队伍编号	mc21170020024
题号	B

基于改进遗传算法的原子团簇结构预测

摘 要

原子团簇的结构预测是结构化学与材料化学领域的一项重要研究内容，对于纳米材料的制作有重要作用。相比于传统的密度泛函理论，启发式算法对于团簇结构预测有重要作用。针对 Au 与 B 两种典型的金属和非金属元素，使用遗传算法对其结构进行预测有着良好的效果。此外，XGBoost 算法作为一种新兴的机器学习算法，也可高效地与遗传算法结合起来。

对于问题一，使用了基于 Gupta 势函数的遗传算法和基于 XGBoost 的遗传算法并将二者效果进行了对比，预测出了 Au_{20} 的团簇结构；问题二由于缺乏数据，使用的是基于 Gupta 势函数的遗传算法寻优；问题三除了使用机器学习以外，还将非金属元素中应用广泛的 L-J 势函数和 REBO 势函数进行了比较，发现 REBO 势函数表现更优。问题四并未使用机器学习，使用 REBO 势函数预测出 B_{40}^- 的基本结构。这一问题的重大意义不只是结构，更重要的在于融合 XGBoost 的遗传算法改进思想，将获得更广泛的应用。

关键词：团簇结构预测　遗传算法　XGBoost　势函数

目录

一、 问题重述

团簇，隶属纳米材料的尺度概念。团簇是由几个乃至上千个原子、分子或离子通过物理或化学结合力组成的相对稳定的微观或亚微观聚集体，其物理和化学性质随所含的原子数目而变化。团簇的空间尺度是几埃至几百埃的范围，用无机分子和小块固体来描述都不太合适，许多性质也和单个原子分子与固体液体有较大差异，于是也不能用两者性质的简单线性外延或内插得到。因此，我们把团簇看成是介于原子、分子与宏观固体物质之间的物质结构的新层次。

团簇可以分为金属团簇和非金属团簇。但由于团簇的势能面过于复杂，同时有时候还需要考虑相对论效应等，所以搜索团簇的全局最优结构显得尤为困难。常规方法如数值迭代法等在高精度计算中有失偏颇，所以我们需要对其进行改进来解决相应问题。我们将原子团簇的结构优化问题进行如下的简化抽象描述:一个原子团簇是由 n 个原子构成(n 是自然数)，原子团簇中的任意两个原子之间都存在相互的作用力，两原子间也就存在一个能量，能量大小只与原子间的距离有关，称之为原子团簇的原子势能。我们的问题就归结到根据一些原子的能量坐标数据来建立相应的势函数模型并搜索到当能量达到最小值时原子团簇中 n 个原子所处的三维空间坐标[1]。

以 Au 原子和 B 原子为代表的金属元素和非金属元素团簇存在着不同的结构。问题的目标即为利用一系列智能算法预测 Au_{20}、Au_{32}、B_{45}^{-}、B_{40}^{-}的结构。

二、 问题分析

2.1 问题一的分析

对于过渡金属金来说，表面原子形成的键比较少，但是每个键都比较强，使用对势能很难精确地建模原子间的相互作用力。为了更好的描述金属团簇原子间的相互作用力，这里使用多体势能 Gupta 势能来进行金原子团簇势能建模。根据 Gupta 势能函数的形式可得知其由两个部分组成，即对势能部分与多体势能部分，其中对势能的作用是给原子间产生排斥力，而多体势能部分的作用是使原子间产生吸引力。在一个稳态的团簇结构中，各原子受到的吸引力与排斥力达到了一种平衡[1]。

可以通过使用遗传算法，利用附件中 1000 个金原子的相关数据来拟合得到 Au_{20} 团簇的 Gupta 势函数的五个参数，然后再进行迭代，运算 1000 次后得到能量最低状态时金团簇 Au_{20} 的最低能量和其内部各金原子的坐标值。此外，近年来在团簇结构预测的研究中有一些研究也采取了机器学习算法进行预测，尤其是利用高斯过程回归来对基于原子中心对称函数坐标的原子团簇进行高斯近似势的预测[2]。类似地，也可以使用其它流行机器学习算法进行辅助学习。

2.2 问题二的分析

问题一中使用的 Gupta 势函数模型及其主要参数，对于 Au_{32} 团簇模型仍然适用。所以运用该模型对参数进行一些微调后即可对 Au_{32} 团簇进行搜索与预测，得到其全局最优结构。

2.3 问题三的分析

对于非金属原子团簇特别是硼原子团簇，其并不主要以三维的结构存在，而多是以准平面，平面的结构形式存在，这样的结构形式也使得它的势能研究具有一定的特殊性。且其相对于金属原子团簇来说，非金属原子团簇原子之间相互作用力较为简单，所以我们使用对势能 REBO 势函数来对其进行建模。另外，对于非金属元素尤其是惰性气体，L-J（Lennard-Jones）势被广泛应用。近年来，在类富勒烯结构中也有大量研究采用 L-J 势作为势函数，硼元素在元素周期表的位置与碳、硅相邻，根据相似相近原则我们有理由推测 L-J 可能也是一种合适的势函数[3]。具体的效果需要进行对比得知，可以由附件中 3751 个硼原子的相关数据来拟合得到。

2.4 问题四的分析

问题三中使用的 REBO 势函数模型对于 B_{40}^-团簇模型仍然适用，所以该模型调整一些参数后即可对 B_{40}^-团簇进行搜索与预测，得到其全局最优结构。

三、 模型假设

3.1 一个团簇内部原子均视为质点

由于我们只关注团簇内原子之间的连接，所以忽略其原子半径的影响，且能计算出两个原子间存在的最小距离，这样大大简化了研究的问题。

3.2 关于 REBO 函数的参数约定

由于 REBO 函数参数过多，这里假设所有的修正项均为同一个常数 b。

3.3 关于原子间距的约束条件假设

为简化问题，式（2）中的上限设置为一个略大一些的常数而并非与 n 有关。

四、 符号说明

表 1　各符号表示及对应解释

符号	说明
V	能量
r_{ij}	i,j 原子间距
$\|\|w\|\|$	距离模量
T	决策树深度
$L(\Phi)$	目标函数
w	分类器分数
λ	正则化参数
$\Omega(f)$	正则化项

σ	原子间最小距离
P,q,A, ξ ,d_0	Gupta 势函数参数
A,B_1,B_3,β_1,β_3,a,b,Q	REBO 势函数参数
ε	LJ 函数势阱深度

五、 模型的建立与求解

5.1 问题一模型的建立与求解

5.1.1 模型的建立

金团簇是金属型团簇，基于此我们建立多体势能 Gupta 势函数模型来模拟其内部势能。Gupta 势函数方程形式如下：

$$V=\sum_{i=1}^{n}\left(A\sum_{j(\neq r)=1}^{n}\exp\left[-p\left(\frac{r_{ij}}{d_0}-1\right)\right]\right)-\left\{\xi^2\sum_{j(\neq r)=1}^{n}\exp\left[-2q\left(\frac{r_{ij}}{d_0}-1\right)\right]\right\}^{\frac{1}{2}} \tag{1}$$

其中 r_{ij} 是团簇中第 i 个原子和第 j 个原子之间的距离，Gupta 势函数就是最终优化目标。对于原子团簇而言，团簇能量越低则结构越稳定，这一问题被化归为一个目标函数求最小值的问题。对于 Au_{20} 而言，原子团簇一共 20 个点，每两个点之间的距离可以根据坐标公式求解：

$$r_{ij}=\sqrt{(x_i-x_j)^2+(y_i-y_j)^2+(z_i-z_j)^2} \tag{2}$$

20 个原子位置，每个位置有三个坐标分量于是有 60 个变量。这是一个 60 维函数的优化问题。在求解之前还需要注意变量的限制条件。在团簇等多原子系统中存在如下约束条件[4]：

$$\begin{cases}0<r_k\le\sqrt{3\times(0.5\times\sqrt[3]{6n}\sigma)^2}\\ r_{ij}>\sigma,i,j,k=1,2,...,n\end{cases} \tag{3}$$

可以观察到，每个原子到原点的距离会达到一个上限，而且与 n 的三分之一次方成正比。这里假设上限是一个略大一些的常数，便于我们使用算法求解。

遗传算法（Genetic Algorithm, GA）是一种通过模拟达尔文理论中遗传变异与自然进化过程搜索最优解的方法。其主要特点是直接对结构对象进行操作，不存在求导和函数连续性的限定；具有更好的全局寻优能力；不需要确定的规则就能自动获取和指导优化的搜索空间，自适应地调整搜索方向[5]。它通过类比种群进化过程中基因与染色体的遗传变异，将解空间编码为一系列基因，并逐代演化出越来越好的近似解，再根据个体对环境的适应程度（即与问题的匹配程度）筛选出更优的个体，并交叉变异进化等获取新的解集。不断重复这一过程直到问题收敛，可以求出问题的近似最优

解。

遗传算法的基本思想如图 1 所示。

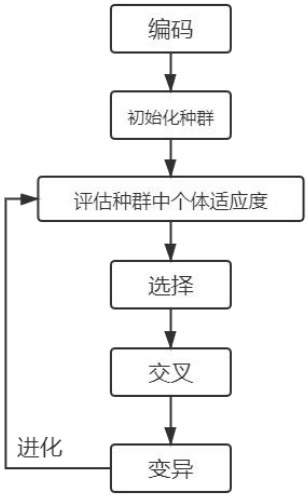

图 1　遗传算法的思想基本流程

另外，可以使用机器学习算法来构建一个高度封装不可观察具体形式的势函数。相比于支持向量机、高斯过程回归等传统机器学习算法，由 Chen 等人提出的 XGBoost 算法具有更高的准确度和更强的鲁棒性[6]。XGBoost 是对梯度提升树框架(Gradient Boost Decision Tree Framework)的一种具体实现，以 CART 决策树作为基学习器，既可以用于分类问题也可以用于回归问题。XGBoost 的基本形式为：

$$\begin{aligned} L(\phi) &= \sum_i l(\hat{y}_i, y_i) + \sum_k \Omega(f_k) \\ \Omega(f) &= \gamma T + \frac{1}{2}\lambda \| w \|^2 \end{aligned} \tag{4}$$

不同于普通的 GBDT，XGBoost 在 L 后加入了与决策树深度 T 和分类器分数 w 有关的正则化项，使得精度本就出众的模型更具有鲁棒性。此外从某种意义上讲，它本身还具有降维的效果，适合处理这种高维数据。

5.1.2 模型的求解

利用附件中给出的 1000 个金原子团簇能量及其三维坐标值，运用自定义函数（非多项式）拟合的方法得到式（1）中对应的参数值分别如表 1 所示：

表 2　　Au 团簇的 Gupta 势函数参数

	p	q	A	ξ	d_0
Au	1.0013	0.9994	0.9870	0.8092	0.8277

据此，我们已经得到了 Gupta 势函数表达式，可以进行目标优化。以 2.49 作为原

子间距离的下限，每个原子到原点距离的平方不超过 85，迭代上限 1000 次。以此为限制条件，势函数在不使用 XGBoost 时使用遗传算法时 Au_{20} 预测的能量值，以迭代次数为横坐标，解空间的表现为纵坐标，可以绘制图 2 的遗传算法训练曲线：

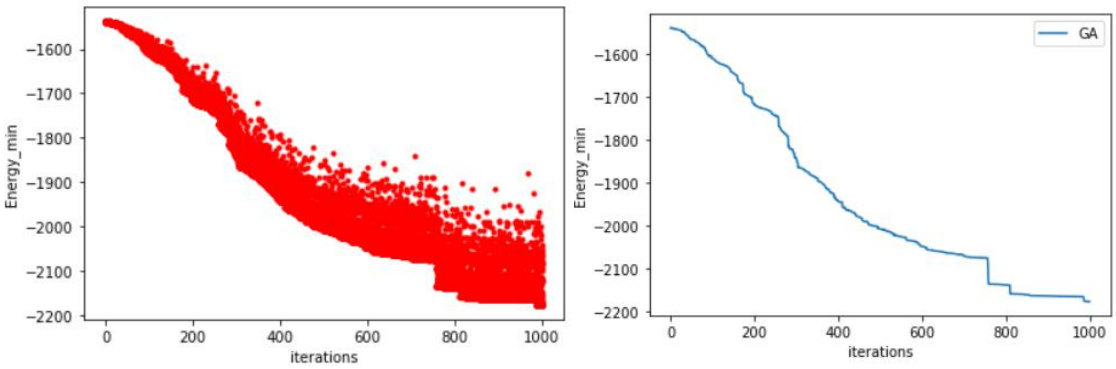

图 2　使用 Gupta 势函数的遗传算法的 Au_{20} 最低能量预测结果

最终预测的最低能量为 $V_{1,\ min}$=-2124.368612，但从结构上来讲，原子分布集中形成了四个团而不具备规则的几何形状与相对均匀的分布。因此，我们认为，这一结果不够合理，需要进行进一步改进。

在没有使用机器学习时使用遗传算法对 Au_{20} 团簇预测的运行时间如图 3 所示。

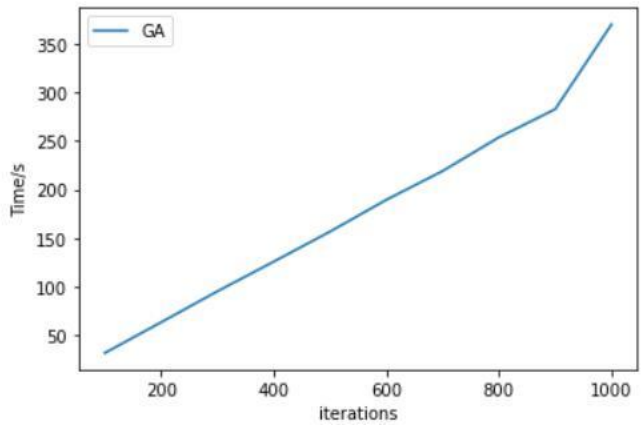

图 3　使用 Gupta 势函数的遗传算法 Au_{20} 的运行时间

可以发现，运行时间在初期与迭代次数近似为一个正比关系，当迭代次数增加以后运行时间会快速增长。

而在配合使用 XGBoost 后再配合遗传算法时 Au_{20} 预测的能量曲线如图 4。通过构建一个合适的 XGBoost 模型，我们可以将 20*3 一共 60 个数据作为自变量得到能量作为响应。回归器的 R^2 分数达到了 0.73，虽然并非高度相关但结果比较正常。因为受到提供数据本身的限制，经过重复实验这一结果已经达到了极限。以 XGBoost 回归器的预测结果作为势函数响应，我们可以定义一个高度封装的 XGB 势函数，这一函数很难写出具体表达式，但效果比 Gupta 可能更好。实验结果也恰恰验证了这一点。

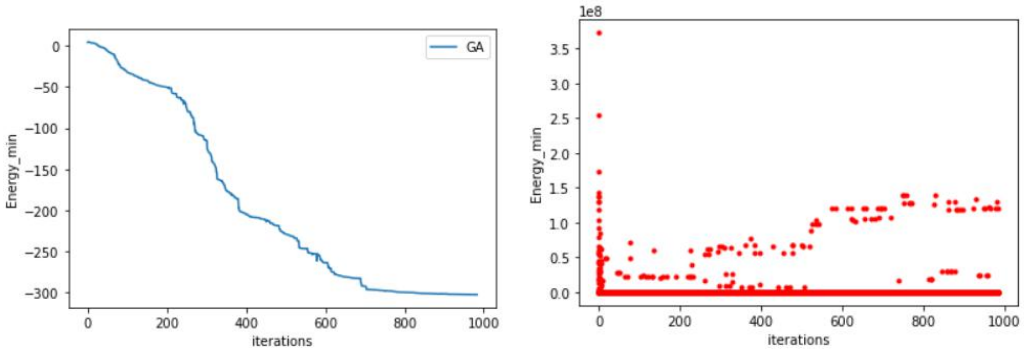

图 4　使用机器学习的遗传算法 Au_{20} 最低能量预测结果

在训练 XGBoost 过程中我们对能量数据和自变量坐标都进行了 Z-score 标准化，其形式为：

$$x = \frac{x - \bar{x}}{S_x} \tag{5}$$

经过还原，测得能量值为 $V_{1,\ min}$= -1561.2942。由此发现结果更加可信，得到的结构更加合理和规则。具体的结构在图 5 中展现。

由以上图线可以看出，在配合使用了机器学习方法后，再使用遗传算法时预测的模型更加符合实际情况且运行时间更短，其对应的金团簇 Au_{20} 的全局最优结构形状如图 5 所示。各原子的大致分布如下，键的连接方式仅供参考：

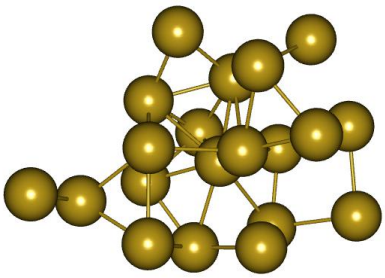

图 5　Au_{20} 全局最优结构预测图

由我们的模拟形状可以看出其大致呈现四面体构型，其中包含了若干个小正四面体。由几何知识得知四面体在空间中重心较低，四个面都是正三角形，构型较稳定，符合我们常规的认知。

5.2 问题二模型的建立与求解

5.2.1 模型的建立

由于 Au_{32} 为金属团簇，则仍可以用问题一中建立的势函数模型来对其全局最优结构进行搜索与预测。唯一不同的是原子个数不同使得自变量增多，维度发生变化。这时，XGBoost 方法接收的输入维度发生改变，又得不到新的训练数据作为补充，所以只能使用势函数法。

5.2.2 模型的求解

同理仍绘制相关曲线来判断遗传算法在 Gupta 势函数上使用的效果。改变输入维度等参数以后遗传算法的曲线如图 6 所示：

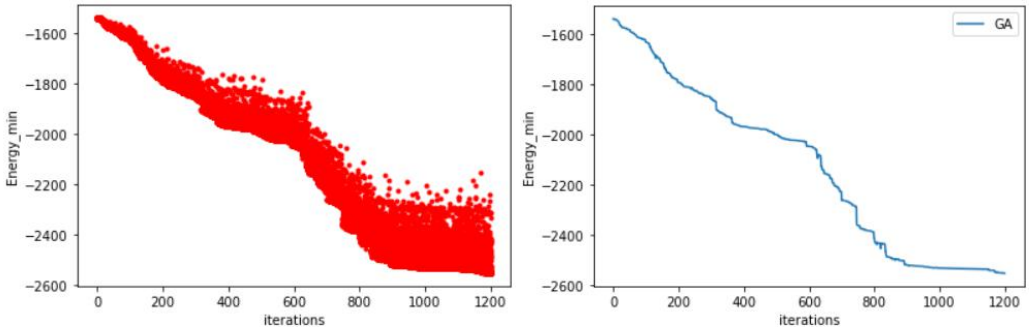

图 6　改变维度后遗传算法的预测曲线

我们预测得到其最低能量值 V_2 为 $V_{2,\ min}$=-2553.13876475，其对应的金团簇 Au_{32} 的全局最优结构形状如图 7 的柱状结构。键的连接方式仅供参考。

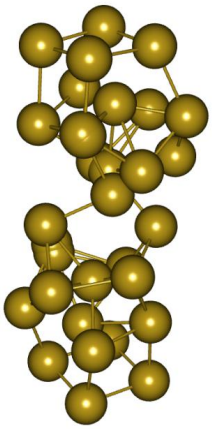

图 7　Au_{32} 的全局最优结构图

由原子之间位置可以得知其大致呈现柱体的结构，柱体结构会使原子之间接触面积增大，吸引力增大，成键增强，构型较稳定。

5.3 问题三模型的建立与求解

5.3.1 模型的建立

与金团簇等其他金属团簇相比，非金属原子团簇结构有所不同。我们引入 REBO 键序势函数模型和 L-J 经验势函数模型来描述非金属团簇中的共价键系统，即可近似认为两个原子之间键的强度不是常量而是取决于原子的局部环境。首先在是 REBO 势函数中涉及到的一些能量关系式[7]。

REBO 模型的总势能为：

$$E_b = \sum_i \sum_{j(>i)} [V^R(r_{ij}) - b_{ij} V^A(r_{ij})] \tag{6}$$

模型可以分成两个部分。第一部分的排斥势为：

$$V^R(r_{ij}) = f^c(r_{ij})\,(1 + Q/r_{ij})\,Ae^{-ar_{ij}} \tag{7}$$

另一部分吸引势可以写成：

$$V^A(r_{ij}) = f^c(r_{ij}) \sum_{n=1,3} B_n e^{-\beta_n r_{ij}} \tag{8}$$

为了使得模型更具有普适性，考虑到团簇的空间几何结构，引入键序的概念：

$$b_{ij} = \frac{1}{2}[b_{ij}^{\sigma-\pi} + b_{ji}^{\sigma-\pi}] + b_{ij}^{\pi} \tag{9}$$

其中，修正项b_{ij}^{π}包括了非局域效应和 B-B 二面角的影响。但为了研究问题的方便，我们假设所有的键序均为常数。

$$b_{ij}^{\pi} = \pi_{ij}^{RC} + b_{ij}^{DH} \tag{10}$$

再来考虑 L-J 势能。L-J 势能的基本形式非常纯粹，只有两个参数：

$$E = 4\varepsilon \sum_{j<i} [(\frac{\sigma}{r_{ij}})^6 - (\frac{\sigma}{r_{ij}})^{12}] \tag{11}$$

这一势能函数被广泛应用于 He、Ne、Ar 等惰性气体原子团簇的结构预测，一部分过渡金属的结构还有 C_{60} 的结构也可以使用 L-J 函数。考虑到硼的化学性质，我们决定对 L-J 函数同样进行尝试。

5.3.2 模型的求解

利用附件中给出的 3751 个硼团簇能量值与分别对应的三维坐标值，我们可以拟合得到在 REBO 势函数模型中对应的参数值（我们把所有键序与修正项视为一个常数 b）以及 L-J 函数的参数，并将其结果分别列举在表 3 和表 4 中。

表 3　B_{45} 团簇的 REBO 势函数参数

	A	B_1	B_3	β_1	β_3	a	b	Q
B	0.3503	0.0457	0.0421	0.4189	0.4242	1.0089	1.9094	3.1001

表 4　B_{45} 团簇的 L-J 势函数参数

	ε	σ
B	-40.1779	0.0422

在不使用 XGBoost 时使用遗传算法时 B_{45}^-预测的能量值如图 8 所示。其中红色代表 L-J 函数，蓝色代表 REBO 函数。在遗传算法之前我们同样进行了 Z-score 标准化工作。

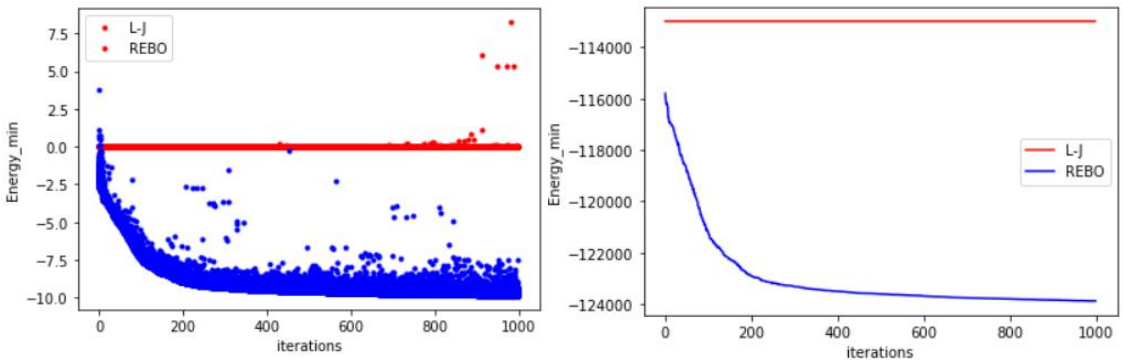

图 8　使用势函数的遗传算法 B_{45}^-最低能量预测结果

使用 L-J 函数的训练时间为 1080.8592 秒，而使用 REBO 需要 1746.1059 秒。尽管如此，L-J 函数的变化不大，似乎并不能很好地描述变化。而 REBO 函数的变化更为明显，二者的能量值分别达到了 L-J=-113003.0500，REBO=-123882.9938。观察两个函数的拟合情况以及生成最优解的空间结构，我们认为，使用 REBO 函数要更加恰当一些。

而在使用 XGBoost 后再与遗传算法结合时 B_{45}^-预测的能量值如图 9 所示。XGBoost 在这一数据集上 R^2 分数达到了 92%，MSE 只有 0.039，可以说是一个非常好的结果了。在这一方法下遗传算法的时间仅需 38.9112 秒。

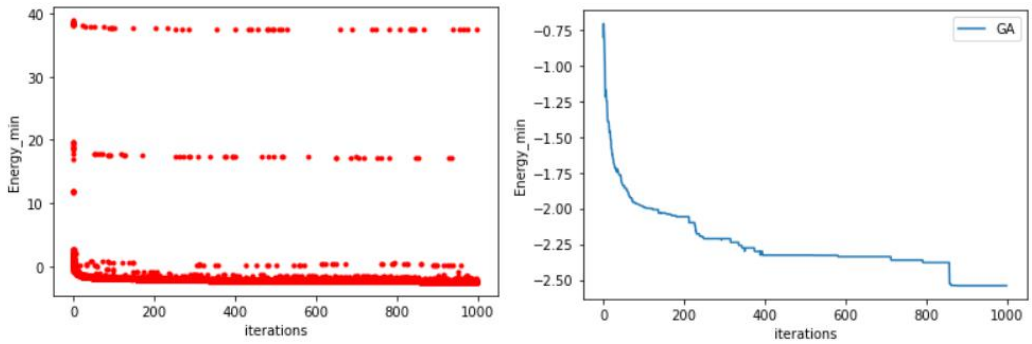

图 9　配合 XGBoost 的遗传算法 B_{45}^-最低能量预测结果

配合机器学习 XGBoost 迭代计算 1000 次后得到势能 V_3 的最小值为 $V_{3,\ min}$=-115804.2，机器学习 XGBoost 方法与 REBO 势函数方法对应的硼团簇的全局最优结构形状分别如图 10 和图 11 所示，它们都近似显现出方体结构，但对比相应文献可以发现前者预测效果较优。

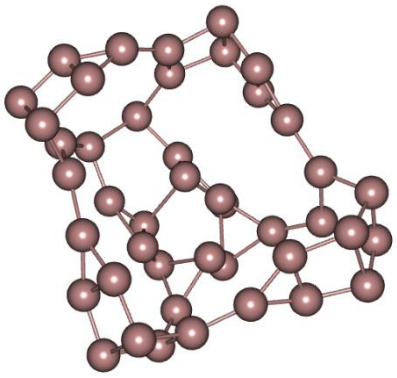

图 10　机器学习 XGBoost 预测的 B_{45}^-结构

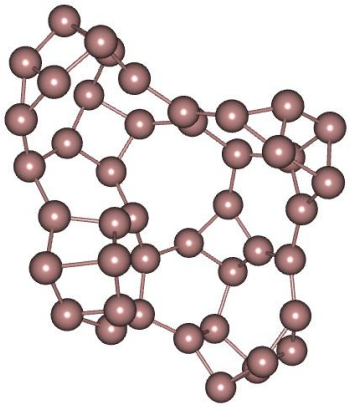

图 11　REBO 势函数预测的 B_{45}^-结构

由两个模拟图可以看出两个结构近似为扁平的长方体结构，准平面型，原子排布也更为规整。其中包含了若干个以五元环六元环为代表的组成面结构，从成键理论来讲这一结构符合化学规律，具备一定稳定性。

5.4 问题四模型的建立与求解

5.4.1 模型的建立

和问题二类似，由于输入维度的变化且缺乏训练数据，这里采用势函数寻优法。从问题三得知，此处应用 REBO 函数更加合理。

5.4.2 模型的求解

同上面的方法，图 12 为遗传算法的训练曲线，散点图中蓝色为 REBO 函数而红

色为 L-J 函数；曲线图中蓝色为 L-J 函数而橙色为 REBO 函数：

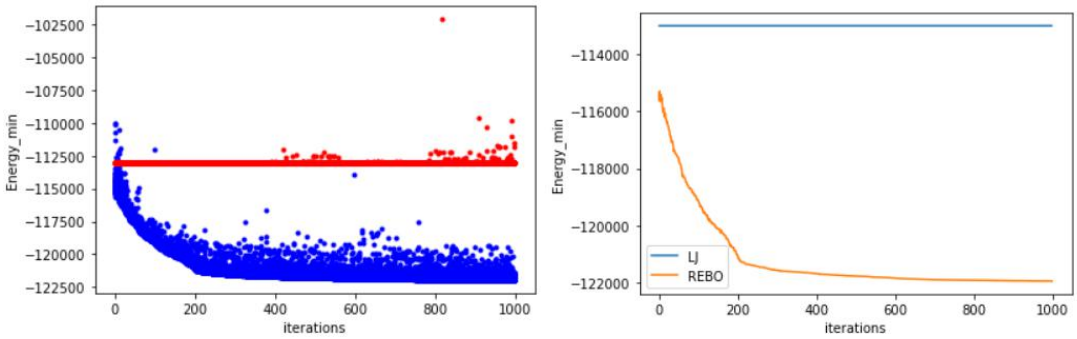

图 12　两种势函数分别对 B_{40}^-最优结构的预测

根据 REBO 函数的表现我们预测得到其最低能量值 V_4 为 $V_{4,\ min}$= -121959.0358 ，其对应的硼团簇 B_{40}^-的全局最优结构形状如图 13 所示：

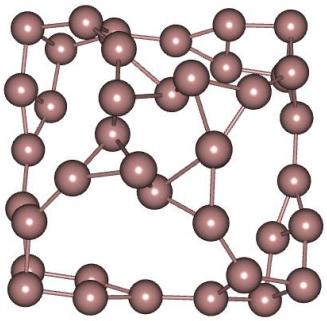

图 13　B_{40}^-的全局最优结构图

相比于图 10 和图 11，图 13 显示出的团簇模型整个平面更趋于正方形，能使得重心更低，能量更稳定。在三维欧几里得空间内，它近似为一个立方体，但在其中有三元环、五元环和六元环，这也说明硼元素的稳定团簇构型与类富勒烯结构类似，都具备一定稳定性。

六、 模型的分析与检验

6.1 只考虑经典势函数

图 14 为问题一中 Gupta 势能的拟合情况以及问题三 L-J 势能与 REBO 势能的拟合情况。其中，橙色表示真实值，蓝色代表预测值。从图 12 也可以看出，REBO 势能确实比 L-J 势能更有说服力。L-J 函数与实际结果偏差太大，不过硼元素的能量中出现两个异常高的数值，使用 REBO 函数也只能拟合相对更小的一个。

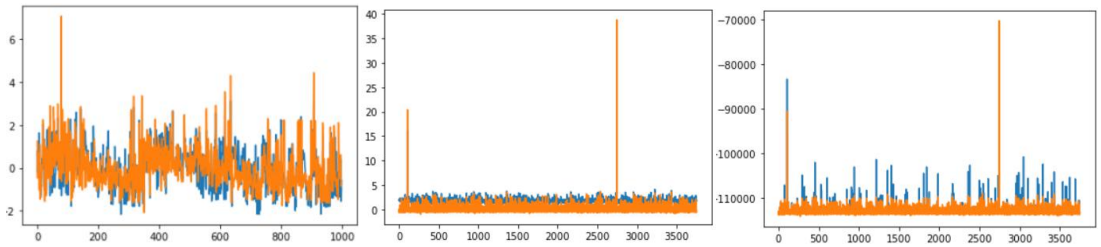

图 14　三种势函数在各自问题中的拟合情况

从模型的拟合情况来看，我们选择的势函数相对而言都能在一定程度上契合能量值与结构分布的关系。

另外，经检验，对于 Gupta 势函数和 REBO 函数，它们对各自的参量都是灵敏的，只是灵敏程度高低。例如对于 REBO 势函数，它对 a,b,Q 灵敏度就低一些。

6.2 以机器学习算法作为势函数

选择的 XGBoost 算法无论是准确度还是运算速度上都会比传统的经验势函数更好。与此同时，以问题一的金原子数据为例，我们也对其它常用机器学习回归算法进行了比较，结果如表 5 所示：

表 5　几种机器学习算法的效果

	支持向量回归	线性回归	高斯回归	决策树回归	XGBoost
R^2 分数	0.1229	0.1111	0.2364	0.0862	0.7259
MSE	1.63	1.63	1.30	1.89	0.23

可以看到，XGBoost 算法确实是这几种当中最合适的机器学习算法。

6.3 智能算法的选择

本问题选取的是遗传算法，属于进化计算方法的典型。而在智能算法中有一类重要的算法就是群智算法，包括蚁群算法、鱼群算法、粒子群算法等。本质上这些算法都可以达到目标函数最小值的求解，这里我们主要比较运行时间。从运行时间上来说，以问题一为例，同样迭代 1000 次，几种算法的运行时间如表 6：

表 6　几种智能算法的运行时间

	模拟退火算法	遗传算法	蚁群算法	粒子群算法
运行时间/s	612.5	371.9	388.6	459.7

遗传算法在达到相同的目标效果得情况下所需时间最短，所以这里使用遗传算法也很合适。

6.4 计算结构的合理性判断

从计算得到的结果来看，尽管结果仍有一定程度的偏差，结果在欧几里得空间中仍然能够呈现出一定的几何规则性，而且与现有的一些研究结果非常相似。我们相信，如果提供更多数据我们可以得到更精确的结果。

七、 模型的评价与推广

7.1 模型的优点

我们提出了基于经验势函数的遗传算法模型，还有基于 XGBoost 的学习势函数改进遗传算法模型。它们有以下优点：

1. **形式简洁** 相比于业界仍然广泛使用的密度泛函理论求解薛定谔方程等方法，本方法计算更为简洁，形式简单。

2. **计算速度快** 相比于传统方法以遗传算法这一启发式方法作为内核进行计算，提高了全局搜索函数最优解的速度。此外，借用机器学习中 XGBoost 算法训练的回归器作为势函数进行预测时间得到了更大程度上的提升

3. **结构预测准确** 在一定程度上模型的预测结构较为合理，表现出一定的几何规则性，与目前的研究结果也很相似。

4. **支持并行化，进一步提升性能** 模型目前采取的遗传算法本身就有：“隐并行化”的思想在其中，目前也有很多智能算法库支持并行化。此外，XGBoost 也是机器学习中一种易于并行化的方法，进一步提升速度。

7.2 模型的缺点

尽管我们的模型在一定程度上能够准确预测原子团簇结构，模型自身还存在一些问题。这里也指出了一些改进方案：

1. **XGBoost 方法的局限性** 在问题一和问题三中的 XGBoost 算法虽然表现很好但有一定局限性。比如，在缺少训练数据的情况下改变输入维度机器学习算法将陷入停滞，只能使用经验势函数。这里我们认为，如果提供 Au_{32} 的训练样本数据我们同样可以训练一个高性能的 XGBoost 回归器，结合遗传算法的预测结果也会更合理。

2. **势函数的局限性** 我们选取的势函数是以往研究中常用的一些势函数，它们对于金元素合硼元素是否有足够的适用性还有待商榷与进一步探讨。

3. **模型的规整性** 我们求解的模型虽然大致具备良好的几何结构但还不够规整，我们认为这是由于训练数据不够大导致模型参数拟合程度不够好导致，若提供更多数据将使得模型的预测结果几何形状更加规则。

7.3 模型的推广

相比于预测的结果本身，这一研究更重要的是研究思想，将机器学习方法用于启发式算法中。预测其它元素的原子团簇结构时可能无法找到一个非常合适的势函数，这时如果提供有模型坐标与能量的训练数据，我们便可以训练一个机器学习算法作为高度封装的势函数，利用启发式算法（遗传算法、粒子群算法、模拟退火算法等）进行目标函数最值的全局搜索，从而找到预测结构。这一想法将在模型的时间复杂度和空间复杂度上大大超过以往基于密度泛函理论等的传统研究方法。

此外，对于非材料化工问题，这一想法可以用于反解最值模型。对于一个 N 维输

入存在一个响应量作为回归目标，可以先构建输入与响应之间的机器学习算法，再根据预测模型函数进行全局搜索从而找到响应最小时对应的输入大致是多少。这一想法将作为一种机器学习的新型问题，在各领域内得到广泛应用。

八、 参考文献

[1] 赖向京. 原子团簇结构预测的现实途径—高性能启发式算法. (Doctoral dissertation, 华中科技大学).

[2] 周营成. 基于粒子群算法和高斯过程回归的新型纳米团簇结构预测研究. (Doctoral dissertation, 北京化工大学)

[3] 蔡文生, 林翼, 邵学广. 团簇研究中的原子间势函数[J]. 化学进展, 2005, 17(4):588-596.

[4] 曾思琴. (2010). 基于智能优化算法的原子团簇结构研究. (Doctoral dissertation, 浙江师范大学).

[5] MATLAB 中文论坛. MATLAB 智能算法 30 个案例分析[M]. 北京航空航天大学出版社, 2010. P17-P22

[6] Chen T , Tong H , Benesty M . xgboost: Extreme Gradient Boosting[J]. 2016.

附录

代码 python 3.8.2，编程工具 jupyter notebook，环境为 Windows10+intel i7+GEFORCE GTX1650

附录 1

介绍：Au 团簇预测代码

```
# -*- coding: utf-8 -*-
"""
Created on Fri Apr 16 22:23:31 2021

@author: mashituo
"""
#environment: scikit-opt
#读取 Au20 原子团簇文件
import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import curve_fit
from sko import GA
import time
import pandas as pd

energy=[]
XYZ=[]
for file in range(1000):
    if file==155:
        continue
    f=open("Au20_OPT_1000/%d.xyz"%file)
    f=f.read()
    xyz=[]
    f=f.split("\n")
    energy.append(float(f[1]))
    for i in range(2,22):
        u=f[i].split(" ")
        u=[u[j] for j in range(0,len(u)) if u[j]!='' and u[j]!='Au']
        xyz.append([float(u[0]),float(u[1]),float(u[2])])
    XYZ.append(xyz)

XYZ=np.array(XYZ)
energy=np.array(energy)
#二维转一维
XYZ1=[]
for i in XYZ:
    XYZ1.append(i.T.reshape(60,))
XYZ1=np.array(XYZ1)
```

```
    #直接利用二维数据的 Gupta 势函数
    def V2(k,p,q,A,xi_2,d):
        Gupta=0
        for i in range(20):
            e1=e2=0
            for j in range(20):
                if i!=j:

e1+=np.exp(-p*(np.sqrt(np.sum(np.square(XYZ[k][i]-XYZ[k][j]))))/d-1))

e2+=np.exp(-2*q*(np.sqrt(np.sum(np.square(XYZ[k][i]-XYZ[k][j]))))/d-1))
                else:
                    e1+=0
                    e2+=0
            Gupta+=A*e1-np.sqrt(xi_2*e2)
        return Gupta

    from sklearn.metrics import mean_squared_error as mse
    from sklearn.metrics import r2_score
    X=XYZ1
    y=energy
    from sklearn.preprocessing import StandardScaler as ss
    X=ss().fit_transform(X)
    #y=ss().fit_transform(y)
    y=(energy-energy.mean())/energy.std()
    from sklearn.model_selection import train_test_split
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.05)
    from sklearn.linear_model import LinearRegression
    from sklearn.tree import DecisionTreeRegressor
    from sklearn.svm import SVR
    from sklearn.neighbors import KNeighborsRegressor
    lr=LinearRegression()
    tree=DecisionTreeRegressor()
    svm=SVR(max_iter=100)
    KNR=KNeighborsRegressor()
    lr.fit(X_train,y_train)
    y_predict1=lr.predict(X_test)
    tree.fit(X_train,y_train)
    y_predict2=tree.predict(X_test)
    svm.fit(X_train,y_train)
    y_predict3=lr.predict(X_test)
    KNR.fit(X_train,y_train)
    y_predict4=lr.predict(X_test)
    print('MSE:',mse(y_predict1,y_test))
```

```
    print('R2-score',r2_score(y_test,y_predict1))
    print('MSE:',mse(y_predict2,y_test))
    print('R2-score',r2_score(y_test,y_predict2))
    print('MSE:',mse(y_predict3,y_test))
    print('R2-score',r2_score(y_test,y_predict3))
    print('MSE:',mse(y_predict4,y_test))
    print('R2-score',r2_score(y_test,y_predict4))

    import xgboost as xgb
    energy=(energy-energy.mean())/energy.std()
    X=XYZ1
    y=energy
    from sklearn.model_selection import train_test_split
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.05)
    XGB=xgb.XGBRegressor(max_depth=                                                    5,
num_boost_round=950,min_child_weight=11, max_iter=100)
    XGB.fit(X_train,y_train)
    y_predict5=XGB.predict(X_test)
    print('MSE:',mse(y_test,y_predict5))
    print('R2-score',r2_score(y_test,y_predict5))

    from sklearn.gaussian_process import GaussianProcessRegressor
    from sklearn.gaussian_process.kernels import RBF, ConstantKernel as C
    kernel = C(0.1, (0.001, 0.1)) * RBF(60, (1e-04, 27))
    X=XYZ1
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.05)
    reg = GaussianProcessRegressor(kernel=kernel, n_restarts_optimizer=60, alpha=0.1)
    reg.fit(X_train,y_train)
    y_predict6=reg.predict(X_test)
    print('MSE:',mse(y_test,y_predict6))
    print('R2-score',r2_score(y_test,y_predict6))

    def r(vec,i,j):
        #vec=vec.T
        return
np.sqrt(np.square(vec[i]-vec[j])+np.square(vec[i+20]-vec[j+20])+np.square(vec[i+40]-vec[j
+40]))
    def V1(vec):
        p,q,A,xi_2,d=1.00129668, 0.99937609, 0.98701858, 0.80920207, 0.8277397
        Gupta=0
        for i in range(20):
            e1=e2=0
            for j in range(20):
                if i!=j:
```

```
                        e1+=np.exp(-p*(r(vec,i,j)/d-1))
                        e2+=np.exp(-2*q*(r(vec,i,j)/d-1))
                    else:
                        e1+=0
                        e2+=0
                Gupta+=A*e1-np.sqrt(xi_2*e2)

        menergy,senergy=-1551.2495687587586,2.8740588124691637
        mv,sv=7.800386281401496,1.609188796167207
        return (mv-Gupta)/sv*senergy+menergy

    from sko import GA
    import time
    start_time=time.time()
    ga=GA.GA(n_dim=60,func=V1,constraint_ueq=cons                    ,lb=[-10]*60,
ub=[10]*60,max_iter=1000)

    general_best,func_general_best=ga.fit()
    end_time=time.time()
    print('best_x:',general_best)
    print('best_y:',func_general_best)
    print('time:',end_time-start_time)

    FitV_history_GA = pd.DataFrame(ga.all_history_Y)
    plt.plot(FitV_history_GA.index, FitV_history_GA.values, '.', color='red')
    plt.xlabel("iterations")
    plt.ylabel("Energy_min")
    plt.show()

    plt_min1 = FitV_history_GA.min(axis=1)
    plt.plot(plt_min1.index, plt_min1, label='max')
    plt.xlabel("iterations")
    plt.ylabel("Energy_min")
    plt.legend(['GA'])
    plt.show()

    def XGBsearch(x):
        y_predict=XGB.predict(np.array([x,x]))
        return y_predict[0]
    start_time=time.time()
    ga=GA.GA(n_dim=60,func=XGBsearch,lb=[-12]*60, ub=[12]*60,max_iter=1000)
    general_best,func_general_best=ga.fit()
    end_time=time.time()
    print('best_x:',general_best)#最优解
```

```
    print('best_y:',func_general_best)#最小函数值
    print('time:',end_time-start_time)#用时
    FitV_history_GA = pd.DataFrame(ga.all_history_Y)
    plt.plot(FitV_history_GA.index, FitV_history_GA.values, '.', color='red')
    plt.xlabel("iterations")
    plt.ylabel("Energy_min")
    plt.show()

    plt_min1 = FitV_history_GA.min(axis=1)
    plt.plot(plt_min1.index, plt_min1, label='max')
    plt.xlabel("iterations")
    plt.ylabel("Energy_min")
    plt.legend(['GA'])
    plt.show()
```

```
#第二问
def r2(vec,i,j):
    #vec=vec.T
    return
np.sqrt(np.square(vec[i]-vec[j])+np.square(vec[i+32]-vec[j+32])+np.square(vec[i+64]-vec[j
+64]))
def V2(vec):
    p,q,A,xi_2,d=1.00129668, 0.99937609, 0.98701858, 0.80920207, 0.8277397
    Gupta=0
    for i in range(32):
        e1=e2=0
        for j in range(32):
            if i!=j:
                e1+=np.exp(-p*(r2(vec,i,j)/d-1))
                e2+=np.exp(-2*q*(r2(vec,i,j)/d-1))
            else:
                e1+=0
                e2+=0
        Gupta+=A*e1-np.sqrt(xi_2*e2)

    menergy,senergy=em,es
    mv,sv=V_Gupta.mean(),V_Gupta.std()
    return (mv-Gupta)/sv*senergy+menergy
cons=[]
for i in range(20):
    cons.append(lambda x:x[i]**2+x[i+20]**2+x[i+40]**2-85.5)
for i in range(20):
    for j in range(20):
        if i!=j:
```

```
        cons.append(lambda x: np.sqrt(np.square(x[i]-x[j])+np.square(x[i+20]-x[j+20])+np.square(x[i+40]-x[j+40]))-2.49)
start_time=time.time()
ga32=GA.GA(n_dim=96,func=V2, constraint_ueq=cons, lb=[-10]*96, ub=[10]*96,max_iter=1200)

general_best32,func_general_best32=ga32.fit()
end_time=time.time()
print('best_x:',general_best32)
print('best_y:',func_general_best32)
print('time:',end_time-start_time)

FitV_history_GA32 = pd.DataFrame(ga32.all_history_Y)
plt.plot(FitV_history_GA32.index, FitV_history_GA32.values, '.', color='red')
plt.xlabel("iterations")
plt.ylabel("Energy_min")
plt.show()

plt_min1 = FitV_history_GA32.min(axis=1)
plt.plot(plt_min1.index, plt_min1, label='max')
plt.xlabel("iterations")
plt.ylabel("Energy_min")
plt.legend(['GA'])
plt.show()
```

附录 2

介绍：B 团簇预测代码

```
def r(vec,i,j):
    return np.sqrt(np.square(vec[i]-vec[j])+np.square(vec[i+45]-vec[j+45])+np.square(vec[i+90]-vec[j+90]))
def REBO(vec):
    A=0.35
    B1=0.04
    B3=0.04
    beta1=0.42
    beta3=0.42
    a=1
    b=2
    Q=3
    E=0
    for i in range(44):
```

```
            for j in range(i+1,45):
                VR=(1+Q/r(vec,i,j))*A*np.exp(-a*r(vec,i,j))
                VA=B1*np.exp(-beta1*r(vec,i,j))+B3*np.exp(-beta3*r(vec,i,j))
                E+=VR-b*VA
      return E

def r(vec,i,j):
      return
np.sqrt(np.square(vec[i]-vec[j])+np.square(vec[i+45]-vec[j+45])+np.square(vec[i+90]-vec[j
+90]))
def LJ(vec):
      lj=0
      epsilon,sigma=-40.17788124,      0.04215116
      for i in range(1,45):
            for j in range(i):
                if r(vec,i,j)>0:
                    lj+=((sigma/r(vec,i,j))**12-(sigma/r(vec,i,j))**6)
      return 4*epsilon*lj

XYZ1=[]
for i in XYZ:
      XYZ1.append(i.T.reshape(135,))
XYZ1=np.array(XYZ1)
from sko.SA import SA
import time
start_time=time.time()
sa = SA(func=LJ, x0=[1]*135, T_max=1, T_min=1e-9, L=300, max_stay_counter=150)
best_x, best_y = sa.run()
end_time=time.time()
print('best_x:', best_x)
print('best_y' , best_y)
print('time', end_time-start_time)

from sko import GA
import time
start_time=time.time()
ga=GA.GA(n_dim=135,func=LJ,    lb=[-10]*135, ub=[10]*135,max_iter=1000)

general_best,func_general_best=ga.fit()
end_time=time.time()
print('best_x:',general_best)
print('best_y:',func_general_best)
print('time:',end_time-start_time)
```

```
start_time=time.time()
ga_REBO=GA.GA(n_dim=135,func=REBO,                                    lb=[-10]*135,
ub=[10]*135,max_iter=1000)

general_best,func_general_best=ga_REBO.fit()
end_time=time.time()
print('best_x:',general_best)
print('best_y:',func_general_best)
print('time:',end_time-start_time)

FitV_history_GA = pd.DataFrame(ga.all_history_Y)
FitV_history_REBO = pd.DataFrame(ga_REBO.all_history_Y)
plt.plot(FitV_history_GA.index,   FitV_history_GA.values*energy.std()+energy.mean(),   '.',
color='red')
plt.plot(FitV_history_REBO.index,
FitV_history_REBO.values*energy.std()+energy.mean(), '.', color='blue')
plt.legend(['L-J','REBO'])
plt.xlabel("iterations")
plt.ylabel("Energy_min")
plt.show()

plt_min1 = FitV_history_GA.min(axis=1)
plt_min2=FitV_history_REBO.min(axis=1)
plt.plot(plt_min1.index, plt_min1*energy.std()+energy.mean(),'r-')
#plt.xlabel("iterations")
#plt.ylabel("Energy_min")
#plt.legend(['L-J','REBO'])
#plt.show()
plt.plot(plt_min2.index,plt_min2*energy.std()+energy.mean(),'b-')
plt.xlabel("iterations")
plt.ylabel("Energy_min")
plt.legend(['L-J','REBO'])
plt.show()

import xgboost as xgb
from sklearn.metrics import mean_squared_error as mse
from sklearn.metrics import r2_score
X=XYZ1
y=energy
em=energy.mean()
es=energy.std()
y=(y-em)/es
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.05)
```

```
XGB=xgb.XGBRegressor()
XGB.fit(X_train,y_train)
y_predict5=XGB.predict(X_test)
print('MSE:',mse(y_test,y_predict5))
print('R2-score',r2_score(y_test,y_predict5))

def XGBsearch(x):
    y=XGB.predict(np.array([x,x]))
    return y[0]

start_time=time.time()
ga=GA.GA(n_dim=135,func=XGBsearch,    lb=[-10]*135, ub=[10]*135,max_iter=1000)

general_best,func_general_best=ga.fit()
end_time=time.time()
print('best_x:',general_best)
print('best_y:',func_general_best)
print('time:',end_time-start_time)

FitV_history_GA = pd.DataFrame(ga.all_history_Y)
plt.plot(FitV_history_GA.index, FitV_history_GA.values, '.', color='red')
plt.xlabel("iterations")
plt.ylabel("Energy_min")
plt.show()

plt_min1 = FitV_history_GA.min(axis=1)
plt.plot(plt_min1.index, plt_min1, label='max')
plt.xlabel("iterations")
plt.ylabel("Energy_min")
plt.legend(['GA'])
plt.show()

def r2(vec,i,j):
    return
np.sqrt(np.square(vec[i]-vec[j])+np.square(vec[i+40]-vec[j+40])+np.square(vec[i+80]-vec[j
+80]))
def LJ2(vec):
    lj=0
    epsilon,sigma=-40.17788124,      0.04215116
    for i in range(1,40):
        for j in range(i):
            lj+=((sigma/r2(vec,i,j))**12-(sigma/r2(vec,i,j))**6)
    return 4*epsilon*lj
```

```
    def r(vec,i,j):
        return
np.sqrt(np.square(vec[i]-vec[j])+np.square(vec[i+40]-vec[j+40])+np.square(vec[i+80]-vec[j
+80]))
    def REBO2(vec):
        A=0.35
        B1=0.04
        B3=0.04
        beta1=0.42
        beta3=0.42
        a=1
        b=2
        Q=3
        E=0
        for i in range(39):
            for j in range(i+1,40):
                VR=(1+Q/r(vec,i,j))*A*np.exp(-a*r(vec,i,j))
                VA=B1*np.exp(-beta1*r(vec,i,j))+B3*np.exp(-beta3*r(vec,i,j))
                E+=VR-b*VA
        return E

    from sko import GA
    import time
    start_time=time.time()
    ga=GA.GA(n_dim=120,func=LJ2,    lb=[-10]*120, ub=[10]*120,max_iter=1000)

    general_best,func_general_best=ga.fit()
    end_time=time.time()
    print('best_x:',general_best)
    print('best_y:',func_general_best)
    print('time:',end_time-start_time)

    start_time=time.time()
    ga_REBO=GA.GA(n_dim=120,func=REBO2,                                lb=[-10]*120,
ub=[10]*120,max_iter=1000)

    general_best,func_general_best=ga_REBO.fit()
    end_time=time.time()
    print('best_x:',general_best)
    print('best_y:',func_general_best)
    print('time:',end_time-start_time)

    FitV_history_GA = pd.DataFrame(ga.all_history_Y)
    FitV_history_REBO=pd.DataFrame(ga_REBO.all_history_Y)
```

```
        plt.plot(FitV_history_REBO.index,
FitV_history_REBO.values*energy.std()+energy.mean(), '.', color='blue')
        plt.plot(FitV_history_GA.index, FitV_history_GA.values*energy.std()+energy.mean(),
'.', color='red')
        plt.xlabel("iterations")
        plt.ylabel("Energy_min")
        plt.show()

        plt_min1 = FitV_history_GA.min(axis=1)
        plt_min2 = FitV_history_REBO.min(axis=1)
        plt.plot(plt_min1.index, plt_min1*energy.std()+energy.mean(), label='max')
        plt.plot(plt_min2.index, plt_min2*energy.std()+energy.mean(), label='max')
        plt.xlabel("iterations")
        plt.ylabel("Energy_min")
        plt.legend(['LJ','REBO'])
        plt.show()
```