【摘　要】

随着人工智能的发展与兴起，近年来越来越多的领域在人工智能的帮助下取得了更好的发展，而人工智能在医学领域的应用更是当下十分热门的话题。本文将目光聚焦于基于自然语言处理的医疗问答系统的构建，借助 Transformer 强大且优秀的性能来处理复杂的医学问答数据并进行建模。

本文的主要研究工作如下：

（1）对模型的理论基础进行了一定的阐述，主要包括对注意力机制的简要介绍以及对 Transformer 的结构的介绍，为后续的模型构建提供理论基础。

（2）从原始 Transformer 模型入手，将其从适用于机器翻译问题的原始模型改进为适用于医学问答问题的模型，并详细介绍模型中关键部分的实现：1. 对话数据的预处理；2. 多头自注意力的实现；3. 模型的训练。

（3）从误差分析和问答系统的体验感受展现实验的结果，并对比 Transformer 与其他神经网络来对模型进行评估。

（4）用数学语言解释缩放点积使模型具有更稳定的梯度流以及为何三角函数位置编码包含相对位置信息。

关键词：　医学问答，自然语言处理，Transformer，自注意力

[ABSTRACT]

With the development and rise of artificial intelligence, more and more fields have made better development with the help of artificial intelligence in recent years, and the application of artificial intelligence in the field of medicine is a very hot topic at present. This paper focuses on the construction of medical question answering system based on natural language processing, and processes and models complex medical question answering data with the help of transformer's powerful and excellent performance.

The main research work of this paper is as follows:

(1)This paper expounds the theoretical basis of the model, mainly including a brief introduction to the attention mechanism and the structure of transformer, so as to provide a theoretical basis for the model construction later.

(2)Starting with the original transformer model, this paper improves it from the original model suitable for machine translation problems to the model suitable for medical QA system, and introduces the implementation of the key parts of the model in detail: 1 Preprocessing of dialogue data; 2. Realization of multi head self attention; 3. Model training.

(3)The experimental results are presented from the error analysis and the experience of question answering system, and the model is evaluated by comparing transformer with other neural networks.

(4)Mathematical language is used to explain why scaling point product makes the model have more stable gradient flow and why trigonometric function position coding contains relative position information.

Keywords: Medical QA, Natural Language Processing, Transformer, self-attention

目录

插图目录

表格目录

1 绪论

1.1 选题背景与意义

随着深度学习与基于深度学习的人工智能（Artificial Intelligence,AI）的发展与兴起，AI 开始被应用于各种领域之中。在如今的深度学习领域，我们一般都将目光聚焦于计算机视觉（Computer Vision, CV），语音识别（Voice Recognition）和自然语言处理（Natural Language Processing, NLP）等方向，而这些方向的研究中，对于各类医学数据的分析及建模是十分常见的。随着各类深度学习算法的不断被开发，神经网络（Neural Networks, NN）的不断深层次化，基于临床和问诊数据的 AI 辅助诊断技术正处于飞速发展之中，而下一世代的 AI 医疗辅助诊断技术将不仅仅满足于高精度，而是更希望 AI 像人一样思考甚至思维能力远高于人类。

医疗数据作为各类大数据中最常见的一类数据，其中包含很多不同类型的数据，对于图像数据有如 CT 扫描图片，核磁共振图片以及胃镜图片等，并且基于图像数据的辅助诊断系统更是已经成熟应用于新冠肺炎（COVID-19）的诊断上[1]。而相较于 CV 领域的发展，NLP 领域似乎看起更像一位厚积薄发的学生，在近几年因为 Transformer[2]的兴起而被推到深度学习中的主流地位。Transformer 虽然在刚发布时只是用于机器翻译领域的研究，但后续研究表明，基于 Transformer 的预训练模型（Pre-Trained Model）在各项 NLP 任务中都取得了最优的表现，因此，Transformer 已然成为 NLP 领域的首选框架。显而易见，在 Transformer 当道的当下 NLP 界，用来处理文本类医学数据的模型当属是 Transformer 的各类衍生模型，对基于这类模型的辅助诊断系统的研究也开始逐渐增多。

得益于各类基于 Transformer 的模型的发展，对于医疗文本数据的分析更加的得心应手，其中任务类型大致被分为以下几类：医疗命名实体识别（Named Entity Recognition, NER）, 医疗关系抽取 (Relationship Extraction , RE), 医疗文本分类 (Text Classification)。在早期的 NLP 领域，深度学习的方法还未被引用，对于这些问题的研究还是基于启发式规则的方法[3]，词典匹配的方法[4]以及机器学习的方法，如支持向量机 (Support Vector Machines SVM)[5], 最大熵[6], 条件随机场 (Conditional Random Field,CRF)[7]以及隐马尔科夫模型 (Hidden Markov Model,HMM)[8]等，而现代 NLP 领域更多的是利用 BERT[9]或是 GPT[10]等大规模预训练模型，这类模型以其超大的模型参数规模，较强的模型泛用能力以及优秀的模型综合性能在上述医疗文本任务中均有出色的表现。然而这些模型往往是基于各类医学专有数据集

或是临床数据集而构建，基本都是应用于临床分析或是大规模的疾病预测，往往在日常生活中，问答类（Questioning Answering，QA）数据会更加常见，为了让患者能够通过 AI 初步判断自己的患病情况，基于 NLP 的 QA 辅助诊断系统便进入研究的范围内。本文将基于开源的 Medical QA 数据集，利用 Transformer 构建一个 QA 辅助诊断模型，并研究模型的一部分数学意义以及如何去对参数进行优化。

1.2 本文的论文结构与章节安排

本文共分为六章，各章节内容安排如下：

第一章绪论。本章简单说明了本文章的选题背景与意义。

第二章模型的理论基础。本章将从注意力（Attention）机制的数学基础入手，进而尽可能使用数学语言来解释 Transformer 的原理。

第三章模型的实现以及参数的调整。本章将从最基础的 Encoder-Decoder 结构入手，构建本文所需要的 Transformer 模型，并针对数据集进行调参。

第四章模型的评估与总结。本章将对模型从误差，超参数的优化等方面，利用可视化技术进行评估并在最后作出总结。

2 模型的理论基础

本章对模型的理论基础进行了一定的阐述，主要包括对注意力机制的简要介绍以及对 Transformer 的结构的介绍，为后续的模型构建提供理论基础。

2.1 注意力机制

2.1.1 生物学中的注意力提示

自从经济学开始研究稀缺资源的分配以来，事实上我们正处于“注意力经济”时代，也就是说，人们的注意力被视为一种可交换的、有限的、有价值的稀缺商品。许多商业模式也被开发出来利用这一点：在音乐或视频流媒体服务中，我们要么把注意力集中在在广告上，要么花钱去屏蔽广告；为了在网络游戏世界中成长，我们要么把注意力花在游戏战斗上，帮助吸引新玩家，要么付费马上变得强大。简而言之，注意力不是免费的。

那么在生物学中，注意力是如何应用于视觉世界的呢？在当今十分普及的双组件（Two-Component）框架中，受试者基于自主性提示和非自主性提示有选择性地引导注意力的焦点。

非自主性提示是基于环境中物体的显眼程度。想象一下，加入你的面前有四个物品：一份报纸，一本书，一份试卷以及一个香蕉。所有的纸制品都是黑白的，唯独香蕉是黄色的。也就是说，这个香蕉在此视觉环境中是十分显眼的，人的注意力会不由自主的被吸引，所以你把视线集中在香蕉上。

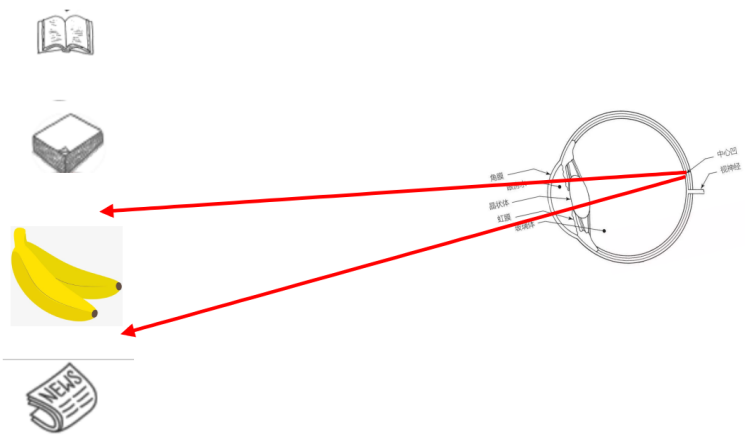


图 2.1 由于显著性的非自主性提示（黄色香蕉），注意力不自主地指向了香蕉

吃完香蕉后，你补充了足够的能量，于是想要读书来度过空闲时光。所以你转过头，重新聚焦你的眼睛，然后看看书。与由于显著性导致的选择不同，此时选择书是受到了意识的控制，因此注意力基于自主性提示去辅助选择时将更具有目的性。

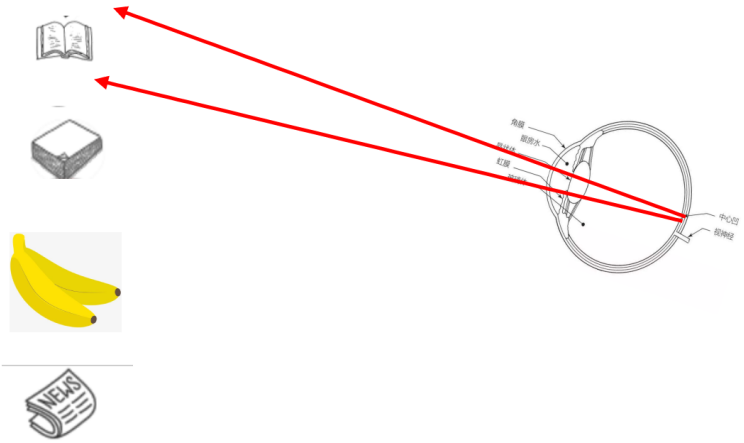


图 2.2 依赖于任务的自主性提示（想读一本书），注意力被自主引导到书上

2.1.2 查询（Query），键（Key）与值（Value）

自主性与非自主性的注意力提示展现了人类的如何通过注意力机制去获取信息，本节将通过这两种注意力提示，用神经网络来设计注意力机制的框架。

在 2.1.1 中，首先介绍了机制较为简单的非自主性提示。想要将选择偏向于感官输入，我们可以简单的使用非参数化的最大汇聚层（Maximum Pooling Layer）或是平均汇聚层（Average Pooling Layer）。

因此，“是否具有自主性提示”将注意力机制与普通的汇聚层区别开来。在注意力机制中，我们将自主性提示称为查询（Query）。给定任何查询，注意力机制通过注意力汇聚（Attention Pooling）将选择引导至感官输入。在注意力机制中，这些感官输入被称为值（Value），事实上每个值都有对应的键（Key），而这些键可以想象为非自主性提示。

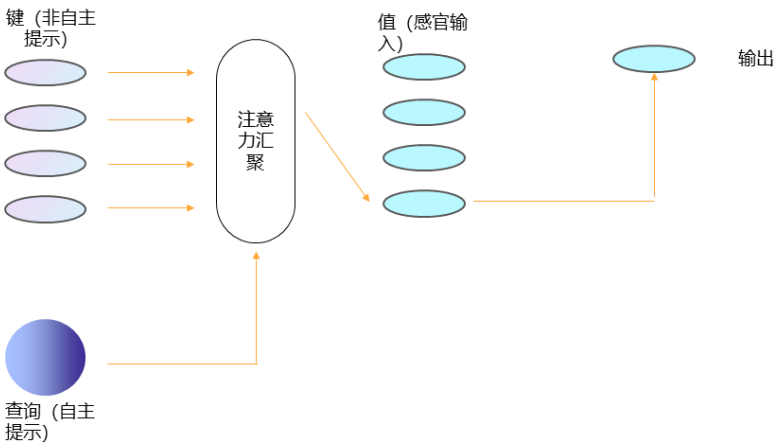


图 2.3 注意力机制通过注意力汇聚将查询（自主性提示）和键（非自主性提示）结合在一起，实现对值（感官输入）的选择倾向

2.1.3 Nadaraya-Watson 核回归与注意力汇聚

尽管注意力机制是近两年在深度学习领域最为重要的进展之一，但早在 1964 年，由 Nadaraya 和 Watson 提出的 Nadaraya-Watson 核回归[11]（下文简称 NW 核回归）可以作为注意力汇聚的一个早期统计模型。为了解释 NW 核回归，我们考虑以下这种简单的回归问题：

给定的成对的“输入 - 输出”数据集 $\{(x_1, y_1), \ldots, (x_n, y_n)\}$ 如何学习 f 来预测任意新输入的 x 的输出 $\hat{y} = f(x)$？

首先我们使用简单的模型来解决回归问题：基于平均汇聚来计算所有训练样本输出值得平均值：

$$f(x) = \frac{1}{n} \sum_{i=1}^{n} y_i \tag{2.1}$$

显然这个模型是不精确的，因为数据集在大多数情况下不是输出值恒定的，而不精确的原因主要是因为平均汇聚最大的问题：忽略了输入值 x_i 对模型的影响。NW 核回归的核心思想就是根据输入值 x 与 x_i 的关系来对 y_i 加权：

$$f(x) = \sum_{i=1}^{n} \frac{K\left(x - x_i\right)}{\sum_{j=1}^{n} K\left(x - x_j\right)} y_i \tag{2.2}$$

其中，K 是核函数（Kernel Function）。公式（2.2）所描述的模型被称为 Nadaraya-

Watson 核回归。受此启发，我们可以从图（2.3）的注意力机制框架的角度改写公式（2.2），成为一个更加通用的注意力汇聚公式：

$$f(x) = \sum_{i=1}^{n} \alpha\left(x, x_i\right) y_i \tag{2.3}$$

其中 x 是查询，(x_i, y_i) 是键值对（Key-Value pair）。事实上，从公式（2.3）可以看明显看出，注意力汇聚本质上就是对 y_i 的加权平均。我们将 $\alpha(x, x_i)$，也就是查询 x 与键 x_i 之间的关系称为注意力权重（attention weight），这个权重将被分配给每一个对应值 y_i。

2.1.4 注意力评分函数

事实上，在（2.3）式中，$\alpha(x, x_i)$ 作为权重，应该是一个概率分布函数，即 $\alpha(x, x_i)$ 满足：

$$0 \leq \alpha(x, x_i) \leq 1, i = 1, 2, \cdots, n$$

$$\sum_{i=1}^{n} \alpha(x, x_i) = 1$$

往往在实践中，很难找到这样一个函数 $\alpha(\cdot)$ 既能满足上述概率分布的条件，又能很好的刻画 x 与 x_i 的关系（数量关系或位置关系），因此我们将权重进一步改写为 $\alpha(a(x, x_i))$，其中，我们称 $a(\cdot)$ 为注意力评分函数（Attention Scoring Function），这样一来，我们就可以利用注意力评分函数来刻画 x 与 x_i 的关系，利用 $\alpha(\cdot)$ 来对注意力评分进行概率化。

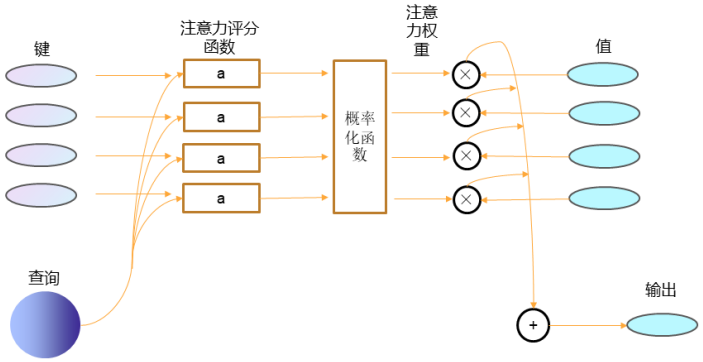


图 2.4 注意力汇聚的计算

用数学语言来表述上述过程，我们假设有一个查询 $\mathbf{q} \in \mathbb{R}^q$ 和 m 个 “键-值” 对 $(\mathbf{k}_1, \mathbf{v}_1), \ldots, (\mathbf{k}_m, \mathbf{v}_m)$ ，其中 $\mathbf{k}_i \in \mathbb{R}^k$， $\mathbf{v}_i \in \mathbb{R}^v$ 。注意力汇聚函数就被表示为加权和的形式：

$$f\left(\mathbf{q}, (\mathbf{k}_1, \mathbf{v}_1), \ldots, (\mathbf{k}_m, \mathbf{v}_m)\right) = \sum_{i=1}^{m} \alpha\left(\mathbf{q}, \mathbf{k}_i\right) \mathbf{v}_i \in \mathbb{R}^v \tag{2.4}$$

其中，查询 $\mathbf{q}$ 和键 $\mathbf{k}_i$ 的注意力权重是通过注意力评分函数 $a(\cdot)$ 将两个向量映射成标量，再经过概率化处理得到的：

$$\alpha\left(\mathbf{q}, \mathbf{k}_i\right) = \text{概率化}\left(a\left(\mathbf{q}, \mathbf{k}_i\right)\right) \tag{2.5}$$

事实上，在实践中我们一般用 softmax 函数来做概率化处理，于是（2.5）式可以写成：

$$\alpha\left(\mathbf{q}, \mathbf{k}_i\right) = \text{softmax}\left(a\left(\mathbf{q}, \mathbf{k}_i\right)\right) = \frac{\exp\left(a\left(\mathbf{q}, \mathbf{k}_i\right)\right)}{\sum_{j=1}^{m} \exp\left(a\left(\mathbf{q}, \mathbf{k}_j\right)\right)} \in \mathbb{R} \tag{2.6}$$

显然，此处的注意力评分函数的选择成为了下一个重点。回顾 2.1.4 中的 NW 核回归，我们对比（2.2）式与（2.6）式，可以看出 NW 核回归中的核函数 $K(\cdot)$ 也就是在注意力机制下的注意力评分函数。我们可以观察到，核函数仅仅是包含了 x 与 x_i 的信息，而在机器学习的角度，我们更希望注意力评分函数中能包含可学习的要素（包括但不仅限于可学习的参数），这样利用数据集进行训练可以尽可能得到合适的注意力评分函数。

此处介绍两个常用的注意力评分函数：

1) **加性注意力评分函数**

给定查询 $\mathbf{q} \in \mathbb{R}^q$，键 $\mathbf{k} \in \mathbb{R}^k$ 加性注意力评分函数定义为：

$$a(\mathbf{q}, \mathbf{k}) = \mathbf{w}_v^\top \tanh\left(\mathbf{W}_q \mathbf{q} + \mathbf{W}_k \mathbf{k}\right) \in \mathbb{R}$$

其中可学习的参数为 $\mathbf{W}_q \in \mathbb{R}^{h \times q}$ ， $\mathbf{W}_k \in \mathbb{R}^{h \times k}$ 和 $\mathbf{w}_v \in \mathbb{R}^h$，值得注意是，此处 h 是一个超参数，也就是说 h 是一个需要在训练前设定的参数而并非学习出来的。显然，加性注意力评分函数一般用于当查询和键的维度不一致，即 $q \neq k$ 时。

2) **点积注意力评分函数**

与加性注意力不一样的是，点积注意力要求查询和键的维度一致，即 $q = k$。

点击注意力评分函数定义为：

$$a(\mathbf{q}, \mathbf{k}) = \mathbf{q}^\top \mathbf{k}$$

相较于加性注意力，虽然点积注意力缺少可学习的参数，但是其可高度并行化的优点，在计算效率层面要优于加性注意力，而这个优点往往在实践中更具有影响力。

2.2 Transformer

2.2.1 自注意力机制

在自然语言处理中，我们的数据集中的输入输出一般为词元序列，而注意力机制一般要求我们的数据拥有“查询，键-值”的结构，因此我们可以尝试将输入的同一组词元序列同时作为查询，键和值，这就是自注意力机制的原理，同时也是 Transformer 的核心模块。下面我们给出自注意力的数学定义：

给定一个由词元组成的输入序列 $\mathbf{x}_1, \ldots, \mathbf{x}_n$，其中任意 $\mathbf{x}_i \in \mathbb{R}^d (1 \leq i \leq n)$。该序列输出一个长度相同的序列 $\mathbf{y}_1, \ldots, \mathbf{y}_n$，其中：

$$\mathbf{y}_i = f\left(\mathbf{x}_i, \left(\mathbf{x}_1, \mathbf{x}_1\right), \ldots, \left(\mathbf{x}_n, \mathbf{x}_n\right)\right) \in \mathbb{R}^d \tag{2.7}$$

$f(\cdot)$ 为（2.4）式中定义的注意力汇聚函数。

2.2.2 多头自注意力机制

事实上，在实践中，数据集往往是给定的，也就意味着“查询，键-值”往往是固定的，我们更希望模型基于相同的自注意力机制学习到更多的且不同的行为，然后将不同的行为组合起来以获取序列内的各种依赖关系。因此，允许注意力机制组合使用查询，键和值的不同子空间表示或许是有益的。

我们可以独立的去学习到 T 组不同的线性变换来变换查询，键和值。然后，这 T 组变换后的查询，键和值并行的送入自注意力汇聚中。最终，将这 T 个自注意力汇聚的输出拼接在一起，并通过另一个可学习的线性变换进行最终的整合以产生最终输出。我们称这种方式为多头自注意力，每一个自注意力汇聚都成为一个头（head）。在深度学习领域，线性变换可以通过全连接层来实现。

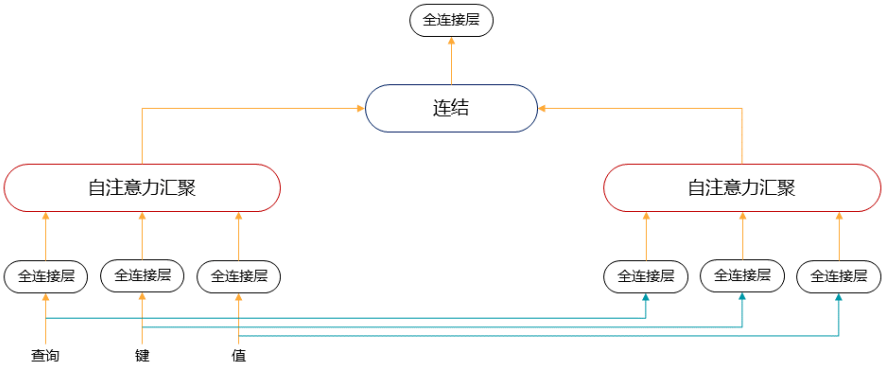


图 2.5 多头自注意力图示

用数学语言将这个模型表示出来：我们给定查询 $\mathbf{q} \in \mathbb{R}^{d_q}$，键 $\mathbf{k} \in \mathbb{R}^{d_k}$ 和值 $\mathbf{v} \in \mathbb{R}^{d_v}$，每个注意力头 $\mathbf{h}_i (i = 1, 2, \cdots, T)$ 的计算方法为：

$$\mathbf{h}_i = f\left(\mathbf{W}_i^{(q)}\mathbf{q}, \mathbf{W}_i^{(k)}\mathbf{k}, \mathbf{W}_i^{(v)}\mathbf{v}\right) \in \mathbb{R}^{p_v} \tag{2.8}$$

而在自注意力的背景下，查询，键和值实际上都是输入的向量，因此可以令 $d_q = d_k = d_v = d$，$\mathbf{q} = \mathbf{k} = \mathbf{v} = \mathbf{v}_{input}$。其中可学习的参数为 $\mathbf{W}_i^{(q)} \in \mathbb{R}^{p_q \times d}$，$\mathbf{W}_i^{(k)} \in \mathbb{R}^{p_k \times d}$ 和 $\mathbf{W}_i^{(v)} \in \mathbb{R}^{p_v \times d}$。多头注意力的输出需要先进行连结（concat）然后经过另一个线性变换：

$$\mathbf{W}_o \cdot \text{Concat}\begin{bmatrix} \mathbf{h}_1 \\ \vdots \\ \mathbf{h}_T \end{bmatrix} \in \mathbb{R}^{p_o} \tag{2.9}$$

其可学习的参数是 $\mathbf{W}_o \in \mathbb{R}^{p_o \times T p_v}$。

2.2.3 位置编码

在多头自注意力机制还并未出现在自然语言处理领域时，对于“序列到序列（seq2seq）”型的任务主要是利用循环神经网络（Recurrent Neural Network，RNN）来进行建模。

简单来说，考虑一个由序列构成的样本。假设输入的序列是 $x_1, \ldots, x_T$，其中 x_t 是输入的文本序列中的第 t 个词元。在时间 t 步，循环神经网络将词元 x_t 的输入

特征 $\mathbf{x}_t$ 和上一步的隐状态 $\mathbf{h}_{t-1}$ 通过一个函数 $f(\cdot)$ 转换为当前步的隐状态 $\mathbf{h}_t$，即：

$$\mathbf{h}_t = f\left(\mathbf{x}_t, \mathbf{h}_{t-1}\right) \tag{2.10}$$

由（2.10）式可以看出，只要我们的输入序列是按照顺序输入的（物理顺序或逻辑顺序），由于循环神经网络的特性（具有时序性），我们不用再考虑每个元素的顺序问题。

对于多头自注意力机制，观察（2.8）式和（2.9）式可以发现，我们始终是利用整个输入序列进行运算而并未考虑序列本身具有的时序性质，相较于传统的循环神经网络，这样的做法可并行程度更高，但并未捕捉到序列的时序性，然而在很多“序列到序列（seq2seq）”型的任务，比如机器翻译，机器问答等任务中，序列的时序性是不可被忽略的性质。

为解决多头自注意力机制无法捕捉到序列时序信息的问题，我们在输入序列时通过添加位置编码（Positional Encoding）来注入绝对或相对位置信息。

用数学语言来表述位置编码的嵌入：假设 $\mathbf{X} \in \mathbb{R}^{n\times d}$ 为包含一个序列中 n 个词元的 d 维嵌入表示，位置编码使用相同形状的位置嵌入矩阵 $\mathbf{P} \in \mathbb{R}^{n\times d}$ 输出矩阵 $\mathbf{X}+\mathbf{P}$。

事实上，位置嵌入矩阵 $\mathbf{P}$ 既可以通过学习得到，也可以在训练前进行人为设置。

2.2.4 Transformer 的原理及结构

2.2.4.1 Transformer 的基本思想

相较于传统的循环神经网络，自注意力机制具有高度并行化的优势，如果使用纯粹的自注意力来设计深度学习的框架时很有吸引力的。事实上，在 Transformer 之前已经有结合循环神经网络和自注意力机制的模型[12] [13]，而 Transformer 模型完全基于注意力机制，舍弃了卷积结构或者循环神经网络。尽管 transformer 最初是应用于在文本数据上的序列到序列学习，但现在已经推广到各种现代的深度学习中，例如语言、视觉、语音和强化学习领域。

2.2.4.2 Transformer 的结构

Transformer 整体结构为“编码器-解码器”，所谓“编码器-解码器”结构，包含两个主要组件：第一个组件是一个编码器（Encoder）：它接受一个可变长度的序

列，将其编码为一个固定长度的编码状态。第二个组件是解码器（Decoder）：它将固定长度的编码映射成长度可变的序列。

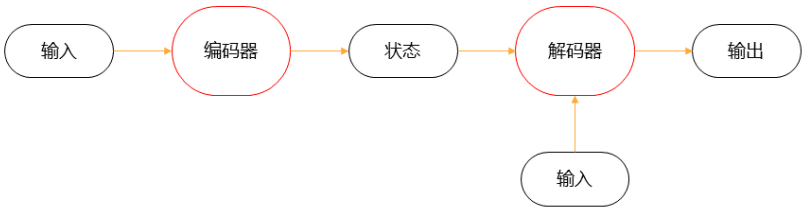


图 2.6 编码器-解码器

与传统的包含卷积层或者循环神经网络层的“编码器-解码器”不同，Transformer 的编码器和解码器都是由多个基于多头自注意力的层叠加而成的。

从宏观上来讲，Transformer 的编码器是由多个相同的层叠加而成的，每个层又由两个子层构成。第一个子层是多头自注意力汇聚，第二个子层是基于位置的前馈网络。具体来说，在计算编码器的自注意力时，查询、键和值都来自于上一个编码器层的输出。另外，受到残差网络（Resnet）[14]的启发，每个子层都采用了残差连接，在残差连接的加法运算后，再应用层规范化（Layer Normalization）[15]。

Transformer 的解码器也是由多个相同的层叠加而成，并且层中采用了残差连接和层规范化。除了拥有和编码器中描述的两个子层外，解码器还在这两个子层中插入了第三个子层，称为“解码器-编码器”注意力层。在“解码器-编码器”注意力层中，查询来自上一个解码器层的输出，而键和值来自整个编码器的输出。在编码器自注意力中，查询、键和值都来自于上一个解码器层的输出，但是解码器中的每个位置只能考虑该位置之前的所有位置，。这种掩码的设计，保留了输出序列的时序性，确保预测仅依赖于已生成的元素。

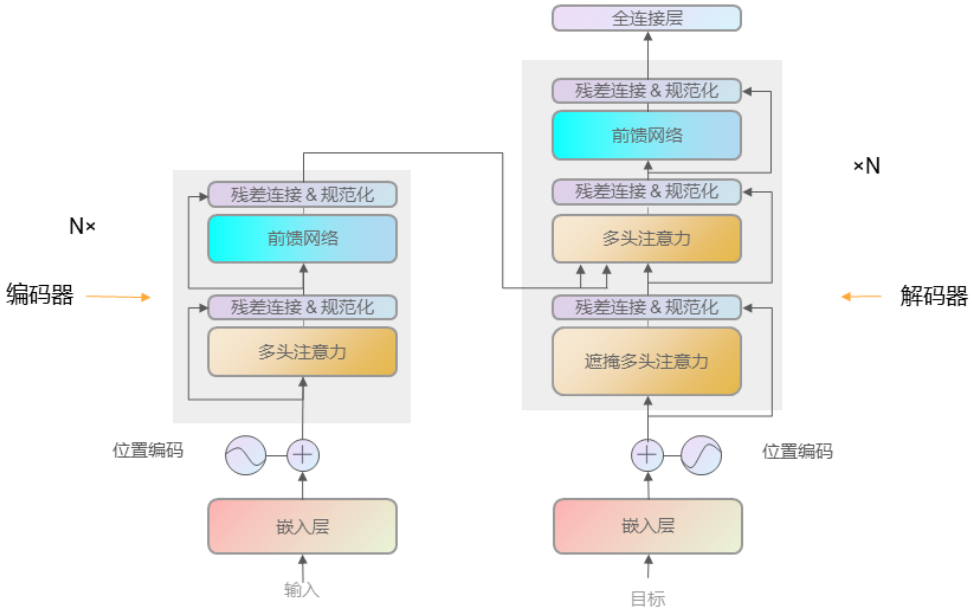


图 2.7 Transformer 的结构图

2.2.4.3 Transformer 中的细节

观察 Transformer 中的细节，我们发现，此时有两个模块需要确定：1. 自注意力评分函数的选择；2. 位置编码的选择。

1) **自注意力评分函数的选择**

在 Transformer 中，我们将选择缩放点积注意力评分函数：

$$a(\mathbf{q}, \mathbf{k}) = \frac{\mathbf{q}^\top \mathbf{k}}{\sqrt{d}} \tag{2.11}$$

d 为查询和键的维度。这样做的好处在于防止因为输入过大而导致在训练时，反向传播经过 softmax 层时梯度被意外“杀死”，具体的数学原理可参见附录 A。

2) **位置编码的选择**

此前我们说明过，位置编码函数是既可以别学习出来，也可以人为设定的，在 Transformer 中我们采用预先设定的位置编码函数：

$$p_{i,2j} = \sin\left(\frac{i}{10000^{2j/d}}\right) \tag{2.12}$$

$$p_{i,2j+1} = \cos\left(\frac{i}{10000^{2j/d}}\right) \tag{2.13}$$

其中 $p_{i,j}$ 为位置嵌入矩阵 **P** 的第 i 行，第 j 列的值。我们选择这样的位置编

码主要是基于三角函数可以同时包含绝对位置信息和相对位置信息，而且相较于学习出来的位置编码，不会受到字典的约束，具体的数学原理可参见附录 B。

2.3 本章总结

本章主要介绍了 Transformer 的一些理论基础。主要从注意力机制入手，介绍了注意力机制的生物学原理从而通过 NW 核回归引入正式的注意力机制的数学表达，然后介绍 Transformer 中使用到的多头自注意力机制以及位置编码，最后在“编码器-解码器”的框架下构建出 Transformer 模型。

3 基于医学问答数据的模型的实现

本章将从原始 Transformer 模型入手，将其从适用于机器翻译问题的原始模型改进为适用于医学问答问题的模型，并详细介绍模型中关键部分的实现：1. 对话数据的预处理；2. 多头自注意力的实现；3. 模型的训练。

3.1 对话数据的预处理

事实上，原始的 Transformer 模型是用于机器翻译问题的，然而问答问题与机器翻译问题既存在共同点：同为“序列到序列”（Seq2Seq）型问题，即翻译问题是解决句子到句子的翻译，问答问题是解决提问到回答的一一对应，两者也具有不同点：翻译问题至少同时包含了两个语种的处理，因此至少需要两个对应语种的字典以便于对数据进行嵌入式表示（Embedding），而问答问题往往是基于同一语种去进行建模的，因此我们只需要一个字典即可。

3.1.1 数据集的描述

本文使用的数据集是一个开源的中文医学对话数据集，原始数据集一共包含六个子数据集，总共包含 792099 个问答对，具体情况如下表：

表 3.1 数据集的基本信息

子数据集（按科室分类）	问答对数量
Andriatria-男科	94596
IM-内科	220606
OAGD-妇产科	183751
ncology-肿瘤科	75553
Pediatric-儿科	101602
Surgical-外科	115991
总计	**792099**

其中每个子数据集都是表格形式，其格式如下：

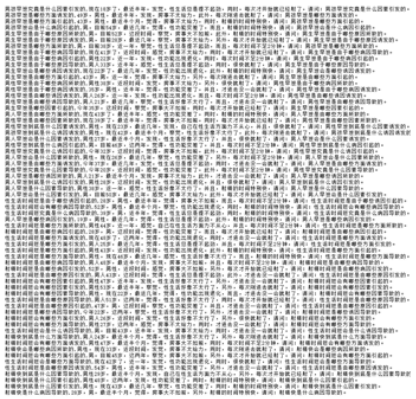
department	title	ask	answer
心血管科	高血压患者能吃党参吗？	我有高血压这两天女婿来的时候给我拿了些党参泡水喝，您好高血压可以吃党参吗？	高血压病人可以口服党参的。党参有降血脂，降血压的作用，可以彻底消除血液中的垃圾，从而对冠心病以及心血管疾病的患者都有一定的稳定预防工作作用，因此平时口服党参能远离三高的危害。另外党参除了益气养血，降低中枢神经作用，调整消化系统功能，健脾补肺的功能。感谢您的进行咨询，期望我的解释对你有所帮助。
消化科	哪家医院能治胃反流	烧心，打隔，咳嗽低烧，以有4年多	建议你用奥美拉唑同时，加用吗丁啉或莫沙必利或援生力维，另外还可以加用达喜片

图 3.1 子数据集格式

3.1.2 数据预处理

事实上，对于数据集来说，我们只需要提取如图 3.1 所示的 **ask** 和 **answer** 两列数据。提取数据后将提问数据和回答数据分别按顺序存储于 source 和 target 文件中，文件格式为 txt，形式如下：

source



target

图 3.2 文本文件的格式

观察已生成的两个文件，因为我们在提取提问和回答数据时，是按照子数据集分类，即科室分类提取的，因此同一子数据集的数据会成堆出现，如果直接用来

训练会导致在数据未完全经过模型就使得模型过拟合。因此我们需要对已提取的数据集进行打乱（shuffle）。与常规的打乱操作不同，因为此处是提问与回答一一对应的数据形式，因此我们需要对 source 文件和 target 文件采取相同的打乱方式，具体做法可见下面的代码：

```
import random

def random_sample(source,target,sample_num):
    idx = []
    while len(idx) < sample_num:
        new_idx = random.randint(0,len(source)-1)
        if new_idx not in idx:
            idx.append(new_idx)
    sampled_source = []
    sampled_target = []
    for i in idx:
        sampled_source.append(source[i])
        sampled_target.append(target[i])
    return sampled_source,sampled_target

def readFromPair(max_samples):
    f = open('./data/source.txt','r',encoding='utf-8')
    question = f.readlines()
    question = [i.replace('\n','') for i in question]
    f = open("./data/target.txt",'r',encoding = 'utf-8')
    answer = f.readlines()
    answer = [i.replace('\n','') for i in answer]
    if max_samples != -1:
        question,answer = random_sample(question,answer,max_samples)
    return question,answer
```

在最后，我们还需要把文字数据转换为模型可识别的数据序列，这里我们将用到基于字典的分词器（Tokenizer）。原理是将数据集中的每一行文字按照空格将每一个字分开，每一个字在字典中对应了一个编号，利用这些编号对句子进行编码，使其成为固定长度的序列，而这个固定长度我们称为最大序列长度，一般不小于所有数据中最长的数据的长度，而对于数据长度不足最大序列长度的数据，我们对剩余的部分采取填充（Padding）的形式，本文默认填充均为 0。

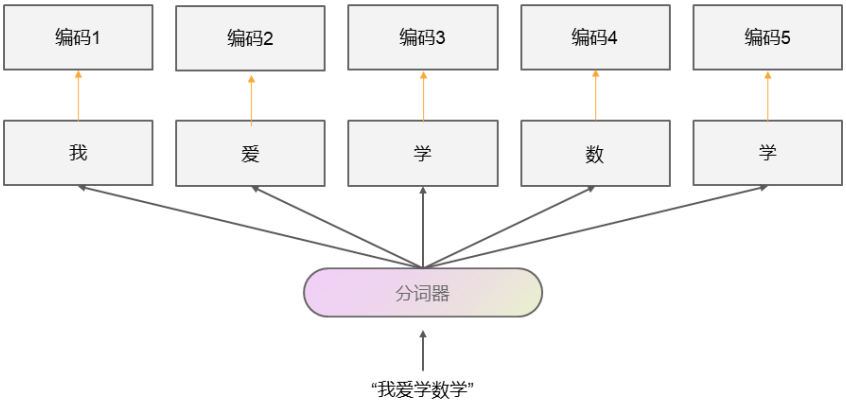


图 3.3 分词器

3.2 多头注意力的实现

回顾（2.2.4）节中我们介绍的 Transformer 的结构，事实上我们只需要分别实现一个编码器层和一个解码器层，然后通过完全复制的方式，生成任意多个编码器层和解码器层。同时我们观察到，编码器和解码器中均包含多头注意力子层，因此这将成为 Transformer 的核心部分。

在实现多头注意力前，我们需要注意一些细节：由于每个编码器与解码器中的子层之间都有残差连接，而残差连接对于输出数据要求必须与原始输入数据维度一致（本质上为 $x + output(x)$），我们为了满足模型内部数据维度的一致性，我们在整个 Transformer 模型之外定义一个超参数 d_{model}，作为模型内数据的标准维度并作为所有嵌入层的强制输出长度。

我们假设多头注意力中头（$head$）的个数为 h，回顾（2.2.2）节，多头注意力允许模型共同关注来自不同位置的不同表示子空间的信息，并且考虑到上述的维度限制，用数学表达模型中实际的运算如下：

$$\text{MultiHead}(Q, K, V) = \text{Concat}\left(\text{head}_1, \ldots, \text{head}_h\right) W^O \tag{3.1}$$

$$\text{where head}_i = \text{Attention}\left(QW_i^Q, KW_i^K, VW_i^V\right) \tag{3.2}$$

其中可学习的参数为 $W_i^Q \in \mathbb{R}^{d_{\text{model}} \times d_k}, W_i^K \in \mathbb{R}^{d_{\text{model}} \times d_k}, W_i^V \in \mathbb{R}^{d_{\text{model}} \times d_v}$ 和 $W^O \in \mathbb{R}^{hdv \times d_{\text{model}}}$。对于每个注意力头，我们令 $d_k = d_v = \frac{d_{model}}{h}$，这样做的可以使得多头注意力输出维度保持为 d_{model} 并且保持总计算成本与具有全维性的单头注意力相同。值得注意的是，其中所有的线性映射均可使用线性全连接层实现。

尽管编码器和解码器的基础结构差别不大，但是解码器需要对输入序列进行遮掩操作，以防止注意力关注后续位置，确保第 i 个位置的预测只依赖于第 i 个位置以前的已知的输出，并且解码器还插入第三个子层，该子层对编码器堆栈的输出执行多头注意。

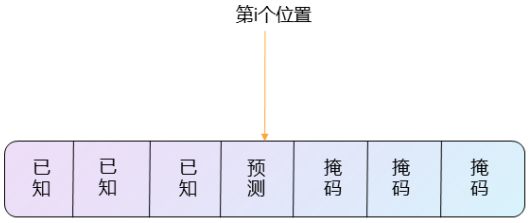


图 3.4 掩码的原理

3.3 模型的训练

在实际模型的训练中，我们要明确模型中的待学习的参数和超参数的数量，根据数据量选择训练用的设备，并且我们也需要明确模型训练的指标，即选取何种损失函数，最后在众多优化器（Optimizer）中选取适合的优化器。

3.3.1 参数的描述

首先我们必须明确模型中需要学习的参数和超参数。回顾（3.1）式和（3.2）式，多头自注意力子层中可学习的参数为 $W_i^Q \in \mathbb{R}^{d_{\text{model}} \times d_k}, W_i^K \in \mathbb{R}^{d_{\text{model}} \times d_k}, W_i^V \in \mathbb{R}^{d_{\text{model}} \times d_v}$ 和 $W^O \in \mathbb{R}^{hdv \times d_{\text{model}}}$，这些参数都可通过线性层实现，而 d_{model} 和 h 为超参数。

除了注意子层，编码器和解码器中的每一层都包含一个全连接的前馈网络，分别相同地应用于每个位置。这由两个线性变换组成，中间有一个 ReLU 激活：

$$\text{FFN}(x) = \max\left(0, xW_1 + b_1\right) W_2 + b_2 \tag{3.3}$$

我们在训练时，只需要指定 ReLu 内部的线性层的维数 d_{ff} 即可，其他的参数均可用线性全连接层实现。具体参数的信息可见下表：

表 3.2 参数信息

参数名	符号及部分取值
编码器和解码器层数	N=6
模型内部数据维度	d_{model}=512
前馈层的内部线性维度	d_{ff}=2048
注意力头的个数	h=8
键的维度	d_k=64
值的维度	d_v=64

3.3.2 损失函数的选择

实际上，我们模型中最后判断的依据是将输出序列与目标序列去比较是否为同一序列，所以我们在模型中采用交叉熵损失函数（CrossEntropy Loss）：

$$\text{Loss} = -\sum_{i=1}^{n} y_i \log y_i' \tag{3.4}$$

其中，y_i 是真实值，y_i' 是预测值。有一点需要注意，由数据处理一节可以看出，数据中含有大量的填充（Padding），这些填充不能作为真正的输入来考虑，因此在损失函数的计算中，需要将这些部分屏蔽掉。

3.3.3 优化器的选择

众所周知，机器学习的主要训练方式是通过优化器（Optimizer）对损失函数进行最优化，使得损失函数能达到全局最小。一般常见的优化器有：梯度下降优化；随机梯度下降优化；动量法等。但在本文中，我们将采用更适合用于深度学习的 Adam 优化器[16]。

Adam 算法实际上是是对随机梯度下降法的拓展，Adam 与经典的随机梯度下降法的区别在于，随机梯度下降保持一个单一的学习速率 (称为 alpha)，用于所有的权重更新，并且在训练过程中学习速率不会改变。每一个网络权重 (参数) 都保持一个学习速率，并随着学习的展开而单独地进行调整，而 Adam 算法从梯度的第一次和第二次矩的预算来计算不同参数的自适应学习速率。具体的算法如下：

算法 3.1: Adam 优化算法

输入: α: 步长

输入: $\beta_1, \beta_2 \in [0, 1)$: 矩估计的指数衰减率

输入: $f(\theta)$: 带参数 θ 的待优化函数

输入: θ_o: 参数初值

$m_0 \leftarrow 0$(初始化一阶矩向量),$v_0 \leftarrow 0$(初始化二阶矩向量),$t \leftarrow 0$(初始化步数)

while θ_t 未收敛 **do**:

$t \leftarrow t + 1$

$g_t \leftarrow \nabla_\theta f_t\left(\theta_{t-1}\right)$（获取在 t 步的函数的梯度）

$m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$（更新有偏的一阶矩估计）

$v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$（更新有偏的二阶原始矩估计）

$\widehat{m}_t \leftarrow m_t / \left(1 - \beta_1^t\right)$（计算修正偏差后的一阶矩估计）

$\widehat{v}_t \leftarrow v_t / \left(1 - \beta_2^t\right)$（计算修正偏差后的二阶原始矩估计）

$\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \widehat{m}_t / \left(\sqrt{\widehat{v}_t} + \epsilon\right)$（更新参数）

end while

Return θ_t（收敛的参数）

具体地说，该算法计算了梯度和平方梯度的指数移动平均值，并且参数 β_1 和 β_2 控制了这些移动平均的衰减率。移动平均值和 β_1 和 β_2 的初始值接近 1.0(推荐值)，这导致了估计时间的偏差为 0。这种偏差是通过第一次计算偏差估计然后再计算比可用偏差校正估计来克服的。

在本文中，我们取 $\beta_1 = 0.9, \beta_2 = 0.95, \alpha = 0.0002$。

3.3.4 硬件与训练时长

我们在一台搭载了 3 块 NVIDIA RTX3090 显卡的机器上训练模型。我们分别取 $batchsize = 32$ 和 $batchsize = 64$ 对模型进行 15 个 epoch 的训练，分别花费了约两个半小时和两个小时。

3.3.5 模型的正则化

在每一个子层的输出做残差连接和标准化前，我们对每一个子层的输出采用 dropout[17]。此外，我们对编码器和解码器堆栈中的嵌入和位置编码的和也应用了 dropout。在本文中，我们使用的 dropout 比例为 $P_{dropout} = 0.1$。

3.4 本章总结

本章主要从数据预处理，多头注意力的实现以及训练的细节入手，介绍了基于医学问答数据的 Transformer 模型的具体实现方法，描述了问答数据的配对性特征，介绍了基于字典的分词器，介绍了多头自注意力的实现方法，描述了本文的模型中的参数信息以及采用的损失函数类型和优化器，简要介绍了 Adam 优化器的实现方法，最后描述本文模型训练所用的设备以及花费的训练时间。

4 实验的结果与评估

本章将从误差分析和问答系统的体验感受展现实验的结果，并对比 Transformer 与其他神经网络来对模型进行评估。

4.1 误差分析

我们将数据集分为 训练集 : 测试集 $= 7:3$。回顾第三章的训练细节，我们分别取 $batchsize = 32$ 和 $batchsize = 64$ 对模型进行了训练, 并作出训练误差与训练轮数的关系图:

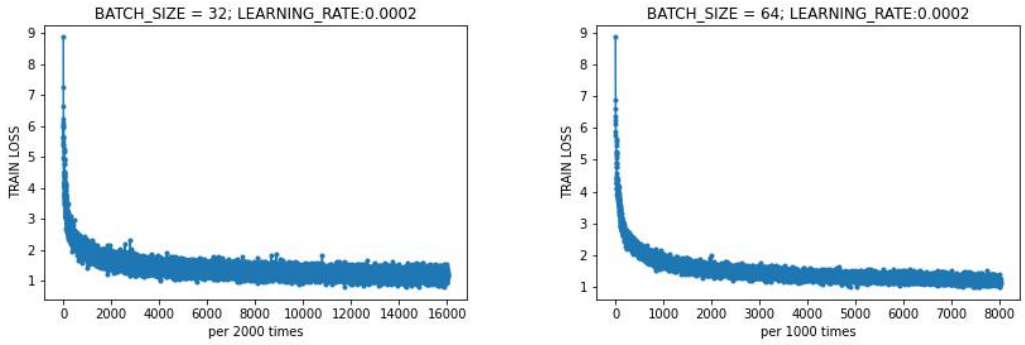


图 4.1 损失图

对应的测试误差如下:

batchsize	测试误差
32	1.1465
64	1.1751

表 4.1 测试误差

从图（4.1）中可看出，无论 $batchsize$ 取 32 还是 64，训练误差均能收敛到一个稳定的值。值得注意是，模型在前 2000 次训练中，训练误差几乎呈直线式下降，而在 2000 次以后训练误差的下降速率明显变缓。与一般的误差图不一样的是，本文的误差图中误差波动较大，这是因为我们在多个 GPU 上并行训练导致的，并不影响模型的训练。

当 $batchsize$ 取不同值的时候，训练误差收敛的速率是不一致的，实际上当

$batchsize$ 取 64 时，训练速度要快于取 32 的训练速度，但是在相同的 epoch 下，$batchsize$ 取 32 时的训练误差能收敛到更小的值。

4.2 问答系统的实测

我们在整个训练过程中，我们使用的数据是编码后的句子序列，但在实际应用中，我们希望能实现简单的人机交互，即现实意义上的问答系统，而我们需要做的就是将预测出的答案序列解码成真实的句子。

在实验中，我们采用贪心解码算法，简单来说就是每次选择概率值最大的对应的字进行输出。本文也简要地做了一些可交互界面，如下图所示：

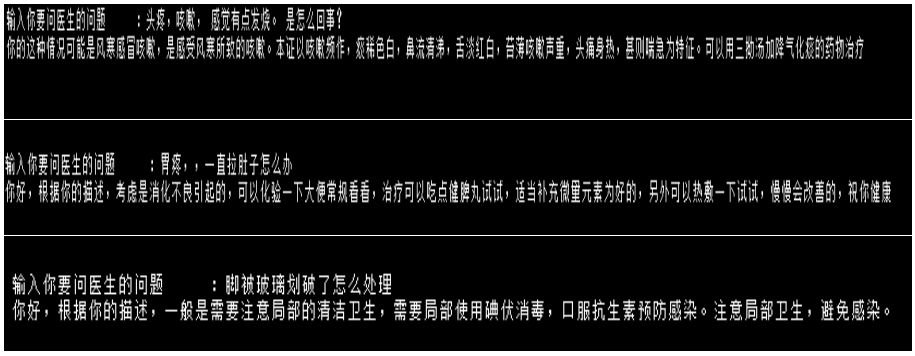


图 4.2 交互界面

4.3 模型评价与优势

为了比较 Transformer 与卷积神经网络和循环神经网络，我们假设输入序列由 n 个词元组成，并映射到另一个同样长度的序列，其中每个词元都是 d 维的嵌入表示，我们将比较自注意力机制（本质上 Transformer 是由多个自注意力层构成的），卷积层和循环层的计算复杂性，顺序操作以及最大路径长度：

网络类型	每层的计算复杂度	顺序操作	最大路径长度
自注意力	$O\left(n^2 \cdot d\right)$	$O(1)$	$O(1)$
循环神经网络	$O\left(n \cdot d^2\right)$	$O(n)$	$O(n)$
卷积神经网络	$O\left(k \cdot n \cdot d^2\right)$	$O(1)$	$O\left(\log_k(n)\right)$

表 4.2 三种网络的比较

这里的顺序操作刻画的是能并行运算的数量。最大路径长度指的是长期依赖关系之间的路径长度。在许多序列转换任务中，学习长期依赖关系是一个关键挑

战。影响学习此类依赖关系能力的一个关键因素是向前和向后信号所需路径的长度在网络中遍历。输入和输出序列中任何位置组合之间的路径越短，就越容易了解长期依赖关系，因此，我们还比较了由不同层类型组成的网络中任意两个输入和输出位置之间的最大路径长度。

从表（4.2）中可以明显看出，自注意力和卷积网络都具有很好的可并行性，而自注意力更是具有最小的最大路径长度，因此具有很优秀的运算效率。

4.4 存在的问题

4.4.1 模型本身的问题

就像上一节提到的，尽管 Transformer 具有高度并行化的优势，但是对于长序列的运算速度是很慢的（计算复杂度与 n^2 成正比），在处理超长序列时由于计算复杂度带来的劣势可能掩盖其并行化的优势。相较于传统的循环神经网络，Transformer 捕捉局部信息的能力较弱。位置编码同样也存在一些问题，在使用词向量的过程中，会做如下假设：对词向量做线性变换，其语义可以在很大程度上得以保留，也就是说词向量保存了词语的语言学信息（词性、语义）。然而，位置编码在语义空间中并不具有这种可变换性，它相当于人为设计的一种索引。那么，将这种位置编码与词向量相加，就是不合理的，所以不能很好地表征位置信息。由于 Transformer 中包含大量的线性层，因此也就存在大量的待训练参数，这使得 Transformer 相比于循环神经网络需要更多的数据量。

4.4.2 训练中的问题

由于训练设备的限制，我们只能训练较小的轮数，并且模型中的编码器与解码器的层数也只能设为比较小的数值，事实上我们是可以将模型拓展成较深的网络，但这需要更大量的数据，若在更适合深度学习的设备上可以适当拓展网络层数以及训练轮数，理论上训练效果会更好。

4.5 本章总结

本章主要是从误差分析的角度评估模型，并且实现了简单的交互界面，通过将自注意力与卷积神经网络和循环神经网络进行比较，分析了模型具有的优势，最后评述了模型本身的问题已经训练中存在的问题。

参考文献

[1] NING W, LEI S, YANG J, et al. Open resource of clinical data from patients with pneumonia for the prediction of covid-19 outcomes via deep learning[C/OL]//Nature. 2020. https://www.nature.com/articles/s41551-020-00633-5.

[2] VASWANI A, SHAZEER N, PARMAR N, et al. Attention is all you need[J]. Advances in neural information processing systems, 2017, 30.

[3] FUKUDA K I, TSUNODA T, TAMURA A, et al. Toward information extraction: identifying protein names from biological papers[C]//Pac symp biocomput: volume 707. Citeseer, 1998: 707-718.

[4] TUASON O, CHEN L, LIU H, et al. Biological nomenclatures: a source of lexical knowledge and ambiguity[M]//Biocomputing 2004. World Scientific, 2003: 238-249.

[5] TSOCHANTARIDIS I, HOFMANN T, JOACHIMS T, et al. Support vector machine learning for interdependent and structured output spaces[C]//Proceedings of the twenty-first international conference on Machine learning. 2004: 104.

[6] LIN Y F, TSAI T H, CHOU W C, et al. A maximum entropy approach to biomedical named entity recognition[C]//Proceedings of the 4th International Conference on Data Mining in Bioinformatics. Citeseer, 2004: 56-61.

[7] ZHOU G, SU J. Named entity recognition using an hmm-based chunk tagger[C]//Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics. 2002: 473-480.

[8] LI Y, LIN H, YANG Z. Incorporating rich background knowledge for gene named entity classification and recognition[J]. BMC bioinformatics, 2009, 10(1): 1-15.

[9] 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.

[10] FLORIDI L, CHIRIATTI M. Gpt-3: Its nature, scope, limits, and consequences[J]. Minds and Machines, 2020, 30(4): 681-694.

[11] NADARAYA E A. On estimating regression[J]. Theory of Probability & Its Applications, 1964, 9(1): 141-142.

[12] CHENG J, DONG L, LAPATA M. Long short-term memory-networks for machine reading[J]. arXiv preprint arXiv:1601.06733, 2016.

[13] LIN Z, FENG M, SANTOS C N D, et al. A structured self-attentive sentence embedding[J]. arXiv preprint arXiv:1703.03130, 2017.

[14] HE K, ZHANG X, REN S, et al. Deep residual learning for image recognition[C]//Proceedings of the IEEE conference on computer vision and pattern recognition. 2016: 770-778.

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

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

[17] SRIVASTAVA N, HINTON G, KRIZHEVSKY A, et al. Dropout: a simple way to prevent neural networks from overfitting[J]. The journal of machine learning research, 2014, 15(1): 1929-1958.

附录 A 缩放点积使模型具有更稳定的梯度流的数学解释

在大多数有关 Transformer 的文章中，我们了解到如果点积的数量级增大可以将 softmax 函数推入梯度极小的区域。在本节将利用数学语言来解释这个说法。

A.1 softmax 函数关于输入序列的敏感性

我们先讨论 softmax 函数在输入的值幅度较大时会产生什么变化。

我们知道，在注意力机制中，为了实现最后的加权和，必须对注意力得分函数的值进行概率化操作，这个操作就是 softmax 函数：

假设我们输入 softmax 函数的序列为 $(x_1, x_2, \cdots, x_n)$，输出的序列为 $(p_1, p_2, \cdots, p_n)$，则：

$$p_i = \frac{e^{x_i}}{\sum_i^n e^{x_i}} \tag{A.1}$$

则 p_i 满足：

$$0 < p_i < 1 \tag{A.2}$$

$$\sum_i^n p_i = 1 \tag{A.3}$$

我们考虑当输入序列扩大 k 倍时，p_i 的最大值 p_{max} 会有怎样的变化。

引理 A.1 p_i 关于 x_i 是单调递增的，即 softmax 函数是单调递增的。

这个引理的证明是显然的，根据这个引理我们可以得到：

$$p_{max} = \frac{e^{x_{max}}}{\sum_i^n e^{x_{max}}} \tag{A.4}$$

在输入序列扩大 k 倍后，（A.4）式变为：

$$p_{max}(k) = \frac{e^{kx_{max}}}{\sum_i^n e^{kx_i}} \tag{A.5}$$

根据引理 A.1，我们知道 $p_{max}(k)$ 关于 k 单调递增，根据（A.2）式，得到 $p_{max}(k)$

是有界的，根据单调有界原理可知，$p_{max}(k)$ 存在极限：

$$\lim_{k\to\infty} p_{max}(k) = \lim_{k\to\infty} \frac{e^{kx_{max}}}{\sum_i^n e^{kx_i}} = \lim_{k\to\infty} \frac{1}{1+\sum_{i\neq max} e^{k(x_i-x_{max})}} = 1 \tag{A.6}$$

因此，我们可以得到结论：随着输入序列扩大不断时，最大的输出值 p_{max} 会逐渐增大并趋向于 1，根据（A.3）这也导致其他的输出值会逐渐趋向于 0，因此，当输入序列过大时，我们的输出序列可能会被输入序列中的极大元素影响。

A.2 输入值对 softmax 函数的雅可比矩阵的影响

事实上，softmax 函数是一个 $\mathbb{R}^n \to \mathbb{R}^n$ 的函数，因此我们考虑 softmax 函数的雅可比矩阵：

$$J_{softmax} = \begin{pmatrix} \frac{\partial p_1}{\partial x_1} & \frac{\partial p_1}{\partial x_2} & \cdots & \frac{\partial p_1}{\partial x_n} \\ \frac{\partial p_2}{\partial x_1} & \frac{\partial p_2}{\partial x_2} & \cdots & \frac{\partial p_2}{\partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial p_n}{\partial x_1} & \frac{\partial p_n}{\partial x_2} & \cdots & \frac{\partial p_n}{\partial x_n} \end{pmatrix} \tag{A.7}$$

为了得到雅可比矩阵的具体表达，我们对矩阵中的元素 $\frac{\partial p_i}{\partial x_j}$ 进行运算。我们假设矩阵中的元素均是正值，则有：

$$\frac{\partial}{\partial x_j} \ln(p_i) = \frac{1}{p_i} \cdot \frac{\partial p_i}{\partial x_j} \tag{A.8}$$

因此，我们只需要求出 $\frac{\partial}{\partial x_j} \ln(p_i)$ 即可。又因为：

$$\ln p_i = \ln\left(\frac{e^{x_i}}{\sum_{l=1}^n e^{x_l}}\right) = x_i - \ln\left(\sum_{l=1}^n e^{x_l}\right) \tag{A.9}$$

则有：

$$\frac{\partial}{\partial x_j} \ln p_i = \frac{\partial x_i}{\partial x_j} - \frac{\partial}{\partial x_j} \ln\left(\sum_{l=1}^n e^{x_l}\right) = 1\{i=j\} - \frac{e^{x_j}}{\sum_{l=1}^n e^{x_l}} = 1\{i=j\} - p_j \tag{A.10}$$

其中，$1\{i=j\}$ 为示性函数，$i=j$ 时取 1，$i\neq j$ 时取 0。最后我们能得到：

$$\frac{\partial p_i}{\partial x_j} = p_i \cdot \frac{\partial}{\partial x_j} \ln(p_i) = p_i \cdot (1\{i=j\} - p_j) \tag{A.11}$$

观察（A.11）式，我们可以发现，当输出序列 $(p_1, p_2, \cdots, p_n)$ 中某一个值取 1，其余值全取 0 时，softmax 函数的雅可比矩阵将会成为零矩阵，这也对应了上一节中，输入序列过大的情况。

A.3 softmax 层在反向传播中的影响

我们假设已经得到反向传播时，进入 softmax 层前的梯度流，记为 $(\frac{\partial L}{\partial p_1}, \frac{\partial L}{\partial p_2}, \cdots, \frac{\partial L}{\partial p_n})$，其中 L 为损失函数。根据反向传播以及链式法则，得到梯度流出 softmax 层时的表达式：

$$\frac{\partial L}{\partial x_j} = \sum_{i=1}^{n} \frac{\partial L}{\partial p_i} \cdot \frac{\partial p_i}{\partial x_j} = \left(\frac{\partial p_1}{\partial x_j}, \quad \frac{\partial p_2}{\partial x_j}, \cdots, \frac{\partial p_n}{\partial x_j}\right) \cdot \begin{pmatrix} \frac{\partial L}{\partial P_1} \\ \frac{\partial L}{\partial P_2} \\ \vdots \\ \frac{\partial L}{\partial P_n} \end{pmatrix}, \quad \forall j = 1, \ldots, n \tag{A.12}$$

所以有：

$$\frac{\partial L}{\partial \boldsymbol{x}} = [J_{softmax}]^{\mathrm{T}} \cdot \frac{\partial L}{\partial \boldsymbol{p}} = J_{softmax} \cdot \frac{\partial L}{\partial \boldsymbol{p}} \tag{A.13}$$

在上一节中，我们知道当输入序列过大时，雅可比矩阵会收敛到零矩阵，根据（A.13），这样会使得梯度流被 softmax 层“杀死”，导致后续的反向传播变缓甚至是终止学习。在 Transformer 的训练中，原始点积注意力往往会使得输入值太大，因此我们需要利用缩放点积注意力来控制输入 softmax 层的值不会过大。

至此，我们解释了为何缩放点积注意力函数使模型具有更稳定的梯度流。

附录 B 三角函数位置编码包含相对位置信息的数学解释

回顾第二章中介绍的位置编码，我们将其预先设定为：

$$p_{i,2j} = \sin\left(\frac{i}{10000^{2j/d}}\right) \tag{B.1}$$

$$p_{i,2j+1} = \cos\left(\frac{i}{10000^{2j/d}}\right) \tag{B.2}$$

其中 $p_{i,j}$ 为位置嵌入矩阵 **P** 的第 i 行，第 j 列的值。我们提到过，这样的编码的好处在于既能包含序列的绝对位置信息也能包含序列的相对位置信息。本章讲重点讨论为何这样编码能包含相对位置信息。

我们将相对位置信息定义为：当序列产生偏差时，我们也能容易的用原始序列去线性表示出来。用数学语言表达即为：

假设 $E \in \mathbb{R}^{n\times d_{model}}$ 表示包含 d_{model} 维列向量 $E_{t,:}$ 的矩阵，其中，$E_{t,:}$ 表示长度为 n 的输入序列中的 t 位置的编码，定义如下：

$$E_{t,:} = \begin{bmatrix} \sin\left(\frac{t}{f_1}\right) \\ \cos\left(\frac{t}{f_1}\right) \\ \sin\left(\frac{t}{f_2}\right) \\ \cos\left(\frac{t}{f_2}\right) \\ \vdots \\ \sin\left(\frac{t}{f_{d_{\text{model}}/2}}\right) \\ \cos\left(\frac{t}{f_{d_{\text{model}}/2}}\right) \end{bmatrix} \tag{B.3}$$

其中，频率由下式计算得到：

$$f_m = \frac{1}{\lambda_m} = 10000^{\frac{2m}{d_{\text{model}}}} \tag{B.4}$$

事实上，我们只需要证明：存在一个线性变换 $T^{(k)} \in \mathbb{R}^{d_{\text{model}}\times d_{\text{model}}}$ 使得对于序列中的任意有效位置 $t \in \{1,\ldots,n-k\}$ 的任意的位置偏移量 $k \in \{1,\ldots,n\}$，都有：

$$T^{(k)}E_{t,:} = E_{t+k,:} \tag{B.5}$$

我们定义：

$$T^{(k)} = \begin{bmatrix} \mathbf{\Phi}_1^{(k)} & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{0} & \mathbf{\Phi}_2^{(k)} & \cdots & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \ddots & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \cdots & \mathbf{\Phi}_{d_{\text{model}}/2}^{(k)} \end{bmatrix} \tag{B.6}$$

其中 $\mathbf{0}$ 为 2×2 的零矩阵，$\mathbf{\Phi}^{(k)}$ 定义如下：

$$\mathbf{\Phi}_m^{(k)} = \begin{bmatrix} \cos(\lambda_m k) & -\sin(\lambda_m k) \\ \sin(\lambda_m k) & \cos(\lambda_m k) \end{bmatrix}^\top \tag{B.7}$$

因此，我们只需要证明：

$$\underbrace{\begin{bmatrix} \cos(\lambda_m k) & \sin(\lambda_m k) \\ -\sin(\lambda_m k) & \cos(\lambda_m k) \end{bmatrix}}_{\mathbf{\Phi}_m^{(k)}} \begin{bmatrix} \sin(\lambda_m t) \\ \cos(\lambda_m t) \end{bmatrix} = \begin{bmatrix} \sin(\lambda_m (t+k)) \\ \cos(\lambda_m (t+k)) \end{bmatrix} \tag{B.8}$$

展开（忽略 m）可得：

$$\begin{aligned} \sin(\lambda k + \lambda t) &= \sin(\lambda k)\cos(\lambda t) + \cos(\lambda k)\sin(\lambda t) \\ \cos(\lambda k + \lambda t) &= \cos(\lambda k)\cos(\lambda t) - \sin(\lambda k)\sin(\lambda t) \end{aligned} \tag{B.9}$$

这显然是成立的，继而完成证明。

至此，我们通过数学证明解释了模型中采用的位置编码为何能包含相对位置信息。