Selection Pressure and an Efficiency Of Neural Network Architecture Evolving

Halina Kwaśnicka[1], Mariusz Paradowski[1]

Wydzialowy Zaklad Informatyki, Informatyka i Zarzadzanie, Politechnika Wroclawska

Abstract. The success of artificial neural network evolution is determined by many factors. One of these factors is the fitness function used in genetic algorithm. Fitness function determines selection pressure and Therefore influences the direction of evolution. It decides, whether received artificial neural network will be able to fulfill its tasks. Three fitness functions are proposed and examined in the paper, every one of them gives different selection pressure. Comparison and discussion of evolution results for every function is made.

1 Introduction

For a long time, GAs and NNs - both proposed as the imitation of natural process, biological evolution and real neural systems respectively - have been developed separately. During last decade attempts to combine these two approaches are observed [**?**]. The papers describing attempts to combine the two above mentioned techniques started to appear in the late 1980's [**?**,**?**]. GAs can be used to optimize NNs in different way: to optimization of the number of layers, the number of neurons or connections [**?**]; to settle of connection weights instead of use a training algorithm [**?**]; to select of optimal values of adjustable parameters associated with a model of NNs [**?**]; and to design whole structures of networks [**?**,**?**].

There exist some reasonable methods for selection of suitable training algorithm and evaluation of trained networks, but the design of NNs architecture is still rather a matter of art, not the routinized task. A designer of NN must rely on his experience and on the informal heuristics arising from the works of others researchers [**?**].

We can distinguish two types of non-evolutionary approaches of designing ANN architecture - *constructive* and *destructive* . Constructive methods start from a neural network made of small number of nodes and add new nodes until expected effect is achieved. Destructive methods do the opposite. They start from a network made of large number of nodes and remove existing nodes until expected effect is achieved. Features of the domain of possible architectures cause these methods not to give satisfying results [**?**].

The smaller the network is, the larger capability to generalize it has, because of large networks have enough capacity to memorize all presented examples during training process. Therefore, developing a neural network for a given task we

tray to reduce a network size as much as possible, taking into account learning and generalizing possibilities. Genetic Algorithm (GA) is a global optimum search heuristic that does not require any additional information about the optimized function, gives a clear advantage over other methods. It can be used as a tool for searching 'good' architecture of designed network.

Presented paper is focused on the examination of selection pressure on effectiveness of developing ANN's architectures able to solve classification problems to which they were designed. We use Backpropagation training method, it is a gradient method and tends to find only local minima. It is crucial for success to pick proper training parameters. Usage of incorrect parameters can cause neural networks training process to fail. The division of data set into training and test sets is another crucial problem. It is important to have all classes of decision space in training and test sets. The algorithm of evolution of artificial neural network architectures is based on the algorithm schema used in GA [**?**]. The additional element it uses is the training and testing mechanism used during fitness function evaluation [**?**]. The separation between architecture evolution and neural network training is the advantage of this solution. Very large computation time makes the separation very important. From a technical point of view it allows to distribute training process on many independent computers.

2 Genotype and genetic operators

We use a genotype with direct encoding. Genotype contains two elements: vector V and matrix E. Vector V is used to store information about nodes, E is used to store information about edges. Both V and E contain only binary information.

$$V = [\, v_1 \quad v_2 \quad \dots \quad v_n \,] \quad ; \quad E = \begin{bmatrix} e_{11} & e_{12} & \dots & e_{1n} \\ e_{21} & \ddots & & \vdots \\ \vdots & & \ddots & \vdots \\ e_{n1} & \dots & \dots & e_{nn} \end{bmatrix} \tag{1}$$

Where n is an assumed maximal number of nodes (including input and output neurons), $v_i = 1$ denotes that i-th node is active and $v_i = 0$ – i-th node is inactive; $e_{ij} = 1$ means that output of i-th node is connected to input of j-th node and $e_{ij} = 0$ – a lack of such connection.

For feedforward artificial neural networks only a half of matrix E is used. The lower-left part of the matrix is irrelevant, but the genotype is designed to handle recurrent network as well, in which every cell of the matrix is used. From the algorithmic point of view vector V and matrix E can represent a directional graph, in which edges and vertices can be added and removed. The biggest advantage of direct encoding is its ability to evolve every neural network architecture. This makes direct encoding suitable for such tasks as: artificial neural network pruning or elimination of unnecessary inputs. The biggest disadvantage of direct encoding is its memory complexity. For k genotypes and maximum number of nodes set to n, memory complexity equals $O(kn^2)$.

Two types of mutation (*Node* and *Edge mutation*) and one type of single-point crossover are used. Let us denote: n - maximum number of nodes; p_{va} – probability of adding a node; p_{vr} – probability of removing a node; p_{ea} – probability of adding an edge; p_{er} – probability of removing an edge; x – split point used in V and E; y – secondary split point used in E.

Upper indices ^{1} and ^{2} – the first and the second parent genotype, respectively.

Node mutation

```
For every node v[i] in V:
  r := random number between <0; 1>, uniform distribution;
  if r <= pva then v[i] := 1;
  if pva < r <= pva + pnr then v[i] := 0;
```

Setting one v_i to 1 can increase number of edges in phenotype in the range $[0, n]$, similarly, setting to zero decreases number of edges.

Edge mutation

```
For every edge e[i,j] in E:
  r := random number between <0; 1>, uniform distribution;
  if r <= pea then e[i,j] := 1;
  if pea < r <= pea + per then e[i,j] := 0;
```

Single-point crossover

$$v_i = \begin{cases} v_i^1 & if \quad i \leq x \\ v_i^2 & otherwise \end{cases} \quad ; \quad e_{ij} = \begin{cases} e_{ij}^1 & if \quad (i < x) \vee ((i = x) \wedge (j \leq y)) \\ e_{ij}^2 & otherwise \end{cases}$$

All nodes with indices lower than x and all their input edges are taken from the first parent. For node with index x all input edges from nodes lower or equal to y are taken from the first parent. Otherwise all edges are taken from the second parent.

3 Evaluation functions

Three fitness functions with different selection pressure are proposed and examined, each is a combination of two simpler fitness functions – training fitness function f_t, and two size fitness functions f_s^1, f_s^2.

Functions f_s^1 and f_s^2 are based only on the number of edges in phenotype. Number of nodes is not used directly, but it has a very strong indirect influence on this fitness function. Number of nodes has strong direct influence on number of edges, as was stated in section 3.

3.1 Training fitness function

The following denotations are used: n_o – a number of output nodes; k – a number of test examples used in evaluation; m – a number of training-testing processes; e_i – an error for i-th test example; e – an error for all test examples; f_t^{ij} – a

training fitness function for j-th process of training-testing, for i-th division of the problem set; f_t^i – a training fitness function for i-th division of the problem set.

$$e_i = 0.5 \sum_{y=1}^{n_o} (o_y - t_y)^2 \quad ; \quad e = \sum_{i=1}^{k} e_i$$

$$f_t^{ji} = \left(\frac{max(2n_o k - e, 0)}{2n_o k} \right)^4 \quad ; \quad f_t^i = max(f_t^{1i}, f_t^{2i}, \ldots, f_t^{mi}) \quad ; \quad f_t = \frac{1}{s} \sum_{i=1}^{s} f_t^i$$

Training and testing process is repeated m times for every problem set division. The maximum training fitness function from all m testing processes is taken. This allows to test whether the neural network is able to provide correct answers for as much test examples as possible. For $m = 1$ the noise in results, caused by Backpropagation training method and its tendency to stuck in local minima, is too strong and often makes evolution impossible. For higher values of m (e.g. $m = 3$) computation time increases m times, but results are much more satisfying. All fitness function f_t^{ji} values are in range $[0, 1]$. The higher the fitness function is, the lower is the penalty for the current phenotype. Maximum acceptable error is defined. Its value is set to 0.01 maximum acceptable error e equals 1% of the maximum possible error. Term to pass training test is used in this paper. An artificial neural network passes training test when it has $f_t > 0.99^4$. Value 0.99^4 is referred as mf_t in next sections of the paper.

3.2 Size fitness functions

s size of the population; r_i number of edges in i-th phenotype; s^p number of phenotypes that passed training test; r_i^p number of edges in i-th phenotype if training test passed, 0 elsewhere.

$$r^1 = 1 + \frac{\sum_{i=1}^{s} r_i}{s} \quad ; \quad r^2 = 1 + \frac{\sum_{i=1}^{s} r_i^p}{s^p}$$

$$f_s^1 = \frac{max(0, r^1 - r_i)}{max(r_1, r_2, \ldots, r_s)} \quad ; \quad f_s^2 = \frac{max(0, r^2 - r_i^p)}{max(r_1^p, r_2^p, \ldots, r_s^p)}$$

Size fitness function f_s^1 and f_s^2 causes selection pressure dependent only on the size of neural networks in current generation. In function f_s^1 all phenotypes larger than the average size value of the whole population have fitness function equal to 0, the smallest phenotype is given the fitness function equal to 1. In function f_s^2 the average value is not taken from the whole population but only from phenotypes that have passed training test. This makes selection pressure for finding smaller solutions higher than in f_s^1.

3.3 Fitness functions

$$f_1 = \begin{cases} f_t(1 + f_s^1) & if \quad f_t > mf_t \\ f_t & if \quad f_t \leq mf_t \end{cases} \tag{2}$$

ANNs with architecture capable of solving stated problems are preferred. The main purpose is to evolve ANNs that answers to test examples correctly, the secondary purpose is to minimize the size of artificial neural network.

$$f_2 = \begin{cases} f_t(1 + f_s^2) & if \quad f_t > mf_t \\ f_t & if \quad f_t \leq mf_t \end{cases} \tag{3}$$

Usage of f_s^2 size fitness function causes higher than in f_1 selection pressure to minimize phenotypes that passes training test. Artificial neural networks that have not passed training test have smaller than in f_1 chance to reproduce.

$$f_3 = f_t(1 + f_s^1) \tag{4}$$

Every phenotype is rewarded for its size. Selection pressure on phenotypes that have very good results during testing is reduced. Both purposes - to minimize size and error - have similar priorities.

4 Results

Presented approach was tested on 6 benchmark problems: *Exclusive-OR*, *Thermometer*, *Parity of a number*, *ZOO*[**?**], *IRIS*[**?**], *HOUSING*[**?**].

Fig. 1. *ZOO* - Training fitness function and number of edges

Fig. 2. *IRIS* - Training fitness function and number of edges

The importance of the maximum acceptable error has been shown. This factor is responsible for two behaviours :

1. finding an accpetable solution at the beginning of evolution
2. disalowing to find too small networks which are unable to learn properly

In case of Exclusive-OR and Parity of a number the f_3 function gives better results much quicker than function f_1 and f_2. In case of larger problems, function f_3 almost in every case minimizes the network too much and in result finds unacceptable solutions. Often, there is no selection pressure to find solutions

that are better than average during testing. At the beginning of evolution, when all solutions tend to have average results during testing, there is a very small chance of appearance solutions which will increase the average training fitness function for the whole population. Functions f_1 and f_2 force evolution to find neural networks that pass the training test at the very beginning. There is no room for minimalization of size until an acceptable solution is found. This causes the GA to sweep the domain for optimal solutions for only one criterion. When proper networks are found the second criterion introduces the minimalization process, which is slower than in f_3 and forces the minimalization process to abort when created networks are unable to learn due its too small size. The efficiency and importance of mentioned behaviors is shown by evolution with use of function f_2, in which minimalization criterion is much stronger than in f_1. Despite the strength of minimalization criterion in f_2 evolution is able to stop it when generating smaller networks would result in receiving networks with smaller abilities to solve stated problem.

References

1. BALAKRISHNAN K., HONAVAR V.: Evolutionary Design of Neural Architectures - A Preliminary Taxonomy and Giude to Literature, 1995.
2. BALAKRISHNAN K., HONAVAR V.: Properties of Genetic Representations of Neural Architectures.
3. BORNHOLDT S., GRAUDENZ D.: General Asymmetric Neural Networks and Structure Design by Genetic Algorithms, Neural Networks, Vol.5, pp. 327-334, 1992.
4. GOLDBERG D.E.: Algorytmy genetyczne i ich zastosowania.
5. HARP S.A., SAMAD T., GUHA A.: Towards the genetic synthesis of Neural Networks, Proceeding of the third International Conference on Genetic Algorithms, Morgan Kaufmann Publishers, Inc., San Mateo, California, 1989.
6. MEDSKER L. R.: Hybrid Intelligent Systems, Kluver Academic Publishers, Boston/Dordrecht/London, 1995.
7. MONTANA J.D., DAVIS L.: Training Feedforward Neural Networks Using Genetic Algorithms, BBN Systems and Technologies Corp., Cambridge, 1989.
8. MURRAY D. Tuning Neural Networks with Genetic Algorithms, AI Expert June 1994.
9. PERCHELT L.: Proben1 - A Set of Neural Network Benchmark Problems and Benchmarking Rules, Technical Report 21/94.
10. SCHAFER J.D., WHITLEY D., ESHELMAN L.J.Combinations of Genetic Algorithms and Neural Networks: A Survey of the State International Workshop on Combinations of Genetic Algorithms and neural Networks, Baltimore, Maryland 1992.
11. TADEUSIEWICZ R.: Sieci neuronowe, Akademicka Oficyna Wydawnicza RM, 1993.
12. WHITLEY D.: Genetic Algorithms and Neural Networks, Generic Algorithms in Engineering and Computer Science, 1995.
13. YAO X.: Evolving Artificial Neural Networks, 1999.