Bayesian Evolution of Rich Neural Networks

Matteo Matteucci
Department of Electonics and Information
Politecnico di Milano – Milan, Italy
E-mail: matteucc@elet.polimi.it

Dario Spadoni
Advanced Learning And Research Institute
Università della Svizzera Italiana – Lugano, Switzerland
E-mail: spadoni@alari.ch

Abstract— **In this paper we present a genetic approach that uses a Bayesian fitness function to the design of rich neural network topologies in order to find an optimal domain-specific non-linear function approximator with good generalization performance. Rich neural networks have a feed-forward topology with shortcut connections and arbitrary activation functions at each layer. This kind of topologies is particularly well suited for non-linear regression tasks, but it may suffer for overfitting issues. In this paper we present a Bayesian fitness function to effectively apply genetic algorithms with these models obtaining, in a completely automated way, models well-matched to the problem, with good generalization capability, and low complexity.**

I. Introduction

Artificial neural networks have proved to be excellent approximator of generic non-linear functions and have been extensively used for various purposes like regression, classification and feature reduction. A neural network is a collection of basic units computing a non-linear function of their input. Every input has a weight assigned to it that determines the impact it has on the output of the node. The structure of neurons and their interconnections, known as the network's topology, together with the connection weights, determine the neural network final behavior. In this paper, we focus on feed–forward topologies with arbitrary non–linear, differentiable activation functions for each layer and "shortcut" connections linking two non–subsequent layers. We name this kind of enriched topologies as *Rich Neural Networks* [7].

Although optimization techniques based on back-propagation allow to change the value of the weights to minimize the network regression/classification error, the topology of a neural network has a critical role in whether or not the network can be trained to learn a particular set of data. Clearly, the simpler the topology, the simpler the functions the neural network can approximate; however an over-complex neural network may overfit the data and lose its generalization capability.

There is no known way to systematically create an optimal or even near-optimal topology for a specific task automatically; this is usually solved by means of heuristic algorithms or left to human experts. Moreover the correct topology is application-dependent, so any method that aims to create the correct topology should be data-driven. To face the problem of finding an application-specific network topology, in this paper we introduce a genetic algorithm which explores the space of possible rich topologies using a *Bayesian* fitness.

II. Genetic algorithms for Neural Networks

The problem of finding an optimal neural network architecture can be thought of as a search problem, where the search space is the space of all possible rich topologies, and the goal is to minimize an error function (while avoiding overfitting). To do this, there is a range of heuristic algorithms that modify the structure of a network in a hill-climbing fashion. *Constructive* algorithms start with a small neural network and then add nodes and connections during training. This type of structure search, however, has been found to very susceptible to settling into local optimal [2]. More often a *destructive* approach is used: a big neural network is first trained on the data, and then pruned to increase its generalization capability while preserving its accuracy [4].

Genetic algorithms have been proved to be a powerful search tool when the search space is large, multimodal, and when it is not possible to write an analytical form for the error function to minimize. In these applications, genetic algorithms excel because they can simultaneously and thoroughly explore many different parts of a large solution space seeking for a suitable solution.

At first, completely random solutions are tried, evaluated according to a fitness function, and then the best ones are combined using specific operators. This allows to adequately explore possible solutions while at the same time preserving parts of each solution which are known to work in a proper way. Genetic algorithms have been used to optimize almost all the parameters that characterize a neural network (e.g. weights, learning algorithm, topology) as can be seen from the surveys on the topic [1], [9].

The algorithm we present in this paper, ELeaRNT (Evolutionary Learning of Rich Neural network Topologies), evolves only the network architecture and the functions computed by the different neurons; weights optimization is performed by conjugate gradient descent algorithm. Our main concern is limiting the complexity of the model learnt by the algorithm to reduce overfitting by using a Bayesian fitness function which rewards the most probable network, that is the one which maximizes the posterior probability induced by the data from a prior probability.

Since the Bayesian fitness introduces a probabilistic Occam's razor, the selected network has a good generalization capability and a complexity suitable for the specific problem (neither too complex or too simple).

0-7803-8359-1/04/$20.00 ©2004 IEEE

III. Bayesian Framework Application to Artificial Neural Networks

The Bayesian framework introduced by MacKay [5], [6] provides a practical and powerful way to improve the generalization capability of neural networks and minimize their complexity. The entire framework is centered on Bayes'theorem and on the joint and the prior probability distributions on the entities we are interested in (i.e. network weights, activation functions): more precisely, the prior distribution is combined through the Bayes'rule with the so called *evidence* (that is the probability to observe the data given the model) to compute the posterior probability of the entity of interest.

By giving neural networks a probabilistic interpretation, the problem of selecting a suitable network topology for a particular problem can be considered as a two-level inference. At the first level, every model under consideration is fit to the data by adequately setting its parameters (network weights) using standard optimization techniques such as back-propagation. At the second level of inference, different models are compared and ranked according to their posterior probability obtained by combining their *evidence* with their prior probability through the Bayes'theorem. The prior distribution has a key role in the Bayesian framework since it allows to express in probabilistic terms, a preference towards a particular set of models or introduce expert knowledge into the design of neural networks.

Besides providing an unifying framework for neural network design, Bayesian probability theory can also face the overfitting problem. Indeed, the evidence embodies the "Occam's razor", that is the principle which favors the simplest theory among all the ones describing a particular phenomenon. Therefore, within the Bayesian framework over-complex networks (which will be more likely to overfit the training set) are automatically penalized and the model selection process will look for the simplest network among all the ones compatible to the given data set.

Let us now consider the problem of training a neural network with a given architecture which maps an input $\mathbf{x}$ to an output $y(\mathbf{x}|\mathbf{w})$. This network is trained using a data set $\mathcal{D}$, consisting of N patterns of the form $(\mathbf{x}, t)$, by iteratively adjusting $\mathbf{w}$ as to minimize an objective function, such as the squared error sum:

$$E_D(\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{N}\left(t^{(i)} - y(\mathbf{x}^{(i)}|\mathbf{w})\right)^2. \quad (1)$$

This minimization process is based on repeated evaluations of the gradient of E_D using the back-propagation algorithm. We can give to this classical neural network learning process a probabilistic interpretation where the error function is interpreted as minus the log likelihood of a model M corrupted by noise:

$$p(\mathcal{D}|\mathbf{w}, \beta, M) = \frac{1}{Z_D(\beta)}\exp(-\beta E_D), \quad (2)$$

where

$$Z_D(\beta) = \left(\frac{2\pi}{\beta}\right)^{\frac{N}{2}}. \quad (3)$$

The use of the sum-squared error E_D in Equation (1) thus corresponds to an assumption of a Gaussian noise on the target variables, and β defines a noise level $\sigma_\nu^2 = 2/\beta$. If a regularization term such as weight decay is included, then the objective function changes to:

$$S(\mathbf{w}) = \beta E_D + \alpha E_W \quad (4)$$

where $E_W = \frac{1}{2}\sum_i^W w_i^2$ and W is the number of weights and biases in the network. The weight decay term in Equation (4) can be interpreted as a log prior probability distribution over the parameters (again Gaussian with variance $\sigma_w^2 = 2/\alpha$):

$$p(\mathbf{w}|\alpha, M) = \frac{1}{Z_W(\alpha)}\exp(-\alpha E_W), \quad (5)$$

where

$$Z_W(\alpha) = \left(\frac{2\pi}{\alpha}\right)^{\frac{W}{2}}. \quad (6)$$

Under these assumptions, we can consider the generic objective function $S(\mathbf{w})$ as an *inference* of parameters $\mathbf{w}$ given the data:

$$p(\mathbf{w}|\mathcal{D}, \alpha, \beta, M) = \frac{p(\mathcal{D}|\mathbf{w}, \beta, M)\cdot p(\mathbf{w}|\alpha, M)}{p(\mathcal{D}|\alpha, \beta, M)} = \frac{1}{Z_S}\exp(-S(\mathbf{w})). \quad (7)$$

The $\mathbf{w}$ found by (locally) minimizing $S(\mathbf{w})$ is thus interpreted as the (locally) most probable parameter vector $\mathbf{w}_{\text{MP}}$ given the Gaussian model for the noise, the Gaussian prior over the parameter space and the observed data.

A. The Evidence Framework for α and β

So far, we have assumed that the values of the hyper-parameters α and β are fixed and known. Unfortunately, in many applications, we have little idea of suitable values for α and β. Recalling [5], [6], we need to apply Bayesian techniques also to infer the most probable values α_{MP} and β_{MP} for the hyper-parameters. To infer α and β given the data, we apply again the rules of probability theory:

$$p(\alpha, \beta|\mathcal{D}, M) = \frac{p(\mathcal{D}|\alpha, \beta, M)p(\alpha, \beta|M)}{p(\mathcal{D}|M)}. \quad (8)$$

Assuming we have a little idea of suitable values for α and β, since the denominator in (8) is independent of α and β, the maximum-a-posterior values for these hyper-parameters are found by maximizing the term $p(\mathcal{D}|\alpha, \beta, M)$, which is called the 'evidence' for α and β. If we approximate the posterior probability distribution (7) by a single Gaussian,

$$p(\mathbf{w}|\mathcal{D}, M) \simeq \frac{1}{Z_S^*}\exp(-S(\mathbf{w}) - \frac{1}{2}\triangle\mathbf{w}^T\mathbf{A}\triangle\mathbf{w}), \quad (9)$$

where $A = \nabla\nabla \ln p(\mathbf{w}|\mathcal{D}, \alpha, \beta, M)|_{\mathbf{w}_{\text{MP}}}$, and $\triangle\mathbf{w} = (\mathbf{w} - \mathbf{w}_{MP})$, we can write the evidence for α and β as:

$$\ln p(\mathcal{D}|\alpha,\beta,M) = \ln \frac{Z_S^*}{Z_D(\beta)Z_W(\alpha)} =$$

$$= -S(\mathbf{w}_{\text{MP}}) - \frac{1}{2}\ln\det\left(\frac{A}{2\pi}\right) - \ln Z_W(\alpha) - \ln Z_D(\beta), \quad (10)$$

and using equations (2) and (5) we can write the log of the evidence as:

$$\begin{aligned}\ln p(\mathcal{D}|\alpha,\beta) &= -S(\mathbf{w}_{\text{MP}}) - \frac{1}{2}\ln|A| + \\ &+ \frac{W}{2}\ln\alpha + \frac{N}{2}\ln\beta - \frac{N}{2}\ln(2\pi). \quad (11)\end{aligned}$$

Denoting with λ_i the eigenvalues of the Hessian $H = \beta\nabla\nabla E_D$, the maximum of evidence for α and β satisfies the following implicit equations:

$$2\alpha E_W^{MP} = W - \sum_{i=1}^{W}\frac{\alpha}{(\lambda_i+\alpha)} = \gamma, \quad (12)$$

$$2\beta E_D^{MP} = N - \sum_{i=1}^{W}\frac{\lambda_i}{\lambda_i+\alpha} = N - \gamma, \quad (13)$$

where

$$\gamma = \sum_{i=1}^{W}\frac{\lambda_i}{\lambda_i+\alpha}. \quad (14)$$

In a practical implementation, we need to find the optimum α and β as well as $\mathbf{w}_{\text{MP}}$. A simple solution to this problem is to use a standard iterative training algorithm to find $\mathbf{w}_{\text{MP}}$. We train the network using some initial values assigned to the hyper-parameters to find $\mathbf{w}_{\text{MP}}$; this is done by periodically re-estimating new α and β using:

$$\alpha^{\text{new}} = \gamma/2E_W \quad (15)$$

$$\beta^{\text{new}} = (N-\gamma)/2E_D. \quad (16)$$

B. The Bayesian Fitness Function

Once determined the most probable values for the weight vector $\mathbf{w}$ and hyper-parameters α and β of a given neural network, we can now compare different networks. In order to evaluate a given neural network, we introduce the following function, which is directly derived from (11):

$$\ln p(D|M_k) = -\alpha_{\text{MP}}E_W^{\text{MP}} - \beta_{\text{MP}}E_D^{\text{MP}} - \frac{1}{2}\ln|\mathbf{A}| +$$

$$+\frac{1}{2}\ln(\frac{2}{\gamma}) + \frac{W}{2}\ln\alpha_{\text{MP}} + \frac{N}{2}\ln\beta_{\text{MP}} + \frac{1}{2}\ln(\frac{2}{N-\gamma}). \quad (17)$$

We name this expression *bayesian fitness function* and we will use it in our genetic algorithm to perform model comparison. Using this fitness function to search fitting models for our dataset, we expect that, thanks to the Occam's razor embodied in the Bayesian framework, complex networks will be automatically penalized while small ones will be favoured reducing overfitting through regolarization.

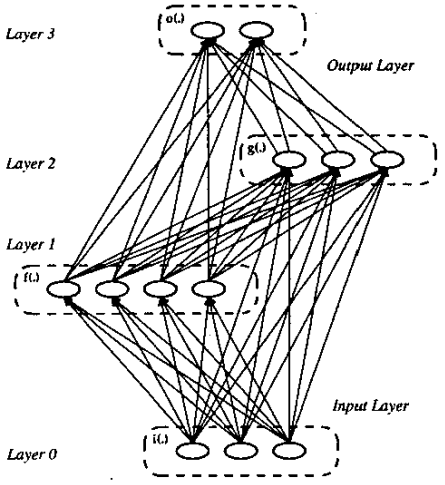


Fig. 1. A phenotype evolved by our genetic algorithm

IV. The ELeaRNT Genetic Algorithm

ELeaRNT [7], [8] is a genetic algorithm which evolves RNN topologies in order to find an optimal domain–specific non–linear function approximator with a good generalization performance. ELeaRNT follows the schema of Goldberg's *Simple Genetic Algorithm* [3].

ELeaRNT uses a direct coding schema to represent a network, that is every detail of the architecture (i.e., number of neurons, activation functions, connections, learning algorithm, etc.) is specified in the genotype: this allows a more focused design of the genetic operators that result to be closed with respect to the chosen phenotype.

In RNNs each layer has at least one neuron, and, potentially, a different activation function. The number of neurons in the first and last layers is fixed, since that is the number of input and output variables of the specific problem. The transfer function for the input layer is usually the *identity function* and for the other layers it can be any of the following: *identity*, *logistic*, *tanh*, *linear*, *gaussian*, *sin*, *cos*. All the neurons in the same layer have the same activation functions and there are no intra–layer connections. This phenotype subsumes the classical fully connected feed–forward architecture and exploits more flexibility due to the use of various activation functions and to the capability of describing non–fully connected topologies with shortcut connections. Figure 1 shows an example of the RNN topology evolved by our algorithm.

Each phenotype is coded by a two part genotype. The first part encodes the layer information (i.e. number of neurons and activation function), and the second part encodes the connectivity of the network. To specify a proper feed–forward neural network, only the elements above the diagonal in the connectivity matrix can differ from 0. It is possible that during the genetic evolution a genotype codes an "invalid" phenotype. That happens when either a column (i.e., the fan–in of a neuron layer) or a row (i.e., the fan–out of a neuron layer) is filled with 0s implying that a layer of neurons is not reachable from

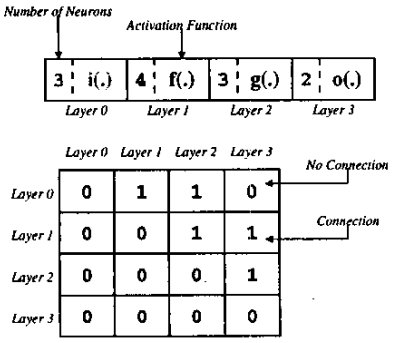


Fig. 2. The genotype for the network in Figure 1

the input or does not participate in the final output. To avoid this issue, we have specifically designed the genetic operators to be closed with respect to the phenotype family.

A. Genetic Operators

We define two crossover operators, and six different mutation operators. Crossover and mutation occurrences have different probabilities, and each crossover or mutation operator has uniform probability once the application of a specific genetic operation has been chosen. Here, we briefly introduce these operators; for a detailed description see [7], [8].

- Single Point Crossover: this operator combines two networks by cutting their topologies in two pieces with a cutting surface that entirely separates the input and the output of the network and then switching the input parts of the two networks.
- Two Point Crossover: this operator combines two networks by extracting a subgraph from each of them, and exchanging these subgraphs between the networks.

In order to guarantee these operators to be closed with respect to the valid genotype family, we have to restore all the connections between the remaining network and the new block obtained by applying a crossover to a neural network. We implemented six different mutation operators; here they are briefly described: for a more detailed description we remand again to [7], [8]:

- Drop node: this operator randomly selects a layer and removes it from the network structure. Before doing that, each input connection of the layer to remove is connected to all the destination of its output connections.
- Add node: this operator adds a layer to the network topology. An existing layer is randomly selected and its connectivity is duplicated. After that, a random activation function and a different number of neurons are initialized.
- Number of neurons: this operator, changes the number of neurons in a specific layer of the network.
- Drop connection: this operator removes a connection from the connectivity matrix of the network.
- Add connection: this operator adds a new connection in the connectivity matrix of the network.
- Activation function: this operator changes the activation function in a network layer.

B. ELeaRNT and the Bayesian Fitness Function

The complete ELeaRNT algorithm is shown in Algorithm 1. At first ELeaRNT generates, an initial population of random neural networks and evaluates each individual using the *Evaluate* procedure presented in Algorithm 2. This procedure initializes the weight vector $\mathbf{w}$ according to a Gaussian prior distribution which penalizes big value of the weights which may cause the network function to have large curvatures and thus to overfit.

After the initialization steps, each individual is trained using standard optimization techniques in order to minimize the regularized error function of Equation (4). After a given number of training epochs, hyper-parameters α and β are re–estimated by using the formulae of Equation (15) and (16) with γ given by Equation (14). After the training phase, each individual is evaluated using the Bayesian fitness function; to avoid local minima, initialization, training and evaluation are performed several times using batch learning with the conjugate gradient descent and the average performance over the different restarts

Algorithm 1 ELeaRNT Algorithm

Begin ELeaRNT
Create a Population P with N Random Individuals
for all $i \in P$ **do**
  Evaluate(i)
end for
repeat
  repeat
    Select i_1 and i_2 according to their Fitness
    with probability p_{cross}
      Select Crossover Operator **Cross**
      $i'_1, i'_2 \leftarrow \mathbf{Cross}(i_1, i_2)$
    otherwise {with probability $1 - p_{cross}$}
      $i'_1 \leftarrow i_1$ and $i'_2 \leftarrow i_2$
    end with
    with probability p_{mut}
      Select Mutation Operator **Mut**
      $i''_1 \leftarrow \mathbf{Mut}(i'_1)$
    otherwise {with probability $1 - p_{mut}$}
      $i''_1 \leftarrow i'_1$
    end with
    with probability p_{mut}
      Select Mutation Operator **Mut**
      $i''_2 \leftarrow \mathbf{Mut}(i'_2)$
    otherwise {with probability $1 - p_{mut}$}
      $i''_2 \leftarrow i'_2$
    end with
    Add Individuals i''_1 and i''_2 to New Population
  until (Created a new Population P')
  for all $i \in P'$ **do**
    Evaluate(i)
  end for
until (Desired number of generation reached)
End ELeaRNT

Algorithm 2 Evaluate Algorithm

Begin Evaluate (*Individual i*)
repeat
Initialize weights according to a Gaussian distribution
Choose initial values for the hyper-parameters
Train i to minimize the error function (4)
Every M epochs re–estimate α, β using (15) and (16)
until Desired number of restarts reached
Return the average fitness over the restarts
End Evaluate

is used. This choice is not mandatory and we could simply use the best performance. Average performance is used to reduce the noise on the fitness function by smoothing; the best restert is kept as best individual in implementing *elitism.*

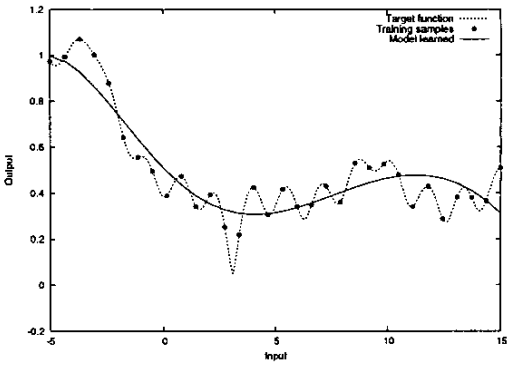
After the evaluation of the population, selection takes place. After selection, the crossover and mutation operators described in the previous sections are applied to the selected individuals and a new generation is created. This new population will be trained and evaluated as the previous one and the algorithm will cycle for a given number of generations or until a given stopping criterion is met.

V. ELEARNT EXPERIMENTAL VALIDATION

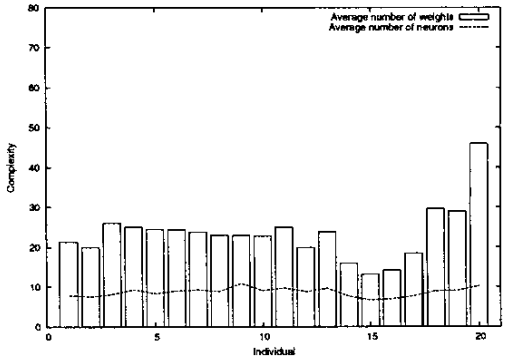
In this section we validate the ELeaRNT algorithm by applying it in regression tasks. In order to prove the effectiveness of the Bayesian fitness, we use an improper prior over the network space, that is we assume that all the network topologies have the same prior probability. Using this improper prior for the experiments we are interested in:

- proving that ELeaRNT with the Bayesian fitness function is able to select rich neural networks well-matched to the data and with good generalization capability
- verifying the effectiveness of Occam's razor, embodied in the Bayesian fitness function, in favoring simple models over complex ones even without specifying a prior.

In all the experiments we use a population with 20 individuals, with crossover and mutation probability respectively $p_{cross} = 0.75$, $p_{mut} = 0.25$ and network weights have been initialized according to a Gaussian distribution. The choice of the initial values for hyper-parameters α and β is quite critical. In general, α determines the impact of the prior over the network weights, and thus determines the extent to which complex networks are penalized over simple ones. Conversely, β determines the impact of the network error on the training data, hence small values of β will result in networks which poorly approximate the training set. We have initialized β to a value larger than expected while we have assigned α a small value: in this way, network overfitting is initially favored to avoid poor solutions; then, as the network is trained and hyper-parameters are re-estimated over-complex networks will be penalized. All the results are averaged over 10 runs of the algorithm using different randomization seeds.



(a)



(b)

Fig. 3. Best model (a) and average number of parameters (b) in population

A. Benchmark 1

In this experiment, we use a small number of samples (32) extracted by sampling regularly in $[-5, 15]$ the function

$$y(x) = \sqrt{0.1 \sin(2x)^2 + \frac{2\arctan(x-3)^2}{(x+7)(cos(x)+2)}}. \qquad (18)$$

In this test case, we evolved 20 individuals for 25 generations and re-estimated the hyper-parameters every 20 epochs of training. In this benchmark the performance of the approximation is affected by the few examples in the training set and by the fact that they are not fully representative of the real function to be approximated. However, the algorithm finds a reasonable non-linear approximator for the training set, as shown in Figure 3(a). The average complexity of the population after the genetic evolution with the models ordered in descending order with respect to their fitness is reported in Figure 3(b). As we can see, the average complexity of each individual is quite small, and not considering the two worst cases, each network has less than 30 weights (corresponding to an average of 9 hidden neurons). In this case, ELeaRNT has proved to be able of finding minimal rich neural networks with both good accuracy and small topologies, (even when only few patterns of the function to approximate are available).

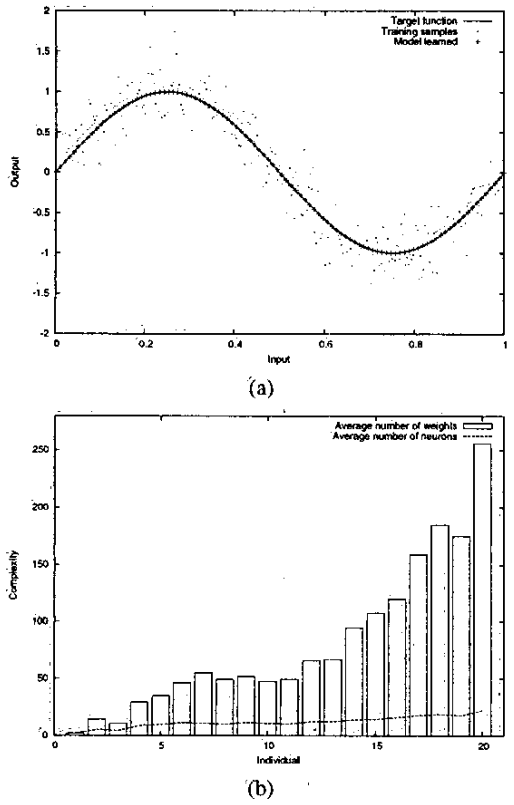


Fig. 4. Best model (a) and average number of parameters (b) in population

B. Benchmark 2

In this experiment, the algorithm has to find the correct model to fit the noisy sinusoidal training set given by

$$y = \sin(2\pi x) + \epsilon \qquad \epsilon \sim N(0, 0.04) \quad x \in [0, 1].$$

We are interested in this example since the process generating the data belongs to the family of models that can be represented by using a RNN in its most simple topology: a single neuron with a sinusoidal activation function. Our goal is to check if ELeaRNT is able to approximate this data set by selecting a single neuron network with a sinusoidal activation function. The training set is composed of 300 samples drawn uniformly from the $[0, 1]$ interval and the added noise ϵ is Gaussian with 0 mean and variance $\sigma^2 = 0.04$. Figure 4(b) shows the average complexity of the population after the genetic evolution. Here, models are ordered in descending order with respect to their fitness. If we look at the best model found by the algorithm, that is the individual 1, we can see how, in all the different executions, ELeaRNT selects a single neuron model with a sinusoidal activation function: the most simple network able to fitting the data set. To give an idea of convergence rates for the ELeaRNT algorithm, Figure 5 reports the fitness value during learning (averaged over 10 executions) for this benchmark. The best individual converges to the maximum fitness in less than 5 generations while the fitness for the worst individual increases over generations even if it has large oscillations.

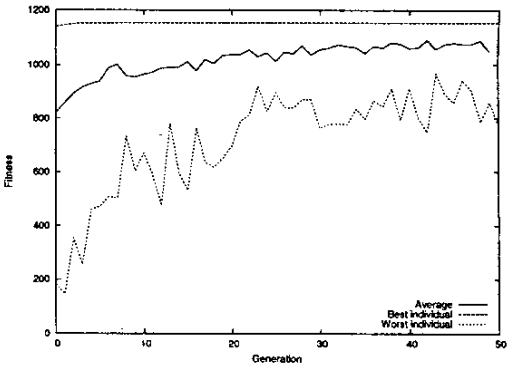


Fig. 5. Bayesian fitness as a function of generations

VI. Conclusions and Future Work

In this paper we have presented a genetic algorithm which evolves rich neural network topologies in order to find an optimal domain-specific non-linear function approximator with a good generalization performance. The algorithm uses a Bayesian fitness function which allows to automatically penalize over-complex models and select networks with a suitable complexity for the problem in exam (thus avoiding overfitting).

This algorithm has been tested on different cases and it has proved to select relatively simple models which are well matched to the data and have good generalization capacity. Further work should be done in using the whole final population to implement a committee of models instead of using only the best individuals. In this case, it might be useful to increase the independence between the neural networks in the final population by adding *speciation* to the evolutionary process in order to pick more areas with high probability at the same time and avoid genetic drifting.

References

[1] K. Balakrishnan and V. Honavar. Evolutionary Design of Neural Architectures: A Preliminary Taxonomy and Guide to Literature. Technical report, Dep. of Computer Science, Iowa State University, Iowa, 1995.

[2] C. Campbell. Constructive learning techniques for designing neural network systems. In CT Leondes, editor, *Neural Network Systems Technologies and Applications*. Academic Press, San Diego, CA, 1997.

[3] D. E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley, Reading, MA, 1989.

[4] Y. LeCun, J. Denker, S. Solla, R.E. Howard, and L.D. Jackel. Optimal brain damage. In D. S. Touretzky, editor, *Advances in Neural Information Processing Systems II*, San Mateo, CA, 1990. Morgan Kauffman.

[5] D.J.C. MacKay. Probable networks and plausible predictions — a review of practical Bayesian methods for supervised neural networks. *Network: Computation in Neural Systems*, 6:469–505, 1995.

[6] D.J.C. MacKay. Comparison of approximate methods for handling hyperparameters. *Neural Computation*, 11(5):1035–1068, 1999.

[7] M. Matteucci. ELeaRNT: Evolutionary learning of rich neural network topologies. Technical Report CMU-CALD-02-103, Carnegie Mellon University, Pittsburgh, PA, 2002.

[8] M. Matteucci. *Evolutionary Learning of Adaptive Models Within a Bayesian Framework*. PhD thesis, Dipartimento di Elettronica e Informazione, Politecnico di Milano, 2002.

[9] X. Yao. *A Review of Evolutionary Artificial Neural Networks*. Commonwealth Scientific and Industrial Research Organiz., Australia, 1992.