Using the Genetic Algorithm to Train Time Dependent Behaviors in Neural Networks

Hugo de Garis

CADEPS Artificial Intelligence and Artificial Life Research Unit, Universite Libre de Bruxelles, Ave F.D. Roosevelt 50, C.P. 194/7, B-1050, Brussels, Belgium, Europe.
tel: + 32 2 650 2783, fax: + 32 2 650 2785
e-mail: CADEPS@BBRNSF11(.BITNET)
&
Center for Artificial Intelligence, George Mason University, 4400 University Drive, Fairfax, Virginia, VA 22030, USA.
tel: + 1 703 764 6328, fax: + 1 703 323 2630
e-mail: HUGODEG@AIC.GMU.EDU

Keywords
Multi Strategy Learning, Genetic Algorithm, Time Dependent Neural Network Modules, GenNets, Genetic Programming.

Abstract

This paper shows that a multi-strategy approach to machine learning (in this case, a dual-strategy) can be more effective than employing a single strategy alone. The dual-strategy used in this paper is to apply the Genetic Algorithm (a form of simulated evolution) to the evolution of neural network dynamics, thus giving them the ability to be trainable, highly time dependent ways. This dual-strategy approach produces richer behaviors than the "state of the art" techniques using the traditional single-strategy (analytical) approaches. This dual-strategy approach overcomes the traditional preoccupation with time independent networks (such as feed-forward "Backprop", Hopfield convergence etc). Using this tool, the rich potential of neural network dynamics can now be explored and exploited. Several new and surprising neural net time-dependent behaviors are presented (sections 5 and 6).

1.0 Introduction

This paper has several aims. The first is to help strengthen the idea that a multi-strategy approach to machine learning can be most profitable. The second and more specific aim is to show that a simple and very powerful conceptual tool now exists to enable neural network researchers to train highly time dependent neural networks to behave in desired ways. This is an exciting development, because until now, most neural network research has been confined to the more mathematically analyzable case of time independent neural networks. There have been some attempts at inventing training algorithms for the more difficult and general case of time dependent neural nets, and the state of the art in this area [WILLIAMS & ZIPSER 1989] will be included in this paper, to serve as a basis for comparison, to show how extraordinarily flexible and powerful the approach to be introduced in this paper has proven to be. The essence of this approach is to apply the Genetic Algorithm [GOLDBERG 1989] to the evolution of the weights of fully connected neural networks, such that the outputs of these networks control or display the time dependent behaviors that are desired. The advantage of this "dual strategy" approach is that the results, in terms of the flexibility and range of new behaviors are quite superior in comparison with traditional (monostatic) approaches.

The art of building complex systems or behaviors using the Genetic Algorithm (GA) has been called Genetic Programming [de GARIS 1991]. Its great strength as a tool for the construction of (hyper) complex systems, is that it is in the nature of the Genetic Algorithm to be able to build/evolve successful and functional systems without caring about the internal complexity of the dynamics or interactions which
determine the quality of the result or performance. Only a scalar performance (or fitness) measure is of relevance in forcing the improvement of evolving solutions of the problem concerned. Thus with a "complexity independent" learning approach (whether it be applied to building/evolving neural nets, artificial nervous systems [de GARIS 1990], or artificial embryos [de GARIS 1991], those systems which were considered too complex to be analyzed in the past (and hence were usually ignored), can now at least be built. They may not be understood in mathematical analytical terms (due to the complexity of their internal dynamics) but they can at least be built now. This "buildability", allows the "qualitative" exploration of the relatively unknown world of neural network dynamics.

This paper will present a family of dynamic neural network (GenNet) examples to show how versatile and interesting GenNet dynamics can be. Some of these examples will describe new neural network dynamics which surprised the author on their (serendipitous) discovery. The existence of these new phenomena challenges the theorists to provide mathematical explanations.

Since this paper is aimed at a rather general machine learning readership, and that its domain (i.e. the "dual-strategy" approach of wedging the GA to neural net evolution) is rather unfamiliar (even for neural network researchers), it will take a rather tutorial tone. The first section will deal with the basic principles of neural networks, the Genetic Algorithm, and GenNets (i.e. how the GA is used to evolve neural net behaviors). Section 3 presents the state of the art in the traditional (analytic) approach to neural nets which show time dependent behaviors. The next four sections present examples of GenNet time-dependent behaviors. The example(s) in each of these four sections are characterized by taking one of four possible combinations of time in dependent inputs/outputs. Section 8 compares the GenNet approach to the analytic state of the art and shows how the GenNet approach is superior in terms of behavioral richness, but poorer in terms of analytical explanation. The final section discusses the idea that GenNets are sufficiently powerful to be used as modules to build artificial nervous systems in artificial creatures (biots) and a plan is introduced for future research in this direction (to build a physical insect robot with 100 behavioral and control GenNets).

2.0 Genetic Algorithm, Genetic Programming and GenNets

The Genetic Algorithm is a form of simulated evolution [GOLDBERG 1989], which uses Darwin's "survival of the fittest" idea for solving optimization problems. Genetic Programming is the art of using the GA to build (evolve) complex systems such as neural nets (GenNets), artificial nervous systems, artificial embryos [de GARIS 1991]. GP is "applied evolution", which de-emphasizes the traditional GA focus upon optimization, in favour of the importance of building hypercomplex systems which function, yet are (probably) too complex for analysis ("black boxes"). A GenNet is a Genetically Programmed Neural Net, i.e. a neural net whose weights are evolved with the GA, so that the neural outputs perform desired functions/behaviors. A GenNet is usually a fully (self) connected neural net whose weights are signed and have an absolute value of less than 1.0. If there are N neurons (N=10-16 typically), there will be N*N weights (connections).

These weight values are converted into binary fraction format (plus one bit for the sign) and concatenated onto long binary strings called "chromosomes". If P bits (P=6-8 typically) are used to code each weight, then the total length of the chromosome will be N*N*(P+1) bits. From a random bit string of this length, one can construct its corresponding GenNet. A fixed size population of these chromosomes (e.g. 50) compete with each other to survive into the next generation in an evolutionary cycle. The outputs of the GenNets thus constructed are used to specify some function or behavior, e.g. the angles of legs of an artificial creature, which vary over time and hence make the creature move in different ways. If the quality measure of the function or behavior is high (has high fitness), then the corresponding chromosome will have more offspring in the following generation. Since the population size is fixed, superior chromosomes will squeeze out inferior chromosomes, with the effect that over many generations, the average fitness increases. Offspring are subject to mutation (occasional bit flipping) and crossover (exchanging portions of two parents to make a new offspring, thus
combining two separate and favorable parental mutations in one individual offspring).

3.0 Recurrent Backprop

This section presents the recurrent Backprop algorithm [WILLIAMS & ZIPSER 1989], which is the "state of the art" in the analytic approach to training recurrent neural networks (i.e., those containing feedback loops) to behave in time dependent ways. This algorithm will later be used as a basis for comparison, in showing the superior and inferior features of the GenNet approach. FIG. 1 shows a recurrent neural network which will be used for the derivation of the algorithm. The network contains N neurons which are fully connected, i.e. each neuron has a connection to every other neuron, including a loop back to itself. Each neuron receives an external input signal, as well as the N internal input signals, from which it calculates its output signal. The network is clocked (or synchronous) meaning that all neurons calculate their outputs from their inputs at the same time.

External input vectors \( X(t) \), for \( t = 0 \), to \( t = Tp \) are presented at each clock cycle and one wishes to choose the weights such that the actual outputs \( Y(t) \) at time \( t \), are as close as possible to the desired outputs \( D(t) \), where these desired output vectors are time dependent. In effect, a mapping is being made between a set of external input vectors and a set of desired output vectors.

The technique used to find these weights, is based on the classic "gradient descent" approach. One begins by defining the total error between the set of desired and set of actual output vectors. The error between the actual and desired vector at time \( t \), for the \( k \)th component is defined as:

\[
e_k(t) = d_k(t) - y_k(t)
\]

The error for the whole output vector at time \( t \) is the sum of these terms:

\[
J(t) = \sum_{k=1}^{N} (e_k(t))^2
\]

and the total error over all output vectors is defined as:

\[
J_{\text{total}}(t_0, \tau_0) = \sum_{t=1}^{\tau_0} J(t)
\]

For given external input and desired output vectors, this \( J_{\text{total}} \) is a function of the weights. We want the actual output vectors to be as close as possible to the desired output vectors, which means that \( J_{\text{total}} \) should be made as small as possible. Since the direction of maximum increase of a function is given by its gradient, the direction of greatest decrease will be in the direction of minus the gradient. Using the fact that the total error is the sum of the errors for each time step, one can compute the gradient of the total error by finding the gradients (and summing them) for each time step. That is, the increments in the weights are defined as follows:

\[
\Delta w_{ij} = \sum_{t=1}^{\tau_0} P; \Delta w_{ij}(t)
\]

where:

\[
\Delta w_{ij}(t) = -\alpha \frac{\partial J(t)}{\partial w_{ij}}
\]

and \( \alpha \) is a learning constant.

\[-\frac{\partial J(t)}{\partial w_{ij}} = \sum_{k=1}^{N} (d_k(t) - y_k(t)) \frac{\partial y_k(t)}{\partial w_{ij}}\]

The actual output \( y_k(t) \) is calculated in the usual way, i.e. finding first the "activity" \( a_k(t) \) of the neuron:

\[
a_k(t) = x_k(t) + \sum_{h=1}^{N} w_{kh}(t) y_h(t)
\]

and then putting it through a squashing (sigmoid) function \( f \):

\[
f(x) = \frac{1}{1 + e^{-x}}
\]

so that the output becomes:-

FIG. 1 RECURRENT NETWORK
\[ y_k(t+1) = f(a_k(t)) \quad \text{Hence,} \]
\[ \forall (\partial y_k(t+1); \partial w_{ij}) = \]
\[ f'(a_k(t)) \forall (\partial y_j(t); \partial w_{ij}) + \forall \sum(h=1;N; w_k h \forall (\partial y_h(t); \partial w_{ij}))) \]

It is assumed that the initial outputs of the network are independent of the weights, so :
\[ \forall (\partial y_k(t_0); \partial w_{ij}) = 0 \]

By rewriting some terms,
\[ p_{ij}(t) = \forall (\partial y_k(t); \partial w_{ij}) , \]
\[ p_{ij}(t_0) = 0 , \]

a recursive relation between the \( p_{ij}(t) \) terms can be expressed :
\[ p_{ij}(t+1) = \]
\[ f(a_k(t)) \forall (\partial y_j(t); \partial w_{ij}) + \forall \sum(h=1;N; w_k h p_{ij}(t))) \]

These \( p_{ij}(t) \) terms can then be used to find :
\[ \Delta w_{ij}(t) = c \forall \sum(k=1;N; e_k(t) p_{ij}(t)) \]
which in turn are summed :
\[ \Delta w_{ij} = \forall \sum(t=1;P; \Delta w_{ij}(t)) \]

In practice, the weights are often modified for each of the \( p \) clock cycles, without major disturbance (provided that the learning constant is not too large).

This algorithm has been applied to such problems as :-
\( i \) A pipelined XOR (i.e. with two inputs, the output at clock cycle \( t+T \), must be the XOR of the binary input at clock \( t \).)
\( ii \) Turing Machine Simulation (i.e. learning the behavior of a Turing Machine finite state controller so that an arbitrary string with left and right brackets could be observed to see if the brackets balanced). The network was allowed to observe the actions of the controller but not its states. A minimum of 12 neurons were needed to learn this task.
\( iii \) Learning to Oscillate Simple output patterns such as \( 101010_2 \) or \( 00110011_2 \) have been generated. More interesting have been attempts to generate specific shapes in the "phase space" of the outputs, such as circles, "figures of 8", etc [PEARLMUTTER 1989].

As we shall see in the next four sections, the above achievements (although considerable by analytic methods) cannot compete with the subtlety and richness of behavior of GenNets. These four sections introduce examples of what the GenNet approach is capable of. The following section then compares the two approaches. The four sections are structured according to the four combinations of the inputs and outputs which do or do not vary in time, as shown in FIG. 2

<table>
<thead>
<tr>
<th>TIME INVARIANT OUTPUTS</th>
<th>TIME VARIANT OUTPUTS</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>TIME INVARIANT INPUTS</strong></td>
<td><strong>TIME VARIANT INPUTS</strong></td>
</tr>
<tr>
<td>CASE 1</td>
<td>CASE 2</td>
</tr>
<tr>
<td>CASE 3</td>
<td>CASE 4</td>
</tr>
</tbody>
</table>

**FIG. 2** **FOUR TIME-DEPENDENT CASES**

4.0 Case 1 GenNets: Time Invariant Inputs, Time Invariant Outputs

For case 1, it is probably more advisable to use traditional techniques such as Backprop, because they are more efficient. GP is blind and hence slow in comparison to more explicit "error correcting" techniques such as error gradient descent. However, to show that GenNets can handle such cases as well, a static vector mapping problem was solved. The problem was to choose two "arm angles" of a two component arm such that the actual final position of the arm was close to a target position (as located by the orientations of two eyes). The GenNet which was evolved to solve this problem, took as input, the two static "eye angles" (EA1, EA2) and had as output, the two static "arm angles" (JA1, JA2).

To train the GenNet, a series of points with corresponding eye angles and desired arm angles were used, as shown in FIG. 3. The fitness was defined as the inverse of the sum of the squared distances between the actual and desired positions over all the training points. Results were adequate [de GARIS 1991]. The arm even moved well to "non training" points, indicating that the evolution had generalized its vector
mapping. GenNets can certainly handle static vector mapping problems.

![Graph](image)

**FIG. 3** ARM POSITIONING PROBLEM

### 5.0 Case 2 GenNets : Time Invariant Inputs, Time Variant Outputs

It is interesting that neural net researchers were so conditioned to think in terms of convergence, that in several early attempts to apply the GA to neural networks, they did not realize that the GA would be just as effective with non convergent as convergent networks. They applied the GA to "static" networks, thus missing a major opportunity to discover a technique to build "dynamic" neural nets. The next three sets of examples will show that it is possible to overcome such tunnel vision by evolving GenNets which belong to the other 3 (non static) cases.

To illustrate case 2, a GenNet was evolved which was capable of generating sinusoid oscillations with a period which depended upon the value of a clamped (external) input control variable. In other words, the GenNet was a frequency generator - "turn up the control value, and the output oscillation period increases (quasi) linearly". This rather extraordinary GenNet was evolved using what is called here a "Behavioral Generalization" technique. The GenNet was first taught to generate two separate oscillations with two different periods (each with its corresponding clamped control value).

![Graph](image)

**FIG. 4** PERIODS OF 40 AND 80 CYCLES

For example, FIG. 4 shows the two oscillations (of periods 40 and 80 clock-cycles (i.e. the time that a synchronous GenNet takes to calculate its outputs from its inputs for each neuron in the net)) and control settings of +0.5 and -0.5 respectively. The fitness definition was the inverse of the sum of two sums (of the squares of the differences between the desired and actual output values for the two clamped settings). Thus the GenNet was evolved to be a "multi-function" GenNet. Actually, the evolution did not take place in one step. At first, two "half" oscillations were evolved, and the GenNet weights resulting from this first step became the initial weights in a second phase of evolution. The behavior resulting from the second phase of evolution retained traces of the behavior evolved in the first phase. This technique is used frequently in GP, and is called "shaping".

When an intermediate control value was applied at the input, an intermediate output frequency was generated. Thus the GenNet had learned to "generalize" its behavior. This phenomenon is quite general. One evolves the GenNet with two (or more) settings of the input control, to produce two desired behavioral outputs. An intermediate control value then produces an intermediate behavior, as shown in FIG. 5. Such a technique will be very useful in producing "steerable" GenNets for artificial nervous systems.
6.0 Case 3 GenNets: Time Variant Inputs, Time Invariant Outputs

The example chosen to illustrate case 3, was the reverse of the above case, namely, a GenNet variable frequency detector. The "stationary" output value depended upon the period of the oscillation received at the input. At first it was not obvious whether a sinusoid input would give a stationary output (using GP), but a test showed that this was possible. The fitness definition for this test was the inverse of the sum of squared differences between the actual and desired (i.e. stationary) values, taken over the 16th to 24th cycles (to allow time for settling). A more elaborate test was undertaken in which there were three desired "stationary" output values (of 0.2, 0.4 and 0.6) for input periods of 8, 16, and 24 clock-cycles, (with amplitude always at 0.5). The fitness was the inverse of the sum of three sums, (for a multifunctional GenNet) of the squared differences between the actual and desired outputs (for the three periods) taken over the 48th to 72nd clock cycles. Amazingly, this GenNet managed to evolve - not always a certainty in GP. When intermediate period oscillations were input to this GenNet, roughly intermediate "stationary" outputs were obtained. A GenNet variable frequency detector had been evolved. Other examples of "Case 3" GenNets were average signal strength (root mean square) detectors, and handling of time dependent (quantized) frequency spectra for radar [de GARIS 1991]. Similar GenNets can be used for speech processing (which also employs time dependent frequency spectra).

7.0 Case 4 GenNets: Time Variant Inputs, Time Variant Outputs

This is the most general case, and not surprisingly, the case for which the corresponding GenNets were the most interesting. When both inputs and outputs are simultaneously varying, what use can one make of such a neural net? Surprisingly (to the author at least), this question is sometimes asked in neural net research papers. The answer (in GP terms) is that "Case 4" GenNets can be evolved to control behaviors, e.g. (as mentioned earlier), if the output values are interpreted to be the angles of legs of an artificial creature (and the variable outputs are fed back to the inputs, thus creating a "Case 4" GenNet), then different behaviors can be evolved. If one wants the creature to learn to walk straight ahead, then choose the fitness definition to be the distance covered, or if one want the creature to learn to rotate clockwise, then choose the fitness definition to be the angle rotated clockwise (in a given number of clock cycles). In this way, one can build up a library of behavioral GenNets (with frozen weights), and switch them on and off at appropriate times to create an artificial nervous system for robots (called "biots", i.e. biological robots). This switching of behaviors is "smooth", i.e. it does not require jerky leg position "resets" between behaviors, because the motions evolved are (presumably) limit cycles, so that no matter when one behavior is switched off and another switched on (i.e. the outputs of GenNet "A" are input to GenNet "B"), the qualitative behavior of GenNet "B" is always generated. These switching decisions can result from outputs coming from GenNet detectors, which process signals coming from the environment. The author has already simulated a 5 behavior artificial creature, shown in FIG. 6, called "LIZZY". LIZZY was able to detect prey, predators and mates, and approach (to peck or mate) or flee appropriately. The ability to evolve behaviors in this way, allows the construction of networks of networks, and hopefully will help overcome the neural network research community's other preoccupation, namely, dealing with "one net" networks. GP is
sufficiently powerful to be able to evolve desired GenNet behaviors, and hence a new world of opportunity opens to Artificial Life researchers who want to build artificial creatures using neural net (GenNet) behavioral modules (or agents).

![Fig. 6 "LIZZY"](image)

Another example of a "Case 4" GenNet was evolved to teach a pair of stick legs to walk [de GARIS 1990]. The GenNet's time dependent output values were interpreted to be the angular accelerations of the 4 angles of the stick leg lines. The (initial) angles and angular velocities (the GenNet's 8 (variable) input values) were updated at each clock cycle using the previous cycle's (angular acceleration) outputs. Knowing the angles, the stick legs can be positioned after each clock cycle. By using shaping techniques and a "distance covered" fitness definition, the stick legs were taught to walk, as shown in Fig. 7.

![Fig. 7 WALKING STICK LEGS](image)

8.0 Comparisons

The recurrent Backprop algorithm is the state of the art in neural network analysis. It is recent [1989] and should stimulate further theoretical contributions to neural net theory. It has been seen as rather a breakthrough, given the preoccupation with "static" neural nets of the previous decade. However, it is not difficult to see its fundamental limitations. The greatest is probably that it is "supervised", i.e. the desired time dependent output vectors need to be known. Imagine trying to build a neural net using recurrent Backprop to get LIZZY to move straight ahead or to turn. The advantage of the GenNet approach is that one does not need to know how to specify in detail and over time, how the leg components should move so that LIZZY moves as desired. The GenNet approach is "unsupervised" in this sense (except for the fitness value which does give some very diluted (but sufficient for learning) form of supervision). The GenNets used to get LIZZY to move, were "self assembling" in a sense. They found their own solution to the evolutionary problem at hand. However, recurrent Backprop has a stronger analytical basis than does the GenNet approach. It is more appealing to analytically minded people, who prefer to understand the systems they study, and treat "black boxes" (such as GenNets) with some disdain. This point will be taken up further in the next section. It would be nice if there were some kind of "Theory of GenNet Dynamics". Unfortunately this is not yet the case, so until such theory is found, the GenNet approach will remain vulnerable to the criticism, "GenNets may be behaviorally interesting, but they are not understood".

9.0 Discussion and Future Directions

The previous section showed the greater flexibility of the GenNet approach (e.g. for building artificial neural networks) than using the recurrent Backprop algorithm. By using Genetic Programming (GP), the neural network research community now has a tool to begin investigating a wealth of unexplored dynamical neural net phenomena, such as those discovered in sections 5 and 6 (generalized behavioral generation and detection). As these phenomena are discovered, motivation will increase to give them analytical explanations, so that (with respect to GenNets), "theory" can catch up with "experiment", (making an analogy with the hand-and-glove situation between theory and experiment in particle physics, for example).

However, it is possible that finding a complete dynamical theory (of GenNet Dynamics) will prove to be too difficult. The fact that most neural net research over the last 30 years has been confined to "static" networks, has not been without reason. Fully connected neural networks are "complex (non linear) systems" in the sense of having complex dynamics (attractor points, limit cycles, and chaos). Such systems are notoriously difficult to analyze mathematically. For centuries, mathematicians and theoretical physicists shied away from modelling non linear...
behaviors. But with modern computers, such behaviors can be simulated and investigated. The phenomenon of GenNets is a part of this new computer simulation revolution. This kind of computerized discovery by simulation will probably become increasingly commonplace in the future as computers become powerful enough, fast enough and parallel enough to simulate such complex systems as artificial nervous systems, artificial embryos, artificial life etc. The attitude that mathematically oriented researchers often take towards simulation (namely that it is less "serious" because there is "too little analysis") will have to be softened, because in a growing proportion of future discoveries, the "cutting edge" will come from the computer people, not the mathematicians. It is likely that the "simulators" will play the important role of providing the phenomena to be explained, and the mathematicians (the analysts) will play the equally important role of attempting to explain the phenomena. The analysts are dependent on the simulators' results, because otherwise what would they have to analyze?

Returning to more concrete considerations, GenNets have proven to be so versatile, that they appear to be excellent candidates as building blocks for artificial creatures, as mentioned in section 7. Future plans are to build a physical 100 GenNet insect robot, with roughly 50 behaviors and 50 controls. This robot will contain electronic GenNet chips, all functioning in parallel, so that the robot can be given a large number of behaviors. Finally then, by using a dual-strategy approach to machine learning (i.e. using the GA to teach a neural net to behave over time in desired ways), a methodology (i.e. the Genetic Programming of Neural Networks (GenNets)) has been developed which is not only more behaviorally versatile than traditional analytical neural net training algorithms (such as recurrent Backprop), but is powerful enough to be used in building much more adaptable artificial creatures. Admittedly the "black box" nature of the GenNet approach may have less appeal to researchers who prefer to have an analytical understanding of the neural networks they are dealing with, but at least a tool now exists which is powerful enough to permit both the exploration and exploitation of the behavioral richness of fully dynamic neural networks. Hopefully, GenNet discoveries such as "Generalized Behavioral Generation" (section 5) and

"Generalized Behavioral Detection" (section 6) presented in this paper, will prove to be only the tip of the iceberg of what remains to be discovered in neural network dynamics.

References:

[de GARIS 1990] "Genetic Programming: Building Artificial Nervous Systems Using Genetically Programmed Neural Network Modules", Hugo de Garis, ICML90 (7th Int. Conf. on Machine Learning), June 1990, Austin, Texas, USA.


