Evolution of Learning Rules for Supervised Tasks I: Simple Learning Problems

Ibrahim KUSCU
Cognitive and Computing Sciences
University of Sussex
Brighton BN1 9QH
Email: ibrahim@cogs.susx.ac.uk

November 10, 1995

Abstract

Initial experiments with a genetic based encoding schema are presented as a potentially powerful tool to discover learning rules by means of evolution. Several simple supervised learning tasks are tested. The results indicate the potential of the encoding schema to discover learning rules for more complex and larger learning problems.

Keywords:Evolution, Genetic Algorithms, Supervised Learning

1 Introduction

Evolution and learning address two different levels of adaptation processes: one taking place during the life time of an organism and the other taking place during the evolutionary history of population of organisms [3] [18].

There are quite a number of research concentrating on the relationship between evolution and learning [2] [16] [11] [19] [1] [17]. The nature of interaction between the two has been shown to be complementary : the presence of learning can facilitate the process of evolution and evolutionary methods can significantly speed up the learning process.

However, in these studies, the methods of learning have been chosen beforehand and frequently back-propagation algorithm is used. Evolution is used as a mean of creating an initial state upon which learning process is applied for increased performance. However, the nature of learning during an evolutionary process has not been investigated.

An interesting experiment was carried out by Chalmers [3]. His primary aim was to observe how learning might evolve during the process of evolution. Rather than looking at the interaction between evolution and learning, the major focus of his study was evolution of learning mechanisms themselves. Starting from a random population of genome coding for the weight-space dynamics of artificial neural networks, he found out that a learning rule which is very close to the delta rule has evolved successfully in learning eight linearly separable tasks of supervised learning. A similar experiment is applied in the area of unsupervised learning to find rules for the Self-Organising Map [4]. They have supported the idea that genetic algorithms can be used to search for optimal learning algorithms by providing evidence for the existence of several learning algorithms that produced an organisation similar to that of Kohonen Algorithm. These experiments encouraged the hope that it is not impossible for such genetic based systems to discover new learning algorithms. However, realising this hope is not easy. For example, trying different supervised learning problems at a larger scale or trying another class of learning such as reinforcement learning would extensively increase the complexity of the problem. Besides, a fundamental problem in attaining such goal is the problem of genetic encoding (i.e. representation). How can we code for such large and complex dynamics? How and what prior knowledge should be introduced to reduce the scale and the complexity of the space?

The way a genome is represented has a significant affect on the way Genetic Algorithms (GA)[12] perform in finding the solution to a particular problem. GA works *directly* on the representation of the problem encoded on genome. Thus, the search space within which the possible solution lies is *directly* influenced by the representation. It is widely recognised that fixed length character strings pose severe limitations for the solution of particular problems [5] [6] Several representation schema which improve the capabilities of Genetic Algoprior-knowledge (domain-specific knowledge) to the representation of the problem. Although this reduces the scale and the complexity of the search space, it also effectively introduces possible low level solutions to the problem in hand. This form of human intervention makes it less attractive for a learning paradigm. This issue is extensively discussed in [15].

In this paper, as part of an ongoing research an encoding schema will be presented. Combined with genetic algorithms it can successfully produce evo-

This expression is randomly produced for a problem with two input values. *I1* and *I2* are the variables to be instantiated to the input values from the patterns at each time of evaluation.

When generating the expressions a variable parameter called *percentage* is used to impose how complex we want the expressions (i.e. longness or shortness of the expressions). It can have values from 0 to 100. The higher the percentage value the more complex the expression tends to be. In the experiments variable *percentage* values are used depending on the complexity of the problem (in the range of 75 to 85).

Internally each of the expressions are represented as threes. This structure is used as a basis of applying genetic operators: crossover and mutation. A random point in selected expression (tree) is chosen as crossover or mutation point. More details will be given about this later. The typical structure of an expression would look like as in Figure 1.

```
The expression:

    (*I1* - ((*I2* + 1) * 0))

The tree representation:
```

$$\begin{tikzcd}
 & - \arrow[dl] \arrow[dr] & & \\
I1 & & * \arrow[dl] \arrow[dr] & \\
 & + \arrow[dl] \arrow[dr] & & 0 \\
I2 & & 1 &
\end{tikzcd}$$

Figure 1: Tree representation of an expression.

In order to balance the behavior of the expressions (i.e. the bias toward positive expressions) half of the expressions are given a minus sign in front of them. This is to achieve, potentially an equal chance of producing negative and positive values when generating the expressions in the initial population.

2.2 Genetic Algorithms

The Schema Theorem developed by Holland[12] based on genetic search has been proven to be useful in many applications involving large, complex and deceptive search spaces [7]. So genetic search is most likely to allow fast, robust evolution of genotypes encoding for potential learning rules as mathematical expressions. Using Genetic Algorithms (GA) the model is implemented in LISP. The top level structure of the system exhibit the following:

1. Initialize the population of expressions
2. Evaluate each expression and determine its fitness
3. Select expressions to reproduce more
4. Apply genetic operators to create new population
5. If the solution found or sufficient number of generations are created then stop; if not go to 2.

The initialization technique is randomly generating mathematical expressions. This introduces the least amount of domain specific knowledge into the initial population through the variables used in the expressions. Unlike Koza's genetic programming applied to particular problems there are no domain specific functions. Only three mathematical functions are allowed; addition, subtraction and multiplication.

In the following sections, I will describe the rest of the steps in applying GA.

2.2.1 Evaluation

In order to provide a basis to determine the fitness of the expressions, each generation the expressions are evaluated using Lisp's "EVAL" statement by instantiating input values for each of the patterns from the training set.

The fitness of an expression is based on its success in learning a specific task. Since the target outputs are in the range of 0 to 1, the values, once obtained after the evaluation of the expressions, are mapped to values between 0 and 1 by using a squashing function. Several functions have been tested in this mapping including logistic activation function used by [20]. One of the functions which showed the most success, especially in mapping to binary target outputs, was the following:

```
if value > 1 return 1
if value < 0 return 0
otherwise return the value
```

The fitness (success) of the individual expression is computed by testing them on all training patterns, and dividing the total error by the number of patterns, subtracting from 1 and multiplying by 100 yielding a fitness percentage between 0 and 100.

The expressions are ranked after each generation according to their success. Those who are higher in the rank (higher scoring ones) are said to be most fitting expressions.

2.2.2 Selection

The purpose of selection in GA is to give better opportunity of reproducing to those members of the population which shows better fitness. For the model this means to select those expressions with higher scores (beginning part of the rank) and give them more chance to reproduce.

In the model, parent selection technique for reproduction is normalizing by using an exponential function taken from Whitley's [23] rank-based selection technique. The function generates integer numbers from 1 to population size. The generation of numbers exhibits characteristics of a non-linear function where there is more tendency to produce smaller numbers (since higher scoring expressions are on top of the rank).

The function is $Z = X - \sqrt{}$

binary tree representations. In order to choose a point in this tree two different probabilities are used. One probability determines whether we want to go to the left or right branches of the tree and the other determines whether to go down more or to stay at that level. Figure 2 shows systematically how these are implemented in the system.

```
if mutate
   then create new expression
else
   if at end node of either tree
      or probability-down > cutoff
      then swap parts of trees
   else
      if probability-left > cutoff
         then go down on the left branch
              and recurse
      else
         go down on the right branch
         and recurse
```

Figure 2: Crossover and mutation algorithm.

When the point is chosen, the next thing to decide is whether there will be a mutation. If there will be a mutation on both of the trees at that point a new expression is added. Otherwise the parts of the trees side apart from that point are swapped.

3 Results

In the preliminary experiments of the model, the emphasis has been on simplicity to keep the computational requirements manageable whilst encouraging the achievement of meaningful results.

The model is applied to several supervised learning tasks involving varying difficulty of input-output mappings. The aim of the experiments is to observe whether a learning rule would emerge during the course of evolution for a specified task. First experiments involved OR, AND and XOR problems.

algorithms for these experiments were 10 percent mutation rate and 50 percent crossover rate. The success of the ect.rate.

learn to recognise a single specific pattern. The other tasks are more complex.

```
      -
      ((*I2* * *I3*) + (*I4* + *I3*)))
      -
      ((((1 - *I1*) - (*I5* + *I5*)) + ((1 - *I5*) - (*I2* + *I3*)))
      +
      (0 + (*I3* - *I3*))))
      -
      (((0 + *I5*) + (0 * *I4*)) - (*I5* - *I4*)))

   Success:  92 percent

   2. ((*I2* * *I5*)

   Success:  92 percent


   3. (((((*I4* + *I2*) - ((*I2* * *I1*) + (*I5* + *I2*)))
      +
      (*I5* * *I3*)) + (((((((*I1* - *I4*) - (0 - *I5*))
      +
      (*I5* - *I4*)) + (*I2* + *I3*)) + (*I3* * *I5*))
      -
      ((*I2* - *I5*) - (((0 - *I4*) + (0 - *I1*))
      +
      (*I4* * *I3*)))) - (*I3* - *I5*)))

   Success:  100 percent

   4. ((*I2* + (*I5* - *I2*))

   Success:  100 percent

TASK 2

   1. ((((*I4* - *I4*) - ((*I5* + *I2*) + ((*I1* + *I1*)
      +
      (*I5* - *I3*)))) + (*I3* - *I2*))
      *
      (((*I4* - 0.056697) * ((*I2* + *I2*) + (*I4* + 0.991166)))
      *
      (*I2* - *I5*)))

   Success:  100 percent
   2. (((*I1* - 0.49447) - ((*I4* + *I5*) - (*I2* + 0.588741)))
      -
      (*I2* + *I5*))

   3. ((((((*I1* - *I5*) + (*I3* + *I4*))
      -
      ((*I4* - 0.827728) + ((*I2* - *I5*) + (*I4* + 0.828348))))
      *
      (((*I3* - *I3*) + (*I1* - 0.416933))
      *
      ((*I2* + 0.128969) * (*I2* - 0.160986))))
      -
      (*I1* - *I5*))
      *
      (((*I3* - 0.260095) + (*I5* - *I2*)) + (*I1* - *I2*)))

   Success:  100 percent

   4. ((((*I4* - 0.354676) * (*I2* + 0.183204)) - 0.050929)
      *
      (((*I5* + 0.614033) * ((*I5* - 0.366376) * (*I5* + 0.14586)))
      -
      (*I2* + *I1*)))

   Success:  100 percent


TASK 3

   1. (((((1 - *I3*) + (1 + *I2*)) + ((*I4* + *I2*)
      +
      (*I1* - *I4*))) - (*I4* + *I2*)) - (*I2* + *I1*))

   Success:  100 percent

   2. (((((((*I1* - *I2*) - ((*I4* * *I5*) + (0 - *I2*)))
      +
      ((1 - *I5*) - ((*I3* - *I1*) - ((0 * *I2*)
      +
      (*I1* - *I3*))))) + (((*I1* * *I5*)
      -
      ((((*I1* - *I5*) + (((0 + *I4*) - (1 * *I1*)) + (1 * *I4*)))
      +
      (1 + *I5*)) + (0 - *I1*))) + (1 + *I2*)))
   +
   (*I2* - *I4*)) + (0 - *I5*))

Success:  92  percent

3. ((((((*I4* - *I2*) - ((*I3* + *I3*) - (1 + *I2*)))
    -
   ((*I5* * *I4*) + (0 * *I1*)))
   -
   (*I4* * *I3*)) + (*I1* + *I5*))

Success:  92  percent
```

In order to test how robust is the evolved learning rules for a given task, the rules have also been tested on unseen exemplars. For example, of all the tasks represented here, task four is the most difficult one. When tested on unseen patterns, the success of all of the four learning rules for this task remained at 100 percent.

```
TASK 4

   1. (*I1* + ((*I4* + *I2*) + (*I5* - 0.893335)))

   Success:  100 percent


   2. ((*I1* + *I4*) * ((((*I4* + 0.350897) * (*I3* + 0.761547))
      *
      (((*I3* + 0.299322)+ ((((*I2* + *I1*) * (*I4* + *I4*))
      -
      (*I4* + *I2*)) - (*I3* + *I4*)))
      +
      (*I2* + *I2*))) + *I1*))

   Success:  100 percent


   3. (((*I1* + *I1*) - 0.781905)
      +
   (((*I2* + *I5*)
   *
   (((*I2* + 0.059971) * (*I3* + 0.113237)) * (*I2* - 0.331763)))
   +
   (((*I4* - 0.148709) * (*I1* + *I2*)) * (*I1* + *I4*))))


Success:  100 percent

4. ((*I1* - *I1*) - (((((*I4* + *I4*) - (*I4* - *I2*))
   *
   (((*I1* + *I1*) - (*I1* + *I4*)) + (*I4* + 0.28141)))
   -
   (*I1* + *I1*)) - (((*I4* + *I3*) * ((*I3* + *I5*)
   +
   ((*I5* + *I3*) * (*I2* - *I4*)))) * (*I4* + 0.30423))))


Success:  100 percent
```

3.3 Parity Problems

A difficult task for many learning algorithms are parity problems [20] where the required output is 1 if the input pattern contains an odd number of 1's; otherwise it is 0. This is a hard learning problem because very similar patterns (even different with one bit) may require completely different output. The XOR problem is one of the parity problems with size two. Although it took longer, expressions successfully evolved to code solution to XOR problem. For three bit parity problem it was difficult to evolve a learning rule with 100 percent fitness by using a population size of 30 (on the average 2 out of 10 runs would produce it). It would take around 15 generation to evolve a learning rule with 87.5 percent success (unsuccessful only on 1 out of 8 training pairs). Ninety percent of the times a solution found for the three bit parity problem with a minimum success level of 87.5 percent. The reason for this low level of performance is probably due to small population size and insufficient number of generations as compared to difficulty of the problem. However, it is clear that the encoding schema is capable of coding and finding a solution for this difficult task though the encoding schema involves as minimal as possible prior knowledge with respect to possible solutions of parity problems.

```
THREE BIT PARITY PROBLEM


  1. ((*I1* + ((((*I1* + *I2*) - (*I3* + *I1*)) + (*I2* - *I3*))
      *
     ((*I1* - 0.427487) * ((*I1* - *I2*)
      -
     ((*I2* - 0.526565) + (*I1* - 0.69849))))))

  Success: 100 percent


  2. ((*I1* - 0.306492)
     +
     (((((((*I2* + *I2*) - ((*I2* - *I2*) + (*I2* + *I2*)))
     *
     ((*I3* + *I1*) * ((*I1* - *I2*) + ((*I2* - 0.457992)
     +
     (((*I3* + 0.731704) * (*I2* + 0.775932))
     -
     ((*I3* - 0.70112) - (*I2* + *I2*)))))))
     +
     (*I3* - *I2*)) + (*I1* - *I2*)) * ((*I2* - *I3*)
     *
     ((*I3* - *I3*) - ((*I3* - *I3*) - (((*I2* - 0.773618)
     +
     (*I1* - 0.415343)) + (*I1* - 0.164109)))))) + 0.530177))

  Success: 100 percent
```

The above solution to parity problems exhibit a complex language. For the moment it is sufficient to observe that encoding schema is able to produce a solution for such difficult problems.

4 Conclusions

The experiments in this paper have shown that an encoding schema involving random expressions observed that using this encoding, evolution can provide a necessary basis for the learning rules to emerge, although any domain specific knowledge in coding the possible solutions are minimal. However, the research using this encoding is at its development stage and there are several issues that must be improved before going further.

As it has been shown in the previous section, it is possible that evolution can result in several different learning rules for the same task in hand. These solutions are sometimes very similar to each other, but are sometimes a totally different way of expressing the same solution. This provides promising evidence that the encoding schema would also be useful for more complex and larger problems since it can always find an alternative expression for a learning rule which is difficult to express and discover. Normally, we would like to have at least near optimal solutions without any redundant subexpressions. This can be accomplished by either (1) starting from simple elementary expressions and building up gradually or (2) eliminating redundant subexpressions from the final solutions. In the next experiments these issues will be of major concern.

The experiments shown here have been kept simple in order to show important characteristics of encoding schema and observe the evolution of learning rules. Only supervised learning tasks have been tested. Although experimenting with other learning methods is essential, the motivation of choosing supervised learning is directly related to the future aims of the research.

In the future, the encoding schema will be used to solve some hard supervised learning problems. These problems have been shown to be difficult to solve using conventional learning algorithms (i.e. back-propagation) due to the fact that the rule of learning contained in the target that fact

target retinvalsoccd0.67(er99.4(a)-14999.4(n)Tj17.j39.839800Td[(n)1000.10.96020Tdoat)Tje

[4]

[18]