Anish Moorthy

Project Two: Randomized Optimization

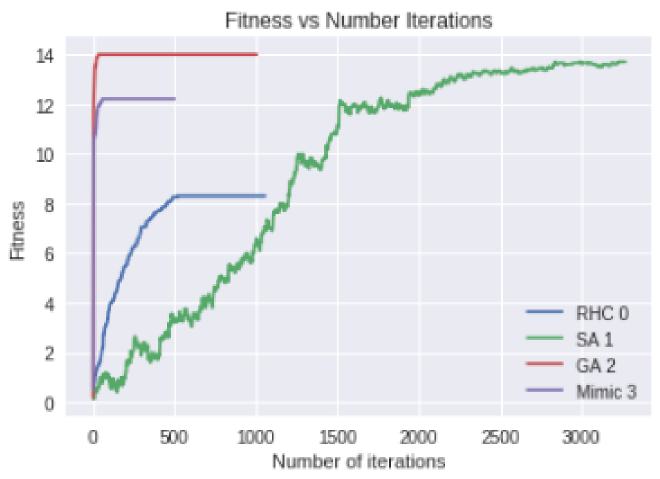
Four Peaks

For my first randomized problem, I chose a variation of the classic Four Peaks problem: the main feature of this problem is that the fitness function exhibits multiple local optima (four, in fact :) over the parameter space: specifically,

fitness(s) = max(# trailing 0s, # leading 1s) + (length(s) if # trailing/leading 0s and 1s > .1 * length(s))

The ability of an algorithm to find the global maximum here provides a measure for how effectively it balances exploration vs exploitation in what is essentially a toy setting.

The length of the state string here is eight. To make things slightly more interesting, I allowed the values of each "bit" to take on eight values (0-7), even though the fitness function essentially ignores everything but 0 or 1: this exponentially expands the state space over allowing just two values, and makes the problem more challenging since many values for bits are equivalent, so the algorithm may have to explore larger neighborhoods / follow paths more deeply to observe changes in fitness



To the left is a fitness-vs-iteration curve for each algorithm. Each curve is an average of twenty runs of each algorithm Maximum iterations for all algorithms was 5000, and all early terminations is triggered by failure to find "better" next points (definition dependent on algo) a thousand times in a row.

Optimal Hyperparameters:

RHC: No restarts, neighborhood of 1000

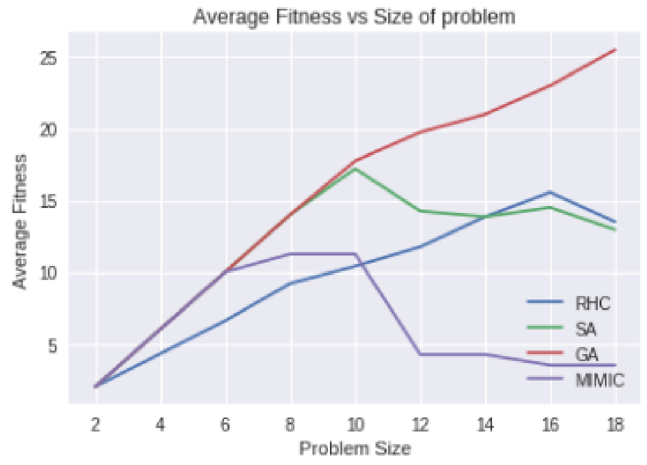
SA: Exponential decay from Temp from 1 to .001 with rate =.005

GA: Population of 400, Mutation prob=.1

MIMIC: Population of 2000, keep top 40%

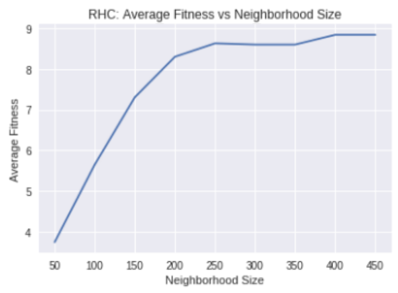
Genetic Algorithms are the top performer in both maximum fitness and number of iterations required to reach it. I believe that the reason is the breeding mechanism used, which is simply selecting the first n (random for each breed) bits of tone parent and the remaining bits of the other. This mechanism allows for large/discontinuous jumps within the state space, which allows for a more complete exploration of the state space and leads to a better estimate of the maximum. MIMIC also performs well since the problem has a relatively clear/exploitable structure: consecutive variables have high mutual information (this is natural from the fitness definition): however it is consistently unable to find the true optima. RHC quickly gets stuck in local minima (as expected). The slow decay chosen for simulated annealing means that it takes a while to converge, but has the key advantage of converging very near the optimum.

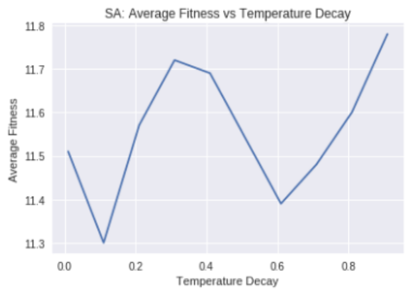
Note: Here, as elsewhere, I opted to use RHC without random restarts, as simply running the algorithm multiple times provides exactly the same effect

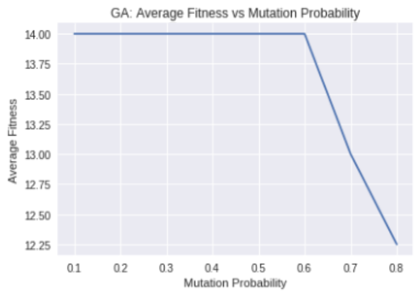


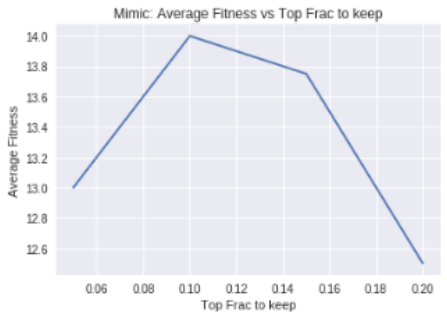
I also tested the effect of state length upon performance of each algorithm. Note that the possible values of fitness change given the state length, so this graph is more useful for *relative* ordering of algorithms. Genetic algorithms continue to be the clear winner for each state size (except smaller very easy sizes), indicating they are uniquely suited for optimizing the given fitness function & state space.

Interestingly, the quality of mimic degrades heavily with problem size. I believe this is because state size increases while population size remains constant at 2000: the same number of samples simply tells you less of the overall distribution, and so irrelevant bit values come to dominate the mutual-info calculation. This leads MIMIC to distribution updates which are fit to noise, breaking a key assumption of the algorithm and naturally causing performance to collapse. The convergence of SA and RHC performance is probably due to a similar effect: SA explores less and less of the state space as size increases (due to temperature decay not slowing with size), and becomes a glorified version of RHC. Appropriately scaling the MIMIC population size and SA decay rate with problem size would hence result in the initial ordering being maintained at larger sizes.







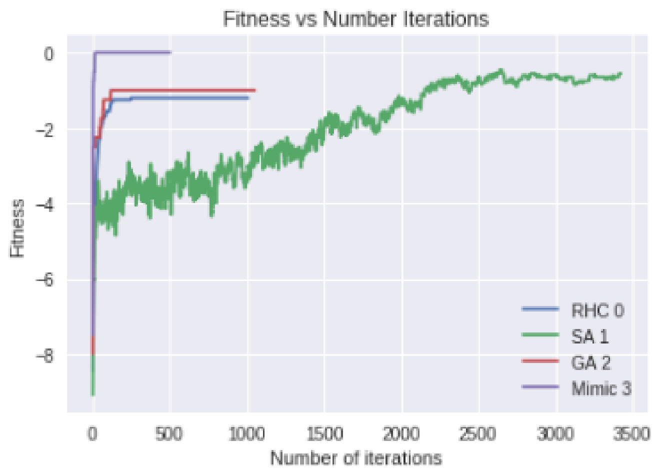


Here I present graphs representing the effects that selected hyperparameters have on the final performance (average of 20) of their respective algorithms: for each graph, other hyperparameters are held constant as given in the above page. **Make sure to note the y-axis values on each graph**, as some heavily over exaggerate any variations. RHC's relation is totally expected. Simulated annealing performance is almost constant despite vs temperature decay rate, which is due to all of the decay rates bringing the temperature down to minimum well within the iteration limit. GA is also robust to mutation rate, though interestingly there is a clear point where random mutation begins to overwhelm natural selection. The graph for MIMIC population has a rather nice inverted-parabola shape, and indicates a clear ideal percentile of points to “keep” at every iteration

N Queens

The second optimization problem I chose was the N Queens problem: here, each state corresponds to a placement of queens on an N-by-N chessboard (with one queen in each row), and here
fitness(s) = -(the number of ways queens can attack each other)
This problem provides an interesting challenge because elements of the state vector will be highly interlinked: moving even one queen can drastically change the number of ways in which it attacks and is attacked by other queen.

I evaluate with N=8, corresponding to a standard chessboard. It is worth noting that the state space of this problem is huge: if one queen is placed in each row, the number of possible arrangements (without saying anything of their fitnesses) is 8^8, which is on the order of 16 million



Again, I tested a variety of hyperparams for each algorithm, chose the optimal set for each, and graphed performance vs fitness.

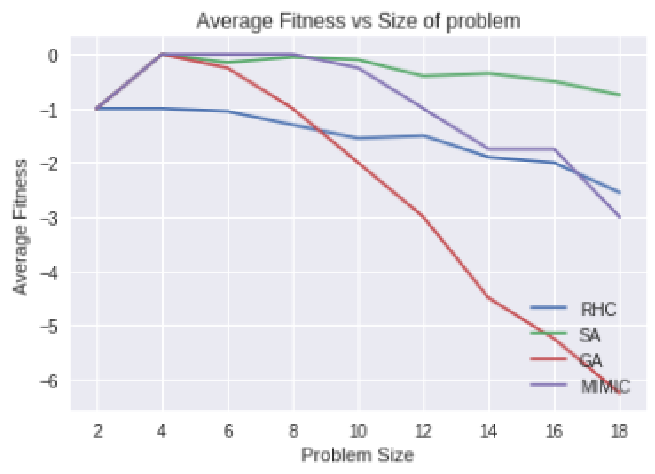
Optimal Hyperparameters:
RHC: No restarts, neighborhood of 1000
SA: T decays from 1->.001 exponentially w/ rate .005
GA: Population of 400, P(mutation) = 0.2
MIMIC: Population of 2000, keep best 30%

MIMIC is the clear winner for this dataset, achieving the global maximum of 0 mutual attacks (and being the only one to do so!) with extremely few iterations. As before, the key to its performance is its ability to exploit structure in the problem. In this case, elements of the state vector for more restrictive fitness cutoffs will exhibit high mutual information in a manner which corresponds to the way queens attack each other in the board: if the *i*th bit of the state vector is *k*, no other bits will have the value *k* for an optimal solution (ie no two queens should be in the same row). Although MIMIC can only determine these links between *pairs* of bits, with N = 8 just that information allows an optimal state to be determined.

RHC consistently reaches an attack number of one, which reflects the fact that much of the difficulty of N-Queens comes from placing the last one or two queens such that they attack the other queens in few ways. That is, as the attack number increases (0 mutual attacks being the hardest of course) it becomes easier and easier to find an arrangement exhibiting that attack number. Again, the low decay rate of simulated annealing is a double-edged sword: it slows things greatly, but allows convergence to a better optima.

Genetic algorithms also can only reach an attack number of one like RHC, but for different a different reason: the breeding mechanism. The same cutoff-based crossover which worked so well for

Four Peaks slightly hurts the algorithm here: if queens in two coordinates attack each other in one of the parents and the coordinate split does not occur between the two rows, then these queens will continue to attack each other in the child. A smarter breeding mechanism would further increase GA performance



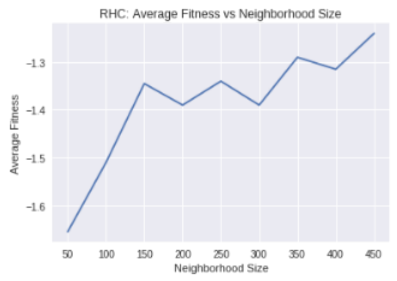
Again, I test average fitness for each algorithm versus problem size. In this problems, the fitness functions are more comparable across different values for N.

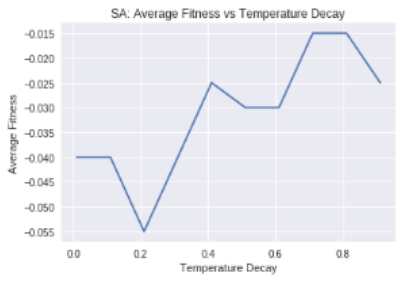
While MIMIC was uniquely advantaged in finding solutions to the 8-Queens problem, the degradation of performance with size illustrates a key fact: sometimes, MIMIC's assumption that each variable depends on at most one other variable *is simply not powerful enough.* Since every coordinate of the state vector depends on every other coordinate, as N enlarges MIMIC increasingly undermodels the system.

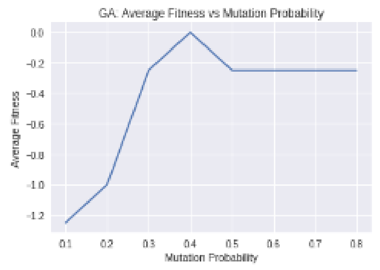
For lower values of N (up to about 10), the simplifying assumption which MIMIC makes are perfectly adequate to reach an optimal state. However, here again we keep the number of samples constant while the state space grows explosively (the number of possible states is n^n). As explained previously, increasing the number of samples with the size of the state space is a key requirement for MIMIC due to the Curse of Dimensionality.

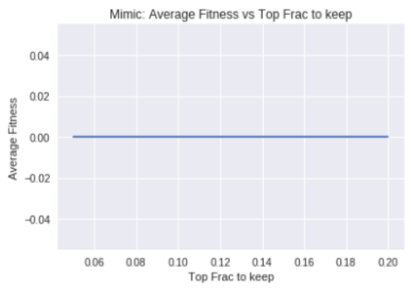
Simulated annealing degrades slower than MIMIC, but this is mostly a consequence of the slow temperature decay rather than SA being particularly suited to the N Queens problem: (indeed, the reader will notice that most of the curves for SA look more or less the same across each algorithm).

Genetic algorithms are the worst performers here, performing significantly worse than RHC (!). Again, this can be explained using the breeding mechanism: as N increases the cutoffs will preserve larger and larger sections of the board, so attacking configurations of queens will often remain unchanged.





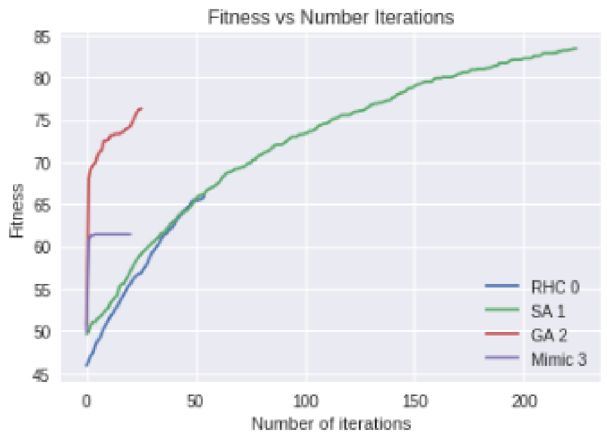




Note the y-axis on the parameter-vs-performance graph on simulated annealing especially, as it extremely exaggerates the trend there. The lessons from the RHC, SA, and GA graphs are largely the same as last time, though it is again interesting that there is such a clear tipping point in fitness for genetic algorithms. The MIMIC graph of fitness vs percentile to keep seems uninteresting, but we should note that this is because MIMIC is very well-suited to this task for values of N around 8 and 10.

FlipFlop

The next problem I selected was the FlipFlop, which is very simple:
fitness(s) = # number of bit alternations in s
Why is this problem interesting you ask? From a human perspective it's surely not, since obviously the best solution is just to "stripe" bits like 10101010... . From a machine learning perspective though, the problem is more interesting since the state space is absolutely *riddled* with local optima: as an example, take the string 00110011... it's clearly not optimal, and yet any change (bit flip) we make will leave its fitness unchanged! (excepting the first and last bits but changing those happens with low probability at any given run). Many such classes of strings exist, so there is an abundance of minima



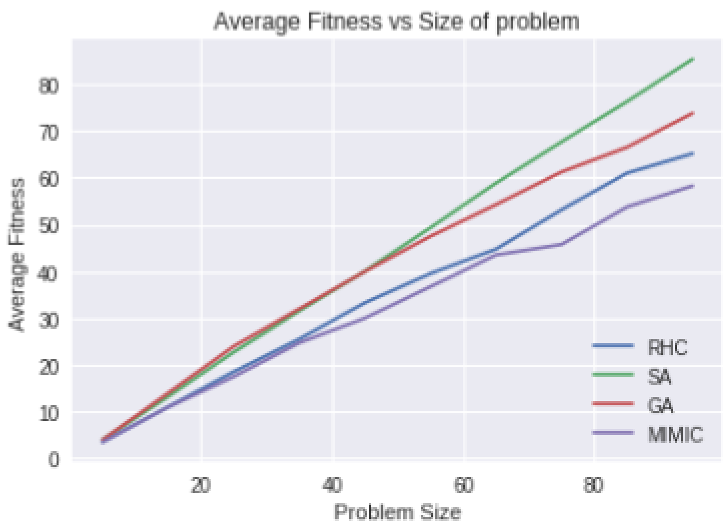
Baseline runs had state length of 100

Optimal Hyperparameters:
RHC: No restarts, neighborhood of 20
SA: Exponential decay from 1->.0001, rate=.003
GA: Population size 1000, P(Mutation) = .15
MIMIC: Population size 1000, keep top 25%

Originally, I expected that the slice-based breeding function would allow genetic algorithms to conserve striped portions of strings from both parents, ie that the breeding function would be highly suitable for the given task. I also expected MIMIC to be able to determine that consecutive bits are related as in Four Peaks.

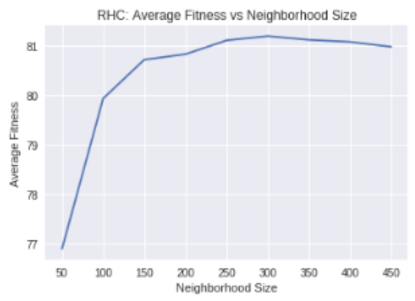
However life is full of surprises, and this happened to be one of them. Here we see the first case where Simulated Annealing has an unambiguous performance advantage over all other algorithms. This can in fact be explained by my earlier observation that the state space for FlipFlop interacts in a challenging fashion with the fitness function. The number of useless neighbors of a string is proportional to the number of alterations already in it, which helps to explain why algorithms make it past the 50-alternation mark (corresponding to half the string length), but struggle to make it to the very top.

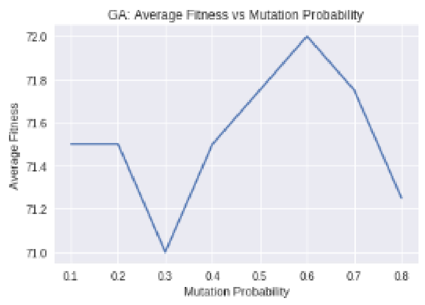
In this situation, the advantage goes to Simulated Annealing with a low decay rate: I have observed many times that low decay leads to a more exploration, but in the previous two problems this high exploration simply allowed SA to achieve performance on par with the best algorithm for the job (and with many more iterations). Here, however, frantic exploration is *precisely* the factor which allows Simulated Annealing to "tune out" the many local optima in the fitness surface and achieve near-optimal performance when the others cannot. Thus, when local optima are extremely abundant, we see that Simulated Annealing might be the way to go.

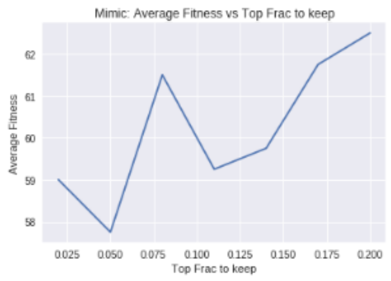


The fitness-vs-size graph reveals a lot about the relationship of the problem to the algorithms. First I will note that, for the first time, increasing problem size has no effect on the relative orderings of each algorithm. What's more, *the fitness of each algorithm seems to be increasing almost exactly linearly with the problem size!*

Although induction from limited data is never valid, we have observed some very strong indicators as to the performances of these algorithms, and we should extrapolate that Simulated Annealing will continue to distinguish itself and widen the gap between it and the other algorithms. Thus, all indications are the SA is the best randomized-optimization tool we have to solve the FlipFlop problem





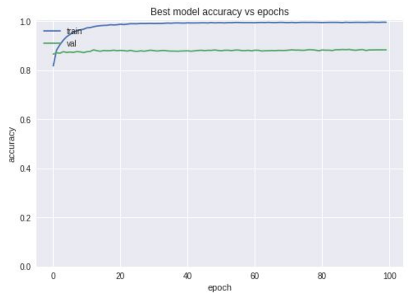


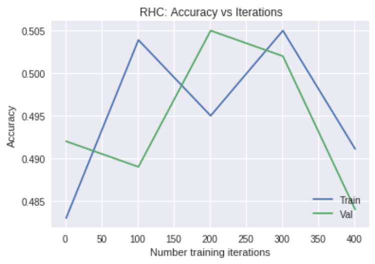
RHC's performance vs neighborhood size appears the same as ever. Genetic algorithms display an interesting robustness to mutation probability which is explainable via the fact that flipping bits of a binary string with certain probability wont, on average, change the number of alterations. MIMIC displays a weak but clear improvement with the percentile-to-keep, indicating that increasing this parameter further may increase performance and possibly vault it over RHC.

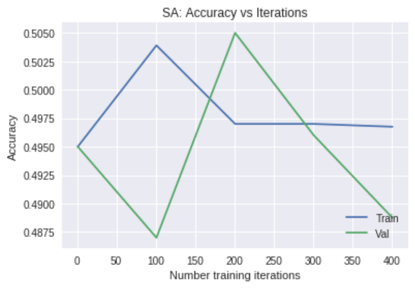
IMDB Sentiment Analysis Dataset

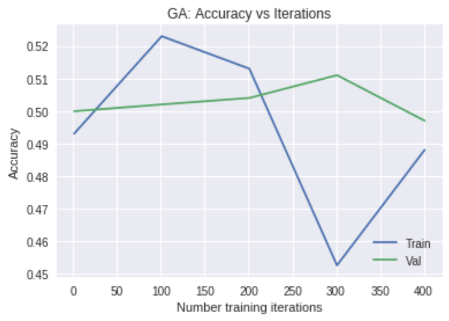
For the final problem, I train a neural network on the IMDB sentiment analysis dataset. I explain this dataset and why it is interesting in more detail in my last project, so I only provide a brief summary here: 50,000 movie reviews are transformed into one-hot vectors of the 10,000 most common word stems, and our goal is to classify each positive or negative. I use a simple network architecture which is unchanged from last time, ie three fully connected hidden layers of 50 nodes each.

Note: due to issues with numerical stability and mlrose's prediction implementation, I changed the loss function from categorical cross-entropy to simply accuracy (in fact, negative accuracy to account for the fact that we generally try to minimize a loss, but we would like to maximize accuracy). Therefore I do not provide loss-vs-parameter graphs for the following runs, as they are redundant. However, the backwards pass seems to be free of those issues: so I can still use categorical cross-entropy as the differentiable loss used in backpropagation









Using a learning rate of .05 and one hundred iterations (epochs across the data), backpropagation (**first graph**) is quickly able to achieve a high accuracy of 80%. Unfortunately, the randomized optimization algorithms are total failures: I tested a wide variety of hyperparameters (believe me, reporting this result was not my first choice :), but none performing any better than a random guesser. I attribute this to the fact that, at each point for all algorithms, only on the order of 10,000 points from the neighborhood are sampled. The number of parameters for the network however, is much much larger: there are [10,000*50 + 2*50*50 + 50*10] ~= 500,000. Therefore, the current set of hyperparameters simply do not allow any of these algorithms to explore enough of the state space to be effective (regardless of the number of iterations used).

Obviously, the natural and obvious change to make in order to alleviate this situation would be to drastically increase the number of point sampled, ideally to something in millions so that each algorithm could obtain a better estimate of the "gradient" (not to imply that any of these algorithms are calculating gradients: I simply mean better regions of the space). However, this proved infeasible because of time constraints: even with a low neighborhood size of 1000, RHC and SA take almost fifteen minutes to complete, and GA takes almost double... given these training times, it is simply infeasible to step up the neighborhood size to a point where it would make a difference.

The failure of the genetic algorithms does, at least, illustrate a critical advantage of the back-propagation algorithm: its computational complexity scales much more manageably with the dimensionality of the space since we can efficiently calculate the numeric gradients in "closed form" (via a known sequence of matrix multiplications) without resorting to sampling. In the modern era, where models often have hundreds of millions of parameters, this efficiency sets back-propagation apart.

Conclusions

Thus far, we have analyzed the performance of various randomized optimization algorithms on a variety of problems. The first three showcase relative advantages of Genetic Algorithms, MIMIC, and Simulated Annealing respectively.

The final problem shows a case in which all tested RandOpt algorithms perform very poorly, and thus illustrates the overriding advantages of the back-propagation algorithm in highly-parametrized domains. While said negative results do in fact provide valuable insight into the weaknesses of RandOpt as a whole, I do wish I had originally chosen a dataset which admitted an even smaller network architecture which would allow me to draw deeper comparisons between said RandOpt algorithms on learnable problem. Simply put, the IMDB dataset is hard given the constraints at hand: to make the problem more amenable to RandOpt I would have to reduce the size of each count vector which, because language is so varied and diverse, would in turn make the problem even harder via throwing away valuable information. However, randomized optimization techniques should be able to perform well on models with fewer parameters, and in fact are a natural choice for doing so when backpropagation is infeasible, such as in the case of nondifferentiable loss.