GENETIC PROGRAMMING

Modular Neural Evolution for Darwin Machines

Hugo de Garis

Machine Learning and Inference Laboratory,
Artificial Intelligence Center, George Mason University,
4400 University Drive, Fairfax, Virginia 22030, U.S.A.
email HUGODEG@GMUVA2.GMU.EDU

Keywords :

Genetic Programming, Neural Networks, Genetic Algorithm, Modular Neural Evolution, Agents, Society of Mind, GenNets (Neural Networks designed with the Genetic Algorithm), Hierarchies of Neural Modules, Sequential Evolution, Genetic Programming Software Shells, Darwin Machines, Wafer Scale Integration, Nanotechnology.

Abstract :

This paper introduces the concept of Genetic Programming, which employs the Genetic Algorithm (GA) [GOLDBERG 1989] to design both Neural Network (NN) [RUMELHART et al 1986] modules (GenNets) and their control circuits. The GA is used to find the weights (and their excitatory-inhibitory signs), of fully connected neural networks with feedback. Once a GenNet module performs sufficiently well, its weights are frozen, and the module is then used as a component in more complex circuits. The outputs of NN control circuits are the inputs to these frozen modules. Once the “control circuit plus modules” (considered as a unit), functions as desired, the weights of the control circuit are then frozen. This larger frozen unit can be considered as a component for a yet higher stage of design. This hierarchical module building is similar to Minsky’s “Society of Mind” theory, where a GenNet module is equivalent to his concept of an “agent”, and Genetic Programming is related to his idea of “mind design” [MINSKY 1986, 1988]. A second concept is also briefly introduced, namely that of the Darwin Machine, which performs GenNet evolution directly in hardware. WSI, (wafer scale integration) (now) and nanotechnology (later), will allow such machines to be built.

Introduction :

The conceptual problems involved in designing and controlling neural computers with m(b,t)illions of processors can be discussed today. It will be impossible to program each processor individually, and the internal dynamics and connections between processors will be too complex to analyse. Several conclusions and suggestions result from this.

a) Modules of neural nets will need to be treated as black boxes. Only their performance will be of concern. Full analysis of their internal behaviour will have to be abandoned.
b) Neural modules will be designed by the Genetic Algorithm, using coded chromosomes which will compete with each other to reproduce, according to the quality of their performance.
c) These neural modules (agents) will be combined to form functional hierarchies (“agencies”, a la Minsky) using neural control circuits which themselves will be evolved by the Genetic Algorithm.
Modular Neural Evolution:

To illustrate the above ideas, a simple example of modular neural evolution is presented. It is the two-eye, two-joint robot arm positioning simulation problem. FIG. 1 shows the basic setup. The aim of the task is to move the robot arm from its vertical start position X to the goal position Y. J1 and J2 are the joints, E1 and E2 are the eye positions, and JA1, JA2, EA1 and EA2 are the joint and eye angles of the point Y.

![Diagram of robot arm](https://via.placeholder.com/150)

FIG. 1

The modular approach is illustrated by specifying that two different neural net modules will be evolved. The first, called the “joint module”, controls the angle JA that a given joint opens to, for an input control signal of a given strength - and the second, called the “control module”, receives inputs EA1 and EA2 from the two eyes and sends control signals to the joints J1 and J2 to open to angles of JA1 and JA2.

FIG. 2 shows the basic circuit design that the GA uses to find the “joint” and “control” modules. The joint modules (two identical copies) are evolved first, and are later placed under the control of the control module. Each module (containing a user specified number of neurons) is fully connected, including connections from each neuron to itself. Between any two neurons are two connections in opposite direction, each with a corresponding (signed) weight. The input and output neurons also have “in” and “out” links but these have fixed weights of 1 unit. The outputs of the control module are the inputs of the joint modules as shown in FIG. 2.

The aim of the exercise is to use the Genetic Algorithm to choose the values of the signs and weights of the various modules, such that the overall circuit performs as desired. Since this is done in a modular fashion, the weights of the joint module are found first. These weights are then frozen, and the weights of the control circuit found so that the arm moves as close as possible to any specified goal point Y. Each weight is coded onto a GA chromosome with a sign, (where 0 means an excitatory synapse, 1 means an inhibitory synapse) followed by a user specified number of places after the “binary point”. For example, an inhibitory weight of 5 binary places, having value 101101 would take the binary value -0.40625. A chromosome coding for a module of N neurons (hence N*N signs and weights) would have a total length of N*N*6 binary positions. All weights are expressed to the same number of places (e.g. 5).

The activation of each neuron is determined in the usual way, namely the sum of the products of the incoming signal strengths and the corresponding weights of the connections. The neuronal transfer function was chosen to be \( \frac{2}{(1+ \exp(-actv))} - 1 \), to give an output with a range of -1 to +1. Weight values also ranged between -1 and +1, so as to avoid unbounded output values. With both weights and transfer functions restricted to the -1 to +1 range, output values stabilised, (usually after about 50 cycles or so for 1% accuracy). In each cycle, the outputs are calculated from the inputs (which were calculated
in the previous cycle). These outputs become the input values for the neurons that the outputs connect to.

The Genetic Algorithm is then used to choose the values of the weights, such that the actual output is as close as possible to the desired output. To evolve the joint module, 21 input values ranging from -1 to +1 in steps of 0.1, were used. The desired output values were chosen to be half the input values, thus ranging from -0.5 to +0.5, and were interpreted as being the number of turns of a joint, (e.g. +0.5, i.e. half a turn, would mean a joint angle of 180 degrees. A positive angle is clockwise).

No crossover or inversion was used in the Genetic Algorithm, [GOLDBERG 1989], since the problem of neural network design is so nonlinear. Changing one weight influences the outputs of all the neurons. Hence the GA used only mutation (a small probability (e.g.0.001) that each binary value, whether sign or weight, on a chromosome would flip) and selective reproduction, (i.e. those chromosomes obtaining a superior score compared to others in the population, reproduces in proportion to their superiority). The quality measure used in the evolution of the joint module was the inverse of the sum of the squares of the differences between the desired and the actual output values.

FIG. 3 shows the set of points Y used to evolve the control module. These points all lie within a radius of 2 units, because the length of each arm is 1 unit. For each of these 32 points, a pair of “eye angles”, EA1 and EA2, is calculated and converted to a fraction of one half turn. These values are used as inputs to the control circuit. The resulting 2 “joint angle” output values JA1 and JA2, are then used to calculate the actual position of the arm Y’, using simple trigonometry. The quality of the chromosome which codes for the signs and weights of the control module is the inverse of the sum of the squares of the distances between the 32 pairs of actual positions Y’ and the corresponding desired positions Y.

<table>
<thead>
<tr>
<th>WEIGTHS</th>
<th>FROM NEURON</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0   1   2  3</td>
</tr>
<tr>
<td>TO</td>
<td>0</td>
</tr>
<tr>
<td>NEURON</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>3</td>
</tr>
</tbody>
</table>

FIG. 4
FIG. 4 shows an example solution for the 16 weight values of the joint module. With these weights (obtained after roughly 100 generations with the GA), the actual output values differed from the desired values by less than 1%. Similar solutions exist for the 36 weights of the control module, which also gave distance errors less than 1% of the length of an arm (1 unit). What was interesting was the fact that every time a set of weights was found for each module, a different answer was obtained, yet each was fully functional. The impression is that there may be a large number of possible adequate solutions, which gives Genetic Programming a certain flexibility and power.

Darwin Machines:

Genetic Programming is a new programming methodology which requires a full research program to develop it. More ambitious projects employing Genetic Programming need to be undertaken, such as trying to design a time dependent system which balances and walks, or a system which can detect certain kinds of objects. Such systems would be hierarchical in nature and appropriate for Genetic Programming.

Within 5 years, it will be possible to put 10 million artificial neurons on a Wafer Scale Integration (WSI) superchip [RUDNICK et al 1989]. Thus machines can be built to implement GP directly in hardware. These machines have been called Darwin Machines in this paper. Later still, nanotechnology (molecular scale engineering) [DREXLER 1986, REED 1988, LANGTON 1989, SCHNEIKER 1989] will be able to build Darwin Machines on a much more impressive scale.

Using Darwin Machines, Genetic Programmers will be able to specify, in a very high level language, such things as functional requirements for GenNets, output/input connections between modules, GA parameter values, number of neurons per module etc. The Darwin Machine will then perform the Genetic Programming directly in hardware and at great speed. It is likely that Darwin Machines will be incorporated as components in real time devices, such as walking robots, and that real time inputs will be fed directly to these Darwin Machines for evaluation. In the immediate future however, industrialists will be able to construct Genetic Programming Software Shells which will perform many of the functions mentioned above, but at a slower software pace.

References: