Machine Learning – Randomized Optimization
October 2017
Timothy Baba, Georgia Institute of Technology

Abstract – The purpose of this project is to explore and analyze three random search algorithms namely: Randomized Hill Climbing, Simulated Annealing and Genetic algorithm. This paper is divided into three parts. In the first part, we use the above random search algorithms in neural networks to analyze how they perform against backpropagation on a specific dataset. In the second part, we apply these three algorithms to three optimization problems where each problem highlights the advantage of one random search algorithm over the others. And lastly, in the third part we prune a genetic algorithm to find the best weights of a neural network that learns to play the Flappy Bird Game.

1. Randomized Optimization and Neural Network Weights

1.1 Introduction

In the last paper, we analyzed the performances of five supervised learning algorithms namely: decision tree, neural network, boosting, support vector machines and k-nearest neighbor algorithms on Pima Indian Diabetes and Breasts cancer Wisconsin dataset. In this paper, we will focus primarily on using three random search algorithms (Randomized Hill Climbing, Genetic and Simulated Annealing) in neural networks to understand how these classifiers perform against backpropagation on the Breasts Cancer Wisconsin Dataset. As a reminder, the Breast Cancer Wisconsin (Diagnostic) dataset has in total 569 instances with 30 attributes and a class variable (0 or 1) where class value 0 is interpreted as "WDBC- Malignant cancer" and class value 1 is interpreted as "WDBC-Benign cancer". The purpose of this classification problem is to predict if a patient has WDBC-Malignant or WDBC-Benign cancer based on the attributes of the dataset which provide useful diagnostic information. For this study, the dataset was randomly split into testing and training sets with 70 percent (398 instances) for training and the remaining 30 percent (171 instances) for testing. The performance of the classifier was closely observed against varying weights and parameters to arrive at good values that optimize performance.

The number of hidden layers and maximum number of nodes in each hidden layer used in the neural networks is 3 and 30 respectively which were the same values found to optimize performance by backpropagation from project1.

1.2 Randomized Hill Climbing (RHC)

RHC is an iterative algorithm which usually begins with an arbitrary solution to a problem and then tries to incrementally change a single element of the solution to improve the solution. Whenever this change results in an improved performance, the incremental change is made to the new solution and the process is repeated to the point when no further improvements can be made. RHC usually achieve solutions in convex problems or otherwise they get stuck to a local optimum.

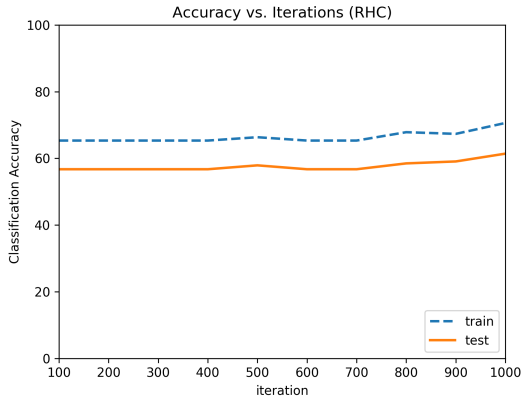


figure1.

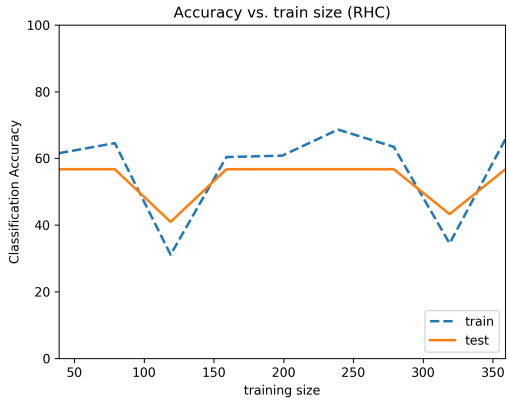


figure2.

.

We can observe from figure1 that the RHC train and test accuracies remained constant up to about 700 iterations after which we notice a slight rise in both accuracies. It can also be noticed that the train and test accuracies both fall at around 60% which is most likely due to the fact the RHC algorithm got stuck in a local optimum and wasn't able to accurately classify the train and test datasets. This is so since RHC starts at a random point and goes for the next neighbor with the highest fitness /lowest cost. More work that could be done to improve the accuracy of RHC is to restart RHC for some random number of times to decrease the chances of getting stuck in a local optimum.

1.3 Simulated Annealing (SA)

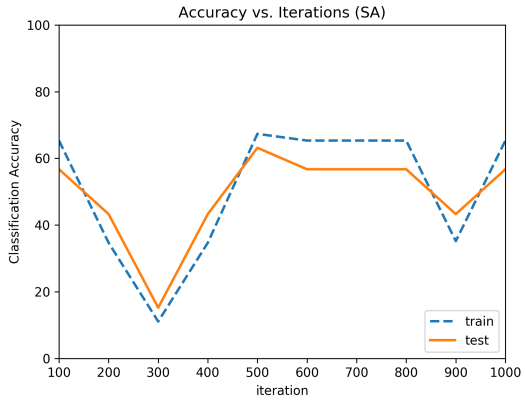
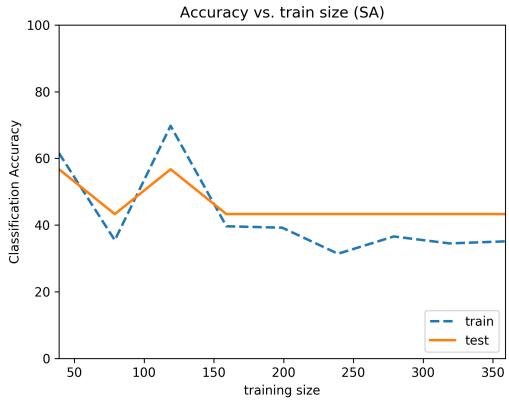


Figure3.



Figur4.

Simulated Annealing is a random search algorithm that is statistically known to guarantee finding an optimal solution. Hence this algorithm approximates the global optimum and tends to avoid getting stuck at a local optimum which is one major advantage it has over RHC. From the graph in figure4, we see that both train and test accuracies converge to a very low value of around 40%. Two hyperparameters that we can prune to improve the performance of SA are cooling exponent and start Temperature. Cooling exponent refers to any value between 0 and I that defines the rate at which temperature decreases. While the start temperature is the initial temperature corresponding to how much the algorithm can jump between data points which and is continuously decreased at each iteration of the algorithm.

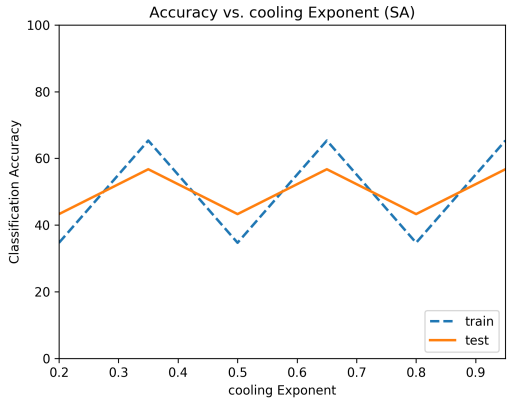


figure5

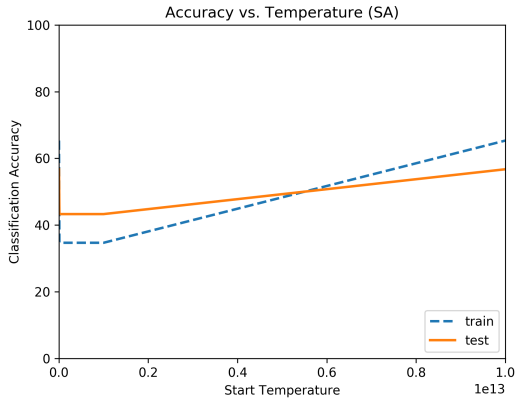


figure6

From figure5, we see that using Colling exponents of 0.35 and 0.65 both result in the highest test and train accuracies of about 55% and 65% respectively. And from figure6, we can also observe that a start temperature of 1E13 maximizes both train and test accuracies. To pick between the two cooling exponents, we could repeat the experiment with varying number of iterations for each cooling exponents and obtain the Colling exponent that converges in the lowest number of iterations and most likely has a consistent performance than the other. However, for this experiment, we chose cooling exponent of 0.35 and start temperature of IE13 as the best hyper parameters.

1.4 Genetic Algorithms (GA)

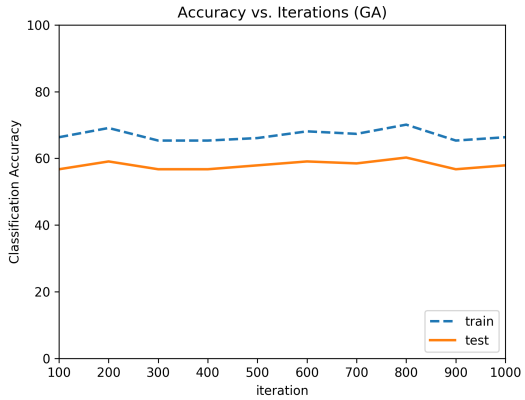


figure7

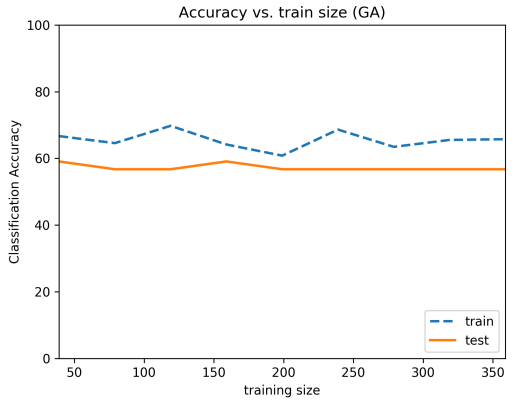


figure8

Genetic algorithm is a metaheuristic inspired by the process of natural selection and relies on bio-inspired operators namely mutation, crossover and selection. GA has three parameters - population size, number to mate at each iteration, and number to mutate at each iteration. We begin by initializing these parameters to 200, 100 and 75 respectively. As can be seen in figure 7 and 8, the train and test accuracies converged to around 65% and 60% respectively. To inspect if we can improve this performance, we prune the hyperparameters to obtain optimal values.

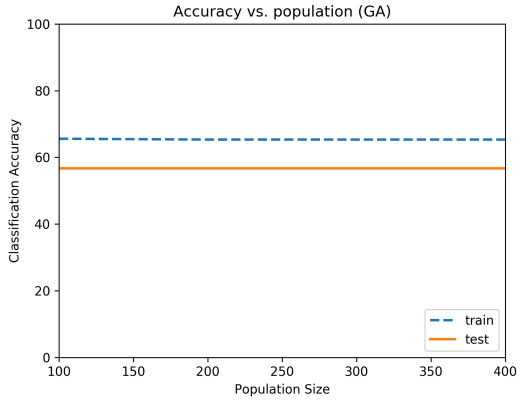


figure9

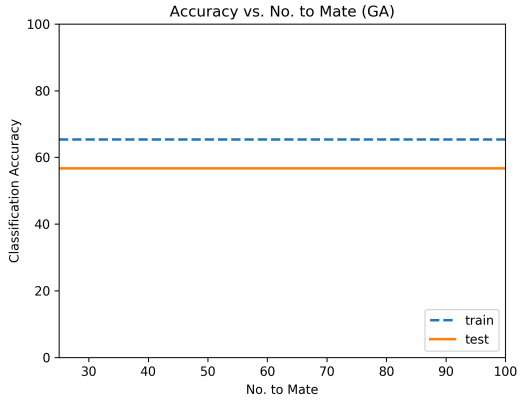


figure10

From figure9 and figure10, we can observe that the population size and number to mate respectively has no effect on the classification accuracy of genetic algorithms.

Similarly, in figure11, we see that the accuracy is constant for mutate size between 0 and 50. However, between 50 and 75 mutate size, we experience a slight rise in accuracy after which the accuracy gradually begins to decrease. Hence, for this experiment, the optimal parameters we chose for GA are 200 population size, 100 No. to Mate and 75 No. to mutate. We can observe that these values correspond to the values used in our initial experiment which implies that the optimal test accuracy we can achieve with GA on the Wisconsin's diabetes dataset is 60%.

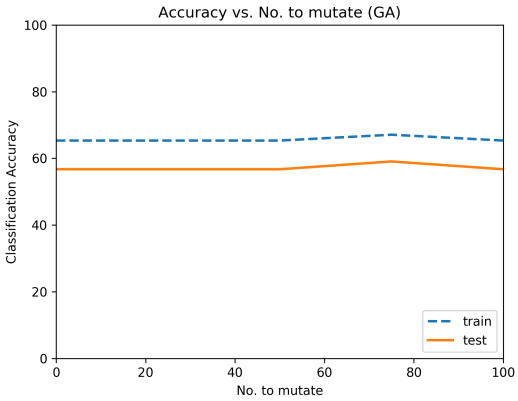


Figure 11

Conclusion

The experiment was repeated with the optimal parameters found over 1000 iterations to find out which algorithm performs best on the Wisconsin Cancer dataset. The graphs below show the final results in addition to the result from backpropagation used in Project1.

We can observe from figure13 that the train accuracies of RHC, GA and SA begin to converge to 70 % starting at around 900 iterations. While from figure12, the test accuracies of RHC, GA and

SA also converge to a score of 60% at around 900 iterations. It is obvious that that the optimal hyperparameters used in pruning resulted in the slightly improved train accuracy. Backpropagation seemed to yield a very high test accuracy of around 97% over the course of 1000 iterations.

One possible explanation to why this was the case could be that backpropagation makes use of gradient descent, and so is able to determine a good optimum with near-certainty instead of randomly sampling points as it is the case with the Random search algorithms. In addition, the dataset appears to have less noise and was well normalized and scaled which might have also helped in boosting the accuracy of back propagation.

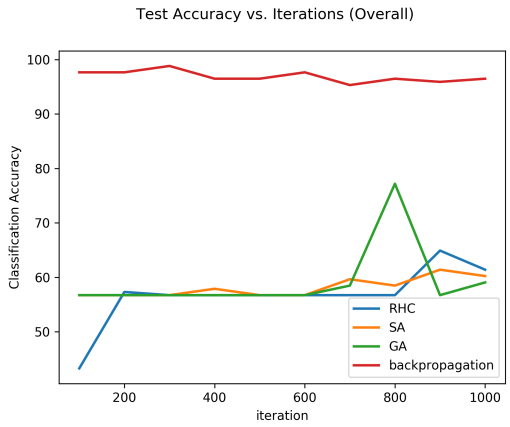


figure12.

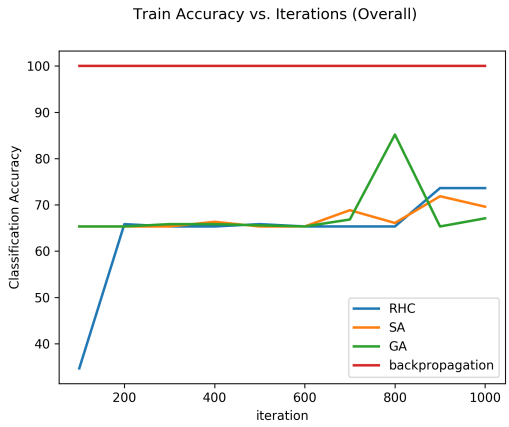


figure13.

The average time required to train on the Wisconsin cancer dataset over one iteration with backpropagation, RHC, SA, and GA algorithm was recorded to be 0.629, 0.0112, 0.0108, and 0.6417 seconds respectively. Hence it will take about 629, 11.2, 10.8, 642 seconds to train with backpropagation, RHC, SA, and GA respectively over 1000 iterations.

Thus, as we can see, even though backpropagation yielded a very high accuracy, it also took a longer time to train on the dataset. And on the other hand, it only took 10.8 seconds for SA to train on the dataset. Hence the tradeoff between accuracy and time complexity suffices. However, for this experiment, we are more concerned on a high accuracy value and as a result, we conclude that backpropagation is the best algorithm for classifying the Wisconsin Cancer dataset.

2. Three Optimization Problems

2.1 Introduction

Having analyzed the performance of the three random search algorithms on Wisconsin Cancer dataset, we now apply these three random search algorithms to three optimization problems namely; Travelling Salesman, Flip Flop and Count Ones problems where each problem highlights the advantage of one random search algorithm over the others.

2.2 Travelling Salesman Problem (GA)

The travelling salesman problem is a common problem whereby a set of cities and distances between each pair of cities are given and we are expected to find the shortest possible route that a salesman can take to visit each city exactly once and return to the starting point. We therefore explore the best randomized search algorithm suited for this problem.

Optimal parameters found for the Random search algorithms for the Travel Salesman problem

Randomized Search Algorithms	Optimal Parameters
Randomized Hill Climbing (RHC)	Iterations: 20,0000
Simulated Annealing (SA)	Temp: 1E12, CoolingExponent: 0.95, Iterations: 20,000
Genetic Algorithm (GA	Population: 200, Mate: 150, Mutate: 20, Iterations:1000

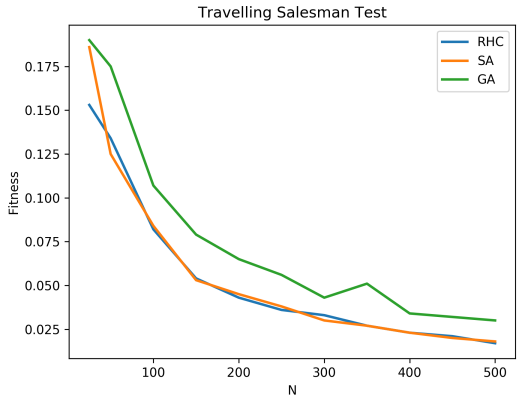


figure14

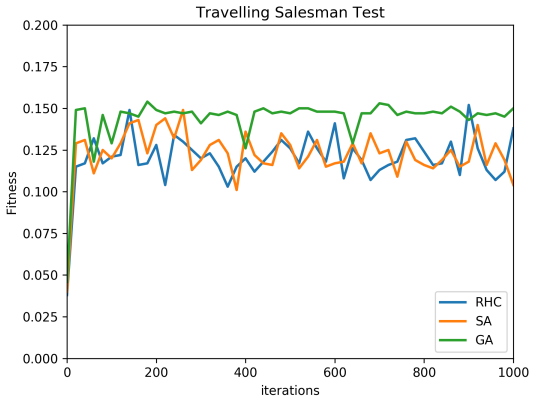


figure15

From figure 14, we see a negative correlation between N (number of cities) and fitness value. This makes sense because as the number of cities increase, the cost to travel from one city to another also increases leading to a very low fitness value. We can see that an N value of about 50 would yield a very high fitness value. Therefore, keeping N = 50 fixed and running the experiment for up to 1000 iterations, we obtain the result in figure15.

From figure15, we see that GA performed better with a fitness value of 0.15 than the other two algorithms. We observe that GA increases quickly to a high fitness value in just few iterations and converges to constant fitness value at about 200 iterations. Whereas the other algorithms got stuck at optima giving them lower fitness values.

Hence, from both graphs (figure14 and fugure15), we deduce that GA performed better than the other two algorithms. One possible reason why GA performed better could be that it is a metaheuristic algorithm and as such finds, selects and generates optimal heuristics that provided a high fitting solution.

2.3 Flip Flop Problem (SA)

The flip Flop problem is a fitness function that takes in a bit string and returns the number of times each bit in the string alternates. An ideal bit string that would maximize the fitness function would be one with entirely alternating bits.

Optimal parameters found for the Random search algorithms for the Flip Flop problem

Randomized Search Algorithms	Optimal Parameters
Randomized Hill Climbing (RHC)	Iterations: 20,0000
Simulated Annealing (SA)	Temp: 100, CoolingExponent: 0.95, Iterations: 20,000
Genetic Algorithm (GA	Population: 200, Mate: 100, Mutate: 20, Iterations:1000

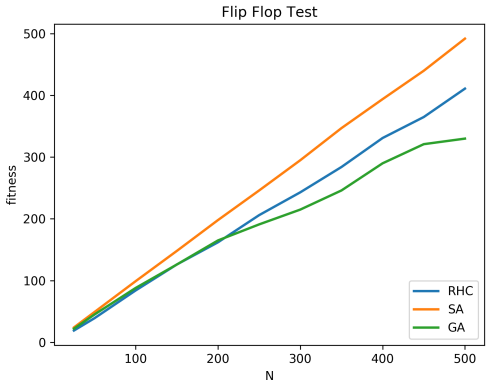


figure16

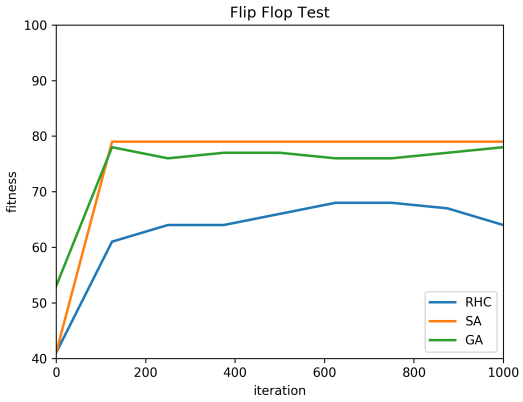


figure17

From figure16, we see that as the length of the passed in bit string increases, SA tends to increase proportionally with higher fitness than GA and RHC which both begin to fall at around 250 bit strings. From figure17, we also see that though SA starts with a lower fitness value than GA, it rises quickly to convergence point at the same amount of iterations GA took to reach convergence point making it fast and consistent. We also see from figure 17 that SA converges to a constant fitness value of 80 at about 100 iterations. One possible reason why SA performed better could be that SA generally tried to find the global maximum by slowly cooling the temperature until the highest point is found, thus giving it an edge over RHC which gets stuck to local maxima.

2.4 Count Ones Test (RHC)

The Count Ones problem uses a discrete fitness function defined for a bit-vector of length N which aims at finding a 1 filled solution. In order words, the Count Ones problem tries to maximize the number of ones in a dataset.

Optimal parameters found for the Random search algorithms for the Count Ones problem

Randomized Search Algorithms	**Optimal Parameters**
Randomized Hill Climbing (RHC)	Iterations: 200
Simulated Annealing (SA)	Temp: 100, CoolingExponent: 0.95, Iterations: 200
Genetic Algorithm (GA	Population: 20, Mate: 20, Mutate: 0, Iterations: 300

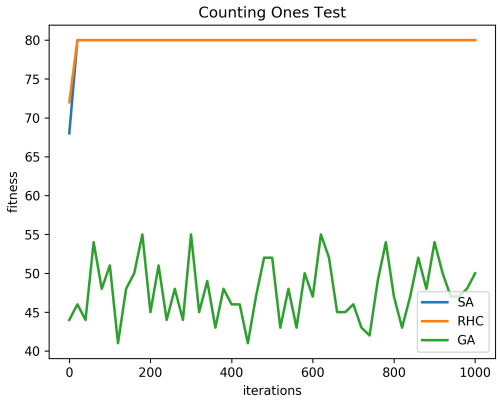


figure18

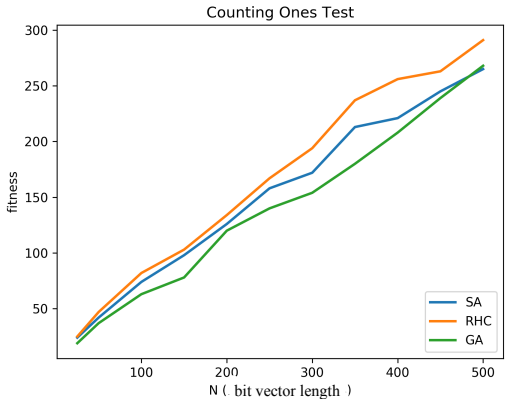


figure19

From figure18, we see that both SA and RHC reach optimal value while GA appears to be very low. The Count Ones problem clearly reflects the strength of SA and RHC when the structure of the problem cannot be discerned. This is true as we can see GA performing poorly from figure18 as the structure of the Count Ones problem cannot be discerned and as such GA does not gain much information from mutation, hence allowing RHC and SA to perform better. Also, we see that it took RHC and SA only about 50 iterations to converge to an optimal fitness value of 80. However, we can clearly notice that GA would need over 1000 iterations if it were to eventually converge.

At this point, we have already eliminated GA from the equation. Now how do we decide which algorithm (RHC or SA) is best for the Count Ones problem. First, we can observe from figure 18 that though both RHC and SA converge at same number of iterations, RHC tend to start off with a higher fitness value that SA. Second, we can also observe from graph 19 that as N (the bit vector length) increases, RHC tends to perform better that SA. As a result, the best random search algorithm for the count Ones problem can be concluded to be RHC.

2.5 Conclusion

At the end of this part we have successfully analyzed the strengths and weaknesses of each random search algorithms. We've seen all three of them defeated by backpropagation in part1 and each excel over the other random search algorithms in this part. And most importantly, we've also observed that while one algorithm might be best for a problem, it could end up been the worst in another as seen by GA in the Traveling Salesman and count Ones problems. Hence, it is always important to analyze our classifiers on different circumstances before using them on our dataset.

3 Flappy Bird Problem

In this part, we prune a genetic algorithm to find the best weights of a neural network that learns to play the Flappy Bird Game.

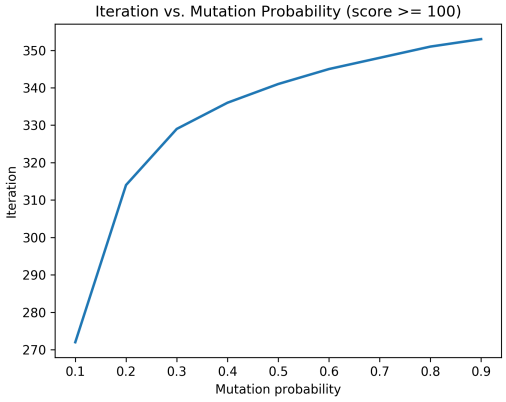


figure20

From figure20, we try to see how many iterations it will take for GA to make a score of 100 by varying the mutation probability and reading average results from 10 trails. We see from figure20 that as mutation probability increased from 0.1 to 0.2, we experience huge increase (from 272 to 315) in the number of iterations required to make a score of 100. However, as the mutation probability increased from 0.2 to 0.3 we see a less steep slope and from 0.3 to 0.9, an even lesser steep slope. This implies that as the mutation probability increases, the number of iterations required for GA to make a score of 100 increases at fairly huge amounts up to a mutation probability of 0.3 where we begin to experience very little increments in the number of iterations required for GA to make a score of 100.

Fixing mutation probability to 0.3, we repeat this experiment and attempt to observe the highest score and fitness value we can achieve while recording the number of iterations taken to achieve that score / fitness.

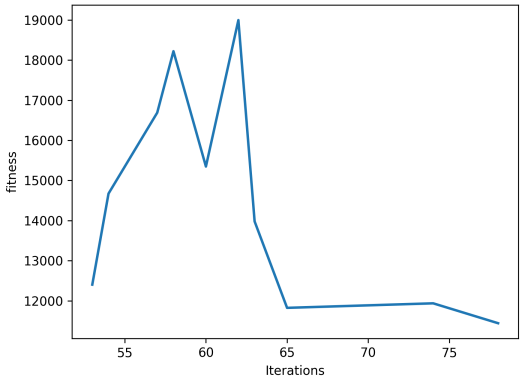


figure21

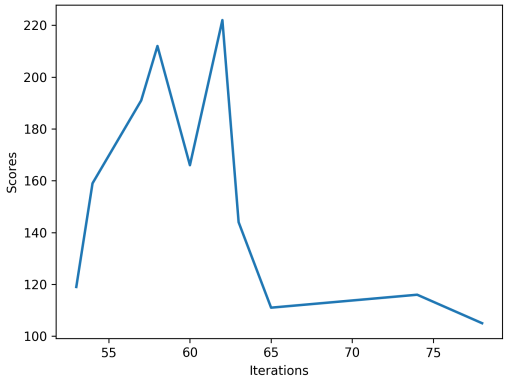


figure22

we see from figure21 and figure22 that GA converges to a fitness value of 12000 and score value of 110 at 65 iterations where we begin to observe a more stable fitness/score value.

One other thing that could be noticed from figure21 and figure22 is that fitness values and score values are directly proportional to each other. This is true because as the neural network continues to learn from its three input parameters (height of the bird, distance to the nearest pipe, and height of the nearest pipe), it gets a higher fitness value which makes the birds smart enough to move safely across more number of pipes leading to an increase in the score value.