现代优化计算方法实验二

姓名：卢韬

学号：20194127

班级：自动化1903

实验二

目录

[我在学习中的遗传算法应用 2](#_Toc101608230)

[指派任务 3](#_Toc101608231)

[实验过程 4](#_Toc101608232)

[主要流程 4](#_Toc101608233)

[初始化种群 4](#_Toc101608234)

[选择 5](#_Toc101608235)

[交叉 5](#_Toc101608236)

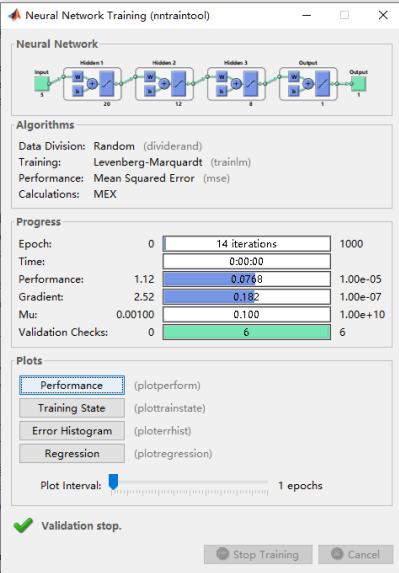
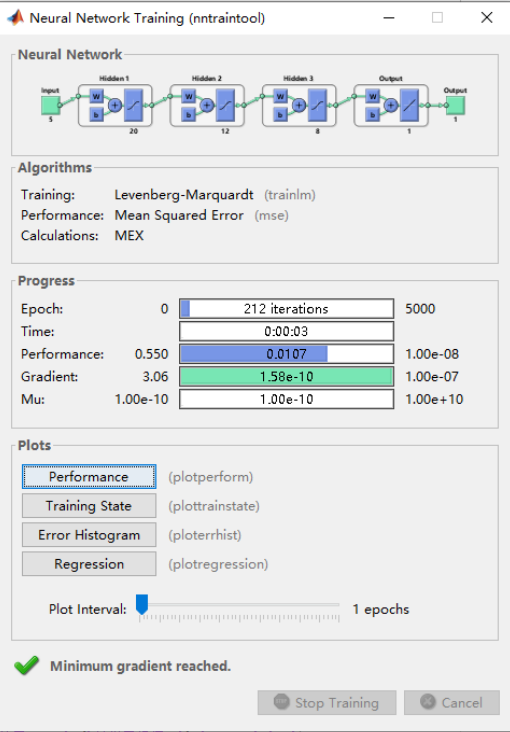
[变异 6](#_Toc101608237)

[实验结果 6](#_Toc101608238)

# 我在学习中的遗传算法应用

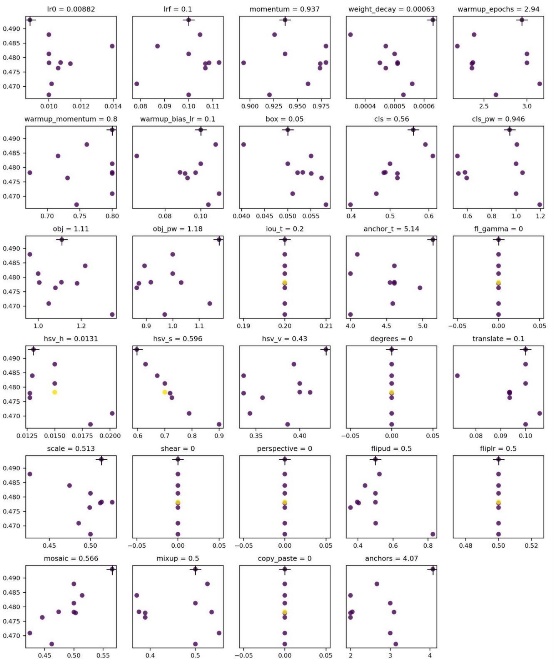
由于喜欢写代码，我对遗传算法的了解比较早

第一次使用遗传算法是DAO的AI100项目，学姐布置了练手作业。当时使用了遗传算法优化神经网络的初始参数。主要工作为将网络初始权重值编码为染色体，指定合适的目标函数、合适的交叉、选择、变异策略。最终我将将神经网络算法收敛位置从212epoch优化到了14epoch。



优化算法前后对比图

第二次是寒假在海康威视研究院实习，使用GA优化目标检测任务的重要超参数，比如NMS、目标框大小、比例、LR等等。由于使用的神经网络规模较大，在公司的V100服务器上跑出一次价值函数值也需要2h左右，也就是跑完一个完整的遗传算法需要四五天的时间。我认为这样的算法相当于将模型训练了太多次，但是没有更好的解决方法。最终我的模型在公司数据集上提升了1.6个百分点。



利用遗传算法对目标检测算法超参数进行超参寻优

同时也利用遗传算法优化的YOLOV5参数，在kaggle比赛海星检测中实现了切换榜单上升509名的效果，最终排名193/2026，获得了铜牌。

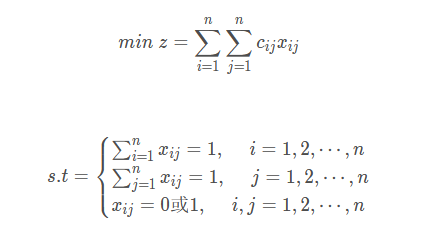


在本学期第一节课和老师交流后，才知道有更适合神经网络优化的套用遗传算法的方法。这也让我意识到了数学理论基础才是决定方法上限的因素，只会低头写代码无法提升工作效率。

# 指派任务

有n项任务，由n个人来完成，每个人只能做一件，第i个人完成第j项任务要k(I,j)​小时，如何合理安排时间才能使总用时最小？

数学模型为：



* 第一个式子表示完成全部n项工作所消耗的总资源数要最少；
* 第二个式子表示第i个人只完成一项工作；
* 第三个式子表示第j项工作只能由一个人完成；
* 第四个式子表示决策变量只能取0或者1。

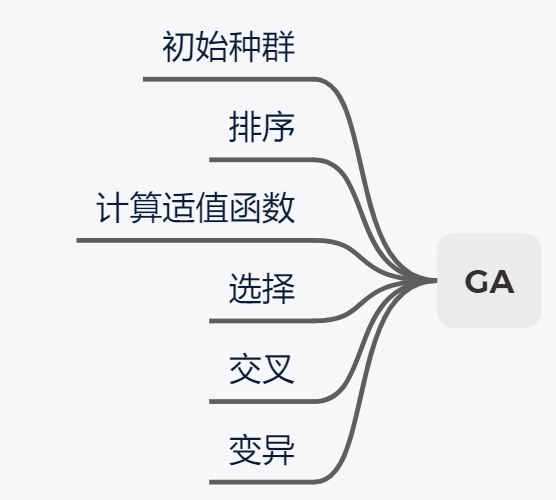
需要注意的是，指派问题可以看作0 - 1整数规划问题来求解，也可以用更简单的匈牙利算法来求解。此处按照规定使用遗传算法GA，但性能其实不如传统运筹学算法。

# 实验过程

线上运行代码：

https://colab.research.google.com/drive/1qUkDmEGFqXCzLO2q7\_jzDhT9KxpT4Kmc?usp=sharing

## 主要流程



算法流程

设计的超参数为：种群数量1000，pc为0.1，pm为0.8。由于期末考试需要准备，因此未对超参数进行实验调整。

## 初始化种群

1. rang = np.zeros((magnit, pop), int)
2. for i in range(magnit):
3. select = [x for x in range(scale)]
4. random.shuffle(select)
5. rang[i, :] = select

## 选择

1. newchoose = []
2. new\_sum\_time = []
3. for \_ in range(magnit):
4. rand = random.random()
5. for row in range(magnit):
6. if rand > fitlevelplus[row, :] and rand < fitlevelplus[row
7. newchoose.append(rang[row, :])
8. new\_sum\_time.append(sum\_time\_pop[row, :])
9. continue
10. newchoose = np.array(newchoose)
11. prechoose = newchoose.copy()
12. new\_sum\_time = np.array(new\_sum\_time)

## 交叉

1. for individual in range(0, (np.shape(new\_solution))[0], 2):
2. if pc > random.random():
3. chrome\_a = new\_solution[individual, :].copy()
4. chrome\_b = new\_solution[individual + 1, :].copy()
5. select = [x for x in range(10)]
6. random.shuffle(select)
7. this\_A = select[0]
8. this\_B = select[1]
9. this\_A, this\_B = min(this\_A, this\_B), max(this\_A, this\_B)
10. cthis\_A, cthis\_B = chrome\_a[this\_A:this\_B + 1].copy()
11. # print(this\_A, this\_B, cthis\_A, cthis\_B)
12. for site in range(this\_B - this\_A + 1):
13. s1 = np.where(chrome\_a == cthis\_B[site])
14. chrome\_a[s1] = (chrome\_a[this\_A:this\_B + 1])[site].copy
15. a = chrome\_a
16. for site in range(this\_B - this\_A + 1):
17. s2 = np.where(chrome\_b == cthis\_A[site])
18. chrome\_b[s2] = (chrome\_b[this\_A:this\_B + 1])[site].copy
19. b = chrome\_b
20. chrome\_a[this\_A:this\_B + 1] = cthis\_B.copy()
21. chrome\_b[this\_A:this\_B + 1] = cthis\_A.copy()
22. new\_solution[individual, :] = chrome\_a.copy()
23. new\_solution[individual + 1, :] = chrome\_b.copy()

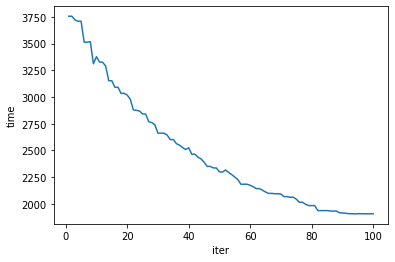
## 变异

1. for individual in range(0, (np.shape(new\_solution))[0]):
2. if pm > random.random():
3. select = [x for x in range(10)]
4. random.shuffle(select)
5. this\_A = select[0]
6. this\_B = select[1]
7. gen = new\_solution[individual, :]
8. gen[this\_A], gen[this\_B] = gen[this\_B], gen[this\_A]
9. new\_solution[individual, :] = gen.copy()
10. rang = np.append(prechoose, new\_solution, 0)
11. bestpop.append(sum\_time\_pop[0, :])

## 实验结果



使用tqdm帮助了解迭代当前位置和迭代时间



运行过程中目标值随迭代次数变化的规律