Genetic Learning of Perceptron Networks

Slušný, Stanislav
2006

Dostupný z http://www.nusl.cz/ntk/nusl-35655

Genetic Learning of Perceptron Networks

Post-Graduate Student:
MGR. STANISLAV SLUŠNÝ
Institute of Computer Science
Academy of Sciences of the Czech Republic
Pod Vodárenskou věží 2
182 07 Praha 8
Czech Republic
stanislav.slusny@matfyz.cz

Supervisor:
MGR.. ROMAN NERUDA, CSC.
Institute of Computer Science
Academy of Sciences of the Czech Republic
Pod Vodárenskou věží 2
182 07 Praha 8
Czech Republic
roman@cs.cas.cz

Field of Study:
Software systems

Abstract

This paper reviews different combinations between the most widely used type of neural networks – a multi-layer perceptron – and evolutionary algorithms. Methods to train the neural network are tested using a real-world classification problems from Proben1 benchmark suite. It is shown, that combining evolutionary algorithms with neural networks can lead to better results than relying on neural networks alone. Searching for a suitable architecture can help to find neural networks with improved performance.

1. Introduction

Artificial neural networks (ANNs) are a computational paradigm modeled on the human brain that have been applied to a variety of classification and learning tasks for a few reasons. Despite their simple structure, they provide very general computational capabilities and they can adapt themselves to different tasks, i. e. they are able to learn.

Evolutionary algorithms can be viewed as an alternative to classical optimization techniques, based on a biological metaphor: over many generations, natural populations evolve according to the principles of natural selection and "survival of the fittest", first clearly stated by Charles Darwin in The Origin of Species [1]. The basic principles of Evolutionary algorithms were first laid down rigorously by Holland [2].

In this paper, we present a comparison of different approaches to ANN learning problem. We will focus on combination of gradient and evolutionary techniques, and we will try to find optimal weights and topology of neural networks. Several experiments will be carried out on data taken from Proben1 [3] benchmark collection.

The organization of this paper is as follows. First we briefly describe perceptron networks and the core of genetic we have used. Then, the two main approaches — evolutionary learning of the weights, and evolution of network architecture — are presented. Finally, the experiments are reported and discussed.

2. Multilayer perceptron network

An ANN is an interconnected network of simple computing units called *neurons*[4]. Each neuron has n real inputs, and each input x_i has assigned a real weight w_i.

The output $y(x)$ of a neuron is defined in equation 1:

$$y(x) = g\left(\sum_{i=1}^{n} w_i x_i\right), \tag{1}$$

where x is the neuron with n input dendrites (x_0 ... x_n), one output axon $y(x)$, (w_0 ... w_n) are weights and $g : \Re \rightarrow \Re$ is the activation function that determines how powerful the output (if any) should be from the neuron, based on the weight sum of the input.

One of the most common activation function is a logistic sigmoid function 2

$$\sigma(\xi) = 1/(1 + e^{-\xi t}), \tag{2}$$

where t determines its steepness. Very important property of the sigmoid function is smoothness and the fact, that it has an easily calculated derivative.

In a *multilayer feedforward* ANN, the neurons are ordered in layers, starting with an *input layer* and ending with an *output layer*. Between these two layers there can be a number of *hidden layers*. Connections in this kind of networks only go forward from one layer to the next. Neurons in input layer are called *input neurons*, similarly neurons in output layer are called *output neurons*. Let us denote n_I, n_O, n_H number of input, output and all hidden neurons, respectively.

Multilayer feedforward ANNs work in two different phases: In a *training phase* (learning phase) the ANN is trained to return a specific output given a specific input. In the *execution phase* the input is presented to the input layer, propagated through all the layers (using equation 1) until it reaches the output layer, where the output is returned. The *architecture* of ANN defines the number of layers, number of neurons in each layer and connections between neurons.

We will focus on a learning situation known as a *supervised learning*, in which a set of input/desired-output patterns is available. The training process can be seen as an optimization problem, where we wish to minimize the *mean square error* (E_{MSE}) of the entire set of training data. E_{MSE} is defined as the squared error of the ANN for a set of patterns:

$$E_{\text{MSE}} = \sum_{k=1}^{p} E_k(w) \tag{3}$$

$$E_k(w) = \frac{1}{2} \sum_{j \in Y} (y_j(w, x_k) - d_{kj})^2, \tag{4}$$

where Y is a set of output neurons, p is the number of patterns in the set, y_j is an output of neuron j given weight vector w and k-th input pattern x_k, and d_{k_j} is desired output of output neuron j for input pattern k.

The *classification problem* is that of assigning an input vector to one of M classes. Each pattern from the training pattern set contains an input vector and its desired output vector. The output of the network must be interpreted as a class. Such interpretation can be performed in different ways. One of them consists in assigning an output neuron to each class — when an input vector is presented to the network, the network response is the class associated with the output neuron with the largest output value. This method is known as the *winner-takes-all* approach.

For classification problems, E_{CLAS} reports — in a high-level manner — the quality of the trained ANN and is defined as a percentage of incorrectly classified patterns.

3. Evolutionary algorithm

Evolutionary algorithm has been investigated by John Holland [2], with a marked increase of interest within the last few years. Evolutionary search refers to a class of stochastic optimization techniques — loosely

based on processes believed to operate in biological evolution — in which a population of candidate solutions (chromosomes) evolves under selection and random "genetic" diversification operators. The problem is to find minimum or maximum of *fitness function*. Every member of population is called *individual* and represents a potential solution to a problem.

The algorithm 1 reveals skeleton of genetic algorithm used in our experiments. To fully describe genetic algorithm, it is needed to define how each solution will be represented, how initial population will be created and what genetic operators will be used in the Reproduction step.

Input: number of individuals in population N, number of elits E, maximum number of populations G_{max}.
Output: the best found solution of a problem.

1. Start: Create initial population of N individuals $P(0) = \{I_1, ..., I_N\}$, $i = 0$.
2. Evaluation of individuals: To compute fitness function for every individual I, build ANN corresponding to I and compute E_{MSE} for a training set. Let $\mathcal{F}(I) = E_{\text{MSE}}$.
3. Stop-condition: If $i = G_{max}$, finish and return solution represented by individual with minimal value of fitness function.
   (a) Creation of new population $P(i+1)$ from population $P(i)$: Create empty population $P(i+1)$.
   (b) Selection: Choose E best individuals from population $P(i)$ and move them to population $P(i+1)$. With chosen operator of selection choose $N - E$ individuals and insert them to population $P'(i)$.
   (c) Reproduction: Apply chosen operators on population $P'(i)$, the result is population $P(i+1)$.
      i. Application of binary operators: If population $P'(i)$ contains odd number of individuals, insert chromosome of the first individual from population $P'(i)$ into population $P''(i)$. From population $P'(i)$ choose pairs of chromosomes C and D and apply on them reproduction operators, new chromosomes C' and D' insert to population $P''(i)$.
      ii. Application of unary operators: On every chromosome from population $P''(i)$ apply chosen unary operator, new chromosome insert to population $P(i+1)$.
4. New generation: $i = i + 1$, Continue by step 2.

Table 1: Skeleton of the genetic algorithm.

We have used the *roulette wheel selection* in all experiments. The selection operator performs the equivalent role to natural selection — it chooses individuals for next population proportionally to their fitness values. As we wanted to minimize the error function in all our experiments, the probability p_i of selection i-th individual is defined by the equation 5

$$p_i = \frac{(1+\varepsilon) * C - \mathcal{F}(I_i)}{N * (1+\varepsilon) * C - \sum_{j=1}^{N} \mathcal{F}(I_j)}, \tag{5}$$

where N is the number of individuals and C is the maximal fitness value in the population, and ε is a small positive constant, which ensures that probability of selection the worst individual will be non-zero.

4. Evolution of weights

Weight training in ANN is usually formulated as a minimization of error function, and is carried out by some gradient descent algorithm such as Back-Propagation, or one of its many variants[5]. Due to its use of gradient descent, these algorithms often get trapped in the local minimum of the error function,

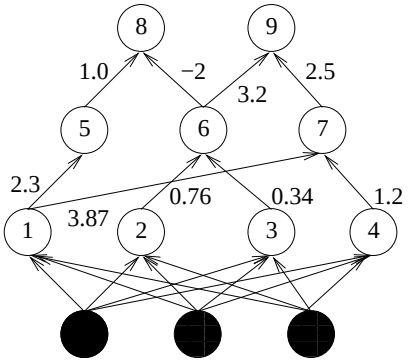

Figure 1: Example of ANN with seven hidden units in two hidden layers, three input and two output neurons.

$$\begin{bmatrix} \otimes & \otimes & \otimes & \otimes & 2.3 & \otimes & 3.87 & \otimes & \otimes \\ \otimes & \otimes & \otimes & \otimes & \otimes & 0.76 & \otimes & \otimes & \otimes \\ \otimes & \otimes & \otimes & \otimes & \otimes & 0.34 & \otimes & \otimes & \otimes \\ \otimes & \otimes & \otimes & \otimes & \otimes & \otimes & 1.2 & \otimes & \otimes \\ \otimes & \otimes & \otimes & \otimes & \otimes & \otimes & \otimes & 1.0 & \otimes \\ \otimes & \otimes & \otimes & \otimes & \otimes & \otimes & \otimes & -2 & 3.2 \\ \otimes & \otimes & \otimes & \otimes & \otimes & \otimes & \otimes & \otimes & 2.5 \end{bmatrix}$$

Figure 2: Example of encoded ANN from the previous figure.

and are incapable of finding global minimum. As the evolutionary algorithm need not to use the gradient information and works with population of potential solutions, it has a better ability to avoid local minimum trap. What more, evolutionary algorithm can be applied in situations, when gradient information is not available at all, it can handle global search problem in a vast, complex and non-differentiable surface. The evolutionary approach also makes it easier to generate ANNs with some special characteristics. For example, ANNs complexity can be decreased and its generalization ability increased by including a special term in the fitness function penalizing large networks.

4.1. Representation

The standard genetic algorithm uses binary strings to encode alternative solutions. In a such representation scheme, each connection weight is represented by a number of bits of certain length. The advantages of this representation are mainly simplicity and generality. It is possible to use well known crossover (such as one-point crossover) or mutation operators for binary strings. Although there are several encoding methods, that encode real numbers with different range and precision, trade off between precision and range often has to be made. Real-world experiments demand big precision, what causes too long chromosome for which the evolution process becomes non-efficient. Therefore, different encoding method is used. The chromosome is interpreted as a matrix of dimensions $n_H \times n$, where $n = n_O + n_H$. In the i-th row and the j-th column there is either a special symbol $\otimes$, if neurons i and j are not connected, or the weight of connection from i to j. The following connection matrix from figure 2 encodes the ANN from the figure 1. The chromosome is then created by concatenating all the rows from the matrix. In case of feedforward ANNs, the entry on i-th row and j-th column can be non-zero only for $i < j$. This reduces the length of the chromosome and the evolutionary process becomes more effective.

4.2. Initial population and operators

Initial population consists of fully connected neural networks with randomly initialized values. To each connection a weight from $[-1.0, 1.0]$ interval is assigned.

For training the weights of ANN, we experimented with operator *Biased-Weights-Mutation*[6], which adds

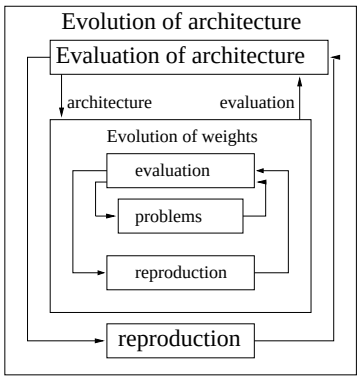

Figure 3: Schema of evolutionary algorithm searching optimal architecture of ANN.

to the weights of randomly selected connections in the parent network with values chosen randomly from the same probability distribution used for initialization. *Weights-Crossover* performs uniform crossover with the basic units exchanged being the set of all incoming weights of a particular node.

5. Evolution of architecture

In the previous section, we have assumed that the evolved ANN has a fixed architecture. Selection of a good architecture is state of art — although it has significant impact on network's information processing capabilities, there are not known satisfiable rules, on how to choose a good one for a particular task. ANN with only a few connections may not be able to satisfiable learn the task because of its limited capacity, while ANN with large number of connections and neurons may over fit and fail to have a good generalization ability.

Similarly to previous section, the problem of determining optimal architecture for a particular task can be formulated as an optimization problem. As stated in [7], evolutionary algorithms are a good method for searching in such a complex surfaces. In the case of *simultaneous evolution of weights and architecture* both weights and characteristics of architecture are encoded in the chromosome. As shown on figure 3, this method contains the previously solved problem of determining optimal set of weights as a subtask. On the other side, the problem of *separated evolution of weights and architectures* is the noisy fitness evaluation, caused by random initialization of weights. Only architecture is encoded in chromosome, weights have to be learned later. The fact that an individual gained better fitness value does not mean that this individual is really better — in is necessary to repeat the evaluation and average the obtained values. This way, the computation time increases dramatically.

5.1. Representation

We used the *indirect encoding scheme* in our experiments [7]. Chromosome consists of several sections, which hold information about some important characteristic of ANN's architecture. In the first section is stored the number of hidden layers, follows section with information about the number of neurons in each hidden layer and in the case of simultaneous evolution of weights and architecture, the section with connection matrix is appended. The operators used in evolutionary algorithm must be aware of this representation — operators from previous section are allowed to modify only the last section of the chromosome. To modify the remaining parts, new operators are introduced.

5.2. Operators

There are two operators that work on the level of units. The *Remove-Neuron* operator removes randomly selected neurons. Creation of new neurons is done by the *Duplicate-Neuron* operator, which selects random neuron and duplicates it. Similarly, there are two operators working on connections, the *Remove-*

Number of hidden layers.
Number of neurons in the first hidden layer.
...
Number of neurons in the last hidden layer.
Connection matrix.

Table 2: Chromosome for evolving architectures of ANN.

Connection operator removes random number of connections from each neuron,while the *Add-Connection* operator randomly adds connection between two neurons.

6. Experiments

A set of experiments has been carried out on several data sets from the Proben1 [3, 8, 9] benchmark. This way we are able not only to provide relative comparison, but also to explore the efficiency of the algorithms with respect to the best results obtained by other methods and authors. In the following we briefly describe several experiments and try to generalize the results.

We have chosen four classification problems: The goal of ANN is to classify tumor as either benign or malignant in the Cancer problem, diagnose diabetes of Pima indians in Diabetes problem, predict the heart disease in Heart problem and recognize one of 19 different diseases of soybean in Soybean problem. Characteristics of each data set are shown in table 3.

	Classes	Examples	b	c	n	Tot.	b	c	m	Tot.
Cancer	0	9	0	9	0	9	0	9	2	699
Diabetes	0	8	0	8	0	8	0	8	2	768
Heart	1	6	6	13	18	6	11	35	2	920
Soybean	16	6	13	35	46	9	27	82	19	683

Table 3: Problems and the number of binary, continuous, and nominal attributes in the original dataset, number of binary and continuous network inputs, number of network inputs used to represent missing values, number of classes, number of examples.

The results of experiments are reported in terms of the values of the two errors, E_{MSE} and E_{CLAS} measured both on the training set and previously unseen test set. In the following tables we use symbols M_t (or M_s) for E_{MSE} over training (or testing) set, and C_t (and C_s) for E_{CLAS} over training (testing, respectively) set.

6.1. Searching for suitable connections

This experiment tested the separated evolution of architecture and weights. Thus, the algorithm consists of two steps - in the first one the direct architecture encoding described in Tab. 2 is used and evolved. To determine a fitness of such an individual, several (30 in our case) randomly initiated runs of rprop algorithm are carried. In the second step, the evolved architecture is trained by full-fledged run of a back propagation algorithm (10 times for different random weights initializations). Results can be compared to Tab. 4, which gathers results of the classical back propagation, the rprop gradient learning algorithm and GA using the Biased-Weights-Mutation and Weights-Crossover (cf. Tab. 4). Typically we ran the gradient algorithm for 5000-25000 epochs, while the GA ran for 1000-2000 generations (with population of 300 individuals).

Results of this experiment were quite satisfiable (cf. Tab. 5): GA was able to find better architectures, containing less connections, and achieving better training error. As a side effect, these specialized architectures usually achieved worse generalization error, in comparison to fully connected architectures, which is understandable (compare Tab. 4 and Tab. 5).

	Diabetes			Cancer		
	BP	RP	GA	BP	RP	GA
M_t	0.2675	0.2473	0.2882	0.0463	0.0390	0.0484
C_t	0.2645	0.2616	0.2083	0.0280	0.0438	0.0286
M_s	0.3217	0.3154	0.3268	0.0242	0.0180	0.0159
C_s	0.2645	0.2615	0.2266	0.0280	0.0438	0.0057
	Soybean			Heart		
	BP	RP	GA	BP	RP	GA
M_t	0.0150	0.2112	0.7416	0.1840	0.0904	0.2372
C_t	0.2536	0.3080	0.5585	0.2033	0.2079	0.1536
M_s	0.1196	0.2907	0.7474	0.2574	0.2559	0.2904
C_s	0.2536	0.3078	0.5513	0.2033	0.2077	0.1913

Table 4: Error comparison for back propagation (BP), rprop (RP) and genetic algorithm (GA) used for weights learning.

	Diabetes	Cancer	Soybean	Heart
M_t	0.4541	0.0436	0.0069	0.0566
C_t	0.3516	0.0157	0.1615	0.1764
M_s	0.4568	0.0238	0.1369	0.2501
C_s	0.3516	0.0157	0.1615	0.1764

Table 5: Search for connections: Error results for architectures recommended by GA.

6.2. Searching for a suitable architecture

The goal of the second experiment was to combine the search for architecture with the evolution of weights in one GA. We have started with individuals with random number of hidden layers (between 1–4), which remained constant during the algorithm. However, the number of neurons and connections were varying, as well as the values of weights. The following genetic operators we used:

operator	p_m
Duplicate-Neuron	0.05
Remove-Neuron	0.05
Add-Connection	0.05
Remove-Connection	0.05
Biased-Weights-Mutation	0.1

Table 6: Operators probability.

The overall results were very satisfiable (cf. Tab. 7, compare with Tab. 4). The GA was able to evolve suitable architectures for a given task. It is interesting, that most of the solutions had just one hidden layer, some of them had two. In some papers authors use penalization for more complex solutions, which was not necessary here, because the evolution tends to exclude more complex networks based on their fitness anyways. The architectures recommended by the GA provided even better results than the ones reported as best (found by human) in [3].

7. Conclusions

As shown, evolution can be introduced into ANN learning problem at different levels. Suggested algorithms were tested on real-world problems from Proben1 benchmark suite. The evolutionary algorithm is a complex and robust method, which can be used to search both optimal weights and architecture of ANN. Although the evolutionary process can be easily parallelized, computation is always very time consuming. The results obtained by simultaneous evolution of weights and architecture were surprisingly good. Not

	Diabetes	Cancer	Soybean	Heart
M_t	0.2679	0.0382	0.0176	0.1025
C_t	0.1927	0.0190	0.0088	0.0551
M_s	0.3108	0.0184	0.0989	0.2342
C_s	0.2266	0.0057	0.0616	0.1435

Table 7: Search for an architecture and weights: Error results for genetically evolved network architectures.

only the resulting ANNs gained excellent results on the training set, they showed a good generalization ability. The complexity of found architecture for a particular task mirrored it's real difficulty.

The combination of evolutionary techniques and ANNs can lead to better intelligent systems, than relying on ANNs alone. With the increasing power of parallel computers, the evolution of large ANNs becomes feasible.

References

[1] Ch. Darwin. The Origin of Species, New American Library, Mentor paperback, 1859.

[2] J. Holland. Adaptation In Natural and Artificial Systems. The University of Michigan Press, Ann Arbour, 1975.

[3] L. Prechelt. PROBEN1 - A set of benchmarks and benchmarking rules for neural network training algorithms. Technical Report 21/94, Universitat Karlsruhe, Germany, September 1994.

[4] J. Šíma, R. Neruda. Teoretické otázky neuronových sítí, Matfyzpress,

[5] M. Riedmiller, H. Braun. A direct adaptive method for faster backpropagation learning: The RPROP algorithm, Proceedings International Conference

[6] D. Montana, L. Davis. Training feedforward neural networks using genetic algorithms, in Proc. 11th Int. Joint Conf. Artificial Intelligence. San Mateo, CA: Morgan Kaufmann, 1989, pp. 762-767. P

[7] X. Yao, Y. Liu. Evolving neural network ensembles by minimization of mutual information,' International Journal of Hybrid Intelligent Systems, 1(1):12-21, January 2004. raha, 1996.

[8] O. L. Mangasarian, W. H. Wolberg. Cancer diagnosis via linear programming, SIAM News, Volume 23, Number 5, September 1990, pp 1 & 18.

[9] R.S. Michalski, R.L. Chilausky. Learning by Being Told and Learning from Examples: An Experimental Comparison of the Two Methods of Knowledge Acquisition in the Context of Developing an Expert System for Soybean Disease Diagnosis, International Journal of Policy Analysis and Information Systems, Vol. 4, No. 2, 1980.

[10] J. R. Koza, J. P. Rice. Genetic Generation of Both the Weights and Architecture for a Neural Network, Proceedings of the IEEE Joint Conference on Neural Networks, pp. 397-404, 1991.

[11] H. Kitano. Designing neural networks using genetic algorithms, 1989.

[12] C. Forgaty, T., R. Huang. Implementing the genetic algorithm on transputer based parallel processing systems. In Proceedings of Parallel Problem from Nature, pages 145-149, 1991.