Hyperparameter Optimization of NN Models for a Single Water Molecule

Group 10

Final Report

Nicole Hu, Marissa Klee, John Pederson, Chengzhai Wang

Introduction

In this project, we created a surrogate model based on neural networks to predict the minimum-energy configuration of a simple chemical system, a water molecule. We calculated the stable configuration of a water molecule based on the surrogate model with in-house optimization methods, direct search and genetic algorithm (GA). When successfully demonstrating our model on a single water atom, we moved on to investigate the hyperparameter optimization problem of the neural network potentials. We applied GA, Tabu search and Nelder-Mead on three hyperparameters (one continuous, two discrete) to obtain better settings for neural network training. We benchmarked the performance of the implemented methods with available hyperparameter optimization packages on the market, `hyperopt`[1].

Optimization Problem Definition

Problem 1

The first optimization problem we studied is to find the stable configuration of a water molecule by minimizing the potential energy. The first step is to build a neural network potential based on density functional theory calculations of a single water molecule at different bond lengths and bond angles. Details of the reference dataset can be found in section Data. The trained neural network potential ($f(\tilde{\mathbf{R}})$) maps the atomic coordinates ($\tilde{\mathbf{R}}$) to potential energy (E):

$$f(\tilde{\mathbf{R}}) = E$$

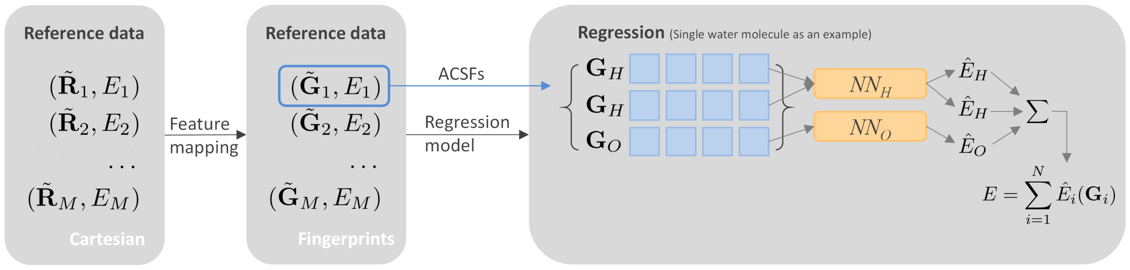
Reference data
$(\tilde{\mathbf{R}}_1, E_1)$
$(\tilde{\mathbf{R}}_2, E_2)$
$\cdots$
$(\tilde{\mathbf{R}}_M, E_M)$
Cartesian
Feature mapping
Reference data
$(\tilde{\mathbf{G}}_1, E_1)$
$(\tilde{\mathbf{G}}_2, E_2)$
$\cdots$
$(\tilde{\mathbf{G}}_M, E_M)$
Fingerprints
ACSFs
Regression model
Regression (Single water molecule as an example)
$\mathbf{G}_H$
$\mathbf{G}_H$
$\mathbf{G}_O$
NN_H
NN_O
$\hat{E}_H$
$\hat{E}_H$
$\hat{E}_O$
$\sum$
$E = \sum_{i=1}^{N} \hat{E}_i(\mathbf{G}_i)$

Figure 1: Flowchart of a Behler-Parrinello high-dimensional neural network to predict atomic energy contributions for a single water molecule.

We perturbed the atomic configurations of a water molecule by rattling the atoms away from their equilibrium bond length and bond angle. The non-linearity associated with the model would not be ideal to solve with traditional non-linear solvers, as implemented through `pyomo`. Therefore, we decide to implement heuristic and sampling-based optimization methods directly to energy minimization of the surrogate model.

Optimization Formulation: The objective is to minimize the approximated energy calculated by the neural network potential. The variables are the x, y, z positions of three atoms. This NLP problem has a formulation as shown below:

$$\min(f(\tilde{\mathbf{R}}))$$
$$\text{s.t. } \tilde{\mathbf{R}} \in \mathbb{R}_0^+$$

Problem 2

The second optimization problem we propose is to optimize the hyperparameter settings of the neural network potential for the chemical system of a single water molecule. The loss function of the neural network potential is defined below[2]:

$$\mathcal{L} = \frac{1}{N_{\text{struct}}} \sum_{i=1}^{N_{\text{struct}}} \left[\left(E_{\text{NN}}^{i} - E_{\text{Ref}}^{i}\right)^{2} + \frac{\beta}{3 \cdot N_{\text{atom}}^{i}} \sum_{j=1}^{3N_{\text{atom}}^{i}} \left(F_{j,\text{NN}}^{i} - F_{j,\text{Ref}}^{i}\right)^{2} \right]$$

where N_{struct} is the number of training images, E_{NN}^{i} is the predicted potential energy, E_{Ref}^{i} is the ground truth potential energy generated through DFT calculations, β is the force coefficient, N_{atom}^{i} is the number of atoms in the image, $F_{j,\text{NN}}^{i}$ is the predicted force component, and $F_{j,\text{Ref}}^{i}$ is the ground truth force.

Optimization Formulation: The training process of the neural network potential can be thought of as a black-box function ($g(\vec{x})$), which has the variables of multiple hyperparameters ($\vec{x}$) and yields the final error metrics of the surrogate model. We identified hyperparameters for this problem as learning rate (continuous), number of nodes of neural networks (discrete) and number of layers of neural networks (discrete). The optimization formulation here would be:

$$\begin{aligned} &\min(g(\vec{x})) \\ &\qquad \vec{x} = [\text{learning rate (continuous, non-negative reals)}, \\ &\qquad\quad \text{number of nodes (discrete, } (10, 50)) \\ &\qquad\quad \text{number of layers (discrete, } (2, 10))] \end{aligned}$$

In this problem, we would like to explore the optimization approach by implementing heuristic methods like GA and Tabu search to optimize the hyperparameters. Instead of using existing solvers, we would focus more on local optimization algorithm implementations and make comparison with hyperparameter optimization software, `hyperopt`.

Data Description

The dataset for the optimization study consisted of a single water molecule with perturbed bond lengths and bond angles. The energies and forces are calculated via DFT. Quantum Espresso is used for DFT calculations with PBE exchange-correlation, 500 plane-wave cutoff, and K-point of (1, 1, 1). The total dataset consists of 400 images, 3 atoms per image.

Every data points has 3 features, the coordinates in 3D for every atom. The atomic coordinates are then used to calculate the corresponding atomic fingerprints as shown in Figure 1. Depending on the choice of parameters for atomic fingerprint generation, the number of features for the input of neural network model might vary. In this case, we chose a set of parameters that would yield a total of 80 dimensions.

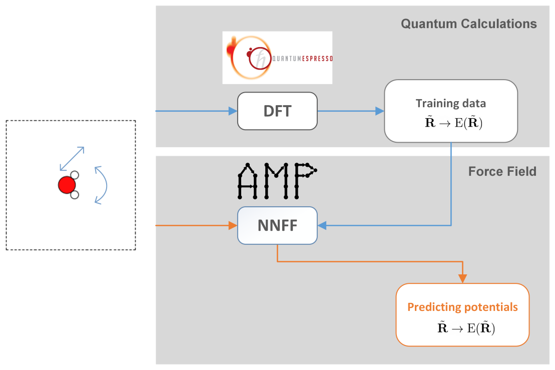

Figure 2: Schematic of data generation and training process in `AMP`[2] for neural network potentials. On the left hand side is a snapshot of a single water molecule in the training dataset.

Figure 3 plots the 2D potential energy surface mapped out by enumerating the O-H stretch and H-O-H bend while fixing the other O-H stretch at equilibrium bond length for better visualization.

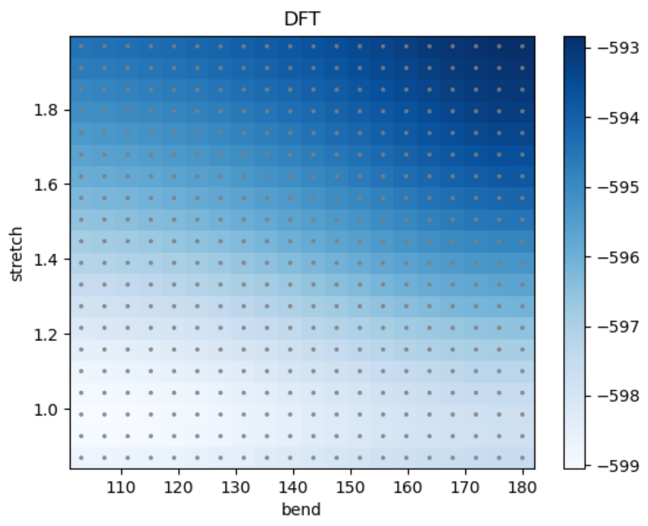

Figure 3: A schematic of the 2D potential energy surface mapped from H-O-H bend and O-H stretch as x and y axes from 400 training data points. The z-direction represented by the color map is the calculated potential energy in eV via DFT calculations. Training data are marked as gray points in the plot.

Optimization methods and solvers

Problem 1: Stable configuration with least potential energy

We build a surrogate NNP model based on 360 training images and 40 test images via `AMPtorch`. The training MAE is 14.1 meV and the test MAE is 15.0 meV, both lower than chemical accuracy 43 meV.

For the optimization part, we manually implemented two gradient-free methods, direct search and genetic algorithm (GA). We compared the results to geometry optimization implemented through DFT calculations and optimization based on Nelder-Mead implemented in `SciPy`.

For the first optimization problem, we implemented a heuristic optimization method: the GA. This algorithm starts with a population of randomly generated individuals. Then the heuristic is used to improve the population using natural selection concepts: inheritance, mutation, crossover, and selection. For every generation, we rank the population of individual solutions by the potential energy calculated from surrogate model. We then select the individuals with the minimum energy before crossing over with bond angles and bond lengths. It will also mutate some of the solutions based on uniform distribution and record the best

Algorithm 1 Genetic algorithm (GA). NNP is short for neural network potential.

```
1: Generate a population of $N_{children}$ solutions randomly from the bound;
2: for each $k \in N_{genenration}$ do
3:     Rank population, $P_{k-1}$
         Compute $fitness(i)$ for each $i \in P_{k-1}$;          ▷ $fitness(i) = -NNP$
4:     Select $\chi$ best parents, $P_k$
         Based on ranked population, $P'_{k-1}$;
5:     Crossover
         Select $\gamma$ members of $P_k$; exchange variables as children; insert children into $P_k$
6:     Mutate
         Invert a randomly-selected variable based on $p_{mute}$;
7: end for each
```

pairings. The main shortcoming, besides low speed, is that heuristic methods will not yield an exact solution but a solution in close proximity to the actual result. Algorithm 1 shows a pseudocode of GA.

Additionally, we implemented a positive-spanning basis direct search algorithm to the geometry optimization problem. This direct search algorithm follows the principles of searching over a positive spanning set. The initial guess is taken to be the center of the first iteration of the algorithm, and $N + 1$ other points are selected in the directions of the minimal positive spanning basis, where N is the dimensionality of the problem. This basis is considered to the be the unit vectors aligned with each positive axis plus a unit vector which is 45 degrees from each $N - 1$ dimensional hyperplane composed of the negative axes. These unit vectors are scaled by some factor, a, and added to the center-point to get the remaining samples. All of these $N + 2$ points are evaluated using the neural network model. If any sample point is found to fall outside of the input space feasible boundaries, such as 0 and 180 degrees or 0 Å in this case, its function value is set to positive infinity. The point which has the lowest function evaluation will then become the new center-point. If this point is the same as the previous center-point, then this means that the gradient over the potential surface is smaller than vector scaling factor, a. At that point, the vector scaling factor, a, will be reduced by an amount determined by the user. The algorithm will proceed until the vector scaling factor decreases below some pre-determined tolerance or until a maximum number of iterations has been reached. This is algorithm is summarized by Algorithm 2 below.

Algorithm 2 Positive spanning set direct search Algorithm. NNP is short for neural network potential.

```
1: Determine an initial center-point $x_c$;
2: Calculate the set of $N + 1$ vectors in the minimal positive span of the input space, $S \in \mathbb{R}^N$;
3: while $a > \epsilon$ & $i < N^{max}_{iterations}$ do
4:     Get points non-center points, $x_j \in X$
         Compute locations of non-center points by adding each vector $s_j \in S$ scaled by $a$ to the center-point;
5:     Evaluate points, $f(x)$, using NPP
         Set function value of points outside of input bounds to positive infinity;
6:     Compare
         Select $x_j$ with lowest function value to be the new center, $x_c$;
7:     Decrease $a$ by some amount $\delta a$;
         if $x_c^i = x_c^{i-1}$;
8: end while
```

Results

Problem 1: Stable configuration with least potential energy

We obtained results with manually implemented GA and direct search method as shown in Table 1. To demonstrate how direct search converges to a local optimal point, Figure 4 tracks the solutions from every iteration at a starting point of 120 degrees and 1.5 angstroms for the bond angle and bond distance, respectively. Notice in this figure that the center points track through the input-space along the positive spanning vectors, which, in this case, are the along the positive bond angle axis, the positive bond distance axis, and the bisector of the negative bond angle and bond distance axis. The center point appears to move along the negative bond angle axis in later iterations, but this is due to the scaling factor approaching zero along the bond distance axis such that it does not account for a significant amount of the magnitude of the negative axis bisector vector. The algorithm then settles at a point of convergence after several iterations.

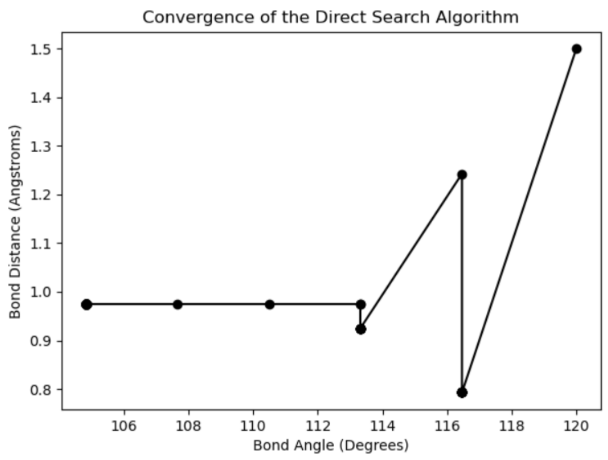

Figure 4: Center-points of the direct search algorithm as it converges to the ground state bond angle and bond distance from an initial guess of 120 degrees and 1.5 angstroms, respectively.

Table 1 compares DFT geometry optimization results as ground truth, GA, direct search method and `SciPy` implementation of Nelder-Mead as a benchmarking system. Among current implementations, direct search method gives a solution closest to DFT calculations within 0.38% and 0.83% for stretch and bend, respectivelly. Next along the line, the benchmarking Nelder-Mead yielded errors on stretch and bend of 0.41% and 1.23%. Since GA has a stochastic initialization process, we ran GA 20 times with different random seeds to generate an ensemble of solutions and took the mean of the best 20 solutions as our final results. GA ensemble method performs the worse, giving errors of stretch and bend at 1.13 ± 2.38% and 3.34 ± 4.28%.

Table 1: This table compares optimizations results from different methods. DFT geometry optimization is taken as ground-truth and is compared to different optimization methods based on surrogate models.

Method	**Stretch**	**Stretch Error %**	**Bend**	**Bend Error %**
DFT	0.9710	-	103.9690	-
GA	0.9820±0.0231	1.1329 ± 2.3790	107.4511±4.4482	3.3492±4.2784
Direct search	0.9747	0.3806	104.8326	0.8306
Nelder-Mead	0.9750	0.4119	102.6911	1.2291

Optimization methods and solvers

Problem 2: Neural network potential hyperparameter optimization

For the second optimization problem on hyperparameter tuning, we implemented GA and Tabu search to compare the performances with the designated hyperparameter optimization toolkit, `hyperopt`. The optimization methods are adapted to hyperparameter optimization from Problem 1 by changing the fitness evaluation to evaluate the test mean average errors (MAEs). GA is particularly useful for this problem as it is compatible with discrete and categorical variables like number of layers and nodes and activate functions.

Additionally, we sought to modify the positive spanning set direct search algorithm in order to non-continuous inputs to serve as another basis of comparison with the algorithms described above. The theory for positive spanning sets does not hold when considering discrete, non-differentiable feature (or, in this case, hyperparameter) spaces, and so a modified approach was required in order to handle these inputs. Although it cannot guarantee convergence to local optima, We implemented a form of the Tabu search (TS) algorithm for the hyperparameter optimization [3]. Tabu search is a heuristic local search method, and the basic idea of TS is to employ a local, or neighborhood, search procedure to generate "moves" that result in neighborhood solutions. For each current move, all neighborhood solutions are evaluated with the objective function, and the best point is selected to be the center of the new neighborhood of potential solutions. This method can avoid the tendency to be stuck in suboptimal regions by constantly replacing a recent solution with a best non-visited neighboring solution within solution space. The algorithm tracks and adds recent neighborhood centers to a list of taboo solutions to not revisit until a given number of iterations. The procedure will stop at meeting a predetermined criterion, which, in our case, is a maximum number of iterations. The algorithm, as implemented for this work, is described below in Algorithm 3.

Algorithm 3 Tabu search Algorithm. NNP stands for Neural Network Potential.

```
1: Determine an initial center-point $x_c$ for the first neighborhood of points;
2: while $i < N_{iterations}^{max}$ do
3:     Get other points in neighborhood, $x_j \in X$
         Compute locations of non-center points by adding and subtracting 1 for integers or some increment,
         $a$, for continuous variables for a total of $2d$ points, where $x_j \in \mathbb{R}^d$;
4:     Evaluate points, $f(x)$, by training and evaluating NNP
         Set function value of points outside of input bounds or on the Tabu list to positive infinity;
5:     Compare
         Select $x_j$ with lowest function value to be the new center, $x_c$, and record $x_j$ with lower overall
         function value than any point previously evaluated;
6:     Add new center point, $x_c$ to Tabu list
         if $L_{Tabu} > L_{max}$, remove first entry from Tabu list;
7: end while
```

Results

Problem 2: Neural network potential hyperparameter optimization

Table 2 compares the performances of four methods, two baseline models, Nelder-Mead and `hyperopt` package as the current state-of-the-art, and our implemented GA and Tabu algorithms. All methods prefer a lower learning rate, and a shallow neural network with a small amount of layers (2-3) and a large number of nodes (14-30) for optimal generalizability measured by test mean average errors (MAEs) with the exception of Tabu search. Overall, `hyperopt` has the least test MAEs for total trial numbers of 100. Then comes GA with a performance a bit worse than `hyperopt` (~2 meV) but scores ~10 meV lower test MAEs than Nelder-Mead. Finally, the Tabu search algorithm performs worse than even the baseline Nelder-Mead, with an MAE ~6 meV higher than that algorithm and ~18 meV higher than that optimized using `hyperopt`. We also investigated implementing Nelder-Mead search after optimal GA results which gives a small performance boost (0.2 meV decrease in test MAE) at the cost of evaluating 20 more models.

Table 2: This table compares hyperparameter optimization results for three variables, learning rate, number of nodes and number of layers, containing continuous and discrete values.

Method	Learning Rate	Number of Nodes	Number of Layers	Test MAE (meV)
Nelder-Mead	1.03×10^{-3}	19	3	21.7
GA	2.15×10^{-3}	45	2	10.8
Tabu	1.00×10^{-3}	14	17	27.0
`hyperopt`	1.43×10^{-3}	30	3	8.8
GA + Nelder-Mead	2.15×10^{-3}	45	2	10.6

The Tabu search, although an elegant solution to non-differentiable input spaces in theory, did not converge to hyperparameters that were comparable with the other optimization methods. This was due to a number of factors, all of which may be said to stem from the limits of available computational resources. The algorithm itself has parameters which must be tuned or estimated before reliable implementation, such as the length of the Tabu list or the number of pre-determined iterations. Due to the fact that the function evaluations are computationally expensive neural network trainings and evaluations, even performing twenty iterations of the method required several hours of computation on available resources. As such, there was not a great opportunity to tune the hyperparameter optimization. Additionally, the method is very sensitive to the initial guess, so better guesses will lead to better convergence, but this would not necessarily be known *a priori*. The result given by the table above used a Tabu list length of four, a maximum number of iterations of twenty and an initial guess of fifteen nodes per layer, fifteen layers, and a learning rate of 0.001. Although Tabu is designed to be adventurous, this set of parameters did not allow the search to proceed far from this initial point.

Figure 5 traces the evolution of trials and best models in GA's case. Nelder-Mead and `hyperopt` both have 100 trials plotted as the lower x-axis, GA has 8 generations as the upper x-axis and only the best model in every generation is plotted. Nelder-Mead evaluate models at close proximity to initial guesses and therefore is not able to capture or test better models that are distinct from the initial guess. `hyperopt`, to the contrary, would go beyond the regions that are likely local optimal in order to have a better search results. GA, compared to `hyperopt`, has the trend to obtain a better overall result after generations. The best parent of GA is plotted against number of generations (upper x-axis). Nelder-Mead and `hyperopt` are plotted against the number of trials (lower x-axis).

Compared with a sampling-based heuristic method like GA, `hyperopt` uses Gaussian Processes as surrogate models to approximate the original response space[1] via sequential model-based global optimization (SMGO). Specifically, `hyperopt` uses tree-structured Parzen estimator (TPE) to model $p(x|y)$ and $p(y)$. The active learning-like SMGO algorithm estimates $p(x|y)$ from two different densities, one from observations less than a defined loss, $l(x)$, and another by the rest observations, $g(x)$:

$$p(x|y) = \begin{cases} l(x) & \text{if } y < y^* \\ g(x) & \text{if } y \geq y^* \end{cases}$$

The TPE then chooses some y^* aggressively at γ quantile of the observed y values to obtain $p(y < y^*) = \gamma$. Modeled $p(x|y)$ and $p(y)$ from TPE are then used to optimize the expected improvement. TPE algorithm implemented in `hyperopt` has more information based on the surrogate model given enough sampling to estimate the response surface while GA relies solely on a stochastic sampling process for improvements. We think using surrogate models to approximate the response surface and estimating the expected improvement help TPE (`hyperopt`) outperform GA in this case. We would like to refer to the paper should more information on TPE or SMGO be needed[1]

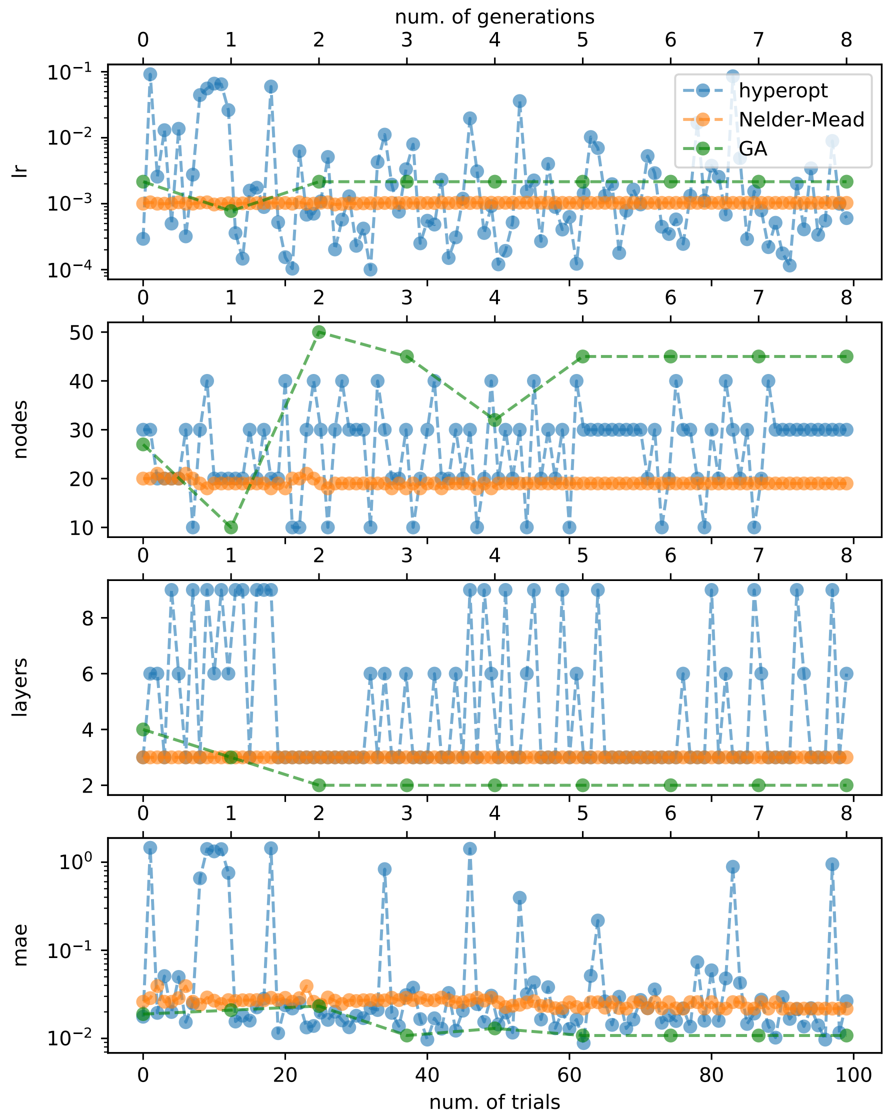

Figure 5: Comparison of training results based on different optimization algorithm in the sampling space, including learning rate (lr), number of nodes and number of layers, and test mean average errors (MAEs).

To make GA a better optimization algorithm in this applied problem, we identified some issues and implemented some changes to crossover and mutation methods of GA. Based on the undesired slight increase in MAE after generations in Figure 5, we noticed that the best parent in every generation is also subjected to mutation. To resolve this, we reserved the best 2 parents from previous generation and made them immune to mutations to assure an decreasing trend in MAE. Additionally, we add an average row of the best parents in crossover solutions to provide more diversity. These new features did gave the same answer as the previous GA shown in Table 2, but the searching process requires less number of generations, indicating a reduced computational cost.

Conclusion

In this work, we self-implemented a gradient-based method, direct search by positive spanning sets, and a heuristic method, genetic algorithm on a continuous space 2D geometry optimization problem. Furthermore, we also compared GA and Tabu search with current hyperparameter tuning package, `hyperopt` to hyperparameter optimization.

Problem 1: Stable configuration with least potential energy

In the minimalistic 2D PES geoemtry optimization, Direct search method works well because optimization (Figure 3) does not have many local optima, and within tolerance, it is able to get very close to the accurate optimization results based on gradients. The performance of GA is expected because heuristic methods like GA would yield results in close proximity to the actual solution, but cannot solve the problem exactly. However, the DFT geometry optimization results are still well within the bounds given by ensembles of GA optimizers.

Problem 2: Neural network potential hyperparameter optimization

Table 2 and Figure 5 showed that a heuristic method like GA is able to find better hyperparameter settings and has better performance than method like Nelder-Mead alone. Although the current performance is a bit worse than `hyperopt` package, we believe the GA is still a viable stochastic solution to hyperparameter optimization problems given enough computational resources.

Although GA did not yield the best results in the hyperparamter optimization problem, the flexibility of GA and easy adaptations to variable types, categorical, integers, binaries and continuous, and customized crossover or mutation methods indicate the potential to perform better in reduced computational costs or better search results.

Individual Contribution

Each member of this group contributed in different ways. Nicole obtained the original data and prepared it for our optimization. For the two optimization coding problems, Nicole and Marissa took the GA algorithm and Chengzhai and John focused on the direct search. The group also divided the writing sections of this final report. John formulated the description for direct search, Nicole wrote the sections on GA algorithm, optimization formulation and data, Marissa drafted the introduction and GA algorithm portions and Chengzhai contributed to the introduction, the results and drafted introduction of Tabu Algorithm.

Please refer to the list below for details.

Nicole Hu = NH, Marrisa Klee = MK, John Pederson = JP, Chengzhai Wang = CW

1. Data and surrogate model - [NH]
2. Codes
   (a) GA of Problem 1 - [NH, MK]
   (b) Direct search of Problem 1 - [JP, CW]
   (c) GA of Problem 2 - [NH, MK]
   (d) Tabu search of Problem 2 - [JP, CW]
   (e) `hyperopt` baseline of Problem 2 - [NH]
3. Report
   (a) Introduction - [MK, CW]
   (b) Data Description - [NH]
   (c) Description and discussion of GA in Problem 1 - [MK, NH]
   (d) Description and discussion of Direct Search in Problem 1 - [JP, CW]
   (e) Description and discussion of Tabu search in Problem 2 - [JP, CW]
   (f) Discussion of GA in Problem 2 - [MK, NH]
   (g) Conclusion - [NH, MK, JP, CW]

References

[1] J. Bergstra, D. Yamins, and D. D. Cox. Making a Science of Model Search: Hyperparameter Optimizationin Hundreds of Dimensions for Vision Architectures. Technical Report 1, 2013.

[2] Alireza Khorshidi and Andrew A. Peterson. Amp: A modular approach to machine learning in atomistic simulations. *Computer Physics Communications*, 207:310–324, 10 2016.

[3] Peng Hao, Ziran Wang, Guoyuan Wu, Kanok Boriboonsomsin, and Matthew Barth. Intra-platoon vehicle sequence optimization for eco-cooperative adaptive cruise control. In *IEEE Conference on Intelligent Transportation Systems, Proceedings, ITSC*, volume 2018-March, pages 1–6. Institute of Electrical and Electronics Engineers Inc., 3 2018.