Creating a Genetic Algorithm to Optimize an Artificial Neural Network to Drive a Car

By: James Peralta, Nathaniel Habetgergesa, and Albert Choi

When you find yourself in an elevator with Geoffrey Hinton, what will you tell him you worked on this semester?

1 Introduction

In our project, we are using a genetic algorithm (GA) to train the weights of a neural network that will drive a car. There are different types of optimization functions used for Artificial Neural Networks (ANNs), including derivative optimizations and derivative-free optimizations. Derivative-based optimizations (DBOs) iteratively improve on the parameters by using the derivative to find the best search direction and it essentially climbs the hill towards the global maxima. DBO's have been shown to have problems when there are disconnected areas in the search space and when the search space is multimodal causing the algorithm to settle in local maximas, instead of the global one. DBO techniques have been very popular for training the latest and greatest neural networks and the most popular DBO technique is the backpropagation algorithm. Derivative free optimization (DFO) is also a mathematical optimization technique but doesn't require derivative information to optimize a function. DFO's have been used to address some of the challenges that come with DBO techniques but have mostly been stuck as an active research topic. Motivated by the potential of this emerging topic we have decided to use as popular DFO algorithm (a genetic algorithm) to find the optimal weights of a neural network that will be used to drive a car in Unity.

2 Related Work

Some of the earliest literature on genetic algorithms used to optimize the weights of an Artificial Neural Network date back to the 1990s and is still very alive in academia and work that's done today [1]. A great example is David J. Montana [2] who was able to demonstrate that a GA was capable of outperforming the backpropagation algorithm on a sonar image classification problem. Their GA contained mutations and crossovers when searching for the correct weights, but they also introduced a gradient operator. Kinjal Jadav [3] also attempted to use a GA to optimize weights and obtained some helpful results. He was able to show that a very promising way to breed generations is to use a Tournament selection method with Elitism to select the best individuals from a given population. He also showed different ways of encoding chromosomes to improve the efficiency of the genetic algorithm.

3 Implementation

Every part of the neural network will be implemented from scratch using Python. This includes each perceptron, connections between each perceptron, activation functions, and loss functions. We will create a sandbox in Python where a user can test out our neural network. This sandbox will include a car attempting to navigate a race track. If it hits a wall it will disappear and spawn a new one. Over time you will see that the car will get better and better at navigating the race track. In terms of our genetic algorithm implementation, each genome will be a weight configuration with each gene representing a value of a connection in the ANN. Each gene will be decoded into a phenotype which will be an implementation of an ANN with all of its connections initialized to the weights defined in the phenotype. This ANN will be used to steer a car and their fitness function will be how far they are able to drive in the course in meters. The chromosomes of the ANNs that are able to steer their respective cars the furthest will be allowed to breed.

References

[1] K. De Jong, D. Fogel, and H.-P. Schwefel, "A history of evolutionary computation", in. Jan. 1997, A2.3:1–12.

[2] D. J. Montana and L. Davis, "Training feedforward neural networks using genetic algorithms.", in *IJCAI*, vol. 89, 1989, pp. 762–767.

[3] K. Jadav and M. Panchal, "Optimizing weights of artificial neural networks using genetic algorithms", *Int J Adv Res Comput Sci Electron Eng*, vol. 1, no. 10, pp. 47–51, 2012.