本科毕业论文（设计）

深度学习中的稀疏优化方法

Sparse Optimization Methods in Deep Learning

学　　院：数学与统计学院

专　　业：信息与计算科学

学生姓名：_____

学　　号：_____

指导教师：_____

北京交通大学

2024 年 5 月

学士论文版权使用授权书

本学士论文作者完全了解北京交通大学有关保留、使用学士论文的规定。特授权北京交通大学可以将学士论文的全部或部分内容编入有关数据库进行检索，提供阅览服务，并采用影印、缩印或扫描等复制手段保存、汇编以供查阅和借阅。

（保密的学位论文在解密后适用本授权说明）

学位论文作者签名：　　指导教师签名：

签字日期：　　年　　月　　日　　签字日期：　　年　　月　　日

学士论文诚信声明

本人声明所呈交的毕业论文（设计），题目 深度神经网络的稀疏优化方法 是本人在指导教师的指导下，独立进行研究工作所取得的成果。尽我所知，除了文中特别加以标注和致谢中所罗列的内容以外，论文中不包含其他人已经发表或撰写过的研究成果，也不包含为获得北京交通大学或其他教育机构的学位或证书而使用过的材料。

申请学位论文与资料若有不实之处，本人承担一切相关责任。

本人签名：_____ 日期：_____

中文摘要

摘要：随着近年来深度学习的迅速发展，模型规模大幅度增加，千亿参数大语言模型层出不穷。随之出现的还有模型训练、部署和推理的难度、以及过拟合的问题。对模型进行稀疏优化能够大幅度降低难度的同时，还能够在一定程度上降低过拟合。因此，针对上述问题，我们提出了多层神经网络中的全局稀疏优化算法，即将整个网络看成一体，然后对其稀疏优化。然而，我们在实验中发现全局稀疏优化算法存在两个不足。一方面，对大规模模型中的所有参数进行排序，耗时太长；另一方面，全局策略倾向于将模型最后一层参数全部置为零，导致稀疏优化效果不理想。因此针对上述缺陷，我们又提出了层级稀疏优化算法，也即网络层与层之间的稀疏相互独立。为验证两类稀疏策略的有效性，我们在两个较小规模的模拟数据上进行了分类与回归实验，以及在一个大规模实际网络漏洞描述文本数据上进行了分类实验。实验结果表明，两个算法能够在模型泛化效果不发生显著下降甚至有所提高的前提下，大幅度降低了模型中非零参数的数量，从而间接降低模型的训练、部署、推理成本。具体来说，对于两个模拟数据，全局稀疏优化算法能够将模型中的非零参数比例分别降低至 50%和 40%；对于大规模实际文本数据，它能够将模型中的非零参数比例降低至 20%。相较而言，对于两个模拟数据，层级稀疏优化算法能够将模型中的非零参数比例分别降低至 40%和 20%；对于大规模实际文本数据，他能够将模型中的非零参数降低至 2%。

关键词：深度学习；神经网络；稀疏优化

ABSTRACT

ABSTRACT: With the rapid development of deep learning in recent years, there has been a significant increase in model scale, leading to the emergence of language models with billions of parameters. Alongside this trend, challenges in model training, deployment, and inference have surfaced, as well as issues related to overfitting. Sparse optimization of models can greatly alleviate these difficulties while also mitigating overfitting to some extent. Addressing these concerns, we propose a global sparse optimization algorithm for multi-layer neural networks, treating the entire network as a whole and then optimizing its sparsity. However, we identified two shortcomings of the global sparse optimization algorithm through experimentation. Firstly, the time-consuming process of sorting all parameters in large-scale models, and secondly, the tendency of the global strategy to zero out all parameters in the final layer, resulting in suboptimal sparsity. To address these deficiencies, we introduce a hierarchical sparse optimization algorithm, where sparsity is enforced independently between network layers. To validate the effectiveness of these two sparse strategies, we conducted classification and regression experiments on two smaller simulated datasets and a classification experiment on a large-scale real-world dataset consisting of network vulnerability description texts. Experimental results demonstrate that both algorithms can significantly reduce the number of non-zero parameters in models while maintaining or even improving model generalization performance, thereby indirectly reducing the costs associated with model training, deployment, and inference. Specifically, for the two simulated datasets, the global sparse optimization algorithm reduces the proportion of non-zero parameters in models to 50% and 40%, respectively, while for the large-scale real-world text dataset, it reduces the proportion to 20%. In comparison, the hierarchical sparse optimization algorithm achieves reductions to 40% and 20% for the two simulated datasets, and to 2% for the large-scale real-world text dataset.

KEYWORDS： Deep learning; Neural network; Sparse optimization

目　　录

1 引言

1.1 论文选题背景及意义

深度神经网络在许许多多的领域内（例如：图像分类、机器翻译、语音合成）展现出了最先进的水平。虽然一方面模型的质量在随着模型和训练数据的增大而提升，但是随之而来的训练和部署方面的成本也水涨船高。举个例子来说，目前图像分类和机器翻译类的神经网络的参数量是以千万计的，并且通常预测一个结果就需要百亿量级的浮点数运算。这就导致这些领域内的先进模型的部署、预测成本到达一个不可接受的程度，尤其是移动和嵌入式设备上的模型部署、预测。一些近年来大热的巨型模型像目标检测领域的先进模型 Inception-V3[1]，一次预测需要进行 57 亿次运算，拥有 2,700,000 个参数；自然语言处理领域的 GPT-3 模型需要 175,000,000 个参数（假设其参数为 16 bits 精度的话大约需要 350GiB 的存储空间）。这些巨型模型虽然在执行对应任务上的效果远超传统的一些基于规则的模型，但是如果想要在小型设备上部署它们则基本是不可能的。并且即便是使用服务器集群去部署它们，设备成本和电力成本都是不可估量的。除了部署与预测，训练这些大模型成本更高，据斯坦福大学发表的《2024 年人工智能指数报告》[2]，OpenAI CEO 山姆奥特曼提出，训练 GPT-4 的支出已经超过了一亿美元的天文数字。除此之外，谷歌的 Gemini Ultra 的训练成本估计为约两亿美金。训练成本如此之高的原因是训练成本与模型在训练中的计算量是成正比的。引用自《2024 年人工智能指数报告》的图 1-1 展示了模型计算量与训练成本之间的关系。

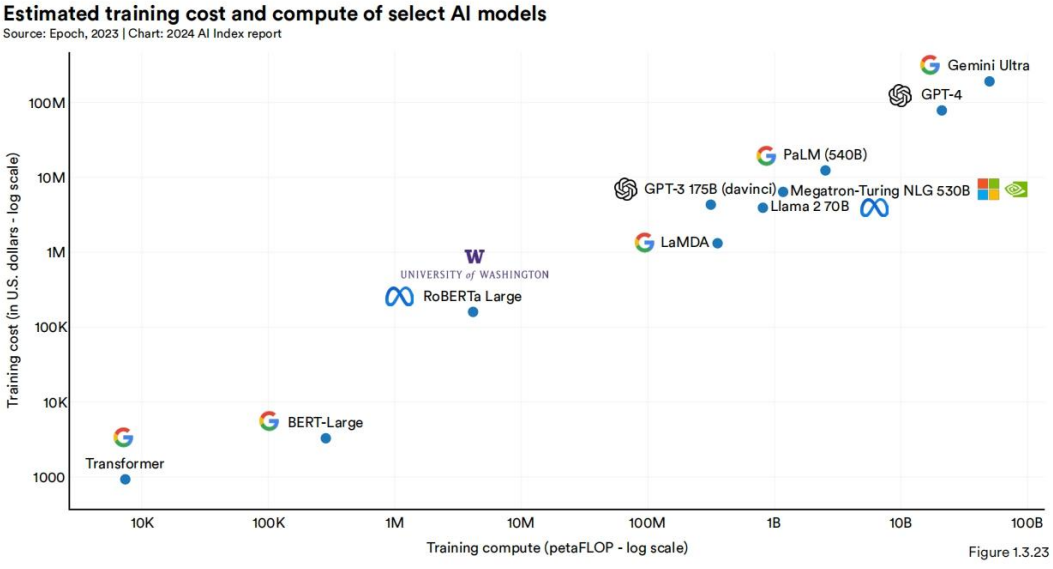

图 1-1 大语言模型训练计算量与模型训练成本

为了解决上述问题，稀疏优化成为了一个可行手段。通过稀疏优化，我们可以使得参数的一个子集置为 0，而任何与权重 0 相乘的运算将被跳过，这样就能够大幅度减少运算量。并且由于参数是稀疏的，可以采取稀疏存储的方式节省内存空间，极大程度上降低了模型的部署和预测成本[3]。比如 Minerva 模型[4]通过对激活值较小的参数的舍弃，大约节约了训练过程中 50%的能源消耗；Eyeriss 模型[5]通过将激活值置为 0 节约计算成本;Cnvlutin[6]也是通过类似的方法达到了与优化之前的模型 13 倍的速度差距和降低 10 倍的能源消耗。由此可见，对模型参数的稀疏优化的确能够大幅度降低模型训练、推理、部署的成本。

虽然对模型的稀疏优化能够极大程度上降低成本，但是有许多人认为这种大幅度的减少模型参数的方法势必也会伴随模型性能的大幅度下降。但是事实证明并非如此。有研究已经证明深度神经网络对参数的高稀疏程度具有相当强的容忍度[7]。甚至有研究发现深度神经网络中的 95%的参数能够通过剩余的 5%的参数预测出来[8]。这些都说明了一定程度的稀疏优化不会对模型的性能造成明显的影响。事实上模型性能与稀疏程度的关系通常如下图所示，一定的稀疏程度非但不会使模型的性能下降，反而会让模型的性能得到提升。一些研究和奥卡姆剃刀原则都能够解释这种现象：提高模型的稀疏程度能够降低模型中存在的噪声。Torsten Hoefler 在他的研究[9]中也提出了类似的结论，其作者画出来模型的表现随着模型稀疏程度的变化曲线示意图如图 1-2 所示，根据作者的结论，模型的表现应该随着稀疏程度的增加呈现先提高后下降的趋势。这个现象非常直观，因为模型的参数量有一定程度的冗余，那么对应会存在过拟合的现象，所以提高模型的稀疏程度会先提高模型的泛化效果，但是当稀疏程度过高时，模型已经不具备拟合的能力了，此时模型的表现就会发生明显下降。

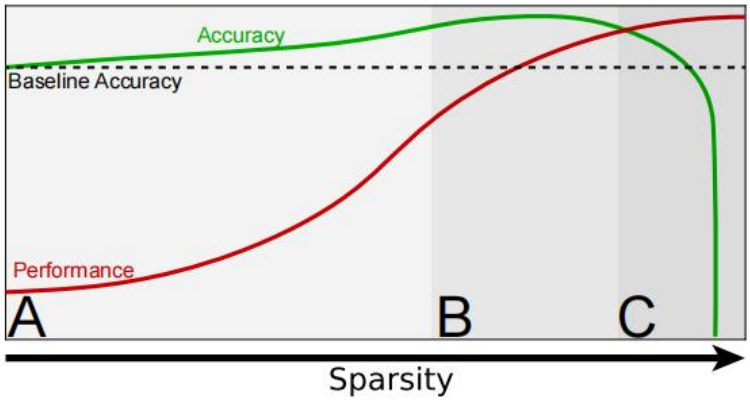

图 1-2 模型表现随模型稀疏度变化示意图

在过去的几年里，学术界提出了很多稀疏优化的方案，比如变分丢弃法[10]、l_0正则法、结构剪枝。然而这些方法并非对模型参数的直接稀疏优化，而是通过训练过程中产生的梯度或激活值为指标，丢弃掉神经网络中的一些结构，从而达到稀疏优化的目的。

或者是采取在损失函数中增加正则项的方式，通过训练隐式地让模型规模减小，间接地达到稀疏优化的目的。这些方案往往以直觉为导向，缺乏足够的数学理论支持。而本论文计划将深度神经网络抽象为最优化问题，从理论出发，提出了神经网络中的全局稀疏优化算法和层级稀疏优化算法，并通过三个实验验证了我们所提出的算法能够大幅度稀疏化模型参数，一定程度上提高模型的泛化效果，并且我们所提出的稀疏优化算法是一个通用的算法，能够用在不同任务形式和不同大小的模型的训练过程中。同时值得注意的是，我们的第三个实验建立了以 Transformer 为基本架构的分类模型，与现在基本所有的大语言模型的基础架构相同，并且我们的算法在第三个实验上展示出了相当可观的稀疏优化效果，预示了我们的算法对大语言模型进行稀疏优化从而降低大语言模型训练、部署、推理成本的极大应用潜力。

万物之始，大道至简，衍化至繁。——老子《道德经》

1.2 计算机领域经典稀疏优化方法

目前的稀疏优化方法种类繁多，最主要可以分为结构化的稀疏方法和非结构化的稀疏方法。结构化的稀疏方法比如直接去掉神经网络中的神经元、卷积神经网络中的卷积核[11]、Transformer 类模型[12]中的注意力头或者更复杂一些的神经网络中的某一个模块这类具有某种固定模式的参数。结构化的稀疏方法虽然能够减少记录需要稀疏的参数的位置信息的存储开销，但是结构化的稀疏方法往往会导致模型效果的明显下降，这是因为结构化的稀疏方法会大大降低模型稀疏时的自由度。因此，结构化的稀疏往往是一种粗粒度且低开销的稀疏优化方法。相比之下，非结构化的稀疏方法，通常是根据一些衡量参数重要程度的指标来选择重要程度高的参数保留，其他参数置为 0 来达到稀疏优化的目的，在实际运用中发现，非结构化的稀疏方法是一种更细粒度的稀疏优化方法，因此往往能够同时带来高稀疏度和较好的模型效果。而我们所设计的全局稀疏优化算法和层级稀疏优化算法也属于非结构化稀疏方法的一种，因此，我会先在接下来的模块简单介绍一下目前的非结构化稀疏优化方法。

1.2.1 训练无关类

最简单也是最直觉性的一种非结构化稀疏优化是移除绝对值最小的参数，这种方法可以追溯到 1993 年，Hagiwara 使用这种方法将 75%的参数置为 0，同时提高了 10%的泛化效果[13]。还有 Gale 将卷积神经网络中的卷积核稀疏[14]，以及 Guiying 将这种方法推

广到 RNN 中，在保证模型的泛化性能不明显下降的前提下，去掉了 96.84%的连接[15]。虽然对于该方法的研究已经很多，但是大多数研究所讨论的模型参数相对于目前的千亿参数级别的大语言模型来说，规模还比较小。因此，在之前的研究中将全局参数进行排序，选取绝对值最大的前 s 个参数所花费的时间是可以接受的，但是当参数规模到达千亿级别时，排序所花费的时间已经不可忽略了，这也是我们提出层级稀疏优化算法的主要原因之一。我们注意到在模型参数增加的同时，模型深度也在增加，因此我们提出了模型层之间相互独立的假设，对模型的每一层单独进行稀疏，这样就能够并行稀疏模型的每一层，大幅度缩短稀疏所消耗的时间。

1.2.2 数据驱动类

这一类稀疏优化方法主要考虑模型神经元或整个模型的输出的数值敏感度。在这一类方法中，通常存在一个训练数据的集合来确定哪些参数需要被移除。举例来说，输入该集合中的不同例子，激活值变化最小或者几乎不变的参数将被移除，因为它们对模型判别不同的输入基本没有贡献。因此，这个敏感度指标可以用来衡量参数的重要性，而重要性较低的参数将被移除。在移除后，这些移除参数的激活值会被加入到下一层神经元的偏置中，以维持整个神经网络的不变性[16]。Castellano[17]泛化了这一方法，并在最后移除参数之后，将根据与移除之前的激活值变化最小为依据，重新计算参数。上述方法同样也可以运用在卷积神经网络中，比如 Luo[18]以卷积层输出的激活值为依据，删除在整个小批量训练数据中对激活值改变最小的卷积层参数。

1.2.3 训练感知类

除了上述两种稀疏优化方法以外，我们还可以使用一些训练感知的方法。其中一种最简单的方法就是考虑训练过程中的总参数变化量。我们可以存储所有参数在整个训练中的更新量的大小，然后移除更新量最少的参数[19]。直觉上的解释是如果参数自随机初始化开始到训练结束，其改变量很小的话，那么这些参数可以被认为是“不重要的”。

这些训练感知类方法中比较著名的一种是在损失函数中加入正则项，举例来说 $L^{'}(x,w)=L(x)+P(w)$，其中 L(x)是原始损失函数，而$P(w)$是关于模型参数的惩罚项。惩罚项的设计可以是关于任何神经元本身的[20]，也可以是关于一些指标的，比如所需浮点数运算[21]，添加惩罚项能够间接让模型在训练过程中变得稀疏，同时降低了复杂度。

1.3 本文的主要工作和章节安排

本文针对目前深度学习中大规模模型的部署、推理成本高和过拟合问题，提出了全局稀疏优化算法和层级稀疏优化算法，并设计了两个模拟实验和一个实际网络漏洞描述文本分类实验证明了我们的两个算法能够在不大幅度损失甚至提高泛化效果的前提下大幅度减少模型中非零参数占比，同时我们还提供了许多可视化结果来直观地展示我们的算法的工作流程。

我们首先在第二部分提出了稀疏优化算法的一些理论支撑和具体算法设计，在第三部分展示了分别展示了三个实验的实验结果和对应的讨论。

2 理论依据

最优性条件在最优化问题的研究中有着重要的地位，从实际运用的角度来说，最优性条件是许多数值解法的基础，所以我们希望从解的最优性条件出发去支撑我们的算法。然而神经网络中的最优化问题通常是非凸的，所以不存在充分必要条件来确保找到最优解，因此我们参考了 Amir Beck 所提出的理论[22]，将其推广到了神经网络中，在下面的板块列举出最优解的一些必要条件，并且分析它们之间的关系，并在 2.2 中详细介绍这些必要条件如何引导我们设计算法，找到满足这些条件的解的。

2.1 稀疏优化的必要最优性条件

2.1.1 记号与假设

我们先假设数据集 $D = \{(x_n, y_n)\}_{n=1}^{N}, \overrightarrow{x_n} \in \mathbb{R}^m, \overrightarrow{y_n} \in \mathbb{R}$。记数据集的标签$\vec{y} = \{y_n\}_{n=1}^{N}$，模型的预测值为$\vec{\hat{y}} = (f(\overrightarrow{x_n}, w))_{n=1}^{N}$。

对于单层神经网络，我们要优化的问题可以写做：

$$P_0: \quad \min h(w) := ||\vec{y} - \vec{\hat{y}}||^2$$

$$s.t. \ ||w||_0 \leq s$$

这里的$w \in \mathbb{R}^k$和 $s \in (0,1)$并且 $s \ll k$，$||w||_0$是该单层神经网络所有参数的l_0范数，即参数矩阵 w 中的非零参数的个数。对于多层神经网络来说，如果令 $w = \{w_1, w_2, \ldots, w_I\}$，I 为神经网络的层数，则原优化问题可写做：

$$P_0': \quad \min h(w) := ||\vec{y} - \vec{\hat{y}}||^2$$

$$\text{s.t. } \sum_{i=1}^{I} ||w_i||_0 \leq s, i = 1,2,3,\ldots,I$$

假设 2.1. 神经网络除第一层和最后一层以外，每一层参数之间相互独立。

由假设，我们可以推得原问题在第二层到$I-1$层上每一层是一个有稀疏约束条件的优化问题，不失其一般性，我们假定为第 i 层：

$$P'_{0,i}: \quad \min h(w) := ||\vec{y} - \vec{\hat{y}}||^2$$

$$\text{s.t. } ||w_i||_0 \leq s_i$$

在第一层以及最后一层上没有稀疏约束条件：

$$P'_{0,I}: \quad \min h(w) := ||\vec{y} - \vec{\hat{y}}||^2$$

可以看到实际上$P'_{0,i}$与P_0相同，为记号简便，后续讨论中直接讨论P_0。

我们记C_s为最多 s-稀疏的参数矩阵集合：

$$C_s: \{w: ||w||_0 \leq s\}。$$

我们记$I_1(w)$为参数矩阵 w 中非零参数的位置集合：

$$I_1(w) \equiv \{(i,j): w_{i,j} \neq 0\}$$

对应零参数的位置集合：

$$I_0(w) \equiv \{(i,j): w_{i,j} = 0\}$$

记$M_i(w)$是指 $w \in \mathbb{R}^k$中绝对值最大的第 i 个元素：

$$M_1(w) \geq M_2(w) \geq \ldots \geq M_k(w)$$

2.1.2 基本可行解

定义 2.1. $w^* \in C_s$是P的一个基本可行解，如果有：

1. 当$||w^*||_0 < s$时，$\nabla f(w^*) < 0$;
2. 当$\left|\left|w^*\right|\right|_0 = s$时，$\nabla_{i,j} f(w^*) = 0, \forall (i,j) \in I_1(w)$

定理 2.1. 如果令w^*是P的最优解，那么w^*是一个基本可行解。

2.1.3 L-稳定性

对于问题

$$C: \quad \min\{g(w): w \in C\}$$

这里 C 是闭凸可行域，g 是一个连续可导的函数，可能是非凸的。那么该问题的一个解$w^* \in C$可以被称为是稳定点，当：

$$\langle \nabla g(w^*), w - w^* \rangle \geq 0, \forall w \in C$$

如果w^*是该问题的最优解，那么显然w^*也是稳定点，因此，稳定点是一个必要最优性条件。许多解决上述非凸问题的优化方法都只能保证收敛到稳定点。

实际上述性质可以转化为$\forall L > 0$，解w^*是一个稳定点，当且仅当

$$w^* = P_C(w^* - \frac{1}{L}\nabla g(w^*))$$

这里假设 $D \subseteq \mathbb{R}^n$，$P_D(\cdot)$记作 D 上的正交投影，即

$$P_D(y) \equiv \underset{w \in D}{\operatorname{argmin}} ||w - y||^2$$

上式推广至有稀疏约束的问题 P 中，记为 L-稳定性。

定义 2.2. 一个解$w^* \in C_s$是问题 P 的一个 L-稳定点，如果w^*满足关系

$$[NC_L] \quad w^* \in P_{C_s}(w^* - \frac{1}{L}\nabla f(w^*))$$

引理 2.1. $\forall L > 0$，w^*满足$[NC_L]$条件当且仅当$||w||_0 \leq s$ 和

$$|\nabla_{i,j} f(w^*)| \begin{cases} \leq L M_s(w) & \text{if } (i,j) \in I_0(w^*), \\ = 0 & \text{if } (i,j) \in I_1(w^*) \end{cases}$$

假设 2.2. 目标函数的梯度$\nabla f(w)$是李普希茨连续，设其李普希茨常数 L(f)，即

$$||\nabla f(w_1) - \nabla f(w_2)|| \leq L(f)||w_1 - w_2||, \quad \forall w_1, w_2$$

引理 2.2. (Decent Lemma) 假设 f 连续可导且满足假设 3.1，$\forall L > L(f)$

$$f(w) \leq h_L(w_1, w_2) \quad \forall w_1, w_2$$

$$h_L(w_1, w_2) = f(w_2) + \langle \nabla f(w_2), w_1 - w_2 \rangle + \frac{2}{L}||w_1 - w_2||^2$$

引理 2.3. 如果假设 2.2 成立，$L > L(f)$。对于任意 $x \in C_s$，y 是一个参数矩阵，满足

$$y \in P_{C_s}(x - \frac{1}{L}\nabla f(x)),$$

我们有

$$f(x) - f(y) \geq \frac{L - L(f)}{2}||x - y||^2$$

定理 2.2. 假设 2.2 成立的前提下，$L > L(f)$，假设w^*是问题 P 的最优解，那么

(1) w^*是一个 L-稳定点

(2) $P_{C_s}(w^* - \nabla f(w^*)/L)$只包含一个元素

2.1.4 理论总结

通过上述定理我们发现这几个最优解的必要条件之间的关系如图 2-1 所示。

图 2-1 最优解必要条件关系示意图

2.2 算法设计

现在我们根据上述提供的理论结果，设计两个稀疏优化算法，分别是全局稀疏优化算法和层级稀疏优化算法。

- 全局稀疏优化算法主要使用*L*-稳定性必要最优性条件，该算法与 Amir Beck 在其论文[22]中所提出的 IHT 硬阈值算法比较类似，但是我们将该算法推广至了多层神经网络之中，我们迫使算法得到的解满足*L*-稳定性条件，并且证明了该算法得到的解是收敛的。
- 层级稀疏优化算法主要源自假设 2.1 神经网络参数层与层之间相互独立。我们提出将原问题看作具有 I 个约束条件的最优化问题，并分解为 I 个子问题，分别进行求解。

2.2.1 全局稀疏优化算法

现对于原问题P_0

$$P_0: \quad \min h(w): = ||\vec{y} - \vec{\hat{y}}||^2$$

$$s.t. ||w||_0 < s$$

我们可以采取下面的解迭代方法来“迫使”解满足 L-稳定性条件：

$$w^{k+1} \in P_{C_s}(w^k - \frac{1}{L}\nabla f(w^k)), k = 0,1,2\ldots$$

上述迭代方式相当于在每次梯度更新后选取参数矩阵中绝对值最大的 s 个参数保留，其余的参数置为 0 即可。

引理 2.4. 令$\{w^k\}_{k\geq 0}$为全局稀疏优化算法产生的解的序列，取固定更新步长为 $1/L$，$L > L(f)$即可。则有结果

1. $f(w^k) - f(w^{k+1}) \geq \frac{L-L(f)}{2}||w^k - w^{k+1}||^2$

2. $\{w^k\}_{k\geq 0}$是一个非增的序列。

3. $||w^k - w^{k+1}|| \to 0$

4. 对于每一个 $k = 0,1,2,3\ldots$，如果$w^k \neq w^{k+1}$，那么 $f(w^{k+1}) < f(w^k)$。

上述结果说明了只要目标函数满足李普希茨连续，只要在算法迭代的过程中取合适的步长（足够小）就可以得到收敛的结果。

对于多层神经网络来说，如果从全局参数的角度出发，我们可以将问题$P_0^{'}$：

$$P_0^{'}: \quad \min h(w): = ||\vec{y} - \vec{\hat{y}}||^2$$

$$s.t. \sum_{i=1}^{I} ||w_i||_0 < s, i = 1,2,3\ldots I$$

看作单层神经网络的最优化问题P_0：

$$P_0: \quad \min h(w): = ||\vec{y} - \vec{\hat{y}}||^2$$

$$s.t. ||w||_0 < s$$

因此我们的全局稀疏优化算法的伪代码可以表示如下：

全局稀疏优化算法：

给定保留参数个数（保留参数占总参数量比例）s，初始化参数矩阵w_i^0，$i = 1,2,3\ldots I$，步长$\gamma < 1/L$，终止信号θ

重复

令中间变量$z^{k+1} = w^k - \gamma\nabla h(w^k)$

$\tilde{z}^{k+1}$为将z^{k+1}中绝对值最大的 s 个参数保留，其余参数置为 0 后结果

$w^{k+1} = \tilde{z}^{k+1}$

直到$||w^{k+1} - w^k||^2 < \theta$

2.2.2 层级稀疏优化算法

由于全局稀疏优化方法并未考虑不同层参数之间的重要性差异，导致最终稀疏优化的效果不好，具体可以参照实验部分结果。因此，我们假设第一层与最后一层的重要性远高于中间层。这样假设的原因是直觉上第一层的参数与输入相关，如果第一层参数部分置为 0，那么将有部分输入信息损失，如果将最后一层参数部分参数置为 0，那么在前面层的参数中学习到的知识则无法反映到输出中。

因此我们设计了层级稀疏优化算法，将多层神经网络按层级划分，只对中间层进行稀疏优化，而第一层与最后一层参数不进行稀疏优化。

我们的层级稀疏优化算法伪代码表示如下：

层级稀疏优化算法：

给定保留参数个数（保留参数占总参数量比例）s，初始化参数矩阵w_i^0，$i = 1,2,3\ldots I$，步长$\gamma < 1/L$，终止信号θ

重复

令中间变量$z_i^{k+1} = w_i^k - \gamma\nabla h(w_i^k)$，$i = 1,2,3\ldots I$

分别得到$\tilde{z}_i^{k+1}$为将z_i^{k+1}中绝对值最大的 s 个参数保留，其余参数置为 0 后结果

$w_i^{k+1} = \tilde{z}_i^{k+1}$，$i = 2,3\ldots I-1$

$w_1^{k+1} = z_1^{k+1}$，$w_I^{k+1} = z_I^{k+1}$

直到$||w^{k+1} - w^k||^2 < \theta$

3 实验

在本章中，我们设计了三个实验来证明我们所设计的算法的有效性。第一个实验是拟合 Sinc 函数的回归实验，第二个实验是双螺旋数据的分类实验，第三个实验是基于 Transformer 架构的实际文本分类实验。通过前两个实验我们可以验证算法的通用性。因为目前几乎所有大模型都是以 Transformer 为基础架构，所以第三个实验验证了我们的算法对大模型进行稀疏优化的潜力。

实验环境为 Python3.8，显卡配置为 NVIDIA RTX A6000

3.1 拟合 Sinc 函数的回归实验

我们考虑使用神经网络从有噪声的数据中拟合 Sinc 函数，其中噪声指的是均值为 0，方差$\sigma^2 = 0.005$ 的高斯噪声。

$$g(x) = \frac{\sin(x)}{x}$$

假设训练数据集样本数量 $N = 300$，我们将在该数据集上训练一个隐藏层为 2 的神经网络即

$$f(x) = w_3\rho(w_2\rho(w_1x + b_1) + b_2) + b_3$$

神经网络示意图如图 3-1 所示。

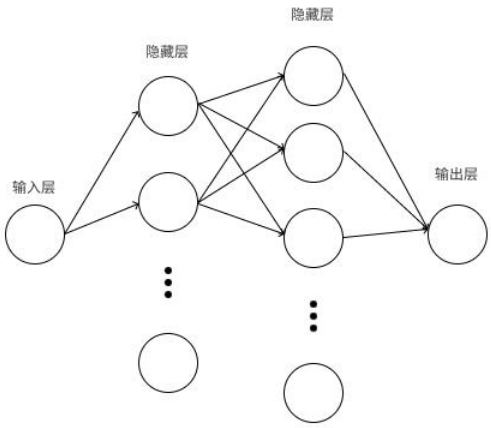

图 3-1 神经网络示意图

超参数设置为ρ,为保留的参数占模型总参数量的比例，

$$\rho = \frac{s}{k} \in \{0.05,0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1\},$$

这里当$\rho = 1$ 时相当于没有对模型进行稀疏优化。优化器为 Adam 优化器，学习率设置为 0.001，算法终止条件为更新前后模型所有参数之间的距离小于$\theta = 0.001$。

3.1.1 稀疏优化中的参数更新可视化

为了了解在不同稀疏度下，神经网络中参数的实际更新情况和层内与层间参数重要性的差异。我们先在一个小型的神经网络上使用了全局稀疏优化算法，画出了参数保留频次热力图和算法停止时模型最终的参数情况，如图 3-2 所示。图中从左至右分别对应神经网络中第一层、第二层和第三层的参数保留频次热力图，其中每一幅图中对应位置颜色深度代表了该位置的参数保留频次多少，颜色越深代表保留频次越高。

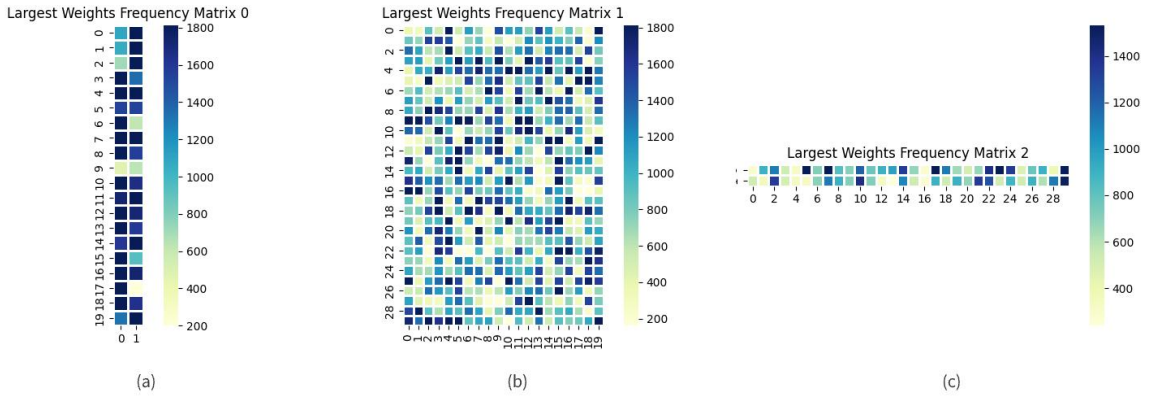

图 3-2 全局稀疏优化算法中神经网络权重更新频次热力图

这里以ρ = 40%，即保留 40%的参数时举例。假设以参数的绝对值大小决定参数重要性。首先在同一层中，可以看到参数矩阵第一层保留频次都在 16000 次左右，不同位置的参数保留频次相差很小。而第二层和第三层中，同一层内一部分参数保留频次在 16000 次左右，而大量参数保留频次在 2000 次左右，相差约 8 倍。所以在同一层中不同位置的重要性在第一层中相差不大而在第二层和第三层中相差很大。其次不同层之间，第一层平均保留频次远高于第二层和第三层，所以可以推测第一层的重要性远高于第二层和第三层。

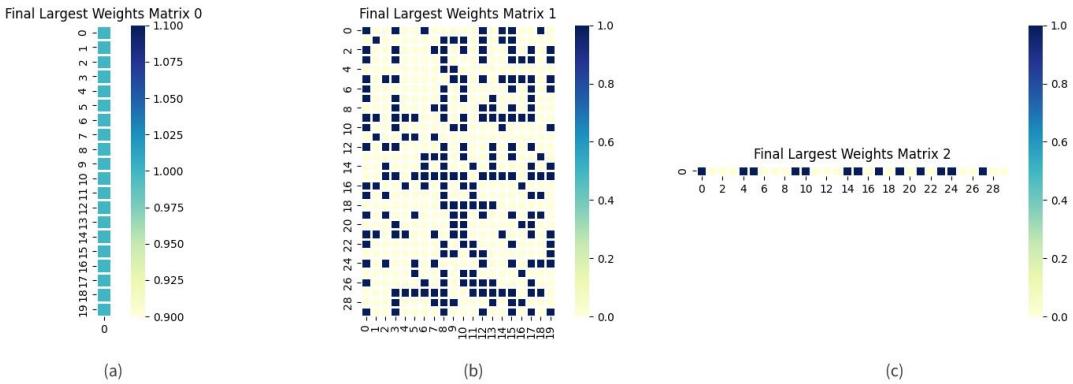

图 3-3 全局稀疏优化后最终神经网络保留参数示意图

图 3-3 是算法停止后，最终模型中的参数情况，其中白色位置的参数为 0，非白色位置的参数非 0。我们可以直观地看出，模型中第一层参数全部被保留，第二层和第三层大量参数被置为 0，这符合我们的预期结果，说明了我们的稀疏优化算法能够将模型中大量参数置为 0，提高模型参数的稀疏度。

这三张图直观反映了我们的算法确实能够将模型内部的参数矩阵变得稀疏，能够达到我们的目的。通过对比三张图我们可以发现最终模型第一层没有值为 0 的参数，这进一步印证了我们前文关于不同位置参数重要性的猜想。并且通过对比图和图，我们发现算法停止后，模型内部非 0 参数的位置和参数保留频次热力图中颜色较深的位置重合度较高，这启示我们模型权重在训练后期进行稀疏优化的位置是相对固定的，在后续的研究中可以重新设计算法，在训练中期挑选出更新频次比较固定的参数位置，在训练后期无需再计算全局参数的梯度，只计算目标位置的梯度，更新目标位置的参数，其他参数直接置为 0 即可。这样就能够大大降低模型训练过程中的计算量。

3.1.2 实验结果

3.1.1 中的实验为了能够可视化参数矩阵因而故意将隐藏层神经元数目降低。在正式实验中，我们同样使用隐藏层数目为 2 的神经网络，其中第一个隐藏层的神经元数目为 200 个，第二个隐藏层的神经元数目为 300 个，则该神经网络的总参数量为 60500（不计算偏置项），激活函数为 Sigmoid 函数。为减少随机性带来的影响，我们进行了多次实验，最终结果取其平均值。

对于该回归任务，我们的评价指标是 RMSE，即$||\vec{y}-\vec{\hat{y}}||^2$。

表 3-1 拟合 Sinc 函数曲线实验结果

参数保留比例 s	全局稀疏优化		层级稀疏优化	
	$RMSE_{train}$	$RMSE_{test}$	$RMSE_{train}$	$RMSE_{test}$
100%	0.0701	0.0931	0.0701	0.0931
90%	0.0724	0.1329	0.0706	0.0971
80%	0.0715	0.1156	0.0704	0.0851
70%	0.0720	0.1111	0.0711	0.0943
60%	0.0713	0.1032	0.0705	0.0887
50%	0.0703	0.1123	0.0694	0.0932
40%	0.0743	0.1626	0.0695	0.0901
30%	0.0842	0.2829	0.0855	0.2756
20%	0.0805	0.2184	0.0875	0.3064
10%	0.0941	0.3885	0.1037	0.6664
5%	0.1766	0.8886	0.1067	0.6575

通过表格数据可知，对于全局稀疏优化算法来说，能够在模型泛化性能不发生显著下降的前提下，将模型参数减少到原来的 50%。拟合曲线对比如图 3-3 所示，其中黄色曲线是 Sinc 函数，蓝色曲线为模型拟合曲线，左侧是未进行稀疏优化的拟合结果，右侧是参数保留比例为 50%时，全局稀疏优化后的拟合结果。我们可以直观地看出模型的拟合能力基本与未进行稀疏优化时相同。

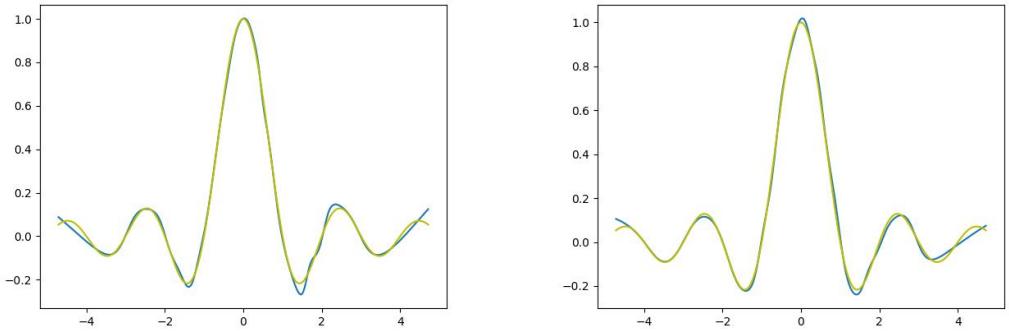

图 3-4 全局稀疏优化中拟合曲线对比图

对于层级稀疏优化算法来说，能够在模型泛化性能不发生显著下降的前提下，将模型参数减少到原来的 40%。并且能够在参数量为原来的 60%时获得优于不进行稀疏优化时的泛化效果。拟合曲线对比如图 3-5 所示，其中左侧拟合曲线图为未进行稀疏优化时的拟合结果，右侧拟合曲线图为参数保留比例为 40%时，层级稀疏优化后模型的拟合结果，对比左右两个拟合曲线图，可以发现层级稀疏优化后的模型拟合效果未发生明显下降，基本与未进行稀疏优化时相同。

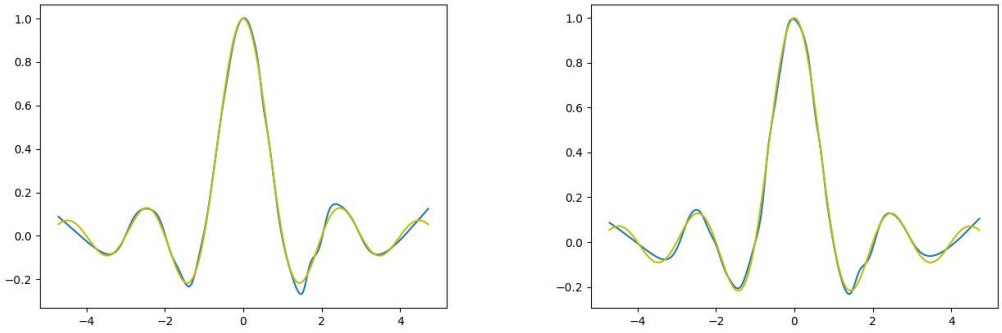

图 3-5 层级稀疏优化中拟合曲线对比图

对比全局稀疏优化算法和层级优化算法，两者在保留同一比例的参数时，层级稀疏优化算法的泛化效果往往优于全局稀疏优化算法，这是因为全局稀疏优化算法倾向于将最后一层的大部分参数置为 0，这会导致在前两层学习到的参数变相丢失，更直观的示意图如图 3-6 所示，假设第二层参数$w_{1,1}^2, w_{2,1}^2 \ldots, w_{n,1}^2$不全为 0，即从训练数据集中学习到了“知识”，但是如果第三层参数$w_{1,1}^2$被置为 0 后，神经网络前向传播到这里激活值

全被置为 0，所以即便前面层中的参数学习到了数据中的“知识”，因为最后一层的参数被置为 0，实际上是一种变相的“丢弃”。

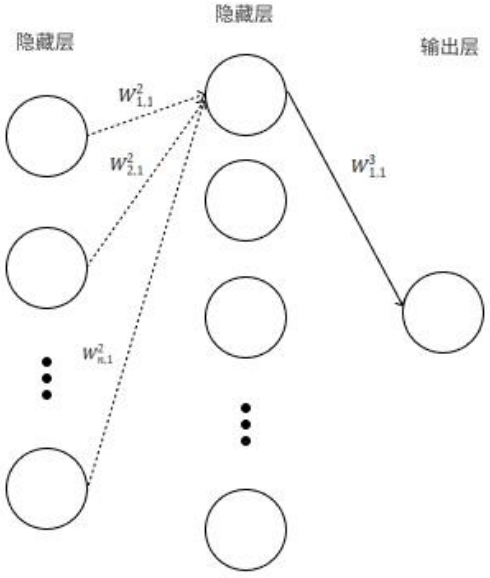

图 3-6 全局稀疏优化算法导致“知识”丢失示意图

3.2 双螺旋数据分类实验

我们考虑使用神经网络将双螺旋数据进行分类，其中两类点各 1000 个样本按 80%、20%的比例分为训练集和验证集。其中数据集的构建方式参考论文[23]，对于单个数据(x^n, y^n, b^n)，详细构建方式参考下列公式：

$$x^n = r_n \sin\alpha_n, \qquad y^n = r_n \cos\alpha_n,\ r_n = \frac{n}{1000}$$

$$\alpha_n = \begin{cases} \pi + \dfrac{2n\pi}{1000} + \sigma, \text{if } b^n = 1 \\ \dfrac{2n\pi}{1000} + \sigma, \text{if } b^n = 0 \end{cases}$$

其中 σ 为服从标准正态分布的高斯噪声，所生成数据点如图 3-7 所示。超参数设置为ρ为保留的参数占模型总参数量的比例，

$$\rho \in \{0.05, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1\},$$

这里当$\rho = 1$时相当于没有对模型进行稀疏优化。优化器为 Adam 优化器，学习率设置为 0.001，算法终止条件为更新前后模型所有参数之间的距离小于$\theta = 0.001$。

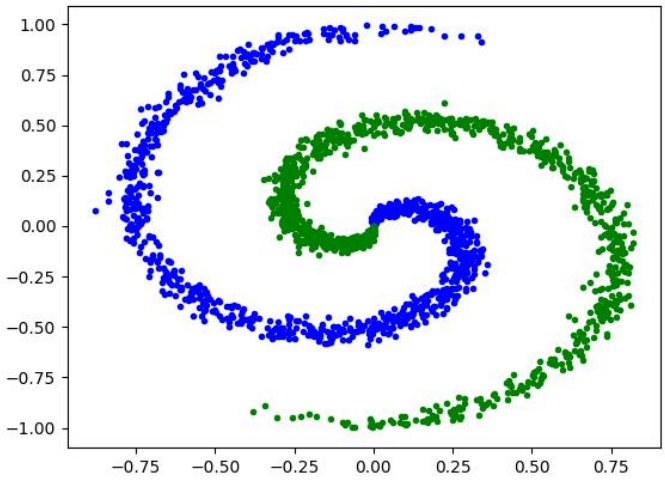

图 3-7 双螺旋数据点示意图

3.2.1 稀疏优化中的参数更新可视化

为了可视化稀疏优化算法中权重保留过程，我们使用了小型的神经网络，并使用全局稀疏优化算法，画出了参数保留频次热力图和算法停止时模型最终的参数情况。其中参数保留频次热力图如图 3-8 所示，图中从左至右分别对应神经网络中第一层、第二层和第三层的参数保留频次热力图，其中每一幅图中对应位置颜色深度代表了该位置的参数保留频次多少，颜色越深代表保留频次越高。可以直观的看出第一层中不同位置参数保留频次的差距较小，而在第二层和第三层中差距较大，其更新频次最多的位置可以达到 1800 次，而最少的位置只有 200 次，相差大概九倍，这再次说明了同一层中不同位置的参数重要性差异较大，重要性较低的位置的参数就可以被置为 0，并且不会影响模型泛化效果。

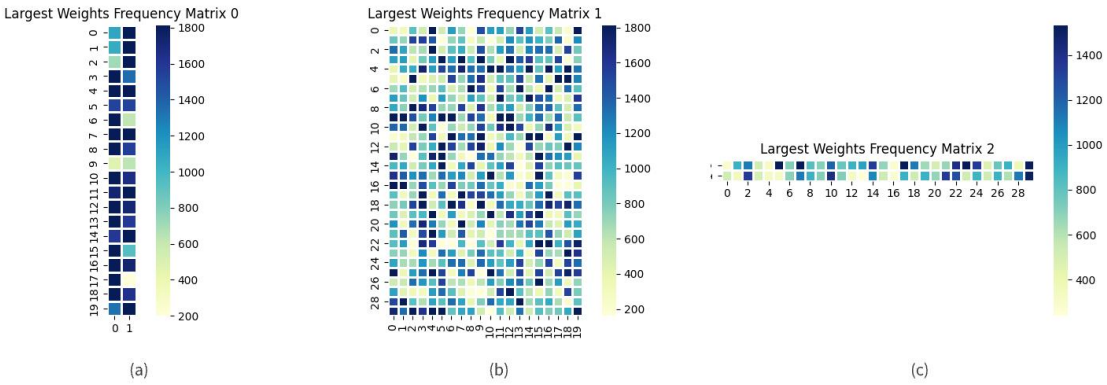

图 3-8 层级稀疏优化算法中神经网络权重更新频次热力图

这里以ρ = 20%时为例，即保留神经网络中 20%的权重参数。权重保留频数图和最终模型内权重情况可视化结果如下图 3-9 所示，结论与 3.1.1 相同。

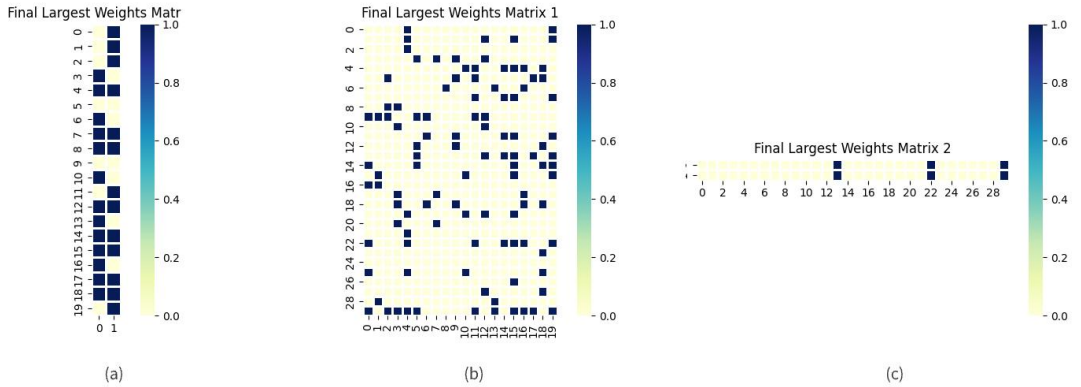

图 3-9 层级稀疏优化后最终神经网络保留参数示意图

3.2.2 实验结果

3.2.1 中的实验为了能够可视化参数矩阵因而故意将隐藏层神经元数目降低。在正式实验中，我们同样使用隐藏层数目为 2 的神经网络，其中第一个隐藏层的神经元数目为 200 个，第二个隐藏层的神经元数目为 300 个，则该神经网络的总参数量为 60500（不计算偏置项），激活函数为 Sigmoid 函数。为减少随机性带来的影响，我们进行了多次实验，最终结果取其平均值。

对于该回归任务，我们的评价指标是分类正确率。

表 3-2 双螺旋数据分类实验结果

参数保留比例 s	全局稀疏优化		层级稀疏优化	
	acc_{train}	acc_{test}	acc_{train}	acc_{test}
100%	99.72%	99.75%	99.72%	99.75%
90%	99.66%	99.50%	99.64%	99.50%
80%	99.75%	99.75%	99.85%	99.92%
70%	99.88%	100.00%	99.79%	99.71%
60%	99.81%	100.00%	99.88%	99.75%
50%	99.84%	100.00%	99.82%	99.58%
40%	99.81%	100.00%	99.49%	99.75%
30%	80.25%	75.63%	99.28%	99.71%
20%	59.12%	51.13%	99.79%	99.75%
10%	50.25%	49.00%	63.89%	60.04%
5%	50.25%	49.00%	63.40%	59.79%

通过表格数据可知，对于全局稀疏优化算法来说，能够在模型泛化性能不发生显著下降的前提下，将模型参数减少到原来的 40%。分类结果如图 3-10 所示，左侧是未进行稀疏优化时模型分类结果，分类边界清晰可见，右侧是保留参数比例为 40%时模型分类结果，可以直观地看出，全局稀疏优化后模型的分类效果基本不下降。

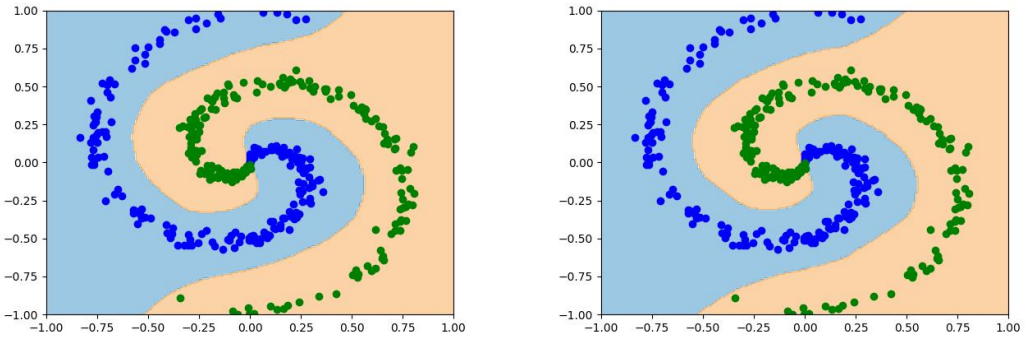

图 3-10 全局稀疏优化中分类结果对比图

对于层级稀疏优化算法来说，能够在模型泛化性能不发生显著下降的前提下，将模型参数减少到原来的 20%。并且能够在参数量为原来的 80%时获得优于不进行稀疏优化时的泛化效果。分类结果如图 3-11 所示，左侧是未进行稀疏优化时模型分类结果，分类边界清晰可见，右侧是保留参数比例为 20%时模型分类结果，可以直观地看出，层级稀疏优化后模型的分类效果基本不下降。

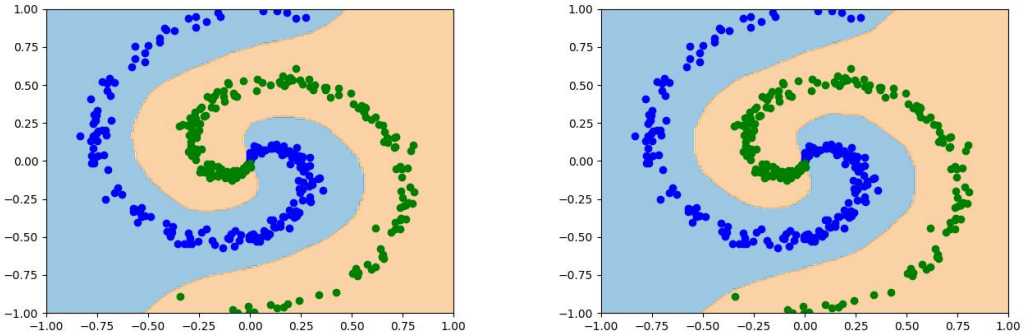

图 3-11 层级稀疏优化中分类结果对比图

3.3 基于 Transformer 架构的实际文本分类实验

为了验证我们的算法除了在一般规模的神经网络和简单的模拟数据上，在更大规模和实际应用任务上也具有良好的稀疏优化效果，我们建立了有约 1,800,000 参数量的神经网络对实际网络服务的描述文本进行风险等级预测。并且我们还希望验证我们的算法对大语言模型也能够进行稀疏优化的潜力。但是在 1.1 中我们已经提到目前大模型的巨大参数规模，由于硬件设施的限制，我们不能够直接将我们的算法应用在大语言模型中。所以我们建立的模型以 Transformer 为基础架构，间接展示我们的算法对大语言模型的稀疏优化的潜力。

本章的安排是，我们首先在 3.3.1 中简单介绍 Transformer 架构，然后在 3.3.2 中介绍所使用的数据集，最后在 3.3.3 中展示实验结果和结论。

3.3.1 Transformer 架构

Transformer 架构最早是由 Vaswani 等在 2017 年提出的[24]。Transformer 架构目前已在自然语言处理领域获得了非常广泛的应用，特别是在大语言模型领域，基本所有的领域内效果领先的大语言模型都采用了 Transformer 架构，比如 Chat-GPT[25]、Chat-GLM[26]、Baichuan[27]和 LLaMA[28]等等。Transformer 架构通常包含编码器和解码器两部分，由于我们的任务形式是文本分类任务，因此只需要用到编码器部分，即图中左侧红色圈出的部分，如图 3-12 所示，原图引用自[24]。

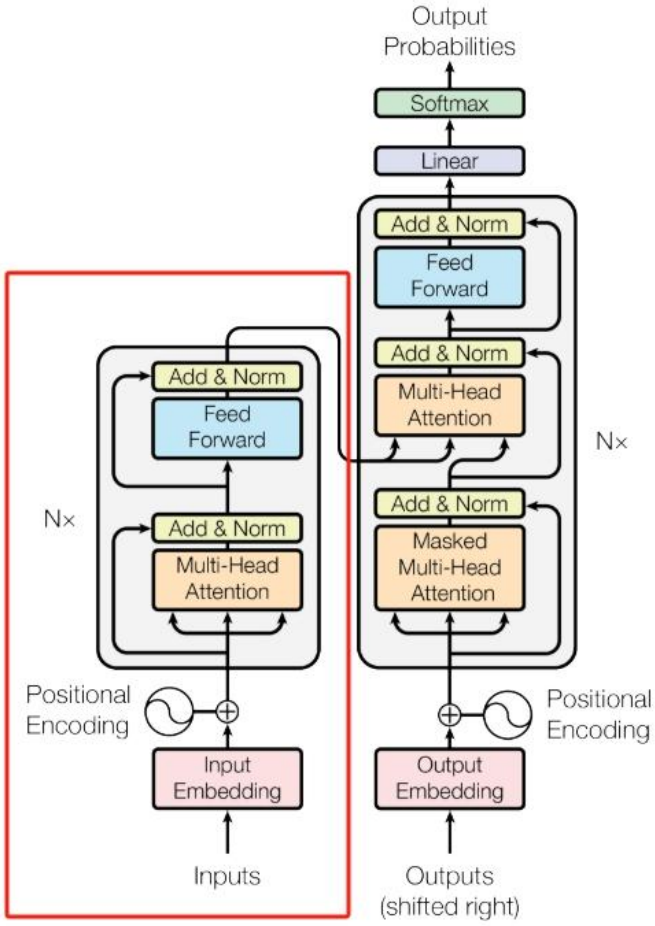

图 3-12 Transformer 结构示意图

首先对于要分类的文本，需要使用文本分词工具，例如 jieba 分词将文本分成 Tokens，而每个 Token 将对应词汇表中的一个独热向量，将独热向量与词嵌入矩阵相乘即可得到该 Token 的词嵌入，进而能够得到整个文本的词嵌入。得到词嵌入以后由于 Transformer

编码器是对整个词嵌入并行编码，因此还需要给文本的词嵌入加上位置编码，这样才能让 Transformer 得到文本的完整语义信息。得到上述处理过后的词嵌入之后，需要使用多头注意力机制对词嵌入进行特征提取，多头注意力机制的处理方式可以由下列公式进行简单概括，其中 Q、K、V 分别是词嵌入与W_q、W_k、W_v三个方阵相乘所得，而W_q、W_k、W_v即 Transformer 中的一部分参数组成。

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

多头注意力机制得出的结果还需要与原输入残差连接和进行归一化以后输入线性层，并重复残差连接和进行归一化最终得到 Transformer 编码器的输出，然后将输出再作为下一个编码器的输入再进行编码，重复上述操作。在堆叠多个编码器后，将最后一个编码器的输出连接一个线性层，即可完成分类任务的训练。所以本章所构建的模型结构图如图 3-13 所示。

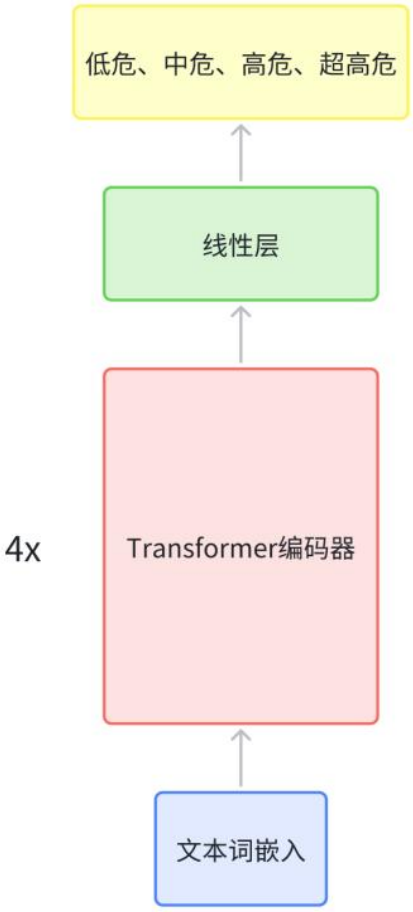

图 3-13 基于 Transformer 的文本分类器结构示意图

我们建立的模型中，序列长度取 256，分类类别为 4，模型维度取 256，嵌入维度取 256，隐藏层维度取 512，编码器的数目取 4，则可以估算出模型的总参数量为 1,800,000。

3.3.2 实验数据集

本实验中使用的数据集是从国家信息安全漏洞共享平台上爬取的，包含 146,514 条关于网络服务漏洞的文本描述，以及相关的风险等级评定。其中几条展示如下表。

表 3-3 实际文本分类实验数据集举例

描述文本	风险等级
IBM InfoSphere Information Server 8.1，8.5 FP3 之前版本以及 8.7 版本和 InfoSphere Business Glossary 8.1.1，8.1.2 版本中的 Information Services Framework (ISF)中存在漏洞，该漏洞源于程序没有将登录页面中密码字段的自动填表功能关闭。通过利用无人看守的工作站，远程攻击者可利用该漏洞获得访问权限。	低危
IBM Informix Dynamic Server (idS) 10.00.xC8 之前的 10.x 版本中的多个未明程序允许本地用户通过列出环境变量中的 target 文件创建任意文件，该变量的所有权被转移到应用这些程序的用户。	中危
Restlet 1.1.10 版本中存在代码漏洞。远程攻击者可利用该漏洞获取敏感信息。	高危
WordPress Icegram Email Subscribers & Newsletters 插件 4.1.7 及之前版本中存在 SQL 注入漏洞，该漏洞源于基于数据库的应用缺少对外部输入 SQL 语句的验证，攻击者可利用该漏洞执行非法 SQL 命令。	超高危

按照训练集占 80%，测试集占 20%的比例划分总数据集。最终训练集包含 117,193 条数据，测试集包含 29,321 条数据。

3.3.3 实验结果

我们使用分类正确率作为评估标准，这里我们发现对于层级稀疏优化算法，在参数保留比例降至 10%以下时，其泛化效果相比于不进行稀疏优化时都没有大幅度的下降，因此我们将参数保留比例 ρ 的选择范围拓展至 {0.01,0.02,0.03,0.04,0.05,0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1}。此外，由于模型参数量太大并且过拟合问题严重，如果以稀疏优化前后模型参数的距离作为算法停止的标志，训练时间将是不可接受的，因此我们在该实验中采用早停策略控制算法的停止，即模型在测试集上连续五轮正确率不变时，算法终止。最终实验结果如下表所示。

表 3-4 实际文本分类实验结果

参数保留比例ρ	全局稀疏优化		层级稀疏优化	
	acc_{train}	acc_{test}	acc_{train}	acc_{test}
100%	71.70%	66.63%	71.70%	66.63%
90%	65.25%	65.46%	95.05%	65.69%
80%	74.36%	66.95%	94.34%	65.68%
70%	88.10%	66.89%	93.54%	65.75%
60%	86.54%	66.61%	92.00%	66.14%
50%	90.12%	66.44%	88.56%	66.21%
40%	85.72%	66.29%	83.89%	66.45%
30%	84.93%	66.36%	79.36%	66.11%
20%	81.18%	65.97%	74.72%	66.00%
10%	60.55%	28.81%	78.36%	65.71%
5%	62.03%	36.46%	74.59%	64.95%
4%	47.86%	30.03%	74.51%	64.62%
3%	61.62%	50.02%	75.09%	64.19%
2%	60.42%	50.02%	71.74%	62.70%
1%	59.85%	50.02%	64.75%	53.94%

针对上述实验结果，有以下四点讨论：

- 首先从全部实验结果来看，该文本分类实验存在非常严重的过拟合问题，基本对于不同的稀疏优化算法以及不同的参数保留比例，最终训练集上的正确率都远远高于测试集上的正确率，比如对于层级稀疏优化算法，当参数保留比例为90%时，模型在训练集上的正确率比在测试集上的正确率高出了约 30%，这验证了我们在 1.1 中提到的目前的模型具有非常严重的参数冗余的问题。
- 对于全局稀疏优化算法来说，保留参数在 20%以上时，模型的泛化效果可以近似看做不变，在保留参数比例 70%、80%时，模型的泛化效果超过了不进行稀疏优化的时候，这再次验证了模型参数的大量冗余以及我们的算法的有效性。对于保留参数比例从 10%到 4%的结果，从直觉上来说，对于四分类问题，如果模型未从训练集上学习到任何知识，其在验证集以及训练集上的正确率都应该趋近于 25%，即随机猜测。我们原本对模型效果变化趋势的预测是，随着模型保留参数比例下降，模型效果应该随之下降并且正确率应趋近于 25%。而实际在保留参数比例 10%时，模型在测试集上的正确率最低。这可能是早停策略的原因，模型训练终止的条件由收敛变成了测试集上正确率连续五轮不升高。从宏观角度来说，模型在训练轮次趋于无穷时，模型在测试集上的正确率变化应呈现我们预测的趋势，因此这是一个正常的现象。
- 对于层级稀疏优化算法来说，参数保留比例在 2%及以上时，模型的泛化效果可

以看做不发生明显下降，注意这里保留参数比例 2%意味着我们可以使用压缩稀疏行和压缩稀疏列的存储方式来存储我们的参数矩阵了，这大幅度地降低了我们的模型的部署难度，并且验证了参数的冗余和我们的算法的有效性。我们发现层级稀疏优化算法的实验结果中没有出现全局稀疏优化算法中的反常现象，这可能是由于层级稀疏优化的模型对稀疏算法的容忍度较高，训练过程中在测试集上的正确率呈稳定上升的趋势，因此并未出现全局稀疏优化算法中的反常现象。于是我们分别画出了当保留参数为 10%时，模型在测试集上的正确率随训练轮次变化的曲线图，如图 3-14 所示，左图是全局稀疏优化算法，右侧是层级稀疏优化算法，可以看到全局稀疏优化算法的测试集正确率波动很大，而层级稀疏优化算法的测试集正确率则非常稳定，这就验证了我们的假设。

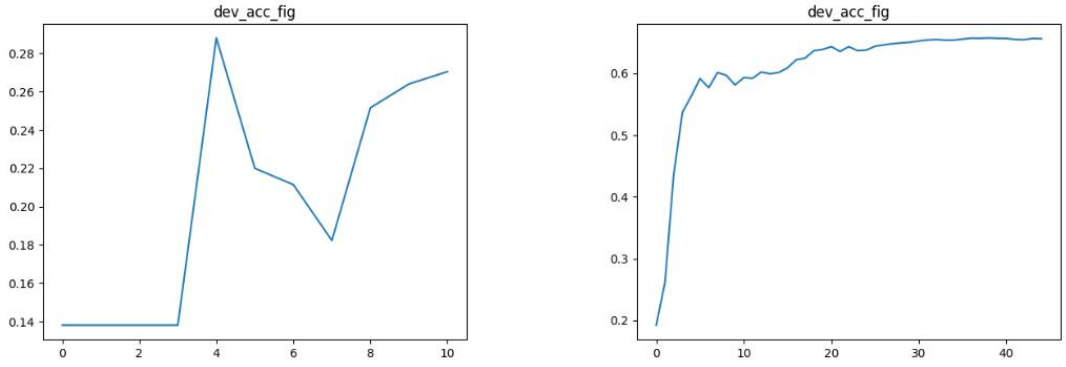

图 3-14 全局稀疏优化算法和层级稀疏优化算法测试集正确率随训练轮次变化对比图

- 对比全局稀疏优化算法和层级稀疏优化算法，我们发现层级稀疏优化算法能够进一步降低保留参数比例，造成这个结果的原因除了有第三点中阐述的层级稀疏优化算法更加稳定以外，我们还检查了保留参数比例为 10%时，该模型的最后一个线性层中非零参数的比例，假设为W_{final}，$W_{final} \in \mathbb{R}^{65536\times4}$。即最后一个线性层共有 262144 个参数，而最终算法停止后，最后一个线性层中非零参数共 2 个，与 3.1.2 拟合 Sinc 函数的回归实验结果中提到的结论相同，即全局稀疏优化算法会让最后一层参数基本全部置为 0，这也是导致全局稀疏优化算法不如层级稀疏优化算法的原因之一。

除此之外，我们还测试了全局稀疏优化算法和层级稀疏优化算法在时间成本上的差异，之前在第一部分中提到了全局稀疏优化算法的主要速度瓶颈是在将模型中的所有参数进行排序这一步，由于我在这一步是使用了堆排序，因此时间复杂度从理论上来说应该是 O(n$\log_2 n$)，主要对比排序这一步的时间。值得注意的是，我们的实际文本分类模

型主要包含四个 Transformer 的解码器，加上最后的线性层一共有约 180 万参数，而我们的层级稀疏优化算法在对模型进行稀疏优化的时候，这里的“层”主要指的是每一个解码器，根据时间复杂度可以推知层级稀疏优化算法所消耗的时间应该是全局稀疏优化算法的四分之一，实际实验结果如表 3-5 所示。

表 3-5 全局稀疏优化算法和层级稀疏优化算法排序阶段消耗时间对比

稀疏优化算法	排序阶段所消耗时间/s
全局稀疏优化算法	41.55
层级稀疏优化算法	10.58

虽然在我们的模型中全局稀疏优化算法在排序阶段所消耗的时间还不算不能接受，但是这里要注意我们的模型规模相比于现在参数上千亿的大语言模型来说已经算是很小了，如果不考虑内存粗略估算，对一个 7B 规模的模型使用全局稀疏优化算法排序阶段所消耗的时间大约在 2.95 天左右，这是完全不可接受的。但是对于层级稀疏优化算法来说，以 LLaMA2-7B 为例，其中共有 32 个解码器，则如果对 32 个解码器并行排序则只需要 2.21 个小时来完成排序的步骤，这是完全可以接受的，因此，对于规模更大的模型在训练阶段基本只能考虑层级稀疏优化算法来进行稀疏优化。

3.4 实验小结

前两个实验主要使用了小规模的神经网络分别完成了拟合 Sinc 函数曲线和双螺旋数据分类任务，在这两个实验中我们不仅提供了可视化结果来帮助我们直观的看到算法中模型参数的变化，还提供了主实验数据来证明我们的算法的有效性，同时给出了为什么在同一保留参数比例下全局稀疏优化算法效果往往不如层级稀疏优化算法的解释。最后一个实验主要使用了更大规模的基于 Transformer 的神经网络，建立了一个实际文本分类模型。在这个实验中我们详细分析了主实验结果，还给出了对算法速度瓶颈以及可能造成的影响的分析。这些实验以及讨论证明了我们所提出的算法的有效性，尤其是在实际文本分类模型上使用层级稀疏优化算法能够在不影响模型泛化效果的前提下只保留 2%的参数。

4 结论

我们从理论出发设计了全局稀疏优化算法和层级稀疏优化算法，并设计了三个实验验证了我们的算法在对参数矩阵进行稀疏化上的有效性，间接证明了我们提出的算法能够降低模型训练、部署、推理的成本，达到了我们的目的。但是我们在实验的过程中发

现，参数保留频次高的位置与最终模型中非零参数的位置重合度非常高，这启示我们可以在训练的中途提前确定这些位置，后续就无需再计算所有参数的梯度了，从而大幅度降低模型训练过程中的计算量。至于对最终稀疏优化效果的影响则还需要在未来设计具体实验来证明。

参考文献

[1] SZEGEDY C, VANHOUCKE V, IOFFE S, et al. Rethinking the inception architecture for computer vision; proceedings of the Proceedings of the IEEE conference on computer vision and pattern recognition, F, 2016 [C].

[2] PERRAULT R, CLARK J. Artificial Intelligence Index Report 2024 [J]. 2024.

[3] KALCHBRENNER N, ELSEN E, SIMONYAN K, et al. Efficient neural audio synthesis; proceedings of the International Conference on Machine Learning, F, 2018 [C]. PMLR.

[4] REAGEN B, WHATMOUGH P, ADOLF R, et al. Minerva: Enabling low-power, highly-accurate deep neural network accelerators [J]. 2016, 44(3): 267-78.

[5] CHEN Y-H, KRISHNA T, EMER J S, et al. Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks [J]. 2016, 52(1): 127-38.

[6] ALBERICIO J, JUDD P, HETHERINGTON T, et al. Cnvlutin: Ineffectual-neuron-free deep neural network computing [J]. 2016, 44(3): 1-13.

[7] HAN S, POOL J, TRAN J, et al. Learning both weights and connections for efficient neural network [J]. 2015, 28.

[8] DENIL M, SHAKIBI B, DINH L, et al. Predicting parameters in deep learning [J]. 2013, 26.

[9] HOEFLER T, ALISTARH D, BEN-NUN T, et al. Sparsity in deep learning: Pruning and growth for efficient inference and training in neural networks [J]. 2021, 22(241): 1-124.

[10] MOLCHANOV D, ASHUKHA A, VETROV D. Variational dropout sparsifies deep neural networks; proceedings of the International conference on machine learning, F, 2017 [C]. PMLR.

[11] POLYAK A, WOLF L J I A. Channel-level acceleration of deep face representations [J]. 2015, 3: 2163-75.

[12] MICHEL P, LEVY O, NEUBIG G J A I N I P S. Are sixteen heads really better than one? [J]. 2019, 32.

[13] HAGIWARA M. Removal of hidden units and weights for back propagation networks; proceedings of the Proceedings of 1993 International Conference on Neural Networks (IJCNN-93-Nagoya, Japan), F, 1993 [C]. IEEE.

[14] ELSEN E, DUKHAN M, GALE T, et al. Fast sparse convnets; proceedings of the Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, F, 2020 [C].

[15] LI G, YANG P, QIAN C, et al. Stage-wise magnitude-based pruning for recurrent neural networks [J]. 2022.

[16] SIETSMA, DOW. Neural net pruning-why and how; proceedings of the IEEE 1988 international conference on neural networks, F, 1988 [C]. IEEE.

[17] CASTELLANO G, FANELLI A M, PELILLO M J I T O N N. An iterative pruning algorithm for feedforward neural networks [J]. 1997, 8(3): 519-31.

[18] LUO J-H, WU J, LIN W. Thinet: A filter level pruning method for deep neural network compression; proceedings of the Proceedings of the IEEE international conference on computer vision, F, 2017 [C].

[19] YANG D, GHASEMAZAR A, REN X, et al. Procrustes: a dataflow and accelerator for sparse deep neural network training; proceedings of the 2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO), F, 2020 [C]. IEEE.

[20] ZHUANG T, ZHANG Z, HUANG Y, et al. Neuron-level structured pruning using polarization regularizer [J]. 2020, 33: 9865-77.

[21] MOLCHANOV P, TYREE S, KARRAS T, et al. Pruning convolutional neural networks for resource efficient inference [J]. 2016.

[22] BECK A, ELDAR Y C J S J O O. Sparsity constrained nonlinear optimization: Optimality conditions and algorithms [J]. 2013, 23(3): 1480-509.

[23] ZHAO W, HUANG D-S, YUNJIAN G. The structure optimization of radial basis probabilistic neural networks based on genetic algorithms; proceedings of the Proceedings of the 2002 International Joint Conference on Neural Networks IJCNN'02 (Cat No 02CH37290), F, 2002 [C]. IEEE.

[24] VASWANI A, SHAZEER N, PARMAR N, et al. Attention is all you need [J]. 2017, 30.

[25] KOCOŃ J, CICHECKI I, KASZYCA O, et al. ChatGPT: Jack of all trades, master of none [J]. 2023, 99: 101861.

[26] DU Z, QIAN Y, LIU X, et al. Glm: General language model pretraining with autoregressive blank infilling [J]. 2021.

[27] YANG A, XIAO B, WANG B, et al. Baichuan 2: Open large-scale language models [J]. 2023.

[28] TOUVRON H, MARTIN L, STONE K, et al. Llama 2: Open foundation and fine-tuned chat models [J]. 2023.

致　　谢