CS7641 Machine Learning Assignment II
Randomized Optimization

Wang Lu, GTid: 903355610
10th October ,2018

Find Optimal Weights for ANN

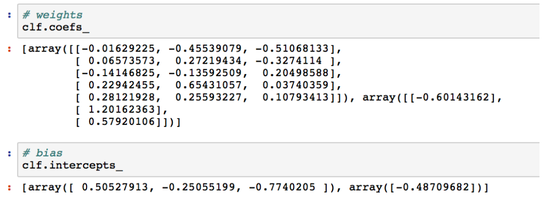
Here I choose the dataset I for task one in last assignment: predicting individuals with income more than 50K USD from demographic statistics, which is an imbalanced binary classification problem with originally 8 categorical features and 5 numerical features on 45K entries. This ANN optimization experiment is composed of two parts: **an initial toy project with a 5-input extremely simple network and the second part, the close study on the convergence characteristics of optimization algorithms.** (Due to the length limit of Abagail[1]'s Share.Instance class, I have to reduce the 103 features used in assignment1 to less than 64. 41 variables one-hot encoded from the categorical feature 'native-country' were not included and final inputs size were made 62.)
Notice: 1) No cross validation is conducted since we care only about optimizing the network to best fit the data; 2) Sum of squared error and accuracy score are both collected and used as metrics. But only one of the two metrics is shown for a specific experiment due to the limit of paragraph.

Toy Project: "Under-Challenge" the Algorithms

Unsatisfied with only treating the network of interest as black box, I started by exploring the simplest possible scenario with only 5 numerical features, ['age', 'education-num', 'capital-gain', 'capital-loss', 'hours-per-week'] (normalized). The optimal hidden_layer structure (3 nodes, single layer) is chosen by re-experimenting different 'hidden_layer_sizes' on backpropagation ANN model. Less hidden neurons are shown uncapable of modeling the trend properly and brings high bias to the network; more hidden neurons instead make the classifier vulnerable of overfitting into random noise. (Number of nodes in input layer was set to 5, equals to the number of features. Only one node was used for output because it's a simple binary problem.) This network structure was applied across all of the training models.

A glance of parameters at initialization point and convergence point

For the toy project the optimization methods need only to find suitable values for 5 * 3 + 3 * 1 = 18 weights and 3 + 1 = 4 biases[2], which allows us to understand the way of how the hypothesis space is searched in a much clearer and easier manner. Use the backpropagation network[3] as an example, the weights and biases are initialized as stochastic small numbers to reduce overfitting risk from beginning:

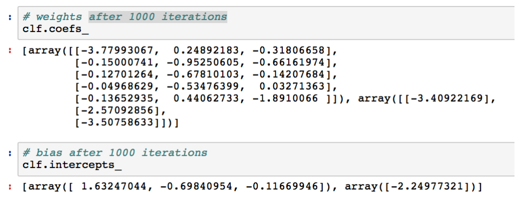


```
# weights
clf.coefs_
```

```
[array([[-0.01629225, -0.45539079, -0.51068133],
        [ 0.06573573,  0.27219434, -0.3274114 ],
        [-0.14146825, -0.13592509,  0.20498588],
        [ 0.22942455,  0.65431057,  0.03740359],
        [ 0.28121928,  0.25593227,  0.10793413]]), array([[-0.60143162],
        [ 1.20162363],
        [ 0.57920106]])]
```

```
# bias
clf.intercepts_
```

```
[array([ 0.50527913, -0.25055199, -0.7740205 ]), array([-0.48709682])]
```



```
# weights after 1000 iterations
clf.coefs_
```

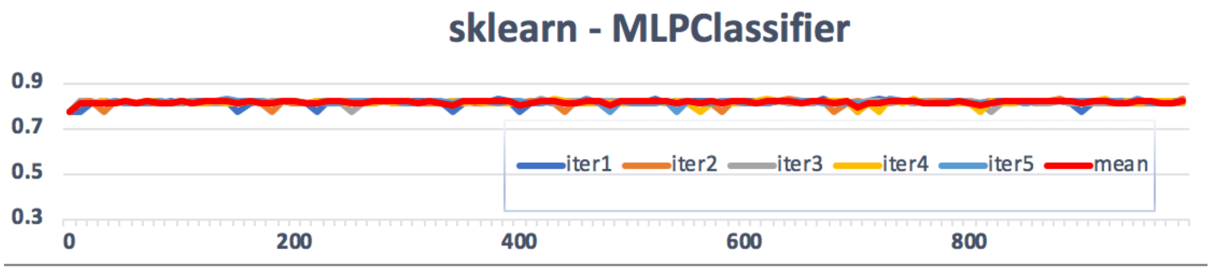
```
[array([[-3.77993067,  0.24892183, -0.31806658],
        [-0.15000741, -0.95250605, -0.66161974],
        [-0.12701264, -0.67810103, -0.14207684],
        [-0.04968629, -0.53476399,  0.03271363],
        [-0.13652935,  0.44062733, -1.8910066 ]]), array([[-3.40922169],
        [-2.57092856],
        [-3.50758633]])]
```

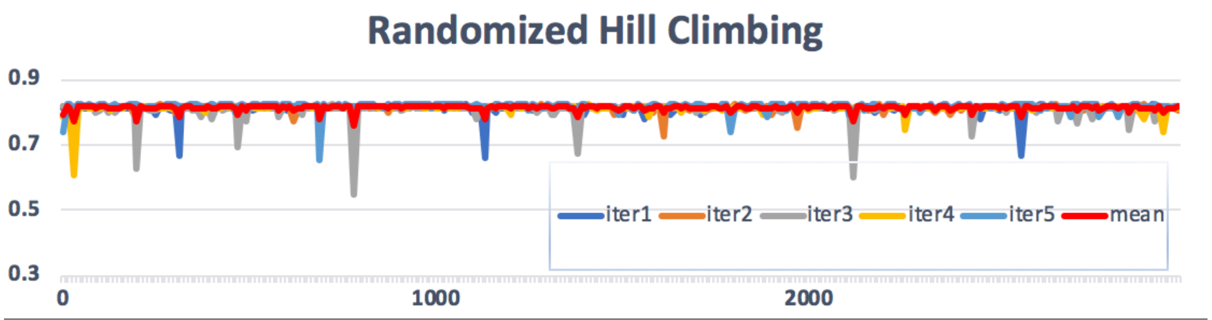
```
# bias after 1000 iterations
clf.intercepts_
```

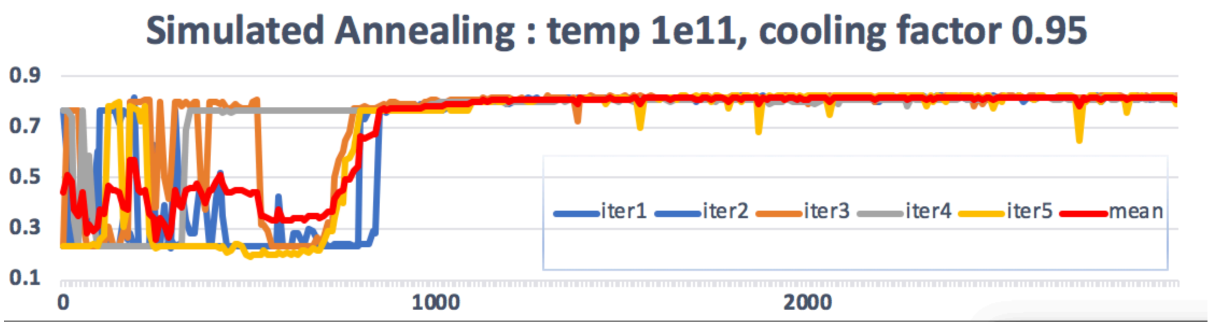
```
[array([ 1.63247044, -0.69840954, -0.11669946]), array([-2.24977321])]
```

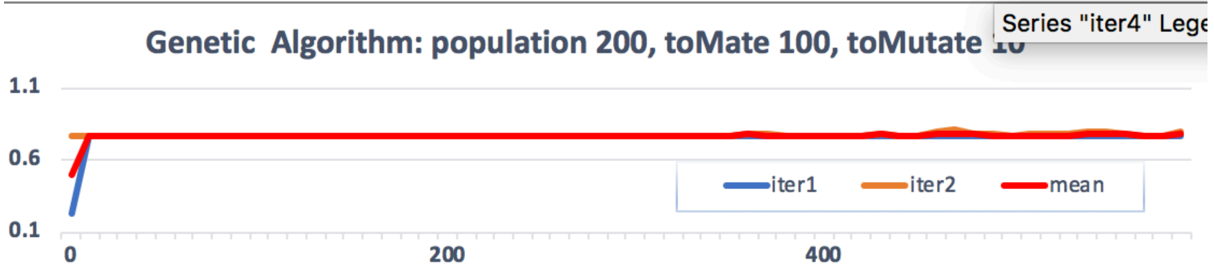
After 1000 iterations, model converges and some of the parameters get bigger in absolute value. (Notice the values didn't grow unlimitedly with iteration due to the existence of penalty term.)

Comparison of performance with sklearn's MLPClassifier as a standard









Upper limit of expressiveness
3 of the models, MLPClassifier, Randomized Hill Climbing and Simulated Annealing reach high score quickly with only tens of iterations and converge around accuracy 82.+ % due to simplicity of the searching problem. But it's interesting to notice the stochastic convergence pattern for all of them. For the case of a very expressive network we would expect the score continuously grows until reaches 100% given enough training time. However, the expressiveness of network used here is limited by the number of parameters it contains. One hidden-layer with 3 hidden nodes allows the network to represent some Boolean functions (though not all, since 3 << 2**5) and a bunch of continuous functions. Thus, the major variance is captured, while modeling the residue information and noise is beyond its capability. It must be noticed that weights (and biases) in networks are **continuous values** and the hypothesis space is therefore infinite. However, VC dimension of the model is finite and thus allows PAC learning.

Convergence depending on initialization states
Especially for simulated annealing which wasted many initial rounds wandering around the neighborhood, when exactly the algorithms are going to converge is difficult to predict and subject to randomness. As elaborated by the big divergence among the 5 trials of simulated annealing experiments, iterations needed before convergence depends heavily on their initialization states, probably because the network is too simple to differentiate good and bad performers. A more complicated optimization problem such as the one in next part should give a better examination on the true capabilities of the models.

Local optima
Besides, there are quite a few traps (local optima) that could deceive and hold the algorithms, which results no improvement for the algorithms after continuous training despite the algorithms manage to escape most of time. For example, Genetic algorithm was heavily stuck at accuracy 76.9%; Simulated annealing got fooled at 23.1% in all trials. RHC stumped a few times but escaped quickly. This is consistent with intuition that “Take not a musket to kill a butterfly”.

Time and computational cost (22-variable optimization)
Notice Backprop run on a different platform:

	MAX accuracy	MIN SumofSquaredError	Iterations taken to converge	converged accuracy	time (sec) per iter
Backprop (5 trials)	83.50%	84.5	<10	83%	~~0.01
RHC(5trials)	82.80%	62.6	~10	82%	0.0115
SA (5 trials)	82.80%	62.6	<800	82%	0.0116
GA (2 trials)	80.70%	78.4	~10	76.90%	1.17

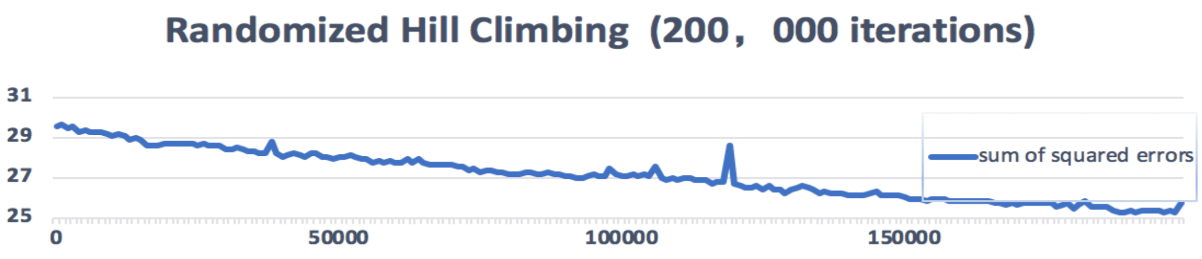
Another useful observation is about the time and computational cost. RHC and SA are equally cheap; cost of backpropagation is also cheap but varied around 0.01 sec per iteration due to infrastructure difference (run in python environment); GA is 100 times as expensive as the former algorithms However, GA converges very fast at least under this setting. Cutting iteration rounds saved lots of time without changing the results.

In summary
1) The toy network has limited ability in describing the data, and the upper limit is near accuracy 84%. **2)** Initialization affect final convergence. **3)** Even in this simple network, there are many local optima, among which the two basins of with corresponding accuracy 23.1,76.9% caused big trouble for algorithms. **4)** Backpropagation network leads the performance, while the simple `restart many times` strategy of RHC is proved very successful for simple scenarios.

A Big Neuron Network: Challenge the Algorithms

Again, I did some preliminary experiments to find the optimal structure for 62-input network. And it turned out (30,) is most suitable hidden layer sizes. Now this network has 62 * 30 + 30 * 1 = 1890 weights and 30 + 1 = 31 biases waiting to be optimized, which is 100 times bigger in size than the toy one. This time the problem is complex enough to allow a gradual convergence and more attention is put on model tuning.

Expressiveness and iterations required to reach ceiling



The new network is so expressive that accuracy keeps increasing (93.7 % at 200,000 iteration). This trend is more obvious using sum of squared error as metric: it keeps getting better and better with more iterations and moreover, this process doesn't even slow down. With single go takes 3994 secs yet hit the upper limit, it's reasonable to deduce that this network has potential to model the dataset with 100% accuracy with extremely unrealistic long time. The final experiment iteration is set to 10000 times.

Time and computational cost (1921-variable optimization)
Notice Backprop run on a different platform:

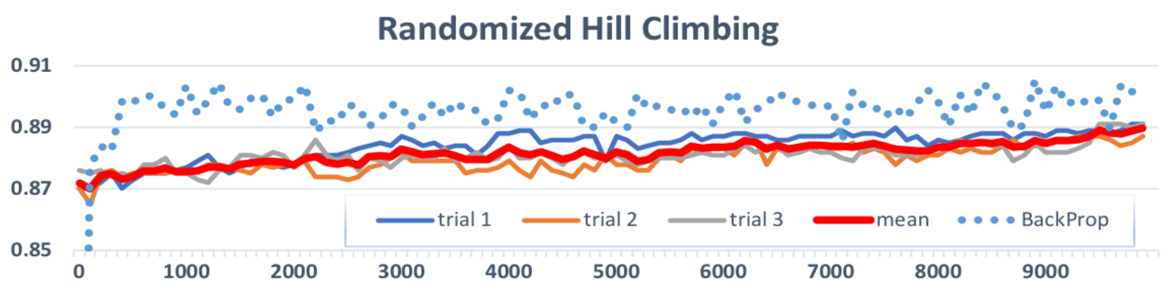
	MAX accuracy	MIN SumofSquaredError	Iterations taken to converge	converged accuracy	time (sec) per iter
Backprop (2 trials)	91.2%	44.0	~ 500	90%	0.0016
RHC (3 trials)	89.1%	43.04	(Gradually)	88%	0.025
SA (3 trials)	87.2%	46.51	< 4000	84%	0.024
GA (2 trials)	84%	58.26	~ 200	81%	2.06

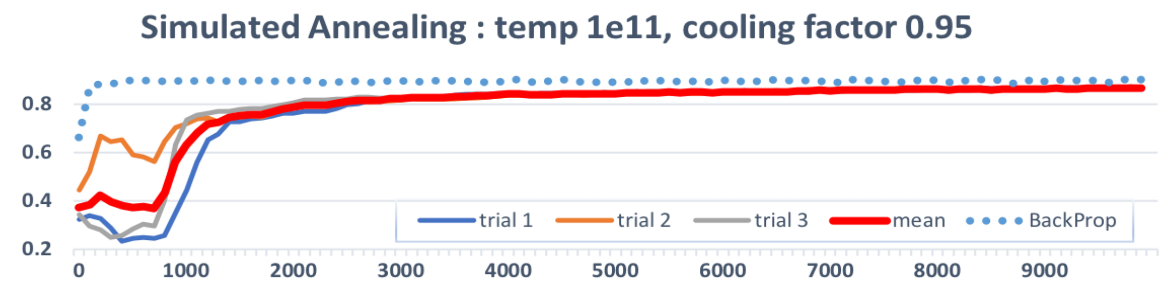
Although the optimizers have 100 times more variables to fit, the time cost per iteration didn't increase linearly: SA and RHC require 20 folds of time per iteration now. GA just need 2 times. This indicates the updating of each parameters in are not totally independent process, and that Genetic Algorithm benefits more by taking advantage of information, which is possibly due to the nature of this algorithm: maintaining a structure by mating and producing generation-wise (iteration and iteration) information flow.

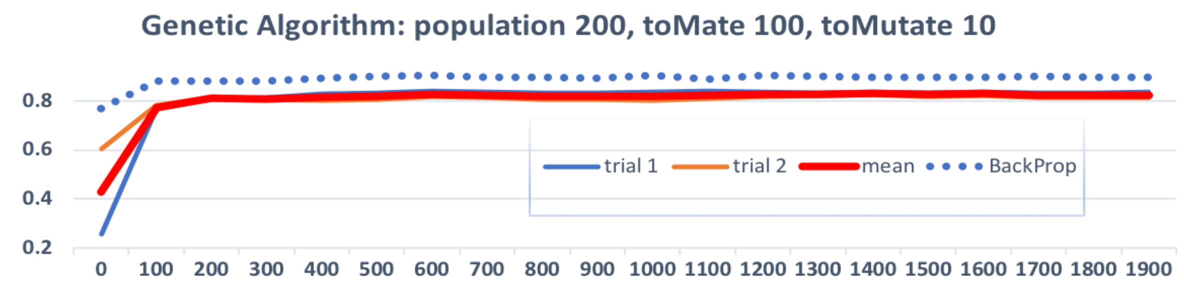
RHC and SA share similar computational characteristics. Actually, looking at the pseudocode we know that the only computational difference is the extra decision-making step SA takes when found no improvement on neighbor: a probabilistic value P $=e^{(f(xt)-f(x))/T}$ of how likely the algorithm would choose to continue, which is recessive compared to the evaluation and restarting cost.

Performance of 3 optimizers using their default settings
Since GA is 100 times more expensive than the other two while usually converges fast, a reduced number of iterations (2000) and repeats (2) was used for GA's part. 10000 iterations x 3 trials were used for RHC and SA. Accuracy curve of backpropagation averaged from 2 rounds, each with 10000 epochs, is used as a control across three groups (plotted as dotted blue lines). Notice it may look rough in the upper panel while much smoother in the lower graphs, but that's due to the choice of different y axis, instead of a change of standards. The thick red line is the aggregated curve in all graphs.

Similarly, RHC hit a high score at very beginning. Genetic Annealing was the second to converge, while SA took the longest time. However, the arrival of stable phase took 10 ~ 100 more rounds than in the toy project, consistent with the intuition that optimization on a more complex network requires also more time and efforts.

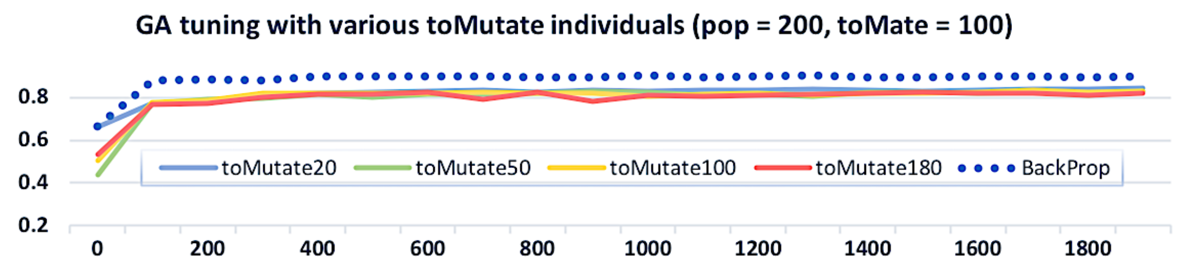




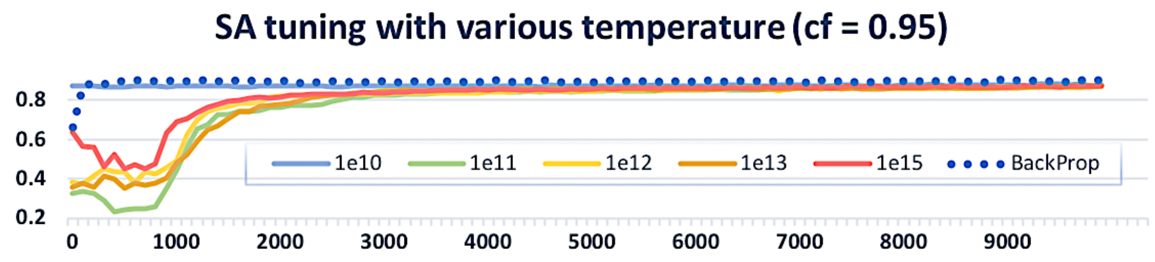


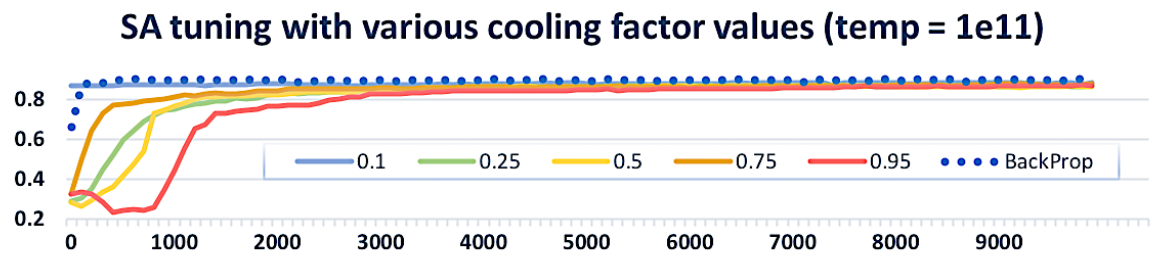
Hyperparameters tuning for Simulated Annealing
The gradient change from cold tones to warm tones (blue-green-yellow-orange-red) are used to represent an increasing of initial temperature from 1e10 to 1e15 in the upper graph. The same color coding is used in lower graph as well but to represent increasing cooling factor value, as bigger cooling factor value equals to slower cooling process/higher overall temperature. (The blue dotted line is from the same backpropagation experiments.)

Hyperparameters tuning for Genetic Algorithms



GA are shown in previous sections as worst performer and vulnerable to local optima. Introducing variability to population by intuition should improve its performance. However, the results tell another story: even mutating 90% of the population renders no relative superiority to algorithm. The true reason is hard to deduce due to the complicated intervening between stochastic crossover mating, mutation and selection process. Further exploration on other parameters wasn't implemented due to time concern.





Similarities and differences of cooling factor and temperature in their way affecting SA
We can see cooling factor and temperature seem have much similarity in the way they affect the convergence of SA: very low temperature/small cooling factor makes SA behave like a typical RHC; Very high temperature/big cooling factor significantly elongate the converging process (SA tends to spend more time exploring rather than exploiting). Their differences lie on the starting point and the divergence pattern. SAs with different cooling factor but same temperature are on the same starting line, but once the whistle blowed their curve grow far and far apart and only rejoin in the stable phase; SAs with different temperature but same cooling factor start differently (due to initial temperature difference) and go closer and eventually synchronized. We can see the winning SA model is actually the one most like RHC, which means short to no exploration for this specific task.

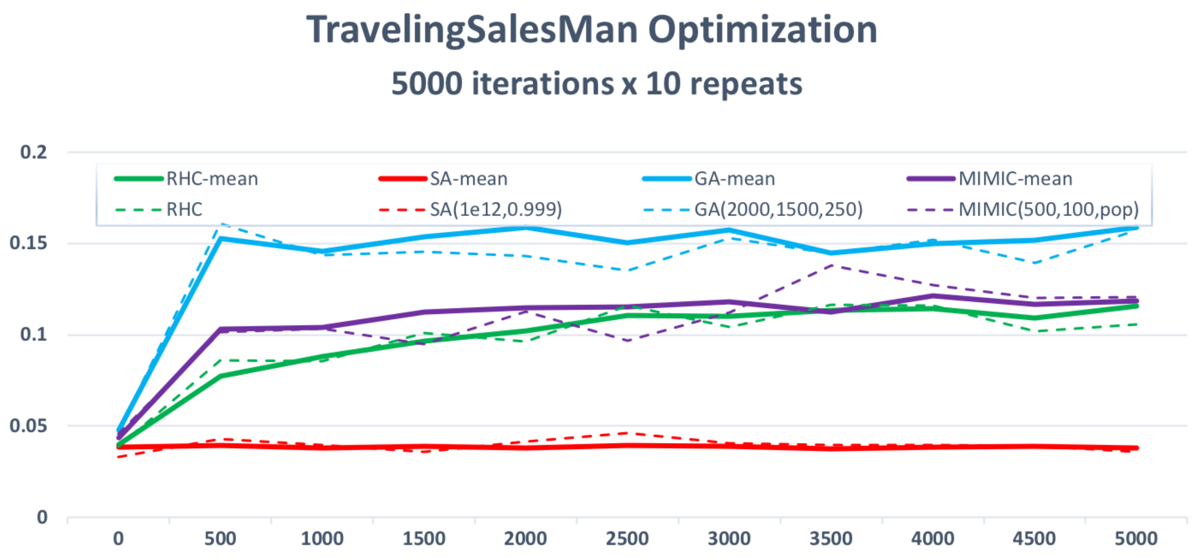
In Summary
Given either sum of squared error or accuracy as a metric, backpropagation is always the fastest converging and best performing method. While RHC leads among 3 optimizers, followed by SA similar with RHC, and tailed by GA. Time and computational cost aren't linear with the number of parameters. This experiment tells us that time and performance don't have to be in a painful trade-off relationship. Performance doesn't have to be purchased by paying high time and computational fees if we know the optimizing problem well and pick the most suitable algorithm.

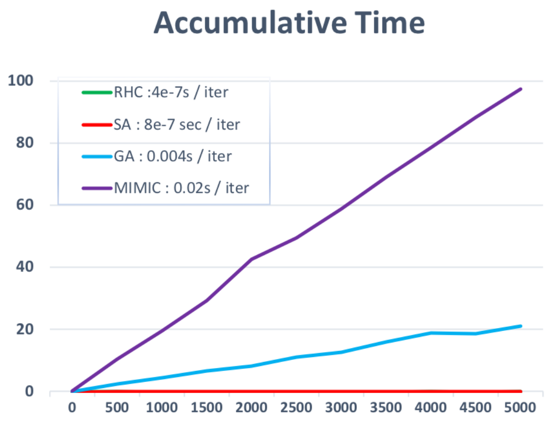
Three Optimization Problems to Elaborate Weakness and Strengthens of Chosen Algorithms

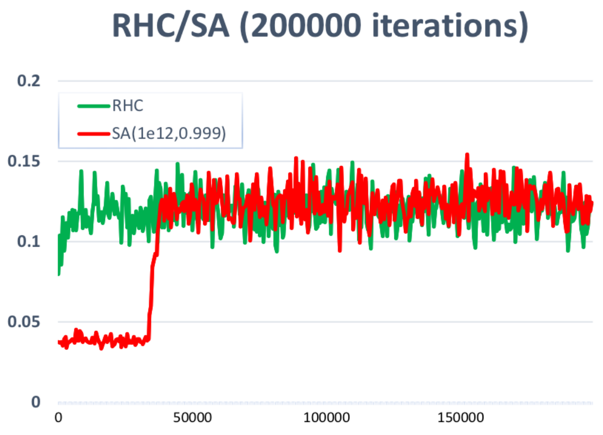
Green, red, sky-blue and dark purple are constantly used in this section to represent curves for RHC, SA, GA and MIMIC respectively.

Genetic Algorithm's Strength Problem: Traveling Salesman

Why it's interesting: Travelling Salesman Problem (TSP), is a classical optimizing problem intensively studied and often used as benchmark for optimization problems. It asks: "given a set of cities and distance between every pair of cities, return the shortest possible route that visits every city exactly once and returns to the starting point?" It is an NP-hard problem in combinatorial optimization and is applicable to many fields such as logistics, the manufacture of microchips, and even DNA sequencing and astronomy[4]. Using the inverse of total distances travelled as score of fitness, the best optimizer is able to find the maximum value possible with shortest possible time if it exists.







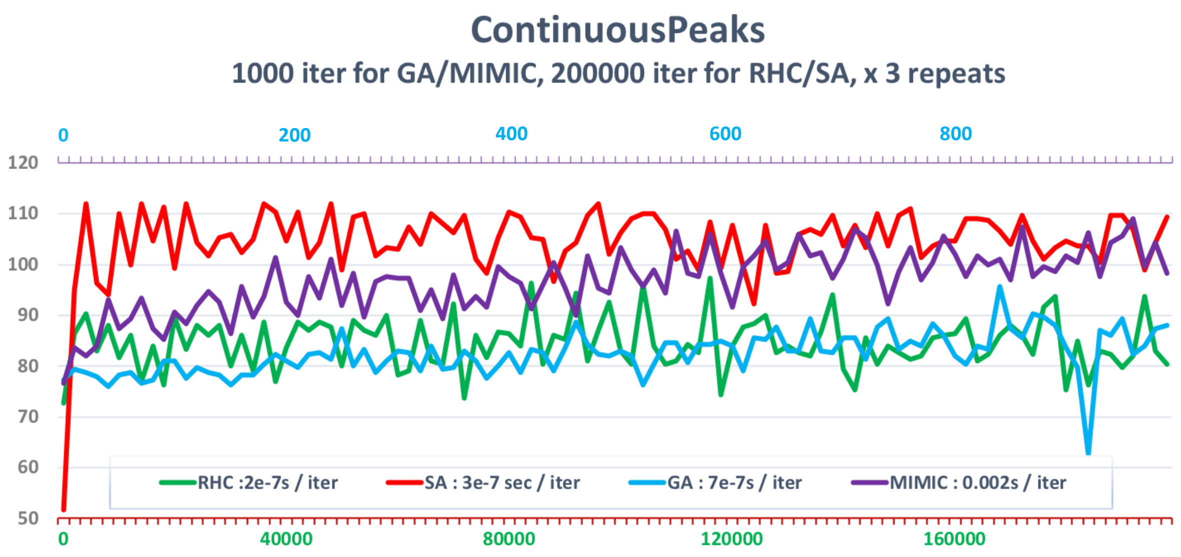
GA is the best performer with acceptable time cost
10 sampling points (sample every 500 iterations till 5000 iterations) are collected and this procedure was repeated 10 times. Genetic Annealing leads the group in finding the shortest route. MIMIC, the method inspired by GA and share with which most similarity here, ranks two. Simulated Annealing performs worst in the first 5000 iterations, not surprising if we link

in the bottom right graph: it converges and gets comparable performance with RHC only after 5000 iterations (Notice that the experiment got run only once and sampled much more frequently that's why the curves look noisy).

Both GA and MIMIC[5] took a lot of computation time (1e-3s and 1e-2s), compared to which the cost of RHC and SA are just neglectable (1e-7s). However, MIMIC took 10 folds more than GA but still lost, perhaps because true underlying dependency relationship is much more complicated than its assumption, while the pool of hypothesis maintained by GA gives more possibility in inter-variable relationship modeling.

Simulated Annealing's Strength Problem: Continuous Peaks

Why it's interesting: the continuous peak problem is derived from the original four-peaks[6] problem, the fitness function of which is: $FourPeaks(T, X) = MAX(head(1, X), tail(0, X)) + R(T, X)$ The major change of Continuous Peaks Problem is that the reward-worth sequences don't have to be on head or tail. Four-Peaks problem cares only about the heading/tailing 1/0 sequences, while in continuous peaks scenario, all continuous 1/0 sequences are potential to be rewarded. Not surprisingly, continuous peaks problem has more local optima, but four-peaks problem has less but steep `basins of attractions`[7].



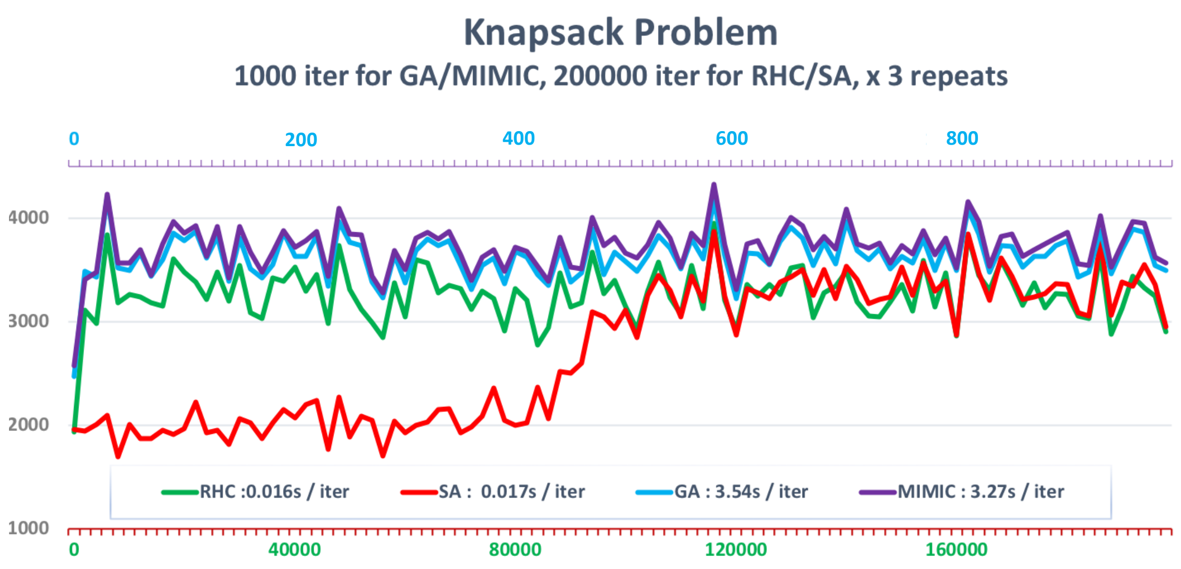
Two horizontal axes are used. The upper one belongs to GA/MIMIC as they only run 1/20th of iterations run by RHC/SA at the same time. Notice out of time concern, repeating number is reduced to 3 instead of 10 in ContinousPeaks and Knapsack Problem. But total sampling number has increased from 10 to 100 for the running experiment.

Simulated Annealing leads in both fitness scoring and time efficiency
As we can see, there are many more local optima, indicating a much different fitness landscape from four-peak problem. Simulated Annealing this time topped the team, but it was caught up quickly by rising MIMIC. However, if we consider the time consumption, which is always an important measure for optimizer, Simulated Annealing wined over MIMIC without questioning (1e-7s/iteration vs 1e-3s/iteration).

Compared to other two problems, continuous peak problem has a relative smaller hypothesis space: |H| = 2^60 = 1.1529215e+18. (|H| of Traveling salesman is 50! = 3.0414093e+64. |H| of knapsack problem is bigger in the way that algorithm has 160 items to pack from.) This is consistent with the observation that algorithms, especially computation demanding algorithms like GA and MIMIC, spent less time for each round and as well as for converging.

MIMIC's Strength Problem: Knapsack

Why it's interesting: The knapsack problem is a problem which is more than a century year old in combinatorial optimization field. Its name is derived from everyday problem that how a fixed-sized knapsack can fit the most valuable items. More precisely, it asks: "Given a set of items, each with a weight and a value, can you return the number of each item to include in a collection so that the total weight is within a given limit and the total value is as large as possible?" [8] This problem can also to extended to fields such as computer science, cryptography, applied mathematics, and daily fantasy sports. For abagail's Knapsack example, by default there are 40 species of items, 4 copies each, for the optimizer to choose from. Weight and volume of every species is generated by random process. The optimizer must maximize the total weight with the constraint that the total volume must not go beyond the set max volume of knapsack,3200.



MIMIC is the best performer with second highest time cost

Knapsack problem is the hardest (at least for optimizers) among the three problems. As the time cost for all algorithms is bigger than other two about 3 to 5 orders of magnitude. In the extreme case of Genetic Algorithm, it jumped from 1e-7s iteration cost level in Continouspeaks Problem, suddenly to 1s level, as big as 7 orders of magnitude. SA took much longer time (more than 10000 rounds) to converge with the same setting.

MIMIC proves itself in this challenging scenario. The returned fitness by MIMIC is always higher than others. However, there is unneglectable similarity in the curve shape of GA and MIMIC this time, which indicates a synchronized candidate evolving process for them generation to generation.

In Summary

So far discussion in this part of analysis has introduced a very important topic: combinatorial optimization[9]. It consists of finding an optimal object from a finite set of objects. Different from ANN, where continuous numbers are those to be optimized, here discrete finite values are the main focus. Classical benchmarks like traveling salesman problem, four-peaks problem (where continuous peaks problem is derived) and knapsack problem are put together to evaluate 3 optimization methods.

Difference of the 3 methods is mainly from the different approaches of how candidate solution is generated, or more specifically from how they deal with a worse solution. The idea that the fitness evaluation has to be exploited to avoid brute force searching the fact that a lower fitness solution may lie on the way to global optima has to be both handled in a proper way. Simulated Annealing balance the two by introducing a temperature parameter and took decision probabilistically[10]. Genetic Algorithm solve it by employing multiple searching agents and make them work parallelly and independently[11], while MIMIC by maintaining and refining an underlying dependency tree.

2 pairs of algorithms had caused attention with their similarity in last sections. The first pair is RHC and SA, and the second pair is GA and MIMIC. The first pair is always low in cost, excel in handling simple combinatorial optimization (continuous peaks) and continuous problem (both toy ANN and challenge ANN). Actually, RHC can be seen as an extreme case of SA mathematically. The second pair, GA and MIMIC, is similar in algorithmic aspect: maintaining a pool of decision maker and communicating information generation-wise, and in practice aspect: demanding in computation and excellent in complicated combinatorial optimization problems.

	|H|	Run time (in terations)	Fitness rank	Time cost per iter (in order of magnetude)
ContinuosPeaks	2^60 ~= 1e18	1000 for GA/MIMIC, 200000 for RHC/SA	SA > MIMIC > RHC > GA	RHC/SA/GA: -7, MIMIC: -3
TravelingSalesman	50! ~= 3e64	5000	GA > MIMIC > RHC > SA	RHC/SA: -7, GA/MIMIC: -3 ~ -2
Knapsack	Biggest	1000 for GA/MIMIC, 200000 for RHC/SA	MIMIC > GA > RHC > SA	RHC/SA: -2, GA/MIMIC: 0

In conclusion, RHC/SA are good at combinatorial optimization problem with small hypothesis space, as they can quickly exhaust the version space and come up with a fair answer when GA/MIMIC are still struggling with refining their decision structure[12]. However, computation investment of GA/MIMIC in early rounds would benefit them in long term for a relative difficult problem, which give them great potential for really challenging tasks. The exact comparison of RHC and SA, or GA and MIMIC are hyperparameter and problem dependent, which is hard to conclude from the limited experiments for now.

References:
[1] ABAGAIL – JAVA library for ML and AL
[3] sklearn.neuron_network.MLPClassifier
[2] How many parameters are there for ANN?
[4] Travelling salesman problem
[5] JS De Bonet et al, 1996, conference paper, MIMIC: Finding Optima by Estimating Probability Densities.
[6] Shumeet Baluja and R. Caruana, 1995, tech report, Removing the Genetics from the Standard Genetic Algorithm.
[7] Shumeet Baluja and Scott Davis,1997, Using Optimal Dependency-Trees for Combinatorial Optimization: Learning the Structure of the Search Space
[8] Knapsack Problem
[9] Combinatorial optimization
[10] S Kirkpatrick et al, 1993, science, Optimization by Simulated Annealing.
[11] Eiben, A. E. et al, 1994, PNAS, Genetic algorithms with multi-parent recombination.
[12] Genetic Algorithm versus Simulated Annealing