arXiv:2303.04436v2 [math.OC] 7 Sep 2023

有理逼近与基于神经网络的逼近方法比较

Vinesha Peiris、Reinier D´ıaz Mill´an、Nadezda Sukhorukova、Julien Ugon

摘要

有理逼近和基于神经网络的逼近是现代逼近领域中的高效工具。这些方法能够对非光滑函数和非利普希茨函数（包括多变量定义域函数）进行精确逼近。本文比较了采用有理逼近、神经网络及其组合进行函数逼近的效率。研究发现，在决策变量数量相同的情况下，有理逼近优于基于神经网络的方法。数值实验表明，即使逼近参数的数量（即相应优化问题的维数）较少，有理逼近仍然高效。本文的另一项重要贡献在于改进有理逼近算法。具体而言，可以调整基于优化的有理逼近算法，以控制约束矩阵的条件数。这一简单调整使我们能够处理高维优化问题，并改进神经网络的设计。神经网络的主要优势在于能够处理变量数量庞大的模型：复杂模型可分解为若干简单的优化问题。因此，决策变量数量庞大是神经网络的固有特征。

数学主题分类（2020）主要分类：41A50, 41A20, 41A63, 65D10, 65D12, 65D15

关键词：切比雪夫逼近、有理逼近、神经网络逼近、多变量逼近

1 引言

神经网络之所以高效且广受欢迎，其核心在于通用逼近定理 [17], ：该定理指出，只要激活函数是非多项式函数，深度神经网络就能够以任意准确率逼近任意连续函数。这一强有力的理论结果为研究哪些激活函数在实践中效果最佳铺平了道路。

本文特别关注的是， [6] 提出了有理激活函数。有理函数颇具吸引力，因为网络本身可以学习其参数，而且 [6] 已证明，这种方法能够取得良好的逼近结果。关于有理函数一致逼近的全面文献综述、其几何

性质及其与多项式逼近的联系，可见 [35]。

在本文中，我们以这项工作为基础，并补充了一些观察。首先，有理函数本身可以以任意准确率近似任意函数。因此，可以用某个函数的有理近似来替代任何其他激活函数。这引出了一个问题：这样做是否有意义？也就是说，基于近似有理激活函数构建的网络，其表现是否与基于实际激活函数构建的网络相似。进一步地，如 [6], 所示，有理系数本身也可以学习，这将使采用有理近似的网络从根本上比其他网络更强大。

第二个观察是，有理函数族在加法、乘法和复合运算下是封闭的。具有有理激活函数的神经网络本身就是一个有理函数。这引出了一个问题：它们与有理逼近的经典方法相比如何。对此，理论上的 答案显然是，经典有理逼近更好，因为其搜索空间严格包含神经网络方法的搜索空间。实际差异是否显著，仍有待研究。这是本文的第二个目的。

为实现这些目标，本文比较了两种函数逼近技术：有理逼近和基于神经网络的逼近。我们还考虑了这两种方法的一些组合，以提升学习系统的性能。具体而言，我们提出了一种新方法，采用交替方法优化具有有理激活函数的神经网络的参数：固定一组参数时优化另一组参数，反之亦然。

本研究使用的神经网络具有特定结构，也称为深度学习。深度学习是现代人工智能领域的关键工具之一。深度学习有许多实际应用，包括数据分析、信号处理、图像处理等 [11, 33]。尽管有这些实际应用，深度学习本质上只是一种特定类型的函数逼近：待逼近的函数仅通过有限个点上的取值（训练数据）来确定，而逼近原型（近似类）则是仿射映射与所谓激活函数（特殊类型的单变量函数）的复合。关于现代视角下深度学习的全面详尽教材，可参见 [11]。

我们的第二种主要方法是有理逼近。在这种情况下，近似就是多项式函数之比。选择有理函数是很自然的：它们能灵活地近似各种函数，包括非光滑和非利普希茨函数 [19,30,31]。这种灵活性甚至可与自由节点分段多项式逼近 [28]相媲美。与此同时，相应的

优化问题是拟凸的 [7,18,26] ，而且有多种高效的计算方法可以求解这些问题 [9,20,24] ，这里只举几种。

我们的数值实验凸显了两种方法之间的差异。直接有理逼近方法能够更好地控制有理函数和目标，因此最终得到的近似结果比采用有理激活函数的神经网络更加准确。

除了比较不同的逼近方法外，我们还对有理逼近算法进行了额外改进。该改进的思路简单明了，可以作为我们方法的一部分轻松实现，并提高结果的稳定性。具体而言，我们将有理逼近中辅助线性规划问题所涉及矩阵的条件数限制在一定范围内。这一额外限制是一个简单的线性约束，可以自然地加入线性规划的约束条件中。 [32] 中曾针对矩阵函数提升提出过类似方法。在本文中，我们应用这一思路来改进有理逼近算法，尤其是在多维逼近的情况下。数值实验的结果非常令人鼓舞。

本文的组织结构如下。第2节介绍本文所采用逼近方法的背景：神经网络、有理逼近及其组合。第3节解释本文使用的逼近模型。随后，我们在第4至6节对这些模型进行比较。最后，在第7节中给出结论，并指出可能的未来研究方向。附录 A.1包含补充结果，包括隐藏层中仅有两个节点时的结果。隐藏层中仅有两个节点的近似结果并不准确，但为了一致性和便于比较，我们仍将这些结果列出。

2 预备知识

2.1 深度学习

深度学习是用于数据和函数逼近的强大工具，也可用于数据分析和数据分类。深度学习的强大之处在于其结构：从现代优化理论的角度来看，系统需要解决的子问题都非常简单，因此系统能够高效地处理包含数百甚至数千个决策变量的问题。

深度学习的起源本质上是数学的。从本质上说，深度学习的目标是解决一个近似问题：优化网络的权重（参数）。这些权重是某些优化问题中的决策变量，而这些问题的目标函数表示逼近误差。因此，自然可以采用这种方法来处理

问题，使用现代优化工具 [11, 12, 33]。

深度学习坚实的数学基础建立于 [10,14,17,29]。这些研究依托于著名的柯尔莫哥洛夫–阿诺德定理 [3, 15] 的结果。柯尔莫哥洛夫–阿诺德表示定理指出，每个多元连续函数都可以表示为连续一元函数在二元加法运算下的复合。一般而言，不存在用于构造这些复合函数的算法。因此，现代深度学习技术并不构造这种表示，而是通过仿射变换与所谓的激活函数的复合来近似该复合函数。激活函数是一元非多项式函数。最常用的激活函数是 Sigmoid函数和修正线性单元（ReLU）函数（单调分段线性函数 $\varphi(x) = \max\{0, x\}$）。

[6] 展示了使用有理逼近函数的神经网络与 ReLU网络之间的等价性，另见 [34]。具体而言，他们证明，任意 ReLU网络都可以由规模为 $\mathcal{O}(\log(\log(1/\varepsilon))$ 的有理网络以误差不超过 $\varepsilon > 0$ 进行近似；任意有理网络也可以由规模为 $\mathcal{O}(\log(1/\varepsilon))^3$ 的网络以误差不超过 ε 进行近似。他们还进行了数值实验，以比较两种有理激活函数。其方法的一个有趣特点是，激活函数的参数本身也可以作为学习过程的一部分进行优化。他们的研究提出了一个问题：采用直接的有理逼近方法，与基于神经网络的近似相比效果如何?

大多数深度学习算法依赖基于最小二乘的误差度量（损失函数）。由于损失函数用于度量误差，优化的目标就是通过最小化相应的损失函数来优化权重。因此，基于最小二乘的模型是在最小化一个平滑的二次函数，因而可以采用快速而简单的优化技术（例如梯度下降）。然而，在某些情况下，其他损失函数更有效，例如基于一致范数的损失函数等。

本文旨在比较深度学习与其他逼近技术所获得的近似结果。具体而言，我们关注有理逼近，因为它具有较强的逼近能力 [28]。下一节将进一步介绍有理逼近。另一方面，在 [34], 中，作者证明了有理函数与采用标准 ReLU 激活函数的神经网络具有同等的能力。这一结果促使深度学习领域的许多研究者将神经网络、有理函数和有理逼近技术结合起来，以提升彼此的性能 [6, 22, 27]。

2.2 有理逼近

在切比雪夫（一致）意义下的有理逼近曾是20世纪50至70年代非常热门的研究方向 [1, 5, 8, 21, 30, 31] 以及许多其他方向。

有两大类有理逼近方法。第一类方法 [23]旨在求得“近乎最优”的解。这种方法（也称为 AAA 方法）效率很高，因此广受欢迎，但它只是“近乎最优”。AAA 方法向多变量情形的扩展仍有待研究。此外，该方法仅适用于无约束优化；正如本文稍后将会看到的，这会限制我们处理病态矩阵的能力。第二类方法基于现代优化技术：相应的优化问题是拟凸的，可以使用通用拟凸优化方法求解。

有很多有理逼近方法 [4,16,25,30] （这里只列举几种），但大多数方法的实现都依赖于求解线性规划问题。因此，可以轻松地在这些实现中添加额外的线性约束，而不会使问题变得复杂。目前，有理逼近最常用的优化方法是针对拟凸函数的二分法和微分校正法。这两种方法的优点是，它们可以轻松扩展到非单项式情形（广义有理逼近） [8,26] 以及多变量设置 [2]。总体而言，微分校正法具有二次收敛性 [4], ，而二分法则呈线性收敛 [7]。因此，在单变量有理逼近实验中，我们采用微分校正法。同时，由于二分法简单易用，它仍是不错的选择。因此，在多变量有理逼近实验中，我们采用二分法。

3 计算模型

我们比较了六种用于逼近给定连续函数的不同方法。其中四种基于神经网络（NN），其余两种则纯粹基于有理逼近。具体方法如下：

1. 使用 ReLU 激活函数的神经网络；2. 使用 ReLU 有理逼近激活的神经网络（我们采用微分校正法来逼近 ReLU）；3. 使用有理激活函数的神经网络，有理激活函数的系数由神经网络学习；4. 使用有理激活函数的神经网络，采用分割方法学习有理激活函数的系数和网络参数，训练过程分为三个步骤：先在输入层与隐藏层之间进行，然后在隐藏

层与输出层之间进行，最后学习有理激活函数的系数，

5. 有理逼近（微分校正），这是输入层与输出层之间的直接有理逼近，

6. 有理逼近（AAA），这是输入层与输出层之间直接进行的“近乎最优”有理逼近。

我们使用三层神经网络：输入层、隐藏层和输出层。对于隐藏层的节点数量，我们采用以下设置：

- 隐藏层有 2 个节点的网络（见附录 A.1），
- 隐藏层有 10 个节点的网络。

在实验中，我们使用不同的训练轮次数：50、100 和 200。我们记录每种网络训练过程每个训练轮次所需的训练时间。

对于神经网络方法，我们采用基于均方误差的损失函数（MSE 代表均方误差，简单来说就是“最小二乘”）和均匀损失。对于基于均方误差的损失，我们使用 ADAM；对于均匀损失，我们使用 ADAMAX（ADAM 的一种专为均匀损失设计的版本）。所有基于有理函数的逼近模型均针对均匀损失设计；与最小二乘相比，均匀损失是一种更严格的准则，如果目标是逼近一个函数，它显然更为合适。

在我们的实验中，我们考虑两种不同的设置。

- 定义域为一条线段（单变量函数）。我们对非光滑函数进行逼近

$$f(x) = \sqrt{|x - 0.25|}, \quad x \in [-1, 1],$$

  通过使用上述方法并设置不同条件（隐藏层中的节点数、训练轮次等）。选择函数 $f(x)$ 是因为它对大多数逼近技术而言都很难：该函数在 $x = 0.25$ 处是非光滑且非利普希茨的。

- 定义域为双变量。也可以将结果扩展到高维情形，但所有这些扩展都不在本文讨论范围之内。

下一节中，我们仅展示数值实验的重要结果。具体而言，我们报告隐藏层包含 10 个节点且损失函数采用一致范数形式的神经网络相关结果。附录 A 对其余结果进行了十分详尽的讨论。

训练轮次	最终损失	最小损失	运行时间（每个周期）
50 45	0.130828	 0.117747	2.52s ± 79.3ms
100 97	0.139669	 0.111249	2.51s ± 96.8ms
200 187	0.138172	 0.088032	2.49秒 ± 59.2毫秒

表1：结果：采用 ReLU 激活的神经网络实验

4 结果：基于神经网络的近似。

本节的发现来自一个隐藏层包含 10 个节点的神经网络。损失函数基于一致范数，优化器采用 ADAMAX。

4.1 采用 ReLU 激活的神经网络

我们从标准 ReLU 激活函数开始进行实验。表1展示了不同训练轮次下的损失函数值和计算时间。

可以看出，随着训练轮次增加，损失函数值并未显著改善。此外，观察发现，报告的损失函数最小值可能出现在最后一轮训练之前。例如，在训练 200 轮的情况下，最小值（0.088032）出现在第 187 轮，而报告值（训练 200 轮后）为 0.138172。该表最后一列报告了每轮训练的运行时间，各轮训练的计算时间没有显著差异。

图1展示了使用 ReLU 激活的网络计算得到的近似。与隐藏层节点较少的网络相比，这些近似的准确率显著更高。此外，在采用均匀损失时，“困难点” $x = 0.25$ 附近的近似效果似乎也优于采用均方误差时的效果（均方误差对应的图可见附录 A）。因此，当需要在这类点附近获得准确的近似时，均匀损失可能是更好的衡量标准。

4.2 使用 ReLU 激活的有理逼近神经网络

本节将介绍数值实验的结果。实验设置与第4.1节类似，但激活

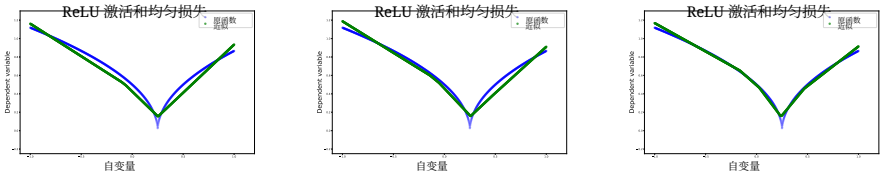

图1：近似由采用 ReLU 激活函数的神经网络计算，训练轮次分别为 50、100 和 200。

训练轮次	最终损失	最小损失	运行时间（每个周期）
50 46	0.175439	 0.125264	2.82s ± 90.5ms
100 99	0.105346	 0.071646	2.77秒 ± 79.5毫秒
200 195	0.084491	 0.053663	2.83秒 ± 382毫秒

表2：结果：采用 ReLU 有理逼近的神经网络实验

函数采用 ReLU 的有理逼近。该近似是通过微分修正法找到的。

我们的激活函数是一个次数为 (3,2) 的有理函数：该有理逼近是两个多项式的比值，分子的次数为 3，分母的次数为 2。

有理激活函数的系数来自对 ReLU 函数的最佳有理 (3,2) 逼近。这些系数在整个训练过程中保持固定。我们不会与网络的其余参数一同学习这些系数。

表2展示了与第4.1节中的标准 ReLU 激活相比，准确率的提升。当训练轮次为 100 或 200 时，这一现象尤为显著。

图2显示，即使对于“难点”，近似的准确率也在提高，尤其是训练轮次达到 100 轮或更多时。

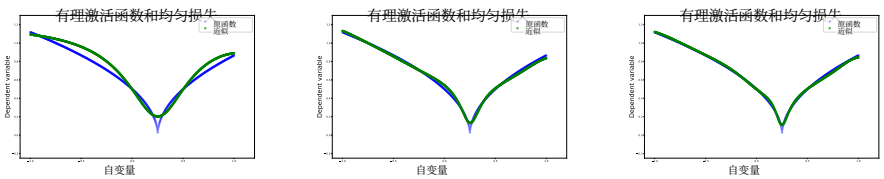

图2：由具有有理激活函数的神经网络近似 ReLU，训练轮次分别为 50、100 和 200。

训练轮次	最终损失	最小损失	运行时间（每个周期）
50 44	0.131586	 0.114195	2.75秒 ± 77.9毫秒
100 96	0.104754	 0.097785	2.64秒 ± 62.1毫秒
200 177	0.078658	 0.055616	2.76秒 ± 423毫秒

表3：结果：有理激活函数实验

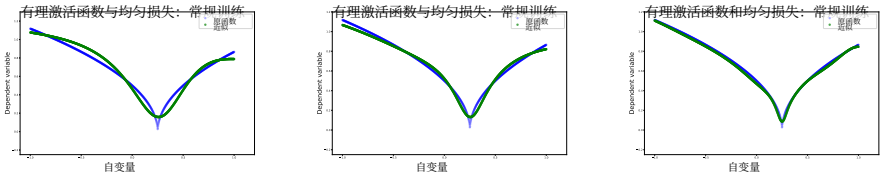

图3：近似通过常规训练过程计算，训练轮次为50、100、200。

总体结论：使用 ReLU 的有理逼近作为激活函数，而非直接使用 ReLU 时，近似结果更加准确。这一观察对于均方误差和均匀损失均成立。均方误差相关结果见附录 A。

4.3 使用有理激活函数的神经网络

在这种情况下，我们的激活函数是一个次数为 (3，2) 的有理函数。有理激活函数的系数现在属于参数集的一部分。在训练过程中，我们在学习其他参数的同时也学习这些系数。这类使用系数可学习的有理激活函数的网络称为“有理神经网络”，更多详情请见 [6]。本节的 Python代码也来自 [6]。

表3显示，当隐藏层中的节点数为10时，最优损失函数值与 ReLU 被有理函数近似时的情况非常接近。图3证实了这些结果。

当训练轮次为 200 时，即使在“困难点”附近，近似结果似乎也很准确（与均方误差类似）。

4.4 使用有理激活函数的神经网络：采用分割训练方法进行学习

在本节中，我们的激活函数是一个次数为 (3，2) 的有理函数。这些系数现已成为参数集的一部分。与该网络类似

训练轮次	最终损失	最小损失	运行时间（每个周期）
50 49	0.168947	 0.113316	2.65秒 ± 72.2毫秒
100 75	0.093841	 0.087595	2.71秒 ± 662毫秒
200 190	0.061735	 0.053655	2.82秒 ± 384毫秒

表 4：结果：使用分割方法进行的实验

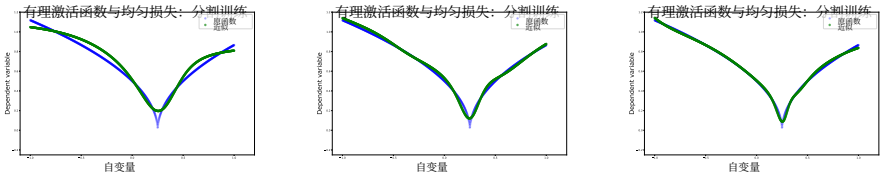

图 4：通过分割训练过程计算近似，训练轮次为 50、100、200。

在第 4.4 节中，我们在训练过程中学习这些系数，就像学习其他参数一样。训练过程分为三个步骤：

1. 输入层与隐藏层之间的权重，
2. 隐藏层与输出层之间的权重；
3. 有理系数。

这种方法基于分块逐坐标优化；因此，即使它能生成一个使函数值不断下降的序列，也无法保证其极限点是全局最优的。

表 4 显示，就损失函数的最优值而言，该方法与采用有理激活函数的方法，以及对 ReLU 激活函数进行有理逼近的方法，所得结果相当。

图 4 显示，随着训练轮次增加，近似效果不断提升，甚至在“困难点”附近也是如此。

总体结论：与常规训练方法相比，使用有理激活函数并采用分割训练方法时，近似结果更加准确。

5 直接有理逼近方法：微分修正与 AAA 方法。

本节中的所有实验均采用均匀损失。决策变量是有理函数的系数。我们比较两种主要的

次数	误差（均匀范数）	运行时间
(21,20)	0.04376536257200335	10.9s
(20,20)	0.04841949102524923	11.5秒
(21,21)	0.04411128396473672	12.4秒

表 5：通过微分校正法对次数为 (21, 20)、(20,20) 和 (21,21) 进行直接有理逼近。

构造有理近似的方法：微分修正法（收敛至全局最优解）和 AAA 方法（一种实用方法，可收敛至“近乎最优”解）。这两种方法的代码均使用 Python 实现。

5.1 微分修正方法

5.1.1 次数为 (21,20) 的有理逼近，微分修正方法。

对于一个三层神经网络，如果隐藏层只有 10 个节点，且采用次数为 (3,2) 的有理激活函数，其输出就是一个次数为 (21,20) 的有理函数。因此，我们可以使用微分修正方法，通过次数为 (21, 20) 的有理函数来逼近函数 $f(x) = \sqrt{|x - 0.25|}$，并将此近似结果与均匀损失函数、隐藏层为 10 个节点时得到的结果进行比较。隐藏层节点数为 2 时的相关结果见附录 A。

理论上，直接使用次数为 (21,20) 的有理函数进行逼近会更加灵活。我们在实验中使用 Python。

我们还计算了次数为 (20,20) 的有理近似以供比较。此外，我们还将利用这些额外结果与 AAA 方法进行比较，因为 AAA 方法要求分子和分母的次数相同。表 25 总结了计算时间和最优损失。结果表明，在这三种设置下，计算时间都很短，且相应的误差非常相似。有趣的是，提高次数并不总能改善最优损失函数值。这可能是因为随着相应优化问题的维度增加，计算不稳定性也随之加剧。

表 5 总结了次数为 (21,20)、(20,20) 和 (21,21) 时的计算时间和最优损失值。这些结果非常相似。

图 5 展示了次数为 (21,20) 的近似。该近似精确，但存在明显的振荡；对于高次数有理函数而言，这并不令人意外。

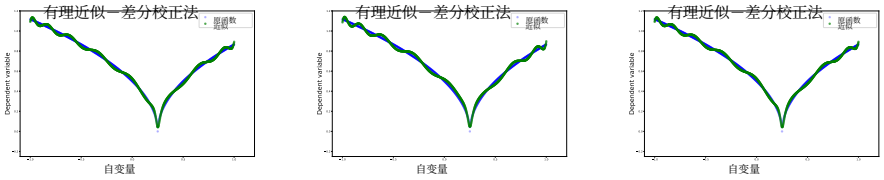

图 5：通过微分校正法进行近似，有理逼近次数分别为 (21，20)、(20，20) 和 (21，21)。

次数	误差类型	误差	运行时间
(20,20)	一致误差	0.00026068574873364114	450毫秒
	MSE	0.0005497563210130126	
(21,21)	一致误差	3.67961692604446e-06	496毫秒
	MSE	4.8434496396255 75e-05	

表6：结果：AAA 方法，次数 (20,20) 和 (21,21)

5.2 AAA 方法

5.2.1 使用 AAA 方法进行次数为 (21,20) 的有理逼近

为了将微分修正方法与 AAA 方法进行比较，我们在与微分修正方法相同的实验设置下应用 AAA 方法。由于 AAA 方法的局限性，我们无法为分子和分母使用不同的次数。因此，在这些实验中，我们使用次数为 (20,20) 的有理逼近和次数为 (21,21) 的有理逼近。代码由 C. Hofreither [13] 使用 Python 开发。

表6和图6显示，近似结果非常准确，计算时间也很短。这些结果表明，即使次数很高，AAA 方法仍然非常快速且准确。从图中可以看出，函数与近似结果几乎无法区分。

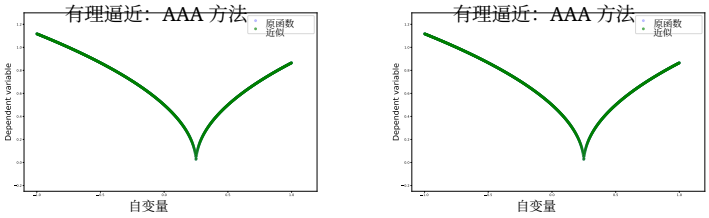

图 6：使用 AAA 方法计算近似，次数为 (20,20) 和 (21,21)。

6 多元定义域的结果

本节重现了 [6], 中第4.1节所展示的实验，作者在该节中近似求解 KdV（Korteweg-de Vries）方程:

$$u_t = -uu_x - u_{xxx}, \quad u(x,0) = -\sin(\pi x/20)$$

通过使用四种神经网络（分别包含 ReLU、正弦、有理式和多项式激活函数）。在本节中，我们的目标是使用多元二分法在 (x_i, t_i) 上逼近u_i ，该算法已在 MATLAB 中实现。数据集的具体信息如下:

- $(u_i)_{1\le i\le N}$- 大小为 512×201 的观测数据（已知函数值） .
- $(x_i, t_i)_{1\le i\le N}$ - 时空点（已知定义域值）。
- $x_i \in [0, 40]$ 步长为 1。
- $t_i \in [-20, 19.84375]$ 步长为 0.390625。

在 [6], 中，作者使用欧几里得范数计算误差，而在这里，我们使用一致范数计算近似的误差。因此，几乎无法将我们的结果与 [6] 中的发现进行比较。我们的所有实验都是通过选取一组定义域点构建的（使用每个维度上的每第 k^{th} 个点及其对应的函数值）。近似和一致误差项是在同一个约化定义域上计算的。采取这一额外步骤是为了确保大量数据不会导致 MATLAB 程序崩溃。

6.1 原始数据集

以下图片（图7、图8、图9）展示的是原始数据集。部分图像是通过选取一组定义域点构建的。具体而言，为了降低原始问题的维度，我们使用定义域（沿两个维度）的每第 20th 个点所组成的点对，以及定义域中每第 10th 个点所组成的点对。为简便起见，我们有时将这些点对称为“定义域中的每第 10th 个点”或“定义域中的每第 20th 个点”。

现在，我们计算不同次数的有理近似，并报告 $k = 20$ 和 $k = 10$ 时各个次数对应的一致近似误差和计算时间。我们的实验从次数为 (2，2) 的有理逼近开始，并逐步增加次数，直到 (20，20)。为了获得更好的近似，我们将有理函数中的经典单项式替换为切比雪夫单项式。

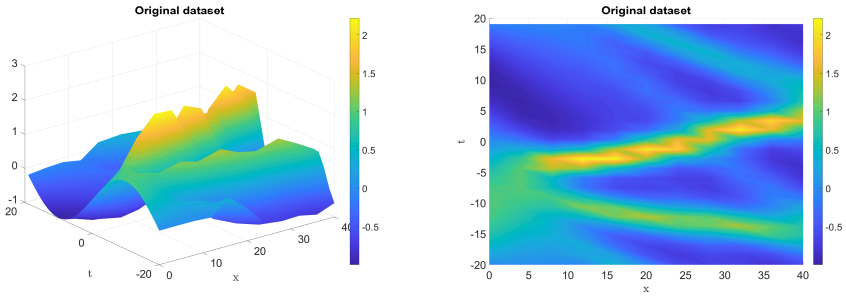

图7：通过取定义域中每个 20th 点的点对，为原始数据集构建的图像：原始数据集的三维视图和二维视图

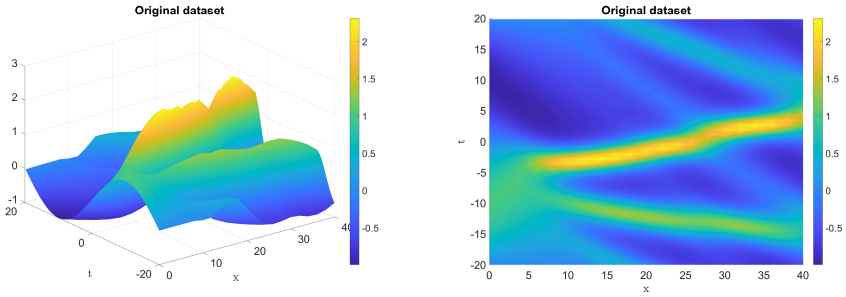

图8：通过取定义域中每个 10th 点的点对，为原始数据集构建的图像：三维视图和二维视图

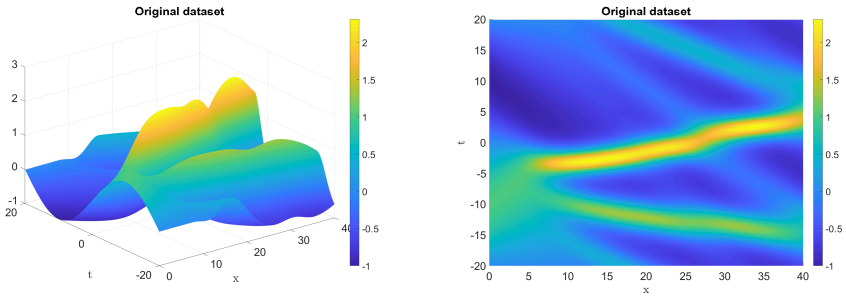

图9：通过取定义域中的所有点，为原始数据集构建的图像：三维视图和二维视图

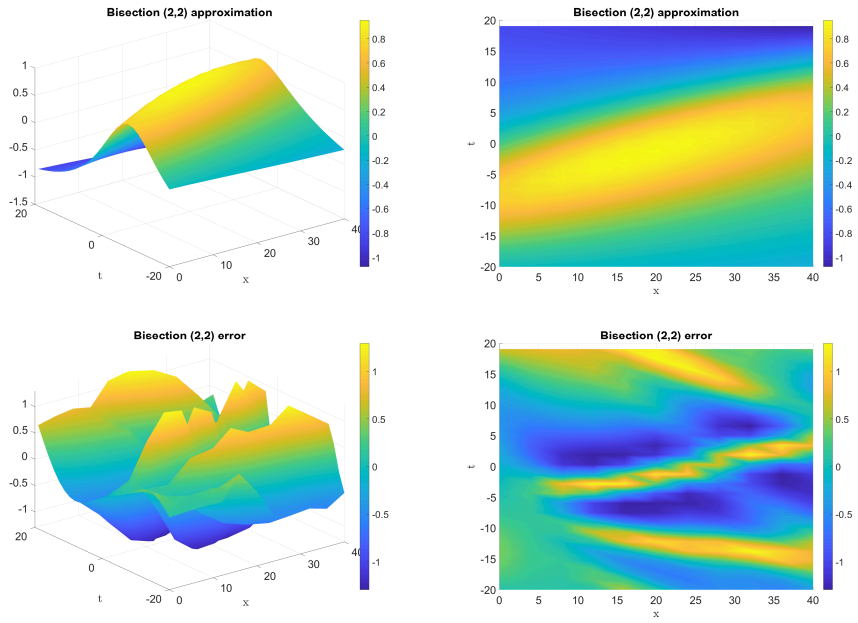

图10：近似（上方）和误差曲线（下方）是在由原始定义域中每隔20个点取一对点所构成的定义域上构造的：三维和二维视图

	一致误差	时间（秒）
定义域中的每第 20th 个点	1.29449744612994	2.049840
定义域中的每第 10th 个点	1.36991465992617	2.504411

表7：(2,2) 次近似的一致误差和计算时间

6.2 (2,2) 次有理逼近

我们展示了 (2,2) 次有理逼近的三维和二维图像，以及其误差曲线。

在图10中，我们仅考虑定义域中每隔 20th 个点所构成的点对来计算近似；在图11中，则在包含定义域中每隔 10th 个点所构成的点对的定义域上计算近似。计算时间和一致误差见表7。

显然，定义域中离散点的数量较少时，算法运行得更快。

6.3 (5,5) 次有理逼近

这里，我们将近似的次数提高到 (5,5) 次。

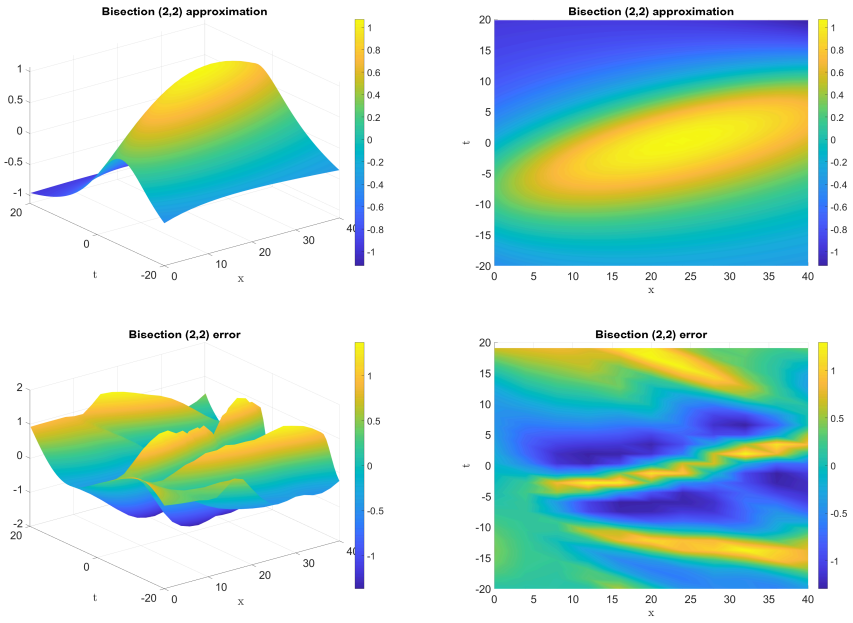

图11：近似（上）和误差曲线（下）是在由原始定义域中每个 10th 点组成的点对所构成的定义域上构造的：三维和二维视图

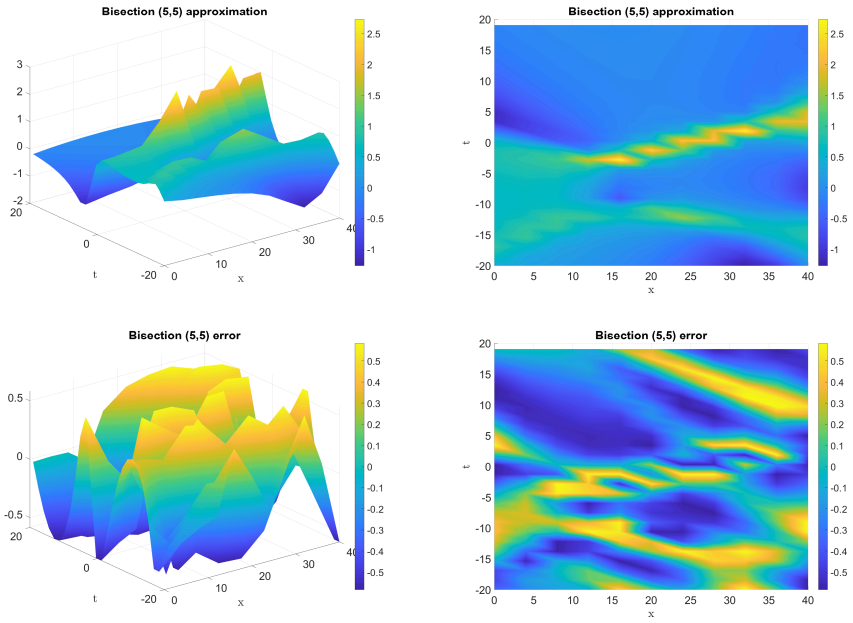

图 12：近似（上）和误差曲线（下）是在由原始定义域中每个 20th 点组成的点对所构成的定义域上构造的：三维和二维视图。

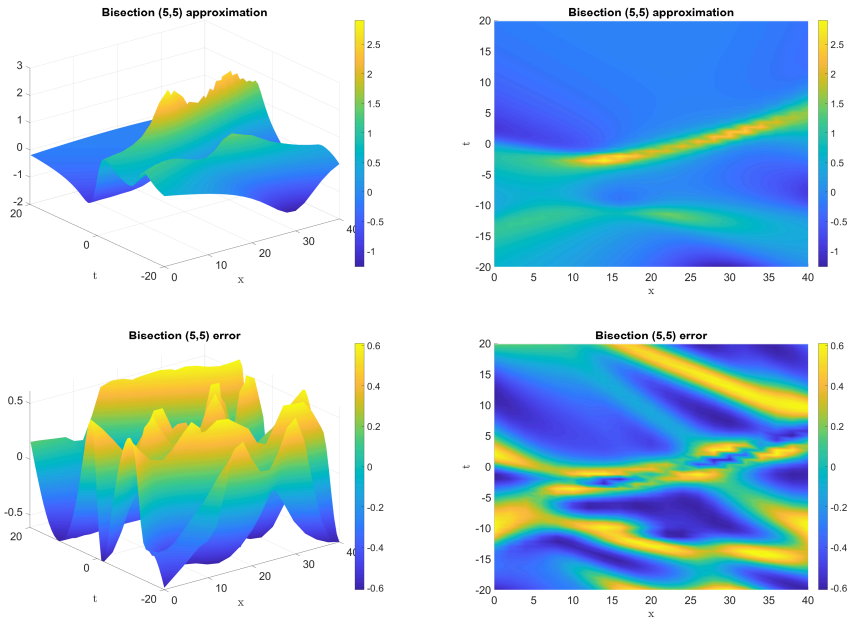

图 13：近似（上图）和误差曲线（下图）构建于由原始定义域中每个 10th 点的点对组成的定义域上：三维和二维视图。

	一致误差	时间（秒）
定义域中的每个 20th 点	0.583136100002126	7.823996
定义域中的每个 10th 点	0.609975243086065	10.455391

表 8：次数为（5,5）的近似的一致误差和计算时间

图 12 和图 13 分别展示了在定义域上进行近似计算的结果，其中每个 20th 点和定义域的 10th 点构成一对。计算时间和一致误差列于表 8。相比 (2,2) 次的情形，这些近似效果更好，一致误差也大幅降低。

6.4 (10,10) 次有理逼近

本节使用 (10,10) 次有理逼近来近似原函数。近似结果和误差曲线如图 14 和图 15 所示。在这两个定义域上，近似结果都很好地贴合原函数。

计算时间和一致误差列于表 9。一致误差项远小于次数为

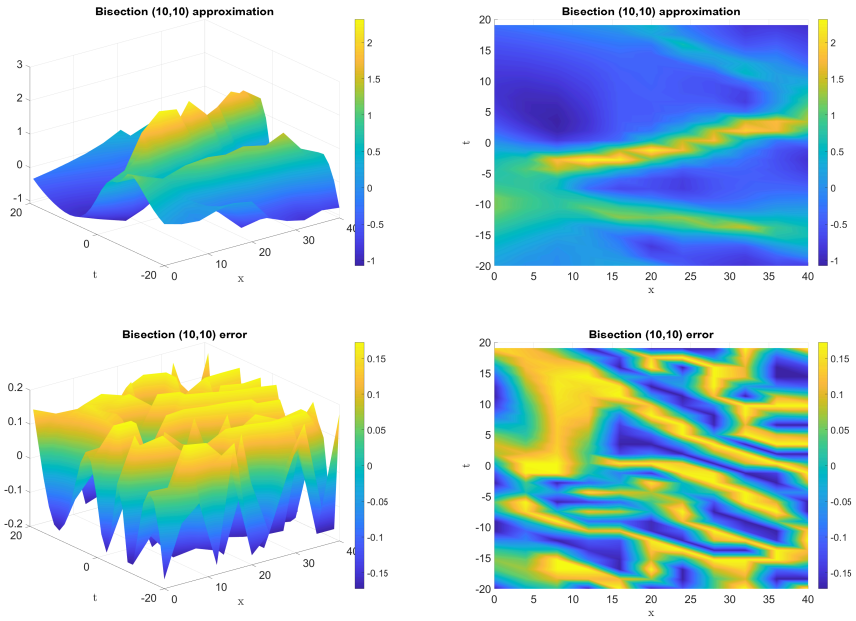

图 14：在由原始定义域中每个 20th 点的配对组成的定义域上构造的近似（上）和误差曲线（下）：三维和二维视图。

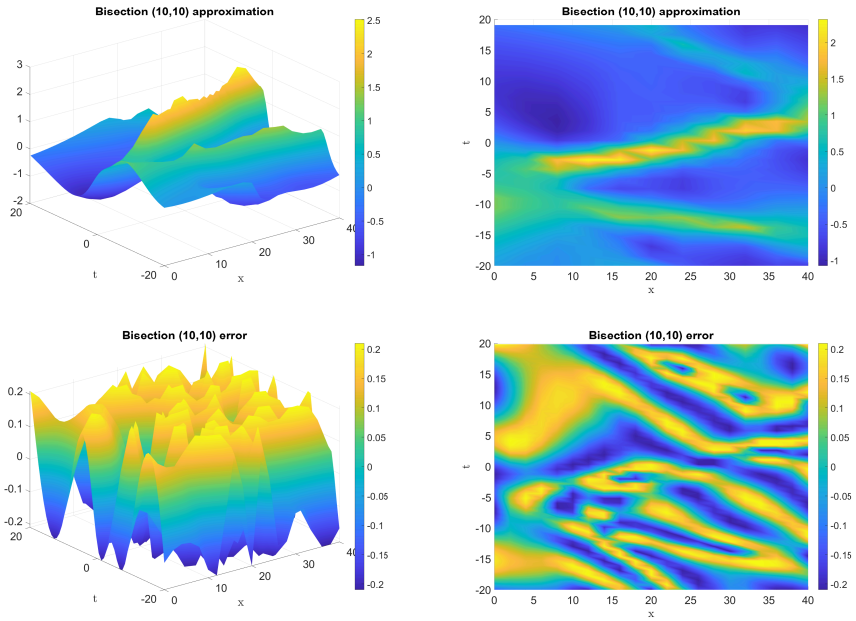

图 15：在由原始定义域中每个 10th 点的配对组成的定义域上构造的近似（上）和误差曲线（下）。

	一致误差	时间（秒）
定义域中的每个 20th 点	0.172559761245446	22.208227
定义域中的每个 10th 点	0.209717782244555	42.384391

表 9：(10,10) 次近似的一致误差和计算时间

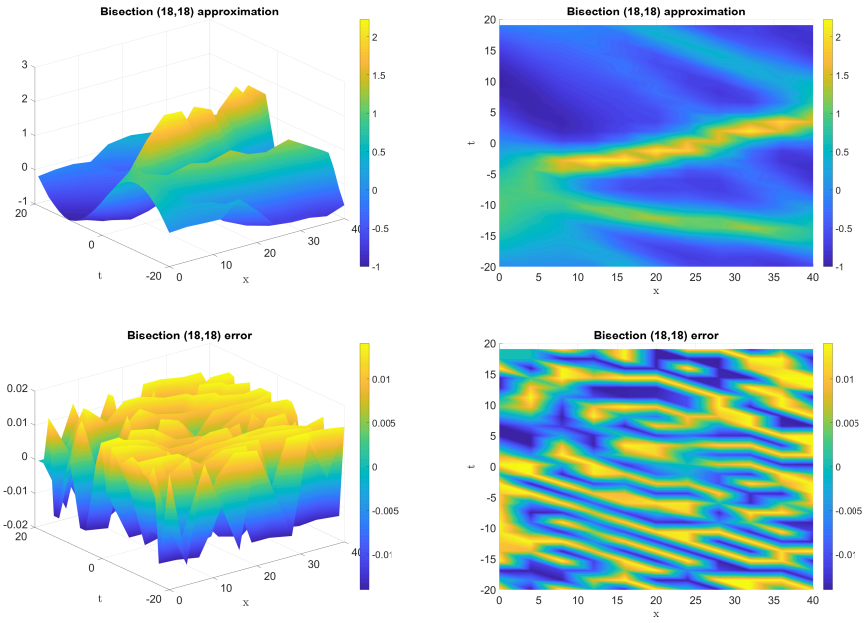

图16：近似（上方）和误差曲线（下方）构造于由原始定义域中每个 20th 点的配对组成的定义域上：三维和二维视图。

该近似的次数为 (5,5)。另一方面，随着次数增加，算法计算近似所需的时间也会变长，

6.5 次数为 (18,18) 的有理近似

现在，我们将近似的次数提高到 (18,18)。图16和图17所示的近似与图7和图8所示的相应原函数非常相似。从表10中还可以看出，一致误差比之前的情况小得多。

6.6 次数为 (20,20) 的有理近似

本节中，我们考虑用次数为 (20,20) 的有理函数近似图7和图8中的原函数。尽管次数

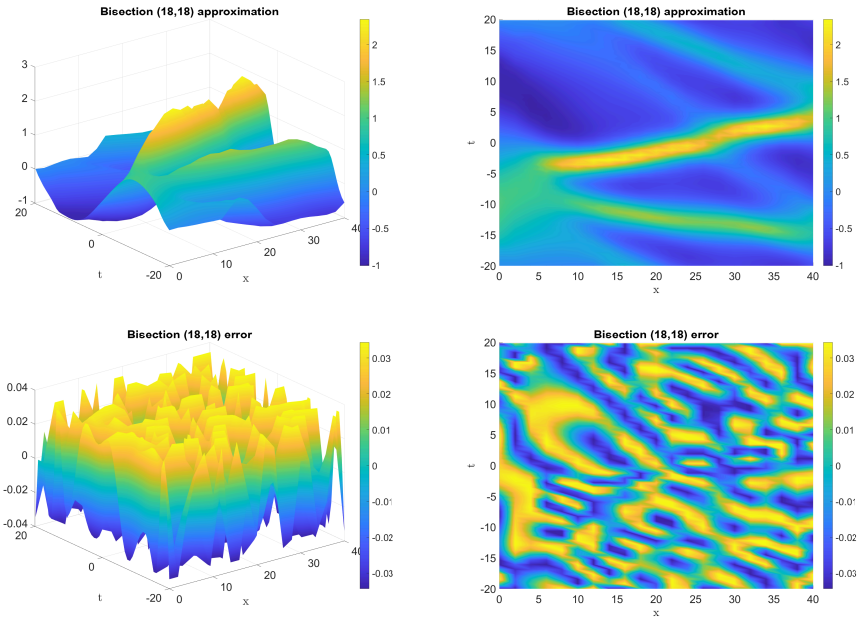

图17：近似（上方）和误差曲线（下方）构造于由原始定义域中每隔 10th 个点组成的定义域上：三维和二维视图。

	一致误差	时间（秒）
定义域中的每隔 20th 个点	0.014061551772703	66.460222
定义域中的每隔 10th 个点	0.034389125232539	167.995076

表10：次数为 (18,18) 的近似的一致误差和计算时间

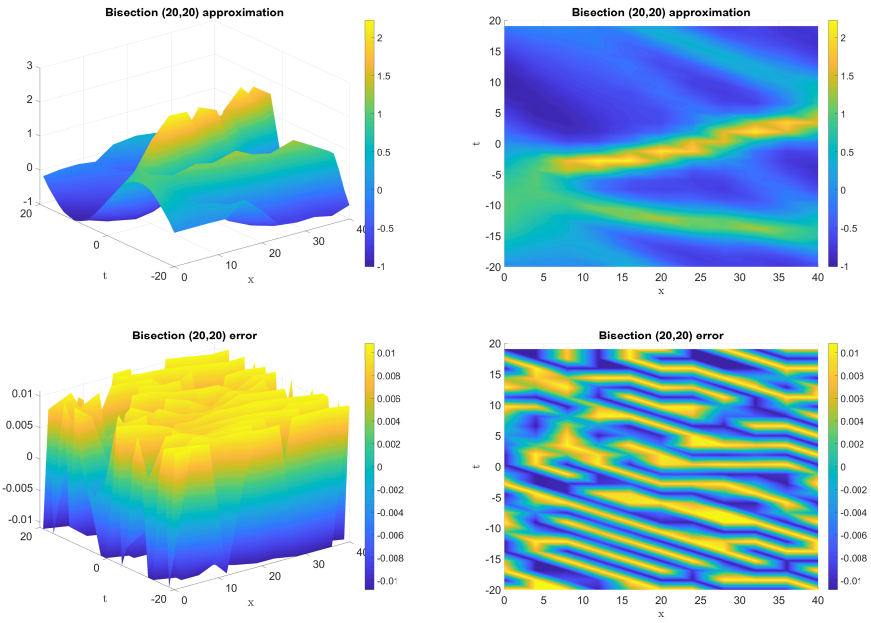

图18：近似（上）和误差曲线（下）是在由原始定义域中每个 20th 点的配对所组成的定义域上构造的：二维和三维视图。

有理函数的次数更高，但参数数量（分子和分母多项式的系数）较少（≈ 400），相比之下，作者在实验中使用的神经网络包含的参数数量为（≈ 8035）。 [6]

随着有理函数次数的增加，二分法辅助线性规划问题中出现的约束矩阵的条件数也会增大，在某些情况下，MATLAB 代码会报错并崩溃。可以通过在二分法的辅助问题中增加一个额外的线性约束来解决这一问题，该约束限制分母多项式的上界。二分法可以灵活地添加约束，这一额外调整使我们能够计算次数更高的有理近似。在 [32] 中也可以发现类似的观察，作者在矩阵函数求值中利用了这一性质进行单变量逼近（矩阵函数提升）。本节中近似次数为 (20，20) 的实验是在增加了限制分母上界的约束后进行的。数值实验表明，这项额外改进能带来更准确的近似，因为程序可以处理分子和分母中次数较高的多项式。

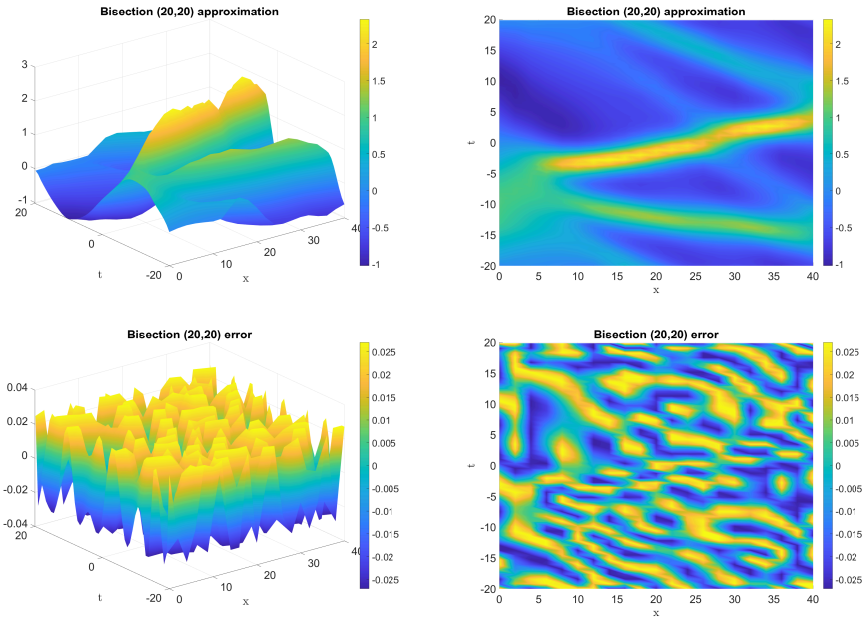

图19：近似（上方）和误差曲线（下方）是在由原始定义域中每个 10th 点的点对所构成的定义域上构建的：三维和二维视图。

	一致误差	时间（秒）
定义域中的每个 20th 点	0.010816799001175	158.933609
定义域中的每个 10th 点	0.027058335297138	438.548139

表11：次数（20,20）近似的一致误差和计算时间

	次数	一致误差
每一对 20th 点定义域的	(2,2)	1.49062088756304
	(5,5)	1.57543434735584
	(10,10)	245.34679368905
	(18,18)	11404.1284016
	(20,20)	88023.9109319522
每一对 10th 点定义域的	(2,2)	1.40803015961408
	(5,5)	0.702071289628257
	(10,10)	0.334154891477345
	(18,18)	15.3976686595557
	(20,20)	9.191409640164363

表12：在整个定义域上计算的一致误差

显然，图18和图19中的近似与原函数相吻合，且其一致误差远小于低次数近似的误差。

总体结论：由于误差是在不同范数下计算的，因此我们的结果无法与 [6] 中的结果进行比较。不过，我们证明了多元定义域上的有理逼近与神经网络一样强大，并且能够以比神经网络少得多的参数来近似复杂函数。

对于每个次数，我们还在选定的约化定义域上完成近似后，计算了整个定义域上的一致误差。结果见表12。请注意，在次数为 (20,20) 的情况下，分母有上界。

可以清楚地看到，当近似次数较低时，在整个定义域上计算的一致误差与在选定点较少的定义域（约化定义域）上计算的一致误差非常相似。当定义域由其中每隔 10th 个点组成的点对构成时，这一现象尤为明显。对于较高次数，当定义域中的点数少得多时（即定义域由整个定义域中每隔 10th 个点组成的点对构成时），在整个定义域和约化定义域上计算的误差之间的差异会大得多。

7 结论与未来研究方向

本文比较了几种函数逼近方法，每种方法都是对另一种方法稍作修改而来。我们从采用 ReLU 激活函数的经典神经网络入手，逐步调整设置：首先，将激活函数替换为逼近 ReLU 的有理函数；然后，将有理激活函数的系数纳入学习参数；最后，替换神经网络

，改用参数数量相同的最佳有理近似，并采用传统优化方法。

我们的发现是，随着每一步增加可行空间的灵活性和规模，我们观察到近似效果有所提升。尤其是，基于优化的近似比神经网络部分取得了更好的近似效果。

训练时间也有利于基于优化的近似，尤其是因为通常可以用较少的参数取得良好的近似效果。随着待近似函数的维度增加，这一优势是否仍然存在尚不明确。

我们的研究没有考察近似的其他方面，例如泛化能力。众所周知，深度学习通常具有良好的泛化能力，并能避免过拟合。然而，增加额外的灵活性（例如将激活函数的系数纳入学习参数）是否会保留这些泛化能力，尚不明确。对此还需要进一步研究，以加深理解。

致谢、声明与宣告

我们感谢澳大利亚研究理事会通过 Discovery Project DP180100602 资助本研究。

参考文献

[1] N. I. Achieser (1965) 逼近理论，Frederick Ungar，纽约。[2] A. Aghili、N. Sukhorukova 和 J. Ugon (2021)广义温度积分的双变量有理逼近，《数学化学杂志》，第59卷第9期，2049–2062页。[3] V. Arnold (1957)关于三变量函数，《苏联科学院通报》，第114卷第5期，679–681页；英文译文见《美国数学会译丛》第28卷（1963），第51–54页。[4] I. Barrodale、M. Powell 和 F. Roberts (1972) 有理逼近的微分修正算法 L_∞，《SIAM 数值分析杂志》，第9卷第3期，493–504页。[5] B. Boehm (1964) 最佳有理切比雪夫逼近为多项式的函数，Numerische Mathematik，第6卷，235–242页。[6] N. Boull´e、Y. Nakatsukasa 和 A. Townsend (2020) 有理神经网络，《神经信息处理系统进展》，第33卷，14243–14253页。

[7] S. Boyd 和 L. Vandenberghe (2010) 《凸优化》，剑桥大学出版社，纽约州纽约市，美国。[8] E. W. Cheney 和 H. L. Loeb (1964) 广义有理逼近，《工业与应用数学学会杂志》B辑：数值分析，工业与应用数学学会，第1卷第1期，11–25页。[9] J. P. Crouzeix (1980) 拟凸函数的凸性条件，《运筹学数学》，第5卷第1期，120–125页。[10]G. Cybenko (1989) *S*形函数叠加的近似，《控制、信号与系统数学》，第2卷第4期，303–314页。[11] I. Goodfellow、Y. Bengio 和 A. Courville (2016) 深度学习，麻省理工学院出版社。[12] B. D. Haeffele 和 Ren´e Vidal (2017) 神经网络训练的全局最优性，2017年IEEE计算机视觉与模式识别会议（CVPR），美国夏威夷州檀香山，2017年，第4390–4398页，doi: 10.1109/CVPR.2017.467。[13] C. Hofreither (2021)基于重心有理插值的最佳有理逼近算法，《数值算法》，第88卷第1期，365–388页。[14] K. Hornik (1991) 多层前馈网络的逼近能力，《神经网络》，第4卷第2期，251–257页。[15] A. N. Kolmogorov (1957)关于通过单变量连续函数的叠加与加法表示多变量连续函数，《苏联科学院通报》，第114卷第5期，953–956页。[16] C. M. Lee 和 F. D. K. Roberts (1973) 有理逼近算法的比较 l_∞，《计算数学》，第27卷第121期，111–121页。[17] M. Leshno、V. Ya. Lin、A. Pinkus 和 S. Schocken (1993) 具有非多项式激活函数的多层前馈网络能够逼近任意函数，《神经网络》，第6卷第6期，861–867页。[18] H. L. Loeb (1960) 利用线性形式之比进行切比雪夫逼近的算法，《工业与应用数学学会杂志》，工业与应用数学学会，第8卷第3期，458–465页。[19] H. L. Loeb (1957) 离散点上的有理分式逼近，*Convair Astronautics*，数学，The Computer Journal，第9卷。

[20] J. E. Mart´ınez-Legaz（1988）基于广义共轭方法的拟凸对偶理论，《Optimization》，Taylor and Francis，第19卷（第5期），603– 652。[21] G. Meinardus（1967）函数逼近：理论与数值方法，Springer-Verlag，柏林。[22] A. Molina、P. Schramowski 和 K. Kersting（2019）填´充激活单元：在深度网络中端到端学习灵活的激活函数，载于国际学习表征会议（ICLR）。[23] Y. Nakatsukasa、O. Sete 和 L. N. Trefethen（2018）用于有理逼近的 *AAA* 算法，《SIAM Journal on Scientific Computing》，第40卷（第3期），A1494–A1522。[24] J. X. Da Cruz Neto、J. O. Lopes 和 M. V. Travaglia（2011）拟凸最小化算法，《Optimization》，Taylor and Francis，第60卷（第8–9期），1105–1117。[25] M. R. Osborne 和 G. A. Watson（1969）非线性情形下极小极大逼近的算法，《The Computer Journal》，第12卷（第1期），63–68。[26]V. Peiris、N. Sharon、N. Sukhorukova 和 J. Ugon（2021）广义有理逼近及其在改进深度学习分类器中的应用，《Applied Mathematics and Computation》，Elsevier，第389卷，125560。[27] V. Peiris（2022）基于一致范数损失函数的神经网络有理激活函数及其在分类中的应用，《Communications in Optimization Theory》，文章3。[28] P. Petrushev 和 V. Popov（1987）实函数的有理逼近，剑桥大学出版社。[29] A. Pinkus（1999）神经网络中多层感知机模型的逼近理论，《Acta Numerica》，剑桥大学出版社，第8卷，143–195。[30] R. Anthony（1965）基于 *Remes* 算法的有理切比雪夫逼近，《Numerische Mathematik》，Springer-Verlag New York, Inc.，美国新泽西州锡考克斯，第7卷（第4期），322–330。[31] T. J. Rivlin（1962）某些有理函数的最佳一致逼近多项式，《Numerische Mathematik》，Springer，第4卷（第1期），345–349。[32] N. Sharon、V. Peiris、N. Sukhorukova 和 J. Ugon（2021）灵活的有理逼近及其在矩阵函数中的应用，arXiv 预印本 arXiv:2108.09357。

训练轮次	最终损失	最小损失	运行时间（每个周期）
50 44	0.045868	 0.045840	2.51s ± 103ms
100 63	0.045864	 0.045840	2.7s ± 71.9ms
200 114	0.045934	 0.045839	2.77秒 ± 453毫秒

表 13：结果：实验集1

[33] R. -Yu. Sun（2020）深度学习优化：综述，《中国运筹学会学报》，Springer，第8卷（第2期），249–294。

[34] M. Telgarsky（2017）神经网络与有理函数，《国际机器学习会议》，PMLR，3387–3393。

[35] A. Alimov 和 I. Tsar'kov（2011）几何逼近理论，Springer Cham。

A 附录

A.1 结果：基于神经网络的近似。

A.1.1 带 ReLU 激活的神经网络

集合1：隐藏层中有 2 个节点、均方误差损失、ADAM优化器、ReLU 激活
表 13 总结了隐藏层中有 2 个节点时的结果（神经网络、均方误差损失、ADAM）。可以看出，随着训练轮次增加，损失函数的值并未显著改善。此外，损失函数报告的最小值似乎可能出现在最后一个训练轮次之前。例如，在训练轮次为 200 的情况下，最小值（0.045839）出现在第 114 轮，而报告的值（训练 200 轮后）为 0.045934。总体而言，当训练轮次从 50 增加到 200 时，每轮的计算时间并未显著增加。本表最后一列报告了每轮的运行时间，因此计算时间几乎与训练轮次成正比。因此，增加训练轮次会按比例增加总计算时间，却无法显著提高准确率。

图 20 展示了函数 $f(x)$ 以及本次实验得到的近似结果：训练轮次分别为 50、100 和 200（集合1）。图中可以看出，增加训练轮次并未显著提高近似精度，这与表 13 的结果一致：不同训练轮次下的损失函数值相近。

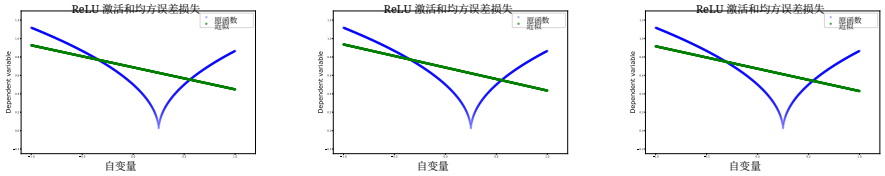

图 20：集合1：近似由具有 ReLU 激活的神经网络计算，训练轮次为 50、100 和 200。

训练轮次	最终损失	最小损失	运行时间（每个周期）
50 13	0.492186	 0.469656	2.92s ± 707ms
100 83	0.505962	 0.467289	2.94s ± 535ms
200 83	0.503380	 0.467289	2.84s ± 314ms

表14：结果：实验集2

集合2：隐藏层中有 2 个节点，均匀损失，ADAMAX 优化器，ReLU 激活
在这组实验中，我们使用均匀损失函数。这是它与集合1实验的主要区别。我们还使用 ADAMAX 优化器代替 ADAM，因为 ADAMAX 专为均匀损失函数开发。

由于集合1和集合2对应的不是准确度量，因此比较它们的损失函数值没有意义。比较集合2中不同训练轮次数的结果时，可以看到与集合1实验中类似的模式：计算时间大致与训练轮次数成正比；随着训练轮次数增加，损失函数值并未显著改善，且最优损失不会随着训练轮次数单调下降。有趣的是，记录到的最佳损失函数值0.467289是在训练83轮后达到的，与仅训练13轮后达到的0.469656相差不大。总的来说，增加训练轮次数会导致计算时间增加，却无法显著改善最优损失值。

图 21 展示了本次实验中得到的函数 $f(x)$及其近似结果：50、100和200个训练轮次（集合2）。图中结果证实，增加训练轮次数并不能显著提高近似精度，这与表14的结果一致，因为不同训练轮次数对应的损失函数值相近。

比较集合1和集合2中的实验结果，尚不清楚哪种损失函数（均方误差还是均匀损失）更合适，但可以确定的是，隐藏层中的节点数量不足以获得准确的结果。

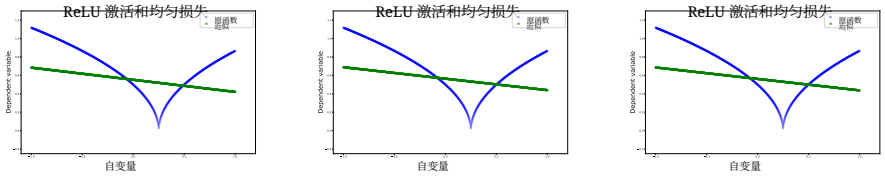

图 21：集合2：近似由采用 ReLU 激活的神经网络计算，训练轮次为 50、100 和 200。

训练轮次	最终损失	最小损失	运行时间（每个周期）
50 50	0.001848	0.001848	2.89s ± 136ms
100 99	0.000915	0.000908	2.5秒 ± 59.2毫秒
200 141	0.000914	0.000903	2.53秒 ± 264毫秒

表15：结果：实验集3

集合3：隐藏层中有10个节点，均方误差损失、ADAM优化器、ReLU 激活 我们对隐藏层中有2个节点的情况进行的研究表明，需要增加节点数量。在这组实验中，隐藏层中有10个节点。

表15总结了集合3实验的结果。损失函数值显著优于集合1实验中隐藏层只有2个节点时的结果。仍可观察到最优损失函数值存在一定的非单调性，但不如集合1实验中明显。根据这些结果，训练轮次保持在100至150之间是合理的。

图 22展示了集合3实验的结果。图中清楚地表明，增加隐藏层中的节点数量提高了近似精度，并且在训练100或200轮次时，准确率要高得多。在“困难点” $x = 0.25$附近，$f(x)$不光滑且非利普希茨，因此即使训练轮次更多，误差仍然显著。另一个值得注意的现象是，尽管隐藏层中的节点数量有所增加，集合1和集合3的计算时间仍非常相近。

A.1.2 使用有理近似逼近 ReLU 激活函数的神经网络

本节展示数值实验的结果。实验设置与 A.1.1 节类似，但激活函数采用 ReLU 的有理逼近。这种近似是通过

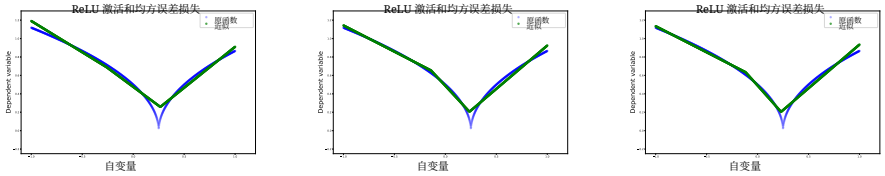

图 22：集合3：采用具有 ReLU 激活的神经网络进行近似，训练轮次为 50、100 和 200。

训练轮次	最终损失	最小损失	运行时间（每个周期）
50 50	0.004621	 0.004621	2.85 秒 ± 83.5 毫秒
100 100	0.002096	 0.002096	2.94秒 ± 576毫秒
200 199	0.000578	 0.000550	2.87秒 ± 105毫秒

表16：结果：实验集1，ReLU 的有理逼近

微分修正方法得到的 d.

我们的激活函数是一个次数为 (3,2) 的有理函数：该有理逼近是两个多项式的比值，分子的次数为 3，分母的次数为 2。

有理激活函数的系数来自对 ReLU 函数的最佳 (3,2) 近似。这些系数在整个训练过程中保持固定。我们不会像网络的其他参数那样学习这些系数。

集合1：隐藏层中有 2 个节点，均方误差损失，ADAM优化器，ReLU 的有理逼近作为激活函数 表16总结了均方误差损失、隐藏层中有 2 个节点且使用有理逼近作为激活函数时的结果。结果清楚地表明，这种方法优于在隐藏层中有 2 个节点时直接使用 ReLU 作为激活函数。其结果与隐藏层中有 10 个节点时直接使用 ReLU 作为激活函数的结果相当。

图23证实，当隐藏层中只有2个节点时，将 ReLU 的有理逼近用作激活函数，比直接使用 ReLU 更准确。“困难点”仍然是一个挑战。

集合2：隐藏层中有2个节点，采用均匀损失和 ADAMAX 优化器，ReLU 的有理逼近作为激活函数表17总结了均匀损失、隐藏层中有2个节点的情况下

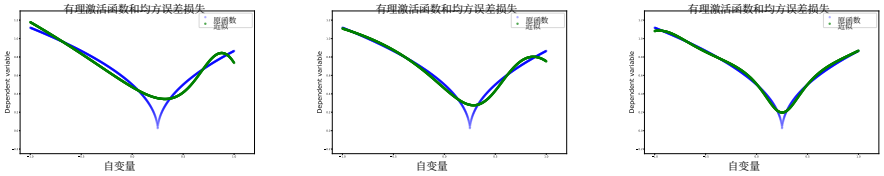

图23：集合1：使用带有 ReLU 的有理逼近的神经网络计算近似，训练轮次为50、100 和 200。

训练轮次	最终损失	最小损失	运行时间（每个周期）
50 37	0.211703	 0.204005	2.88s ± 726ms
100 87	0.212665	 0.170603	2.8秒 ± 125毫秒
200 197	0.166955	 0.124133	2.84秒 ± 337毫秒

表17：结果：实验集2，ReLU 的有理逼近。

以及激活函数的有理逼近的结果。与均方误差损失类似，结果清楚地表明，这种方法优于在隐藏层中有2个节点时直接使用 ReLU 作为激活函数的方法。其结果与隐藏层中有10个节点时直接使用 ReLU 作为激活函数的结果相当。

集合3：隐藏层中有10个节点，均方误差损失，ADAM优化器，ReLU 的有理逼近是激活函数 表18显示，当隐藏层中的节点数量从2增加到10时，准确率进一步提高。当训练轮次为200时，这一现象尤为明显。

图25显示，即使是在“困难点”处，近似的准确率也在不断提高，尤其是在训练轮次为 200 时。

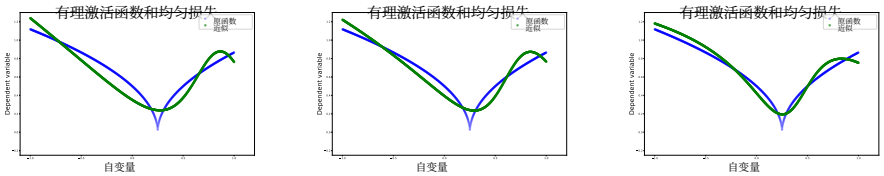

图24：集合2：由具有有理激活函数的神经网络计算近似，训练轮次为50、100和200。

训练轮次	最终损失	最小损失	运行时间（每个周期）
50 50	0.000885	 0.000885	2.78s ± 93.5ms
100 100	0.000153	 0.000153	2.85s ± 539ms
200 190	0.000046	 0.000046	2.79秒 ± 93.1毫秒

表18：结果：实验集3，ReLU 的有理逼近

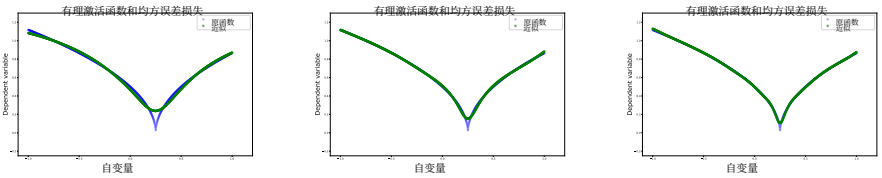

图25：集合3：使用具有有理激活函数的神经网络对 ReLU 进行近似，训练轮次为50、100和200。

A.1.3 具有有理激活函数的神经网络

在这种情况下，我们的激活函数是一个次数为 (3，2) 的有理函数。有理激活函数的系数现在是参数集的一部分。在训练过程中，我们会像学习其他参数一样学习这些系数。

集合1：隐藏层包含2个节点，均方误差损失，ADAM优化器，有理激活函数 表19显示，损失函数的取值优于使用 ReLU 的情况，与使用 ReLU 的有理逼近所得结果相当（ReLU 的有理逼近精度略高）。图26证实了这一观察结果。

有趣的是，每个训练轮次的计算时间有所增加，但总运行时间与 ReLU 以及通过有理函数逼近的 ReLU 相近。

集合2：隐藏层中有2个节点，均匀损失，ADAMAX优化器，有理激活函数 这种情况与均方误差损失类似。表 20 显示，损失函数的数值优于使用 ReLU 时的情况，并且与使用 ReLU 的有理逼近时的结果相当。图27证实了这一观察结果。

集合3：隐藏层中有10个节点，均方误差损失，ADAM优化器，有理激活函数 表21显示，当数量

训练轮次	最终损失	最小损失	运行时间（每个周期）
50 50	0.005893	0.005893	2.54s ± 111ms
100 100	0.001330	0.001330	2.81s ± 339ms
200 170	0.000773	0.000745	3.05秒 ± 433毫秒

表19：结果：实验集1，有理激活函数

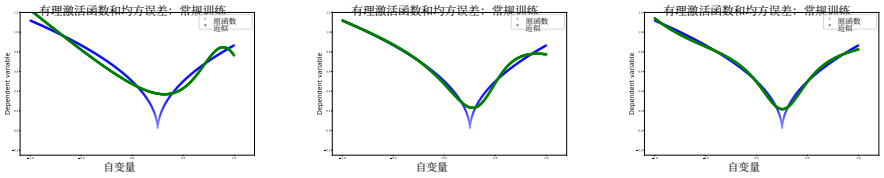

图26：近似通过常规训练过程计算，训练轮次为 50、100 和 200。

训练轮次	最终损失	最小损失	运行时间（每个周期）
50 43	0.219781	0.206314	2.57s ± 616ms
100 100	0.183087	0.183087	2.84s ± 82.3ms
200 198	0.151559	0.123221	2.87s ± 85.4ms

表20：结果：实验集2，有理激活函数

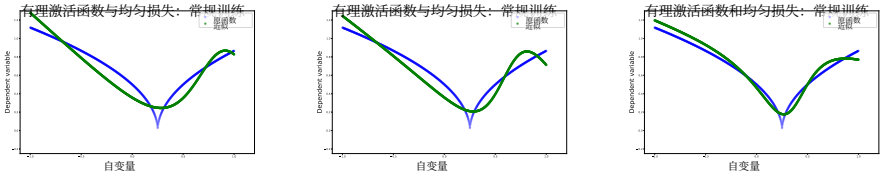

图27：近似通过常规训练过程计算，训练轮次为50、100和200。

训练轮次	最终损失	最小损失	运行时间（每个周期）
50 50	0.000935	 0.000935	2.91秒 ± 711毫秒
100 100	0.000190	 0.000190	2.76秒 ± 91.6毫秒
200 198	0.000070	 0.000043	2.52秒 ± 295毫秒

表21：结果：实验集3，有理激活函数

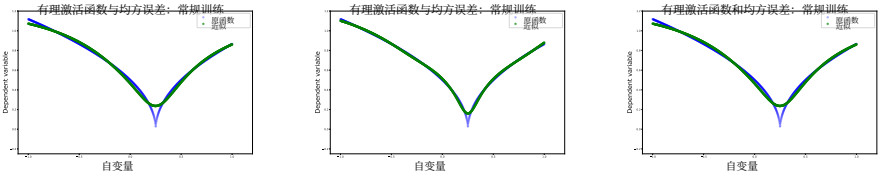

图28：使用常规训练过程计算近似结果，训练轮次分别为50、100和200。

将隐藏层中的节点数增加到10后，最优损失函数值与使用有理函数近似ReLU时的情况非常接近。图28证实了这些结果。值得注意的是，增加隐藏层中的节点数并不会显著增加计算时间。

当训练轮次为200时，即使在“困难点”附近，近似结果看起来也很准确。

A.1.4 使用有理激活函数的神经网络，采用分割训练方法进行训练

本节中，我们的激活函数是一个次数为 (3,2) 的有理函数。系数现在是参数集的一部分。我们在训练过程中使用分割方法学习这些系数，同时学习其他参数。

集合1：隐藏层中有 2 个节点，均方误差损失，ADAM优化器，分割方法
表22显示，从最优损失函数值及相应计算时间来看，结果与上一节中使用ReLU 的有理逼近所得结果相似。计算时间也相近。随着训练轮次的增加，结果有所改善。图29证实了这一相似性。

集合2：隐藏层有2个节点，均匀损失，ADAMAX 优化器，分割方法。隐藏层只有2个节点时的结果

训练轮次	最终损失	最小损失	运行时间（每个周期）
50 50	0.005291	0.005291	2.51秒 ± 70.5毫秒
100 100	0.002445	0.002445	2.65秒 ± 448毫秒
200 197	0.000602	0.000577	3.39秒 ± 585毫秒

表22：结果：实验集1，分割方法

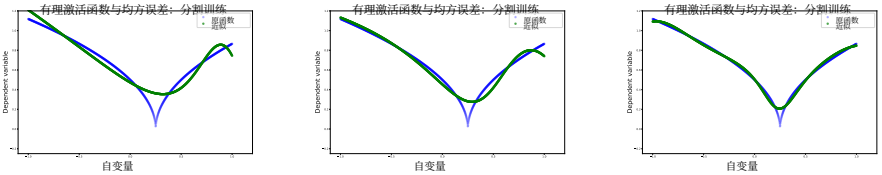

图29：通过分割训练过程计算近似，训练轮次为50、100和200。

就最优损失函数值而言，分割方法的结果优于使用有理激活函数或 ReLU 激活的有理逼近的情况（表23）。分割方法的计算时间略长，但差异极小。随着训练轮次增加，每个训练轮次的平均计算时间保持不变。

图30显示，逼近误差大多出现在“困难点”处。

集合3：隐藏层中有10个节点，均方误差损失，ADAM优化器，分割方法
当隐藏层中的节点数量增加到10个时，以最优损失函数值衡量的近似精度有所提高，但结果不如采用有理激活函数和 ReLU 激活的有理逼近时那样好

训练轮次	最终损失	最小损失	运行时间（每个周期）
50 43	0.196001	0.144596	3.02s ± 700ms
100 78	0.144665	0.106687	3.07s ± 549ms
200 145	0.122333	0.085358	3.03s ± 110ms

表23：结果：实验集2，分割方法。

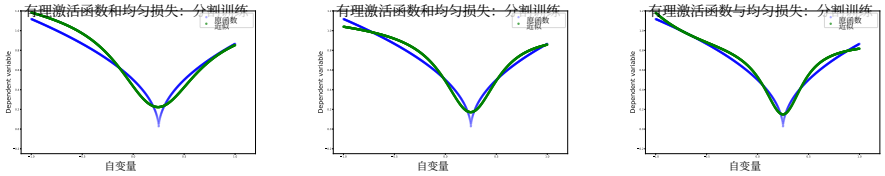

图30：通过分割训练过程计算近似，训练轮次为 50、100 和 200。

训练轮次	最终损失	最小损失	运行时间（每个周期）
50 50	0.000318	 0.000318	2.68秒 ± 131毫秒
100 100	0.000107	 0.000107	2.59秒 ± 79毫秒
200 143	0.011278	 0.000064	2.65秒 ± 305毫秒

表24：结果：实验集3，分割方法

（表24）。

图31显示，当训练轮次为200时会出现一些振荡。不过，如果恰好在第143轮停止训练过程，则可以获得更好的近似（图32）。

A.2 直接有理逼近方法：微分修正和 AAA 方法。

在本节中，所有实验均对应均匀损失。我们比较了构造有理近似的两种主要方法：微分修正方法和 AAA 方法。

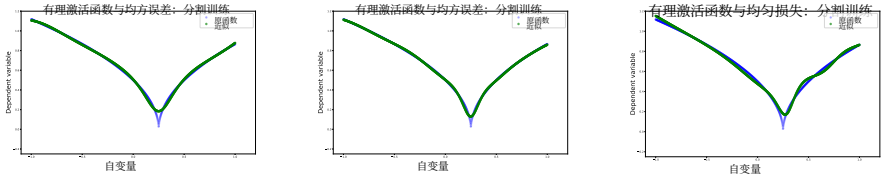

图31：通过分割训练过程计算得到的近似，训练轮次为50、100和200。

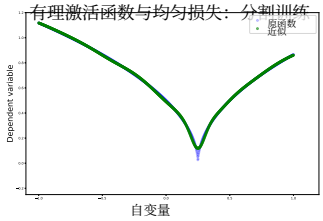

图32：通过分割训练过程进行143个训练轮次后得到的近似。

次数	误差（均匀范数）	运行时间
(5,4)	0.05598587273362793	2.22s
(4,4)	0.08595288703758228	1.73s
(5,5)	0.03475461848979777	2.59秒

表 25：次数为 (5，4)、(4，4) 和 (5，5) 的直接有理逼近结果

A.2.1 微分修正方法

次数为 (5,4) 的有理逼近，微分修正方法。当考虑一个三层神经网络，其隐藏层只有 2 个节点，并采用次数为 (3，2) 的有理激活函数时，输出是一个次数为 (5，4) 的有理函数。因此，我们可以使用微分修正方法，以次数为 (5，4) 的有理函数近似函数 $f(x) = \sqrt{|x - 0.25|}$，并将此近似与均匀损失函数及隐藏层有 2 个节点时得到的结果进行比较。理论上，直接用次数为 (5，4) 的有理函数进行近似具有更大的灵活性。我们在实验中使用 Python。我们还计算了次数为 (4，4) 和 (5，5) 的有理近似以作比较。我们还将使用这些额外结果与 AAA 方法进行比较，因为该方法要求分子和分母的次数相同。表 25 总结了计算时间和最优损失。结果表明，在这三种设置下，计算时间都很短，且相应的误差非常接近。有趣的是，次数增加并不总能改善最优损失函数值。这可以归因于随着相应优化问题的维度增加而产生的计算不稳定性。

图33展示了函数以及相应的次数为 (4,4)、(5,4) 和 (5,5) 的有理逼近。

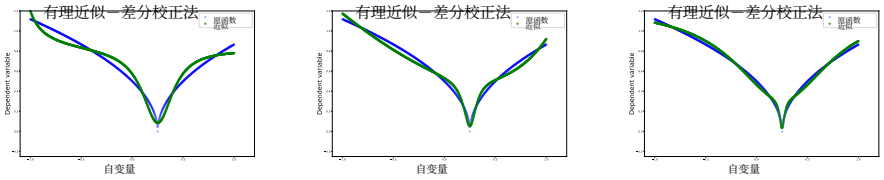

图33：采用微分校正法进行近似，有理逼近次数分别为 (4,4)、(5,4) 和 (5,5)。

次数	误差类型	误差	运行时间
(4,4)	一致误差	1707.081229434771	53.1毫秒
	MSE	2658.9489333999118	
(5,5)	一致误差	0.14229129790010392	85.5毫秒
	MSE	1.1112525485550502	

表26：结果：AAA 方法，次数 (4,4) 和 (5,5)

A.2.2 AAA 方法

次数为 (5,4) 的有理逼近：AAA 方法 为了将微分修正方法与 AAA 进行比较，我们在与微分修正方法相同的实验设置下应用 AAA 方法。由于 AAA 方法的局限性，分子和分母无法采用不同的次数。因此，在这些实验中，我们使用次数为 (4,4) 和 (5,5) 的有理逼近。代码由 C. Hofreither 使用 Python 编写 [13]。

表26总结了 AAA 方法的结果。AAA 方法旨在优化均匀损失，但为了便于比较，我们也报告了相应的均方误差损失。可以看出，最优损失较高，这是由不稳定性造成的。遗憾的是，限制条件数（微分修正）无法解决这种不稳定性。

图34突显了近似中的困难。在次数为（4,4）的情况下，这些困难尤为突出。

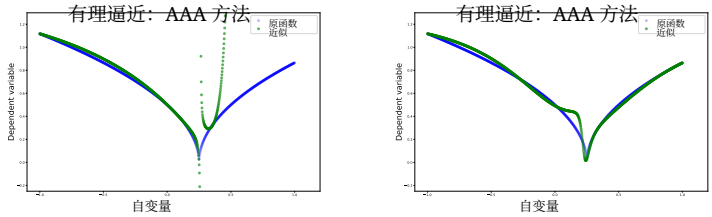

图34：近似由 AAA 方法计算，次数为 (4,4) 和 (5,5)。