Clonal Selection-based Neural Classifier

A. Lanaridis, V. Karakasis, *Member IEEE*, and A. Stafylopatis, *Member IEEE*
School of Electrical and Computer Engineering, National Technical University of Athens
aristeides@gmail.com, bkk@cslab.ece.ntua.gr, andreas@cs.ntua.gr

Abstract

Artificial Immune Systems (AIS) constitute an emerging and promising field, and have been applied to pattern recognition and classification tasks to a limited extent so far. This work is a first attempt of applying the clonal selection principle to the training of Multi-Layer Perceptrons (MLPs). The Clonal Selection-based Neural Classifier (CSNC) uses the basic concepts of clonal selection to evolve MLPs, which are represented as real-valued linear antibodies. The proposed system is actually a multi-classifier, consisting of multiple sets of MLPs, each one devoted to the recognition of a different class of the input data. The final trained classifier is comprised of the best MLPs from each set. The proposed classifier is tested against a set of benchmark problems and yields promising results.

1. Introduction

Quite recently, the human immune system and the mechanisms it uses to protect the human body from invaders, has raised an intense interest from the community of artificial intelligence. The reason behind this is that the natural immune system incorporates perfectly a variety of different artificial intelligence techniques, such as pattern recognition through a network of collaborating agents (e.g., immune network of B-cells), adaptive and associative learning through memory (e.g., memory B-cells), and an advanced selection mechanism of the best B-cells, that mimics the natural selection mechanism, which is found in other biological phenomena, such as natural selection and physical evolution.

In this work, we apply one of the most established techniques of artificial immune systems, the clonal selection principle, to the training of MLPs. We implement a variation of the well-known CLONALG algorithm [2] by removing some features, which do not offer any benefit to MLP training, and by adding some new ones that fit better to the problem at hand. The original CLONALG algorithm maintains two sets of cells: a set of memory cells, which are used to recognize new antigens, and a set of non-memory cells, which actually contribute to the evolution and amelioration of the memory cells. Originally, CLONALG used a single memory cell for each different pattern presented. In our implementation, we eliminate the non-memory cells, since we found that they offered little benefit to the final training accuracy. Instead, we use a "wide" memory, where more than one memory cell are used for each pattern or class of input data, thus allowing us to train multiple MLPs in parallel for each class of the problem at hand. The rationale behind this decision is that a single MLP configuration (number of layers and hidden nodes) is not always appropriate for every class of the problem, thus leading to lower overall accuracy. Another important aspect of our implementation is the representation of the cells, for which we use real-valued vectors, instead of the commonly used bit-strings.

The rest of the paper is structured as follows. Section 2 presents a brief overview of the previous work on the field, Section 3 provides an overview of the clonal selection principle from a biological point of view, Section 4 presents the proposed training algorithm, and Section 5 describes the application of the algorithm on the training of neural classifiers. Section 6 presents the experimental results, and Section 7 concludes the paper.

2. Previous Work

The clonal selection principle was originally proposed by Burnet [1] in 1959, in order to explain the reinforcement learning of the immune system of mammals. According to Burnet's theory, mammals acquire immunity through mutation, selection, and proliferation of the mature B-lymphocytes. The clonal selection principle quickly attracted the attention of computer

scientists, since it appeared as a more flexible alternative to genetic and evolutionary algorithms. A first attempt to model the clonal selection principle was performed by Weinard [11], though from a more biological point of view. Fukuda et al. [3] presented an abstraction of the principle and applied it to computational problems. However, the most influential implementation of the clonal selection principle was the CLONALG algorithm of de Castro and Von Zuben [2]. CLONALG is not hard to implement and uses the basic concepts of clonal theory in a consistent manner. An important feature of the CLONALG algorithm is that the mutation and the cloning of antibodies depend on their affinity or fitness. An improvement of the original CLONALG algorithm is the CLONCLAS [12] algorithm of White and Garrett, and the algorithm of Nicosia et al. [8], that uses a probabilistic half-life of B-cells and a termination criterion based on information theory. A more sophisticated approach of the clonal selection principle is the AIRS [10] supervised learning system, which also incorporates concepts of immune network theory. Finally, a generalization of the CLONALG algorithm is the Enhanced Clonal Algorithm (ECA) [4] [5], which adds a receptor editing step to the original CLONALG algorithm and defines a more sophisticated management of the memory pool. In addition, the cloning of B-cells is more dynamic. ECA was applied to pattern recognition and data mining tasks in [5] and yielded promising results, but its configuration may become a tedious task, since it allows a bunch of parameters to be set.

The efficient training of neural networks has been extensively studied in the past, and a large number of different techniques have been proposed. Among these techniques, attempts have been made to use evolutionary algorithms in the training of neural network classifiers. These techniques can be divided into two categories: (a) techniques that use evolutionary algorithms to evolve the structure of the neural network, but still use Back Propagation (BP) for the adaptation of the weights, and (b) techniques that employ evolutionary algorithms for tuning both the structure and the weights of the neural network.

3. The Clonal Selection Principle

The clonal selection principle (Figure 1) describes the mechanism used by the natural immune system to react to foreign antigens. It was proposed by Burnet [1] and established the idea that the lymphocytes which better recognize a foreign antigen have a higher probability of reproduction. In particular, the clonal selection theory establishes the following fundamental concepts:

1) The new cells are clones of their parents and are subjected to somatic mutations with high rate (hyper-mutation).
2) The new cells that recognize self cells are eliminated.
3) The mature cells are proliferated and differentiated according to their stimulation by the foreign antigen.

According to clonal theory, the cells that bind to the foreign antigen start proliferating through cell cloning and undergo mutation in order to differentiate. The number of clones that each activated B-cell will produce depends directly on its level of activation, i.e., the quality of its binding to the antigen. The better the binding is, the more the clones that will be produced. The opposite is true for the mutation rate, i.e., clones with good affinity are mutated with lower rates. This fact allows for the fast adaptation of the immune response, since low affinity B-cells are differentiated through high mutation rates, which will probably lead to cells with better affinities. However, this differentiation process may lead to B-cells with self-reactive receptors or B-cells with very poor affinities, but these cells are effectively eliminated. The elimination of poor quality B-cells and the affinity-proportionate cloning of B-cells reminds of the natural selection process, where only the best individuals are selected to reproduce. The clonal selection process, however, is more elaborate and more flexible.

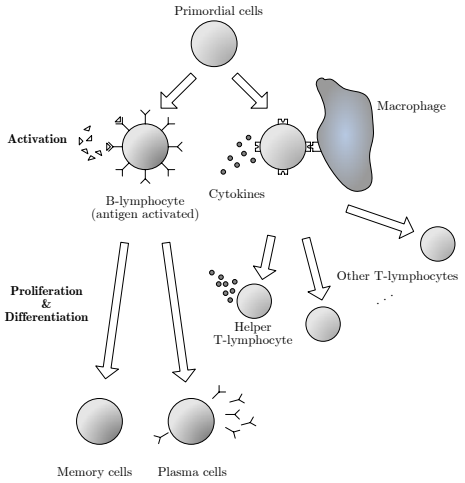


Figure 1. The clonal selection principle and the affinity maturation process.

4. The training algorithm

The clonal algorithm used for training the neural classifier is inspired by the CLONALG [2] algorithm, but a number of its features have been modified, in order to better fit to the problem in question. Specifically, the antibody pool of non-memory cells was removed, and multiple memory cells for each input class were used instead. The quality of memory cells proved to be much superior to that of the non-memory cells right after the first few generations, thus, the memory cells were always selected to reproduce, and the non-memory cells remained inactive. On the contrary, we used multiple memory cells for each class of the problem, each one representing an MLP of different structure, since a fixed-structure MLP may not fit all the problem classes equally well.

The steps of the algorithm used are described below:

1) All antibodies are initialized randomly using a normal distribution in the [0,1] interval.
2) Each antibody is presented with the whole training set.
3) The affinity of each antibody to its respective class is calculated.
4) A set of clones is created for every antibody.
5) The clones undergo a hyper-mutation process.
6) The affinity of each of the mutated clones to its respective class is calculated again.
7) For each set of mutated clones, the clone with the highest affinity is selected.
8) The affinity of the best mutated clone of each set is compared to the affinity of the original (parent) antibody. If the clone is better in term of its affinity, it replaces the original antibody.

Steps 2–8 are repeated for each training generation. After the training is complete, the best antibody of each set is selected to build the final classifier.

5. Application to classification problems

The algorithm presented above implements a generalized evolutionary process, in which antibodies represent abstract entities, which can be cloned, mutated, evaluated, and selected in each generation. The actual use of the algorithm and its efficiency, however, largely depend upon the exact entities represented by these antibodies. The nature of the data used to represent the antibodies (e.g., bit strings, real-valued vectors, etc.) designates the choice of genetic operators to be applied, while the represented entity determines the evaluation criterion to be used.

In the present paper, the algorithm is applied to a common classification problem, and the antibodies represent neural networks. An important characteristic of the immune algorithm is that certain antibodies correspond to certain patterns or classes of the dataset. Thus, instead of implementing a single neural classifier, the system implements a multi-classifier system, consisting of a single-output MLP for each class of the dataset. The output of each such classifier can be either 1 or −1, depending on whether the pattern in question belongs to the specific class of data or not, respectively. The multi-classifier assigns a pattern to a specific class if the output of the classifier corresponding to that class is 1 and the outputs of the classifiers corresponding to the remaining classes equal −1.

For each data class, MLPs of different sizes are trained simultaneously and are evaluated at the end of the training process. Therefore, the final multi-classifier may consist of classifiers of various sizes, different for each class of the data.

The weights and biases of the MLPs are coded into real-valued antibodies. The mutation process uses the non-uniform mutation operator [6]. Finally, the evaluation of the trained MLPs is based on the least square error metric. In the following, these concepts are further explained.

5.1. Antibody representation

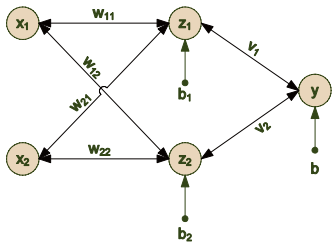
An antibody is a real-valued vector, representing a neural network with an input layer of n nodes, a hidden layer of p nodes, and one output node. The MLP used consists of $n \cdot p$ weights connecting the input to the hidden layer, p hidden layer biases, p weights connecting the hidden layer to the output, and the bias of the output node. Hence, the length of the resulting vector will be

$$l = (n+2)\,p + 1$$

Let x_i be the value of the i-th input node, z_j the value of the j-th hidden node, y the output, w_{ij} the weight connecting the i-th input with the j-th rule node, v_j the weight connecting the j-th hidden node with the output node, b_j the bias of the j-th hidden node, and b the bias of the output node. If Ab is the vector representing the antibody, then

$$w_{ij} = Ab_{(i-1)p+j}$$
$$b_j = Ab_{np+j}$$
$$v_j = Ab_{(n+1)p+j}$$
$$b = Ab_i$$

Figure 2 illustrates an example of the representation and the encoding discussed.



(a)

$$Ab = \{w_{11}, w_{12}, w_{21}, w_{22}, b_1, b_2, v_1, v_2, b\}$$

(b)

Figure 2. Encoding (b) of a MLP (a) into an antibody.

The activation function used to calculate the values of the hidden and output layer nodes is the logistic function:

$$f(x) = \frac{1}{1+e^{-x}}$$

The values of the hidden nodes are computed by the following formula:

$$z_j = f\left(\sum_{i=1}^{n} w_{ij} x_i - b_j\right)$$

Finally, the output of the network is then given by the equation below:

$$y = f\left(\sum_{j=1}^{p} v_j z_j - b\right)$$

5.2 Mutation process

During their lifetime, the antibodies undergo an evolutionary process, which is capable of producing offspring of greater fitness than that of the parent antibodies. In general, the evolution process is implemented by applying genetic operators to every antibody. The most frequently used operators are variations of (a) crossover and (b) mutation.

In the particular problem, the crossover operators were tried, but did not produce satisfying results. This is mainly due to the nature of the network representation. Since the antibodies represent networks devoted to recognizing different classes of data, they evolve into very dissimilar real-valued vectors after the first few generations. Consequently, a crossover between such distant vectors cannot produce new antibodies of high affinity. For this reason, we focused on the mutation operators. Among the common mutation operators, the non-uniform operator produced the best results. The idea behind the non-uniform mutation process is that we start mutating antibodies to a large extent, and as the training proceeds and the affinity of the antibodies increases, the extent of the mutation process decreases. Thus, in essence, the lower the quality of an antibody, the larger the extent of the modification it is subjected to. The formal definition of the operator will be discussed next.

The non-uniform mutation operator

Let $x = \{x_1, x_2, \ldots, x_n\}$ be the vector on which the operator will be applied, where $x_i \in [x_{imin}, x_{imax}], x \in \mathbb{R}$. The mutation operator assigns to x_i the value

$$x_i = \begin{cases} x_i + \Delta(t, x_{imax} - x_i), & Z \geq 0.5 \\ x_i + \Delta(t, x_i - x_{imin}), & Z \leq 0.5 \end{cases}.$$

In the above equation, Z is a uniformly distributed random variable in the interval [0,1] and $\Delta(t,a)$ is defined as

$$\Delta(t,a) = a\left(1 - Z^{(1-t)^b}\right),$$

where a is the mutation range, t is the percentage of the training that has been completed, that is,

$$t = \frac{\text{number of current generation}}{\text{total number of generations}},$$

and b is a system parameter, which determines the degree of dependency on the value of t.

$\Delta(t,a)$ returns a value in the range [0,1], but the probability of this value being close to zero increases as t increases. This property causes the operator to initially search the space uniformly, but very locally at later stages by generating a new number close to its predecessor.

The only theoretical issue with the use of this operator is that the actual minimum and maximum values of a neural network weights are not known a priori. In practice, however, this is hardly a problem. Extensive experimentation in the literature has shown that the resulting weights of trained neural classifiers almost always lie in tight intervals. Thus, we can easily overcome the problem by assigning large enough mutation margins to the operator. This does not decrease its ability to search locally, since, as explained before, the probability distribution of the mutation gets narrower with every generation.

5.3 Evaluation criterion

The second important feature of the evolutionary process is the selection of the antibodies with the greatest affinity. Having defined an antibody as a single-output MLP devoted to recognizing a specific class of data, we base our evaluation criterion on the most usual method of evaluating the performance of a neural classifier, the least square error. Let $x_i = \{x_{i1}, x_{i2}, \ldots, x_{in}\}$, $x_{ij} \in \mathbb{R}$ be the i-th pattern, $d_i = \{d_{i1}, d_{i2}, \ldots, d_{im}\}$, $d_{ij} \in \mathbb{R}$ be the desired output for that pattern, and y_i be the actual output of the network for that pattern. The square error for the classifier devoted to recognizing the j-th class is then defined as

$$e = \sum_{i=1}^{N} \left(y_i - d_{ij} \right)^2$$

The error e varies in the interval $[0,+\infty]$; lower values of e denote a network of greater classification accuracy. However, the affinity measure of an antibody should lie in the interval [0,1] and increase as the classification accuracy of the network it represents increases. Therefore, we define the affinity measure as

$$\text{affinity} = \frac{1}{1+e}$$

6. Experimental results

The proposed algorithm was tested on a set of benchmark problems form the UCI repository [7]. We measure the classification accuracy of the trained neural network and compare the results with MGNN [9], AIRS [10], and C4.5. The MGNN algorithm was chosen because it also employs a mutation-based evolution of the neural network weights. We also compared with AIRS, since it is a typical immune-based classifier. On the other hand, we also used C4.5 as a typical example of a classification algorithm. This work is one of the first attempts to apply principles of the immune system to neural network training, and the first results are satisfactory and promising, since they are comparable to well-known classifiers.

In order to evaluate the classification accuracy of the proposed training algorithm, the 5-CV (cross validation) method was used. Specifically, each dataset was split into five equal subsets. In every run of the algorithm, one of the subsets was used as test set, and the remaining four comprised the training set. Before splitting each dataset into subsets, the tuples were shuffled, and any tuples containing missing values were removed. We performed ten runs of the algorithm in total. The differences between each run were negligible, so we present the results of a single 5-CV test. The reason behind this decision is that the number of rules of the evolving network may differ from run to run, although the classification accuracy does not vary significantly.

Table 1 lists the dimension of the input vector, the number of classes, and the number of instances of each dataset.

Table 1. Dataset information

Dataset	*#inputs*	*#classes*	*#instances*
Iris	4	3	150
Cancer	9	2	683
Diabetes	8	2	768
Spiral	2	2	1000
Wine	13	3	178

Regarding the parameters of the algorithm used for the training, the number of clones was set to 100 for each antibody, the range of weights and biases was set to the interval [-50,50], and the dependency of the mutation operator on the iteration number was set to $b = 2$. The total number of training generations and the sizes of MLPs trained for each class were specific to each problem and are listed in Table 2.

Table 2. Training parameters specific to each dataset.

Dataset	*#generations*	*#rules*
Iris	50	2, 4, 8
Cancer	100	1, 2, 4
Diabetes	100	2, 5, 10
Spiral	20	20, 50, 100
Wine	200	4, 12, 20

The average training execution time for neural network training is presented in Table 3. The algorithm was implemented using the MATLAB programming environment, and the tests were run on an Intel Core 2 Quad processor running Windows Vista. It is apparent that the performance of the proposed algorithm in terms of training execution time is very satisfactory. We should note that the presented execution times correspond to the training of multiple MLPs, and much better results can be achieved if a single MLP is trained for each class. For example, the training of the "Iris" and the "Diabetes" datasets using a single MLP with two hidden nodes lasted 1.51*s* and 5.12*s*, respectively, and yielded very satisfactory results.

Table 3. Training times and size of the resulting classifiers per class.

Dataset	*Training time (s)*	*#rules per class*
Iris	10.63	2, 8, 8
Cancer	23.58	1, 1
Diabetes	38.26	4, 2
Spiral	458.65	50, 50
Wine	50.29	12, 12, 20

The classification accuracy results obtained by the proposed training technique in comparison to the classifiers previously mentioned are presented in Table 4. The accuracy values for the other classifiers are those reported in the literature, wherever available. We also present the standard deviation of the accuracy for our algorithm among the different folds of the test set. The classification accuracy of the proposed training technique outperforms the other algorithms in the "Iris" and "Wine" datasets and yields results that are very close to the best performance in the "Cancer" and "Diabetes" datasets. The results obtained for the "Spiral" dataset are considered very satisfactory compared to results found in the literature, but no results were available for the rest of the algorithms in Table 4.

Table 4. Classification accuracy.

Dataset	*CSNC*	*MGNN*	*AIRS*	*C4.5*
Iris	**96.00 ± 2.49**	95.32	**96.00**	93.90
Cancer	96.20 ± 1.69	**96.77**	–	94.70
Diabetes	74.50 ± 3.47	–	74.20	**74.80**
Spiral	95.90 ± 2.08	–	–	–
Wine	**96.01 ± 2.99**	95.32	–	91.60

7. Conclusions and Future Work

The clonal selection principle is responsible for the adaptation and the regulation of the immune response in the natural immune system and has already been successfully applied to a number of pattern recognition and classification tasks. In this paper, we present a first approach to applying the clonal selection principle to the training of a neural multi-classifier. Our training algorithm is inspired by the well-known CLONALG algorithm, but it discards some features that prove to be redundant in the context of neural network training and introduces the concept of an extended memory, where more than one memory cells are simultaneously maintained for each input class. The proposed technique yields very satisfactory results in terms of classification accuracy and appears to be a promising alternative method for training neural classifiers.

In the future, generalization criteria will be examined, in order to neutralize the effect of overfitting inherent to MLPs. In addition, more effective evolution strategies will be examined, in order to avoid local optima in the search space. Finally, we will examine the potential of the proposed technique to the training of different types of pattern recognition tasks.

8. References

[1] F. M. Burnet. *The Clonal selection theory of acquired immunity*. Vanderbilt Univ. Press, Nashville TN, 1959.

[2] L. N de Castro and F. J. Von Zuben. Learning and optimization using the clonal selection principle. *IEEE Transactions on Evolutionary Computation*, 6:239–251, June 2002.

[3] T. Fukuda, K. Mori, and M Tsukiyama. Immune networks using genetic algorithm for adaptive production scheduling. In *15th IFAC World Congress*, 1993.

[4] V. K. Karakasis and A. Stafylopatis. Data mining based on gene expression programming and clonal selection. In *Proceedings of the 2006 IEEE Congress on Evolutionary Computation*, pages 514–521, Vancouver, BC, Canada, 16-21 July 2006. IEEE Press.

[5] V. K. Karakasis and A. Stafylopatis. Efficient Evolution of accurate classification rules using a Combination of Gene Expression Programming and Clonal Selection. *IEEE Transaction on Evolutionary Computation*, 2008 (to appear).

[6] Z. Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs*. Springer-Verlag, Berlin, Germany, 1992.

[7] D. J. Newman, S. Hettich, C. L. Blake, and C. Z. Merz. UCI repository of machine learning databases, 1998.

[8] G. Nicosia, V. Cutello, and M. Pavone. A hybrid immune algorithm with information gain for the graph coloring problem. In *Genetic and Evolutionary Computation Conference (GECCO-2003)* LNCS 2723, pages 171–182, Chicago, Illinois, USA, 2003.

[9] P. P. Palmes, T. Hayasaka, S. Usui. Mutation-based Genetic Neural Network. *IEEE Transactions on Neural Networks*, 16:3:587–600, May 2005.

[10] A. Watkins and J. Timmis. Artificial Immune Recognition System (AIRS): An Immune-Inspired Supervised Learning Algorithm. *Genetic Programming and Evolvable Machines*, 5:291–317, 2004.

[11] R. G. Weinard. Somatic mutation, affinity maturation and antibody repertoire: A computer model. *Journal of Theoretical Biology*, 143:343–382, 1990.

[12] J. A. White and S. M. Garrett. Improved pattern recognition with artificial clonal selection. In *2nd International Conference on Artificial Immune Systems (ICARIS-03)*, pages 181–193, 2003.