EvoAAA：一种用于自动神经自编码器架构搜索的进化方法

Francisco Charte* Antonio J. Rivera Francisco Mart´ínez Mar´ía J.del Jesus

2023年1月18日

arXiv:2301.06047v1 [cs.NE] 15 Jan 2023

摘要

如果向机器学习模型提供筛选后的特征，其性能会更好。特征工程方法通常被用作预处理步骤，以获取或构建适当的特征集。近年来，自编码器（一种特定类型的对称神经网络）被广泛用于执行表征学习，证明了其相较于经典特征工程算法的竞争力。使用自编码器的主要障碍在于寻找良好的架构，而这一过程大多由专家手动完成。本文提出了一种基于进化方法的自动自编码器架构搜索过程。该方法在九个异构数据集上进行了测试。所得结果表明，这种方法能够找到更优的架构，在更短的时间内通过最小化的编码集中大部分有用信息。

1 引言

I基于机器学习系统的智能设备 [1] 在日常生活的许多任务中都能见到。它们负责
fi过滤垃圾邮件 [2], 检测欺诈交易 [3], 向买家推荐新产品 [4] 以及许多其他
a看似简单的工作。为此，机器学习方法需要从原始数据中提取知识。这些知识的效用
m主要取决于数据特征的质量。目标是根据具体任务，生成描述性和/或预测模型，
t以满足当前任务的需要。

选择一组经过筛选的属性，或根据原始特征构建一组新的属性，往往能带来更好的结果 [5]
t使用原始变量。因此，近年来人们对特征工程技术愈发关注，其中包括广为人知的预处理
p流程 [6]，例如特征选择 [7, 8] 和特征提取 [9]。表征学习（REPL） [10, 11] 是一个
t与运用深度学习技术进行特征工程密切相关 [12, 13]。

自编码器 [14, 15, 16]是一类基于深度学习的现代通用工具，可用于表征学习。自编码器是一种无监
p督对称神经网络 [17] ，旨在构建一种编码，使其最大限度地重构数据模式。所
o获得的编码可用于许多不同任务 [18], ，包括可视化、异常检测、哈希和噪声
r移除。然而，为每个数据集和函数找到合适的自编码器架构并非易事。通常，这是一项
a 需要专家应对的挑战。

本研究提出了 EvoAAA，这是一种用于设计自编码器架构、以最大限度提升其重建能力的自动化方法
w在针对特定数据集使用时。该任务被视为一个难以通过穷举搜索解决的优化问题 [19],
s。我们的假设是，进化方法 [20] 能够在合理的时间内找到近乎最优的
A自编码器架构。

为验证 EvoAAA 的竞争力，我们开展了全面的实验，涵盖九个数据集和五种搜索方法，其中三种基于进化优化。通过这种方法找到的架构远优于穷举搜索得到的架构。

1.1 问题表述

自编码器是 REPL 的有效工具，这一点已广为人知 [14, 15, 16], ；已有研究证明，它们优于主成分分析、线性判别分析、等距映射和局部线性嵌入等经典特征工程方法 [21]。它们也是与非标准学习 [22] 问题密切相关的工具。自编码器是一种对称的人工神经网络 [17], , 因此具有人工神经网络的许多特性。

训练人工神经网络意味着使用反向传播方法 [23]以及随机梯度下降等梯度下降算法的变体 [24]，调整连接神经元的权重。人工神经网络的架构，即层数、每层单元数、激活函数等，都要在训练过程开始前预先设定。

不恰当的人工神经网络架构可能会导致糟糕的输出结果，无论训练过程中学到的权重如何。如果人工神经网络过于简单，可调整的参数太少，就无法学习。相反，如果过于复杂

*通讯作者：Francisco Charte，计算机科学系，A3-241 哈恩大学´，拉斯拉古尼亚斯校区，23071 哈恩´，
西班牙。

表1：术语及缩略语汇总 yms

术语/缩略语	描述
AE	自编码器
ANN	人工神经网络
染色体	表示种群中一个个体的参数集合的编码
CNN	卷积神经网络
交叉	通过混合两个或多个个体（父代）的基因来生成新染色体的算子
DE	差分进化
DL	深度学习
精英策略	在进化方法中保留每一代最优个体的技术
EM	进化方法
ES	进化策略
EvoAAA	用于自动编码器架构搜索的进化方法
适应度	与每个个体相关联的质量度量
GA	遗传算法
Gene	染色体中的每个值
代	进化方法中的每一次迭代
个体	搜索空间中的一个潜在解，由染色体及其对应的适应度值表示
LDA	线性判别分析
LLE	局部线性嵌入
ML	机器学习
MLP	多层感知机
MSE	均方误差
变异	通过改变现有个体中的基因来生成新染色体的算子
PCA	主成分分析
种群	一代中的个体群体
REPL	表征学习
SGD	随机梯度下降

人工神经网络会因拥有足够多的参数来记忆全部训练数据而出现过拟合 [25] 。这些问题同样会影响自编码器。如何实现平衡的架构——使人工神经网络能够从已见数据中提取足够的信息，从而在处理未来从未见过的模式时实现良好的泛化——仍是一个尚未解决的问题。因此，每年都会发表数十篇致力于设计人工神经网络以解决特定问题的论文 [26, 27] 。

我们关注的是为特定数据集选择合适的自编码器架构，以便自编码器能够学习数据的最优表征。迄今为止，自编码器的设计一直由人类专家负责。这项任务并不容易自动化，因为它是一个复杂的组合问题 [19]。事实上，寻找良好的架构可以看作一个优化问题。进化方法 [20] 是一个领域，而进化方法过去已在其中展现了成效。

我们提出了一种编码方案，可对任意自编码器架构进行编码，使其能够通过进化方法进行演化。目标是尽可能降低重构误差。这样，自编码器编码将尽可能集中通用信息，而不是采用旨在提高类别可分性或实现其他特定目标的编码方式。至于进化技术，它们将用作优化一组参数的工具。本文将采用该领域的经典术语，这些术语及文中出现的缩写汇总于表1。

1.2 文献综述

特征工程是一项手动或自动化任务，旨在获得一组优于原始特征的特征。特征选择 [6] 是指在保留数据中大部分有用信息的同时，选择属性的一个子集。这可以由该领域的专家手动完成，但大多数情况下会采用基于特征相关性 [7]和互信息 [8]的自动化方法。相比之下，特征提取方法会变换原始数据特征，以生成一组新的、通常更精简的属性。用于此目的的常见算法包括 PCA [28] 和 LDA [29], ，其数学基础相对容易理解。

更深入的研究以这样一个假设为基础：原始数据中变量的分布位于一个较低维度的空间中，这个空间通常称为流形。流形空间使用生成原始高维空间中数据点的参数。寻找这一嵌入空间是流形学习 [30] 算法的任务。与 PCA 或 LDA 不同，流形方法采用非线性变换，因此属于非线性降维 [31]类别。

如 [17], 所述，自编码器是一种具有对称架构的人工神经网络，如图1所示。输入层和输出层的单元数与数据中的特征数相同。内部层通常包含较少的单元，从而生成数据所含信息的更紧凑表征。其目标是尽可能忠实地将输入模式重构到输出中。

尽管自编码器主要用于执行特征融合 [17], ，以搜索可找到用于重建数据的参数的流形，但它们还有许多其他实际应用 [18]。配置得当的自编码器可以将任意

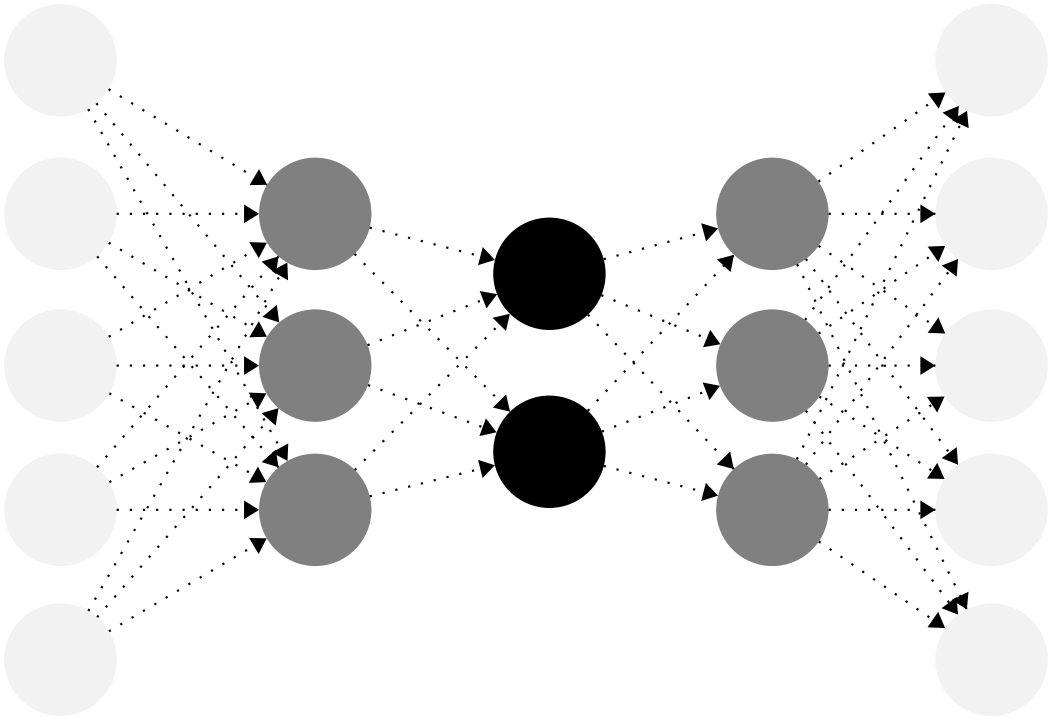

图1：自编码器的经典架构。黑色节点表示二维编码层。深灰色节点表示中间隐藏层。浅灰色节点表示输入层（左侧）和输出层。

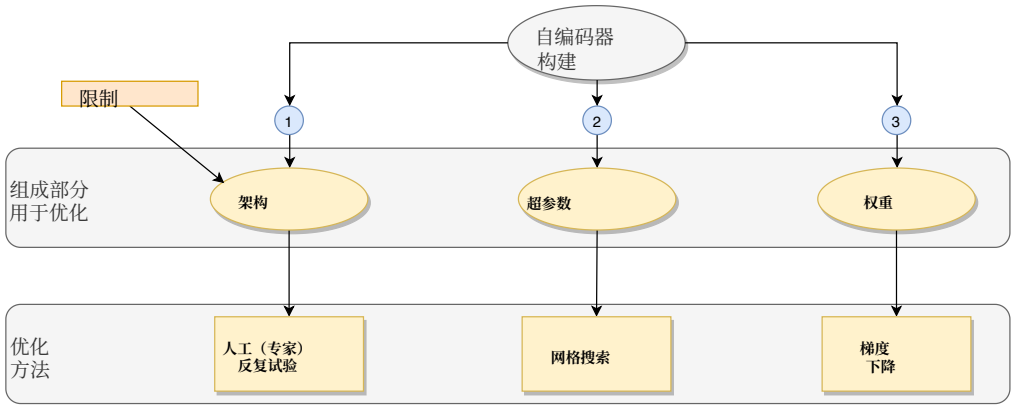

图2：构建自编码器时需要优化的组成部分及所采用的方法。圆圈中的数字表示通常的优化顺序：首先设定架构，然后选择一组超参数，最后调整权重。

维度的数据投影到 2 维或 3 维，以便以图形方式可视化模式 [32]。自编码器可用于检测异常 [33, 34], ，方法是训练它们忠实地重建正常模式。当异常数据进入自编码器时，它会产生较高的输出误差，表明这些模式不符合已知分布。自编码器的另一种有趣应用，特别是去噪自编码器的应用，是去除数据噪声。这类自编码器已成功用于图像 [35]和语音 [36]去噪。通常，需要根据所要处理的具体任务调整自编码器的损失函数，使获得的编码能够提升可分性、保持拓扑，或具备其他期望的特性。如果只追求重建输入模式时的最高性能，自编码器将在其内部层中生成通用编码。

自编码器可以设置不同数量的内部层，每层的长度也各不相同。合适的架构主要取决于待重构模式的复杂度，以及编码层所受到的限制。这些限制可防止自编码器直接将输入复制到输出 [17], ，例如通过减少编码层中的神经元数量，强制大多数神经元的输出为零（稀疏表示）等。

为机器学习模型寻找最佳参数是一项艰巨的任务。通过内部验证过程进行网格搜索是一种常见方法。然而，它仅适用于参数集有限且取值范围已知的情况。可采用各种搜索算法、元启发式算法 [37] 和旨在执行组合优化的优化方法 [38],

。这些方法从相对简单的搜索方法 [39, 40, 41, 42] （如 A* 或 IDA）到更先进、复杂的方法（如模因算法 [43], ），其中包括模拟退火 [44] 或禁忌搜索 [45]等经典方法。进化方法 [46, 47] 长期以来也被用于解决优化问题 [48, 49, 50, 51] 。其中许多算法基于特定种群的行为，例如蚁群 [52] 和粒子群 [53]。下文将重点介绍基于自然进化 [54]的方法。

在进化算法中， [55] 搜索空间由染色体定义。染色体由多个基因组成，每个基因代表一种特定的特征或维度。染色体中基因的取值范围有限，每种取值组合都可能构成一个解（搜索空间中的一个点）。这些解也称为个体。通常会随机生成一组个体，形成初始种群。每个个体都会获得一个适应度值，用以表示该解的优劣。从这一步开始，算法会进行多次迭代，每次迭代都包含相同的步骤。首先，根据适应度选择种群中的部分个体进行繁殖。然后，对选中的个体应用一组算子，以生成新的个体。常用算子包括交叉，即混合两个个体的基因以产生新的染色体；以及变异，即随机改变一个或多个基因的取值。最后，通过计算新的适应度值来评估整个种群并更新种群，通常会用新个体替换旧个体，不过也可能保留总体表现最佳的解（精英策略），无论这些解是否为新解。三种常见的进化方法是遗传算法 [56], 、差分进化 [57] 和进化策略 [58]。第一种方法遵循上述选择、交叉、变异和评估流程。尽管遗传算法可被视为一种较老的方法，但由于其简单且性能良好，至今仍被广泛使用 [38, 50, 59] 。第二种方法通过现有个体之间的差异生成新个体，而第三种方法则着重于仅使用变异算子作为唯一工具来进化少数个体。

进化方法也适用于组合优化，例如已被用于支持向量机 [60] ，近年来还被用于深度学习网络 [61]。尽管进化方法早已用于优化人工神经网络，但许多研究方案都侧重于学习与每个连接相关联的权重。这被称为传统神经进化 [62, 63, 64, 65] ，其主要基础是使用进化方法而非传统的梯度下降算法来优化权重。因此，固定的网络架构是整个种群的基础。个体之间的唯一差异在于连接各层的权重矩阵，这些权重矩阵的值由进化方法进行优化。后来出现了能够同时进化网络拓扑的神经进化算法 [66], ，其中大多数旨在优化多层感知机。构建人工神经网络架构的方法多种多样，其中最常见的方法之一是子网络组合 [67]。关于这一主题的最新综述可见 [68]。

近来，为了全面自动化选择合适的模型结构、调整超参数和权重等流程，人们将不同技术融合起来，由此催生了一个名为自动化机器学习（AutoML） [69, 70, 71]的新领域。自动化机器学习工具可以基于进化方法，也可以基于贝叶斯技术和其他优化算法。现有的自动化机器学习工具大多旨在辅助设计多层感知机和卷积神经网络 [72], ，它们或许是最受欢迎的两类人工神经网络。这些工具本质上拥有数量有限的单元或模块，可将它们组合起来定义人工神经网络拓扑结构。针对不同任务，会采用特定策略来选择这些模块。例如，AutoKeras 工具 [73]定义了一些任务，能够自动设计用于图像、文本和结构化数据分类的人工神经网络。人工神经网络架构由预定义模块构成，例如图像模块、文本模块或结构化数据模块，并带有一些可调整的参数。因此，这些自动化机器学习工具的粒度较粗，只能用于构建某些特定类型的人工神经网络。

相比之下，在本文提出的方法中，进化方法用于演化自编码器的架构。自编码器是一类人工神经网络，具有一些特定特点，例如对称布局或无监督学习方式。如下文所述，自编码器布局以较细的粒度生成，而不是使用预定义的模块。权重通过通常的反向传播算法进行学习。本文提出的 EvoAAA 过程可以作为自动化机器学习流程中的一个环节。

本文其余部分的结构如下：第2节概述了构建自编码器时需要优化的不同方面，并介绍了所提出的架构搜索方法。第3节描述实验框架，分析性能结果并进行讨论。最后，在第4节给出结论。

2 使用 EvoAAA 进行自编码器架构搜索

本节介绍针对第1.1小节所提出问题的解决方案，概述构建自编码器时需要考虑的各个方面，并详细说明所采用的方法。

2.1 构建自编码器时需要优化的组成部分

构建新的自编码器并对其进行调整，以便从输入模式中生成良好的表征，这一过程包含多个步骤。每个步骤都与某个组成部分的优化相关。图2展示了该流程、组成部分和方法的概要。各组成部分按以下顺序进行调整：

1. **架构：** 必须设定自编码器的结构，确定其包含的层数、每层单元数、各单元所使用的激活函数等。多年来，这项工作一直由专家手动完成。传统的试错过程也是一种常见方法，即在完成三个构建步骤并评估自编码器性能后，重新调整自编码器架构

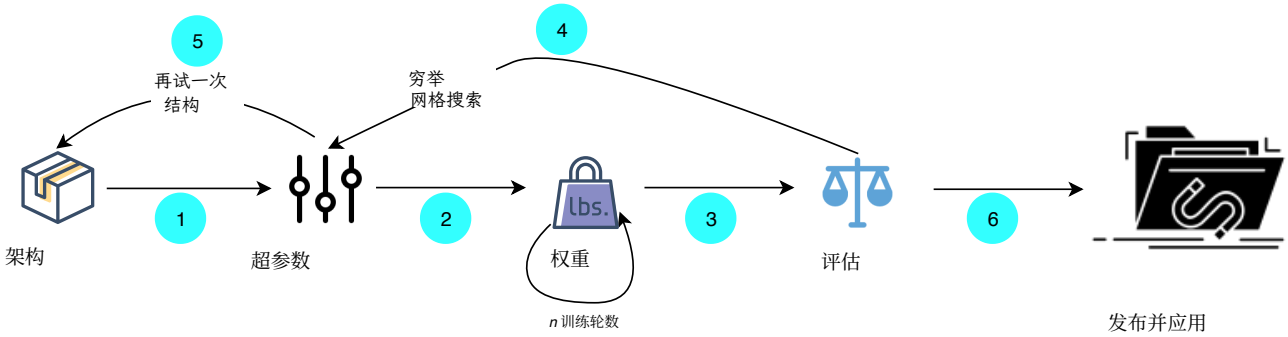

图3：在应用自编码器之前，必须使用多种超参数组合测试每种架构，并调整每种配置对应的权重。

。在此步骤中，必须考虑特定限制，例如网络的对称结构。

2. **超参数：** 确定自编码器的设计后，下一步是为若干超参数选择合适的取值。这些参数既不属于网络结构，也不属于权重配置。它们用于控制训练过程中权重的调整。最常见的超参数包括学习率、批量大小以及网络的训练轮数。虽然可以手动选择这些取值，但通常会使用网格搜索算法。再次通过试错，并将部分训练样本用作验证数据，即可找到最佳配置。

3. **权重：** 确定网络架构及其超参数后，最后一步是调整每一层中各单元与下一层所有单元之间的连接权重。与寻找合适的网络结构和优良超参数不同，关于如何调整这些权重，已有扎实的数学基础。导数能够揭示每个连接对整体误差的贡献，从而可以沿着正确方向进行小幅调整。这正是著名的梯度下降法的基础。

如图3所示，前述三个步骤以嵌套方式迭代进行。通常会测试多个不同的架构。对于每种架构，第一步（圆圈内标注为1）是获取不同的超参数组合以供尝试。接下来是调整每种配置的权重，这一过程本身也是迭代的。第三阶段是在确定架构后，选定一组超参数并调整权重，然后评估模型。此操作的结果将引导我们进入步骤4或步骤6，具体取决于是否满足某项标准。若满足标准，则会使用其他超参数组合，这些组合通常通过网格搜索获得。如果已尝试所有可能的组合，则需后退一步（标注为5），调整自编码器架构。步骤6标志着流程结束，此时自编码器已准备好在系统中使用。

2.2 EvoAAA方案

如前所述，针对优化过程的第 2 步和第 3 步，已有广为人知的方法（见图2）：通过网格搜索寻找超参数（第 2 步），以及通过梯度下降调整权重（第 3 步）。相反，如何找到合适的自编码器架构仍然是一个尚未解决的问题。

在某个数据集上积累的经验，未必适用于数据发生变化的情况。组合数量可能是无限的，因此通过穷举搜索自动化这一过程并不可行。大多数专家会根据需要处理的变量数量，采用一些启发式方法来选择层数和单元数，而激活函数等方面则是静态指定的。

为了实现自编码器设计过程的自动化，如果预先限制组合数量，就可以通过简单搜索找到合适的架构。这与针对超参数采用的网格搜索过程类似。这种方法可以从一组预先设计好的结构中选出最佳结构。然而，它无法保证这些预先设计的模型具有良好的性能。也可以采用其他直接的搜索启发式方法，例如从基础结构出发，逐步添加或删除层/单元，只要自编码器的性能有所提升即可。

EvoAAA 方法提出通过进化方法解决这项任务，因为进化方法已充分展现出解决各种优化问题的能力。具体而言，该方法旨在利用基于种群的算法，随时间推移逐步演化出一组自编码器架构。为此，文中引入了一种编码所有可能自编码器配置的方法，这种表征将作为进化方法处理的染色体表示。

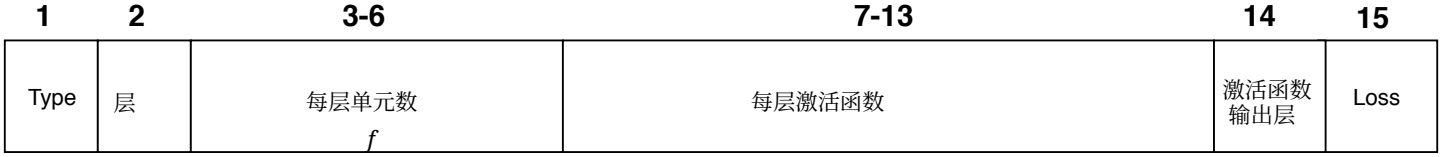

图4：染色体基因、名称及其取值范围。

表2：各基因的用途及其取值说明。

Name	用途	取值
Type	设置要使用的自编码器类型	1）基本型，2）去噪型，3）收缩型，4）鲁棒型，5）稀疏型，6）变分型-化
层	编码器/解码器中的附加层数	0）仅有编码层，1–3）编码器和解编码器
单元	设置每层的单元数，其中 f 为数据集中的特征数量	第一个整数（第 3 代）用于配置外层中的单元数而最后一个（第 6 代）用于设置编码长度。
激活函数	每层使用的激活函数，包括编码器和解码器	1) 线性，2) sigmoid，3) 双曲正切，4) ReLU，5) SELU，6) ELU，7) Softplus，8) Softsign
输出激活函数	输出层的激活函数	1) 线性，2) ReLU，3) ELU，4) Softplus
Loss	拟合过程中使用的损失函数	1) 均方误差，2) 平均绝对误差，3) 平均绝对百分比误差，4) 二元交叉熵，5) 余弦邻近度

2.2.1 染色体表示

进化算法通常使用二进制或实数基因。一组基因构成种群中的一条染色体或一个个体。在 EvoAAA 中，每条染色体都将编码一个自编码器的完整架构。不过，这里使用的是整数基因表示，而非二进制或实数基因。

如图4所示，染色体由15个基因组成。每个基因的编号显示在上方，名称显示在内部，可赋予的取值范围则显示在其正下方。表2说明了每个基因的用途及其取值的含义。这种自编码器编码的主要特点如下：

- 根据施加的限制，自编码器可以学习不同类型的表示。其中一些限制与自编码器的类型有关 [17], ，旨在实现稀疏性、从含噪样本中学习等。染色体中的第一个基因可从六种不同类型的自编码器中进行选择（详情见表2）。

- 自编码器将包含一个编码层，以及最多六个额外的隐藏层，这些隐藏层必须成对设置：2、4 或 6 层。第二个基因的取值范围为 0 到 3，表示隐藏层的层对数量。因此，最简单的自编码器仅有 3 层：输入层、编码层和输出层；而最复杂的自编码器总共由 9 层组成。

- 基因 3 至 6 指定隐藏层中的单元数量。最后一个值对应最内侧的层，因此决定编码长度。其余三个值分别与各层对相关，从外到内排列。数据中的特征数量记为 f，它限制了任意一层中的单元数量。此外，内层不能比外层更大。

- 除输入层仅负责将数值传递到下一层外，自编码器中的所有其他层都使用激活函数。尽管自编码器的所有单元通常使用相同的激活函数，但并没有任何限制要求必须如此。所提出的编码使用 7 个基因（7 至 13）为每个内部层选择要使用的激活函数，共有八种不同选项（见表2）。输出层由基因 14 单独控制，因为只允许使用能产生正输出的激活函数。

- 染色体中的最后一个基因可以取值 1 到 5，表示训练自编码器时要在内部评估的损失函数。这些取值的含义见表2。

通过对基因 3 至 6 施加的限制，生成的自编码器始终是欠完备的 [17]。这意味着学习到的表示会比原始表示更紧凑，向量包含的数值也更少。此外，在调整权重时评估的损失函数将只关注重构误差。自编码器可以通过训练来提高类别可分性、降低数据复杂度，以及实现其他目标。在本例中，主要目标是尽可能准确地重构数据模式。这会促使内部层尽可能集中通用且有用的信息，使自编码器之后能够用于任何类型的任务。

2.2.2 自编码器复杂度惩罚

设计用于学习新表示的自编码器时，主要关注的是原始性能。自编码器的性能通常通过根据学习到的编码重构数据模式时产生的误差来衡量。不过，获得编码所需的时间也是需要考虑的因素。因此，EvoAAA 引入了惩罚因子： α。它将用于进化方法的适应度函数，以评估每种自编码器配置的优劣。

适应度函数用于确定各个解的质量，其计算方式如式（1）所示。其中， $trainloss$ 表示使用训练数据时自编码器产生的重构损失， $Layers$ 表示额外隐藏层的数量（第2个基因），$Units\ coding$ 表示编码层的大小（第6个基因）， α 则是根据自编码器复杂度设定惩罚程度的系数。

$$fitness = trainloss + \alpha(Layers \times Units\ coding) \tag{1}$$

因此，与较复杂的自编码器相比，重构性能相近但架构更简单的自编码器（更多细节见第3.5.8小节）会更受青睐。促使模型采用较少层数和较短编码的偏好程度由 α 值调节。这是 EvoAAA 所需的唯一参数。

2.2.3 搜索空间

采用复杂的进化算法来寻找合适的自编码器架构，是因为搜索空间十分庞大。因此，需要一种策略，在无需探索整个搜索空间的情况下找到优质解。那么，假设采用前文所述的染色体，搜索空间有多大?

除基因 3-6 外（这些基因的取值会随数据集中的特征数量而变化），共有超过十亿种组合（见式 2）。

$$\begin{gathered} Type \times Layers \times Act^7 \times Act\ out \times Loss = \\ 6 \times 4 \times 8^7 \times 4 \times 5 = 1\ 006\ 632\ 960 \end{gathered} \tag{2}$$

对于仅有几十个属性的超小型数据集，组合数量会增长到数十亿种。对于高维数据集，解的数量将达到数万亿，甚至更多。目前，要评估所有这些解并找出最佳解是不可行的。因此，通过蛮力法寻找最优自编码器架构并非总是可行。不过，我们可以借助基于进化策略的优化机制找到足够好的解。

3 EvoAAA 的算法框架

EvoAAA 方法是一种通用的进化方案，用于搜索优良的自编码器架构。它可以通过任何基于种群的进化算法来实例化。本研究测试了 EvoAAA 的三个具体实例：一个以遗传算法作为底层搜索方法，一个依赖差分进化，另一个采用进化策略。我们将对这三种方法进行相互比较，并与两种基线搜索方法——简单的穷举搜索和随机搜索——进行比较。实验将使用九个不同的数据集。

3.1 进化搜索算法

我们提出使用三种不同的进化算法来实例化 EvoAAA。它们都将采用前述染色体表示。这三种方法分别是:

- **遗传算法（GA）**。一种经典的遗传算法 [56], , 种群中的个体通过交叉算子进行演化，产生新的个体，并以一定概率对其施加变异算子。
- **进化策略（ES）**。一种积极的求解算法 [58], , 仅使用少量个体，并且只通过变异产生新的个体。
- **差分进化（DE）**。一种基于种群的优化算法 [57] , 通过两个随机智能体之间的差异相对于另一个作为参照的智能体来生成新的个体（智能体）。

表3总结了运行这些方法时所使用的主要参数。对于遗传算法和进化策略，染色体中的每个基因都以 1/15 的概率发生变异，该值根据染色体长度设定。遗传算法采用精英策略，保留适应度较高的前百分之十个体。进化策略本身具有精英特性，只从新变异个体与旧种群的并集中选择最佳候选个体。对于遗传算法，每次迭代都会选出并保留适应度最高的 5 个个体。随后，使用轮盘赌概率从其余个体中随机选出两个父代。

表 3：进化算法的主要参数 法。

参数	GA	ES	DE
种群规模	50	4	150
迭代次数	100	500	30
变异概率	1/15	1/15	NA
交叉概率	1.0	NA	0.5
精英策略（个体）	5	NA	NA
终止代价	0	0	0

表4：实验研究中使用的数据集

数据集	Type	变量	实例	来源
CIFAR-10	整数	1 024	60 000	[75]
Delicious	二进制	983	16 105	[76]
Fashion-MNIST	整数	784	70 000	[77]
玻璃	Real	9	214	[78]
电离层	Real	34	351	[79]
MNIST	整数	784	70 000	[80]
Semeion	二进制	256	1 593	[81]
Sonar	Real	60	208	[82]
Spect	二进制	22	267	[83]

采用多点交叉算子，交叉点按照图4中的示意图确定。这样，作为父代的两个个体便可交换其若干基因以产生后代。通过这种方式不断生成新个体，直至达到种群规模，并替换所有旧个体。最后，按照前文所述的概率对部分个体使用变异算子。可以看出，这是一种激进的方案，旨在最大限度地探索如此庞大的搜索空间。对于差分进化，采用 DE/local-to-best/1 优化方法。种群规模根据表示长度（15 个基因 $\times$ 10）确定。交叉概率以及其他特定参数均按照 [57, 74] 中的建议设定。每种方法的迭代次数经过调整，以确保评估次数大致相同。

3.2 数据集

为了比较 EvoAAA 的三个实例以及穷举和随机搜索方法的性能，本研究使用了九个数据集。它们的主要特征详见表4。最后一列列出了每个数据集的来源文献。选择这些数据集遵循以下标准：

- **属性类型：** 自编码器是一种专门的人工神经网络，通过一系列计算重构输入值。这些输入值必须是数值型，本文考虑了三种情况：实数值、整数值和二进制值。目标是评估自编码器的性能如何随其需要重构的属性类型而变化。

- **属性数量：** 一半的数据集包含数百个属性；其中，CIFAR-10 的属性甚至超过一千个。其余五个数据集规模较小，仅包含少量变量。这样便可分析所发现的自编码器在变量较少与较多时的表现。

由于自编码器是一种无监督的重 对于表示学习方法，所有数据集均已移除类别属性 .
变量的数量如 表4所示，表示经过处理的有效变量数量。

3.3 限制与评估

每个数据集将获得五种自编码器架构：EvoAAA-Gen，即基于遗传算法的实例；EvoAAA-Evo，即采用进化策略作为底层搜索方法的实例；EvoAAA-Dif，即基于差分进化的实例；以及分别通过穷举（Exh）和随机搜索（Ran）生成的两种架构。以下限制和评估流程适用于所有架构：

- **性能评估：** 使用常见的均方误差指标来评估自编码器的性能。该指标没有上限，取值取决于原始数值范围。均方误差越低，自编码器的性能越好。该值是公式1中的 $trainloss$ 因子。

• **终止代价：** 如表3所示，三种进化算法的终止代价均设为 0。这意味着，如果找到一个返回 $MSE = 0$ 的配置，搜索就会停止；否则，只有达到最大运行时间或最大迭代次数时才会停止。

• **最大运行时间：** 五种搜索方法的运行时间均限制为 24 小时。时间到后，将返回截至当时找到的最佳解，作为首选的自编码器结构。实际中，这一限制会阻碍穷举搜索遍历所有可能的自编码器配置，也会限制随机搜索所测试的组合数量。

除了均方误差之外，执行过程中还会记录每种方法测试的架构数量，以及每个个体的解。组合数受最大运行时间以及进化方法的种群规模和迭代次数共同限制。

3.4 实验与结果

45 次实验[1] （9 个数据集乘以 5 种搜索方法）均在相同硬件上执行：一台配备 NVidia RTX-2080 GPU 的 PC，并使用以下软件配置：Arch Linux [84], CUDA 10.1 [85], cudnn 7.6 [86], Tensorflow 1.14 [87], Keras 2.24 [88] 以及 ruta 1.1 [89]。使用了 Tensorflow 的 RMSProp [90] 优化器。由于它依赖自适应学习率，因此无需优化这一超参数。批量大小采用 Keras 中的默认值 32。最后，自编码器的训练轮数为 20。虽然在尝试对表征进行最终优化时，通常会使用更多训练轮数，例如 100 轮甚至更多，但本研究主要关注比较搜索过程，而不是获得完全优化的自编码器。减少训练轮数可以在一定时间内尝试更多架构。关于数据实例的使用，其中 20% 从一开始就留作评估自编码器性能，并计算均方误差。其余 80% 的模式则交由 Keras，用于训练每种自编码器架构。

结果汇总见表5。对于每个数据集，各列分别表示其名称、搜索方法、测试的个体（自编码器配置）数量、最佳个体的隐藏层数和编码长度、其复杂度（假设 $\alpha = 0.0001$）以及使用测试数据得到的均方误差。对于最后两列，数值越低，自编码器的表现越好，即复杂度越低、重构性能越优。每个数据集的最佳配置（均方误差最低）已用粗体标出。这些数值包含复杂度惩罚，实际的最佳均方误差见表 6。表中还列出了根据所有已评估的解计算得出的平均误差值和最佳值。

对于三个 EvoAAA 实例，测试个体的数量受种群规模、迭代次数以及最大运行时间的限制。此外，探索过程中产生的所有无效配置都会在进入训练和测试阶段之前被舍弃。穷举和随机策略只生成有效的自编码器架构，其唯一限制是运行时间。

3.5 讨论

表5所示的结果可从多个角度进行分析：原始性能，即均方误差越低越好；探索空间的大小，尽管检查的个体越多通常越好，但也必须考虑能否通过探索较小的空间找到较优的解；以及解的复杂度，即层数越少、编码长度越短越好。此外，还可以研究运行时间、收敛情况等其他方面。

3.5.1 性能分析

不难看出，误差值最高的始终是穷举搜索方法。均方误差没有上界，且取决于原始属性值。因此，它无法用于比较不同的数据，只能比较每个数据集上的五种方法。总体而言，**Exh** 产生的误差比 EvoAAA-Dif、EvoAAA-Gen 和 EvoAAA-Evo 高出一到两个数量级。

比较三个 EvoAAA 实例可知，EvoAAA-Gen 在 9 个数据集中有 5 个表现最佳；EvoAAA-Dif 紧随其后，在 9 个数据集中有 3 个表现最佳，尽管在某些情况下，它与 EvoAAA-Evo 的差异几乎可以忽略。最大差异出现在 Fashion-MNIST 数据集上：EvoAAA-Dif 方法取得的误差仅为 EvoAAA-Gen 的四分之一，同时采用了更简单的架构。本分析考虑了自编码器的复杂度，即均方误差会乘以惩罚因子。

除了最佳值之外，分析搜索策略的平均表现也很有意义。为此，表 6 按数据集（列）和方法（行）展示了每种方法在每个数据集上达到的均方误差平均值和最小值（最佳值）[2] 。

[1]用于重现这些实验的 R 代码以及实验中使用的数据集，均可从 https://github.com/fcharte/EvoAAA 下载。要运行这些代码，您需要安装当前版本的 R、若干 R软件包（详见所提供的脚本）以及本节列出的框架。每次运行都会生成两个结果文件，其中包含所有评估过的解，以及最佳解的自编码器结构。

[2]这些是原始均方误差值，而非经过复杂度惩罚的均方误差（惩罚因子已在第 2.2.2 小节中说明，它取决于 α 参数；在所述实验中，该参数设为 0.0001）。因此，表 6 中的最佳值低于表5中的值。

表5：结果汇总。针对每种数据集-优化策略组合，列出了评估的个体数量、自编码器的层数和编码长度、复杂度以及均方误差（包括惩罚因子）。

数据集	方法	个体	层	编码长度	复杂度	误差（均方误差）↓
CIFAR-10	EvoAAA-Dif	2 537	1	38	0.004	0.0137
	EvoAAA-Evo	4 001	1	52	0.005	0.0133
	Exh	1 739	1	1	0.000	0.0395
	EvoAAA-Gen	1 115	1	39	0.004	**0.0131**
	Ran	2 101	3	5	0.002	0.0260
Delicious	EvoAAA-Dif	4 650	2	19	0.004	0.0124
	EvoAAA-Evo	4 001	1	39	0.004	0.0119
	Exh	5 837	1	1	0.000	0.0165
	EvoAAA-Gen	1 401	1	32	0.003	**0.0102**
	Ran	5 901	2	10	0.002	0.0134
Fashion-MNIST	EvoAAA-Dif	2 604	1	256	0.026	**444.8169**
	EvoAAA-Evo	755	3	61	0.012	565.4571
	Exh	1 494	1	1	0.000	4 180.2370
	EvoAAA-Gen	439	5	156	0.047	1 881.0680
	Ran	1 501	2	35	0.007	782.3572
玻璃	EvoAAA-Dif	4 650	2	3	0.001	24.8785
	EvoAAA-Evo	4 001	3	3	0.001	5.4093
	Exh	35 368	1	1	0.000	29.2589
	EvoAAA-Gen	4 505	1	5	0.000	**0.4473**
	Ran	35 301	3	7	0.002	1.2705
电离层	EvoAAA-Dif	4 650	2	23	0.005	**0.0740**
	EvoAAA-Evo	4 001	3	15	0.003	0.0931
	Exh	29 905	1	1	0.000	0.2099
	EvoAAA-Gen	2 302	3	15	0.003	0.0917
	Ran	29 901	3	23	0.007	0.1049
MNIST	EvoAAA-Dif	2 754	1	162	0.016	**192.5133**
	EvoAAA-Evo	732	3	332	0.066	431.6960
	Exh	1 491	1	1	0.000	4 104.1400
	EvoAAA-Gen	403	1	156	0.016	254.7397
	Ran	1 401	2	35	0.007	611.5036
Semeion	EvoAAA-Dif	4 650	1	136	0.014	0.0459
	EvoAAA-Evo	4 001	1	134	0.013	**0.0355**
	Exh	19 861	1	1	0.000	0.1940
	EvoAAA-Gen	4 505	1	110	0.011	0.0376
	Ran	19 901	2	132	0.026	0.0604
Sonar	EvoAAA-Dif	4 650	1	29	0.003	0.0142
	EvoAAA-Evo	4 001	3	17	0.003	0.0162
	Exh	34 557	1	1	0.000	0.0462
	EvoAAA-Gen	4 505	3	7	0.001	**0.0139**
	Ran	35301	4	8	0.003	0.0145
Spect	EvoAAA-Dif	4 650	2	13	0.003	0.0959
	EvoAAA-Evo	4 001	3	14	0.003	0.0834
	Exh	30 740	1	1	0.000	0.1829
	EvoAAA-Gen	4 505	5	15	0.004	**0.0703**
	Ran	30 701	3	14	0.004	0.1145

表 6：按数据集和搜索策略划分的平均值和最佳性能（原始均方误差）

	CIFAR-10	Delicious	Fashion-MNIST	玻璃	电离层	MNIST	Semeion	Sonar	Spect
EvoAAA-Dif（平均值）	0.0256	0.0186	8979.9337	604.3130	0.2719	5073.9444	0.2348	0.0404	0.2575
EvoAAA-Evo（平均值）	**0.0085**	0.0086	**891.1958**	562.3818	**0.1152**	**616.6640**	**0.0322**	0.0223	0.1145
Exh（平均值）	0.0498	0.0195	18734.2379	611.4017	0.3687	9646.3325	0.2836	0.1331	0.3213
EvoAAA-Gen（平均值）	0.0097	**0.0075**	2852.7007	**345.4758**	0.1227	3323.4105	0.0405	**0.0157**	**0.1006**
随机（平均值）	0.0461	0.0206	8557.0936	606.7591	0.3303	4997.0538	0.2886	0.0761	0.2910
EvoAAA-Dif（最佳）	**0.0020**	0.0131	**444.7913**	563.1547	**0.0694**	**192.4971**	0.0323	**0.0106**	0.0924
EvoAAA-Evo（最佳）	0.0080	0.0080	565.4449	316.0380	0.0963	431.6296	**0.0221**	0.0128	0.0806
Exh（最佳）	0.0394	0.0164	4180.2370	576.7005	0.2098	4104.1400	0.1939	0.0461	0.1828
EvoAAA-Gen（最佳）	0.0090	0.0055	1881.0210	**0.4468**	0.0872	254.7241	0.0266	0.0115	**0.0658**
Rnd（最佳）	0.0047	**0.0021**	782.3502	564.4037	0.0980	564.2114	0.0339	0.0114	0.1098

表7：测试方法的性能排名

数据集	**EvoAAA-Dif**	**EvoAAA-Evo**	**Exh**	**EvoAAA-Gen**	**Ran**
CIFAR-10	1	3	5	4	2
Delicious	4	3	5	2	1
Fashion-MNIST	1	2	5	4	3
玻璃	3	2	5	1	4
电离层	1	3	5	2	4
MNIST	1	3	5	2	4
Semeion	3	1	5	2	4
Sonar	1	4	5	3	2
Spect	3	2	5	1	4
平均值	2.00	2.56	5.00	2.33	3.11

案例中。可以观察到，穷举搜索和随机搜索得到的最佳均方误差远逊于 EvoAAA-Dif、EvoAAA-Gen 和 EvoAAA-Evo 的平均均方误差。由此可得出结论：即使只进行少量迭代，EvoAAA 的结果也会优于穷举搜索或随机搜索 24小时 的结果。在大多数案例（CIFAR-10、Delicious、电离层、Semeion 和 Sonar）中，各搜索方法的最佳值与平均值之间的差异往往很小。总体而言，EvoAAA-Evo 的这些差异似乎小于 EvoAAA-Gen 或 EvoAAA-Dif。例如，在前者中，Fashion-MNIST、玻璃和 MNIST 的平均值与最佳值比在后者中更接近。这表明，如果只能负担少量迭代，EvoAAA-Evo 会是更好的选择。

如果严格只关注各方法在所有迭代结束时取得的最佳值（未进行复杂度惩罚），那么 EvoAAA-Dif 脱颖而出，胜过其他算法。它取得了 5 个最佳值，而 EvoAAA-Gen 有 2 个，EvoAAA-Evo 有 1 个，随机搜索也有 1 个。为评估各方法的整体性能，表7给出了排名，最后一行显示平均排名。可以看出，EvoAAA-Dif 的表现最佳，EvoAAA-Gen 和 EvoAAA-Evo 紧随其后。随机搜索与穷举搜索之间的排名差异相当显著。穷举搜索存在偏差：它会依次尝试连续的配置，直到运行时间耗尽，因此探索解空间的机会少于随机搜索。对最佳均方误差值（表 6 下半部分）进行弗里德曼统计检验后，得到了 p值 $= 0.0004248178$ 。这意味着所评估的优化策略之间存在统计学上的显著差异。

3.5.2 已探索空间的分析

如表5所示，总体而言，穷举搜索和随机搜索方法探索的解空间远大于进化方法。在某些情况下，例如 glass、ionosphere、Semeion、Sonar 和 Spect，这些方法检查的配置数量最多可达 EvoAAA 的15倍。这是因为这些搜索算法将全部时间用于分析候选解，而进化方法还需要执行其他任务，例如个体选择、交叉、变异等。然而，考虑到自编码器复杂度，随机搜索和穷举搜索都无法达到最佳性能。相反，这些方法的均方误差始终较高。正如其复杂度值所示，穷举搜索会陷入较简单的架构，尽管其运行时间通常更长（将在后文分析），且总是达到24小时。这种行为源于穷举搜索的实现方式：按照图4中的表征，从右向左尝试所有可能的解，就像遍历一个区间内的值时通常从起点到终点逐一尝试一样。随机搜索能够探索整个解空间，但它是一种无引导的方法。从某种意义上说，穷举搜索和随机搜索采用的策略截然相反。前者选择利用局部空间，尝试一个极小区域内的所有可能解；后者则在整个解空间中跳跃，不利用过去的配置来发掘最有希望的解的局部性。

在处理 glass、Semeion、Sonar 或 Spect 等小型数据集时，EvoAAA-Dif、EvoAAA-Gen 和 EvoAAA-Evo 方法探索的解数量相当接近。相较之下，使用较大型数据集时，EvoAAA-Evo 和 EvoAAA-Dif 方法能够检查更多候选结构。这可能是因为与 EvoAAA-Gen 相比，EvoAAA-Evo 生成后代的过程更简单，无需交叉，且种群规模较小；此外，EvoAAA-Dif 的迭代次数也少于 EvoAAA-Gen。

3.5.3 解的复杂度分析

比较不同搜索过程时，还需考虑所找到架构的复杂度。从理论上讲，在性能相近的情况下，更简单的架构优于更复杂的架构。

从表5第六列可以看出，Exh 方法的复杂度始终最低。如前所述，穷举搜索局限于较小的架构，这些架构的激活函数和损失函数组合有数千种（见图4）。然而，均方误差值表明，这些并非最佳解。

正如预期，随机搜索也会产生各不相同的自编码器配置。有时，这些配置比 EvoAAA 方法生成的更简单；有时则更复杂。

除 Fashion-MNIST 和 Spect 等少数例外情况外，EvoAAA-Gen 通常生成层数更少、编码更紧凑的自编码器，其复杂度低于 EvoAAA-Dif 和 EvoAAA-Evo。因此，乍一看，EvoAAA-Gen 搜索似乎是最佳选择，因为它能提供更简单且重构能力更强的自编码器。然而，这也需要付出代价，原因将在下文解释。

3.5.4 运行时间分析

表8列出了实验期间记录的每种配置的运行时间。与表5一样，每个数据集对应五行，分别对应五种搜索方法。分析这些信息，可以得出以下结论：

表8：运行时间汇总

	运行时间 ↓				
数据集	**EvoAAA-Dif**	**EvoAAA-Evo**	**Exh**	**EvoAAA-Gen**	**Rnd**
CIFAR-10	24h	24h	24h	24h	24h
Delicious	18小时4分钟	17小时54分钟	24h	6小时51分钟	24h
Fashion-MNIST	24h	24h	24h	24h	24h
玻璃	1小时41分钟。	1小时20分钟。	24h	2小时20分钟。	24h
电离层	1小时33分钟。	0小时44分钟。	24h	1小时59分钟。	24h
MNIST	24h	24h	24h	24h	24h
Semeion	2小时28分钟。	1小时32分钟。	24h	5小时0分钟。	24h
Sonar	1小时24分钟。	1小时10分钟。	24h	4小时13分钟。	24h
Spect	1小时31分钟。	0小时59分钟。	24h	5小时57分钟。	24h

- 穷举和随机搜索策略总是会达到24小时的时间上限，无法检查足够多的配置来找到一个好的解，尽管在某些情况下，随机搜索方法与进化方法的表现相近。

- 对于较大的数据集（CIFAR-10、Fashion-MNIST 和 MNIST），五种方法都无法完成搜索过程，因为运行时间耗尽。

- 一般而言，EvoAAA-Dif 和 EvoAAA-Evo 只需使用 EvoAAA-Gen 所耗时间的一小部分，就能找到复杂度略高但性能相近的自编码器架构。

根据这项分析，可以得出一条简单的指导原则：如果性能是首要且唯一的目标，请选择 EvoAAA-Dif 实例；如果运行时间很重要，同时可以牺牲少许重建能力，则可考虑 EvoAAA-Evo 实例。对于大型数据集，后一种方法可以节省数小时，从而获得足够好的自编码器架构。

3.5.5 随时间探索的解

接下来要分析的是各方法如何探索解空间。为此，图5至图12展示了随时间（X轴）变化而考察的各个解的均方误差（Y轴）[3]。由于各方法的运行时间并不相同，因此针对给定数据集，X轴的范围也有所不同。

[3]此处所述分析基于总体结果进行，尽管图中仅展示了一些示例。代码仓库中提供了所有数据集 × 搜索方法组合对应的图表。

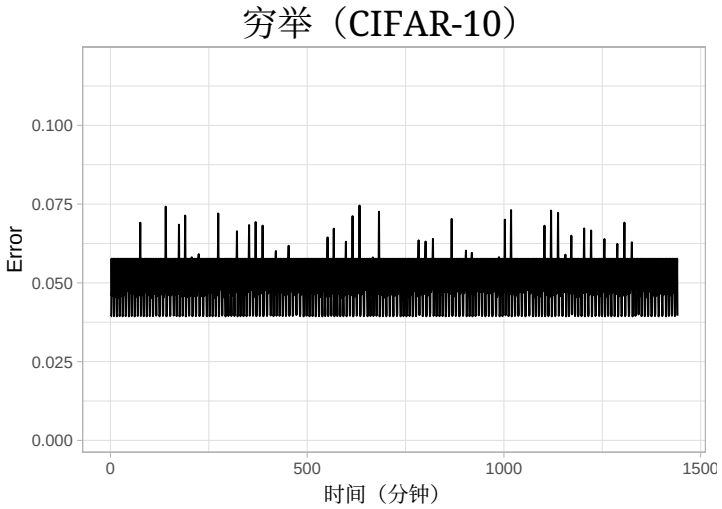

图5：随时间探索的解

可以看出，穷举搜索（图5）从开始到结束都保持着较高的误差率。对于属性相当相似的 Fashion-MNIST 和 MNIST，随着时间推移，较好的解和较差的解混杂在一起（见图6）。对于其余数据集，随着新解不断得到评估，搜索似乎并未取得太大改进。

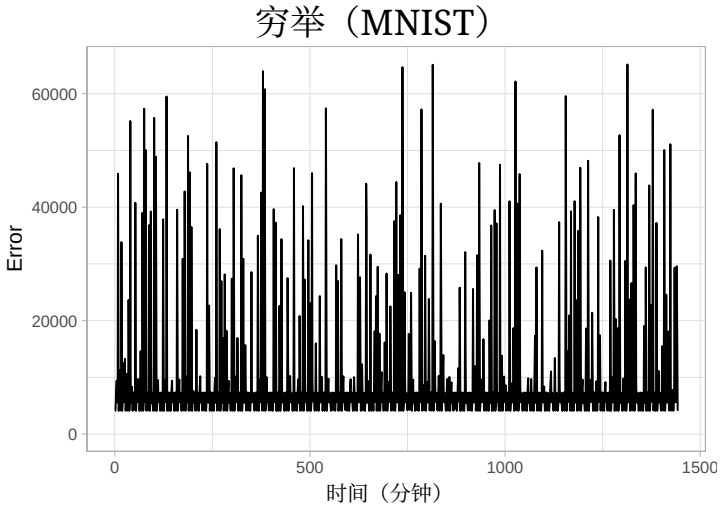

图6：随时间探索的解

如图7所示，随机搜索方法的行为与穷举搜索相似，但在已探索的个体中，其适应度变异更大（曲线更分散）。这是因为它能够跳跃遍历整个解空间，而不必逐一尝试每种可能的参数组合。

如图8和图9所示，EvoAAA-Gen 和 EvoAAA-Evo 候选个体的行为相似。两者开始时的误差率都低于穷举和随机策略，随后随着运行时间的增加而不断改进。图中并未突出显示解的质量提升，因为Y轴保持固定，以便比较这五种方法。

如图10和图11所示，EvoAAA-Evo 和 EvoAAA-Gen 方法在处理玻璃数据集时表现异常。如表4所述，该数据集仅有9个属性和少量实例。这可能是自编码器最难处理的情况，因为数据表示没有太多可压缩的空间，也没有足够的模式可供学习。

与 EvoAAA-Gen 和 EvoAAA-Evo 相比，EvoAAA-Dif 算法在大多数情况下似乎具有更广的探索范围，这可能是因为其种群规模更大。图12展示了该搜索方法随时间评估的个体质量。

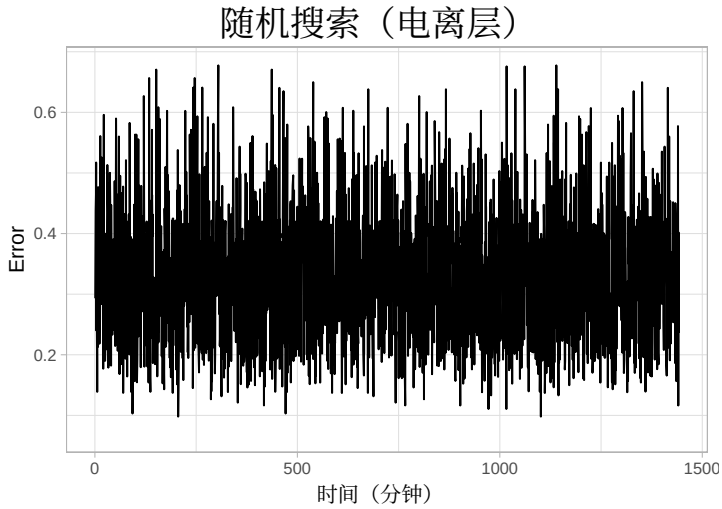

图7：随时间探索的解

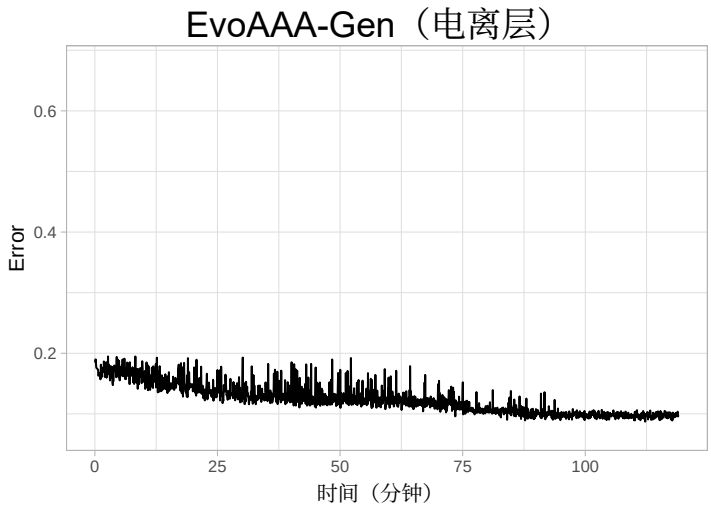

图8：随时间探索的解

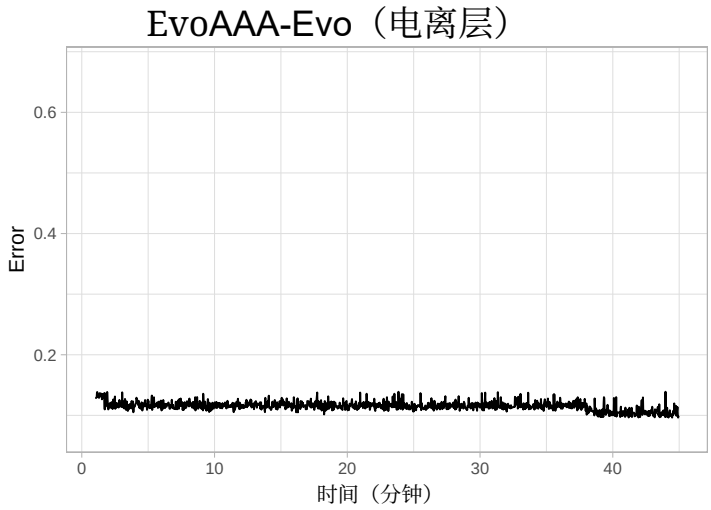

图9：随时间探索的解

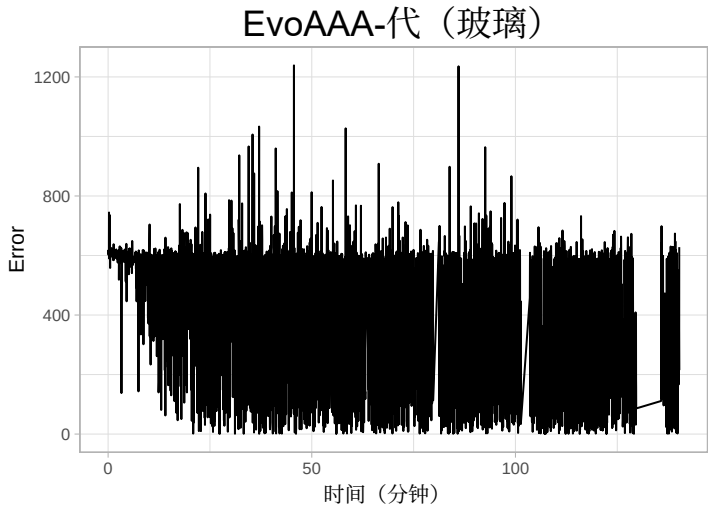

图10：随时间探索的解

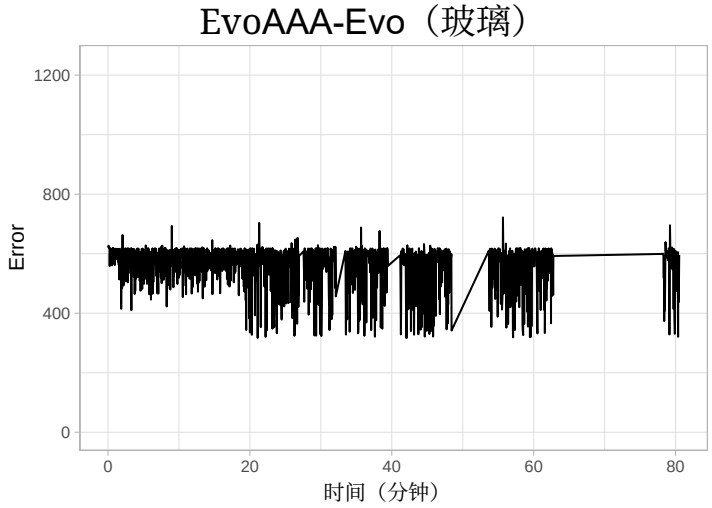

图11：随时间探索的解

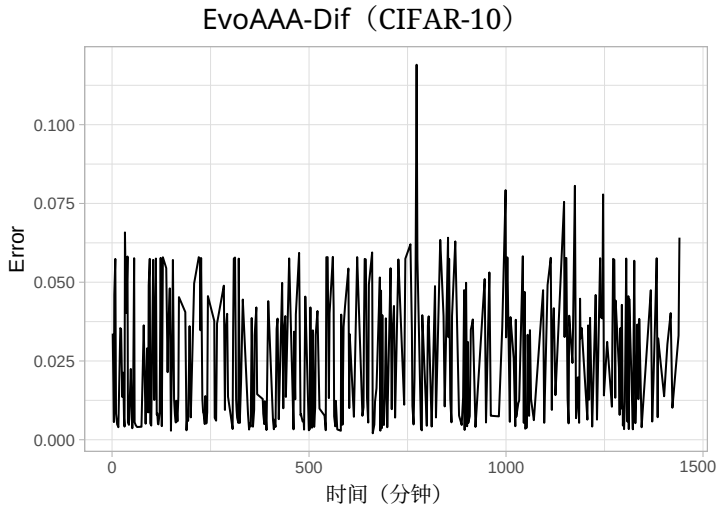

图12：随时间探索的解

3.5.6 已实现的改进与探索的解

在图 13 至图 15 中，X轴从时间改为已探索解的数量，Y轴则显示截至目前找到的最佳解的误差。每个数据集的 Y 轴刻度保持固定，但 X 轴刻度有所不同，因为每种

方法考察的候选个体数量各不相同。我们的目标是分析各优化策略随着探索更多解空间而取得的改进。所有图表均可在代码仓库中查看。

正如预期的那样，随着考察的可能架构数量增加，五种方法都能找到更好的解。然而，穷举法和随机搜索在搜索过程中都呈现出一些阶梯状停滞，有时会长时间停留在同一误差水平。图 13 展示了这种现象的一个例子。

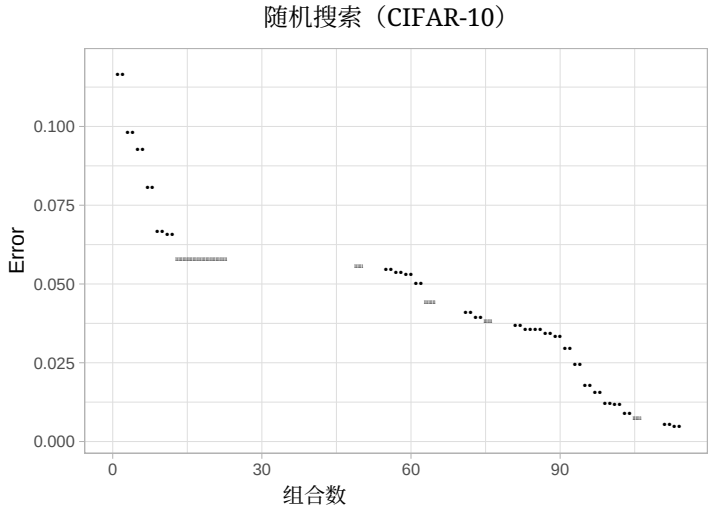

图 13： 探索解时取得的改进 过程中

EvoAAA-Gen 和 EvoAAA-Evo 取得的改进似乎更加循序渐进，直至达到最低水平（见图14）。此外，在探索较少数量的潜在解后，误差率便已降低至这一水平。

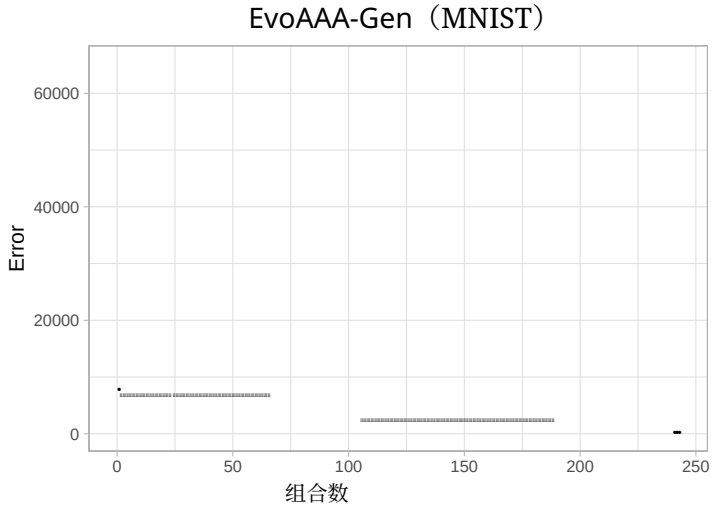

图14：探索解时取得的改进

EvoAAA-Dif 算法的表现多少是前几种算法的混合：结果持续改进，且初始误差较低，但与随机策略仍有一定相似之处。

3.5.7 搜索收敛

接下来要分析各方法的收敛速度。代码仓库中有九张图，每个数据集对应一张。其中一张如图16所示，将作为后续分析的参考。这样，五种优化策略共用相同的X轴和Y轴刻度。X轴表示运行时间。图中只展示了部分耗时，否则表示EvoAAA实例的曲线只会占据左侧很小一部分区域。曲线继续向右延伸，意味着后来找到了更好的值；而未延伸至X轴边界的曲线，则表示图中已经显示了所达到的最佳值（例如，使用MNIST数据集的随机搜索）。从这些图中可以得出以下结论：

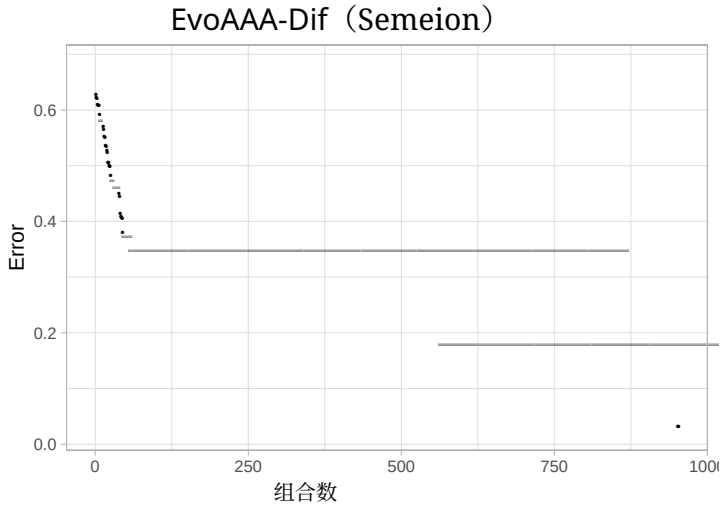

图15：随着解的探索，性能得到的提升

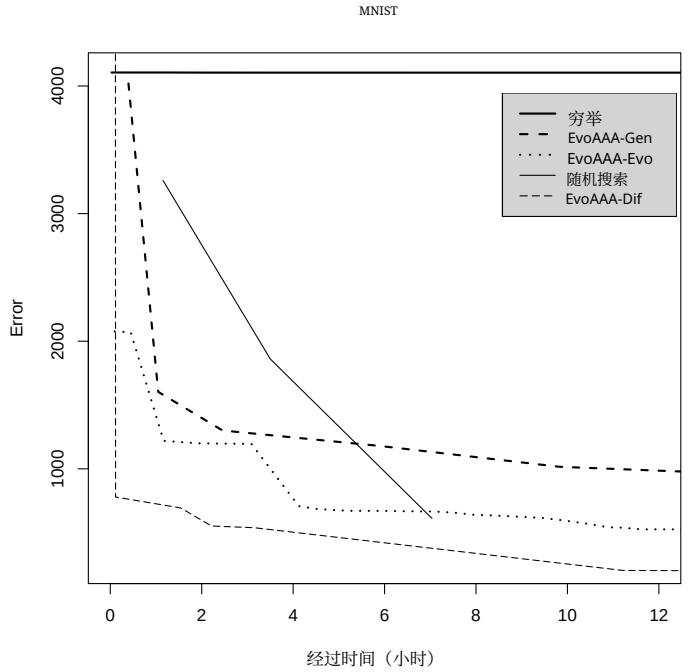

图16：各 EvoAAA 方法相对于穷举搜索和随机搜索的收敛速度

• 总的来说，穷举方法（粗实线）大部分时间都停留在较高的误差率。唯一的例外是玻璃数据集，在这种搜索方法下，其表现几乎赶上了三种EvoAAA策略。

• 随机搜索（细实线）的表现如预期般毫无方向。有时，它会随着时间推移缓慢进展，类似于穷举策略（例如 CIFAR-10 和 Delicious）；有时则能很快找到一个较好的解，之后便再也找不到更好的解（例如 Fashion-MNIST 和 MNIST）。这种策略可能在短时间内取得不错的结果，但无法提供任何保证。

• 总体而言，三个 EvoAAA 实例比两种基线策略更快地收敛到较低的误差值，随后趋于稳定，并以较慢的速度持续改进。

• 在大多数情况下，进化策略方法（点线）先于遗传算法（粗虚线）达到其最优值（最低误差值），然后结束执行。相反，遗传算法方法会持续改进更长时间。由于这一特性，它在许多情况下能够胜过进化策略。

• 差分进化策略（细虚线）的初始解通常比遗传算法和进化策略的解更差，但在大多数情况下，它的收敛速度

更快，尤其是在 CIFAR-10、Fashion-MNIST 和 MNIST 等最复杂的数据集上。在某些情况下，它能以其他方案所需时间的一小部分取得最佳结果。

基于这一分析，可以调整差分进化和进化策略算法的参数，增大种群规模或迭代次数，以便运行更长时间并像遗传算法那样持续改进。对于某些较小的数据集（例如 Sonar、玻璃和 Spect），它们或许能取得与遗传算法方法相当的结果；使用差分进化甚至可能取得更好的结果。

3.5.8 惩罚因子的影响

最后，在本分析中，我们将考察与自编码器复杂度相关的惩罚因子如何影响所得结果。为此，我们在声呐数据集上采用差分进化策略，并按照对数尺度将 α 从 1 逐渐调整至 0。

α 惩罚因子的目标是在重构性能相近时优先选择更简单的自编码器架构。直观而言，较低的 α 值会产生重构能力更强、但层和单元也更多的自编码器；较高的 α 值则相反（即架构更简单，但重建准确率较低）。

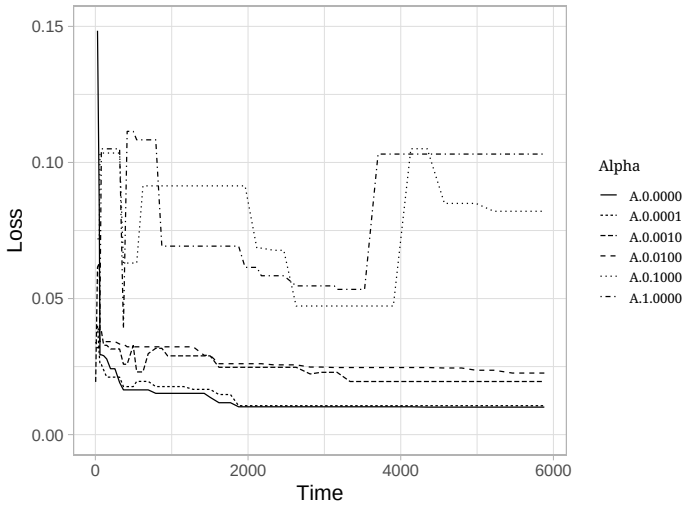

图17：使用 EvoAAA-Dif 和声呐数据集时，不同 α 值下损失函数随时间的变化

首先，图17展示了在所考虑的 α 值下，差分进化算法探索各个架构时，均方误差如何随时间变化。如图所示，最高的两个惩罚值严重影响了搜索过程，这反映在损失函数的骤变上。差分进化方法尝试通过测试更复杂的差分进化来提升性能时，惩罚值会增大。随着 α 值降低，这些跃变也随之减弱，直至 $\alpha = 0$时不再产生影响。因此，首先可以得出的结论是，较小的惩罚因子更为理想，具体而言，惩罚值应为均方误差的一小部分。

为了更好地了解随着 α 增大，性能下降的程度，图18展示了使用声呐数据集时，差分进化算法在各个惩罚值下得到的最终结果。黑色柱形对应左侧Y轴，表示均方误差（柱形越高，结果越差）；折线表示复杂度水平（右侧Y轴）。请注意，对于 $\alpha = 0$ 和$\alpha = 0.0001$，性能几乎没有变化，但复杂度却有显著差异。惩罚因子越大（图中越靠左），复杂度得分越低，而重构误差则相反。

表9：损失函数和 A 使用声呐数据集，通过 EvoAAA-Dif 获取的每个 α 值对应的 E 配置

α	**Loss**	**层和单元**
0.0000	0.00993972	57, 47, 36, 22, 36, 47, 57
0.0001	0.01062195	37, 32, 8, 32, 37
0.0010	0.01951711	23, 2, 23
0.0100	0.02262429	38, 3, 38
0.1000	0.08206710	1
1.0000	0.10308630	1

表9总结了所考察的每个 α 的确切均方误差（MSE）值和配置。如上所述， $\alpha = 0$ 与 $\alpha = 0.0001$ 之间的性能差异几乎可以忽略不计，但前者使用了7层，而后者只有5层；前者的编码长度为22个单元，而后者仅为8个。显然，在这个特定案例中，较小的惩罚因子比更大的惩罚因子或完全不施加惩罚能产生更好的自编码器架构。

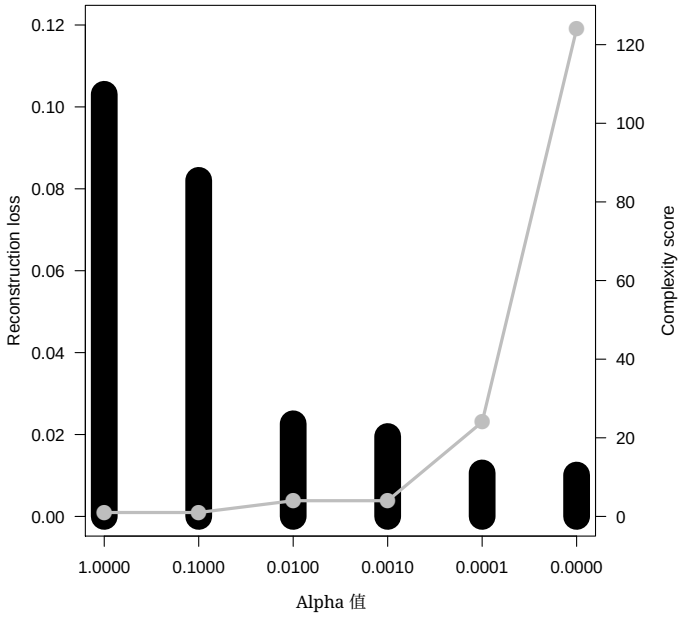

图18: 使用 EvoAAA-Dif 和 Sonar 数据集时，不同 α 值下损失函数与自编码器复杂度的关系 et

基于这项分析，本研究实验中将 α 设为0.0001，正如第3节开头所述。

4 总结

自编码器是解决表征学习问题的有用工具。然而，找到适用于每种情况的最佳自编码器架构是一项艰巨的任务。内部交叉验证是调整超参数的常用方法。但是，如果还要调整自编码器架构，解空间就会非常庞大。因此，需要更强大的方法来解决这一问题。

本文提出了 EvoAAA，这是一种基于进化的方法，用于为每个数据集寻找最佳自编码器架构。首先，本文提出了一种将自编码器架构编码到染色体中的方法。该方法足够灵活，能够涵盖大多数自编码器变体，包括不同数量的层和单元、每层各自的激活函数以及多种损失函数。随后，本文设计了三种搜索方法：一种基于差分进化，一种基于遗传算法，另一种基于进化策略。最后，本文进行了全面的实验和分析，将三种 EvoAAA 策略的结果与两个不同的基线方法——穷举搜索和随机搜索——进行了比较。

总体而言，已有研究证明，所提出的方法能够为每个数据集找到良好的自编码器结构。它未必是最优结构，因为更先进的搜索算法和优化策略或许能进一步改善这些结果；但如果需要在合理时间内得到结果，该方法优于随机选择或通过穷举查找找到的结构。所开展的实验表明，EvoAAA 是一种能够胜任这项任务的竞争性方法。

本研究部分得到了西班牙经济与竞争力部项目 TIN2015-68854-R（欧洲区域发展基金）的资助。

参考文献

[1] C.M. Bishop，《模式识别与机器学习》，Springer，2006。[2] T.S. Guzella 和 W.M. Caminhas，《垃圾邮件过滤的机器学习方法综述》，《*Expert Systems with Applications*》**36**(7)(2009)，10206–10222。

[3] S. Bhattacharyya, S. Jha, K. Tharakunnel 和 J.C. Westland，信用卡欺诈的数据挖掘：一项比较研究，《*Decision Support Systems*》**50**(3)（2011），602–613。

[4] J.B. Schafer、J. Konstan 和 J. Riedl，电子商务中的推荐系统，载于：第1届ACM电子商务会议论文集，ACM，1999，第158–166页。[5] P. Domingos，关于机器学习的一些实用知识，ACM通讯 55(10)（2012），78–87。[6] S. Garc´ıa、J. Luengo 和 F. Herrera，数据挖掘中的数据预处理，Springer，2015，第163–194页，第7章。ISBN ISBN 978-3-319-10247-4。[7] M.A. Hall，面向机器学习的基于相关性的特征选择，博士论文，怀卡托大学，汉密尔顿，1999。[8] H. Peng、F. Long 和 C.H.Q. Ding，基于最大依赖、最大相关性和最小冗余互信息准则的特征选择，IEEE模式分析与机器智能汇刊 27（2005），1226–1238。[9] I. Guyon 和 A. Elisseeff，特征提取导论，载于：特征提取：基础与应用，I. Guyon、M. Nikravesh、S. Gunn 和 L.A. Zadeh 编，Springer Berlin Heidelberg，柏林，海德堡，2006，第1–25页。[10] D.E. Rumelhart、G.E. Hinton、R.J. Williams 等，利用反向传播误差学习表示，认知建模 5(3)（1988），1。[11] Y. Bengio、A. Courville 和 P. Vincent，表征学习：综述与新视角，IEEE模式分析与机器智能汇刊 35(8)（2013），1798–1828。[12] Y. LeCun、Y. Bengio 和 G. Hinton，深度学习，《自然》521(7553)（2015），436。[13] I. Goodfellow、Y. Bengio 和 A. Courville，深度学习，麻省理工学院出版社，2016。[14] G.E. Hinton 和 R.R. Salakhutdinov，利用神经网络降低数据维度，《科学》313(5786)（2006），504–507。[15] P. Vincent、H. Larochelle、Y. Bengio 和 P.-A. Manzagol，利用去噪自编码器提取并组合鲁棒特征，载于：第25届国际机器学习会议论文集，ACM，2008，第1096–1103页。[16] P. Vincent、H. Larochelle、I. Lajoie、Y. Bengio 和 P.-A. Manzagol，堆叠去噪自编码器：在具有局部去噪准则的深度网络中学习有用表示，《机器学习研究杂志》11(12月)（2010），3371–3408。[17] D. Charte、F. Charte、S. Garc´ıa、M.J. del Jesus 和 F. Herrera，非线性特征融合自编码器实用教程：分类体系、模型、软件与指南，《信息融合》44（2018），78–96。[18] D. Charte、F. Charte、M.J. del Jesus 和 F. Herrera，自编码器在特征学习应用中的应用示例，载于：从仿生系统与生物医学应用到机器学习，J.M. Ferr´andez Vicente、´J.R. Alvarez-S´anchez、F. de la Paz L´opez、J. Toledo Moreo 和 H. Adeli 编，Springer International Publishing，2019，第412–421页。ISBN ISBN 978-3-030-19651-6。[19] M.R. Garey 和 D.S. Johnson，计算机与难解性，第29卷，W. H. Freeman，纽约，2002。[20] T. B¨ack 和 H.-P. Schwefel，参数优化进化算法概述，《进化计算》1(1)（1993），1–23。[21] F.J. Pulgar、F. Charte、A.J. Rivera 和 M.J. del Jesus，基于数据复杂度和分类器选择合适的特征融合自编码器：分析、建议与指南，《信息融合》54（2020），44–60。[22] D. Charte、F. Charte、S. Garc´ıa 和 F. Herrera，非标准监督学习问题概览：分类体系、关系、问题转换与算法适配，《人工智能进展》8(1)（2019），1–14。[23] R. Hecht-Nielsen，反向传播神经网络理论，载于：用于感知的神经网络，爱思唯尔，1992，第65–93页。[24] H. Robbins 和 S. Monro，随机逼近法，《数学统计年刊》（1951），400–407。[25] S. Lawrence 和 C.L. Giles，过拟合与神经网络：共轭梯度和反向传播，载于：IEEE-INNS-ENNS神经网络国际联合会议论文集。IJCNN 2000。神经计算：新千年的新挑战与新视角，第1卷，IEEE，2000，第114–119页。[26] M. Ahmadlou 和 H. Adeli，采用局部决策圆的增强型概率神经网络：一种鲁棒分类器，《集成计算机辅助工程》17(3)（2010），197–210。

[27] N.K. Benamara、M. Val-Calvo、J.R. ´Alvarez-S´anchez、A. D´ıaz-Morcillo、J.M. Ferr´andez-Vicente、E.Fern´andez-Jover 和 T.B. Stambouli，《基于深度神经网络集成的社交机器人实时情感识别》，载于：国际施普林格，2019年，第171–180页。[28] H. Hotelling，《统计变量复合体的主成分分析》，《教育心理学杂志》24(6)（1933），417。[29] R.A. Fisher，《多重测量的统计利用》，《人类遗传学年鉴》8(4)（1938），376–386。[30] L. Cayton，《流形学习算法》，技术报告，加利福尼亚大学圣迭戈分校，2005。[31] J.A. Lee和M. Verleysen，《非线性降维》，施普林格科学与商业媒体，2007。[32] W. Yu、G. Zeng、P. Luo、F. Zhuang、Q. He和Z. Shi，《带自编码器正则化的嵌入》，载于：机器学习与数据库知识发现联合欧洲会议，Springer，2013年，第208–223页。doi:10.1007/978-3-642-40994-3 14。[33] M. Sakurada和T. Yairi，《使用非线性降维自编码器进行异常检测》，载于：2014年机器学习用于感官数据分析第二届研讨会论文集，ACM，2014年，第4–11页。ISBN ISBN 978-1-4503-3159-3。doi:10.1145/2689746.2689747。[34] S. Park、M. Kim和S. Lee，《使用卷积自编码器进行超文本传输协议异常检测》，《IEEE开放获取期刊》6（2018），70884–70901。doi:10.1109/ACCESS.2018.2881003。[35] J. Xie、L. Xu和E. Chen，《使用深度神经网络进行图像去噪与图像修复》，载于：神经信息处理系统进展，2012年，第341–349页。[36] X. Lu、Y. Tsao、S. Matsuda和C. Hori，《基于深度去噪自编码器的语音增强》，载于：国际语音通信协会会议，2013年，第436–440页。[37] C. Blum和A. Roli，《组合优化中的元启发式算法：概述与概念比较》，《ACM计算调查》35(3)（2003），268–308。[38] H. Adeli和K.C. Sarma，《结构成本优化：模糊逻辑、遗传算法与并行计算》，约翰·威立父子出版公司，2006。[39] E. Aarts和J. Lenstra，《组合优化中的局部搜索》，Wiley，纽约（1997）。[40] M. Den Besten、T. St¨utzle和M. Dorigo，《迭代局部搜索算法的设计》，载于：进化计算应用研讨会，Springer，2001年，第441–451页。[41] R.E. Korf，《实时启发式搜索》，《人工智能》42(2–3)（1990），189–211。[42] W. Zhang，《组合优化算法》，载于：状态空间搜索，Springer，1999年，第13–33页。[43] F. Neri、N. Kotilainen和M. Vapa，《一种在点对点网络中发现资源的模因神经方法》，载于：《组合优化进化计算最新进展》，C. Cotta和J. van Hemert编，Springer Berlin Heidelberg，柏林，海德堡，2008年，第113–129页。ISBN ISBN 978-3-540-70807-0。doi:10.1007/978-3-540-70807- 0 8。[44] P.J. Van Laarhoven和E.H. Aarts，《模拟退火》，载于：《模拟退火：理论与应用》，Springer，1987年，第7–15页。[45] R. Battiti和G. Tecchiolli，《反应式禁忌搜索》，《ORSA计算杂志》6(2)（1994），126–140。[46] A.A. Freitas，《数据挖掘进化算法综述》，载于：《数据挖掘与知识发现手册》，Springer，2009年，第371–400页。[47] T. B¨ack和H.-P. Schwefel，《参数优化进化算法概述》，《进化计算》1（1993），1–23。[48] Q. Wang、H.-L. Liu、J. Yuan和L. Chen，《通过进化多目标算法优化蜂窝系统的能谱效率》，《集成计算机辅助工程》26(2)（2019），207–220。[49] C. Kyriklidis和G. Dounias，《项目管理中的资源均衡优化进化计算》，《集成计算机辅助工程》23(2)（2016），173–184。

[50] M. Kociecki 和 H. Adeli，两阶段遗传算法用于自由曲面钢空间桁架屋盖结构的拓扑优化复杂曲率结构，人工智能工程应用 **32** （2014），218–227。

[51] M. Kociecki和H. Adeli，《使用进化计算优化具有复杂几何形状的自由曲面钢空间桁架屋盖结构》，《人工智能工程应用》38（2015），168–182。[52] C. Blum，《边加权 k 基数树问题的蚁群优化》，载于：第4届遗传与进化计算年会论文集，Morgan Kaufmann 出版公司，2002年，第27–34页。[53] R. Poli、J. Kennedy和T. Blackwell，《粒子群优化》，《群体智能》1(1)（2007），33–57。[54] J.A. Foster，《计算遗传学：进化计算》，《自然综述：遗传学》2(6)（2001），428。[55] T. B¨ack、D.B. Fogel和Z. Michalewicz，《进化计算1：基本算法与算子》，CRC出版社，2018。[56] L. Davis，《遗传算法手册》（1991）。[57] R. Storn和K. Price，《差分进化——一种用于连续空间全局优化的简单高效启发式方法》，《全局优化杂志》11(4)（1997），341–359。[58] I. Rechenberg，《进化策略：达尔文进化的数学模型》，载于：《协同学——从微观秩序到宏观秩序》，Springer，1984年，第122–132页。[59] H. KIM和H. ADELI，《使用浮点遗传算法优化组合楼板的离散成本》，《工程优化》33(4)（2001），485–501。[60] F. Friedrichs和C. Igel，《多个支持向量机参数的进化调优》，《神经计算》64（2005），107–117。[61] S.R. Young、D.C. Rose、T.P. Karnowski、S.-H. Lim和R.M. Patton，《通过进化算法优化深度学习超参数》，载于：高性能计算环境中的机器学习研讨会论文集，ACM，2015年，第4页。[62] H. Kitano，《使用遗传算法训练神经网络的收敛速度实证研究》，载于：AAAI，1990年，第789–795页。[63] M. Scholz，《基于改进进化策略的神经网络学习策略》，载于：自然启发式并行问题求解国际会议，Springer，1990年，第314–318页。[64] L.D. Whitley和T. Hanson，《使用更快、更准确的遗传搜索优化神经网络》，载于：第3届国际遗传算法会议论文集，Morgan Kaufmann 出版公司，1989年，第391–397页。[65] K. Chellapilla和D.B. Fogel，《进化神经网络，在不依赖专家知识的情况下下西洋跳棋》，《IEEE神经网络汇刊》10(6)（1999），1382–1391。[66] D. Floreano、P. D¨urr和C. Mattiussi，《神经进化：从架构到学习》，《进化智能》1(1)（2008），47–62。[67] F. Gruau，《模块化神经网络的自动定义》，《自适应行为》3(2)（1994），151–183。[68] K.O. Stanley、J. Clune、J. Lehman和R. Miikkulainen，《通过神经进化设计神经网络》，《自然·机器智能》1(1)（2019），24–35。[69] I. Guyon、L. Sun-Hosoya、M. Boull´e、H.J. Escalante、S. Escalera、Z. Liu、D. Jajetic、B. Ray、M. Saeed、M. Sebag等，《AutoML挑战赛系列2015–2018分析》，载于：《自动化机器学习》，Springer，2019年，第177–219页。[70] C. Ying、A. Klein、E. Christiansen、E. Real、K. Murphy和F. Hutter，《NAS-Bench-101：迈向可复现的神经架构搜索》，载于：国际机器学习会议，2019年，第7105–7114页。[71] M. Feurer、A. Klein、K. Eggensperger、J. Springenberg、M. Blum和F. Hutter，《高效且鲁棒的自动化机器学习》，载于：神经信息处理系统进展，2015年，第2962–2970页。[72] B. van Stein、H. Wang和T. B¨ack，《通过并行高效全局优化自动配置深度神经网络》，载于：2019年国际神经网络联合会议（IJCNN），IEEE，2019年，第1–7页。[73] H. Jin、Q. Song和X. Hu，《Auto-Keras：高效的神经架构搜索系统》，载于：第25届ACM SIGKDD知识发现与数据挖掘国际会议论文集，ACM], 页码=1946—1956，年份=2019，。[74] K. Price、R.M. Storn和J.A. Lampinen，《差分进化：全局优化的实用方法》，施普林格科学与商业媒体，2006年。

[75] A. Krizhevsky, 《从微小图像中学习多层特征》（2009）。[76] G. Tsoumakas、I. Katakis 和 I. Vlahavas, 《在标签数量庞大的领域中实现有效且高效的多标签分类》, 载于：ECML/PKDD多维数据挖掘研讨会论文集，比利时安特卫普，MMD08，2008年，第30–44页。[77] H. Xiao、K. Rasul 和 R. Vollgraf, 《Fashion-MNIST：用于机器学习算法基准测试的新型图像数据集》, 2017年。[78] I.W. Evett 和 E.J. Spiehler, 《法医学中的规则归纳》, KBS in Goverment（1987），107–118。[79] V.G. Sigillito、S.P. Wing、L.V. Hutton 和 K.B. Baker, 《使用神经网络对来自电离层的雷达回波进行分类》，约翰斯·霍普金斯APL技术文摘 10(3)（1989），262–266。[80] L. Deng, 《用于机器学习研究的手写数字图像MNIST数据库》 [网络精选], IEEE信号处理杂志 29(6)（2012），141–142。[81] M. Buscema, 《Metanet*：独立判断者理论》, 物质使用与滥用 33(2)（1998），439–461。[82] R.P. Gorman 和 T.J. Sejnowski, 《对经训练用于分类声纳目标的分层网络中隐藏单元的分析》, 神经网络 1(1)（1988），75–89。[83] L.A. Kurgan、K.J. Cios、R. Tadeusiewicz、M. Ogiela 和 L.S. Goodenday, 《用于自动心脏SPECT诊断的知识发现方法》, 医学人工智能 23(2)（2001），149–169。[84] J.D. Castro, 《Arch Linux》, 载于：《Linux发行版导论》, Springer，2016年，第235–252页。[85] M. Garland、S. Le Grand、J. Nickolls、J. Anderson、J. Hardwick、S. Morton、E. Phillips、Y. Zhang 和 V. Volkov, 《CUDA并行计算实践》, IEEE Micro 28(4)（2008），13–27。[86] S. Chetlur、C. Woolley、P. Vandermersch、J. Cohen、J. Tran、B. Catanzaro 和 E. Shelhamer, 《cuDNN：用于深度学习的高效原语》, arXiv预印本 arXiv:1410.0759（2014）。[87] M. Abadi、P. Barham、J. Chen、Z. Chen、A. Davis、J. Dean、M. Devin、S. Ghemawat、G. Irving、M. Isard 等, 《TensorFlow：一种面向大规模机器学习的系统》, 载于：第12届 {USENIX} 操作系统设计与实现研讨会（{OSDI} *16*），2016年，第265–283页。[88] F. Chollet 等, 《Keras：用于Theano和TensorFlow的深度学习库》, 网址：https://keras. io/k 7(8)（2015），T1。[89] D. Charte、F. Herrera 和 F. Charte, 《Ruta：R语言中的神经自编码器实现》, Knowledge-Based Systems 174（2019），4–8。[90] T. Tieleman 和 G. Hinton, 《第6.5讲——RMSProp：用近期幅度的运行平均值除梯度》, Coursera：机器学习神经网络 4(2)（2012），26–31。