

# 一种训练深度神经网络的梯度引导进化方法

王泽毅

## 摘要

神经网络的权值对分类性能至关重要，而神经网络的高复杂性对给定数据集的最优权值提出了严峻挑战。虽然一系列基于梯度的方法已经被开发出来，但很容易陷入局部最优且对超参数十分敏感。进化算法由于其较高的鲁棒性和广泛的适用性，近年来被认为是训练神经网络的一种很有前途的替代方法。然而，进化算法遭受着维度的诅咒，并且在训练神经网络时效率低下。通过结合基于梯度的方法和进化算法的优势，提出了一种梯度引导的进化方法来训练神经网络。该方法提出了一种新的遗传算子来优化搜索空间中的权值，其中搜索方向由权值的梯度决定。此外，该方法考虑了网络稀疏性，大大降低了网络复杂度，缓解了过拟合。

**关键词：**深度神经网络、进化算法、遗传算子、梯度下降、稀疏性。

## 1 引言

深度神经网络（Deep neural networks, DNNs）已广泛应用于图像处理和计算机视觉、自然语言处理和目标检测等多个领域。DNNs 中的权值设定对分类性能具有决定性的作用，然而 DNNs 的高复杂性对于获得给定数据集的最优权值提出了严峻的挑战。在过去的几十年里，人们开发了一些优化方法来训练神经网络，其中大多数方法都是根据梯度信息来优化权值的，如随机梯度下降（stochastic gradient descent, SGD）<sup>[1]</sup>和 Adam 等。

梯度信息提供的快速收敛速度，使得基于梯度的方法在训练深度神经网络中是有前途的。然而，梯度方法存在一些局限性。例如，基于梯度的方法容易陷入局部最优点或鞍点，有时还需要引入正则项来缓解过拟合。此外，一些参数如学习率、动量和下降率等，需要预定义一个适合的值。

进化算法（Evolutionary algorithms, EAs）作为一组具有种群进化特征的元启发式算法，在解决不同研究领域的许多复杂优化问题方面显示出了有效性。与基于梯度的方法相比，基于种群的特性使得 EAs 具有优异的搜索能力，且对局部最优解不敏感。因此，自 20 世纪 80 年代以来，许多 EAs 被建议用于训练 DNNs<sup>[2]</sup>。

但是，EAs 在优化大规模 DNNs 的权重上具有较差的可扩展性，因为维数的增加使得 EAs 在计算上承受巨大的压力。为了解决维数的诅咒，人们试图提高 EAs 的可扩展性来训练大规模神经网络，特别是 DNN 网络。公等人<sup>[3]</sup>提出了一种双目标 EA，只考虑稀疏 DNN 的权重，而不是所有的权重，因为偏差是控制隐藏层稀疏性的主要因素。孙等人<sup>[2]</sup>开发了一种单目标 EA 来优化 DNNs 的权值，其中权值优化转换为权值空间中正交偏置向量的优化。崔等人<sup>[4]</sup>提出了一种通过 SGD 和 EA 交替优化权重的进化方法，其中 EA 可以提高 SGD 的稳定性，保证训练损失的稳定减少。在已有的训练大规模神经网络的进化方法中，大多数都是通过减少搜索空间（即需要优化的权重数）来缓解维数的压力，但这可能会错过最优搜索区域，增加陷入局部最优的概率。

本次课程的论文复现工作提出了一种梯度引导的进化方法来训练深度神经网络，旨在继承梯度下降和进化算法的优势，弥补现有进化方法的不足。首先，提出了一种基于梯度的模拟二值交叉（simulated

binary crossover, SBX) 算子, 称为 gSBX, 用于优化 DNNs 的权值。然后, 提出了一种梯度引导的进化多目标方法来训练 DNNs, 称为 GEMONN。

## 2 相关工作

### 2.1 多目标神经网络训练

给定一个训练集  $D$  和一个神经网络, 进化算法通常通过最小化以下双目标优化问题来搜索权值的最优值<sup>[5]</sup>:

$$\min_x F(x) = \begin{cases} f_1(x) = f_{complexity}(net, x) \\ f_2(x) = f_{loss}(net, D, x) \end{cases} \quad (1)$$

$x$  是所有的权重优化,  $f_1(x)$  表示权重为  $x$  的网络的复杂度,  $f_2(x)$  表示  $D$  中所有样本的净损失训练平均值。在实际应用中, 网络复杂度可以通过非零权值的个数或绝对权值的和来度量, 而训练损失可以是分类误差或重构误差<sup>[6]</sup>。此外, 由于每个解都包含两个目标值, 因此解的质量通常由帕累托支配关系决定。具体来说, 对于两个解  $x_1, x_2 \in \Omega$ , 如果  $x_1$  在所有目标中大于或等于  $x_2$ , 并且在至少一个目标中严格优于  $x_2$ , 则认为  $x_1$  主导  $x_2$ , 具体条件表示为:

$$\begin{cases} \forall i = 1, 2, 3, \dots, m & f_i(x_1) \leq f_i(x_2) \\ \exists j = 1, 2, 3, \dots, m & f_j(x_1) < f_j(x_2) \end{cases} \quad (2)$$

如果一个解不被任何其他解所支配, 则称为帕累托最优解。所有的帕累托最优解构成了帕累托最优集, 帕累托最优集在目标空间中的投影称为帕累托最优前沿。因此, 多目标神经网络训练的目标是找到一组近似于帕累托最优前沿的解, 它可以在网络复杂性和训练损失之间进行权衡。

### 2.2 进化算法

进化算法是在遗传算法的基础上发展演变而来的, 都具有基于种群的特性。在实际问题中, 不同目标之间经常会出现冲突约束, 所以多目标优化问题通常没有单一的最优解。基于种群的特性可以让算法在一次运行过程中得到多个解, 经过多次迭代, 最终能得到一组包含多个最优解的解集, 因此多目标进化算法在解决多目标优化问题时非常受欢迎。多目标进化算法的主要思想和遗传算法相同, 通过生物进化的规则, 引导个体向适应环境的方向进化发展。在进化算法中, 首先将亲本个体的值转换成二进制的形式, 用来表示该个体的基因。随后在种群内选择个体进行基因的交叉和变异, 产生新的子代个体并构成子代种群。然后从子代种群中寻找最适应环境的个体作为下一代的亲本个体。通过不断迭代循环, 最终得到最适应环境的种群, 也即最符合条件的解集。

进化算法是基于种群的算法, 因此其算法流程类似于遗传算法, 通过生成初代种群, 并对种群进行操作, 得到下一代种群。经过多代循环, 得到最终的结果。图 1 描述了进化算法的算法流程图。

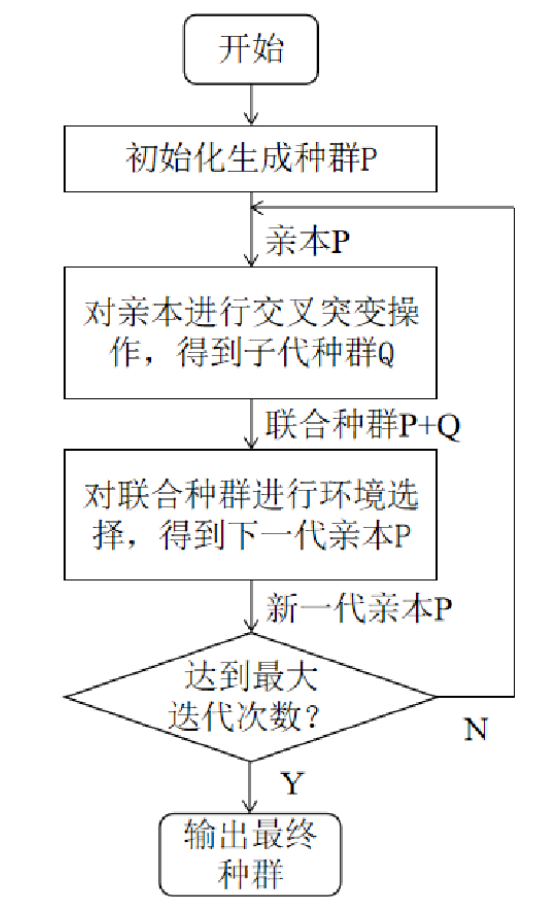


图 1: 方法示意图

### 2.3 训练神经网络的进化算法

自 20 世纪 90 年代以来，进化算法尝试了一系列训练网络的工作。在这些方法中，主要的问题之一是提高进化算法在处理大规模神经网络时的可扩展性，因为进化算法在大规模优化<sup>[7]</sup>中通常遭受维数诅咒。因此，有人提出了各种方法来解决这一问题。

Stanley 提出了一种混合编码方法来同时优化神经网络的结构和权重<sup>[8]</sup>。具体地说，网络结构和节点之间的权值用一个可变长度的向量表示，并采用遗传算法，利用两个定制的遗传算子进化出一组解。然而，由于结构和权值的显式编码，每个解的长度将显著增加，这使得算法无法优化低训练损失的权值。

吴等人<sup>[9]</sup>通过基于分解的多目标进化算法<sup>[10]</sup>和两种新的遗传算子优化了神经网络的权值和连接结构。该方法将权值和连接结构编码在混合决策向量中，混合决策向量通过基于分解的多目标进化算法进行优化，并配备定制的交叉算子和突变算子。这种方法在训练大约 6000 个权重的大规模神经网络时表现出更好的可伸缩性。然而，在优化更大规模的神经网络的效率还需要进一步提高。

公等人设计了一种基于进化算法的诱导学习方法来优化进化算法的大规模权值<sup>[3]</sup>。除了最小化重构误差外，还考虑了隐藏单元的稀疏性，以获得更好的泛化。更重要的是，在基于梯度的局部搜索策略的帮助下，进化算法只优化了偏差向量，以直接控制稀疏性。因此，权值优化被简化为一个小规模的优化问题，并且很容易被解决。由于进化算法可以保证种群中最佳的适应度不会退化，因此种群可以稳定地进化以实现更好的收敛。

刘等人<sup>[6]</sup>提出了一种基于进化算法的分层结构学习方法来优化权重的连接。为了减少搜索空间，每个权重的值首先被限制为 0 或 1。然后，设计了一个多阶段初始化策略，在不同的阶段中不同程度

地减少搜索空间，其中多个权值根据与输入层的邻接关系共享相同的值。这种方法在训练深度神经网络中具有良好的可伸缩性。

朱等人设计了一种基于进化算法的超参数优化方法来搜索神经网络的最优结构<sup>[11]</sup>。为了提高大规模神经网络进化的可伸缩性，提出了一种改进的稀疏进化训练策略，以避免神经网络在最后一步更新最小的部分权值。通过将修改后的稀疏进化策略编码为两个超参数，可以通过基于定制的多目标算法优化所有超参数来实现整个学习过程。尽管网络在神经网络上有良好的性能，但网络的训练过程主要依赖于基于梯度的方法。

Koutnik 等人提出了一种基于傅里叶变换的进化算法来优化大规模神经网络的权重，用于基于视觉的强化学习<sup>[12]</sup>。该方法用一个傅里叶型系数序列来表示权值，它是可变长度的，并且比权值向量要短得多。总之，该方法还将大规模神经网络的训练转化为小规模优化问题。

Real 等人使用进化算法从平凡的初始结构演化出神经网络的整个结构。为了发现更多不同的模型，我们为他们的方设计了一个相当不受限制的搜索空间，没有固定的深度和任意的跳过连接。配备了定制的突变操作符，可以基于亲本模型生成更有前途的子代模型。虽然这种方法只是优化了网络的结构，但为进化算法与网络的结合提供了广阔的前景。

最近，孙等人提出了一种创新的进化方法，用于训练无监督的深度神经网络<sup>[2]</sup>。这种方法通过用一组正交向量的加权和来表示大量的权值来减少搜索空间，其中这些转换变量的数量只是权值数量的平方根。此外，其他超参数也与权值一起进行了优化。实验结果表明，该方法在训练具有数百万个权重的深度神经网络上优于最先进的方法。

## 2.4 进化算法的局限

尽管进化方法在训练神经网络方面具有良好的性能，但大多数进化方法在优化深度神经网络中的大量权重时遇到了困难。这些方法的局限性主要是由于采用了降维策略，它在一定程度上解决了降维策略的诅咒，但更容易陷入局部最优。然而，由于进化算法在大规模优化中的可搜索能力的限制，这些方法必须减少优化权重的数量。此外，还有一些方法直接将进化算法和梯度下降结合在原始搜索空间中进行搜索，其中权值通过传统的遗传算子和梯度下降进行连续更新。然而，进化算法在大规模优化问题上性能不佳的障碍尚未得到克服。换句话说，这些方法中的进化算法仅仅保证了种群中的最佳适应度不会下降，而梯度下降主要负责在原始搜索空间中寻找更好的解。

为了更好地说明梯度下降对遗传算子的影响，图 2 介绍了遗传算子在加入梯度下降前后的差异。具体来说，图 2 中以遗传算子 SBX 为例，得到了亲本和后代解的分布，其中，亲本个体由三角形符号表示，后代个体由圆形符号和正方形符号表示。可以发现，生成的后代解分布在亲本的两侧，因为 SBX 随机决定搜索方向，可能探索更多未知的区域。因此，SBX 做出了许多不必要的搜索工作，而在高维搜索空间中优化深度神经网络的权重时将变得更加严重。

在加入梯度下降后，由于搜索方向总是与  $x$  的梯度相反，此时后代解只有圆形符号表示的个体，它具有更好的利用能力。因此，在遗传算子中引入梯度信息可以限制未来搜索的方向，从而大大减少不必要方向上的搜索努力。同时，生成多个子代解可以大大降低进入局部最优的概率，提高梯度下降的探索能力，而且消除了学习率的设置。

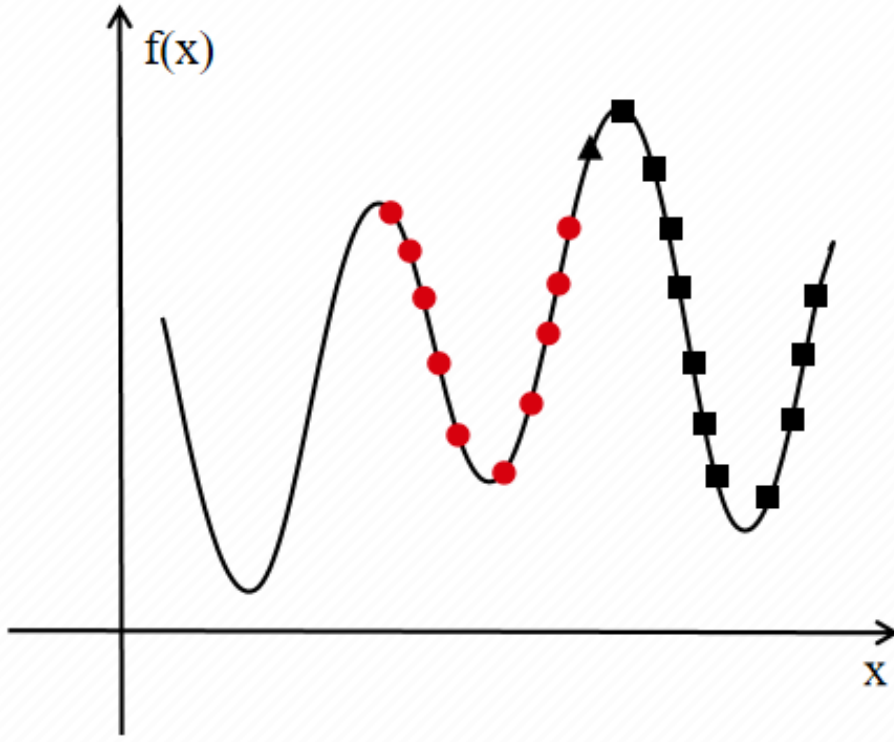


图 2: 方法示意图

基于这一思想，通过设计一个基于梯度的遗传算子来训练深度神经网络，使进化算法能够在不减少搜索空间的情况下优化深度神经网络的权重。更重要的是，现有的方法通常是针对特定类型的深度神经网络，而本方法旨在实现一种训练不同类型的深度神经网络的通用方法。

### 3 本文方法

#### 3.1 gSBX 算子

近年来，有许多交叉算子被提出，如 SBX 和 DE。在这些交叉算子中，SBX 在大多数测试问题上表现出更好的性能。因此，SBX 经常被作为基本的遗传算子。形式上，给定两个父本  $x_1$  和  $x_2$ ，带有  $n$  个决策变量，SBX 生成两个子代解  $c_1$  和  $c_2$ 。

$$\begin{cases} c_1^i = [(1 + \beta)x_1^i + (1 - \beta)x_2^i]/2 \\ c_2^i = [(1 - \beta)x_1^i + (1 + \beta)x_2^i]/2 \end{cases} \quad 1 \leq i \leq n \quad (3)$$

其中  $x_1^i$  表示决策向量  $x_1$  的第  $i$  个变量， $\beta$  是一个随机值，需要为每个维度重新生成。

$$\beta = \begin{cases} \frac{1}{(2\mu)^{\eta+1}} & \mu \leq 0.5 \\ \frac{-1}{(2(1-\mu))^{\eta+1}} & \mu > 0.5 \end{cases} \quad (4)$$

其中  $\mu$  是  $[0,1]$  中采样的均匀分布随机值， $\eta$  是控制子代解分布的参数。其中， $\eta$  的值建议为 20。

1)、基于梯度的 SBX (gSBX) 的核心思想：将公式 (3) 变形，可以重新表述为公式 (5)：

$$\begin{cases} c_1^i = x_1^i + \alpha_1(x_1^i - x_2^i)/2 \\ c_2^i = x_2^i - \alpha_2(x_1^i - x_2^i)/2 \end{cases} \quad 1 \leq i \leq n \quad (5)$$

$$\alpha_{1,2} = \beta - 1 \quad (6)$$

为了将梯度  $g_1$  和  $g_2$  纳入遗传算子，可以很容易地得到以下结论：

$$\begin{cases} c_{1,2}^i < x_{1,2}^i, g_{1,2}^i > 0 \\ c_{1,2}^i \geq x_{1,2}^i, g_{1,2}^i \leq 0 \end{cases} \quad 1 \leq i \leq n \quad (7)$$

其中，其中， $c_{1,2}$  代表  $c_1$  或  $c_2$ ， $x_{1,2}$  代表  $x_1$  或  $x_2$ ， $g_{1,2}$  代表  $g_1$  或  $g_2$ 。为此，根据公式 (5)， $\alpha_1(x_1^i - x_2^i)$  应该与  $g_1^i$  有不同的符号，而  $\alpha_2(x_1^i - x_2^i)$  应该与  $g_2^i$  有相同的符号。由于  $(x_1^i - x_2^i)$  是确定的值，因此  $g_1^i$  和  $g_2^i$  的符号可以按照预期进行变化，这由根据公式 (4) 和公式 (6) 的  $\mu$  来控制。因此，所提出的 gSBX 算子通过以下方式设置  $\mu$ ，将梯度信息整合到遗传操作符中：

$$\mu_1 = \begin{cases} \text{sample in } [0, 0.5] & g_1^i(x_1^i - x_2^i) > 0 \\ \text{sample in } (0.5, 1] & g_2^i(x_1^i - x_2^i) < 0 \\ \text{sample in } [0, 1] & \text{otherwise} \end{cases} \quad (8)$$

$$\mu_2 = \begin{cases} \text{sample in } [0, 0.5] & g_1^i(x_1^i - x_2^i) < 0 \\ \text{sample in } (0.5, 1] & g_2^i(x_1^i - x_2^i) > 0 \\ \text{sample in } [0, 1] & \text{otherwise} \end{cases} \quad (9)$$

随后， $\mu_1$  和  $\mu_2$  分别通过公式 (4) 和公式 (6) 为每个维度重新生成生成  $\alpha_1$  和  $\alpha_2$ 。此外，如果梯度等于零或不可用，则在原始区间  $[0,1]$  中生成  $\mu_1$  和  $\mu_2$ ，这与原始 SBX 相同。

2)、gSBX 中的统一交叉：在 gSBX 中也采用了统一交叉的思想，这已在许多现有的遗传操作符中用于提高多样性<sup>[7]</sup>。具体来说，均匀交叉交换两个后代解  $c_1$  和  $c_2$  的每个维上的变量的概率为 0.5。根据公式 (6)，我们有：

$$\begin{aligned} c_1^i &= x_1^i + \alpha_1(x_1^i - x_2^i)/2 \\ &= x_1^i + (\beta - 1)(x_1^i - x_2^i)/2 \\ &= (1 - \beta)x_2^i/2 + (1 + \beta)x_1^i/2 \\ &= x_2^i - (-\beta - 1)(x_1^i - x_2^i)/2 \end{aligned}$$

因此，如果我们设置为  $\alpha_2 = -\beta - 1$ ，则可以将  $c_2^i$  更改为  $c_1^i$ 。类似地，如果我们设置为  $\alpha_1 = -\beta - 1$ ，则  $c_1^i$  可以更改为  $c_2^i$ 。

在交换  $c_1^i$  和  $c_2^i$  时，还需要考虑另一件事。假设梯度  $g_1^i > 0$ ，当  $x_1^i < x_2^i$  时，将  $c_1^i$  更改为  $c_2^i$  是不合理的，因为  $x_1^i$  应该沿着  $g_1^i$  的负方向移动。因此，应该施加一些条件来限制均匀交叉，即只有当  $g_1^i(x_1^i - x_2^i) > 0$  和  $g_2^i(x_1^i - x_2^i) < 0$  时，才能进行均匀交叉。综上所述，所提出的 gSBX 将  $\alpha_1$  和  $\alpha_2$  设为

$$\alpha_{1,2} = \begin{cases} -\beta - 1 & \lambda \leq 0.5 \text{ and } g_1^i(x_1^i - x_2^i) > 0 \text{ and } g_2^i(x_1^i - x_2^i) < 0 \\ \beta - 1 & \text{otherwise} \end{cases} \quad (10)$$

其中， $\lambda$  是从每个维度的均匀分布  $U[0,1]$  中采样的。

3)、考虑 gSBX 中的稀疏性：为了更好地控制网络的稀疏性，在 gSBX 中保持了后代解的稀疏性，其中一个直观的想法是使后代解比它们的父母具有相同或更高的稀疏性。为此，如果相应的父变量的变量为零，则所提出的 gSBX 将子代解的每个变量设为零。因此，子代解中的零权值的数量总是等于或大于父代解中的零权值的数量。根据公式 (5)，如果我们设置为  $\alpha_1 = 0$ ，则可以保持  $c_1^i = x_1^i = 0$ 。因

此，所提出的 gSBX 进一步完善了  $\alpha_1$  和  $\alpha_2$

$$\alpha_{1,2} = \begin{cases} 0 & x_{1,2}^i = 0 \\ \alpha_{1,2} & otherwise \end{cases} \quad (11)$$

### 3.2 基于梯度引导的进化算法

在算法 1 中给出了 gSBX 的详细过程。可以看出，从种群  $P$  中随机选择两个亲本  $x_1$  和  $x_2$ ，用它们生成两个子代解  $c_1$  和  $c_2$ 。对每个维度，依次确定  $\mu_1$ 、 $\mu_2$ 、 $\beta$ 、 $\alpha_1$ 、 $\alpha_2$  的值，并据此计算决策变量  $c_1^i$  和  $c_2^i$ 。重复这个过程，直到  $P$  中的所有解都被访问，并且所有新生成的解都作为子代种群返回。

---

#### Procedure 1 gSBX(P)

---

**Input:** Parents  $P$

**Output:** Offspring population  $O$

$O \leftarrow \emptyset$ ;

**while**  $P \neq \emptyset$  **do**

    Randomly select two solutions  $x_1$  and  $x_2$  from  $P$ ;

$P \leftarrow P \setminus \{x_1, x_2\}$

**for**  $i$  **in**  $1$  **to**  $n$  **do**

        Sample  $\mu_1$  and  $\mu_2$  by (8) and (9);

        Calculate  $\beta_1$  and  $\beta_2$  by (4);

        Calculate  $\alpha_1$  and  $\alpha_2$  by (10);

        Repair  $\alpha_1$  and  $\alpha_2$  by (11);

        Calculate  $c_1^i$  and  $c_2^i$  by (5);

**end**

$O \leftarrow O \cup \{c_1, c_2\}$

**end**

---

算法 2 给出了所提出的 GEMONN 的过程。首先，创建一个具有预定义大小的总体，其中每个初始解的决策变量（即权重）被设置为  $[-1,1]$  或  $0$  内的随机值。然后，计算出每个解的目标值，并计算出训练损失方面的梯度。在每一代 GEMONN 中，通过 NSGA-II<sup>[13]</sup> 的交配选择策略从种群中选择一些解，并通过所提出的 gSBX 生成相同数量的子代解。然后，将后代种群与原种群结合，通过 NSGA-II 的环境选择策略，保留组合种群中一半的解用于下一代。在进化结束后，应该从最终的种群中选择一个单一的解作为输出。为此，提出了一种帕累托最优解更新策略和一种最终解选择策略。下面，我们分别详细介绍了 GEMONN 的主要组成部分，包括方案评估、帕累托最优方案更新和最终方案选择。

1)、质量评价：给定一个训练集  $D=(s_i, y_i)_{i=1}^N$  和一个 DNN，所提出的 GEMONN 为每个解  $x$  计算以下两个目标：

$$\min_x F(x) = \begin{cases} f_1(x) = \|x\|_0 \\ f_2(x) = \frac{1}{N} \sum_{i=1}^N L(\hat{y}_i) \end{cases} \quad (12)$$

其中，决策向量  $x$  表示网络中所有权重值的集合， $\hat{y}$  为权重为  $x$  时网络的输出。第一个目标  $f_1$  表示 DNN 的复杂性，其中较小的  $f_1$  表示更稀疏的网络。这里，采用  $l_0$ -norm 来度量复杂性，这被验证是有效的，因为它直接计算非零权重<sup>[7]</sup>。第二个目标  $f_2$  表示所有训练样本的平均训练损失，其中有监督学习任务采用交叉熵（cross entropy, CE）损失，无监督学习任务采用均方误差（mean squared error, MSE）<sup>[5]</sup>。

具体来说,  $f_2$  被定义为

$$f_2(x) = \begin{cases} L_{CE}(\hat{y}_i) = - \sum_j^l y_i^j \log(\hat{y}_i^j) \\ L_{MSE}(\hat{y}_i) = \frac{1}{m} \sum_j^m (s_i^j - \hat{y}_i^j)^2 \end{cases} \quad (13)$$

式中,  $y_i^j$  表示第  $i$  个样本的第  $j$  个标签,  $s_i^j$  表示第  $i$  个样本的第  $j$  个特征,  $l$  为标签总数,  $m$  为特征总数。由于网络只通过最小化训练损失来训练集可能导致过拟合, 我们考虑网络的复杂性来缓解过拟合。直观地说, 这两个目标  $f_1$  和  $f_2$  是相互冲突的, 因为一个训练损失较低的网络通常对应于较高的复杂度。然而, 由于进化多目标优化提供了微妙的选择策略, 可以找到一组在两个目标之间进行良好权衡的解, 并期望得到一个有希望的训练损失但不过拟合训练集的网络。

2)、帕累托最优解更新: 演化结束后, 采用帕累托最优解更新策略进行最终开发。具体来说, 去除种群中的主导解, 然后用 SGD 对每个非主导解进行微调, 一步学习率为 0.1。值得注意的是, 这里只更新了非零的权值, 而为零的权值保持不变。这种技巧可以减少训练的损失, 但不会失去网络的稀疏性, 因此我们可以称之为稀疏 SGD (SSGD)。

3)、最终解选择: 最后, 从总体中选择一个解输出。直观地说, 选择训练损失或复杂度最低的解是不合理的, 因为前者往往会过拟合, 而后者通常没有得到很好的训练。由于膝关节区域的解在两个目标<sup>[14]</sup>之间有更好的权衡, 它们通常在许多情况下中是首选<sup>[6]</sup>。基于这一想法, 所提出的 GEMONN 根据以下标准选择了一个解:

$$\begin{aligned} X_1 &= \{x \mid \int (\log_{10} f_2(x)) - \int (\log_{10} f_2(x^+)) < 1\} \\ X_2 &= \{x \mid f_2(x) < 2 * f_2(x^+), x \in X_1\} \\ p &= \arg \min_x \gamma(x), x \in X_2 \end{aligned} \quad (14)$$

第一个准则的目的是找到训练损失与  $x^+$  具有相同数量级的解, 其中  $x^+$  是总体中训练损失最低的解, 即  $x^+ = \arg \min_{x \in P} f_2(x)$ 。第二个标准也是保证所选解的训练性能。第三个准则根据  $\gamma(x)$  求出最终解, 计算解<sup>[15]</sup>的左右邻域之间的夹角。

## 4 复现细节

### 4.1 与已有开源代码对比

复现论文作者提供了参考代码进行学习, 因此直接从作者的 github 仓库中下载了 GEMONN 算法的实现代码。图 3 描述了提供的源代码结构图。



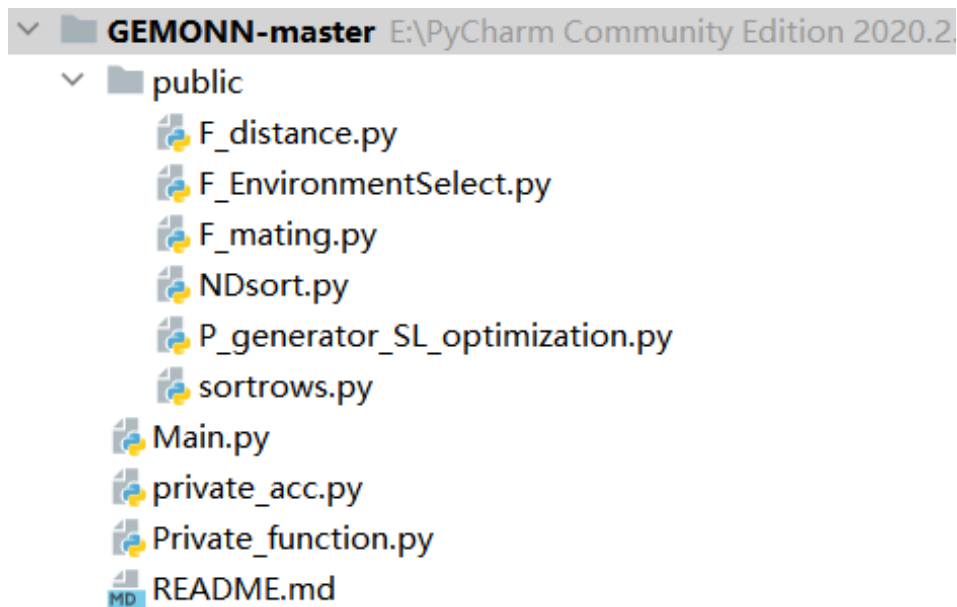


图 3: 源代码结构图

其中，private\_acc.py 和 Private\_function.py 提供了进化算法检验神经网络的测试方法，此处选择了 Private\_function.py 进行参考引用，并对其进行了修改。public 文件夹下放置了进化算法的基本操作，包括非主导排序方法 NDsort.py、交配池的构建方法 F\_mating.py，拥挤记录的计算方法 F\_distance.py，环境选择策略 F\_EnvironmentSelect.py，gSBX 算子的构造方法 P\_generator\_SL\_optimization.py 等。在复现过程，将进化算法的这些操作集成到一个文件中，并对其进行了部分修改。图 4 展示了修改后的代码结构图。

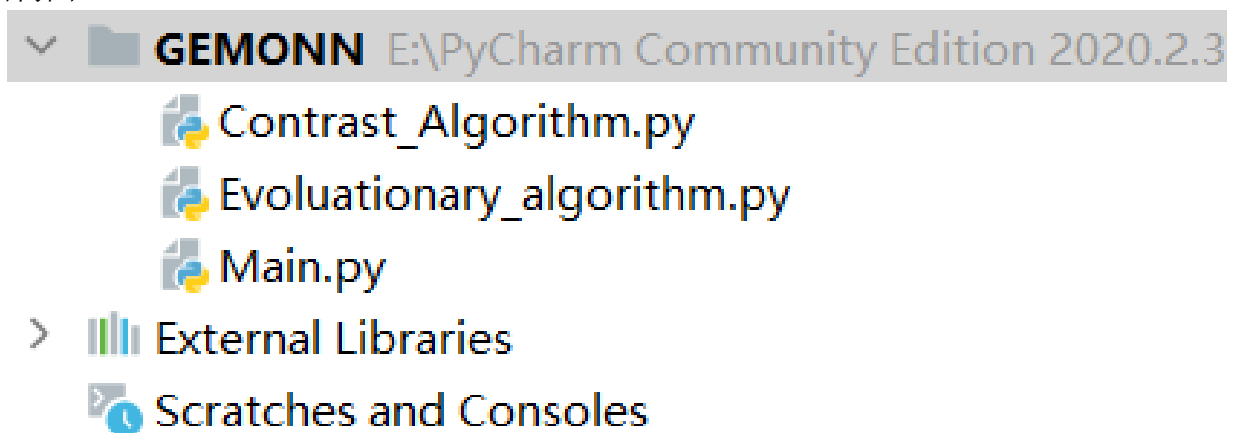


图 4: 源代码结构图

## 4.2 实验环境搭建

硬件要求：装有 Windows10 系统的计算机一台，服务器。

软件要求：Pycharm，MobaXterm

## 4.3 创新点

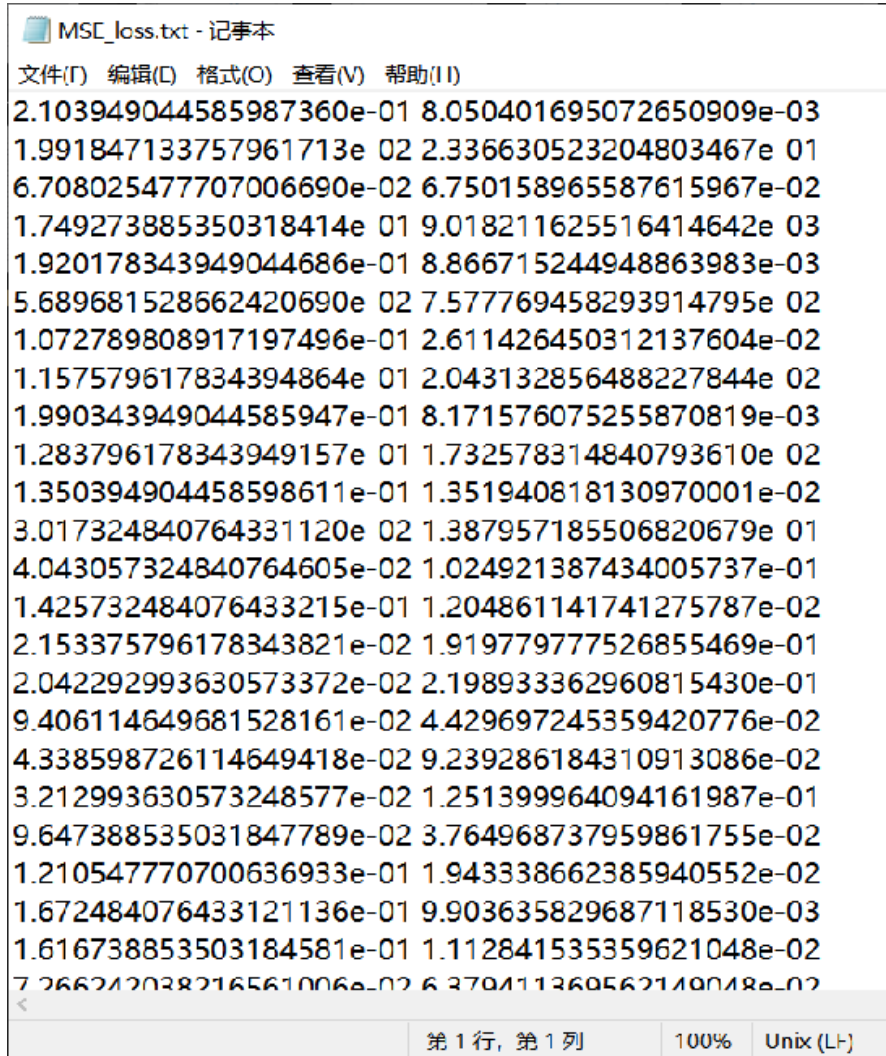
本次复现结合了作者提供的源代码进行复现，大致上实现了 GEMONN 的功能。代码上与作者提供的源代码对比，缺少了许多算法细节。在这一基础上，对代码进行了完善，从而大致实现了与源代码类似的结果。此外，在代码大致复现完成后，对其中的一些参数范围进行了调整尝试，如  $\mu_1$  和  $\mu_2$  的取值。但尝试的实验结果远不如 GEMONN 得到的实验结果，因此无理论背景支撑的无意义的调整并不能提升算法的性能。

## 5 实验结果分析

在自动编码器（Autoencoder, AE）上验证了所提出的 GEMONN 的性能。下面简要介绍 AE。

作为一个完全连接（fully connected, FC）且带有隐藏层的前馈神经网络，AE 常用于降维，其中期望输出与输入相同，而隐藏层可以被视为训练样本的一个有意义的和简化的表示。因此，AE 可以以无监督的方式进行训练，并采用了许多现有的进化方法对其进行训练<sup>[6]</sup>。

为了使用 GEMONN 算法训练 AE，选择了 MNIST 数据集测试，采用 MSE 和  $l_0 - norm$  范数作为两个目标。此外，还对隐藏层应用了 s 型激活函数。实验结果在下图中展示。图 5 展示了复现代码得到的种群中个体对应的 MSE 和稀疏性。



MSE_loss.txt - 记事本	
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)	
2.103949044585987360e-01	8.050401695072650909e-03
1.991847133757961713e-02	2.336630523204803467e-01
6.708025477707006690e-02	6.750158965587615967e-02
1.749273885350318414e-01	9.018211625516414642e-03
1.920178343949044686e-01	8.866715244948863983e-03
5.689681528662420690e-02	7.577769458293914795e-02
1.072789808917197496e-01	2.611426450312137604e-02
1.157579617834394864e-01	2.043132856488227844e-02
1.990343949044585947e-01	8.171576075255870819e-03
1.283796178343949157e-01	1.732578314840793610e-02
1.350394904458598611e-01	1.351940818130970001e-02
3.017324840764331120e-02	1.387957185506820679e-01
4.043057324840764605e-02	1.024921387434005737e-01
1.425732484076433215e-01	1.204861141741275787e-02
2.153375796178343821e-02	1.919779777526855469e-01
2.042292993630573372e-02	2.198933362960815430e-01
9.406114649681528161e-02	4.429697245359420776e-02
4.338598726114649418e-02	9.239286184310913086e-02
3.212993630573248577e-02	1.251399964094161987e-01
9.647388535031847789e-02	3.764968737959861755e-02
1.210547770700636933e-01	1.943338662385940552e-02
1.672484076433121136e-01	9.903635829687118530e-03
1.616738853503184581e-01	1.112841535359621048e-02
7.266242038216561006e-02	6.370411360562140048e-02
<	
第 1 行, 第 1 列	100% Unix (LF)

图 5: 种群中个体对应的 MSE 和稀疏性

图 6 为复现代码在迭代过程中 MSE 和  $l_0 - norm$  范数的变化过程，其中 x 轴为 MSE，y 轴为  $l_0 - norm$ 。

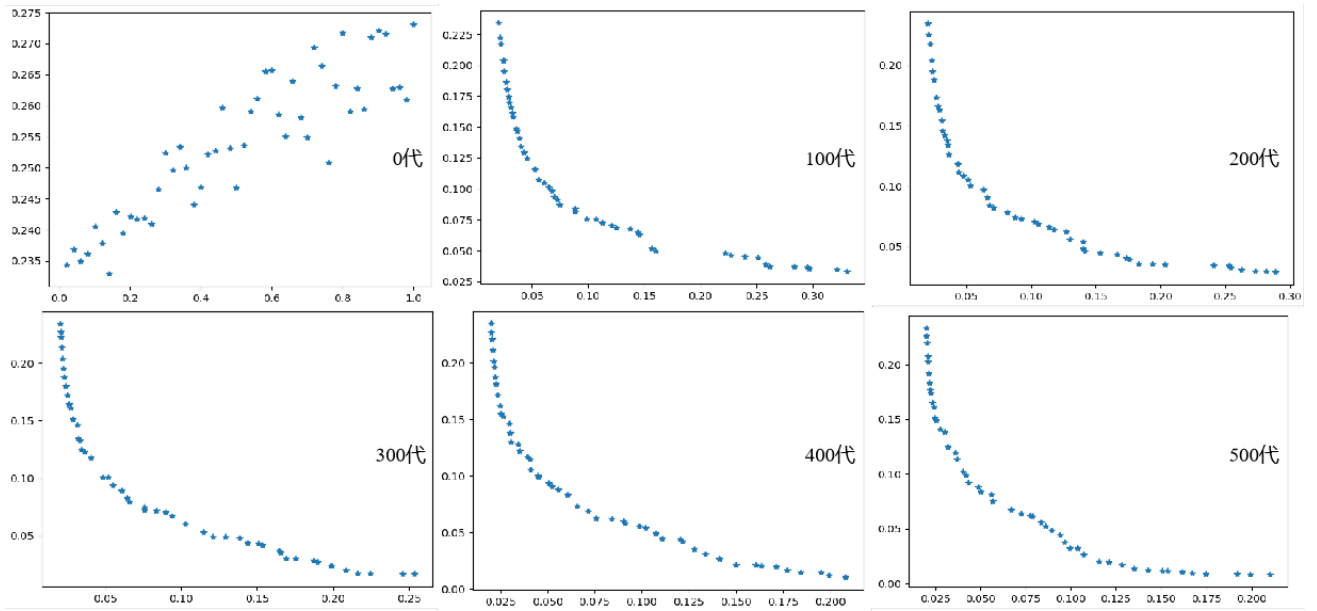


图 6: 种群迭代中 MSE 和  $l_0 - norm$  范数的变化过程

图 7 为源代码运行出的种群中 MSE 和  $l_0 - norm$  范数的变化过程，其中 x 轴为 MSE，y 轴为  $l_0 - norm$ 。可以看出，复现的结果基本达到了源代码的算法效果。

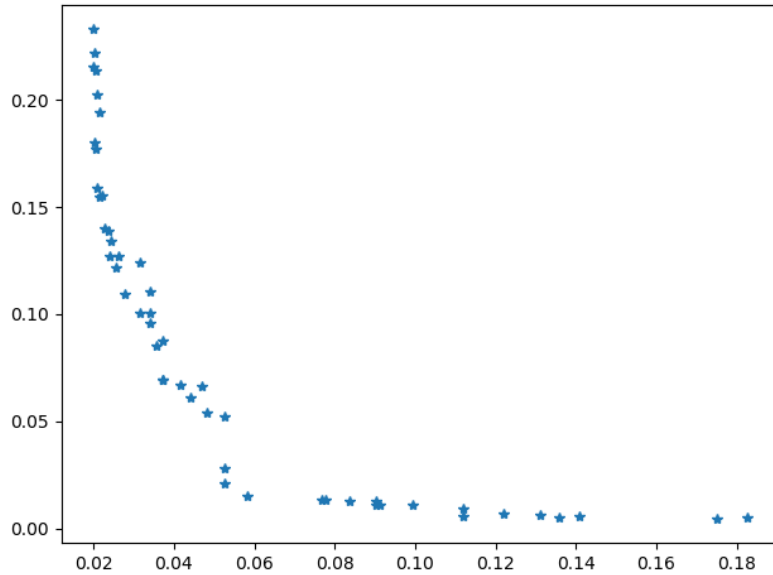


图 7: 源代码所得最终种群 MSE 和  $l_0 - norm$  范数

## 6 总结与展望

复现论文提出了一种梯度引导的进化方法来训练 DNNs，所提出的 GEMONN 设计了一个强大的遗传算子 gSBX，通过利用梯度信息来直接优化 DNNs 的权值。同时，对网络稀疏性和训练损失进行了优化，降低了网络复杂度，缓解了过拟合。此外，通过实验可以证明 GEMONN 训练的神经网络比基于梯度的方法具有更好的稀疏性，不仅可以缓解过拟合，而且有利于许多深度学习任务，可以认为所提出的 GEMONN 是一种很有前途的、有效的训练方法。

在本次复现中，尝试了参考源代码进行复现，并对其进行了简单的修改。从中可以体会到研究的困难之处。在前人的基础上进行创新需要不断丰富自己，借由自身的知识底蕴来对前人的工作进行总结，发现其可提升之处，并付诸实验，通过不断地尝试才能真正做出科研成果。

## 参考文献

- [1] ROBBINS H, MONRO S. A Stochastic Approximation Method[J]. Annals of Mathematical Statistics, 1951, 22(03): 400-407.
- [2] SUN Y, YEN G G, YI Z. Evolving unsupervised deep neural networks for learning meaningful representations[J]. IEEE Transactions on Neural Networks and Learning Systems, 2019, 23(01): 89-103.
- [3] GONG M, LIU J, LI H, et al. A Multiobjective Sparse Feature Learning Model for Deep Neural Networks [J]. IEEE Transactions on Neural Networks and Learning Systems, 2015, 26(12): 3263-3277.
- [4] CUI X, WEI Z, TÜSKE Z, et al. Evolutionary Stochastic Gradient Descent for Optimization of Deep Neural Networks[J]. Advances in Neural Information Processing Systems, 2018.
- [5] JIN Y, SENDHOFF B. Pareto-Based Multiobjective Machine Learning: An Overview and Case Studies [J]. IEEE Transactions on Systems Man & Cybernetics Part C, 2008, 38(03): 397-415.
- [6] LIU J, GONG M, MIAO Q, et al. Structure Learning for Deep Neural Networks Based on Multiobjective Optimization[J]. IEEE Transactions on Neural Networks and Learning Systems, 2018, 29(06): 2450-2463.
- [7] TIAN Y, ZHANG X, WANG C, et al. An Evolutionary Algorithm for Large-Scale Sparse Multiobjective Optimization Problems[J]. IEEE Transactions on Evolutionary Computation, 2020, 24(02): 380-393.
- [8] STANLEY K O, MIKKULAINEN R. Evolving Neural Networks through Augmenting Topologies[J]. Evolutionary Computation, 2002, 10(02): 99-127.
- [9] WU Y, ZHANG Y, LIU X, et al. A Multiobjective Optimization-Based Sparse Extreme Learning Machine Algorithm[J]. Neurocomputing, 2018, 317: 88-100.
- [10] ZHANG Q, HUI L. MOEA/D: A Multiobjective Evolutionary Algorithm Based on Decomposition[J]. IEEE Transactions on Evolutionary Computation, 2008, 11(06): 712-731.
- [11] ZHU H, JIN Y. Multi-objective Evolutionary Federated Learning[J]. IEEE Transactions on Neural Networks and Learning Systems, 2020, 31(04): 1310-1322.
- [12] KOUTNÍK J, CUCCU G, SCHMIDHUBER J, et al. Evolving large-scale neural networks for vision-based reinforcement learning[J]. Conference on Genetic & Evolutionary Computation, 2013: 1061.
- [13] DEB K, PRATAP A, AGARWAL S, et al. A fast and elitist multiobjective genetic algorithm: NSGA-II [J]. IEEE Transactions on Evolutionary Computation, 2002, 06(02): 182-197.
- [14] RACHMAWATI L, SRINIVASAN D. Multiobjective Evolutionary Algorithm With Controllable Focus on the Knees of the Pareto Front[J]. IEEE Transactions on Evolutionary Computation, 2009, 13(04): 810-824.
- [15] BRANKE J, DEB K, DIEROLF H, et al. Finding knees in multi-objective optimization[J]. Parallel Problem Solving from Nature, 2004.