Neuro-evolution Techniques for Self-Driving Cars

Jonathan N. Schoeman
April, 2016

Abstract

This review aims to evaluate which modern machine learning techniques might be most applicable to self-driving cars. A brief background in AI and machine learning is provided along with the natural explanation of progression into neuro-evolution. The key differences between direct and indirect encoding techniques are discussed as well as some of the advantages and disadvantages of both. Finally some applications of neural-networks are discussed and what has currently been done with regard to self-driving cars and neuro-evolution.

1 Introduction

Over recent decades huge advances in the field of machine learning have been made that have paved the way for new and exciting applications that were not possible in the past. One such application is that of intelligent and robust self-driving cars. Every year thousands upon thousands of people die in car accidents because of human error. Imagine a world where cars could drive from destination to destination autonomously and efficiently without the potential risks involved of having human drivers (such drunk drivers and road rage). Creating such an advanced controller for an autonomous car would require a sophisticated "Artificial Intelligence" that is able to react dynamically to new situations. Current techniques for creating such an AI are discussed and compared below.

2 AI and Machine Learning

Machine learning is a sub category of Artificial Intelligence. It allows for a computer to potentially find acceptable solutions to problems that may have no known or exact solution or that might be difficult to describe or explicitly program. One example of such a class of problem is designing a controller for an agent that must efficiently navigate through a partially observable environment (the agent only knows about what it can see) while dynamically avoiding moving obstacles. Several different approaches to creating an effective machine learning algorithm have been proposed; many of which are inspired by nature. In this section Artificial Neural Networks (ANNs) and Evolutionary Algorithms (EAs) are discussed as well as how these machine learning techniques might be applicable to creating effective self-driving cars.

2.1 Artificial Neural Networks

The concept of Artificial Neural Networks (ANNs) was first proposed in 1943 [18]. The principles behind ANNs are modelled closely after the human brain; a biological entity that has been highly evolved to be capable of advanced learning and efficient processing of complicated cognitive tasks. The brain works by receiving a set of inputs in this case sensory inputs such as sight or touch and feeding those inputs to a connected network of neurons as an electrical signal. The network of neurons is interconnected by synapses that have the ability to alter their internal electrical potential or firing threshold. If the signal strength exceeds the current threshold, the cell will fire and the altered signal will continue to the next adjacent neuron [1]. This can be modeled to a certain extent as a directed graph with weighted edges. An ANN has one or more inputs nodes (neurons) connected to one or more output nodes with any number of hidden nodes connected in-between. The sum of the weighted inputs is inserted into some internal activation function of a node and an altered output is produced and sent to all forward connected nodes until an output node is reached.

By intelligently adjusting the connection weights of an ANN it is possible, in theory, to approximate any continuous function that has any number of inputs [6]. This makes ANNs an extremely powerful tool for use in control systems and approximations.

However ANNs are not without limitations. The ability of an ANN to correctly approximate a function largely depends on two main factors. The first of these factors is the topology of the ANN. Finding an optimal topology for a non-trivial task can be extremely difficult and time consuming. This was originally done through trial and error. The second factor is finding acceptable weight values for the ANN’s connections. This is typically done through reinforcement techniques such as back propagation. The back propagation technique uses gradient descent to minimize the mean square error between the actual output of the ANN and some expected output for a given input [15]. However this technique is prone to getting stuck in local optima and also requires explicit training examples and gradient data which may not be available or known.

It would seem that, in theory, an ANN could be used to effectively model the behavior of a self-driving car. However additional techniques will need to be employed to alleviate some the limitations associated with creating effective ANNs.

2.2 Genetic Algorithms

Evolutionary algorithms (EAs) are a class of algorithm that use techniques inspired by evolution in nature to try and iteratively and automatically optimize solutions to problems. Genetic algorithms (GAs) [12] are a sub class of EAs that “procreate” high scoring solutions. Essentially the GA searches a solution space for a set of parameters that try to optimize the results of a predefined fitness function. These parameters are encoded into a compacted representation that is known as a genotype. A population of genotypes is maintained and every iteration of the algorithm. Each genotype is expanded into its corresponding phenotype and its fitness rating is evaluated via a predetermined fitness function. Depending on the implementation of the genetic algorithm, higher scoring pairs of individuals are chosen to reproduce and a crossover function is performed on their respective genomes. This is done in the hope that the resulting genome will inherit useful traits from both parents. Some genomes may also be randomly chosen for mutation and have some of their genome parameters randomly changed in the hopes of developing new useful traits or improving on old ones. This process can be repeated until a suitably high fitness level is achieved or a set number of generations have elapsed.

Unlike the back propagation technique the optimizations performed by a GA are less likely to get stuck in local optima due to the stochastic nature of their searches. It is also worth noting that the way that genotypes are encoded can significantly change the number of iterations it takes to find an acceptable solution and can even make certain solutions impossible to obtain [19].

3 Neuro-evolution

Neuro-evolution is a machine learning technique that combines the use of artificial neural networks and genetic algorithms. A GA is used to evolve the weights of the ANN which can somewhat alleviate the common problem of getting stuck in local optima due to the stochastic nature of GAs. It is also possible to evolve the topology of a neural network however separating the evolution of connection weights and network topology can make fitness evaluation inaccurate and mislead the evolution. Therefore it has been found that evolving weights and topologies simultaneously is generally a more effective approach [19]. This is known as a Topology and Weight Evolving Artificial Neural Network (TWEANN). It should be noted that it is also possible to evolve the learning rule of an ANN however global search procedures such as GAs are computationally expensive and it is not recommended to evolve all three levels at once.

TWEANNs are a seemingly excellent candidate for developing a controller for a self-driving car because network weights and topologies can be obtained automatically. Evolving the network topology of a TWEANN also introduces some new challenges however. If the topology can change in size as it evolves then its genome representation must also change in size. This can result in the crossover operator no longer make sense when dealing with genomes of varying size and it becomes non-trivial to combine two different topologies in a useful way. Furthermore another problem is introduced known as the competing conventions problem or the permutations problem [14]. This occurs when multiple topologies evolve the same structure but are encoded differently this allows for offspring to receive the same traits from both parents effectively damaging the fitness rating of the offspring by overriding a potentially useful trait with a redundant one.

As discussed previously, depending on the task, the encoding techniques used when representing genomes can drastically affect the performance of the GA. In this section different encoding techniques are discussed. As well as how they overcome common problems associated with TWEANNs and how they might be beneficial when designing a self-driving car.

3.1 Direct Encoding

Direct encoding refers to a type of encoding where a genotype maps directly onto its phenotype counterpart. In general direct encoding techniques are considered to converge slower than indirect techniques; but are able to obtain a more accurate solution by allowing for single changes to be made to the phenotype each mutation and not being restricted only to a class of topologies that an indirect encoding can represent [4]. Some techniques that use a direct encoding approach are discussed below.

3.1.1 CNE and SANE

Conventional Neural Evolution (CNE) [19] uses direct encoding and avoids the competing conventions problem and the variable length genome problem by only evolving a concatenated string of connection weights. This means that the topology of the networks being evolved remains fixed which leads to the size of the search space being limited by predefined topologies. As a result of this CNE may converge prematurely to a sub optimum solution however it also converges to a candidate solution quickly. CNE is relatively simple to implement and is still applicable to a variety of applications.

Symbiotic Adaptive Neuro-Evolution (SANE) [13] aims to improve upon CNE by maintaining a higher population diversity and therefore avoiding premature convergence. This is done by instead of evolving networks as whole; individual nodes are evolved separately. Each generation the nodes are assembled together into fully connected networks and are assigned the average fitness rating of the networks that they are a part of. This lowers the fitness of high performing neurons making them less likely to appear multiple times in the same network this is desirable as networks with a higher diversity of neurons are likely to perform better.

3.1.2 ESP

The Enforced Sub-Populations (ESP) [9] approach is an extension of SANE. This means that it is a direct encoding technique that evolves individual neurons separately and does not evolve network topology. However ESP separates neurons into their own sub populations. Neurons only have to compete with other neurons in the same sub population. Networks are always formed from individuals from different sub populations and neurons are evaluated on how well they perform their specialized task within the global population. This stops two genomes that have been specialized for completely different tasks from breeding with each other and creating a potentially weaker offspring. As a result of this ESP is able to converge much faster to a solution than SANE.

Neurons in a specific sub population can also learn what kind of neuron they are normally attached to in a reliable way; unlike with SANE where neurons are randomly selected from a global population. This allows for recurrent topologies to be evolved reliably. A recurrent neural network (RNN) is one such that neuron connections are allowed to loop back to neurons further back in the network. This allows for an ANN to make decisions based on inputs it has processed before. Due to these key differences ESP is able to perform more effectively than SANE at much more complicated tasks [9]; however the implementation details are significantly more complicated.

3.1.3 NEAT

The neuro-evolution techniques discussed so far have all required static network topologies to be predefined before the evolution begins. Neuro-Evolution for Augmenting Topologies (NEAT) [17] was developed in 2002 to overcome the problems associated with automatically evolving the topologies of ANNs. NEAT uses a special kind of direct encoding that separates the genome into node genes and connection genes. Each node gene is explicitly and uniquely numbered within the genome and has a type associated with it (input, hidden, output). Each connection gene contains an explicit reference to an out-node and an in-node number. This allows for genotypes to be expanded into their corresponding phenotypes without any redundancy. The connection genes also each contain a connection weight, an innovation number and an activation bit. The innovation number is inspired by the *principle of homology* from genetics. Every time a connection gene is added to a genome a global innovation number is incremented and assigned to that new genes innovation number. If two identical innovations occur through independent mutations within the same generation those new genes are assigned the same innovation number. To keep the innovation numbers consistent genes are never deleted but they can be disabled by toggling the activation bit. Keeping track of the innovation history of genomes is computationally cheap. It allows NEAT to align genes with common homology and perform meaningful crossover without analyzing topology. This effectively solves the variable-length genome problem.

When new innovations are made to a topology the overall fitness of the network usually experiences an initial decrease. This could lead to the network being disregarded from the population even though it could have an excellent fitness rating once its new connection weights have had time to evolve. To remedy this NEAT uses a technique known as speciation. A difference metric is calculated between topologies and if a networks difference metric is higher than a certain threshold that network is moved into its own sub population or species. This is similar to what is done in ESP [9] only with whole networks and not just individual neurons and also species are created dynamically and not predefined. Genomes that belong to the same species must share their fitness rating with their sub population according to the *explicit fitness sharing* equation [8]. This ensures that solutions do not converge too quickly and that new innovations in topologies are given time to evolve their weights before deciding whether they are potentially effective solutions.

Unlike some other TWEANN techniques [19, 2] that start the initial population with randomized topologies; all initial NEAT topologies start off in their simplest form with all input nodes fully connected to all output nodes. Only the connection weights are randomized. NEAT is able to do this, unlike other TWEAN techniques, because it protects new innovations that are made to topologies allowing for a diverse set of topologies to develop. This essentially increases the search space of the algorithm. This also encourages faster evolution because useless topological features that were randomly generated do not have to be “unlearned” over time.

NEAT is an extremely powerful neuro-evolution technique that allows for weights and network topologies to be evolved automatically. As a consequence the size of the search space grows dynamically with the size of the neural networks allowing NEAT to find solutions to much more complicated tasks.

3.2 Indirect Encoding

Indirect encoding schemes are when genomes are encoded to specify indirectly how networks should be constructed. This can allow for impractically large genotypes to be represented more compactly. A neuro-evolution technique that uses indirect encoding is reviewed below.

3.2.1 HyperNEAT

Hypercube-based Neuro-Evolution of Augmenting Topologies (HyperNEAT) [16] was proposed in 2009 and uses a special indirect encoding called connective Compositional Pattern Producing Networks (connective CPPN). CPPNs are capable of representing huge networks compactly by exploiting repeating patterns, motifs and symmetry. This allows for networks that more closely resemble that of the human brain which contains trillions of connections. NEAT is employed to evolve the encoding of the connective CPPNs that in turn can generate complicated topologies by representing the solution as a function of the task geometry.

Research has shown that indirect encoding has an advantage over direct encoding as direct encoding techniques have difficulty optimizing very large networks with upwards of 15 000 connections [3, 11]. Therefore it is seems that the future of neuro-evolution will require some form of indirect encoding.

It would be interesting to see how HyperNeat and NEAT would compare on a task with the complexity of that of a self-driving car controller.

3.3 HybrID

Hybridized Indirect and Direct encoding (HybrID) [5] as the name suggests combines indirect and direct encoding techniques. HybrID initially evolves a population using HyperNEAT which is able to exploit the geometry of the problem. After a certain number of generations have elapsed and the algorithm reaches a switching point the entire phenotype population is switched to FT-NEAT encoding. Fixed-Topology NEAT (FT-NEAT) is a special implementation of NEAT where only the weights are evolved and the topologies remained fixed. Even though the topologies remain fixed all other aspects of NEAT remain the same such as its crossover and diversity mechanisms.

In a paper published in 2009 [5] HybrID significantly outperformed HyperNEAT in three different benchmarking tests including a quadruped controller problem. This makes HybrID a very strong candidate for use in evolving a self-driving car controller. It would be interesting to see how HybrID performs when compared directly to NEAT at such a task which is not considered in the paper.

4 Applications of Neuro-evolution

Neuro-evolution has an almost unlimited amount of applications. An example of such an application would be an intelligent agent in a video game that can learn and adapt to a players actions. Initial research in this field has already been performed in the form of a game called *NERO* and has shown promising results [7]. The most obvious real world application however would be automated, intelligent and adaptive controller models for physics based objects such as robots or cars.

4.1 Self-Driving Cars

A paper published in 2004 shows how virtual cars can be effectively trained using NEAT to drive around a track while avoiding other cars [7]. After sufficient training the cars learnt to anticipate corners and steer to the other side of the road to retain maximum velocity coming out of the turn. Champion networks were able to avoid crashing 100% of the time. Although these results seem promising the simulation was kept relatively simple and the networks were only tested on the single track they were evolved on. Whether the networks would have performed as well in an unseen environment is unknown. Unfortunately not much academic research on using newer neuro-evolution techniques such as HybrID and HyperNEAT to develop effective self-driving car controller models has been published. It would be interesting to see how these techniques perform when tasked with such a problem.

5 Summary

In summary neuro-evolution seems to be the most effective and widely used machine learning technique for solving complicated controller problems such as that of a self-driving car model. Namely NEAT, its indirect encoded extension HyperNEAT and the hybridization of the two HybrID seem to be most effective for such tasks. All three of these neuro-evolution techniques allow for complicated topologies to be evolved and optimized on their own without the need to limit the effective search space by manually specifying topologies. By combining the advantages of both direct and indirect encoding HybrID was able to consistently outperform HyperNEAT in three separate tests. This makes HybrID the most likely candidate for producing high performance self-driving car controllers. The lack of openly available research in the implementation and evolution of self-driving cars field however; prompts a direct comparison between the techniques which may yield interesting results.

References

1. ABRAHAM, A. 2005. Artificial Neural Networks. *Handbook of Measuring System Design* 8, 129. .

2. ANGELINE, P.J., SAUNDERS, G.M. AND POLLACK, J.B. 1994. An evolutionary algorithm that constructs recurrent neural networks. *Neural Networks, IEEE Transactions on* 5, 54-65. .

3. BENTLEY, P.J. AND KUMAR, S. 1999. Three Ways to Grow Designs: A Comparison of Embryogenies for an Evolutionary Design Problem. In *GECCO,* Anonymous , 35-43.

4. BRAUN, H. AND WEISBROD, J. 1993. Evolving neural feedforward networks. In *Artificial Neural Nets and Genetic Algorithms,* Anonymous Springer, , 25-32.

5. CLUNE, J., BECKMANN, B.E., PENNOCK, R.T. AND OFRIA, C. 2009. HybrID: A hybridization of indirect and direct encodings for evolutionary computation. In *Advances in Artificial Life. Darwin Meets von Neumann,* Anonymous Springer, , 134-141.

6. CYBENKO, G. 1989. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems* 2, 303-314. .

7. GHOSH, J., KUIPERS, B.J. AND MOONEY, R.J. 2004. Efficient evolution of neural networks through complexification. .

8. GOLDBERG, D.E. AND RICHARDSON, J. 1987. Genetic algorithms with sharing for multimodal function optimization. In *Genetic algorithms and their applications: Proceedings of the Second International Conference on Genetic Algorithms,* Anonymous Hillsdale, NJ: Lawrence Erlbaum, , 41-49.

9. GOMEZ, F.J. AND MIIKKULAINEN, R. 1999. Solving non-Markovian control tasks with neuroevolution. In *IJCAI,* Anonymous , 1356-1361.

10. GRUAU, F., WHITLEY, D. AND PYEATT, L. 1996. A comparison between cellular encoding and direct encoding for genetic neural networks. In *Proceedings of the 1st annual conference on genetic programming,* Anonymous MIT Press, , 81-89.

11. HORNBY, G.S. AND POLLACK, J.B. 2001. The advantages of generative grammatical encodings for physical design. In *Evolutionary Computation, 2001. Proceedings of the 2001 Congress on,* Anonymous IEEE, , 600-607.

12. MITCHELL, M. 1998. An introduction to genetic algorithms. MIT press, .

13. MORIARTY, D.E. AND MIKKULAINEN, R. 1996. Efficient reinforcement learning through symbiotic evolution. *Machine Learning* 22, 11-32. .

14. RADCLIFFE, N.J. 1993. Genetic set recombination and its application to neural network topology optimisation. *Neural Computing & Applications* 1, 67-90. .

15. RUMELHART, D.E., SMOLENSKY, P., MCCLELLAND, J.L. AND HINTON, G. 1986. Sequential thought processes in PDP models. *V* 2, 3-57. .

16. STANLEY, K.O., D'AMBROSIO, D.B. AND GAUCI, J. 2009. A hypercube-based encoding for evolving large-scale neural networks. *Artificial Life* 15, 185-212. .

17. STANLEY, K.O. AND MIIKKULAINEN, R. 2002. Evolving neural networks through augmenting topologies. *Evolutionary computation* 10, 99-127. .

18. WARREN S. MCCULLOCH AND WALTER PITTS. 1943. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics* 5, 115-133. .

19. YAO, X. 1999. Evolving artificial neural networks. *Proceedings of the IEEE* 87, 1423-1447. .