**GenNETS : GENETICALLY PROGRAMMED NEURAL NETS**

**Using the Genetic Algorithm to Train Neural Nets**
**Whose Inputs and/or Outputs Vary in Time**

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
email: CADEPS@BRRNSF11(.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
email: HUGODEG@AIC.GMU.EDU

**Keywords :**


**Abstract :**

The neural network research community's obsession with convergent networks is not unreasonable. Relatively little analytical work has been done on neural networks whose inputs and/or outputs vary in time, hence few guidelines exist on how to train such networks. Consequently, research has concentrated on more restrictive "static" neural nets such as "feedforward" (Backprop) [RUMELHART et al 1986] and "Hopfield" (clamped inputs, convergent outputs) [HOPFIELD 1984]. This emphasis on convergence is unfortunate, because the true richness of neural network dynamics is to be found when inputs and/or outputs are time dependent. This paper shows that the Genetic Algorithm can be applied successfully to training nonconvergent networks, and presents some examples of their extraordinary behavioral versatility.

**Introduction :**

The major reason why the neural network research community has concentrated upon convergent neural networks (where the output is static or converges to stationary values) is that such networks are much more easily mathematically analyzable. Theorists have only vague ideas concerning the dynamics of nonconvergent neural networks. Without theory to guide the creation of "dynamic" neural network training algorithms, the field of nonconvergent neural nets has remained largely stagnant. This is most unfortunate, because it is generally recognized that the true richness of neural network behavior is to be found in the realm of nonconvergence, especially when both inputs and outputs vary in time.

The widespread belief that no generally acceptable procedure exists to train nonconvergent networks is false however. There is a means to train nonconvergent neural nets to perform useful functions. It is called the Genetic Algorithm (GA). The overpowering advantage of the GA is that, provided one can supply a scalar performance measure of the
function or behavior determined by the neural outputs, the internal complexity of the neural dynamics is irrelevant. The GA is only concerned with the final measure of the quality of performance (i.e., the fitness). This means that the neural dynamics can be as complex as one wishes. One can evolve neural nets which perform useful functions, despite time varying inputs and/or outputs. This paper presents at least one example of a GenNet for each of the four possible cases of time variant/invariant inputs/outputs, as shown in FIG. 1.

<table>
<thead>
<tr>
<th>TIME IN Variant</th>
<th>TIME IN Variant</th>
</tr>
</thead>
<tbody>
<tr>
<td>TOP INPUTS</td>
<td>CASE 1</td>
</tr>
<tr>
<td>CASE 2</td>
<td>TIME TOP INPUTS</td>
</tr>
<tr>
<td>CASE 3</td>
<td></td>
</tr>
<tr>
<td>CASE 4</td>
<td></td>
</tr>
</tbody>
</table>

FIG. 1 THE FOUR TIME-DEPENDENCY CASES

Section 1 gives a brief summary of the Genetic Algorithm, and the Genetic Programming (GP) of neural nets. (GP is the art of using the GA to "build" (evolve) complex systems). Section 2 shows how GP techniques were used to evolve GenNets whose operating conditions satisfied the 4 cases of FIG. 1, as well as showing some of the extraordinary capacities of time dependent GenNets. Section 3 makes a plea to the neural network research community to "shift its sights upwards" by devoting more effort to thinking about "dynamic" neural networks in general, and the theory of GenNet dynamics and "evolvability", in particular.

1. 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) [de GARIS 1990a,b, 1991a,b,c], artificial nervous systems [de GARIS 1990b], artificial embryos [de GARIS 1991b], Lisp S expressions [KOZA 1990]. 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).

2. The 4 Cases

i) Case 1: Time Invariant Inputs, Time Invariant Outputs

For case 1, it is probably more advisable to use traditional techniques such as Backprop [RUMELHART et al 1986], 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).

![Image of arm positioning problem]

**FIG. 2  ARM POSITIONING PROBLEM**

To train the GenNet, a series of points with corresponding eye angles and desired arm angles were used, as shown in FIG. 2. 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 1990a]. The arm even moved well to "non training" points, indicating that the evolution had generalised its vector mapping. GenNets can certainly handle static vector mapping problems.

ii) Case 2: 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 Generalisation" technique. The GenNet was first taught to generate two separate oscillations with two different periods (each
with its corresponding clamped control value). For example, FIG. 3 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. It is called "shaping", as shown in FIG. 3.

![FIG. 3 PERIODS OF 40 AND 80 CYCLES](image)

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. 4. Such a technique will be very useful in producing "steerable" GenNets for artificial nervous systems.

![FIG. 4 VARIABLE OUTPUT PERIODS](image)

Setting = -0.7  Setting = +0.3

iii) Case 3: 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 1991c]. Similar GenNets can be used for
speech processing (which also employs time dependent frequency spectra).

iv) Case 4: 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 [e.g. HIRSCH
1989]. 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. 5, 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. 5 "LIZZY"](image)

![FIG. 6 WALKING STICK LEGS](image)

Another example of a "Case 4" GenNet was evolved to teach a pair of stick legs to
walk [de GARIS 1990b]. 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 could 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. 6

3. A Plea to the Neural Net Research Community

Hopefully, the previous examples have persuaded readers that a technique does indeed exist to train neural nets which are non convergent, and that the behaviors shown by the GenNets evolved using this (GP) technique are more than interesting enough to be worthy of further study. Therefore, it is hoped that more neural net researchers will begin to use GP techniques. Unfortunately, despite the fact that GP has "been around" for nearly two years [de GARIS 1990a,b, 1991a,b,c], few neural net researchers seem to be aware that the neural net "convergence straightjacket" is no longer needed.

However, GenNets are not without their own problems - the major one being the uncertainty of whether a GenNet with a desired functionality or behavior can be evolved at all. In practice, one finds that some GenNets never "get off the ground", i.e. their fitnesses level off at unacceptably low levels, or (equivalently) their evolvability is too low. What are now needed are mathematical criteria for good GenNet evolvability. The author challenges theorists in the domains of neural networks, genetic algorithms, and complex dynamics to discover these criteria, so that GPers no longer need to treat their GenNets as "black boxes", i.e. as interesting, powerful, but mysterious.

References:

[de GARIS 1990b] "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.