机器学习理论与算法
实验报告

姓名： 谭方舟

学号： 23B903077

算法： 遗传算法

时间： 2023 年 12 月 6 日

一、算法理论

1.1 算法概述

遗传算法（Genetic Algorithms, GA）是一种通过模拟自然选择和遗传学原理来解决优化和搜索问题的启发式算法。它最早由 John Holland 在 20 世纪 70 年代提出，并经过他的学生和其他研究员的进一步发展完善。遗传算法的灵感源自达尔文的自然选择理论，即“适者生存”（survival of the fittest），它认为适应环境的个体具有更高的生存和繁殖机会。

在遗传算法中，我们将问题的解决方案表示为一组个体，每个个体代表了问题可能的解。这些个体通常被称为染色体，它们由基因构成。基因是构成染色体的基本单位，代表了解决方案的不同部分。基因可以是二进制位、实数、符号或其他数据类型，取决于问题本身的特点。

遗传算法通过一系列操作来对种群进行迭代和演化，以产生新的解决方案。其中，选择、交叉和变异是最基本的操作。选择过程基于个体的适应度，适应度高的个体有更高的概率被选择为繁殖的父代。交叉操作模拟了生物的交配过程，通过随机交换两个个体的染色体片段来产生新的后代。变异操作以一定的概率随机改变染色体中的基因值，引入新的遗传变异。通过这些操作的组合，种群不断演化，生成新的解决方案。

遗传算法的迭代过程中，新生成的后代将替代部分或全部原始种群，使种群逐步向最优解靠近。迭代过程会持续进行，直到满足停止条件，如达到预定的迭代次数、找到满意的解或超出时间限制。

1.2 算法相关概念

遗传算法中的一些概念共同构成了其核心机制，通过对这些基本过程的不断重复和适当的参数调整，遗传算法能够在广泛的问题域中找到优秀的近似解决方案。其中的核心概念如下：

- 种群（Population）：种群由一系列个体组成，每个个体是问题可能解的表示。种群的大小可以根据问题的复杂性和搜索空间的大小来确定。
- 染色体（Chromosome）：染色体是个体的一个表现形式，代表了一个解决方案。通常，染色体由一串基因组成，它可以是二进制串、实数列表、排列或其他数据结构。
- 基因（Gene）：基因是构成染色体的组成部分，每个基因表示解决方案中的一个属性或参数。

- 适应度函数（Fitness Function）：适应度函数用于评价染色体的质量，即评价它表示的解决方案的好坏。函数的输出通常是一个数值，这个数值越高，表示对应的染色体越“好”。
- 选择（Selection）：选择过程模拟了自然选择过程，即根据适应度函数的结果来选择染色体进入下一代的繁殖群。目的是让适应度高的个体有更高的几率遗传到下一代。
- 交叉（Crossover）：交叉是遗传算法中的一个主要遗传操作，它模拟了生物的交配过程。这一过程通常涉及两个染色体的“父母”，并从中产生一个或多个“子代”染色体。交叉可以创建具有新遗传组合的后代，并增加种群的遗传多样性。
- 变异（Mutation）：变异是另一种遗传操作，它通过随机改变染色体中的一个或多个基因来引入新的遗传变异。变异可以帮助算法探索搜索空间的不同区域，避免早熟收敛，并维持种群的遗传多样性。
- 代（Generation）：在遗传算法中，通过一系列选择、交叉和变异操作生成的新一组个体称为一代。算法以迭代的方式运作，通过连续产生新一代来逐步优化解决方案。
- 编码（Encoding）：将解决方案从其原始表示转换为染色体的过程称为编码。常见的编码方式有二进制编码、实数编码、排列编码等。
- 解码（Decoding）：解码是编码的逆过程，它将染色体转换回问题的解决方案表示，以便适应度函数能够评价该染色体。
- 遗传操作（Genetic Operator）：遗传操作包括交叉、变异和有时的复制（Duplication）等过程，这些都是种群演化和产生新种群的机制。
- 锦标赛选择（Tournament Selection）、轮盘赌选择（Roulette Wheel Selection）等：这些是选择操作的具体实现方式，用于决定哪些个体将用于产生下一代。
- 精英策略（Elitism）：精英策略是指保留一部分最优个体直接进入下一代而不经过选择、交叉和变异过程，以确保最优解不会因遗传操作而失去。

1.3 算法流程

遗传算法的基本流程包括以下步骤：

1. 初始化：随机生成一个初始种群，包含 N 个个体。每个个体代表了一个潜在解决方案。

2. 评价：计算种群中每个个体的适应度（fitness），适应度决定了个体被选中用于繁殖下一代的机会。适应度高的个体更有可能被选中。

3. 选择（Selection）：根据个体的适应度进行选择。常用的选择方法有轮盘赌选择、锦标赛选择等，目的是为了选择适应度较高的个体繁殖后代。

4. 交叉（Crossover）：通过组合两个（或更多）选中的染色体来创建新的后代。交叉点可以是单点、两点或均匀交叉等，这个步骤主要是为了产生新的遗传特性。

5. 变异（Mutation）：以一定的变异概率随机改变某些个体的某些基因。变异引入新的遗传材料到种群中，帮助算法避免局部最优，并保持种群的多样性。

6. 替代：新生成的后代将取代旧的种群中的某些个体，可以是完全替代，也可以是部分替代。这样可以使得种群保持一个恒定的规模。

7. 终止条件：如果满足终止条件，则算法结束。终止条件可以是找到满意的解决方案、达到预设的迭代次数或计算时间等。

重复执行步骤 2 到 6，直到满足终止条件。遗传算法的最终目标是在搜索空间中找到适应度最高的个体，这个个体代表着问题的最优解或者近似最优解。算法的流程图如图 1.1 所示：

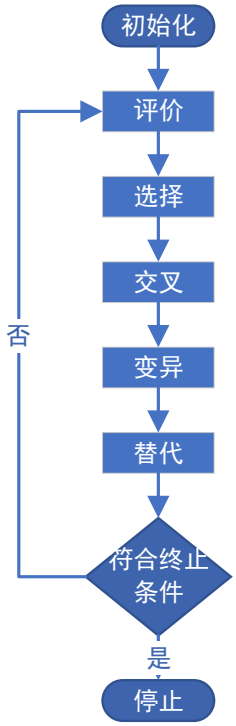

图 1.1 算法流程

遗传算法适用于搜索空间大、问题复杂的情况，它不保证一定能找到全局最优解，但是在许多实际问题中能够找到一个很好的解决方案。

二、程序设计

2.1 目标函数

在本次实验中，目标函数为 $z = 3(1-x)^2 e^{-x^2-(y+1)^2} - 10\left(\frac{x}{5} - x^3 - y^5\right) e^{-x^2-y^2} - \frac{1}{3}^{e^{-(x+1)^2-y^2}}$。本次实验的目的就是通过遗传算法来求得目标函数中最大取值的取值点和最大值。

2.2 解码函数

```
def translateDNA(pop):
    x_pop = pop[:, 1::2]
    y_pop = pop[:, ::2]

    x = x_pop.dot(2 ** np.arange(DNA_SIZE)[::-1]) / float(2 ** DNA_SIZE - 1) * (X_BOUND[1] - X_BOUND[0]) + X_BOUND[0]
    y = y_pop.dot(2 ** np.arange(DNA_SIZE)[::-1]) / float(2 ** DNA_SIZE - 1) * (Y_BOUND[1] - Y_BOUND[0]) + Y_BOUND[0]
    return x, y
```

这个函数用于将种群矩阵 pop 中的二进制编码的 DNA 转化为对应的实数值。它使用奇数列表示 x，偶数列表示 y。通过使用 dot() 函数计算每个个体的二进制编码和对应权重的点积，并通过一系列的数学操作将结果转换为实际的 x 和 y 值。

2.3 适应度函数

```
def get_fitness(pop):
    x, y = translateDNA(pop)
    pred = F(x, y)
    return (pred - np.min(pred)) + 1e-3
```

这个函数用于计算种群中每个个体的适应度。它通过调用 translateDNA() 函数将种群矩阵 pop 转换为具体的 x 和 y 值，并将这些值传递给 F(x, y) 函数来计算适应度。适应度是根据函数 F(x, y) 的输出计算的，并且通过将最小的适应度减去适应度数组中的每个值，并在加上一个小的常数来调整适应度的范围，避免出现负数和 0.

2.4 交叉函数

```
    def crossover_and_mutation(pop, CROSSOVER_RATE=0.8):
        new_pop = []
        for father in pop:
            child = father
            if np.random.rand() < CROSSOVER_RATE:
                mother = pop[np.random.randint(POP_SIZE)]
                cross_points = np.random.randint(low=0, high=DNA_SIZE
     * 2)
                child[cross_points:] = mother[cross_points:]
            mutation(child)
            new_pop.append(child)

        return new_pop
```

这个函数实现了交叉和变异操作。它先复制每个个体，并将复制的个体存储在 new_pop 列表中。然后，对每个父个体，它可以根据交叉率 CROSSOVER_RATE 来决定是否进行交叉操作，如果需要交叉，则随机选择一个母个体，并选择一个交叉点，将交叉点后的基因从母个体复制到子个体。最后，对每个子个体，它可以根据变异率 MUTATION_RATE 来决定是否进行变异操作，如果需要变异，则随机选择一个基因位并对其进行反转。最后，函数返回变异后的种群矩阵 new_pop。

2.5 变异函数

```
    def mutation(child, MUTATION_RATE=0.003):
        if np.random.rand() < MUTATION_RATE:
            mutate_point = np.random.randint(0, DNA_SIZE * 2)
            child[mutate_point] = child[mutate_point] ^ 1
```

这个函数实现了变异操作。它根据变异率 MUTATION_RATE 来决定是否进行变异。如果需要变异，则随机选择一个基因位并对其进行反转，将 0 变为 1，或将 1 变为 0。

2.6 选择函数

```
    def select(pop, fitness):  # nature selection wrt pop's fitness
        idx = np.random.choice(np.arange(POP_SIZE), size=POP_SIZE, re
    place=True,p=(fitness) / (fitness.sum()))
        return pop[idx]
```

这个函数实现了选择操作，根据适应度从种群中选择个体构成新的种群。它

使用 np.random.choice() 函数根据适应度来选择个体，并返回选中的个体所构成的新种群。

三、实验结果

3.1 实验环境

系统：Windows11

语言：python3.11

依赖包：numpy、matplotlib、mpl_toolkits

编辑器：

3.2 实验输出函数

```
def print_info(pop):
    fitness = get_fitness(pop)
    max_fitness_index = np.argmax(fitness)
    print("max_fitness:", fitness[max_fitness_index])
    x, y = translateDNA(pop)
    print("最优的基因型：", pop[max_fitness_index])
    print("(x, y):", (x[max_fitness_index], y[max_fitness_index])
)
```

该函数用于打印种群的信息，包括最佳适应度和对应的个体位置。它会使用 get_fitness() 和 translateDNA() 函数来计算个体的适应度和位置，并打印相关信息。

```
def plot_3d(ax, pop):
    X = np.linspace(*X_BOUND, 100)
    Y = np.linspace(*Y_BOUND, 100)
    X, Y = np.meshgrid(X, Y)
    Z = F(X, Y)
    ax.plot_surface(X, Y, Z, rstride=1, cstride=1, cmap=cm.coolwa
rm)
    ax.set_zlim(-10, 10)
    ax.set_xlabel('x')
    ax.set_ylabel('y')
    ax.set_zlabel('z')

    x, y = translateDNA(pop)
    fitness = get_fitness(pop)
```

```
        ax.scatter(x, y, F(x, y), c='black', s=20 * fitness / fitness
    .max())
```

这个函数用于绘制一个三维图形。它接受一个 Axex 对象 ax，用于绘制图形，并接受一个种群矩阵 pop，用于绘制种群中个体的位置。函数首先创建一个 X-Y 平面的网格，并使用 F(x, y) 函数计算出每个点的值。然后，它使用 ax.plot_surface() 方法在三维坐标空间上绘制曲面。接着，它使用 ax.scatter() 方法绘制种群散点图，其中每个个体的位置由 pop 通过 translateDNA() 函数转换而来。

```
if __name__ == "__main__":
    fig = plt.figure()
    ax = fig.add_subplot(111, projection='3d')
    plt.ion()

    pop = np.random.randint(2, size=(POP_SIZE, DNA_SIZE * 2))  # 初始化种群
    for _ in range(N_GENERATIONS):
        # 绘制每一代的最佳的适应度和个体位置
        plot_3d(ax, pop)

        # 这里更改为绘图逻辑
        plt.draw()
        plt.pause(0.1)

        # 执行遗传算法中的交叉与变异操作
        pop = np.array(crossover_and_mutation(pop, CROSSOVER_RATE))

        # 选择适应度高的个体构成新的种群
        fitness = get_fitness(pop)
        pop = select(pop, fitness)

        # 每一代结束后，清除散点数据，为下一代做准备
        plt.cla()

    # 最后输出最后一代的信息
    print_info(pop)
    plt.ioff()  # 关闭交互模式
    plot_3d(ax, pop)  # 显示最终的三维图形
    plt.show()  # 阻塞显示窗口
```

这是主函数，它调用其他函数来执行遗传算法的主要操作。在 main() 函数中，它首先创建一个三维图形，并初始化一个种群矩阵。然后，它使用 plot_3d() 函数绘制种群的初始状态，并开始循环进行遗传算法的操作：绘制图形、执行交叉

和变异操作、选择操作，并在每一代结束后清除图形数据以进行下一代的绘制。最后，它打印最后一代的信息，并显示最终的三维图形。

3.3 实验输出和分析

```
max_fitness: 3.126629244744552
最优的基因型： [1 0 0 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 0 1 1 0 1 1 1 1 1 1 0 1 1 1 1 1 0 0
 1 1 0 0 1 1 1 0 0 0 0]
(x, y): (-0.029509844154706144, 1.4985196887564474)
```

终端输出的实验结果表明：遗传算法在目标函数上得到的最大值为3.126629244744552，并且在终端输出的坐标中取得。

同时，画图函数也记录了算法迭代的过程，如下图 3.1 所示：

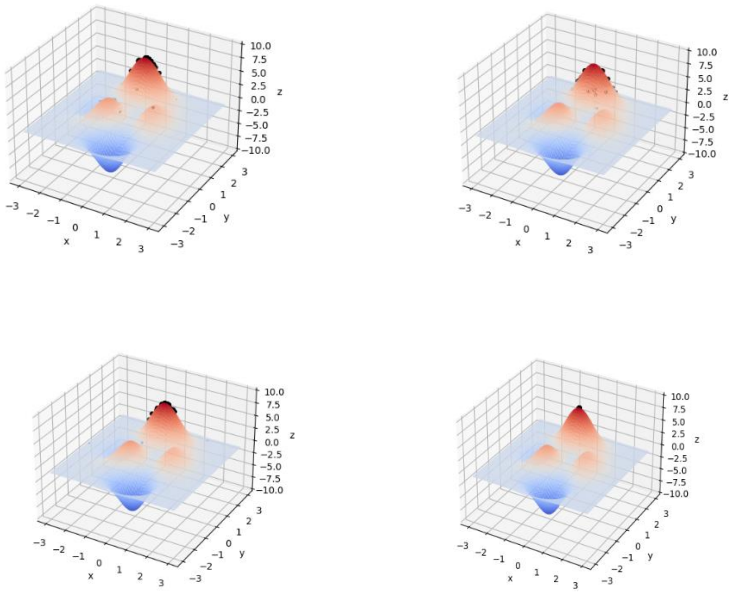

图 3.1 算法迭代过程

通过对比分析上面的实验结果，可以看出：在初始化时种群中的个体全部分散在空间的各个地方，通过交叉、变异和替代，离极值点较远的个体被淘汰，而适应度高的个体被保留下来进行繁殖。最后，种群中的个体聚拢在函数图像的最高点中。

四、总结

4.1 遇到的困难和解决办法

在理解这个实验的过程中，需要理解各个函数的作用，以及数学模型的背景和含义。当对函数或数学模型感到困惑时，我通过参考相关的文档、书籍或网上教程来加深理解。

当运行代码时，遇到了错误或问题。这可能涉及到语法错误、逻辑错误或计算错误等。在遇到问题时，我使用调试工具来检查代码的执行和变量的值，或者输出中间结果来帮助找到问题所在。

在遗传算法中，许多参数需要进行调整和选择，如种群大小、交叉率、变异率等。调整这些参数会对算法的性能产生重要影响。在调参时，我尝试了不同的参数设置，并观察算法的运行情况和结果。

4.2 实验感想

这个实验让我对遗传算法有了更深入的了解。通过阅读和实现这段代码，我了解到了遗传算法的工作原理、实现细节和应用场景。在实验中，我花费了不少时间来调整参数和理解每个函数的作用。我认为这是一个非常实用的算法，可以用于优化问题的求解和机器学习中的参数搜索。最后，我觉得这个实验是一个很好的学习和实践的机会，让我更深入地了解到了遗传算法，并让我把理论知识应用到实践中。