# Fine-tuning Artificial Neural Networks Automatically

FRANCISCO REINALDO, RUI CAMACHO
LUÍS PAULO. REIS
Universidade do Porto
Faculdade de Engenharia & LIACC
Rua Dr. Roberto Frias, sn, 4200-465, Porto
PORTUGAL
{reifeup,rcamacho,lpreis}@fe.up.pt

DEMÉTRIO R. MAGALHÃES
UnilesteMG
Depto de Exatas & CSI
Av. T. Neves, 35170-056, C. Fabriciano
BRASIL
reno@unilestemg.br

*Abstract:* To get the most out of powerful tools expert knowledge is often required. Experts are the ones with the suitable knowledge to tune the tools' parameters. In this paper we assess several techniques which can automatically fine tune ANN parameters. Those techniques include the use of GA and Stratified Sampling. The tuning includes the choice of the best ANN structure and the best network biases and their weights. Empirical results achieved in experiments performed using nine heterogeneous data sets show that the use of the proposed Stratified Sampling technique is advantageous.

*Key–Words:* Fine-tunning, Artificial Neural Networks, Wrapper, Stratified Sampling

## 1 Introduction

Artificial Neural Networks (ANN) are currently a widely used technique to solve complex problems on a wide range of real applications. Such applications vary from financial predictions to control systems design. In order to achieve the expected results, ANN experts hardly work on designing and configuration of ANN control modules. Unfortunately, there is no cookbook explaining how to fine tuning the technical properties that define an ANN characteristic. We foresee that the main reason to this problem be largely unsolved is on the substantial varying of the parameter values. Since a user has extensive experience to design and setup an ANN control module to the problem, it is difficult to obtain reliable results only by using empirical and manual setup. In this study, we propose an automatic method for common users can develop and fine tuning ANNs by interlacing Machine Learning (ML) and probabilistic techniques.

Most of the various studies consider only three main technical properties to design and setup an ANN control module, such as: (a) the topology; (b) the learning rules; and (c) the initial weights on the synaptic connections.

The first property, the topology, represents the ANN structure definition, i.e., the pattern of connections between neurons, the bias, the number of hidden layers, the number of neurons in each hidden and output layer. The second property, the learning rules, represents the tuning of the parameter values that define the characteristics of an adequate model, i.e., the learning rate, the momentum rate, the steepness rate, the kind of transfer functions to each hidden and output neuron and so on. Finally, the third property deals with the selection and distribution of the first weights on the connection links.

Several studies concerning the automatic tuning of ANN parameters may be found in the literature. Most of them use Genetic Algorithm (GA) as a stochastic search method used to find solutions [1]. Within this context, Prado [2] proposes the tuning of the most usual parameter values using GA. In [3], Harp *et al.* describe a study to find a good ANN architecture by setting the number of layers and the number of neurons in hidden layers. Whitley [4] uses GA to determine best weight of an ANN. Regarding the manual setup of parameters, Shamseldin *et al.*[5] combine different transfer functions in a hidden layer to reach the best model with a purpose to apply them in the context of the river flow forecast combination method.

Diverse authors have also been studied suitable methods to initialise an ANN training by choosing the first weights [6, 7, 8]. The methods coexist in using of a traditional technique to randomly or probabilistically extract the first weights from an unique range of values. After all, connections are populated by these weights.

Other approaches to develop GA-based *wrappers* have be attempted. Unfortunately, these approaches work with only traditional parameters, instead of a complete set. Moreover, the weights extraction process from an unique range of values is unable to be effective because it will hardly avoid saturation areas. However, none of the proposed solutions consider an approach to automatically fine tuning an ANN structure and weights by union of ML and statistic methods at the same time and offer an harmonic solution for the problems.

To overcome these drawbacks, we are proposing a method to automatically achieve fine tuning results and, to obtain low error rates. The method is developed in two main parts: a GA is first used as a *wrapper* to find adequate values to the set of ANN structure and parameters. Afterwards, in order to input the first weights, we have used Stratified Sampling (SS) to balance the weights on different synaptic connections. To estimate accuracy, we have used five-fold Cross-Validation. The proposed *wrapper* outperforms manual or semi-automatic setup and building ANN structure techniques described in [2, 4, 5].

The rest of the paper is structured as follows. Section 2 presents an overview of AFRANCI, the tool used to develop the experiments. Section 3 describes how Genetic Algorithms were applied to tune the ANN different parameters and structure. Section 4 explains how we improve the ANN weights values using Stratified Sampling. Experiments and results are presented in Section 5. We draw the conclusions in Section 6.

## 2 AFRANCI Tool

The proposed solutions were empirically evaluated using the AFRANCI tool [9], which is a graphic environment used to develop Intelligent and Cognitive Systems (ICS) [10].

AFRANCI tool has graphic facilities that lead common users to assemble and link together the learning algorithms (control modules) on the screen at different levels of abstraction. A control module can be a Machine Learning or other algorithm. After all, the user can arrange an ICS of heterogeneous clusters of control modules in a form of a circuit diagram[11], which it is very familiar to engineers in digital system projects and in model analysis system tools.

This tool has built-in some classes of open-source PyramidNet Framework platform, preserving Feedforward and Recurrent ANNs. The tool supports external libraries, such as WEKA library [12], CN2 Induction Algorithm [13] and GAlib [14]. In addition, the Tool includes Stratified Sampling and K-fold cross validation. By using these learning libraries and statistic techniques, the users can access an useful repository to work with data pre-processing, classification, regression, evaluation, clustering, stochastic search, fine tuning and association rules.

Basically, AFRANCI core can be divided in three main parts, which are the Graphic User Interface (GUI), the Machine Learning Modules (MLM) and the Automatic Open-Source Code Generator (ACG). Firstly, GUI is a set of main classes for modeling the learning modules as graphic elements on the desktop area. Secondly, MLM links the learning algorithms with the graphic modules, implementing their construction. Finally, ACG uses a high-performance interpretation algorithm to automatically encode the ICS diagram in a clean and ready-to-use standardised C++ open-source code.

The main point that leads us to use this Tool was the easiness to manage, in a unique environment development, the three main stages to build an ANN: Design and Set Up; Module's Training and Test; and Code Generation.

<u>Design and Set Up</u>: the design of the structure can be manually made by disposing graphic objects on the desktop area or automatically made by the use of Wizard. Manual process offers resources to perform drag-and-drop actions to arrange the essential elements of a project, such as input port, the module and output port on the desktop area. An input port can be an attribute or a sensor, as well as, the output port can be as actuator or an out variable. Each control module loads a CSV (Comma Separated Values) sample file. A sample file provides data for learning algorithms in the training stage and defines the respective input and output ports. To develop a complete wired network, the interconnections shall be established from the input port to the input of the module and, consequently, from the output of the module to the output port. The users can change the default parameter values of the objects or choose the learning algorithm in design-time properties. Conversely, in the automatic process, the Wizard helps the user to develop a wired network of modules as circuit diagram by only loading sample files;

<u>Module's Training and Test</u>: this stage checks if the whole ICS was fully interconnected and the datasets were load by the respective modules. The feature of automatic training process uses the data flow sequence to trigger the training sequence, independently from horizontal or vertical architecture level. It is worth mentioning that AFRANCI uses a known K-fold cross validation as an advanced statistical technique to get impartial results of training and test stage.

<u>Code Generation</u>: the code generator will encode the diagram of the ICS in a ready-to-use C++ code.

For instance, if we look inside of an ANN module, we will find input and output ports, connections, synaptic connections and weights, activation functions and other self-sufficient features acquired after the training stage.

## 3 The Automatic Tuning of Artificial Neural Networks

Almost all Machine Learning systems have parameter values that must be tuned to achieve a good quality for the constructed models. An experienced practitioner knows that changes in the parameter values may lead to quite different results. This is most often a severe obstacle to the widespread use of such systems.

As proposed by John [15] one possible approach to overcome such situation is by the use of a *wrapper*. A *wrapper* produces in parallel several models using different combinations of the learning algorithm and returns the "best" model. In our Tool the *wrapper* is a Genetic Algorithm that will improve the ANN construction procedure obtaining low Error Rates. This automatic tuning of parameters completely hides the details of the learning algorithms from the users. It is therefore a way to make the Tool usable by a wider range of users.

Fundamentally, a candidate is a chromosome composed of linear chains of small units named genes. A chromosome uses an alphabet of binary digits, integers or real values to represent in the gene each independent feature or allele as shown in Figure 1. Each gene has a fixed place, named *locus*, in the chromosome. A genotype is a collection of genes and alleles to create a candidate and a phenotype is a collection of the features of this candidate. The adaptation of each candidate is directly related with the phenotype. The traditional GA method[1] uses a binary alphabet, a fixed-length bit string chromosome and a population with a fixed size as well.

We have divided the construction of an ANN into four major items: choosing the structure; choosing the neuron's transfer function; choosing the bias to use and; deciding the values for the ANN learning algorithm. All these items are encoded in a chromosome as we explain next.

The chromosomes[14](see Figure 1) have encoded the following features:

- the learning rate (LR), the momentum rate (M) and the steepness rate(Sn);
- the bias for each hidden (bHL) and output (bOL) layer;
- the transfer functions in every neuron of the hidden layer (TFHL) and output (TFOL) layers;
- the number of neurons in every hidden layer (nHL).

| LR | M | Sn | bHL | bOL | TFHL | TFOL | nHL |
|---|---|---|---|---|---|---|---|

Figure 1: The chromosome structure encoded by the first process.

Another ingredient for using GAs concerns the evaluation of the solutions (chromosomes) produced, that is, defining the fitness function. Before receive a fitness score, first we need to obtain an objective score, which is the error rate extracted from the current candidate in the test phase. Using the linear scaling fitness function [1, 14], the objective score is encoded to a proportional non negative transformed rating fitness or fitness score.

A third ingredient to implement a GA concerns the selection of the most evolved candidates to reproduction. This implies more chances at spreading the candidate features in the next generation, i.e., to "preserve the knowledge" of that candidate. We have implemented one of the most popular methods to choose reproductive candidates called the roulette wheel selection method[1]. This method selects a candidate based on the highest fitness score relative to the remaining part of the population. The portion of the roulette wheel of each candidate is given by Equation (1), where $x_i$ is the candidate with a $f(x_i)$ probability area to be selected.

$$Portion(x_i) = \frac{f(x_i)}{\sum_{i=1}^{N} f(x_i)} \qquad (1)$$

A fourth item required to implement a GA is the combination of existing candidates and use a sexual crossover with one-cut point crossover technique, where the selection point is random. After that, the parents change the right side, generating two offspring and preserving the same previous population size. We have also used, additionally, the mutation operator that "disturbs" the chromosome of an existing chromosome to produce one new offspring.

Finally, the last process sets the stop measure of the GA search. We chose the number of generations achieved because after several evolutional steps, the last generation brings the "best" candidates with the highest fitness scores. After that, the GA evolves.

When used as a search method, the GA strength comes from: the totally parallel search on the solution space. The GA drives the search to promising areas via a population of potential candidates, minimizing the risk of find a solution in a maximum or minimum local; the blind search, referring to known only the necessary candidate cost function; the use of stochastic operators to guide to a extensive search having knowledge accumulated in earlier iterations. In addition, GA combines representation of candidate solutions and problem-specific genetic operators [16], in which there is a trend for good solutions each time an evolve process is re-started [1]. By this way, GA solves the problems by collecting knowledge about the problem and using knowledge to create acceptable solutions.

## 4 Improving the Learning Process of Artificial Neural Networks

The performance of ANN learning algorithms can be strongly affected by the initial values of weights of the connection links. According to Fausett[6], weights can influence the global minimisation of a network training error function. Traditionally the initial values of the weights are randomly generated within a specified range. However, arbitrary values selected from a unique range can be too sparse that the initial input signal falls in a saturate region or too small that causes extremely slow learning. It is well known that in order to find a high weight quality is a difficult task, noting that the amount of connection links and the substantial varying of the weights.

To improve the choice of the initial weights we propose the use of Stratified Sampling (SS).

The Stratified Sampling is an inductive technique of statistic inference used when a heterogeneity population is studied. In SS technique, a population is split into $s$ smaller proportional non-overlapping segments or strata. A stratum is a subset of the population and each sample of the subset has a known non-zero probability of being selected. The SS technique guarantees that all regions have at least one representative sample.

Below, we explain the process. Consider a population of samples, i.e., a set of weights. In order to balance the weights on the connection links, we use SS to proportionally split the population in $s$ disjoint subsets. Considering that each ANN has $n$ synaptic connections $\{w_1, w_2, w_3, \ldots, w_n\}$ that need to be populated by samples from the subsets $\{A_1, A_2, \ldots, A_s\}$, we use arrangement with repetition to distribute the samples on the connection links. Additionally, we use Random Sampling technique of the SS method to select the proportional number of samples from each $s$ to populate every $n$. To put in another way, we will have $s^n$ possible solutions. For example, assuming six different weights $\{w_1, w_2, \ldots, w_6\}$ have two strata $A_1 = [-0.5, 0[$ and $A_2 = [0, 0.5]$, using the arrangement with repetition, the number of ANN being trained will be $T = s^n = (2^6) = 128$, taking into account that SS will be call $S = T \times n = 768$ times both strata. Finally, the best set of first weights is discovered when the error rate of the test phase is closer to zero.

The main benefits of this new method are: fastness and easiness; stratified samples with high quality; flexibility to work with several range of weights; reduction of the sample size; reaches satisfactory outcomes.

## 5 Experiments

### 5.1 Research Data and Experiment Design

The techniques presented in sections 3 and 4 were evaluated using nine heterogeneous data sets from the UCI[17] and DELVE [18] repositories. The data sets are shown in Table 1. The Letter, RingNorm, Splice, Titanic and TwoNorm datasets were set as classification, while the Abalone, Addd10, Boston and Hwang datasets are regression ones. In all of the experiments we have used a K-fold Cross-Validation method with five cycles to estimate the predictive accuracy of the techniques.

| Dataset | Attributes | Examples |
|---|---|---|
| Letter | 16 | 20000 |
| RingNorm | 20 | 7400 |
| Splice | 60 | 3190 |
| Titanic | 3 | 2201 |
| TwoNorm | 20 | 7400 |
| Abalone | 8 | 4177 |
| Addd10 | 10 | 9792 |
| Boston | 13 | 506 |
| Hwang | 11 | 13600 |

Table 1: Data sets used in the experiments.

Three sets of experiments (tests) were devised in order to produce a fair comparison. First we construct the ANN we have experienced user choice of its structure and hand-tuned its parameters. This experiment produced the *base-line* results, the results with which our proposed technique's results are compared. The second experiment uses GA to choose the structure and tune the ANN parameters automatically. In the last experiment Stratified Sampling (SS) was used to improve on ANN basic training.

In the first experiment the ANN was set by hand-tuning. ANN has been set to: three layers; backpropagation learning algorithm; random weights initialisation from $[-0.5, +0.5]$; five neurons set to sigmoid transfer function in the hidden layer, and bias set to value 1; one neuron set to sigmoid transfer function in the output layer, and bias set to value 1; learning rate, momentum rate and steepness rate set to $0.8$, $0.2$ and 1, respectively; stop the training phase when the error rate gets below $0.1$ or the training epochs reaches 50.

In the second experiment the ANN was set by using a GA-based *wrapper*. GA has been set to: 50 candidates; 30 generations; mutation probability set to $1\%$; crossover probability of $90\%$; population replacement percentage set to $25\%$. Consequently, the ANN has been set to: back-propagation learning algorithm; random weights initialization from $[-0.5, +0.5]$; the limit of three times more neurons in the hidden layer than in the input layer; one out of seven transfer functions in each hidden and output neurons, and bias set to value 1; one neuron in the output layer, and bias set to value 1; learning rate, momentum rate and steepness rate set to respective default internal range; stop the training phase when the error rate achieves $0.1$ or the training epochs achieve 50.

In the third experiment the ANN was set by using a Stratified Sampling approach to split the set of weights in two intervals $A_1 = [-0.5, 0[$ and $A_2 = [0, +0.5]$. Other ANN parameters has been set using the parameter values of the first experiment.

## 5.2 Experimental Results

The experimental results are reported in tables 2 to 4. It is presented both the average error rate and std. deviation of training and test datasets. The average represents the obtained result from the arithmetic sum of five cycles (K-fold technique) of the same dataset together and then dividing the total by the number of cycles. The winner result percentage of each dataset was obtained by variance coefficient between two respective techniques. For the sake of clearness, all average error rate and std. deviation values presented in tables 2 to 4 are in the order of $1E - 10$.

From tables 2 to 4, the following conclusions can be drawn. First, it is worth noting that all the tables listed in this study show good results. Every technique which used the wrapper reduced the error rate. Second, as can be observed in Table 2, the use of GA turns out to be better in eight cases out of nine due to the structural risk minimization principle of ANN, such as local minima and over fitting. Third, there were good results in the Stratified Sampling approach, as presented in Table 3. This approach wins in six cases out of nine, in which only the ANN connection weights is balanced. Finally, the use of Stratified Sampling is better in six cases out of nine, as shown in Table 4, and it is an efficient technique in reducing ANNs error rates.

| **Data set** | **ANN hand-tuning test1(T1)** | **ANN with GA test2(T2)** | **Winner** |
|---|---|---|---|
| Letter | 123.6(13.8) | 102.6(1.9) | T2(9.3%) |
| RingNorm | 981.7(123.0) | 267.9(10.7) | T2(8.5%) |
| Splice | 16.8(2.2) | 16.8(0.6) | T2(9.5%) |
| Titanic | 11.3(0.9) | 9.5(0.5) | T2(3.3%) |
| TwoNorm | 475.2(257.2) | 141.8(23.5) | T2(38%) |
| Abalone | 99.9(7.5) | 70.1(5.1) | T2(0.2%) |
| Addd10 | 31223(950) | 30460(855) | T2(0.2%) |
| Boston | 107982(27495) | 84110(17309) | T2(4.9%) |
| Hwang | 1342.8(15.9) | 1174.6(24.6) | T1(0.9%) |

Table 2: Comparing hand-tuned ANN with ANN tuned by a GA. The values in the table were all multiplied by $1E - 10$.

| **Data set** | **ANN hand-tuning test1(T1)** | **ANN with SS test3(T3)** | **Winner** |
|---|---|---|---|
| Letter | 123.6(13.8) | 109.2(1.6) | T3(9.7%) |
| RingNorm | 981.7(123.0) | 238.0(7.0) | T3(9.6%) |
| Splice | 16.8(2.2) | 16.5(2.3) | T1(0.6%) |
| Titanic | 11.3(0.9) | 9.2(0.3) | T3(4.5%) |
| TwoNorm | 475.2(257.2) | 120.2(3.3) | T3(51%) |
| Abalone | 99.9(7.5) | 77.9(12.9) | T1(9.1%) |
| Addd10 | 31223(950) | 30445(910) | T3(0.1%) |
| Boston | 107982(27495) | 95979(95691) | T3(12%) |
| Hwang | 1342.8(15.9) | 1354(24.42) | T1(0.6%) |

Table 3: Comparing hand-tuned ANN with ANN tuned by SS. The values in the table were all multiplied by $1E - 10$.

When comparing the results obtained from the different experiments listed above, we saw that ANN hand-tuning won four cases whereas ANN with GA won eleven cases and ANN with SS had a dominant portion of twelve cases. The third experiment demonstrates that Stratified Sampling is the most feasible and reliable way to get an improved tuning of ANN parameters. In conclusion, the SS technique demonstrates that a balanced connection weight has a big influence on the development of reliable ANNs.

| **Data set** | **ANN with GA test2(T2)** | **ANN with SS test(T3)** | **Winner** |
|---|---|---|---|
| Letter | 102.6(1.9) | 109.2(1.6) | T3(0.5%) |
| RingNorm | 267.9(10.7) | 238.0(7.0) | T3(1.1%) |
| Splice | 16.8(0.6) | 16.5(2.3) | T2(10.1%) |
| Titanic | 9.5(0.5) | 9.2(0.3) | T3(1.2%) |
| TwoNorm | 141.8(23.5) | 120.2(3.3) | T3(13.9%) |
| Abalone | 70.1(5.1) | 77.9(12.9) | T2(9.3%) |
| Addd10 | 30460(855) | 30445(910) | T2(0.2%) |
| Boston | 84110(17309) | 95979(95691) | T3(6.6%) |
| Hwang | 1174.6(24.6) | 1354(24.42) | T3(0.3%) |

Table 4: Comparing ANN tuned by GA with ANN tuned by SS. The values in the table were all multiplied by $1E - 10$.

## 6 Conclusion

In this paper two techniques to fine tune ANNs using AFRANCI have been described. The techniques show that ANNs can be tuned using either a Genetic Algorithm (GA) or a Stratified Sampling (SS) to achieve good results. Under a wrapper, GA builds the best ANN structure by choosing the correct transfer functions, the right biases, and other essential ANN parameters. On the other hand, SS balances the connection weights of an ANN structure hand-tuned. Since it is easy to implement, GA is the most used fine tuning technique to draw the best ANN structure. However, empirical evaluation on nine popular data sets has confirmed that the SS obtained better results with the lowest error rates. Thus, the evaluation of the proposed tune suggests that better results can be achieved when the SS is used.

**Acknowledgements:** The research was supported by the University of Porto(PT) and UnilesteMG(BR) in the case of the first author, it was also supported by FCT under grant # POSI/EIA/63240/2004.

*References:*

[1] Goldberg, D.E.: Genetic Algorithms in Search, Optimization, and Machine Learning. Addison-Wesley Professional (1989)

[2] Prados, D.: New learning algorithm for training multilayered neural networks that uses genetic-algorithm techniques. Electronics Letters **28**(16) (30 JUL 1992) 1560–1561

[3] Harp, S.A., Samad, T., Guha, A.: Towards the genetic synthesis of neural network. In: Proceedings of the third international conference on Genetic algorithms, San Francisco, CA, USA, Morgan Kaufmann Publishers Inc. (1989) 360–369

[4] Whitley, D.L.: Cellular genetic algorithms. In: Proceedings of the 5th International Conference on Genetic Algorithms, San Francisco, CA, USA, Morgan Kaufmann Publishers Inc. (1993)

[5] Shamseldin, A.Y., Nasr, A.E., OConnor, K.M.: Comparison of different forms of the multi-layer feed-forward neural network method used for river flow forecasting. Hydrology and Earth System Sciences (HESS) **6**(4) (2002) 671–684

[6] Fausett, L.V.: Fundamentals of neural networks: architectures, algorithms, and applications. Prentice-Hall, Inc., Upper Saddle River, NJ, USA (1994)

[7] Nguyen, D., Widrow, B.: Improving the learning speed of 2-layer neural networks by choosing initial values of the adaptative weights. Volume 3. (1990) 21–26

[8] Erdogmus, D. Fontenla-Romero, O.P.J.A.B.A.C.E.J.R.: Accurate initialization of neural network weights by backpropagation of the desired response. In: Proceedings of the International Joint Conference on Neural Networks. Volume 3., IEEE (2003) 2005–2010

[9] Reinaldo, F., Siqueira, M., Camacho, R., Reis, L.P.: Multi-strategy learning made easy. WSEAS Transactions On Systems, Greece **5**(10) (2006) 2378–2384 ISSN 1109-2777.

[10] Reinaldo, F., Certo, J., Cordeiro, N., Reis, L.P., Camacho, R., Lau, N.: Applying biological paradigms to emerge behaviour in robocup rescue team. In Bento, C., Cardoso, A., Dias, G., eds.: EPIA. Volume 3808 of Lecture Notes in Computer Science., Springer (2005) 422–434 ISSN: 03029743.

[11] Reinaldo, F., Siqueira, M., Camacho, R., Reis, L.P.: A tool for multi-strategy learning. Research in Computer Science **26** (2006) 51–60

[12] Witten, I.H., Frank, E.: Data Mining: Practical machine learning tools and techniques. 2nd. edn. Morgan Kaufmann, San Francisco (2005)

[13] Clark, P., Niblett, T.: The cn2 induction algorithm. Mach. Learn. **3**(4) (1989) 261–283

[14] MIT, Wall, M.: Galib: a library of genetic algorithm components. http://lancet.mit.edu/ga/ (2006)

[15] John, H.G.: Cross-validated c4.5: Using error estimation for automatic parameter selection. Technical note stan-cs-tn-94-12, Computer Science Department, Stanford University, California (1994)

[16] Holland, J.H.: Adaptation in Natural and Artificial Systems. University of Michigan Press (1975)

[17] D.J. Newman, S. Hettich, C.B., Merz, C.: UCI repository of machine learning databases (1998)

[18] : DELVE data for evaluating learning in valid experiments (2003)