全局相对位置编码矩阵，而 $\mathbf{W}_{q,r,g}, \mathbf{W}_{k,r,g} \in \mathbb{R}^{d\times d}$ 仍然为每层不同的位置映射矩阵。$\mathcal{F}$ 为从相对位置 $\delta(i,j)$ 到全局相对位置编码表征序号的映射，由相对位置决定使用哪个全局相对位置编码表征，即 $\mathbf{P}_g$ 的哪一列。$\mathcal{G}$ 为全局 token 集。为了节省显存，使 $\mathbf{P}_g$ 不至于太大，k_g 设置为远小于 N，即将 k_g 个表征均匀连续地分配给 N 个相对位置编码是比较合理的方案。几个相邻的相对位置能共用一个编码，节省了空间且不会对模型对相对位置的感知能力造成太多损害，在距离已经很远的情况下，准确的距离信息也不是必须的，模糊的信息，知道大概有多远就足够了。

融合模型的第二层注意力与 Poolingformer 相同。

第三章　实验

第一节　对 Poolingformer 模型的实验

对于 Poolingformer，本文在长文本摘要任务上用 arXiv、PubMed[17] 两个数据集测试了模型的效果，并进行了一系列消融实验，证明了双层注意力机制的有效性并寻找最佳参数。

一、数据集：arXiv 与 PubMed

表 3.1　arXiv 与 PubMed 数据集的相关数据

	数据集大小（条）			输入长度		输出长度	
数据集	训练集	测试集	开发集	中位数	90 分位数	中位数	90 分位数
arXiv	203037	6436	6440	6151	14405	171	352
PubMed	119924	6633	6658	2715	6101	212	318

arXiv 的数据由一个各学科的论文预印本收集网站 arxiv.org 上的论文收集、整理而来[17]，模型需要由输入的论文正文生成论文的摘要，完成摘要任务。该数据集的文本长度较长（见表 3.1），最满足期望的测试场景。

与 arXiv 类似，PubMed 数据集中模型同样需要由论文的正文生成论文的摘要，而其数据来自于生命科学与医学领域的数据库及搜索引擎 pubmed.ncbi.nlm.nih.gov。该数据库包含了十万余篇生物医学方面的论文。

和之前的工作一致，本文使用 ROUGE 分数[18] 作为衡量生成摘要与参考摘要文本相似程度的标准。ROUGE 分数中本实验选用了 ROUGE-1，ROUGE-2，ROUGE-L 三个分数，能以句子为单位衡量文本之间单词、词组与最长公共子序列的相似程度。

二、实验细节与参数设置

因为摘要任务需要生成文本，本实验的模型采用了 Encoder-Decoder 架构，编码器和解码器各有 12 层。由于摘要任务的输出文本并不长，所以仅编码器应用了 Poolingformer 的双层注意力机制，而在解码器的所有层中仍保留完全自注意力机制，来最好的优化输出文本。从头开始训练一个 Transformer 模型往往并不划算，需要消耗极大量的时间与算力，加载一个经过了很好调优，且结构差不多的预训练模型，是通用的做法。与之前的一些工作[9] 类似，本实验加载了

BART[5] 的预训练参数，这样就可以最公平地对比实验结果。由于序列最大长度大大增加，BART 的位置编码也必须扩展。实验选择了和 Longformer 相同的做法，将 BART 的位置编码循环拷贝，得到了更长的位置编码。

对于 12 层的编码器，也不是所有层的注意力计算都完全一样。第六到第十一层，也就是后 6 层采用了双层的注意力机制，而其余层则只计算其第一层注意力，即滑动窗口注意力。这是因为相对遥远的信息对靠后的层更加有价值。序列的第一个 token 将作为全局 token，这也和前人的工作一致。

第一层和第二层注意力的接收窗口长度分别为 128 和 512，池化的核大小与步长分别为 5 和 4。嵌入长度 d 为 1024，分为 16 个多头注意力的头。输入序列的最大长度为 4096 或 16384，而输出序列的最大长度为 256。池化运算选择了卷积。在生成输出文本时，beam 的个数为 5，而 length penalty（控制生成的文本长度）设置为 2，no repeat ngram（控制能重复出现的子序列，即词组的最大长度）设为 0。

三、实验结果

表 3.2　相关模型在 arXiv 测试集上的结果。

模型	ROUGE-1	ROUGE-2	ROUGE-L
Sent-PTR[19]	42.32	15.63	38.06
Extr-Abst-TLM[19]	41.62	14.69	38.03
PEGASUS[20]	44.21	16.95	38.83
Dancer[21]	45.01	17.60	40.56
BigBird[12]	46.63	19.02	41.77
LED_{4k}[9]	44.40	17.94	39.76
LED_{16k}[9]	46.63	19.62	41.83
$\text{Poolingformer}_{4k}$	47.86	19.54	42.35
$\text{Poolingformer}_{16k}$	**48.47**	**20.23**	**42.69**

在加载了 BART 的预训练参数之后，Poolingformer 模型在 arXiv 训练集上分别以 4096、16384 的最大序列长度精调。在 PubMed 训练集上只进行了最大序列长度 4096 的精调，这是因为 PubMed 的输入长度相对没有那么长。精调的 batch size 为 128，以 2e-4（最大序列长度 4096）或 1e-4（最大序列长度 16384）的学习率在 10 个 epoch 中选择开发集上最优的模型，warmup step 为 1000。学习率调

度器为 Adam[22]，且学习率以线性方式衰减。

表 3.2展示了各模型在 arXiv 测试集上的结果。最上面的 3 行是一些过去的 SoTA 模型，这些模型有些通过从正文中选择句子来组合成摘要，也有些是基于 Transformers 的生成模型，但能一次处理的文本长度都不长，最大也就是 PEGASUS[20]，为 1024。

表格下方展示了 BigBird，LED，与 Poolingformer 模型的结果。这三个模型都基于 Transformer，且都采用了滑动窗口的全局的稀疏自注意力机制。但是，BigBird 与 LED 都只有单层自注意力，而 BigBird 还额外随机选择了一些除此之外的 token 对计算注意力。BigBird 加载了 PEGASUS 的预训练参数，并且在精调之前还进行了一段延长预训练。况且，PEGASUS 还是专门针对摘要任务设计的模型。LED，即 Longformer Encoder-Decoder 与 Poolingformer 都从 BART 的预训练参数加载，而且都没有进行延长预训练。BigBird 实验的最大序列长度为 3072。从表中可以看出，$\text{Poolingformer}_{16k}$ 的结果大幅超过了已有结果，成为了新的 SOTA。在同样的序列长度下比较，无论 4096 还是 16384，Poolingformer 都远远超过了 LED；即便是 $\text{Poolingformer}_{4k}$ 都在两项分数上超过了 LED_{4k}。

表 3.3　相关模型在 PubMed 测试集上的结果。

模型	ROUGE-1	ROUGE-2	ROUGE-L
Sent-PTR[19]	43.30	17.92	39.47
Extr-Abst-TLM[19]	42.13	16.27	39.21
PEGASUS[20]	45.97	20.15	41.34
Dancer[21]	46.34	19.97	**42.42**
BigBird[12]	46.32	20.65	42.33
$\text{Poolingformer}_{4k}$	**46.72**	**20.86**	42.00

表 3.3展示了各模型在 PubMed 的测试集上的结果。Longformer 没有报告 PubMed 上的结果。$\text{Poolingformer}_{4k}$ 在 ROUGE-1 和 ROUGE-2 两项分数上超过了现有 SOTA，在 ROUGE-L 上则没有超过。Poolingformer 没有能全面刷新最好结果，这可能是因为 PubMed 的输入长度相对 arXiv 没有那么长。但这也已经足够说明 Poolingformer 的在长文本摘要任务上的能力。

除了测试结果以外，在计算资源消耗上，Poolingformer 的双层注意力机制也优于 BigBird 与 LED 的单层注意力。虽然多了一层注意力计算，但在双层注意力机制下，第一层注意力的接收窗口可以显著缩短。LED 的（单侧）窗口长度

为 512，这意味着其复杂度为 $O(1024 \times n)$。而 Poolingformer 的一二级窗口长度分别为 128 和 512，其复杂度为 $O((256 + 1024/4) \times n)$。在最大窗口长度同为 512 的情况下，Poolingformer 的复杂度只有 LED 的一半。无论是复杂度还是任务指标，Poolingformer 都超过了现有模型，这正说明了双层注意力机制高效利用信息的能力。

四、相关消融实验

本节通过两组消融实验证明了池化注意力机制的合理性，并探究了窗口长度与池化运算的设置。为了节约计算资源，所有消融实验都使用编码器和解码器都是 6 层的 Poolingformer 模型，称之为 Poolingformer$_{base}$，并从模型形状参数相同的 BART$_{base}$ 加载预训练参数。除此之外，精调所使用的训练数据为从 arXiv 训练集中选择的 50000 条数据，约占其原有规模的四分之一。Poolingformer$_{base}$ 的嵌入长度 $d = 768$，含有 12 个头。输入序列的最大长度为 4096。池化的核大小与步长分别为 5 和 4。在生成输出文本时，beam 的个数为 5，而 length penalty（控制生成的文本长度）设置为 1，no repeat ngram（控制能重复出现的子序列，即词组的最大长度）设为 3。学习率设为 1e-4。第 3 到第 5 层，也就是后 3 层采用了双层的注意力机制，而其余层则只计算其第一层注意力。

表 3.4　Poolingformer$_{base}$ 在 arXiv 测试集上关于第一、第二层注意力窗口长度 w_1、w_2 的消融实验结果。

模型	w_1	w_2	ROUGE-1	ROUGE-2	ROUGE-L
poolingFormer$_{base}$	128	-	38.27	14.39	33.72
poolingFormer$_{base}$	256	-	38.54	14.43	33.89
poolingFormer$_{base}$	512	-	38.08	14.38	33.61
poolingFormer$_{base}$	128	256	38.39	14.34	33.67
poolingFormer$_{base}$	128	512	38.75	14.58	34.13

第一组消融实验探究了第一层与第二层注意力窗口长度的影响，采用卷积作为池化运算，结果展示于表 3.4中。在前面三行的实验中，w_2 一列留空，这三组实验完全没有计算第二层注意力，即所有层都只计算第一层的滑动窗口注意力。在后面两个实验中，第 3 到第 5 层，也就是后 3 层采用了双层的注意力机制，而其余层则只计算其第一层注意力。为了探究第二层池化注意力计算的有效性，可以选取各自的最优结果，前者为第二行，后者为第五行进行比较，可见后者优于前者。而且后者的窗口长度等效于 $128 + 512/4 = 256$，两者的计算开销几乎相同。这一比较说明了第二层注意力是有效的。

在这几组实验参数中，128 + 512 的窗口长度设置表现最好，因此主实验也采用了该窗口长度。

有趣的是，在第三行与第二行的对比中，将窗口长度从 256 增加到 512 反而导致结果出现了一个相对明显的下降。从常理来说，增大了接受窗口使 token 可以参考更多的信息，本应提高模型的表现。但实验结果告诉我们，256 的窗口长度下模型表现优于 128 或 512 的窗口长度。这也许说明，距离超过一定长度之后，与更远的 token 计算注意力不会带来新的信息，反而更像是干扰或一种噪声。这一点值得进行更加深入与细致的研究探讨。

表 3.5　Poolingformer$_{base}$ 在 arXiv 测试集上关于池化方式的消融实验结果。

模型	池化方式	ROUGE-1	ROUGE-2	ROUGE-L
poolingFormer$_{base}$	卷积	38.75	14.58	34.13
poolingFormer$_{base}$	均值	38.68	14.60	34.05
poolingFormer$_{base}$	最大值	38.42	14.45	33.84

第二组消融实验中后 3 层采用了双层的注意力机制，而分别采用了不同的池化方式，以研究池化对模型的影响，其结果见表 3.5。对比均值、最大值、卷积三种池化方式，在两项分数上卷积池化取得了第一名，而在 ROUGE-2 上是均值池化取得了最好结果。从分数来看，卷积池化与均值池化差异很小。最终，主实验采用了卷积池化的方式进行实验。

第二节　融合模型的对比实验

为了能将 Poolingformer 与 Deberta 的融合模型与 Poolingformer 进行对比，必须让这两个模型在相同的数据集上，以相同的设置完成相同的任务。由于 Deberta 并未被设计用于摘要这类要求生成文本的任务，所以用摘要任务来进行对比实验不可行。实验采用在 Natural Questions 数据集上的问答任务对两个模型进行比较。

一、数据集：Natural Questions

作为问答数据集，Natural Questions（简称 NQ）的问题来自于谷歌搜索引擎的真实问题，而每篇对应的文档则是维基百科的一个词条。模型需要从文档里找到 1 长答案包含回答的段落 2 短答案答案的具体范围文档也有可能不包含答案，模型也要能识别这种情况。由于 NQ 的测试集的 label 不公开，本实验在 NQ 的

开发集上进行测试，并汇报长答案与短答案的 F1 分数。

与之前的工作[23]类似，一篇文档会被通过滑动窗口的方式分割成多个片段。分割后片段长为 4096，而步长为 1568。而每一个样本，即每一次输入模型的数据不是一篇完整的文档，而是包含文档的一个片段，以及放置在开头的问题文本。由于答案只有一个，大部分文档片段不包含答案，所以如此分割必然使不包含答案的负样本较大程度地多于正样本，造成样本不均衡问题。和[24]一样，实验对负样本进行了下采样以减少其数量，比例为 0.5。得到输出序列后，用分类器预测每一个 token 是否为短答案的起始/结束 token。属同一个段落的 token 会经过均值池化得到这个段落的表示，用于预测此段落是否为长答案。所有段落的段落表示经过再一次的均值池化后得到的是这个文档片段的表征，并用以预测该片段有无答案。

二、实验细节，参数与结果

问答任务属于自然语言理解任务，不需要生成文本，所以只有编码器没有解码器。两个模型都采用了 24 层编码器，嵌入长度为 1024，分为 16 个头，其中第 15 到第 20 层计算层采用了双层的注意力机制，而其余层则只计算其第一层注意力。第一层和第二层注意力的接收窗口长度分别为 128 和 512，池化的核大小与步长分别为 5 和 4。因为是问答任务，问题 token 非常重要，所以实验将所有的问题 token 作为全局 token 集。

两个模型都从一个优秀的预训练模型上初始化，然后进行精调训练是省时省力的做法，但这也要求预训练模型满足对比实验的限制，比如模型大小一致。然而，最适合的预训练模型 Deberta 与 Roberta[25]之间存在诸多规格上的差异。同为 24 层的预训练模型，Deberta 的嵌入长度为 1536 而 Roberta 为 1024，这种差异会给模型的对比带来干扰。而融合模型如果也从 Roberta 加载预训练参数的话，相对位置编码矩阵 P 无法加载且两种模型之间位置编码方式差异过大，会使得融合模型的实验结果大幅度下降。最终，实验选择了对两种模型都自己进行预训练，这样就不再受到预训练模型形状参数的限制。预训练采用的语料为英文 Wikipedia，被很多预训练模型，包括 BERT[2]、RoBERTa[25]所采用，大小为 18G。预训练的任务为广泛使用的 MaskedLM[2]，即掩码语言模型，随机遮住一部分 token 再通过上下文预测它们，类似于完形填空。预训练的 batch size 为 128，以 1e-4 的学习率训练了 4 个 epoch。预训练在八块 NVIDIA Tesla V100 GPU 上进行了约一星期的时间，虽然比 Roberta 之类的优秀预训练模型相比训练量少了很

多，但也足以对两个模型进行初步比较了。

精调的 batch size 为 32，以 3e-5 的学习率训练了 2 个 epoch，warmup 阶段占训练的比例为 0.1。学习率调度器为 Adam[22]，且学习率以多项式方式衰减。

表 3.6　经过预训练与精调之后，Poolingformer 与融合模型在 NQ 测试集上的结果。包括长答案（LA）与短答案（SA）的精确率（P），召回率（R），与 F1 分数。

	LA			SA		
	P	R	F1	P	R	F1
Poolingformer	62.3	64.8	63.5	48.7	36.6	41.7
融合模型	65.4	66.8	66.1	50.1	42.3	45.9

表 3.6中展示了两个模型在预训练并且精调之后在 NQ 测试集上的结果。融合模型结果优于 Poolingformer，虽然出于计算资源所限，不能进行更大训练量的预训练，但这也部分说明了融合模型有其优点。

第四章　相关工作

为了解决 Transformers 的完全自注意力机制的 $\mathcal{O}(n^2)$ 在面对长文本任务时计算开销过大的问题，除了一些利用递归、循环的方式进行单向语言建模的模型之外，大致可以分为三大类，其计算复杂度对比见表 4.1。

第一类模型的特点是利用低秩逼近或核函数等方式来对自注意力计算进行近似。Synthesizer[26] 直接通过训练参数的方式由输入序列得到了注意力分数，因此能跳过两两计算点积的步骤。Performer[27] 利用核函数的方式在减少计算量的情况下对两两计算注意力进行近似。Linformer[28] 则是基于低秩逼近的思想，通过映射降低了 key、value 序列的长度。

第二类与第三类模型的共同点是不再让 token 两两之间都计算注意力，而是只挑选出一部分进行注意力计算。在第二类模型中，注意力计算模式，即哪个 token 要和哪些 token 计算注意力是由数据决定的。通常来说，第二类模型会通过一种相似性判断，将 token 分类进入不同的桶中，而只对同一个桶中的 token 计算注意力为主，或者再加入一些其他的注意力计算模式。Reformer[4] 参考了局部敏感哈希作为判断标准，而 Routing Transformer[29] 和 Cluster-Former[30] 则将 k-means 聚类算法纳入相似性判断中。

第三类模型中的注意力计算模式是由模型所确定的，Poolingformer 便归于

模型	复杂度
Transformer[1]	$\mathcal{O}(n^2)$
Synthesizer[26]	$\mathcal{O}(n^2)$
Performer[27]	$\mathcal{O}(n)$
Linformer[28]	$\mathcal{O}(n)$
Reformer[4]	$\mathcal{O}(n \log n)$
Routing Transformer[29]	$\mathcal{O}(n \log n)$
Cluster-Former[30]	$\mathcal{O}(n \log n)$
Longformer[9]	$\mathcal{O}(n)$
BigBird[12]	$\mathcal{O}(n)$
Compressed Attention[31]	$\mathcal{O}(n^2)$
Poolingformer	$\mathcal{O}(n)$

表 4.1　部分相关模型的复杂度

此类。最简单的模式便是分块计算注意力，token 之和同块内的 token 计算，比如 Blockwise[32] 。又或者与 Poolingformer 类似，在 token 的局部窗口内计算注意力，辅以全局或随机注意力，包括前面多次提到的 Bigbird[12] 与 Longformer[9]。除此之外，还有模型着眼于减少序列的长度，比如 Compressed Attention[31] 用跨步卷积的方式实现了这一目标。

第五章　结论

本论文提出了 Poolingformer，一种基于创新的双层自注意力机制的针对长文本的模型，且实现了线性复杂度。其第一层注意力采用了滑动窗口注意力机制，每个 token 只和全局 token 与其临近的接收窗口内的 token 计算注意力。而在额外的第二层注意力中，在接收窗口比第一层更大的同时，模型还通过池化操作压缩了序列长度，所以模型在获得更多信息的同时减少了计算量。在长文本摘要任务上，Poolingformer 在 arXiv 与 PubMed 数据集上都取得了 SoTA 的结果。此外，本文还在 Poolingformer 上融合了 Deberta 的相对位置编码机制，并通过预训练与问答任务说明了该融合模型的潜力。Poolingformer 的模型与结构值得未来的工作进行更多的研究探索，包括其在其他长文本任务上的表现，又或者再加入更多层注意力的模型表现如何等等。

参 考 文 献

[1] VASWANI A, SHAZEER N, PARMAR N, et al. Attention is all you need[J]. arXiv preprint arXiv:1706.03762, 2017.

[2] DEVLIN J, CHANG M W, LEE K, et al. Bert: Pre-training of deep bidirectional transformers for language understanding[J]. arXiv preprint arXiv:1810.04805, 2018.

[3] CHILD R, GRAY S, RADFORD A, et al. Generating long sequences with sparse transformers[J]. arXiv preprint arXiv:1904.10509, 2019.

[4] KITAEV N, KAISER Ł, LEVSKAYA A. Reformer: The efficient transformer[J]. arXiv preprint arXiv:2001.04451, 2020.

[5] LEWIS M, LIU Y, GOYAL N, et al. Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension[J]. arXiv preprint arXiv:1910.13461, 2019.

[6] QI W, YAN Y, GONG Y, et al. ProphetNet: Predicting future n-gram for sequence-to-SequencePre-training[C/OL]//Findings of the Association for Computational Linguistics: EMNLP 2020. Online: Association for Computational Linguistics, 2020: 2401-2410. https://www.aclweb.org/anthology/2020.findings-emnlp.217. DOI: 10.18653/v1/2020.findings-emnlp.217.

[7] HE P, LIU X, GAO J, et al. Deberta: Decoding-enhanced bert with disentangled attention[J]. arXiv preprint arXiv:2006.03654, 2020.

[8] LAN Z, CHEN M, GOODMAN S, et al. Albert: A lite bert for self-supervised learning of language representations[J]. arXiv preprint arXiv:1909.11942, 2019.

[9] BELTAGY I, PETERS M E, COHAN A. Longformer: The long-document transformer[J]. arXiv preprint arXiv:2004.05150, 2020.

[10] DAI Z, LAI G, YANG Y, et al. Funnel-transformer: Filtering out sequential redundancy for efficient language processing[J]. arXiv preprint arXiv:2006.03236, 2020.

[11] COHAN A, DERNONCOURT F, KIM D S, et al. A discourse-aware attention model for abstractive summarization of long documents[C/OL]//Proceedings of the 2018 Conference of the North American Chapter of the Association for Com-

putational Linguistics: Human Language Technologies, Volume 2 (Short Papers). New Orleans, Louisiana: Association for Computational Linguistics, 2018: 615-621. https://www.aclweb.org/anthology/N18-2097. DOI: 10.18653/v1/N18-2097.

[12] ZAHEER M, GURUGANESH G, DUBEY A, et al. Big bird: Transformers for longer sequences[J]. arXiv preprint arXiv:2007.14062, 2020.

[13] MICULICICH L, RAM D, PAPPAS N, et al. Document-level neural machine translation with hierarchical attention networks[J]. arXiv preprint arXiv:1809.01576, 2018.

[14] BA J L, KIROS J R, HINTON G E. Layer normalization[J]. arXiv preprint arXiv:1607.06450, 2016.

[15] TAY Y, DEHGHANI M, BAHRI D, et al. Efficient transformers: A survey[J]. arXiv preprint arXiv:2009.06732, 2020.

[16] WU F, FAN A, BAEVSKI A, et al. Pay less attention with lightweight and dynamic convolutions[J]. arXiv preprint arXiv:1901.10430, 2019.

[17] COHAN A, DERNONCOURT F, KIM D S, et al. A discourse-aware attention model for abstractive summarization of long documents[J]. arXiv preprint arXiv:1804.05685, 2018.

[18] LIN C Y. Rouge: A package for automatic evaluation of summaries[C]//Text summarization branches out. 2004: 74-81.

[19] SUBRAMANIAN S, LI R, PILAULT J, et al. On extractive and abstractive neural document summarization with transformer language models[J]. arXiv preprint arXiv:1909.03186, 2019.

[20] ZHANG J, ZHAO Y, SALEH M, et al. Pegasus: Pre-training with extracted gap-sentences for abstractive summarization[C]//International Conference on Machine Learning. PMLR, 2020: 11328-11339.

[21] GIDIOTIS A, TSOUMAKAS G. A divide-and-conquer approach to the summarization of academic articles[J]. arXiv preprint arXiv:2004.06190, 2020.

[22] KINGMA D P, BA J. Adam: A method for stochastic optimization[J]. arXiv preprint arXiv:1412.6980, 2014.

[23] ALBERTI C, LEE K, COLLINS M. A bert baseline for the natural questions[J]. arXiv preprint arXiv:1901.08634, 2019.

[24] LIU D, GONG Y, FU J, et al. Rikinet: Reading wikipedia pages for natural question answering[J]. arXiv preprint arXiv:2004.14560, 2020.

[25] LIU Y, OTT M, GOYAL N, et al. Roberta: A robustly optimized bert pretraining approach[J]. arXiv preprint arXiv:1907.11692, 2019.

[26] TAY Y, BAHRI D, METZLER D, et al. Synthesizer: Rethinking self-attention in transformer models[J]. arXiv preprint arXiv:2005.00743, 2020.

[27] CHOROMANSKI K, LIKHOSHERSTOV V, DOHAN D, et al. Rethinking attention with performers[J]. arXiv preprint arXiv:2009.14794, 2020.

[28] WANG S, LI B, KHABSA M, et al. Linformer: Self-attention with linear complexity[J]. arXiv preprint arXiv:2006.04768, 2020.

[29] ROY A, SAFFAR M, VASWANI A, et al. Efficient content-based sparse attention with routing transformers[J]. Transactions of the Association for Computational Linguistics, 2021, 9: 53-68.

[30] WANG S, ZHOU L, GAN Z, et al. Cluster-former: Clustering-based sparse transformer for long-range dependency encoding[J]. arXiv preprint arXiv:2009.06097, 2020.

[31] LIU P J, SALEH M, POT E, et al. Generating wikipedia by summarizing long sequences[J]. arXiv preprint arXiv:1801.10198, 2018.

[32] QIU J, MA H, LEVY O, et al. Blockwise self-attention for long document understanding[J]. arXiv preprint arXiv:1911.02972, 2019.