Genetic Algorithms

Manuel Alfonseca
IBM SofTec Lab
Paseo de la Castellana, 4
28046 Madrid
Spain

Abstract

Genetic algorithms, invented by J.H. Holland, emulate biological evolution in the computer and try to build programs that can adapt by themselves to perform a given function. In some sense, they are analogous to neural networks, but there are important differences between them. This paper shows that genetic algorithms are easy to program, test and analyze by means of APL2 functions.

Introduction

The idea of emulating biological evolution to generate a working computer program is not new in computer science. There are references (see reference 1) as old as 1966 or more. The question is to make use of the following mechanisms, already classic for biologists:

- Random change (variability)
- Natural selection

In other words: given a population of similar but slightly different entities, subject the whole population to a "test of fitness" that will decide which elements of the current population are better adapted to the performance of a given piece of work (a computation or whatever). Increase the probability that these elements will survive and reduce the probability for the less adapted elements. Then "reproduce" the most adapted elements, that is, obtain slightly modified copies of them and have these copies replace an equal number of the less adapted elements.

Genetic algorithms are one of the main current approaches to automatic computer learning. Therefore, their range of applications overlaps that of other learning systems, such as neural networks. Genetic algorithms have been applied to game learning, sensory-motor coordination of video-cameras, simulation of gas pipeline systems, learning multiplexer systems, parallel semantic networks, calculation of primitive recursive functions, and many others.

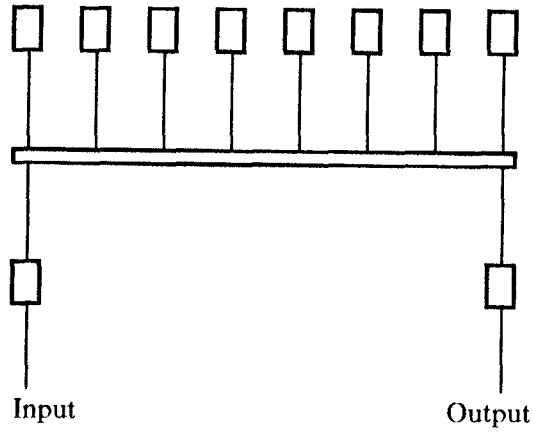
The current approach to genetic algorithms comes from the work of J.H. Holland and his team (see references 2 to 8). In principle, their procedure can be expressed as a three layered system:

- A classifier system
- An award-punishment system
- A genetic algorithm

Classifier Systems

A classifier system is a rule-based production system. It can be considered as a set of many interconnected units called CLASSIFIERS. Each classifier is a rule and its only goal is to send a message (fire). All the classifiers may fire at given instants of time (synchronous fire).

The following figure represents the structure of a classifier system.



The units at the top are the classifiers in the system. Every classifier incorporates a rule of the form:

```
condition : message
```

where condition and message are two character strings usually of the same length.

Classifiers are connected to one another by means of a "message list" (the central rod in the figure). At a given instant in time, a classifier compares the condition of its rule with the messages in the list. If one or more messages fit the rule condition, the classifier is ready to fire. Firing a classifier successfully means inserting the message part of its rule into the message list.

Normal classifiers have an input and an output, both of them connected to the message list. But a few special units (indicated in the lower section of the figure) have only their output or their input connected there, while the other comes from the environment. They are called "input" and "output" units.

Binary logic is enough to represent messages. However, in the case of conditions, ternary logic is usually used, including a "don't care" character. For example, the following rule:

```
01**110 : 1110001
```

can be fired by the following set of messages:

```
0100110 0101110 0110110 0111110
```

Multiple conditions may be accepted. In this case, the joint condition is assumed to be the logical AND of all of them. That is, there must be messages in the list that fit with every sub-condition. Of course, different sub-conditions may be satisfied by different messages. An example is the following:

```
01**110 , *000010 : 1110001
```

which means that this classifier will be ready to fire if there are two messages in the list, one of which satisfies the condition 01**110, and the other satisfies *000010. Of course, depending on the sub-conditions, the same message could satisfy more than one.

Messages stay in the list during a single step in time.

It is possible to prove that classifier systems are computationally complete, the same as:

- Turing machines
- Digital computers
- Analog computers
- Neural networks

This means that any algorithm whatever, that can be solved by a computer, can also be solved by some classifier system. Nothing is said, however, about performance.

Award-Punishment System

A classifier system is a simple computational facility. To convert it into a learning system, some award-punishment system must be added. The award-punishment system most commonly used in genetic algorithms is the "bucket brigade" algorithm. Its aim is to make sure that awards are given, not only to the actual classifier who made the final desired output available, but to all others that made it possible.

For instance: let us assume that the input unit put the following message into the message list:

```
01010101
```

and that the following result message is desired:

```
00001111
```

Then, the following set of classifiers will generate the desired result in four steps:

```
0**10101 : 11111000
**11100* : 10101010
1010**** : 00000000
00000*00 : 00001111
```

The objective of the "bucket brigade" algorithm is to make sure that the efforts of all four classifiers are recognized. It modifies the classifier system on which it acts, in the following way:

- The message list is assumed to have a limited capacity, much smaller than the number of classifiers in the system. If more classifiers are ready to fire, not all of them will be able to do it.
- Each classifier is assigned a "strength", usually an integer.
- A classifier may be ready to fire, because its condition matches one of the messages in the list. However, this does not mean that firing will be successful, for the capacity of the list may have been exceeded. (There may be more classifiers ready to fire than the list capacity allows). In this case, all classifiers ready to fire must "bid". Only those with the highest bid, will be able to put their messages in the list. The final decision can be deterministic, or probabilistic.
- The bid B of a given classifier C is computed with the following formula:

  ```
  B(C)=a.s(C).Sp(C)
  ```

  where

  - a is a small constant
  - s(C) is the current strength of the classifier C
  - Sp(C) is a measure of the specificity of its rule, for example, the ratio of the number of don't care characters to the total number of characters in the rule condition.

In this way, the classifiers with maximum strength and those with maximum specificity have a better possibility of firing than those with smaller strength and those whose rule conditions are able to fire almost always.

- When a classifier fires successfully, it pays the amount of its bid to those classifiers that made it possible. Its strength is reduced by the bid, and the bid is distributed between all those classifiers that in the preceding instant of time put a message in the list that fitted with the rule condition of this classifier.

Genetic Algorithm

A genetic algorithm is a procedure that "reproduces" the most adapted classifiers (those with the largest strength) and generates new (slightly different) children that replace some of the less adapted classifiers in the population and are tested for fitness in the same way as their parents.

The genetic algorithm consists of the following steps:

- Select pairs of classifiers from those with the maximum strength. This selection is usually probabilistic.
- Reproduce these pairs, generating copies of them.
- Modify the copies slightly by applying a "genetic operator".
- Select the classifiers to be replaced from those with the minimum strength. This selection can also be probabilistic.
- Replace the selected classifiers by the new ones.

Different genetic operators can be used, alone, or in combination. The most typical ones are:

- Mutation, that replaces one character in the rule of the classifier by another character chosen in the following way:
  - If the mutation affects the rule condition, the new character is selected randomly from the set '01*'.
  - If the mutation affects the rule message, the new character is selected randomly from the set '01'.
- Crossover, that takes the two parents and recombines their rules exchanging a randomly selected set of characters between them. For instance: if the parents are

```
00*01* : 110110
1*0*11 : 001001
```

and the crossover occurs at the fourth character in the rule, the resulting "offspring" will have the rules:

```
00**11 : 001001
1*001* : 110110
```

The application of genetic algorithms emulate the action of evolution and make it possible the generation of new and possibly better classifiers. In this way, a classifier system that had gone into a dead end by climbing a path to a sub-optimal goal will sometimes be able to jump to a better solution by using the newly generated classifiers.

APL2 Implementation

In the following examples, for simplicity, we will include the restriction that the condition of every classifier is unique. Extension to multiple conditions would be straightforward, anyway.

Classifier systems are easily represented in APL2. Each classifier can be represented as a two element vector, the first element of which is the condition, while the second is the response. Each of these elements will be a vector of characters, selected from the set '01*' in the first case, and from the set '01' in the second.

The whole classifier system will be a vector of classifiers, defined as above.

The strength of the classifiers, needed to apply a bucket brigade algorithm, can either be included as a third element of each classifier, or be maintained in a separate variable. We will choose the second alternative in these examples.

The APL2 function GENETIC generates a classifier system of N elements, where M is the length of both condition and output for each classifier. The global variable CLASS will contain the classifiers, while the global variable ST will contain the corresponding initial strengths. The rules are random, and the strengths are initialized to 100.

The APL2 function ACTION embodies the three layered algorithm. It has two arguments: the input to be applied to the classifier system and the goal to be generated.

The first section (resolution of the classifier system) comprises lines 2 to 8. They work in the following way:

- Line 3 generates an empty message list. The list is a matrix of two rows. The first row contains the

message and the second identifies its sender. This information will be used by the bucket brigade algorithm, to provide payment to the classifiers that have produced a successful message (a message that is the goal or allowed another classifier to fire).

- Line 5 inserts the input in the message list. Since this function works in 1-origin, 0 can be used to identify the input element of the classifier system.
- Line 6 displays the current state of the message list.
- Line 7 identifies those classifiers whose conditions are satisfied by the messages in the message list. Variable A is a Boolean vector where the 'ones' correspond to them.
- Line 8 displays the numbers of the classifiers that are ready to fire.

```
[0]  N GENETIC M
[1]   CLASS←(⊂¨⊂[2]'01*'[?(N,M)ρ3]),¨⊂¨⊂[2]'01'[?(N,M)ρ2]
[2]   ST←Nρ100
```

```
[0]  GOAL ACTION INPUT;A;LIST;B;I
[1]   ⍝(1=≡GOAL)/'GOAL←⊂GOAL'
[2]  L0:I←0
[3]   LIST←2 0ρ0
[4]  L:→(FGEN<I←I+1)/L3
[5]   LIST←LIST,(⊂INPUT),0
[6]   'LIST' LIST
[7]   →(0=+/A←∨/LIST[1;]COMPARE↑¨CLASS)/L3
[8]   'WOULD FIRE:'(A/⍳ρCLASS)
[9]  ⍝ BUCKET ALGORITHM
[10]  B←0.125×A×ST                    ⍝ Get bids
[11]  A[MAXL↓⍒B]←0                    ⍝ Select maximum bidders
[12]  'FIRE:'(A/⍳ρCLASS)
[13] L03:ST[A/⍳ρST]←ST[A/⍳ρST]-A/B ⍝ Decrease strength of winners
[14]  (A/B)ADJSENDER¨⊂[1]LIST[1;]COMPARE(A/1⊃¨CLASS)
[15]  LIST←A/(2⊃¨CLASS),[0.5]⍳ρCLASS
[16]  →(~∨/A←GOAL COMPARE LIST[1;])/L
[17]  'Goal found in list'
[18]  ST[A/LIST[2;]]←1.25×ST[A/LIST[2;]]
[19]  →L
[20] ⍝ GENETIC ALGORITHM
[21] L3:B←(⍒ST)[4?4⌈⌊0.1×ρCLASS]
[22]  A←(⍋ST)[4?4⌈⌊0.1×ρCLASS]
[23]  'Crossover applied to classifiers ',(⍕2↑B)
[24]  '  (',(⍕CLASS[2↑B]),')'
[25]  '  substituting classifiers ',(⍕2↑A)
[26]  '  to become ',⍕CLASS[2↑A]←CROSSOVER CLASS[2↑B]
[27]  'Mutation applied to classifiers ',(⍕2↓B)
[28]  '  substituting classifiers ',(⍕2↓A)
[29]  '  (',(⍕CLASS[2↓B]),')'
[30]  '  to become ',⍕CLASS[2↓A]←MUTATION CLASS[2↓B]
[31]  ST[A]←100
[32]  →L0
```

```
[0]  B ADJSENDER X;N
[1]   N←X/LIST[2;]
[2]   →(0∈N)/0
[3]   ST[N]←ST[N]+B÷+/X
[5]   'Classifier(s) ',(⍕N),' strength increased by ',⍕B÷+/X
```

```
[0]  Z←X COMPARE Y
[1]  ⍝ X is a list, Y is a set of conditions
[2]   Z←∧/¨'*'=(X∘.≠Y)COMPRESS¨((⍴X),⍴Y)⍴Y
```

```
[0]  Z←X COMPRESS Y
[1]   Z←X/Y
```

```
[0]  Z←CROSSOVER X;N;R;A;B
[1]   R←⍴↑↑X
[2]   A←∈↑X
[3]   B←∈2⊃X
[4]   N←?⍴A
[5]   Z←(⊂[2](2,R)⍴(N↑A),N↓B)(⊂[2](2,R)⍴(N↑B),N↓A)
```

```
[0]  Z←MUTATION X;N;A
[1]   A←?2⍴2
[2]   N←?2⍴⍴↑↑X
[3]   Z←X
[4]   (N[1]⊃A[1]⊃↑Z)←'01*'[?A[1]⊃3 2]
[5]   (N[2]⊃A[2]⊃2⊃Z)←'01*'[?A[2]⊃3 2]
```

The second section (award-punishment system) is made of lines 9 to 19.

- Line 10 computes the bids of the ready-to-fire classifiers as one eighth of their respective strengths. In this example, specificity of the rules has not been considered, but the line could be easily modified to account for it.
- Line 11 selects the maximum bidders. The global variable MAXL is supposed to define the capacity of the message list (the maximum number of simultaneous messages it may contain, apart from the input). This number should be much smaller than the total number of classifiers in the system.
- Line 12 lists the actual classifiers that were able to fire in this step of the computation.
- Line 13 decreases the strength of the winners by their respective bids.
- Line 14 identifies the senders of the messages that made it possible for the winners to fire and adjusts their strengths accordingly.
- Line 15 puts the messages of the winners in the message list.
- Line 16 finds out whether the goal is in the list. If not, control is given to line 4 for a new time step.
- Line 17 indicates that the goal was found.
- Line 18 awards the classifier that has generated the goal, by increasing its strength.
- Line 19 returns to line 4 for a new time step.

The third section (the genetic algorithm) is made of lines 20 to 32. The genetic algorithm receives control once in a while. The global variable FGEN defines the number of time steps to simulate before the execution of the classifier system is interrupted to perform a genetic operation on the classifiers. Lines 2 and 4 control this interruption procedure.

The genetic algorithm also receives control if the message list becomes empty at any time.

- Line 21 selects randomly four classifiers from those in the upper 10 percent of the population

(according to their strengths). These will be the ones chosen for reproduction. The first pair is selected for crossover, the second, for mutation.

- Line 22 selects randomly four classifiers from those in the lower 10 percent of the population (according to their strengths). These will be the ones chosen for elimination.

  Of course, other selection strategies may be used here. It would be enough to replace lines 21 and 22 accordingly.

- Lines 23 to 26 perform crossover on the first pair of chosen classifiers, substitute the first pair of eliminated classifiers and inform the user.
- Lines 27 to 30 perform mutation on the second pair of chosen classifiers, substitute the second pair of eliminated classifiers and inform the user.
- Line 31 initializes the strength of the new classifiers to 100 (the initial value).
- Finally, line 32 returns control to the first section of the algorithm.

Auxiliary functions used by the preceding program are listed above.

Conclusion

The paper demonstrates successfully the ease with which genetic algorithms can be simulated in APL2. It will be observed that the number of loops is reduced to the minimum (one): the inescapable time step loop. Different genetic algorithm procedures may be tested easily by changing a single line of code.

This paper does not contain original work, except in the language chosen for the implementation of Holland's algorithms, as they are described in reference 7.

References

1. L.J.Fogel, A.J.Owens, M.J.Walsh, *Artificial Intelligence through Simulated Evolution.* John Wiley and Sons, Inc., New York (1966).
2. J.H.Holland, *Adaptation in Natural & Artificial Systems* University of Michigan Press (1975).
3. J.H.Holland, *Escaping brittleness: The possibilities of General-Purpose Learning Algorithms Applied to Parallel Rule-Based Systems.* In *Machine Learning: An AI Approach,* Vol. 2. Ed: R.S.Michalski, J.G.Carbonell, T.M.Mitchell. Morgan Kaufman Inc (1986).
4. J.J.Grefenstette (ed), *Proceedings of the First Intnal. Conference on Genetic Algorithms and Their Applications,* Lawrence Erlbaum Associates, Hillsdale, NJ, 1985.
5. J.J.Grefenstette (ed), *Proceedings of the Second Intnal. Conference on Genetic Algorithms and Their Applications,* Lawrence Erlbaum Associates, Hillsdale, NJ, 1987.
6. L.Davis (ed), *Genetic Algorithms and Simulated Annealing* Morgan Kaufmann Publishers, Los Altos, Ca., 1987.
7. L.B.Booker, D.E.Goldberg, J.H.Holland, *Classifier Systems and Genetic Algorithms.* The University of Michigan, Cognitive Science and Machine Intelligence Laboratory, Technical Report 8, 1987.
8. D.E.Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning.* Addison Wesley Publishing Co., Inc., Reading, Mass., 1989.