ESANN'2006 proceedings - European Symposium on Artificial Neural Networks
Bruges (Belgium), 26-28 April 2006, d-side publi., ISBN 2-930307-06-4.

Using distributed genetic programming to evolve classifiers for a brain computer interface

Eva Alfaro-Cid, Anna I. Esparcia-Alcázar and Ken Sharman *

Instituto Tecnológico de Informática
Universidad Politécnica de Valencia, Valencia, Spain

Abstract. The objective of this paper is to illustrate the application of genetic programming to evolve classifiers for multi-channel time series data. The paper shows how high performance distributed genetic programming (GP) has been implemented for evolving classifiers. The particular application discussed herein is the classification of human electroencephalographic (EEG) signals for a brain-computer interface (BCI). The resulting classifying structures provide classification rates comparable to those obtained using traditional, human-designed, classification methods.

1 Introduction

BCI technology is based on the idea that different thoughts produce different EEG signals [10]. Therefore it might be possible, by measuring and classifying the EEG signals, to translate certain thoughts into actions. The ultimate goal of the research in this field is to develop a system that will allow an individual to control an external device through his/her EEG signals.

In a BCI system, first, the EEG data are acquired. Then this data needs to be pre-processed to eliminate noise and artifacts (e. g. eye blinking). Then, there is a stage of feature analysis where the EEG signals are compressed. Afterwards, the data are divided into the training set and the test set. BCI systems need to be trained because every individual has different EEG patterns. In the training phase the computer selects a mental state and the user focuses on whatever thought he associates with that state for a while. The EEG data are stored during that time, processed and classified. Finally the user gets some feedback on the success or failure of the classification. Once the training is over, we hope that the BCI system will be able to classify the EEG readings with high accuracy.

The classification stage of the BCI system is the focus of the research reported in this paper. Previous research in this field has used classifiers such as neural networks or support vector machines [4]. However, in this work we have investigated a different approach to the problem. We are using GP [7] to automatically *evolve* a structure able of classifying the EEG signals. GP is provided with a set of basic operations and then evolution leads the search towards better and better classifiers. The objective is to evolve classifiers that yield superior performance (lower error rate) and are *simpler* (less computationally costly) than classical systems. The general goal of a BCI system is online classification, however, due to the long running times, our approach focuses on offline classification.

*This work was supported by the Conselleria de Empresa, Universitat i Ciencia of the Generalitat Valenciana (EvoProSe - Ref GV05/256)

2 Genetic Programming for Classification

There are many approaches for solving classification problems: statistical methods, neural networks, fuzzy logic, etc. Recently, some researchers have reported success using GP techniques for classification purposes [6]. One of the advantages of this technique is that it is not constrained by the level of expertise of the developers. This is especially appropriate in the case of BCI applications, due to the lack of knowledge about how the human brain works and how to relate the EEG activity to particular thoughts.

In the GP algorithm, traditionally, the structure being evolved has a tree shape. The GP scheme is as follows. An initial population of trees is created at random. Each individual in the population is evaluated using a fitness function. This fitness function assigns a value to the individual according to how well it solves the problem at hand (i. e. the classification of the training set of EEG data). A new population is created through reproduction, crossover and mutation. Generally speaking, individuals with a higher level of fitness are selected for reproduction mimicking a "survival of the fittest" strategy. However, occasionally lower fitness individuals are selected to maintain population diversity. The loop is run until a certain termination criterion is met.

2.1 Function and Terminal Set

Prior to creating a GP environment the designer must define which functions (internal tree nodes) and terminals (leaf branches) are relevant for the problem to solve. This choice defines the search space for the problem in question. Table 1 shows the set of functions that we have used. Although this is a fairly small set of functions it allows us to specify a huge range of multivariable functions.

Function	No. arguments	Function	No. arguments
$\mathcal{R}$ (insert random constant)	0	+	2
logarithm	1	-	2
exponential	1	$*$	2
cosine	1	/	2
If $a_1 \leq a_2$ Then a_3 Else a_4	4		

Table 1: GP function set

2.2 Serial Processing of Data

In a conventional classifier the data to be classified is normally presented to the system as a vector, i. e. all the data is presented simultaneously to the classifier. In our case, we have adopted an alternative approach in which the data is presented to the classifier in series. The aim of this is to reduce the complexity of the classifier. A conventional classifier for data with N features requires a system with N inputs. In our approach we have a single input which is applied N times. In effect we are trading off space for time.

To do this our GP has a set of terminals $\{x_0, x_1, \ldots, x_{\mathrm{M}}\}$ which represent the current input and the previous M inputs to the classifier. We also have another set of terminals labeled $\{y_1, y_2, \ldots, y_{\mathrm{K}}\}$ representing K previous outputs from the GP tree (see Figure 1).

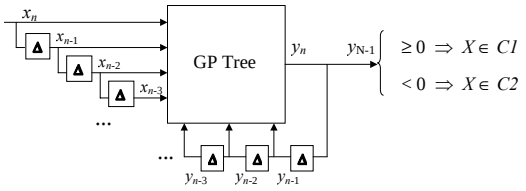

Fig. 1: Serial processing of data using a single input which is applied N times

The upshot of all this is that our GP trees implement functions of the form $y_n = f(x_n, x_{n-1}, \ldots, x_{n-M}, y_{n-1}, \ldots, y_{n-K}) \quad n = 0, 1, 2, \ldots, N.$

2.3 Signal Classification

In this work we are interested in a 2-category classification of the EEG data. In fact we want to determine if the user's thought is meant to be a command for moving a screen cursor up or down.

For the implementation, let the data to be classified be $X = \{x_0, \ldots, x_{\mathrm{N}-1}\}$. Then we apply this data set as input to a GP tree and calculate the outputs y_i, $i = 0, \ldots, \mathrm{N}-1$ (i. e. the tree is executed N times in total). The final output, $y_{\mathrm{N}-1}$, indicates the classification results according to:

$$y_{\mathrm{N}-1} \geq 0, \ X \in C1 \tag{1}$$

$$y_{\mathrm{N}-1} < 0, \ X \in C2 \tag{2}$$

where C1 and C2 are the two class labels.

During evolution, we use a training set of T feature vectors and known labels $\{(X_i, \ c_i), \ i = 1, \ldots, \mathrm{T}\}$ where $X_i \in \Re^N$ is the N-dimensional feature vector, and $c_i \in \{C1, C2\}$.

To evaluate the fitness of each individual, we apply each X_i in turn and calculate its chosen class, c_i', according to (1) and (2). We then count the number of *hits* as the number of times $c_i = c_i'$. Note that to calculate the fitness of an individual, the individual's tree must be evaluated TN times.

3 Implementation Using DREAM/JEO Framework

The GP implementation that we have used is based on the JEO (Java Evolving Objects) library developed by [2] within the project DREAM (Distributed Resource Evolutionary Algorithm Machine) [1]. The project's aim was to develop a complete distributed peer-to-peer environment for running evolutionary optimization applications. We selected this framework for its easy ability to parallelize the evolution and to distribute the computation.

Crossover rate	0.8
Cloning & mutation rate	0.1
Initialization method	Ramped half and half
Max. tree depth & max. initial tree depth	18 & 7
Crossover operator	Bias tree crossover
Mutation operator	Subtree mutation
Selection operator	Tournament selection
Tournament group size	10
Number of islands	2
Number of emigrants per island & generation	1
Population size	200(runs 1-6)/500 (runs 7-8)
Termination	Time limit

Table 2: Parameters used in the GP optimization algorithm

4 Experimental Procedure and Results

4.1 EEG Signals

The EEG signals used in the classification problem presented here were gathered for "The BCI Competition 2003" [3]. The objective of the competition was to evaluate the current state of the art of the BCI field.

The data sets were taken from a healthy subject. The subject was asked to move a cursor up and down on a computer screen, while his cortical potentials were taken. Cortical positivity/negativity lead to a downward/upward movement of the cursor. An interval of 3.5 s of every trial is provided for training and testing. Using a sampling rate of 256 Hz it results in 896 samples per channel for every trial. Samples of the 6 channels being recorded are stored concatenated. Therefore the full raw data set is 5376 samples.

The signals were downsampled by a factor of 6 using the downsample command of Matlab, which also includes an antialiasing filter. Thus our feature vector to be classified consists of 5376/6=896 points (i.e., in our case N=896).

Our training set consisted of 268 data records (T=268). After evolution, we tested the best evolved individuals on a set of 293 *unseen* data records.

4.2 Distributed GP Algorithm

The distributed GP (see Table 2 for the GP parameters) follows the island model [5]. Previous experiments have proven that 2 islands reduce the optimization time by more than a half.

The fitness function consists of two terms. The first term refers to the number of hits in the classification (see Section 2.3) and the second term is an inverse function of the size (number of nodes) of the tree to avoid bloat [8].

Run No.	%Training hits	%Test hits	No. fitness calculations
1	71.6	83.6	3200
2	70.9	70.3	6800
3	72.4	85.0	55200
4	73.1	79.2	11600
5	78.4	81.2	63200
6	72.8	85.7	6600
7	81.7	70.0	64000
8	79.5	82.9	35500

Table 3: Numerical results obtained in the 8 GP classification runs

4.3 Results

The GP classification algorithm was run 8 times and the training and testing results are shown in Table 3 together with the number of fitness calculations needed to reach the result. On average, every fitness calculation took 1.3 seconds.

The results obtained are very reasonable. The rate error obtained in run 6 would have classified in 4th position in the BCI competition and the rate error in run 3 equalizes the result obtained by the contributor that classified in 4th position (see competition results in [11]).

It can be observed that the best results in the training phase do not always provide the best results in the testing. In fact, the three best results (run 1, 3 and 6) performed quite poorly in the training, and the tree that got the best result in the training (run 7) got a very disappointing error rate in the testing.

Analyzing in more detail the trees evolved along the generations in run 7 it can be observed that, in generation 3 the best result in the training gets a hits percentage of 71.6%, far lower that the final hits percentage. However, if that tree is used to classify the signals in the test set, the hits percentage is 84.6% (versus 70.0% obtained in the final generation). This is a clear case of *overfitting*. It seems that, above a certain percentage of hits in the training, further attempts of improving the classification do not imply better percentages of test hits.

Oddly, the percentages of hits obtained in the testing are higher than those of the training. This can be due to the fact that these signals were recorded later in the experiment and the patient was better trained for the task [9].

Table 4 shows the tree structures the GP has converged to in runs 3 and 6 (best results obtained). The best results have been obtained with trees of a very reasonable size. Although they may look a bit convoluted, once analyzed, they are very simple functions. This simplicity becomes more obvious when compared to classifiers such as neural networks or support vector machines.

5 Conclusions

We have shown how GP can evolve high performance classifiers for EEG data. The resulting performance of our classifiers was close to that of the best human

Run No.	Resulting tree
3	$y_n = -x_{n-6} + (if\ x_{n-3} \leq (if\ y_{n-8} \leq \cos x_{n-2}\ then\ \cos x_{n-8}$ $else\ x_{n-3} + x_{n-6})\ then\ y_{n-4}\ else\ (if\ x_{n-3} \leq\ (if\ \cos x_{n-8} \leq x_n$ $then\ \cos y_{n-7} - x_{n-6}\ else\ x_{n-8})\ then\ y_{n-4}$ $else\ (if\ x_{n-3} \leq y_{n-4}\ then\ y_{n-9}\ else\ y_{n-3} - x_{n-6}) - 2x_{n-6}))$
6	$y_n = y_{n-9} - (2x_{n-4} + g(y_{n-5} * \exp y_{n-2}))$ $g(z) = \log(\log(\log(\log(\log(\log(\log(\log(\log(\log(z)))))))))))$

Table 4: Best tree structures obtained in the GP runs

designed classifiers. Interestingly, some of the resulting classifiers have a very simple structure compared to conventional systems and could be a key factor in the real-time performance of the system.

References

[1] M.G. Arenas, P. Collet, A.E. Eiben, M. Jelasity, J.J. Merelo, B. Paechter, M. Preuß and M. Schoenauer, A framework for distributed evolutionary algorithms. In J.J. Merelo el al., editors, *proceedings of the 7^{th} international conference on parallel problem solving from Nature* (PPSN 2002), LNCS 2439, pages 665-675, Springer-Verlag, 2002.

[2] M.G. Arenas, B. Dolin, J.J. Merelo, P.A. Castillo, I. Férnandez de Viana and M. Schoenauer, JEO: Java Evolving Objects. In W.B. Langdon et al., editors, *proceedings of the Genetic and Evolutionary Computation Conference* (GECCO'02), pages 991-994, July 9-13, New York (USA), 2002.

[3] B. Blankertz, K.R. Müller, G. Curio, T.M. Vaughan, G. Schalk, J.R. Wolpaw, A. Schlögl, C. Neuper, G. Pfurtscheller, T. Hinterberger, M. Schröder and N. Birbaumer, The BCI competition: Progress and perspectives in detection and discrimination of EEG single trials, *IEEE Transactions on Biomedical Engineering*, 51(6): 1044-1051, 2004.

[4] D. Garrett, D.A. Peterson, C.W. Anderson and M.H. Thaut, Comparison of linear, non-linear, and feature selection methods for EEG signal classification, *IEEE Transactions on Neural System Rehabilitation Engineering*, 11(2):141-144, 2003.

[5] D.E. Goldberg and E. Cantú-Paz, Modeling idealized bounding cases of parallel genetic algorithms. In J.R. Koza et al., editors, *proceedings of the 2nd Annual Conference on Genetic Programming*, pages 353-361, July 13-16, Stanford (USA), 1997

[6] J.K. Kishore, L.M. Patnaik, V. Mani and V.K. Agrawal, Genetic programming based pattern classification with feature space partitioning, *Information Sciences*, 131:65-86, Elsevier, 2001.

[7] J.R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*, MIT Press, 1992.

[8] S. Luke, Issues in Scaling Genetic Programming: Breeding Strategies, Tree Generation, and Code Bloat. PhD Thesis. University of Maryland, USA, 2000

[9] B.D. Mensh, J. Werfel and H.S. Seung, BCI competition 2003 – Data Set Ia: Combining gamma-band power with slow cortical potentials to improve single-trial classification of electroencephalographic signals, *IEEE Transactions on Biomedical Engineering*, 51(6):1052-1056, 2004.

[10] J.R. Wolpaw, N. Birbaumer, W.J. Heetderks, D.J. McFarland, P.H. Peckham, G. Schalk, E. Donchin, L.A. Quatrano, C.J. Robinson and T.M. Vaughan, Brain-Computer interface technology: a review of the first international meeting, *IEEE Transactions on Rehabilitation Engineering*, 8(2):164-173, 2000.

[11] `http://ida.first.fhg.de/projects/bci/competition/results/index.html`