Sean Dai

CS 4641 - Assignment 2: Randomized Optimization

1. Introduction

For this assignment, the dataset of choice is a modified version of LETTER that was used in Assignment 1. This dataset consists of 20000 instances of randomly distorted capital letters from among 20 different fonts, obtained from the UCI Machine Learning Repository. However, instead of assigning a label (A-Z), each instance is labelled according to the following scheme:

$$Class(x) = \begin{cases} 1 & \text{if the letter is an "O"} \\ 0 & \text{otherwise} \end{cases}$$

for each instance x, which has 16 primitive numerical attributes. This change stems from the difficulty of multiclass classification in ABAGAIL. The modified dataset was then retested in Weka using 10-fold cross validation, and appropriate training and test results were logged. For this assignment, the following randomized optimization (RO) algorithms were run on the dataset to obtain optimal weights for the neural networks: randomized hill climbing, simulated hill climbing, and genetic algorithms. The objective is to compare the classification error of the neural net trained using the backpropagation algorithm and of that trained by RO. We also wish to analyze other statistics such as training time, testing time, and SSE during each time step.

Additionally, three separate optimization problems were also created to illustrate the strengths of the genetic algorithm, simulated annealing, and MIMIC. The three problems are the four peaks problem, the continuous peak problem, and the knapsack problem. Their implementations are located in the project directory and can be run directly from the files *FourPeaksProblem.java*, *ContinuousPeaksProblem.java*, and *KnapsackProblem.java*. Running each of these files produces a separate text file containing statistics tested over 100 different trial iterations. Neural network training and classification were performed using ABAGAIL. The three RO problems were also obtained and modified from ABAGAIL with some additional helper files from Github. Graph visualization was done using the Python libraries, *numpy*, *matplotlib*, and *pylab*.

2. Motivation: Interesting Dataset & Optimization Problems

The results obtained from running the RO algorithms on the LETTER dataset offer tremendous practical applications. For example, a trained neural network that can accurately classify letters may be of use in optical character recognition (OCR) of handwritten text. This is beneficial in the translation of millions of works of academic literature, ranging from the humanities to the sciences, into machine-readable formats that scholars may later use for their research.

One observation noticed by reducing the problem domain from classifying 26 separate letters to a binary classification ("O" or not "O") and training with backpropagation was the improvement in accuracy by approximately 13% (from 86.84% to 99.76%)! This reduction is due in part to the high presence of letters other than "O" in the dataset of 20000 instances, to the extent that the learner can easily discriminate to which label each instance belongs. After the running the RO algorithms, classification accuracy converged toward 96.235% (19247 instances / 20000) after cross-validation but did not improve beyond that obtained with backpropagation. It is interesting to note that while the RO algorithms only misclassified 753 instances, the dataset contained exactly 753 instances that were originally labelled "O". Hence, by classifying each example as 0 (not letter "O"), the trained neural network is able to minimize its final error rate. Like all optimization methods, the RO algorithms suffer from obstacles such as being stuck in local minima and plateaus in continuous search spaces. In this case, we can consider labelling each instance as 0 as a local minimum. But as we shall soon see, the rate at which the RO algorithms converges toward the optimal solution is what truly contributes to the mystery of the neural network problem.

The first optimization problem (*Four Peaks)* illustrates the main advantage of the genetic algorithm (GA): the robustness of natural selection as an optimization technique. The problem takes as input a bit string of length *n* and assigns a value based on the number of leading 1s and trailing 0s and a reward function. Despite numerical results showing that simulated annealing and MIMIC perform better over 100 iterations, a closer inspection of the graphical data suggest that this numerical interpretation may not be entirely true.

The second optimization problem (*Continuous Peaks)* is a variant of *Four Peaks* that counts contiguous blocks of 0s and 1s in the *n*-bit string. Like *Four Peaks*, the problem gives a reward equal to *n* if the number of contiguous 0s and 1s is greater than some integer *t*. The *Continuous Peaks* problem demonstrates mainly the strength of simulated annealing and, albeit to a lesser degree, the MIMIC algorithm. What is interesting to note about this problem is that despite performing relatively well on the *Four Peaks* problem, the GA obtained a much lower average fitness value in comparison to simulated annealing and MIMIC. Furthermore, the GA performed slightly better than the worst RO algorithm, randomized hill climbing, in terms of average fitness values (87.17 vs 83.42).

The third optimization problem is the classic knapsack problem. Given a knapsack of fixed volume V and a finite set of items with each item occupying a certain volume and having a value, fill the knapsack with items without exceeding the knapsack's volume V such that the total value is maximized. Since the test in ABAGAIL specifies that there are a fixed number of copies of each item, this problem is also known as the bounded knapsack problem. This problem is of practical importance in economics and finance. For example, maximizing the total value of a knapsack is analogous to selecting the best investments in a collection of investment portfolios, where the value is the return on investment, and the constraint is a fixed annual budget. This optimization problem illustrates MIMIC's strength in its ability to capture the underlying structure of the problem. Out of 100 trial runs, MIMIC outperforms randomized hill climbing (RHC), simulated annealing (SA), and genetic algorithm (GA) in terms of having the best fitness rank for each trial run and the greatest average fitness value.

<u>3. Retested Neural Network on Modified LETTER Dataset</u>

The modified LETTER dataset consisting of 20000 instances was retested in Weka by varying the number of training iterations and the training size. The backpropagation algorithm was used to assign weights to the neural net. Only one hidden layer consisting of 16 nodes was used for each run. The learning rate was fixed at 0.3, momentum = 0.2, and one node in the output layer.

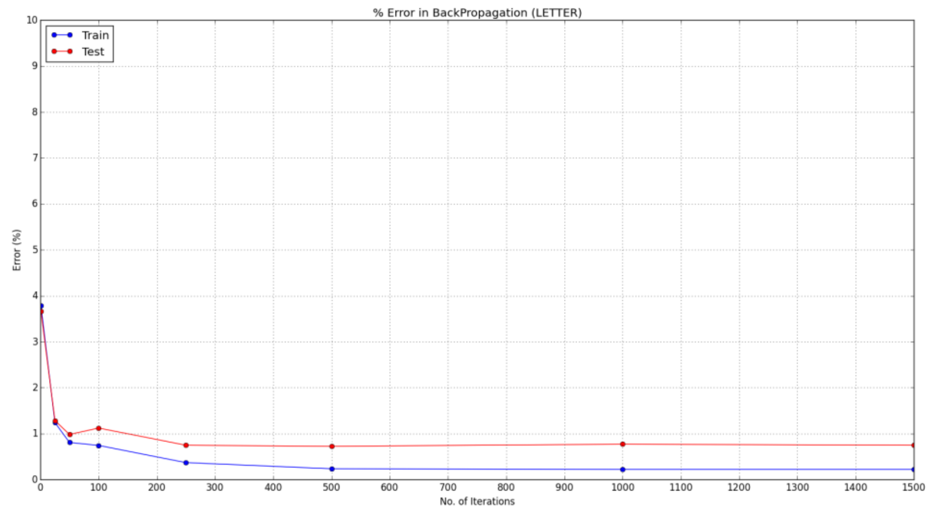


Fig. 1. This graph shows the training and test error (%) as the number of training iterations increases. Hidden layers = 1, hidden layer nodes = 16, learning rate = 0.3, momentum = 0.2, training size = 15000, testing size = 5000, classification labels = {0,1}

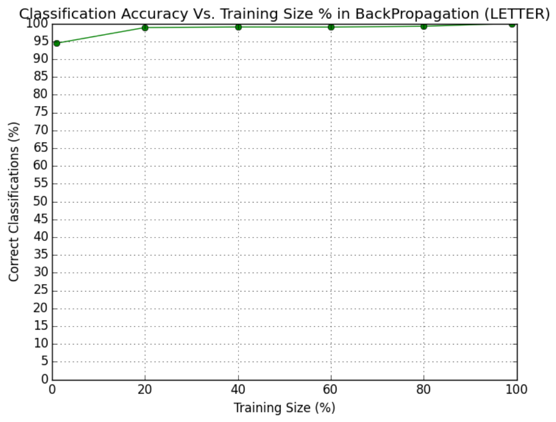


Fig. 2. This graph displays the effect of training size on classification accuracy for the modified LETTER dataset. Hidden layers = 1, hidden layer nodes = 16, learning rate = 0.3, momentum = 0.2, number of iterations = 500, classification labels = {0,1}.

As discussed in the **Motivation** section, one reason for the low test rate (0.7712% for 1000 iterations) is the small number of instances that were originally labelled "O" (exactly 753 out of 20000). Here is how the confusion matrix looks like after 1000 iterations of backpropagation:

```
=== Confusion Matrix ===
   a    b   <-- classified as
4110 12  |    a = 0
  21  136 |    b = 1
```

Now that we have the results for the modified dataset, we can use this neural network as a baseline with which to compare the neural nets trained with the RO algorithms. The parameters for this baseline model are as follows: number of inputs = 16, hidden layers = 1, hidden layer nodes = 16, number of iterations = 500 (this value changes in some cases when it is the parameter being modified), output layer nodes = 1, classification labels = {0, 1}.

4. Neural Networks Trained with Randomized Optimization Algorithms

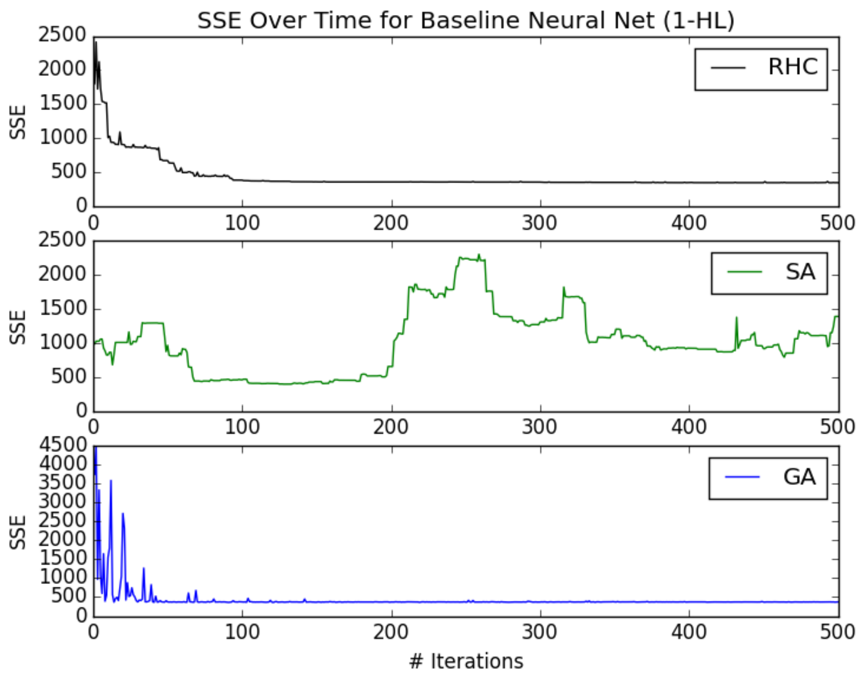


Fig. 3. This graph displays the sum of squared errors between the predicted result and the actual result using our baseline neural net model over 500 iterations. The results are displayed after running randomized hill climbing, simulated annealing, and the genetic algorithm to train the neural net. Detailed numerical results can be found in "results-1-HL-iterations-500-1426221691839.txt".

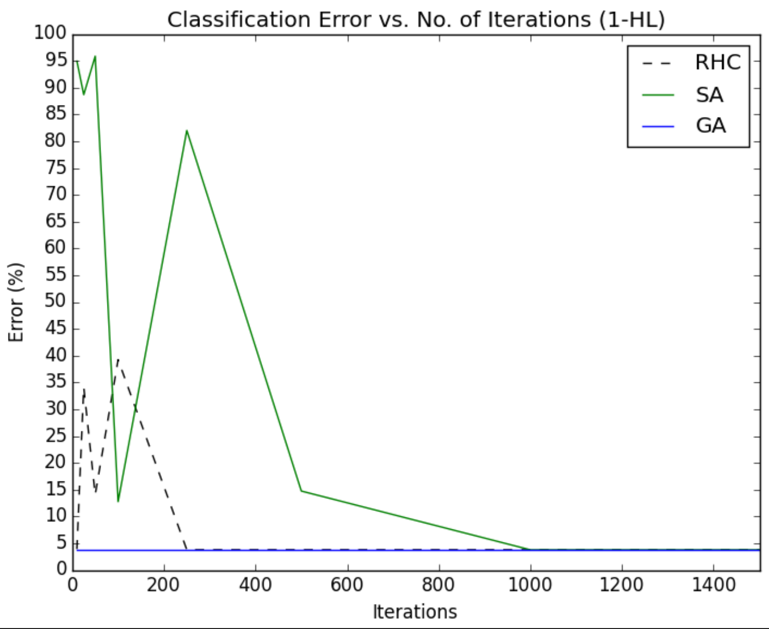


Fig. 4. This graph displays the classification error of a 1 hidden layer-neural net trained over the following iterations = {10, 25, 50, 100, 250, 500, 1000, 1500}. Fluctuation in RHC and SA results may be due to randomization of starting values.

From Fig. 3, several trends are readily noticeable. The SSE plot of RHC appears to decrease almost monotonically as the number of training iterations increases. The graph of SA approaches a minimum at iterations 70-200 but begins to increase sharply after that point. The SSE values of the GA fluctuate wildly in the first 40-50 iterations but quickly achieve the minimum value of 357.114 and remains constant for the remainder of the run.

From Fig. 4, we notice that the classification error after CV for RHC and SA fluctuate up and down over the number of iterations whereas GA remains relatively constant at around 3.765%. This fluctuation may be due to randomization of starting values and the rate at which the RO algorithm converges to its optimal SSE value in a limited amount of time. Other factors for these RO trends are discussed individually in subsequent sections.

Algorithm	Iterations	Classification Error w/ 10-CV (%)	Train Wall-Clock Time (s)	Test Time (s)
	10	3.669	0.12	<.1
BackProp	25	1.285	2.79	<.1
	50	0.982	5.61	<.1
	100	1.122	11.13	<.1
	250	0.7478	27.91	<.1
	500	0.7245	55.17	<.1
	1000	0.7712	112.61	<.1
	1500	0.7478	172.09	<.1
RHC	10	3.900	1.866	0.160
	25	34.09	2.704	0.091
	50	14.05	5.211	0.091
	100	39.23	10.401	0.092
	250	3.765	26.221	0.092
	500	3.765	51.860	0.092
	1000	3.765	105.525	0.091
	1500	3.765	156.354	0.093
SA	10	94.93	0.141	1.602
	25	88.68	2.562	0.089
	50	95.86	5.346	0.090
	100	12.75	10.514	0.091
	250	81.99	26.406	0.093
	500	14.73	52.135	0.092
	1000	3.765	104.466	0.096
	1500	3.765	155.313	0.090

GA	10	3.765	55.593	0.119
	25	3.765	90.903	0.090
	50	3.765	183.124	0.098
	100	3.765	373.256	0.093
	250	3.765	906.565	0.088
	500	3.765	1843.210	0.093
	1000	3.765	3575.770	0.096
	1500	3.765	5494.317	0.106

Fig. 5. This table shows the test results of training the baseline neural net model using backpropagation, RHC, SA, and GA. The parameters of the baseline neural net are discussed in the section **3. Retested Neural Network on Modified LETTER Dataset**.

In ABAGAIL, classification is done by evaluating the output node value using a majority activation function. For example, if the output = 0.6551, the instance is labelled ‘1’ because 0.6551 > 0.50.

4.1. Randomized Hill Climbing

RHC is a local optimization algorithm that is well suited for discrete-value problem domains. The algorithm works by randomly selecting a solution from its neighbors and moves to the neighbor only if it results in an improvement. In ABAGAIL, this improvement measure is evaluated with the fitness function *value(Instance d)* that is different for each RO algorithm (RHC, SA, GA). From the graph shown in Fig. 3, the plot of RHC starts at a high SSE value of 1802.480 and decreases almost monotonically. The algorithm infrequently obtains a worse result, which is due to the randomness of the procedure.

One of the advantages of RHC is its fast convergence (< 100 iterations) toward the lowest SSE value and its small space complexity. The algorithm keeps track of only the current state in the search for local optima. However, even with this randomness component, RHC may not always converge to its optimal value if the initial SSE is too high and not enough iterations are performed. This observation is evidenced in Fig. 4 and 5 for iterations = 100. While the previous run using 50 iterations produced a classification error of 14.05%, the error obtained with 100 iterations increased by 25% to a shocking 39.23%. Upon closer inspection of the results (“results-1-HL-iterations-100-1426220322967.txt”), RHC started with an initial SSE of 7053.798 and was not given adequate time to reach its optimal value, only attaining an SSE of 3076.401 after 100 iterations. This final SSE value for the trained neural net is not sufficient to predict accurately the classification of all 20000 instances.

While RHC fares better than steepest ascent hill-climbing, which selects only the best neighbor, it is still prone to local optima and plateaus as discussed previously. One reason for RHC’s excellent performance in terms of wall clock training time and its relatively quick convergence to its optimal value may be due to the presence of few local minima in our problem (eg. classifying all instances as ‘0’, or not letter “O”) and the low cost of generating a neighboring solution. Furthermore, it is interesting to note that gradient descent in the backpropagation algorithm is essentially a hill-climbing problem, where weights are updated in proportion to the partial derivative of the squared error with respect to each weight. However, as we have seen, local minima still remain a problem.

A method to reduce this occurrence is by using random-restarts. Combined with RHC, the random-restarts technique will repeatedly generate random neural net weights until we reach the goal state. As the number of restarts $\rightarrow \infty$ in the presence of a finite number of local minima, the probability of obtaining a globally optimal solution $\rightarrow 1$. However, this does not imply we have just discovered the best optimization algorithm that exists. As always, overfitting of the neural net on the training data will still be of concern to the experimenter regardless of how well random-restart RHC performs on the dataset.

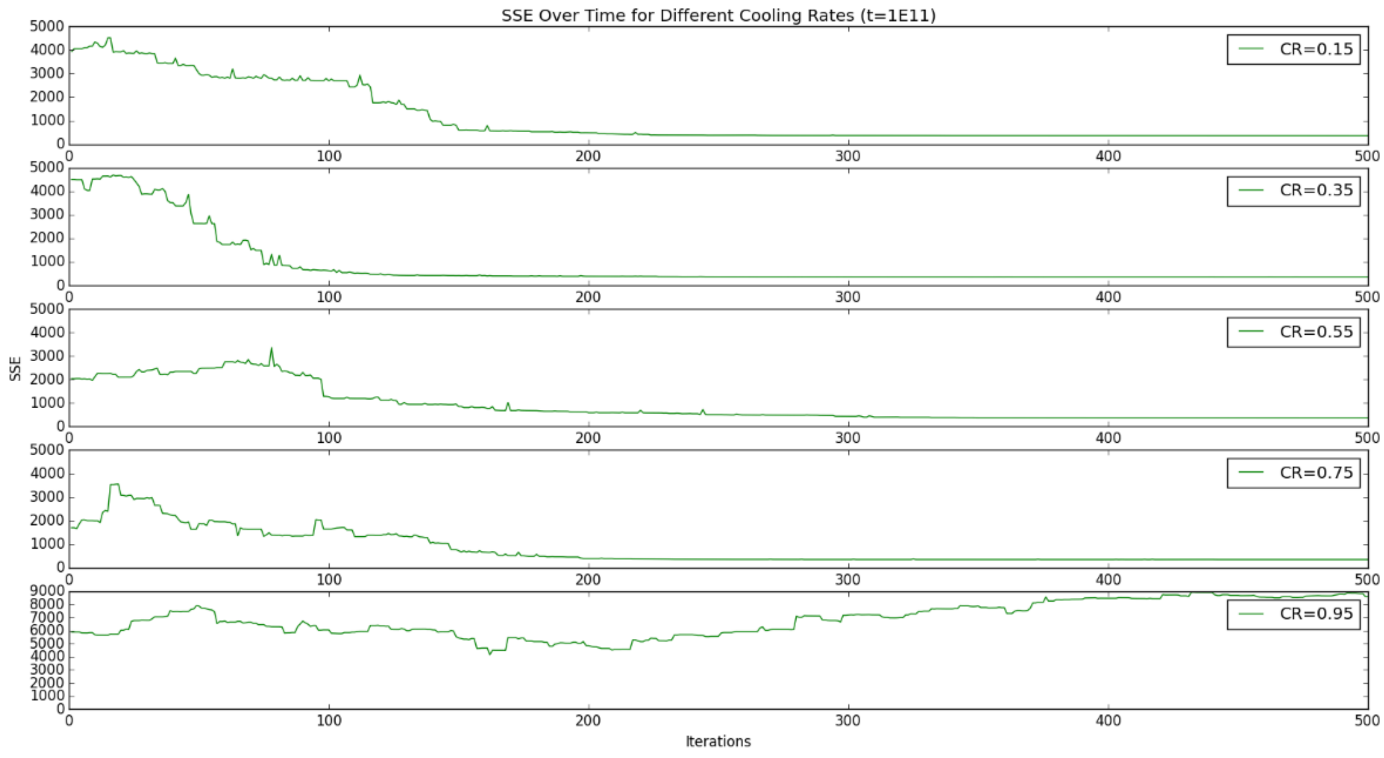
4.2. Simulated Annealing

SA combines some elements of RHC. Instead of selecting the best among random neighbors, SA selects a move at random. If the fitness is improved, the solution is always accepted. Otherwise, the move is picked with probability according to the Boltzmann distribution:

$$P(nextMove|no\ improvement) = e^{\Delta E/t}$$

where ΔE = nextMove.value – currentState.value and t = temperature. Hence, this probability is always between 0 and 1 when next move is suboptimal. After each iteration, the temperature is decreased by a constant factor called the cooling schedule, defined as *coolingRate* in the Java files. Thus, the probability of accepting "bad" moves is high when temperature is also high and becomes unlikely as t decreases.

In Fig. 3, the default parameters for training the neural net with SA are **t = 1E11** (supernova hot!) and **coolingRate = 0.95**. The SSE plot of SA approaches a minimum at iterations 70-200 but begins to increase sharply after that point and "wanders" around aimlessly. It is unable to attain its previous minimum, and as a result, the classification error rate suffers (14.73% from Fig. 4). This suggests that 500 iterations is an inadequate length of time to achieve an optimal result for SA. To improve this result, we can either decrease *coolingRate* so that SA converges to a local minimum at a faster rate, or we can increase the number of iterations so that SA has adequate time to explore the search space. First, let us observe what happens when we vary the cooling parameter.



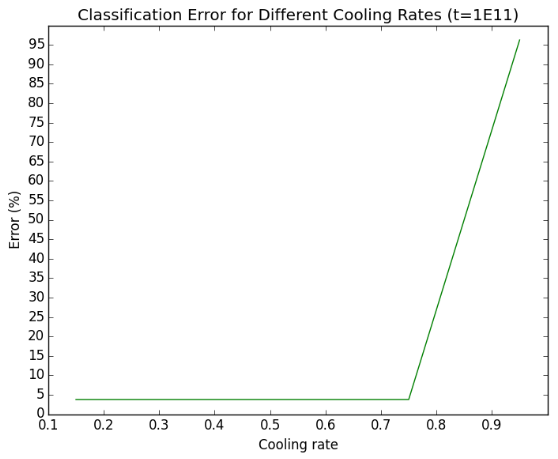


Fig. 6.
a) This graph shows the SSE values during training of our baseline neural net over 500 iterations. The subplots were obtained by varying the coolingRate = {0.15, 0.35, 0.55, 0.75, 0.95}.

b) The graph displays the classification error (%) for the corresponding cooling rates chosen in a).

The graphs (Fig. 6) suggests that our suggestions for improvement are valid. When cooling rate = 0.95, SA selects many suboptimal moves that do not improve SSE. But when cooling rate is decreased to 0.15, SA achieves its minimum value by iteration 150. Furthermore, it does not escape out of this attraction basin for the remainder of the run (Fig 6a). The graphs above suggest that as cooling rate is gradually lowered, SA converges exponentially faster toward its optimal value.

Now suppose coolingRate is fixed back to 0.95, but this time the number of iterations for SA increases to 1500. All three RO algorithms were re-run with 1500 iterations, and the following graph was produced:

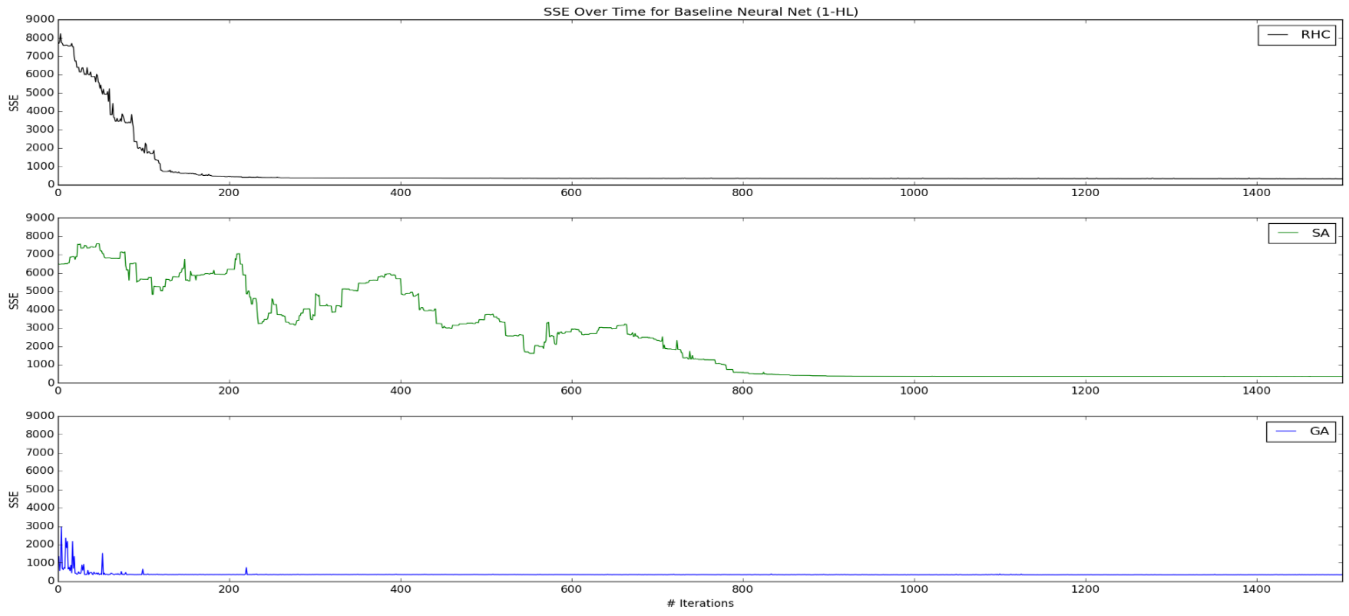


Fig. 7. This graph displays the results of running RHC, SA, and GA over 1500 iterations. The SSE values represent the squared difference between predicted result and the actual result using our baseline neural network model.

It looks like the evidence supports our prediction again (Fig. 7)! By iteration 800, SA stops randomly exploring the search space and converges toward its optimal value. The graphs of RHC and GA, however, are almost identical to what was displayed in Fig. 3.

4.3 Genetic Algorithm

GAs apply a fitness function to the population. The two most fit individuals are allowed to mate, followed by the other pairs in order of decreasing fitness. During the crossover phase, genes (encoded as bit strings) from both parents are swapped to create the offspring's gene string. At the end of each iteration, the offspring are subject to mutation with small probability. This process of mating and mutating continues until termination or until the fittest individual is selected.

In Fig. 3, the default parameters for training the neural net with GA are **population = 200**, **toMate = 100**, **toMutate = 10**, **crossoverOperation = UniformCrossOver()**, **mutate = ContinuousAddOneMutation()**. The SSE values of the GA fluctuate wildly in the first 40-50 iterations but flattens out over time. One explanation for this behavior is that depending on the initial fitness and the percentage of the population mating and experiencing mutations, a population with subpar overall fitness reproduce and propagate "bad genes" to their offspring. However as time increases, the robustness of natural selection becomes evident: individuals with low fitness are culled from the reproducing population, and the overall fitness of the population is improved. One downside of training neural nets with GAs is the space requirement to store each individual in the population. In addition to sorting each individual by fitness and performing numerous crossover and mutation operations, the cost of the algorithm increases significantly as the number of attributes increases. The curse of dimensionality is evidenced by the long training times of the GA, shown in Fig. 5. For instance, the GA took 5494 seconds, or 1.5 hours, to train a neural net for 1500 iterations! One way to alleviate this problem is by reducing the number of total iterations to the point where training time is not a large limiting factor. As illustrated in Fig. 3 and 7, GAs converge to its optimal value of 363.260 before iteration 100. As a treat, let's look at what happens when we vary the number of individuals that experience mutations (*toMutate)*.

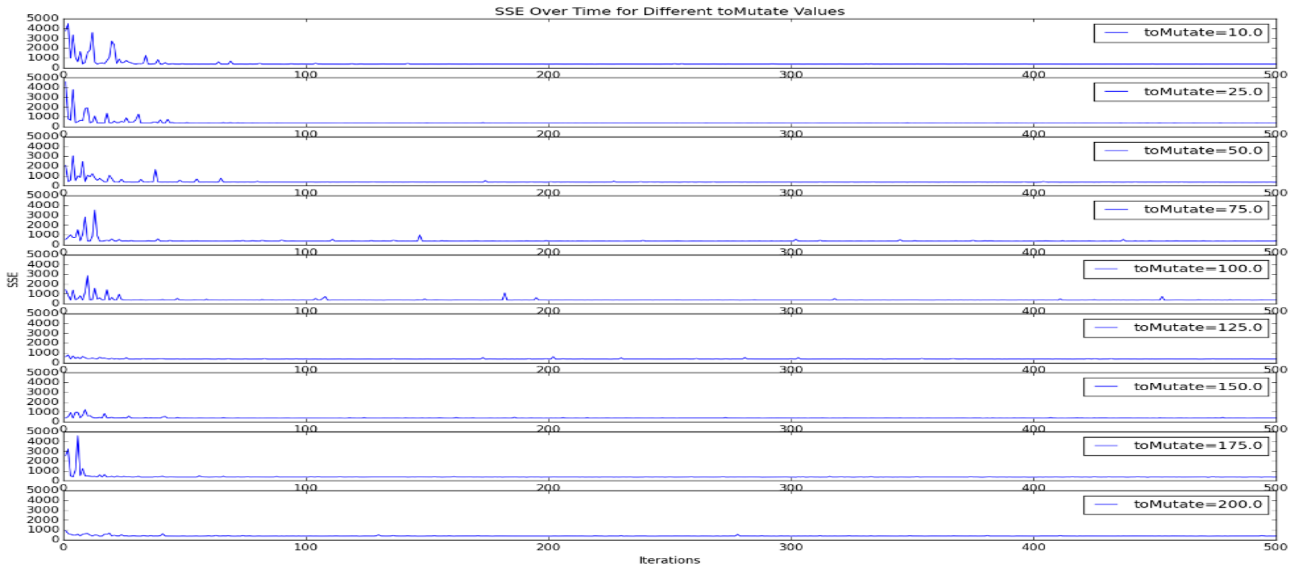


Fig. 8. This graph displays the results of training our baseline neural net with GA over 500 iterations. The population size consists of 200 individuals and mating rate = 50% (or 100 indiv.) of the population for each iteration. The number of individuals experiencing point mutations in their gene strings is a variable.

This result is interesting because given fixed population size and number of individuals mating, populations that experience few mutations (toMutate = 10, 25, 50, 75, 100) require longer iterations to achieve optimal values. Additionally, overall SSE values fluctuate inconsistently between 500-4000 in early iterations. This notion can be understood intuitively from a biological perspective. Some mutations may be inherently beneficial. Without them, populations with poor overall fitness but similar genetic makeup will further propagate these unfavorable genetics to future generations (think of rural isolated communities like the Amish).

5. Three Optimization Problems

5.1 Four Peaks Problem

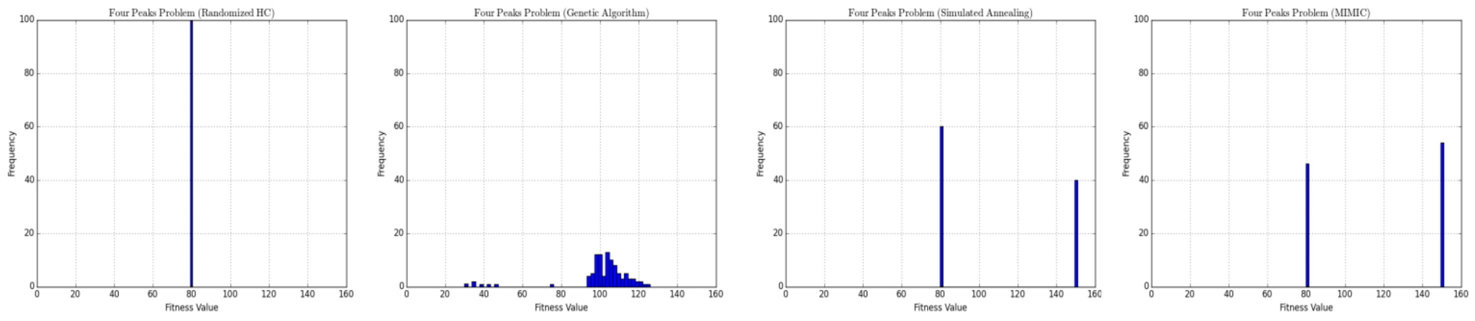


Fig. 9. The histograms represent the frequency of each fitness value over 100 separate trial runs for the *Four Peaks* problem. Parameters: *N* (length of the bit string) = 80 and *T* = 8. The optimal fitness value is then max(71,9) + 80 = 151.

<u>Four Peaks Problem</u>
(100 trials):
Average time (ms):
RHC: 53.66
SA: 79.81
GA: 90.1
MIMIC: 7487.16
Average fitness values:
RHC: 80.0
SA: 108.4
GA: 100.99
MIMIC: 118.34

The *Four Peaks* problem is defined as:

$$f(\vec{X},T) = \max\left[tail(0,\vec{X}), head(1,\vec{X})\right] + R(\vec{X},T)$$ [Isbell 2].

where

$$\begin{aligned} tail(b,\vec{X}) &= \text{number of trailing } b\text{'s in } \vec{X} \\ head(b,\vec{X}) &= \text{number of leading } b\text{'s in } \vec{X} \end{aligned}$$

$$R(\vec{X},T) = \begin{cases} N & \text{if } tail(0,\vec{X}) > T \text{ and } head(1,\vec{X}) > T \\ 0 & \text{otherwise} \end{cases}$$

The optimal solution is determined by selecting a string with N-T-1 trailing 0s and T+1 leading 1s or a string with T+1 trailing 0s and N-T-1 leading 1s. Thus the maximum value of f(X, T) = 2N-T-1. From the histograms (Fig. 9), RHC selects only the locally optimal solution by generating a string of all 0s or all 1s. Both SA and MIMIC select either the locally optimal solution with fitness value (80) or the globally optimal solution (151) with probability ~ 50%. What is interesting about this problem is that GA selects the solution in a manner entirely different from that of RHC, SA, and MIMIC. The single crossover operation used by GA is beneficial in finding an approximate solution to the *Four Peaks* problem by combining the ends of parent strings. A majority of the fitness values obtained by GA roughly fall in between the locally optimal and globally optimal solutions (80 and 151), suggesting that the notion of natural selection is robust in the context of certain problems but is still prone to randomness (note the small scatter of fitness values between 30-50).

<u>5.2 Continuous Peaks Problem</u>

<u>Continuous Peaks Problem</u>
(100 trials):
Average time (ms):
RHC: 67.71
SA: 99.21
GA: 123.47
MIMIC: 3697.82
Average fitness values:
RHC: 83.42
SA: 104.27
GA: 87.17
MIMIC: 101.16

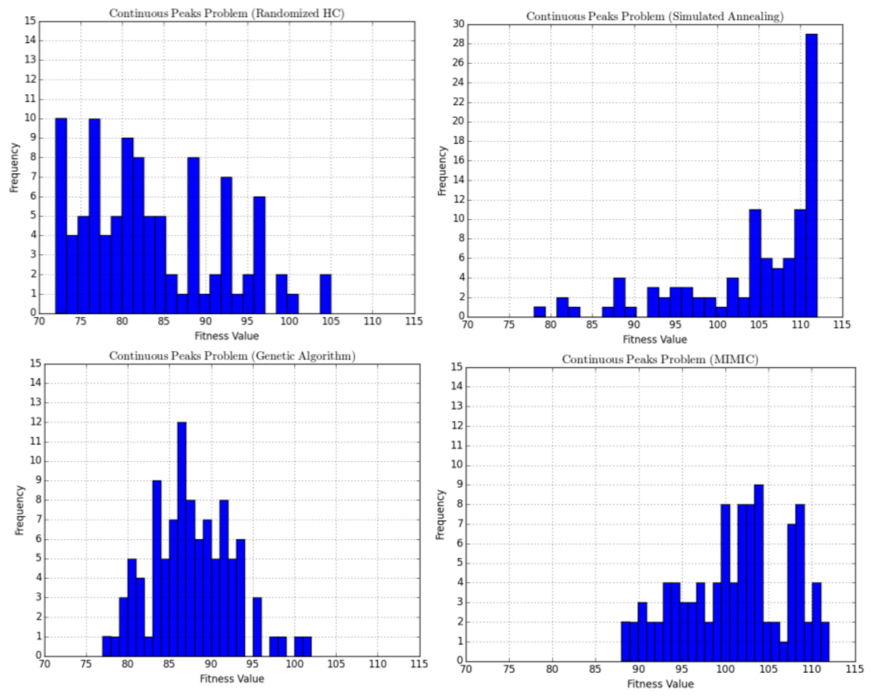


Fig. 10. The histograms represent the frequency of each fitness value over 100 separate trial runs of the *Continuous Peaks* problem. Parameters: *N* = 60 and *T* = 6. The optimal fitness value is thus max(53,7) + 60 = 113.

The *Continuous Peaks* problem is a variant of *Four Peaks* that counts contiguous blocks of 0s and 1s in the *n*-bit string. A reward of *N* is still given for having > T blocks of 0s and > T blocks of 1s. From Fig. 10, we observe that SA is the clear “winner” since it obtains the globally optimal value of 113 in 29 out of 100 separate trials. The frequency distribution of SA is skewed to the left, indicating that the median value is to the left of the mean fitness value, 104.27. MIMIC also performs relatively well on the problem; RHC is still the worst performing optimization algorithm in terms of the fitness values it obtains for each trial run. What is interesting to note is that while GA achieved robustness in *Four Peaks*, it produces subpar results on *Continuous Peaks*. This observation can be explained by the nature of the single point crossover operation. Splitting the bit strings at random points and then recombining them to form offspring breaks contiguous blocks of 0s and 1s. Hence, the results of GA can be expected because the underlying structure of the problem is violated.

5.3 Knapsack Problem

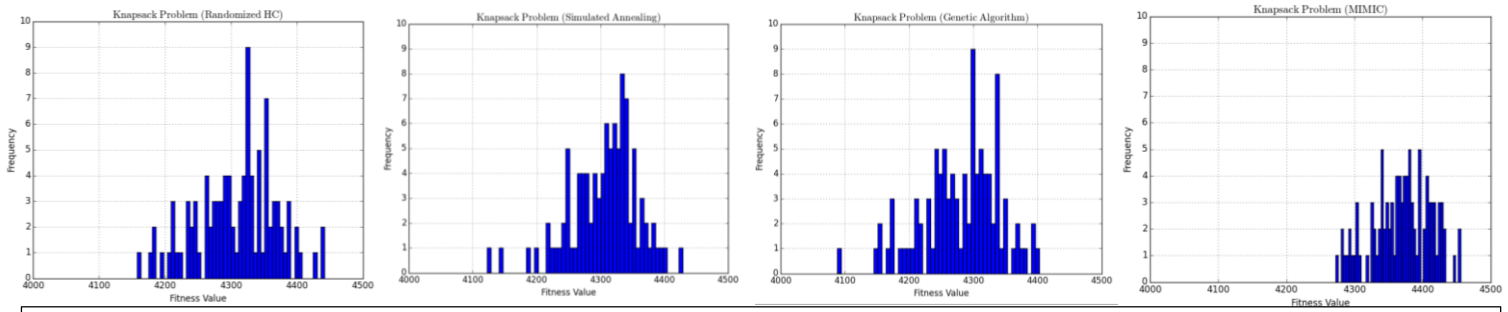


Fig. 10. The histograms represent the frequency of each fitness value over 100 separate trial runs of the *Knapsack* problem. Parameters: *NUM ITEMS = 40*, *COPIES EACH = 4*, *MAX WEIGHT = 50*, *MAX VOLUME = 50*, *KNAPSACK VOLUME = 4000*.

Upon inspection of the graphs in Fig. 10, all four distributions appear to be approximately normal. We note that MIMIC's center is shifted to the right by a value of ~70 in relation to RHC, SA, and GA. It is evident that MIMIC outperforms the other RO algorithms, but this comes at the cost of space and wall-clock time to obtain an optimal value. In fact, for all three problems, MIMIC frequently took 60-80x longer to run than the fastest algorithm, RHC. The O(mn) time complexity arises from the construction of MSTs, sorting and storing each sample, and generating new samples for each iteration of MIMIC, where m = number of attributes in the sample and n = number of samples.

6. Conclusion

The metrics used to quantify the performance of the randomized optimization algorithms were classification error rate (%) after 10-fold CV, SSE values plotted as a graph over the number of iterations, and wall-clock time to train the neural net. The RO algorithms RHC, SA, and GA performed slightly worse on the training data in comparison to backpropagation, which obtained a final classification error of 0.7712% over the 3.765% error from all three RO algorithms. Backpropagation works backwards through the neural net and adjusts the weights in proportion to the derivative of squared error. Whereas the RO algorithms only know that the solution may not be optimal, so they adjust the weights randomly. Clearly, the latter technique is more error prone, so one may expect the classification accuracy of neural nets trained with RO algorithms to be lower than that of backpropagation.

The *Four Peaks* problem illustrates the robustness of the natural selection idea in GAs. *Continuous Peaks* demonstrates that simulated annealing performs well in the presence of many local and global optima. The notion of cutting and pasting bit strings (GA) that was helpful in *Four Peaks* was not of much use in this problem because the breaking contiguous blocks of 0s and 1s harms the overall fitness of the bit string. The MIMIC algorithm outperformed all other RO algorithms on the *Knapsack* problem because it was able to capture the underlying structure of the problem. This result is partially offset by the time and space complexity required to construct the optimal solution.

7. Sources Consulted/Software used

1. Mitchell, Tom. Machine Learning.
2. Russell & Norvig. Artificial Intelligence: A Modern Approach. 3ed. (Ch. 4)
3. ABAGAIL to run the algorithms. Python libraries, *matplotlib*, *numpy*, and *pylab* for data processing and producing graphs.
4. Isbell, Charles Jr. "Randomized Local Search as Successive Estimation of Probability Densities."
5. Code to run the 3 problems and some wrapper classes: https://github.com/tomelm/supervised-learning