An Evolutionary Race: A Comparison of Genetic Algorithms and Particle Swarm Optimization Used for Training Neural Networks

Brian Clow, Tony White
School of Computer Science, Carleton University
bclow@sympatico.ca, arpwhite@scs.carleton.ca

Abstract

This paper compares the performance of genetic algorithms (GA) and particle swarm optimization (PSO) when used to train artificial neural networks. The networks are used to control virtual racecars, with the aim of successfully navigating around a track in the shortest possible period of time. Each car is mounted with multiple straight-line distance sensors, which provide the input to the networks. The cars act as autonomous agents for the duration of the training run: they record the distance traveled and rely on this for fitness evaluations. Both evolutionary algorithms are well suited to this unsupervised learning task, and the networks learn to successfully navigate the course in a minimal number of generations. The paper shows that PSO is superior for this application: it trains networks faster and more accurately than GAs do, once properly optimized.

1. Overview

The proposed problem is a vehicle control problem in which virtual cars navigate a course in response to simple distance sensors. Neural networks provide an elegant way to solve this problem – they are very tolerant to noisy data, and so are well suited to problems involving sensory input. Traditional programming techniques are difficult to use without assuming what the proper behavior of a car is. Cars with such assumptions would not be adaptable to different terrains or sensors configurations. Through the use of neural networks any course can be learned no matter what the environment. Sensors can be added or removed without affecting the training process – even completely new types of sensors can be added with no change to the learning algorithm. The trained networks are more tolerant to faults and changes in their design (such as a broken sensor) than a traditional program would be.

Many experiments compare the performance of algorithms by running them on a suite of small test functions. The choice of the vehicle control problem investigated here was motivated by a desire to compare algorithms on a test function that could conceivably be used in a real world application. A neural network trained for this problem could be used to control an autonomous robot equipped with infrared distance sensors. A more detailed physics model would be needed, but the neural network training procedure would work as-is. While suites of test functions provide an easy and effective comparison method, it is also important to test algorithms in situations that model real world applications of the technology.

The first training algorithms for neural networks were gradient descent algorithms such as back propagation. Such methods assess the error in the network's decision as compared to a supervisor, and propagate the error to the weights throughout the network. They are subject to problems involving local minima – since such algorithms always move towards a better solution they can get stuck at a sub-optimal solution that is nonetheless better than all nearby solutions. Evolutionary algorithms avoid this problem because they are not based on gradient information. The evolutionary algorithms operate on information about the relative performance of the individuals on the population, and are quite suitable for problems with many local minima or problems where gradient information isn't readily available.

Genetic algorithms have been used effectively to find neural network architectures [2], tune network learning parameters [11], and to optimize network weights. However, when optimizing network weights their performance may suffer due to an encoding issue known as the "permutation problem". This paper investigates particle swarm optimization, which is free of such encoding problems. It will attempt to discover if PSO provides a more effective evolutionary training method for neural network weights. In order to be more effective, PSO must consistently produce a trained network that has a higher fitness than a network trained by GA. The difference must be statistically significant – the likelihood that any difference seen is due to random chance must be very small. A two-tailed z-test will be used to determine if the difference is significant.

2. Background: Evolutionary Algorithms

Although particle swarm optimization and genetic algorithms are quite different, they are both evolutionary algorithms. Each maintains a population of individuals that represent possible solutions to the problem at hand. The population members are assigned a fitness based on their performance at the problem. At each generation the population *evolves* – each individual is changed in some way based on the fitness of the individual in the current generation. Each change may result in a member that is more or less fit, but on average the fitness of the population increases over multiple generations until an optimal solution is found.

2.1 Genetic Algorithms

Human evolution and DNA inspire this algorithm. Each population member is represented as a *chromosome* made up of a number of *genes*. The genes represent the parameters of the function to be optimized. Traditionally, the chromosomes are bit strings; in this case, they are arrays of floating point numbers that represent the weights of the neural networks. The population produces a new generation of chromosomes through the use of genetic operators that approximate recombination and mutation of DNA. The *crossover* operator exchanges entire sections of the chromosome of two parents, while the *mutation* operator modifies individual genes by small amounts. A *selection* process exists to determine which individuals reproduce. Individuals with a higher fitness are more likely to reproduce than those with a lower fitness, so the overall fitness increases as beneficial traits are passed on to future generations.

Neural networks weights can be difficult to optimize using GAs due to a problem known as the permutation problem, or the competing conventions problem [7]. This problem arises from the many-to-one relationship of chromosomes to actual networks. The hidden neurons of the network can be permuted into any order without affecting the performance of the network, since it is strongly connected in a forward direction. This makes the crossover operator very inefficient: two similar networks can be encoded differently. Crossover between two highly fit networks may produce an offspring with radically different and possibly much poorer performance.

Crossover is traditionally the driving force behind evolution in GAs, but due to its' inefficiency this convention must be examined. As suggested in previous literature [8], mutation can replace crossover as the primary evolutionary drive, in a similar manner to genetic programming. Experiments were conducted for this paper to determine if a mutation driven GA was more effective than a typical crossover driven GA. For the mutation driven GA, single point crossover was used with a crossover rate of 0.25. The mutation rate was 0.75, and each gene on a mutating chromosome had a 0.50 probability of changing by a random amount in the range [-1.0,1.0]. This paradigm was tested against a more traditional implementation with two-point crossover with a rate of 0.85, and a mutation rate of 0.08. Each gene on a mutating chromosome had a 0.70 probability of changing by a random amount in the range [-1.0,1.0]. 1000 trial runs were conducted, with each algorithm initialized to an identical random position. The mutation driven GA learned faster and more accurately than the crossover-driven GA. A two-tailed z-test showed that the results could be due to chance only 5.5% of the time, which is very close to being statistically significant. Mutation driven GA was used for comparison against particle swarm optimization.

2.2 Particle Swarm Optimization

Particle swarm optimization, while also considered an evolutionary algorithm, is quite different from GA. PSO is based on a social simulation of the movement of flocks of birds. It was first presented in a 1995 paper [6] that presented it as a new way to solve function optimization problems. A *swarm* is a population of *particles*, each of which exists in n-dimensional space. There are as many dimensions as there are parameters to be optimized, so each point in that space represents a possible solution to the function. Each particle maintains its' current position and velocity, and its' personal best position so far. The swarm itself records the global best position, at which the highest fitness so far has been found. Each time tick, the particles' velocity is stochastically adjusted by the direction to the global and personal best positions. The velocity is constrained in each dimension such that it cannot be higher or lower than a set maximum and minimum. The velocity update equation is shown in equation 1.

Velocity = InertiaWeight*Velocity +
2*rand()*(personalBest-currentPosition) +
2*rand()*(globalBest-currentPosition) **(1)**

Conceptually, the particles "fly" through n-dimensional space, exploring the solution space as they travel. When a better global best position is found, the entire swarm will migrate towards it,

exploring the more profitable solution space surrounding the global best position. The personal best provides the drive to explore the local solution space of the particle – as the swarm moves through space, particles will further explore those regions they've individually found to be profitable. The constant factors of 2 are known as the cognitive and social parameters. The cognitive parameter controls how much emphasis is placed on information learned by the particle and the social parameter controls how much emphasis is placed on information learned from other particles. Setting them both to 2 ensures that the particles will overshoot the optimum positions approximately half the time, which provides further drive to explore unknown regions.

The inertia weight controls the global and local exploration ability of the swarm by controlling the emphasis placed on the previous velocity. A low inertia means that the particles move directly towards the global/local bests. A high inertia means that the particles will only move indirectly towards the best positions, exploring a greater area of the solution space. At the beginning of training, it is desirable to have a high inertia to be sure particles explore the global solution space thoroughly. Once a good solution region is found it is important to explore it in detail, so the inertia weight is smoothly decreased down to 0.0 at the end of the training run. As the run progresses and the inertia decreases, particles tend to explore the local solution spaces and fine-tune the solutions discovered. During testing it was shown that decreasing the inertia weight resulted in a significant improvement in the mean population fitness near to the end of the run. If the inertia remains high, particles continue to explore globally and the mean fitness remained low. A constriction term is often used in place of inertia weight, but not in this implementation. The inertia model performs well at the task at hand, and was sufficient to demonstrate meaningful results.

The choice of maximum velocity is also critical to the performance of the algorithm. If the maximum velocity is too high, particles will explore the solution space in large discrete jumps, which may completely bypass good solution spaces. If it is set too low, the particles cannot obtain sufficient velocity to move to a new solution space before being attracted back towards the global/personal best positions. Decreasing the maximum velocity as the training progresses increases the performance in a similar manner to decreasing the inertia weight. The maximum velocity must remain above 0 for progress to continue as the run ends, so it is decreased to ¼ of the initial maximum velocity.

3. Related Work

The autonomous control of vehicles with neural networks has been well documented in many projects. They are effective for such tasks, as seen in the ALVINN system [9], in which a neural network was used to navigate a full sized vehicle over considerable distances. This problem is highly non-trivial since the network controls a physical vehicle rather than a simulated one. A back propagation training algorithm was used to optimize the weights of the neural network; it learned by observing the steering patterns of a human driver. This system achieved considerable success at navigating in different road conditions; it was able to drive at speeds of up to 70 mph for distance of over 90 miles. The authors further improved the performance of the training algorithms by using various other gradient descent training algorithms [3]. Their works shows the effectiveness of this class of algorithm, provided there is a supervisor available to provide error information to the algorithm.

Genetic algorithms have long been known to be effective at training neural network weights. There is extensive literature on their use for this purpose: indeed, most literature does not confine itself to solely evolving the weights, but also covers architecture evolution. Since gradient descent methods are used only for weight training, comparisons between them and GA have been particular areas of interest. Genetic algorithms have been found to be faster and more efficient [12] than back propagation for weight training, particularly in situations where gradient information isn't cheaply available. It is easier to introduce biases and introduce penalties into the fitness function of a genetic algorithm than back propagation, since one does not have to worry about the fitness function being differentiable.

Genetic algorithms are well suited to evolving the topology of networks. Balkanrishnan and Honavar published a paper [2], which outlines the major concerns and related literature. Designing the architecture of a neural network for a particular task is largely a trial-and-error process. If the architecture is not designed correctly, the network will have problems generalizing to unseen data, or even problems learning to respond accurately to training data. This motivates the use of genetic algorithms to search for the optimal network structure. The

architecture can be encoded for the GA in a direct manner where little encoding work needs to be done, or in an indirect manner that requires considerable work to decode. A connectivity matrix where each position represents the existence of one connection is an example of a direct encoding, while a grammatical encoding is considered indirect. After the network structure has been encoded into a chromosome, a standard GA can be used for evolution. To evaluate the fitness of a particular network structure, the network weights must be trained – the fitness will be the best performance of the network after limited amount of training. Any algorithm can perform the weight training – much work has been done combining back propagation and genetic algorithms in just this way.

Both the topology and the initial weights can be evolved simultaneously, as can be seen in papers by Castillo and others [4,5]. They proposed a hybrid training algorithm that used both GA and gradient descent methods. A genetic algorithm was designed that encoded both the number of hidden neurons and the network weights onto a single chromosome. A variant of back propagation called Quickprop was used to train the final network weights before fitness evaluation, and the learning rate of this algorithm was encoded into the chromosome to be optimized by the GA. The authors found that networks trained using their hybrid systems were superior to networks trained with other gradient descent and evolutionary training methods. While genetic algorithms are very good at global searches, back propagation is more efficient at a fine-tuned local search. Combining two training methods into one cohesive training structure proved to be a very effective training method.

Particle swarm optimization is receiving an increasing amount of attention, with a flood of papers in the last few years. There have been several papers comparing the performance of GA and PSO for various tasks, including the training of neural networks. A 2003 paper [10] compared the performance of the two algorithms as used for training recurrent neural networks. They found that PSO performed better for smaller networks and the GA performed better for larger networks. This finding will be investigated to see if the same holds true here.

4. Experimental Setup

The testing software was designed to simulate navigation of a racecar on a virtual course. Each car is controlled by a neural network, and is equipped with simple sensors that report the distance to the nearest object in a straight line in front of the sensor. The number of sensors can be changed: the networks were able to successfully learn to navigate the course with any number of sensors. In this simulation the cars cannot collide with each other – this was necessary to allow large population sizes to train. Cars may collide with a wall, at which point they stop until the car turns or backs away. They do not suffer damage or any sort of penalty for hitting a wall – but since they are not moving their fitness will remain low. The training course is shown in figure 1. The optimal route around the course has not been predetermined – the particle swarm will determine this on its' own.

Fig. 1. The training course is shown during a training run. The population has converged such that there are many similar individuals, as can be seen by the large number of cars in a similar position on the track.

The sensors on the cars are linked directly to the input neurons of the neural networks. Each input neuron uses the distance reported by the sensors as the activation value. The neural networks were originally provided with the car's orientation and velocity, but since in practice the presence of this information made little different to the car's performance, it was eliminated. There is a single hidden layer, and four output neurons. The outputs represent the commands that the network gives to the car. There is one for forward motion, one for backward motion, and one for rotation in each direction. A sigmoid activation function is used, so the outputs will be in the range of 0.0 to 1.0. Each time tick, a car may perform all four actions in varying degrees. Opposing actions balance each other; if the car is given commands to turn both left and right, it will turn in the direction that has the greatest output value. The amount of motion is constrained by the turning and acceleration rate of the

simulation engine. Figure one shows the structure of a typical neural network with 8 hidden neurons, controlling a car with five sensors.

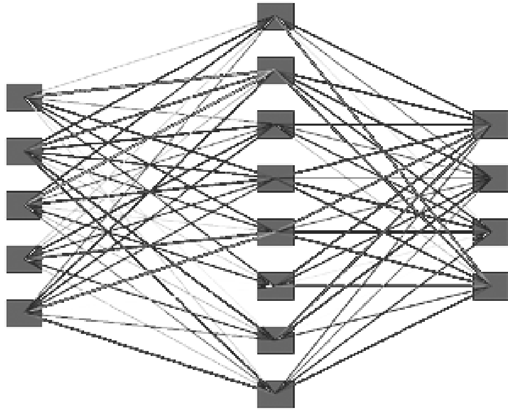

Fig. 2. This shows a typical neural network. The input neurons are on the left hand side, and the output neurons are on the right.

To train the networks, a randomized population is created and allowed to run for 250 time ticks, after which evolution occurs. There are two possible starting directions – facing directly right (90 degrees) or left (270 degrees). All cars begin with the same orientation, which is switched every generation. This helps introduce diversity into the training data. Randomness plays an important part in evolutionary training strategies, so it is necessary to run multiple tests on each algorithm. 1000 training runs were performed for each comparison, and the average best and mean fitness were plotted. The best fitness of every test run was also recorded, and the means of the results were compared using a two tailed z-test to determine if the result is statistically significant.

Both algorithms operated on populations of 50 members for 30 generations each test run. In the genetic algorithm, networks are encoded into chromosomes as arrays of floating point numbers; there is a single floating point number for each weight in the network. It used a single point crossover with a rate of 0.25, and had a 0.75 mutation rate. Each gene on a mutating chromosome had a 0.50 probability of changing by a random number in the range [-1.0,1.0]. Tournament selection was used with a tournament size that is 20% of the total population size.

The position and velocities of the particles are also stored as arrays of floating point numbers. The algorithm had an initial inertia of 1.4, decreasing to 0.0 at the end of the run. The initial maximum velocity was 2.0, and was decreased to 0.5 at the end of the run.

Some work [10] has indicated that PSO is superior for small networks only but is outperformed by genetic algorithms training larger networks. To determine if the same applies here, three sets of comparisons were performed, with different network architecture in each. The number of sensors corresponds to the number of inputs, and the number of hidden neurons was always kept slightly larger than the number of input neurons. The structure of each test is as follows.

- Test 1: 3 sensors, 8 hidden neurons, 4 outputs; 56 weights total
- Test 2: 7 sensors, 11 hidden neurons, 4 outputs; 110 weights total
- Test 3: 12 sensors, 14 hidden neurons, 4 outputs; 224 weights total

5. Results

The results of all three tests have been recorded in figures 3 through 5 below. Each of the figures clearly shows particle swarm optimization outperforming genetic algorithms in terms of both best member and mean performance.

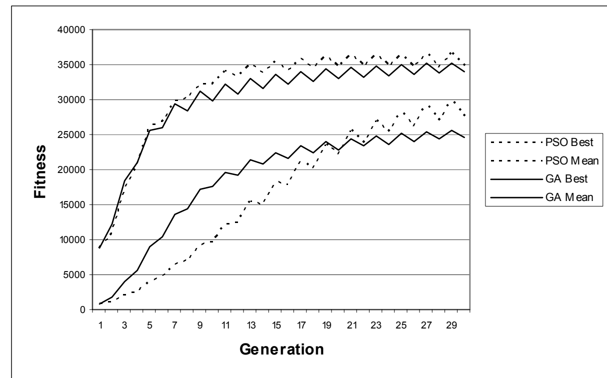

Fig. 3. 3 Sensors, 8 hidden Neurons

A two-tailed z-test (α=0.5) was performed to see if the difference apparent on the graphs was statistically significant. All three results were shown to be significant, with probabilities that the results are due to chance as follows:

- Test 1: 3.75×10^{-6}
- Test 2: 4.8×10^{-8}
- Test 3: 3.72×10^{-10}

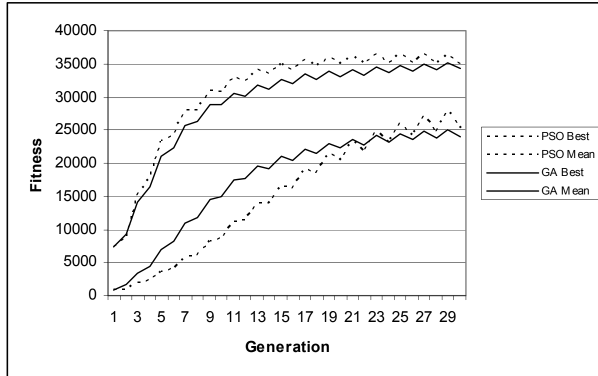

Fig. 4. 7 Sensors, 11 Hidden Neurons

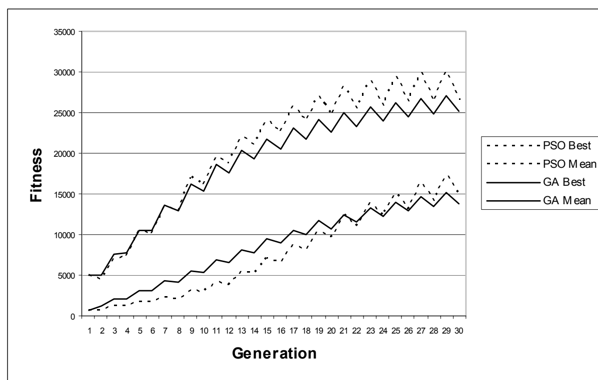

Fig. 5. 12 Sensors, 14 Hidden Neurons

6. Discussion

Graphs of the performance of evolutionary algorithms typically result in a smooth curve, with a rapid increase at the start that gradually slows down. In this case the graphs exhibited the typical growth behavior, but they were not smooth. This is due to the alternating starting orientation of the vehicles: since every vehicle starts with a different orientation each generation, it will have a slightly different fitness each successive generation. Thus the graphs exhibit the fluctuating best fitness value. If a line of best fit is drawn the graphs are a smooth curve similar to most seen in evolutionary literature.

Both algorithms were very effective at training all three different network structures. They learned to successfully navigate the course on the majority of test runs, and often reached an optimal route. PSO proved to be the superior algorithm for all tests, with both the best member and mean fitness growing quicker and higher than with the GA. The likelihood of chance resulting in the observed results was very low, much less than 1% in all cases. Although the difference in mean best member performance is not large, but a large number of test runs were conducted, and the z-test showed that the difference was significant. As the network size grew, there was no corresponding increase in the uncertainty of the results. It is possible this may change when even larger networks are used, but a network with 12 sensors should be sufficient for real world applications.

While most literature involving GA and neural networks mentions the permutation problem, it is worth noting that this is not necessarily the reason that PSO outperforms it. Hancock [7] found that the effect of the permutation problem might not be as severe as previously thought. Complex crossover operators that were designed to avoid the permutation problem were outperformed by simple single point crossover. He mentions that results have been obtained that shows that a crossover rate of 0.25 improves a GA when used for optimizing neural network weights. As mentioned in section 2.1, results obtained from testing for this paper supported these findings; the mutation driven GA outperformed the crossover driven GA.

Genetic algorithms have some abilities that PSO does not: they can be used to find the optimal structure for a network by modifying the existence of connections between neurons rather than the weights themselves. In its' present form, particle swarm optimization cannot be used to do so as it is not a real valued problem. It would be possible to combine the two by using a genetic algorithm to evolve the structure of a network while particle swarms find the optimal weight configuration during the fitness evaluation. Such uses promise to blend together the different evolutionary algorithm into one cohesive training structure. It will also be possible to modify the PSO algorithm to allow the evolution of network architecture. A discrete PSO algorithm must be implemented in which each particle represents a connection matrix specifying the existence of links between the neurons. Future work is planned to explore the effectiveness of such an algorithm.

7. Conclusion

The results of the experiment performed indicate that particle swarm optimization can be a superior training algorithm for neural networks, which is consistent with other research in the area. This paper has demonstrated the statistical significance of the

superiority for the problem described across a reasonably wide range of network architectures. PSO provides a practical alternative to the use of genetic algorithms for non-trivial problems. After many years of research, genetic algorithms have proved to provide working, practical solutions to real world problems. These results promise the same will be true of particle swarm optimization after it has gained more widespread acceptance.

In future work we intend to combine evolutionary algorithms for the generation of neural network architecture as well as weights and to explore the generation of controllers in other domains.

References

[1] Balakrishnan, K., Honavar, V.: A New Optimizer Using Particle Swarm Theory. Artificial Intelligence Research Group, Department of Computer Science. Iowa State University, Ames, Iowa (1995)

[2] Balakirshnan, K., Honavar, V.: Evolutionary Design of Neural Architectures – A Preliminary Taxonomy and Guide to Literature. Rept. CD TR95-01. Department of Computer Science, Iowa State University, Ames, Iowa (1995)

[3] Batavia, P., Pomerleau, D., Thorpe, C.: Applying Advanced Learning Algorithms to ALVINN. Tech report CMU-RI-TR-96-31. Robotics Institute, Mellon University, Carnegie (1996)

[4] Castillo, P.A., Merelo, J.J., et al: G-Prop-III: Global Optimization of Multilayer Perceptrons using an Evolutionary Algorithm.

[5] Castillo, P.A., Rivas, V., et al: G-Prop-II: Global Optimization of Multilayer Perceptrons using GAs.

[6] Eberhart, R.C., Kennedy, J.: A New Optimizer Using Particle Swarm Theory. Proceedings of the Sixth International Symposium on Micromachine and Human Science. Nagoya, Japan (1995) 39-43

[7] Hancock, P.J.B.: Genetic Algorithms and Permutation Problem: A Comparison of Recombination Operators for Neural Net Structure Specification. Proc. Int. Workshop Combinations of Genetic Algorithms and Neural Networks (COGANN-92). Los Alamitos, California (1992) 108-122

[8] Meeden, L.: An Incremental Approach to Developing Intelligent Neural Network Controllers for Robots. Adaptive Behaviour.

[9] Pomerleau, D.A.: ALVINN: An autonomous Land Vehicle in a Neural Network. Advances in Neural Information Processing System. Kaufmann, San Mateo, California (1989)

[10] Settles, M., Rodebaugh, B., Soule, T.: Comparisons of Genetic Algorithm and Particle Swarm Optimizer when Evolving a Recurrent Neural Network. Proc. Of the genetic and Evolutionary Computation Conference. Chicago, Illinois (2003)

[11] Yao, X.: Evolving Artificial Neural Networks. Proc. Of the IEEE, vol. 87, no. 9 (1999) 1423-1447

[12] Yao, X.: Evolutionary Artificial Neural Networks. Encyclopedia of Computer Science and Technology, vol. 33 (1993) 137-170