Int. J. Appl. Math. Comput. Sci., 2004, Vol. 14, No. 3, 423–440

EVOLUTIONARY LEARNING OF RICH NEURAL NETWORKS IN THE BAYESIAN MODEL SELECTION FRAMEWORK

MATTEO MATTEUCCI[*], DARIO SPADONI[**]

[*] Department of Electronics and Information
Politecnico di Milano, Piazza L. da Vinci 32, 20133 Milan, Italy
e-mail: matteucci@elet.polimi.it

[**] ALaRI (Advanced Learning and Research Institute)
University of Lugano, Lugano, Switzerland
e-mail: spadoni@alari.ch

In this paper we focus on the problem of using a genetic algorithm for model selection within a Bayesian framework. We propose to reduce the model selection problem to a search problem solved using evolutionary computation to explore a posterior distribution over the model space. As a case study, we introduce ELeaRNT (Evolutionary Learning of Rich Neural Network Topologies), a genetic algorithm which evolves a particular class of models, namely, Rich Neural Networks (RNN), in order to find an optimal domain-specific non-linear function approximator with a good generalization capability. In order to evolve this kind of neural networks, ELeaRNT uses a Bayesian fitness function. The experimental results prove that ELeaRNT using a Bayesian fitness function finds, in a completely automated way, networks well-matched to the analysed problem, with acceptable complexity.

Keywords: Rich Neural Networks, Bayesian model selection, genetic algorithms, Bayesian fitness

1. Introduction

Suppose we analyse some data $\mathcal{D}$ and we are interested in finding a set of models which might have generated the data: we would probably find an entire set of models $M_1, \dots, M_K$ with different complexity, all compatible with $\mathcal{D}$. *Model comparison* refers to the problem of using the available data to compare different models with respect to some quantity of interest. After having compared $M_1, \dots, M_K$, we might want to select one of the models M_k matching some requirements (i.e., best fitting the data, the lowest model complexity, the best generalization capability, etc.): this process is known as *model selection*. Model selection can be considered as a search in the space of models for the one which satisfies best a particular requirement. Often, this space is multi-modal, non-differentiable and large. Then, it is well suited to be explored by stochastic search algorithms or meta-heuristics such as *Genetic Algorithms* (GAs).

A central issue in choosing the most suitable model for a given problem is selecting the right structural complexity. Clearly, the simpler the model, the smaller the class of problems the model can solve: a model with too few parameters will not be flexible enough to approximate important features in $\mathcal{D}$, and thus will result in *underfitting* the data. On the other hand, an overly complex model may lose its generalization capacity, that is, the ability to give a good prediction on samples not seen during the training process. This loss of generalization is the result of *overfitting* the data set. In fact, instead of capturing the hidden structure of the data, excessively complex models may memorize the training dataset, thus having the ability to approximate only the data samples (eventually affected by noise). A simple example of this phenomenon is described in Fig. 1. Part (a) depicts a linear model $M_1(x)$ underfitting a dataset generated by a quadratic function $M_2(x)$ plus some noise. Part (b) depicts the overfitting of a polynomial model $M_N(x)$ that practically memorizes the dataset generated by the quadratic function $M_2(x)$ including the noise.

In the literature, several alternative techniques have been proposed to determine the right level of model complexity, from regularization theory (Tikhonov, 1963), where analytical constraints, usually involving smoothness, are introduced for the model to cross-validation (Stone, 1974), where part of the training dataset is used to estimate the model generalization error. In this paper, we focus on the Bayesian approach to model selection (Denison *et al.*, 2002; Bernardo and Smith, 1994) since it takes into account the uncertainty of selecting a particular model and gives a formal method to specify model

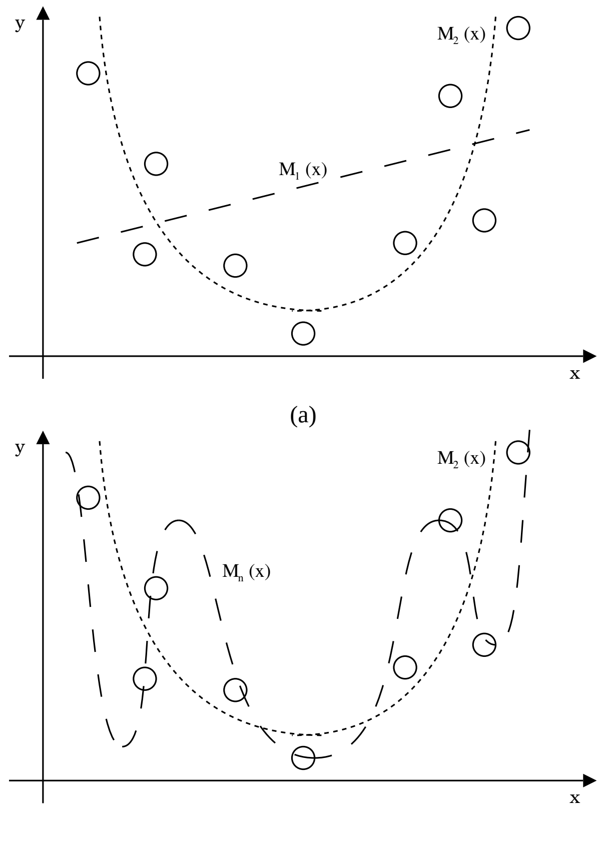


Fig. 1. Underfitting (a) and overfitting (b) on the training data.

requirements through the use of probability distributions. The Bayesian framework does not require to hold out any data and automatically provides a regularization term derived from prior probability distributions.

In the next section, we introduce the Bayesian framework from a theoretical point of view, while in Section 3 we will apply it to artificial neural networks. The following sections introduce ELeaRNT and present some empirical results to validate it.

2. Bayesian Model Comparison and Occam's Factor

The Bayesian framework for model selection (Denison *et al.*, 2002; Bernardo and Smith, 1994) provides a practical and powerful way to improve the generalization capabilities of models while minimizing their complexity. The framework is centered on the use of probability distributions over the model structure and model parameters combined according to Bayes' rule in order to compute its posterior distribution.

If we consider the classical notation $p(A|B,M)$ for conditional probabilities, the statements B and M list the conditional assumptions on which this measure of plausibility is based. For example, if A is "it will rain today", B is "the barometer is rising", and M is a model of the weather, then the quantity $p(A|B,M)$ is a number between 0 and 1 which expresses how probable we would think "rain today" is, given that the barometer is rising, and given the overall assumptions in M. This conditional probability is related to the joint probability of A and B:

$$p(A|B,M) = \frac{p(A,B|M)}{p(B|M)}.$$

Having enumerated the complete list of the conditional degrees of belief about the model and the data, we can then use probability theory to evaluate how our beliefs and predictions should change when we gain new information. For instance, the probability $p(B|A,M)$ measures how plausible it is that the barometer is rising, given that today is a rainy day; this probability can be obtained by the Bayes theorem where the overall model of the weather M is a conditioning statement on the right-hand side of all the probabilities:

$$p(B|A,M) = \frac{p(A|B,M)p(B|M)}{p(A|M)}. \tag{1}$$

Suppose now that a set of L models $\mathcal{M} = \{M_1,\dots,M_L\}$ are under consideration for a training set $\mathcal{D}$, and that under M_k, $\mathcal{D}$ has density $p(\mathcal{D}|\mathbf{w}_k,M_k)$, where $\mathbf{w}_k$ is the vector of parameters that indexes the members of $\mathcal{M}$. A Bayesian approach proceeds by assigning a prior probability distribution $p(\mathbf{w}_k|M_k)$ to the parameters of each model, and a prior probability $p(M_k)$ to each model. Intuitively, this complete specification can be understood as a hypothetical three-stage hierarchical process that generated the training set $\mathcal{D}$:

1. The model M_k was generated according to the distribution $p(M_1),\dots,p(M_L)$.
2. The parameter vector $\mathbf{w}_k$ was generated from $p(\mathbf{w}_k|M_k)$.
3. The data $\mathcal{D}$ were generated from $p(\mathcal{D}|\mathbf{w}_k,M_k)$.

Letting $\mathcal{D}_f$ be the future observations of the same process that generated $\mathcal{D}$, this prior formulation induces a joint distribution $p(\mathcal{D}_f,\mathcal{D},\mathbf{w}_k,M_k) = p(\mathcal{D}_f,\mathcal{D}|\mathbf{w}_k,M_k)p(\mathbf{w}_k|M_k)p(M_k)$. Conditioning on the observed data $\mathcal{D}$, all remaining uncertainty is captured by the joint posterior distribution $p(\mathcal{D}_f,\mathbf{w}_k,M_k|\mathcal{D})$. When the goal is exclusively the prediction of $\mathcal{D}_f$, we should focus on the predictive distribution $p(\mathcal{D}_f|\mathcal{D})$, which is obtained by marginalizing out both $\mathbf{w}_k$ and M_k, that is, after averaging over all unknown models.

In other cases, the focus is on selecting one of the models in $\mathcal{M}$ for the data $\mathcal{D}$. This might be guided by the interest in extracting a useful, simple model from a large class of models. Such a model might, for example, provide valuable scientific insights or perhaps a method for prediction that has a computational load lower than the model average. In terms of the three-stage hierarchical process, the model selection problem becomes that of finding the most probable model in $\mathcal{M}$, which actually generated the data, namely, the model that was selected using $p(M_1), \ldots, p(M_L)$ in the first step. The probability that M_k was in fact this model, conditionally on having observed $\mathcal{D}$, is the posterior model probability

$$p(M_k|\mathcal{D}) = \frac{p(\mathcal{D}|M_k)p(M_k)}{\sum_k p(\mathcal{D}|M_k)p(M_k)}, \tag{2}$$

where

$$p(\mathcal{D}|M_k) = \int p(\mathcal{D}|\mathbf{w}_k, M_k)p(\mathbf{w}_k|M_k)d\mathbf{w}_k \tag{3}$$

is the marginal likelihood of M_k and is called *evidence*. Based on these posterior probabilities, the pairwise comparison of models, say M_1 and M_2, is summarized by the posterior odds

$$\frac{p(M_1|\mathcal{D})}{p(M_2|\mathcal{D})} = \frac{p(\mathcal{D}|M_1)}{p(\mathcal{D}|M_2)} \times \frac{p(M_1)}{p(M_2)}. \tag{4}$$

The expression in Eqn. (4) reveals how the data, through the so-called *Bayes factor* $p(\mathcal{D}|M_1)/p(\mathcal{D}|M_2)$, update prior distribution odds $p(M_1)/p(M_2)$ to yield the posterior odds. The ratio $p(M_1)/p(M_2)$ on the right-hand side of Eqn. (4) measures how much our initial beliefs favored M_1 over M_2, and gives the designer the opportunity of inserting knowledge based on previous experience or on aesthetic grounds. The Bayes factor expresses how well the observed data were predicted by M_1, compared to M_2. As is clearly explained in (MacKay, 1992), this term plays a fundamental role since it implements an automatic Occam razor. Simple models tend to make a small number of predictions while complex models, by their nature, are capable of making a greater variety of predictions. If we consider the models in Fig. 2, a complex model M_N has to spread its predictive probability $p(\mathcal{D}|M_N)$ more thinly over the data space than the simpler one M_1. Thus, in cases where the dataset is compatible with both models, the simpler M_1 will turn out to be more probable than M_N, without having to explicitly express any subjective dislike for complex models. Otherwise, whenever the simple model M_1 is too simple, it will be ruled out by the choice of a more probable model M_N.

Note that, in Bayesian statistics, parameters in the prior probability distributions of the model $p(M_k)$ or of the model parameters $p(\mathbf{w}_k|M_k)$ are called *hyper-parameters*. These hyper-parameters are usually unknown and, as they are fully Bayesian, it could be possible to

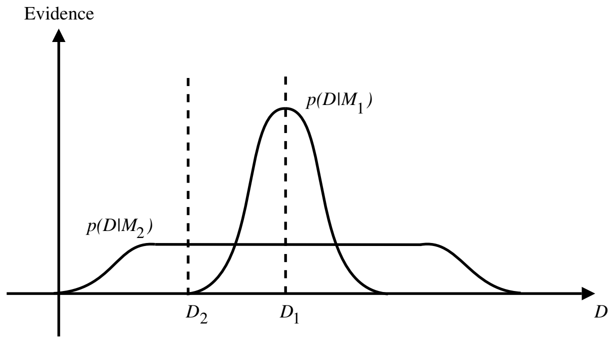


Fig. 2. Example of the automatic Occam razor in model selection.

define a prior distribution over these hyper-parameters and perform the model comparison by considering also these "hyper-priors" (Williams, 1995). Even without using hyper-priors, it might be unfeasible to compute the integral of Eqn. (3) defining the evidence $p(\mathcal{D}|M_k)$, and in these cases it might be preferable to use a computable approximation for it. An effective approximation for this purpose, when

$$h(\mathbf{w}_k) \doteq \log p(\mathcal{D}|\mathbf{w}_k, M_k)p(\mathbf{w}_k|M_k) \tag{5}$$

is sufficiently well-behaved (i.e., the most of the probability is under the $p(\mathcal{D}|M_k)$ maximum), is obtained by the Gaussian approximation given by the Laplace method (Tierney and Kadane, 1986) as

$$\begin{aligned} p(\mathcal{D}|M_k) \approx{}& (2\pi)^{(d_k/2)}|H(\tilde{\mathbf{w}}_k)|^{(1/2)} \\ &\times p(\mathcal{D}|\tilde{\mathbf{w}}_k, M_k)p(\tilde{\mathbf{w}}_k|M_k), \end{aligned} \tag{6}$$

where d_k is the dimension of $\mathbf{w}_k$, $\tilde{\mathbf{w}}_k$ is the maximum of $h(\mathbf{w}_k)$, namely, the posterior mode of $p(\tilde{\mathbf{w}}_k|\mathcal{D}, M_k)$, and $H(\tilde{\mathbf{w}}_k)$ is the negative of the inverse Hessian of $h_{\mathbf{w}_k}$ evaluated at $\tilde{\mathbf{w}}_k$. This is obtained by substituting the Taylor series approximation

$$h(\mathbf{w}_k) \approx h(\tilde{\mathbf{w}}_k) - \frac{1}{2}(\mathbf{w}_k - \tilde{\mathbf{w}}_k)^T H(\tilde{\mathbf{w}}_k)(\mathbf{w}_k - \tilde{\mathbf{w}}_k) \tag{7}$$

for $h(\mathbf{w}_k)$ in the Gaussian approximation of $p(\mathcal{D}|M_k) = \int e^{h(\mathbf{w}_k)} \mathrm{d}\mathbf{w}_k$.

In the classical Bayesian approach, model selection is composed of two levels of inference. On the first level, we assume that a particular model is true and we fit that model to the data inferring which values its free parameters should plausibly take, given the data. This analysis is repeated for each model. The second level of inference is the task of model comparison: here, we assign some sort of preference or ranking to alternative models in the light of the data. Let us write down Bayes' theorem for the two levels of inference:

1. **Model fitting**: we assume that a model M_k is true and we infer its parameters $\mathbf{w}$ given the data $\mathcal{D}$ from

the posterior probability of $\mathbf{w}$:

$$p(\mathbf{w}|\mathcal{D}, M_k) = \frac{p(\mathcal{D}|\mathbf{w}, M_k)p(\mathbf{w}|M_k)}{p(\mathcal{D}|M_k)}, \qquad (8)$$

where $p(\mathbf{w}|M_k)$ is the prior probability of model parameters and the normalizing constant $p(\mathcal{D}|M_k)$ is the evidence for M_k.

2. **Model comparison**: we infer which model is the most plausible given the data. Using (2) and omitting $p(\mathcal{D})$, which is the same for all the models, we obtain

$$p(M_k|\mathcal{D}) \propto p(\mathcal{D}|M_k)p(M_k). \qquad (9)$$

Note that by selecting a single “best” model to make inferences and predictions, we might ignore its uncertainty. An alternative approach to model selection consists in measuring some quantity under each model M_k and then averaging these estimates according to how good each model is (Hoeting *et al.*, 1998). For example, we could average the predictions on a future observation of each model according to how plausible we consider the model. This process is known as *model averaging* (Wasserman, 1999). If we denote by Δ the quantity of interest, we can express the average of the predictions on Δ of different models as

$$p(\Delta|\mathcal{D}) = \frac{p(\Delta|M_k, \mathcal{D})p(M_k)}{\sum_{l=1}^{K} p(\Delta|M_l, \mathcal{D})p(M_l)}. \qquad (10)$$

Although model averaging allows taking into account uncertainty about the model, in many applications its implementation poses several issues:

- the number of terms in (10) may be very large, so that exhaustive summation becomes infeasible;
- the integrals implicit in (10) can be hard to compute in general; Markov chain Monte Carlo methods (Hastings, 1970) have partially overcome the problem, but challenging technical issues remain;
- specification of $p(M_k)$ is challenging, and, for many classes of models, it has received little attention.

Because of these issues, in this paper we do not consider model averaging and we focus on model selection by extracting the most probable model from the model posterior distribution (i.e., Maximum A-Posteriori). In order to do that, we use evolutionary computation to explore the posterior distribution of the adaptive models we are learning. In particular, we use genetic algorithms since they have proved to be a powerful search tool when the search space is large and multimodal, and when it is not possible to write an analytical form for the error function in such a space. In these applications, genetic algorithms are advantageous since they can simultaneously and thoroughly explore many different parts of a large solution space, seeking a suitable solution by implementing a population-based sampling. This sampling approach can be seen as a variation of the Metropolis-Hastings approach (Chib and Greenberg, 1995), and has proven to be extremely efficient.

3. Bayesian Framework for Artificial Neural Networks

Artificial Neural Networks (ANNs) are generic non-linear function approximators which have been extensively used for various purposes such as regression, classification and feature reduction (Bishop, 1995; Haykin, 1999). A neural network is a collection of basic units, called *neurons*, computing a non-linear function of their inputs. Every input has an assigned weight that determines the impact this input has on the output of the node.

In Fig. 3(a) it is possible to see a schematic representation of an artificial neuron, where w_{ji} is the weight of the connection from neuron i to neuron j, and s_j is the activation or output of neuron j. Unit j determines its output by ideally following a two-step procedure:

1. It computes the total weighted input z_j, using the formula

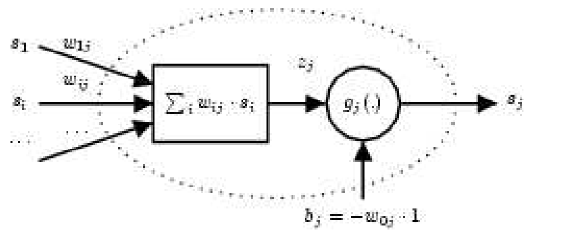
$$z_j = \sum_{i=1} w_{ji} s_i,$$

where s_i is the activity level of the i-th unit in the previous layer and w_{ij} is the weight of the connection between the i-th and the j-th units.

2. It calculates its activity s_j using some non-linear function $g_j(\cdot)$ of its total weighted input z_j minus a bias term $b_j \doteq -w_{j0} \cdot 1$:

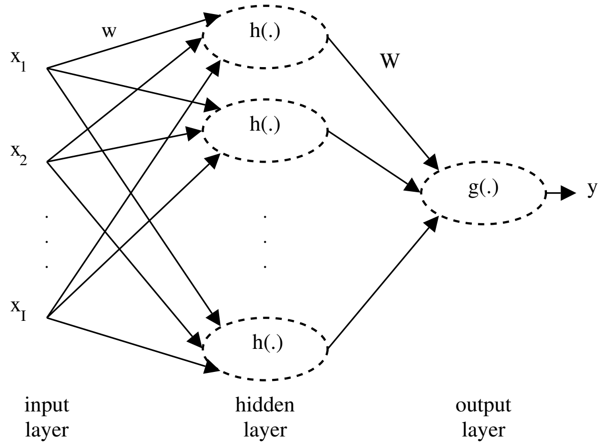
$$s_j = g_j(z_j - b_j) = g_j\Bigg(\sum_{i=0} w_{ji} s_i\Bigg).$$

Functions $g(\cdot)$ commonly used in artificial neural networks are squashing functions, like sigmoid or the hyperbolic tangent. By interconnecting a proper number of nodes in a suitable way and by setting the weights to appropriate values, a neural network can approximate any non-linear function with an arbitrary precision (Hornik *et al.*, 1989). This structure of nodes and connections, known as the *network topology*, together with the weights of the connections, determines the network final behavior. Figure 3(b) describes a simple feed-forward topology, i.e., no loops are present with a single hidden layer, i.e., a layer of neurons neither connected to the input, nor to the output.



Artificial Neuron

(a)



(b)

Fig. 3. Schema of an artificial neuron (a) and a feed-forward network topology (b).

Given a neural network topology and a training set, it is possible to optimize the values of the weights in order to minimize an error function by means of any backpropagation algorithm (Rumelhart *et al.*, 1986), standard optimization techniques (Press *et al.*, 1992) or randomized algorithms (Montana and Davis, 1989). However, the topology of a neural network plays a critical role in whether or not the network can be trained to learn a particular data set. In fact, we cannot easily answer the question of how many nodes, layers, or connections a neural network should have, and no algorithm exists for finding the optimal solution for the design of such a topology. Clearly, the simpler the topology, the simpler the function the neural network is computing. A simple topology will result in a network that cannot learn to approximate a complex function, while a complex topology is likely to result in a network losing its generalization capability. This loss of generalization is the result of *overfitting* the training data: instead of approximating a function present in the data, a neural network that has an overly complex structure may have the ability to memorize the training set, allowing noise within the data to be learned as part of the model, resulting in inaccurate predictions on future samples.

In this paper, we focus on feed-forward topologies with arbitrary non-linear, differentiable activation functions for each layer and with "shortcut" connections linking two non-subsequent layers. We call this kind of enriched topologies *Rich Neural Networks* (RNNs) (Matteucci, 2002a). They were originally inspired by (Flake, 1993) and the main interest in this kind of topology is to state the effectiveness of using various activation functions for the network layers (Mani, 1990; Ronald and Schoenauer, 1994; Lovell and Tsoi, 1992) with a generalized feed-forward structure.

Due to the complexity of the design activity for such networks, we propose to use an automatic tool based on evolutionary computation and to define its fitness function by using the Bayesian framework for the model selection introduced in Section 2. In doing this, we use an improper prior for the neural network topologies meaning that we do not express any explicit belief about the model structure; this is accomplished by assuming the same probability for all the models M_k. Instead, we express our belief about the weights of neural networks $p(\mathbf{w}_k|M_k)$ by using a conjugate Gaussian prior. In the following, we present a detailed description of the priors used in applying the Bayesian framework to rich neural networks and we derive the Bayesian fitness that represents the posterior distribution for the models.

3.1. Prior Distribution of Network Weights

We now consider the prior probability distribution of network weights $\mathbf{w}$. In the absence of any data, we have little idea of what the weight values should be; at this stage, the prior might express some general properties such as the smoothness of the network function but should also leave the weight values fairly unconstrained. Experience suggests that positive and negative weights are equally frequent, that smaller weights are more frequent than larger ones and that very large weights are very unlikely. A Gaussian prior is a formal description for this concern:

$$p(\mathbf{w}) = \frac{1}{Z_W(\alpha)} \exp(-\alpha E_W). \qquad (11)$$

Here, $Z_W(\alpha)$ is the normalization constant

$$Z_W(\alpha) = \int \exp(-\alpha E_W) \mathrm{d}\mathbf{w} \qquad (12)$$

which ensures that $\int p(\mathbf{w}) \mathrm{d}\mathbf{w} = 1$, and E_W (called the "weight error") is defined as

$$E_W = \|\mathbf{w}\|^2 = \frac{1}{2} \sum_{i=1}^{W} w_i^2, \qquad (13)$$

where W is the total number of weights and biases in the network. Combining (11) and (13), we have

$$p(\mathbf{w}) = \frac{1}{Z_W(\alpha)} \exp\Big(- \frac{\alpha}{2} \sum_{i=1}^{W} w_i^2 \Big). \tag{14}$$

This formulation is a straightforward derivation from the belief of independence of weights following a Gaussian distribution with zero mean and variance $1/\alpha$. With such a choice, when $\|\mathbf{w}\|$ is large, E_W is large and $p(\mathbf{w})$ is small: the prior distribution penalizes larger values of weights, reflecting our experience about the network parameters. The hyper-parameter α controls the distribution of model parameters (weights and biases) and for the moment we assume that it as a fixed, known constant. Owing to the choice of a Gaussian prior, the evaluation of the normalization factor $Z_W(\alpha)$ in (12) is straightforward and gives

$$Z_W(\alpha) = \left(\frac{2\pi}{\alpha}\right)^{\frac{W}{2}}. \tag{15}$$

3.2. Artificial Neural Network Learning as Inference

We now consider the problem of training a regression network with a given architecture (i.e., the number of layers, the number of hidden units, etc.): such a network maps an input $\mathbf{x}$ to an output $y(\mathbf{x}|\mathbf{w})$ which is a continuous function[1] of the parameters $\mathbf{w}$. The network is trained using a data set $\mathcal{D}$, consisting of N patterns of the form $(\mathbf{x}, t)$, by iteratively adjusting $\mathbf{w}$ so as to minimize an objective function, e.g., the sum of the squared errors:

$$E_D(\mathbf{w}) = \frac{1}{2} \sum_{i=1}^{N} \Big(t^{(i)} - y(\mathbf{x}^{(i)}|\mathbf{w}) \Big)^2. \tag{16}$$

This minimization is usually based on repeated evaluation of the gradient of E_D using the back-propagation algorithm. We can give a maximum likelihood probabilistic interpretation to this learning process. In fact, let us suppose that the patterns in the training set are independently drawn from a distribution $p(\mathbf{x}, t)$; we model t as a deterministic non-linear function $y(\mathbf{x})$ plus some zero-mean Gaussian noise. Under this assumption, the probability of observing a single datum t for a given input vector $\mathbf{x}$ would be

$$p(t|\mathbf{x}, \mathbf{w}) \propto \exp\left(-\frac{\beta}{2} \left(t - y(\mathbf{x}|\mathbf{w})\right)^2 \right), \tag{17}$$

where $\beta = 1/\sigma_\nu^2$ controls the variance of the noise, and, for the moment, we shall assume this hyper-parameter β is known and constant. Since the data points are drawn independently of this distribution, the probability of the training data $\mathcal{D}$, called the *likelihood*, is

$$\begin{aligned} p(\mathcal{D}|\mathbf{w}) &= \prod_{n=1}^{N} p(t^n|\mathbf{x}^n, \mathbf{w}) \\ &= \frac{1}{Z_D(\beta)} \exp\left(-\frac{\beta}{2} \sum_{n=1}^{N} \left(t^n - y(\mathbf{x}^n|\mathbf{w}) \right)^2 \right) \\ &= \frac{1}{Z_D(\beta)} \exp(-\beta E_D), \end{aligned} \tag{18}$$

where $Z_D(\beta)$ is the normalization factor given by

$$Z_D(\beta) = \left(\frac{2\pi}{\beta}\right)^{\frac{N}{2}}. \tag{19}$$

It is straightforward to derive that the maximum likelihood estimation of the neural network weights is equivalent to the minimization of the error (16) by the back-propagation algorithm.

3.3. Posterior Weight Distribution

Once we have chosen a prior distribution and an expression for the likelihood function, we can use Bayes' theorem to find the posterior distribution of network weights. Using (11) and (18), we get the posterior distribution in the form

$$\begin{aligned} p(\mathbf{w}|\mathcal{D}) &= \frac{1}{Z_S} \exp(-\beta E_D - \alpha E_W) \\ &= \frac{1}{Z_S} \exp\big(- S(\mathbf{w}) \big), \end{aligned} \tag{20}$$

where

$$S(\mathbf{w}) = \beta E_D + \alpha E_W \tag{21}$$

and

$$Z_S(\alpha, \beta) = \int \exp(-\beta E_D - \alpha E_W) \mathrm{d}\mathbf{w}. \tag{22}$$

In order to find the weight vector $\mathbf{w}_{\text{MP}}$ corresponding to the maximum of the posterior distribution, we can minimize the negative logarithm of (20) with respect to the weights. Since the normalizing term Z_S does not depend on the weights, we only need to minimize $S(\mathbf{w})$ given by (21) obtaining

$$S(\mathbf{w}) = \frac{\beta}{2} \sum_{n=1}^{N} (t^n - y(\mathbf{x}^n|\mathbf{w}))^2 + \frac{\alpha}{2} \sum_{i=1}^{W} w_i^2. \tag{23}$$

Apart from an overall multiplicative factor, this is precisely the usual sum-of-squares error function with a

[1] Discontinuous functions are not practical for gradient-based optimization.

weight-decay regularization term. If we are only interested in finding the weight vector $\mathbf{w}_{\mathrm{MP}}$ which minimizes this error function, the overall multiplicative factor is irrelevant and the effective value of the regularization parameter depends only on the ratio α/β. Note that as the number of patterns N in the training set increases, the first term in (23) grows with N, while the second term does not. Thus, if α and β are fixed, then as N increases, the first term becomes more and more dominant, until eventually the second term becomes insignificant. On the contrary, for a small number of patterns, the second term plays an important role in determining the most probable solution.

3.4. 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 no idea of suitable values for α and β. Recalling (MacKay, 1995; 1999), 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:

$$p(\alpha,\beta|\mathcal{D},M) = \frac{p(\mathcal{D}|\alpha,\beta,M)p(\alpha,\beta|M)}{p(\mathcal{D}|M)}. \tag{24}$$

Assuming that we have only a rough idea of suitable values for α and β, since the denominator in (24) is independent of α and β, the maximum *a-posteriori* values for these hyper-parameters are found by maximizing the term $p(\mathcal{D}|\alpha,\beta,M)$. If we can approximate the posterior probability distribution (20) by a single Gaussian function, according to the Laplace approximation of (6), we obtain

$$p(\mathbf{w}|\mathcal{D},M) \simeq \frac{1}{Z_S^*}\exp(-S(\mathbf{w}) - \frac{1}{2}\left(\mathbf{w}-\mathbf{w_{MP}}\right)^T\mathbf{A}(\mathbf{w}-\mathbf{w_{MP}})\right), \tag{25}$$

where $A = \nabla\nabla \ln p(\mathbf{w}|\mathcal{D},\alpha,\beta,M)|_{\mathbf{w}_{MP}}$, and the evidence for α and β can be written as

$$\begin{aligned}\ln p(\mathcal{D}|\alpha,\beta,M) &= \ln\frac{Z_S^*}{Z_D(\beta)Z_W(\alpha)}\\ &= -S(\mathbf{w}_{MP}) - \frac{1}{2}\ln\det\left(\frac{A}{2\pi}\right)\\ &\quad - \ln Z_W(\alpha) - \ln Z_D(\beta). \end{aligned}\tag{26}$$

Using (11) and (19), we can write the log of the evidence as

$$\begin{aligned}\ln p(\mathcal{D}|\alpha,\beta) &= -\alpha E_W^{MP} - \beta E_D^{MP} - \frac{1}{2}\ln|A|\\ &\quad + \frac{W}{2}\ln\alpha + \frac{N}{2}\ln\beta - \frac{N}{2}\ln(2\pi).\end{aligned}\tag{27}$$

As shown in (Gull, 1989), given λ_i as the eigenvalues of the Hessian $H = \beta\nabla\nabla E_D$, the maximum of evidence for α and β satisfies the following implicit equations:

$$2\alpha E_W^{MP} = W - \sum_{i=1}^{W}\frac{\alpha}{(\lambda_i+\alpha)} = \gamma, \tag{28}$$

with $\gamma = \sum_{i=1}^{W}\lambda_i/(\lambda_i+\alpha)$,

$$2\beta E_D^{MP} = N - \sum_{i=1}^{W}\frac{\lambda_i}{\lambda_i+\alpha} = N-\gamma. \tag{29}$$

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

$$\alpha^{\mathrm{new}} = \gamma/2E_W, \tag{30}$$

$$\beta^{\mathrm{new}} = (N-\gamma)/2E_D. \tag{31}$$

3.5. Bayesian Fitness Function

Once the most probable values for the weight vector $\mathbf{w}$ and hyper-parameters α and β of a given neural network have been determined, we can compare different networks. In order to evaluate a given neural network in our genetic algorithm, we introduce the following expression, which is derived from (27):

$$\begin{aligned}\ln p(D|M_k) &= -\alpha_{\mathrm{MP}}E_W^{\mathrm{MP}} - \beta_{\mathrm{MP}}E_D^{\mathrm{MP}} - \frac{1}{2}\ln|\mathbf{A}|\\ &\quad + \frac{W}{2}\ln\alpha_{\mathrm{MP}} + \frac{N}{2}\ln\beta_{\mathrm{MP}}\\ &\quad + \frac{1}{2}\ln\left(\frac{2}{\gamma}\right) + \frac{1}{2}\ln\left(\frac{2}{N-\gamma}\right).\end{aligned}\tag{32}$$

We use this expression as the fitness for the genetic algorithm, while searching for the right topology to perform model comparison. We call it the *Bayesian fitness function*. Using this finess function to search fitting models for our dataset, we expect, due to Occam's razor embodied in the Bayesian framework, that complex networks will be automatically penalized while small ones will be favored, thus obtaining a twofold result: reduce overfitting (thus increasing the generalization capability of the model) and reduce the “bloating phenomenon”. Angeline describes such a phenomenon in his applications (Angeline, 1994); he observes that many of the evolved solutions found by genetic programming contain a code that, when removed, does not alter the produced result. In our case, we would like to obtain small rich neural network models with good generalization capabilities without having to remove the nodes that are not useful using an *a-posteriori* analysis of the weights in the network like in (Weigend *et al.*, 1991; Hassibi and Stork, 1992; Hashem, 1997).

4. ELeaRNT Genetic Algorithm

ELeaRNT (Evolutionary Learning of Rich Neural network Topology) (Matteucci, 2002a; Matteucci, 2002b) 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 scheme of Goldberg’s *Simple Genetic Algorithm* (Goldberg, 1989). It uses non-overlapping populations and at each generation creates an entirely new population of individuals by selecting from the previous one, and then mating them to produce offspring for the new population. In all our experiments we use *elitism*, meaning that the best individual from each generation is carried over to the next generation; however, this is not mandatory.

4.1. Rich Neural Network Representation

ELeaRNT uses a direct coding scheme to represent a network, i.e., each detail of the architecture (i.e., the number of neurons, activation functions, connections, the learning algorithm, etc.) is specified in the genotype: this allows a more focused design of the genetic operators that are closed with respect to the chosen phenotype.[2] Direct encoding has proved to be less effective with larger genotypes because the effects of crossover and mutation are often unfavorable for retaining any kind of high level network structure that may have been evolved (Liu and Yao, 1996). For this reason, the coding we propose in Section 4.1.2 is suitable for keeping the network representation compact, avoiding the “competing convention” issue that arises from the fact that the order of the nodes in the hidden layers of neural networks is irrelevant (Hancock, 1992).

[2] We define a genetic operator to be closed with respect to the phenotype if applying it to a valid genotype that codes a rich neural network topology always produces another valid genotype that codes another rich neural network topology.

4.1.1. Network Model: Phenotype

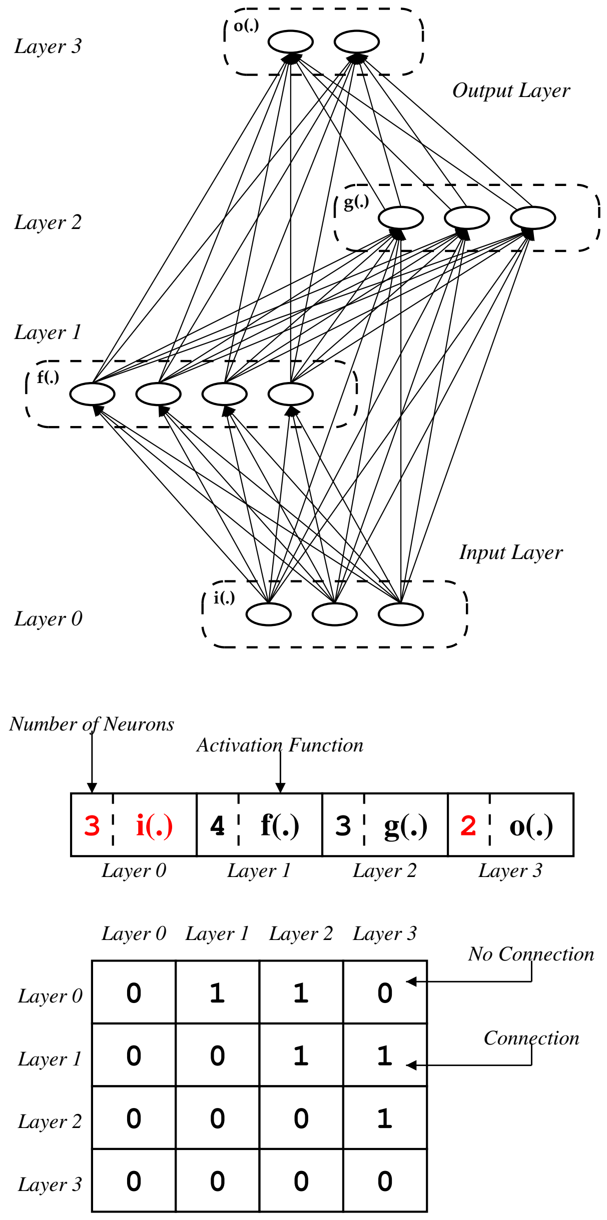
In RNNs each layer has at least one neuron, and, potentially, a different activation function. The numbers of neurons in the first and last layers are fixed, since these are the numbers of input and output variables of the specific problem. The transfer function for the input layer is usually the *identity function* and for the other layers it can be any of the following choices: *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 a 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 4 shows an example of the RNN topology evolved by our algorithm.

4.1.2. Genetic Coding: Genotype

Each phenotype is coded by a two-part genotype. The first part encodes the layer information (i.e., the number of neurons and the activation function), and the second part encodes the connectivity between the layers using a matrix. To specify a proper feed-forward neural network, only the elements above the diagonal in the connectivity matrix may differ from 0. Since we chose the identity function for the first layer, the activation function for that part of the genotype cannot be changed during the evolution. 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; this implies that a layer of neurons is not reachable from the input or it does not participate in the final output. To avoid this issue, we designed the genetic operators to be closed with respect to the phenotype family. This is not mandatory, as it was introduced only to increase the efficiency of the search algorithm by drastically reducing the number of unfeasible solutions to be rejected.

4.2. Genetic Operators

In our implementation, 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. We will shortly introduce these operators in the next paragraphs. For a more detailed description, see (Matteucci, 2002a; Matteucci, 2002b). Notice that, to keep a valid offspring after crossover, we might have to increase the number of connections with respect to parents; this increased number of connections produced by



	Layer 0	Layer 1	Layer 2	Layer 3
Layer 0	0	1	1	0
Layer 1	0	0	1	1
Layer 2	0	0	0	1
Layer 3	0	0	0	0

Fig. 4. Example of a phenotype evolved by our genetic algorithm and its coding.

the crossover operators is not a major issue in the algorithm, and we could easily solve this increased complexity by introducing a post pruning operator to be used after training (Bebis *et al.*, 1997; Castellano *et al.*, 1997).

4.2.1. Single-Point Crossover

The single-point crossover operator combines two networks by cutting their topologies in two pieces with a surface that entirely separates the input and the output of the network and then switching the input parts of the two networks. In order to guarantee this operator to be closed with respect to the valid genotype family, we have to restore all the connections between the two pieces of the networks. Connections coming from the input part of the first network have to be joined with connections going into the output part of the second network, and vice versa. In this way, the final number of connections in the newly generated individuals might be greater than the original one, but the validity of the genotype is preserved.

Figure 5(a) describes the effect of the single-point crossover operator. Two random points in the first part of the two genotypes are chosen. Note that cell (i, j) in the top right sub-matrix of the genotype has a connection *iff* at least one of the cells in the i-th row of the parent providing the input part has a connection and at least one of the cells in the j-th column of the parent providing the output part has a connection.

4.2.2. Two-Point Crossover

The two-point crossover operator combines two networks by extracting a subgraph from each of them, and exchanging these two sub-graphs. In order to guarantee this operator 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. Connections coming off or going into the new block have to be joined to connections going into or coming out of the old block. Also, with this crossover operator the final number of connections between the newly generated individuals might be greater than the original one.

Figure 5(b) illustrates an example of the application of this operator. Note that, to join the new block into the "hosting" network, the top middle and right middle sub-matrices have to be filled in a specific way. A cell (i, j) in the top middle sub-matrix has a connection *iff* any of the cells in the i-th row of the original top middle sub-matrix of the parent network hosting the new block has a connection and any of the cells in the j-th column of the parent providing the block has a connection. A cell (i, j) in the right middle sub-matrix has a connection *iff* any of the cells in the j-th column of the original right middle sub-matrix of the parent network hosting the new block has a connection and any of the cells in the i-th row of the parent providing the block has a connection.

4.2.3. Mutation

In order to guarantee the eventual exploration of the entire model search space, we implemented six different mutation operators. Here they are briefly described (for a more detailed description, see (Matteucci, 2002a; Matteucci, 2002b)):

- Drop layer: this mutation operator randomly selects a layer and removes it from the network structure. Before removing the layer from the network structure,

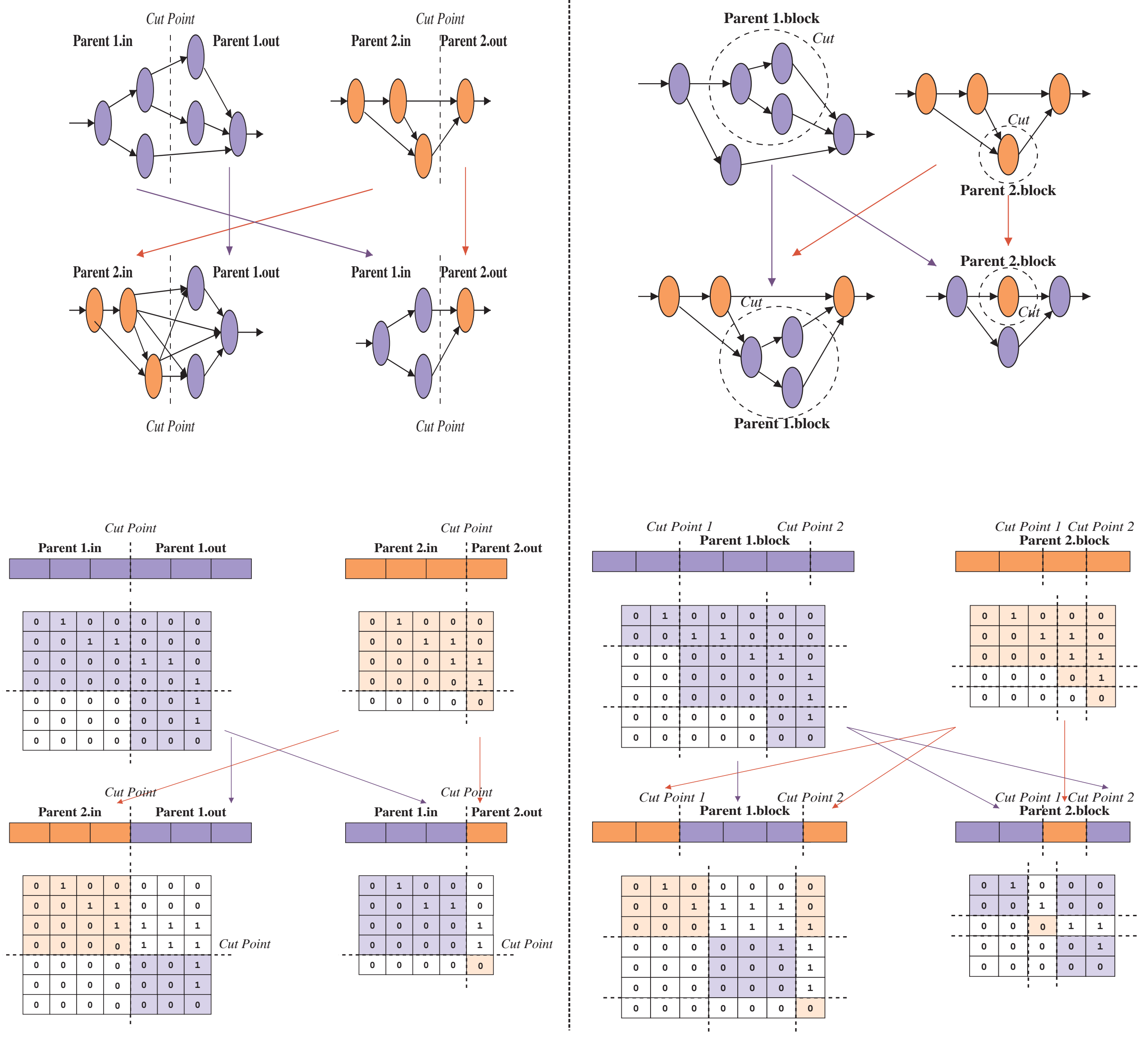


Fig. 5. Single-point crossover (a) and two-point crossover (b) genetic operators.

its input connections are directly connected to all the destinations of its output connections.[3]

- Add layer: this mutation 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. Since a valid copy of an existing neuron connectivity sub-matrix is used, this operator is guaranteed to be closed with respect to the valid genotype family.

- Number of neurons: this mutation operator changes the number of neurons in a specific layer of the network. A random mutation point is chosen and the number of neurons in the specific layer is changed according to a uniform distribution.

- Drop connection: this mutation operator removes a connection from the connectivity matrix of the net-

[3] This is equivalent to setting the activation function of the layer to identity, but reduces the number of weights of the network to be trained and thus the number of free parameters.

work. Once the connection is removed, the operator checks for layers that are neither reachable from the input nor participate in the output of the network. These layers have to be removed and this may lead to a complete destruction of the network. This operator is not guaranteed to be closed with respect to the valid genotype family, and thus a modified genotype has to be checked. In case a non-valid genotype is generated, a new one is randomly initialized and substituted for the original one.

- Add connection: this mutation operator adds a new connection in the connectivity matrix of the network. If the network is completely connected, the genotype is left unchanged.
- Activation function: this mutation operator changes the activation function in a network layer. A random mutation point is chosen and the activation function in that specific layer is changed according to a uniform distribution over the available activation functions.

Notice that two out of six mutation operators introduce pruning into the genetic algorithm. Conversely, two out of six mutation operators increase the size of the networks by adding connections and nodes.

4.3. ELeaRNT and the Bayesian Fitness Function

We now describe the complete ELeaRNT algorithm which allows us to create, train and evolve rich neural networks within the Bayesian framework discussed in Section 2. The ELeaRNT algorithm evolves network topologies in order to maximize the Bayesian fitness function defined in Section 3. In fact, each individual is evaluated using the Evaluate procedure presented in Algorithm 1: at first, the weight vector $\mathbf{w}$ is initialized according to a Gaussian prior distribution and its hyper-parameters α and β are set by the user to an initial value. After initialization, each individual is trained using a standard non-linear optimization technique based on the conjugate gradient descent in order to minimize the regularized error function (23). After a given number of training epochs, hyper-parameters α and β are re-estimated by using the formula (30) and (31) with γ given by (28).

Algorithm 1. Evaluate Procedure

```
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 (23)
    Every $M$ epochs re-estimate $\alpha$, $\beta$ using (30) and (31)
until Desired number of restarts reached
Return the average fitness over the restarts
End Evaluate
```

Network training is performed using batch learning with the Polak-Ribière deterministic conjugate gradient algorithm and golden line search (Fletcher, 1987). Note that this algorithm does not require an explicit evaluation of the Hessian matrix. Instead, the re-estimation of α and β requires the computation of the Hessian matrix and its eigenvalues. To reduce the computational load of re-estimating hyper-parameters, this operation is performed only every M epochs, computing the Hessian matrix with the efficient Pearlmutter algorithm (Pearlmutter, 1994) and its eigenvalues by LU decomposition (Press *et al.*, 1992). Finally, to reduce the dependence of $\ln|\mathbf{A}|$ on small eigenvalues, it is possible to set a (positive) cut-off value ϵ for the eigenvalues λ_i: all the eigenvalues smaller than this cut-off will be set to ϵ.

Each individual is then evaluated using the Bayesian fitness function. Initialization, training and evaluation are performed several times to avoid local minima, and the average performance over the different restarts is used. Taking the average over the restarts is just one of the possible choices. In fact, we could use the best result as the fitness value for the individual we are evaluating. We decided to use the average to give more emphasis to the network model and not to the particular local minimum found by the optimization algorithm.

ELeaRNT, described in Algorithm 2, starts by generating an initial random population of neural networks: individuals can have an arbitrary number of layers, neurons and different types of activation functions. Once this first population has been evaluated, selection takes place. In order to prevent the loss of the best found solution, we use elitism and therefore the best individual in the current generation is carried over to the next generation. 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. The algorithm will cycle for a given number of generations or until a given stopping criterion is met.

5. ELeaRNT Experimental Validation

In this section we validate the ELeaRNT algorithm by applying it to three regression tasks. In these experiments, we use an improper prior over the network space and the Gaussian prior over model parameters as described in Section 3. We are mainly interested in:

- Proving that ELeaRNT with the Bayesian fitness function is able to select small neural network

Algorithm 2. 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 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 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
    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
```

topologies that are still well-matched to the data and thus possess a good generalization capability.

- Verifying the effectiveness of Occam's razor embodied in the evidence-based Bayesian fitness function, that is, verifying if simple models are favored over complex ones without having to specify any explicit initial preference over the models.

All the experiments are repeated 10 times, and, unless explicitly reported, the results are averaged over these runs. The number of individuals in the population is critical for the success of the genetic algorithm. However, the right number of individuals is problem dependent and there is no definitive answer to this issue. Before starting the experiments, we have tried a different number of individuals per population, up to 50–100. In all the experiments we use a population with 20 individuals, with crossover and mutation probability, respectively, $p_{cross} = 0.75$ and $p_{mut} = 0.25$. With the benchmark we used, there was no real improvement in using larger populations. The choice of the remaining parameters was not optimized; however, the algorithm converged in all the experiments. Usually, a very good individual is found very quickly (i.e., after less than 10 generations) and the rest of the evolution process is spent on exploring alternative solutions. With optimized probabilities for the genetic operators, a larger population, and longer evolutions, the obtained results might outperform our solutions. However, finding better solutions does not add much to the validation of the Bayesian fitness with respect to the results we present here.

Network weights were initialized according to a Gaussian distribution. Since the choice of the initial weights determines to which minimum of the weight space the training algorithm will converge, we trained each individual a few times with different initial values for the weights. The choice of the initial values for hyper-parameter α and β is quite critical since if they are completely wrong, the genetic algorithm will not be able to find acceptable solutions. In general, α determines the impact of the prior over the network weights, and thus determines the extent to which complex networks are penalized with respect to simple ones. Conversely, β determines the impact of the network error on fitting the training data, and hence small values of β will result in networks that poorly approximate the training set. In order to find good solutions, we initialized β to a value larger than expected while we 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, complex networks will be penalized.

5.1. Benchmark 1

In this experiment, we use a small number of samples (32) in the training set, sampling regularly on the $[-5, 15]$ interval the non-linear function

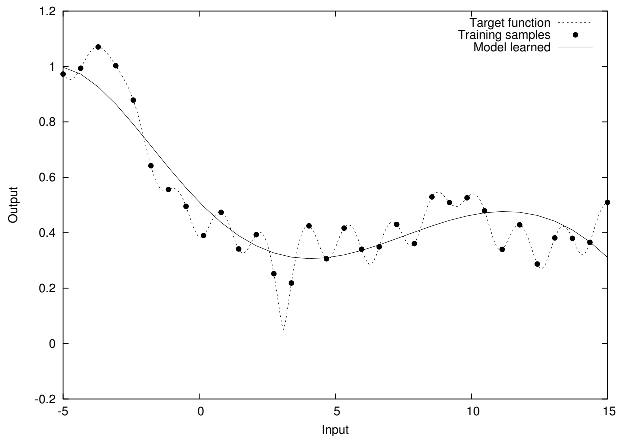
$$y(x) = \sqrt{0.1\,\sin(2x)^2 + \frac{2\arctan(x-3)^2}{(x+7)(\cos(x)+2)}}. \tag{33}$$

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 several 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 by the individual in Fig. 6(a).

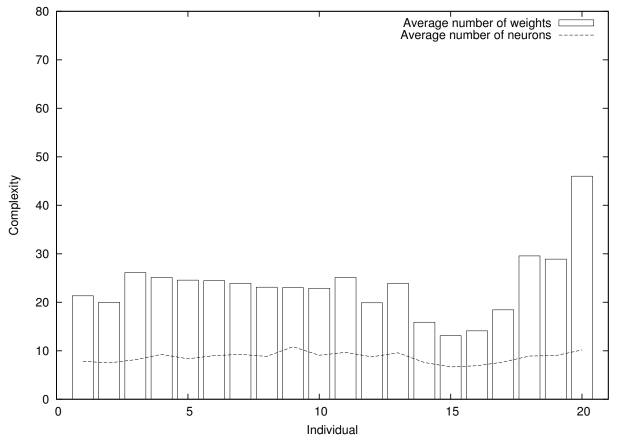
To give an idea of convergence rates for the ELeaRNT algorithm, we report the fitness value during

Table 1. Complexity of best, worst and average individuals in the three benchmarks.

		Best individual	Average	Worst individual
Benchmark 3	Complexity	2	80.94	255.9
	Standard deviation	0	7.06	12.48
Benchmark 1	Complexity	21.53	34.10	51.15
	Standard deviation	4.68	6.99	8.58
Benchmark 2	Complexity	33.36	98.46	157
	Standard deviation	3.67	12.92	13.11

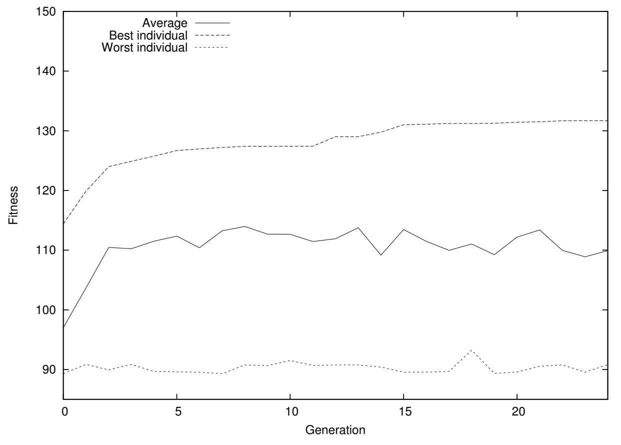


(a)

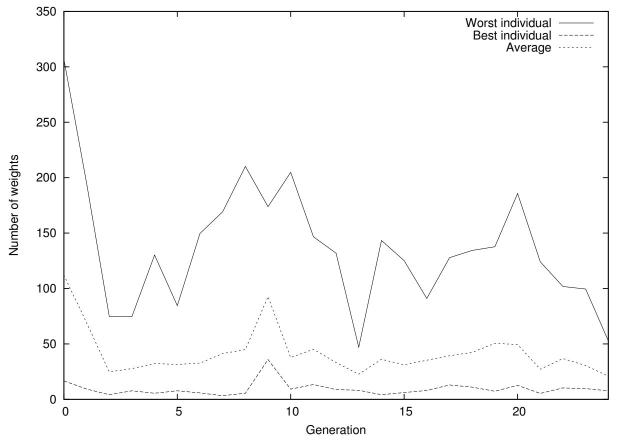


(b)

Fig. 6. Best model fitting (a) and model complexity (b) in the final population.



(a)



(b)

Fig. 7. Fitness value (a) and model complexity (b) during evolution.

learning in Fig. 7(a), where the best, mean, and average fitness values in the population are plotted. In Fig. 7(b) we report the complexity of models during the evolution. Notice that it is not strictly increasing since the complexity of the model is only partially related to the Bayesian fitness function.

The average complexity of the final population of the genetic evolution is reported in Table 1, and in Fig. 6(b) this complexity is reported for each individual of the last generation in descending order with respect to the fitness. As we can see, the average complexity of each individual is quite small, and not considering the worst two cases, each network has less than 30 weights (corresponding to an average of 9 hidden neurons). In this case, ELeaRNT proved to be able to find minimal rich neural networks with both good accuracy and small topologies (even when only few patterns of the function to be approximated are available), also preventing the "bloat phenomenon."

5.2. Benchmark 2

In this second example, we test ELeaRNT on a real data set. The training patterns of this case were used in biological studies in order to find a reliable method for determining the age of the European rabbits (*Oryctolagus cuniculus*) from the weight of their eye lens. In this study, the dry weight of the eye lens was measured for 71 free-living wild rabbits of the known age (Dudzinski and Mykytowycz, 1961).

We are interested in proving the ability of ELeaRNT to select rich neural networks well-matched to real data extracted from an unknown phenomenon and with an unknown noise model. We evolved 20 individuals for 25 generations and re-estimated the hyper-parameters every 20 epochs of training. Before running ELeaRNT, we preprocessed the training set by normalizing all its samples which are shown in Fig. 8(a). Figure 8(a) also shows the model learned by the most probable individual in the population: this model is very well matched to the observations and it does not overfit them.

The fitness value during learning is reported in Fig. 9(a), where the best, mean, and average fitness values in the population are plotted. In Fig. 9(b) we report the complexity of models during the evolution. Notice again that it is not strictly increasing since the complexity of the model is only partially related to the Bayesian fitness function.

Figure 8(b) and Table 1 report the average complexity of the population after the genetic evolution. As we can see, the average number of the weights of the best individuals (34 weights corresponding to 8 neurons) is sensibly smaller than the average complexity of less probable individuals. We can therefore say that also for this test case ELeaRNT selected networks with a relatively small complexity but still well matched to the data.

5.3. Benchmark 3

In this experiment, the algorithm has to fit a noisy sinusoidal training set given by

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

We are interested in this example since the process generating the data belongs to the family of models that can be represented by using an 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 (i.e., the correct model of the data). The training set is composed of 300 samples drawn uniformly from the $[0, 1]$ interval and the added noise ϵ is Gaussian with zero mean and variance $\sigma^2 = 0.04$.

This time we evolved 20 individuals for 50 generations and re-estimated the hyper-parameters every 20 epochs of training. Figure 10(a) shows the model learned by the most probable individual in the population and its complexity is reported in Table 1. Figure 10(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, i.e., Individual 1, we can see that in all the different executions ELeaRNT selected a single neuron model with a sinusoidal activation function. Moreover, the complexity of less probable individuals increases and the worst individuals turn out to be the most complex.

The fitness value during learning is reported in Fig. 11(a), where the best, mean, and average fitness values in the population are plotted. We see that 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. Finally, the average fitness increases quite smoothly over generations approaching the fitness of the best individual. In Fig. 11(b) we report the complexity of the models during the evolution.

6. Discussion and Conclusions

In this paper we presented a probabilistic approach to the problem of model selection using genetic algorithms within a Bayesian framework. We detailed the *Bayesian Fitness Function* to evaluate the posterior probability of neural networks within the Bayesian framework. Such a fitness was used by a genetic algorithm, ELeaRNT, to

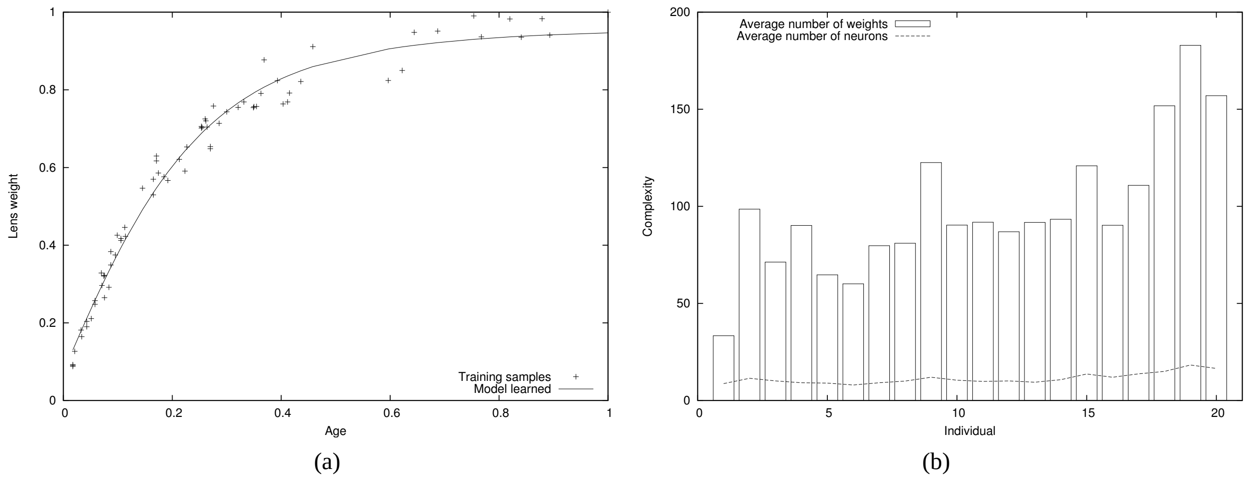


Fig. 8. Best model fitting (a) and the average number of parameters (b) in the population.

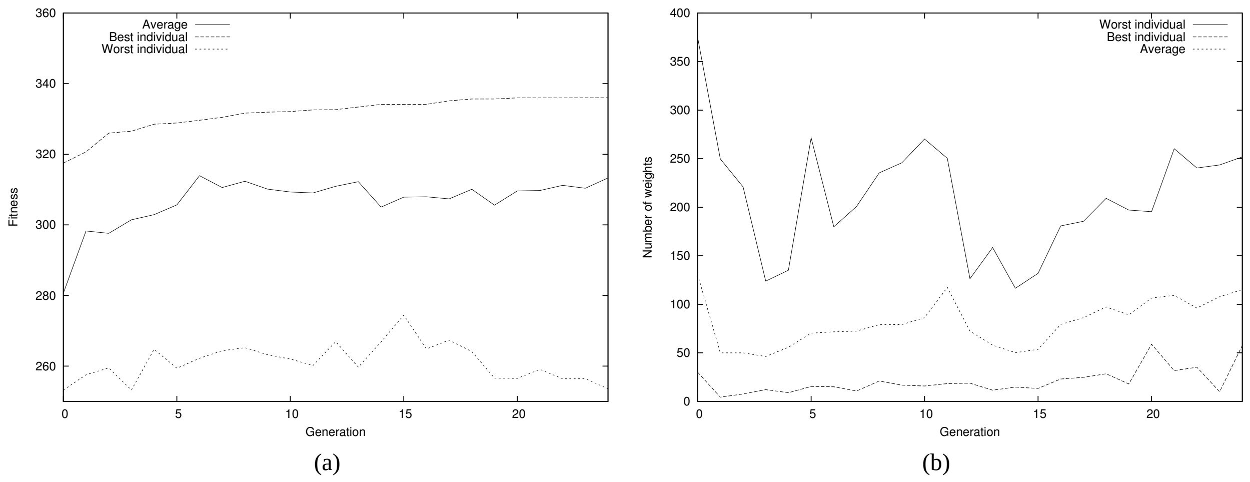


Fig. 9. Fitness value (a) and model complexity (b) during evolution.

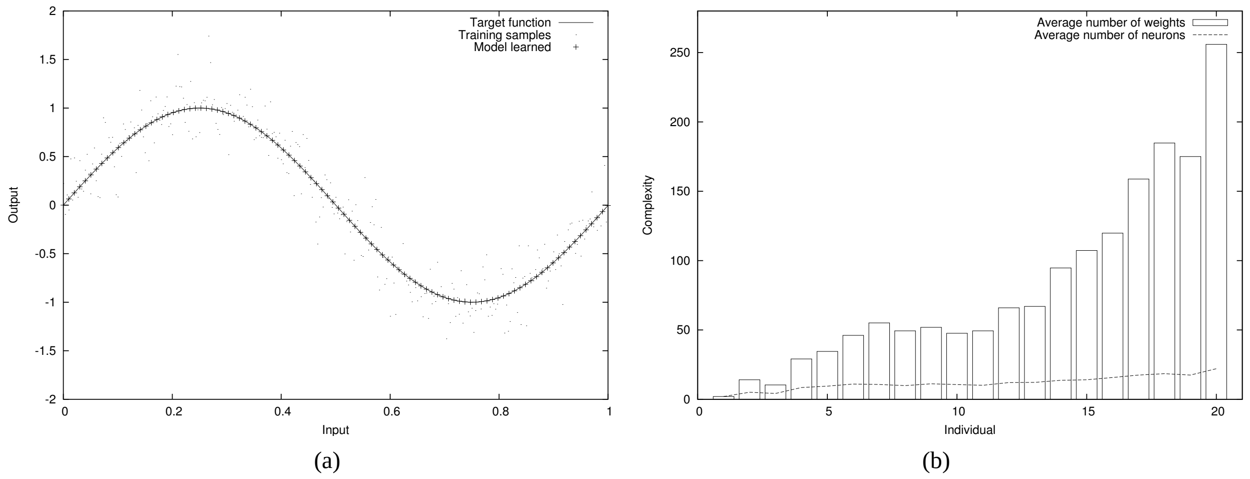


Fig. 10. Best model fitting (a) and the average number of parameters (b) in the population.

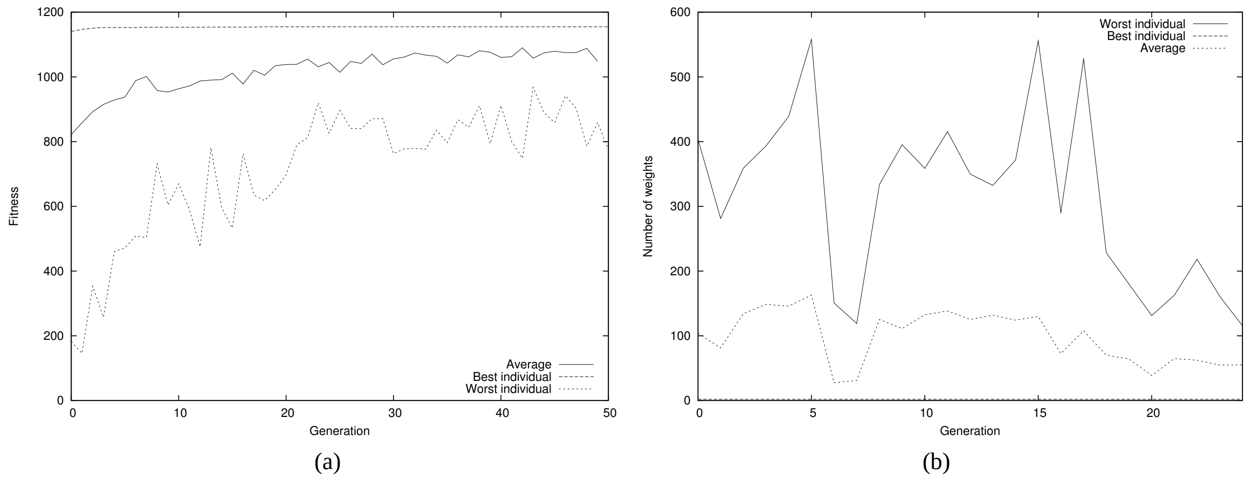


Fig. 11. Fitness value (a) and model complexity (b) during evolution.

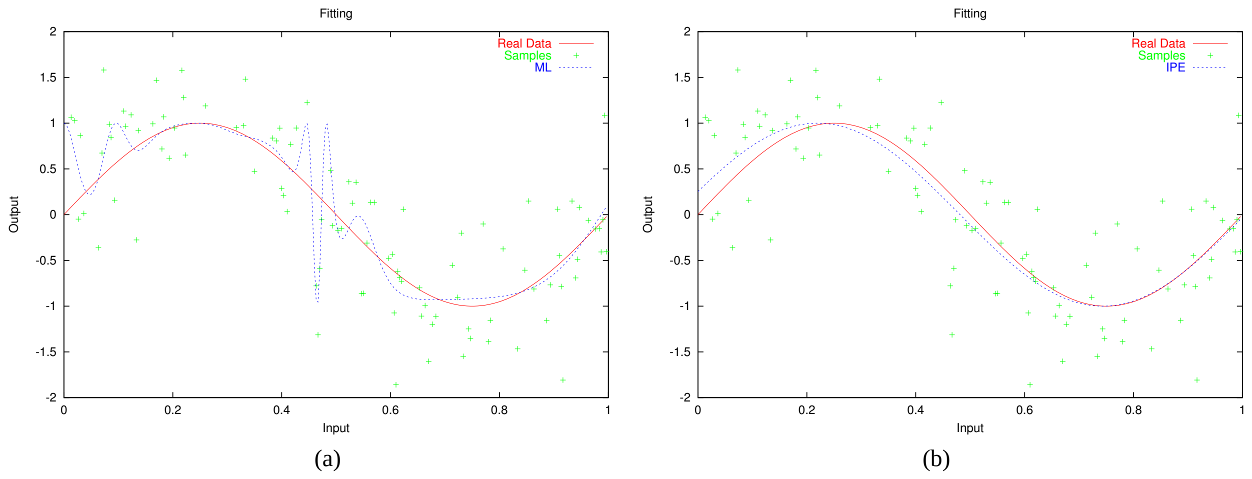


Fig. 12. Sum-of-squares error (a) and the approximated Bayesian fitness (b).

Table 2. Number of weights in the models (10 runs average).

	Final population average	Final best individual
Sum-of-squares error	488.92	433.6
Approximate Bayes	5.075	2

explore the space of neural networks and select the most probable RNN in the light of the prior probability and observed data. This algorithm was tested on different cases, both natural and artificial, and it proved to select relatively simple models which are well matched to the data and have good generalization capabilities. Moreover, experiments proved that ELeaRNT is not affected by the "bloat phenomenon", which is a structural problem of genetic algorithms.

In the past (Matteucci, 2002b), we used an ELeaRNT version making use of the classical sum-of-squares error function instead of the Bayesian fitness. On the benchmark of Section 5.3 we compared that version of the algorithm with a version of ELeaRNT using an approximation of the Bayesian fitness presented in this paper, and the results of Fig. 12 present the best two models learned by the two algorithms. Also, the difference in their complexity is astonishing as reported in Table 2. These were only preliminary results obtained with an approximated Bayesian fitness, but we consider them quite promising. In future works, we plan an extensive comparison with "adversarial" genetic algorithms or fitness functions. Notice that the approximated Bayes fitness used in preliminary experiments of Table 2 was practically equivalent to the Bayes Information Criterion and did not require the determination of the Hessian matrix. Thus with a computational cost comparable with the classical sum-of-squares error fitness function, a simple approximation Bayesian fitness is able to obtain good results with respect to model selection.

From a more general point of view, the present work was focused on the establishment of a probabilistic approach to apply evolutionary computation techniques in a Bayesian framework for the model selection. The work presented in this paper could be easily extended to the Bayesian model averaging since we could use the final population, or part of it, for averaging models predictions. Moreover, since the most difficult step in the Bayesian design of an adaptive model is the prior definition, it might be possible to use a stratified procedure in order to adapt during learning an empirical prior using the information about the individuals in previous populations. Finally, a logical step forward should extend such a framework to other families of models like decision trees, Bayesian networks, and learning classifier systems.

Acknowledgements

This work was partly supported by the project P.R.I.N. 2003 on *Tecnologie software e modelli di conoscenza per progetto, sviluppo e sperimentazione di sistemi robotici multi-agente in ambienti reali,* funded by M.I.U.R. We would like to thank Prof. Andrea Bonarini for the suggestions and help given in improving the overall quality of the paper.

References

Angeline P.J. (1994): *Genetic Programming and Emergent Intelligence*, In: Advances in Genetic Programming (Jr. Kinnear and E. Kenneth, Eds.). — Cambridge, MA: MIT Press, pp. 75–98.

Bebis G., Georgiopoulos M. and Kasparis T. (1997): *Coupling weight elimination with genetic algorithms to reduce network size and preserve generalization*. — Neurocomput., Vol. 17, No. 3–4, pp. 167–194.

Bernardo J.M. and Smith A.F.M. (1994): *Bayesian Theory*. — New York: Wiley.

Bishop C.M. (1995): *Neural Networks for Pattern Recognition*. — Oxford: Oxford University Press.

Castellano G., Fanelli A.M. and Pelillo M. (1997): *An iterative pruning algorithm for feedforward neural networks*. — IEEE Trans. Neural Netw., Vol. 8, No. 3, pp. 519–531.

Chib S. and Greenberg E. (1995): *Understanding the Metropolis-Hastings algorithm*. — Amer. Stat., Vol. 49, No. 4, pp. 327–335.

Denison D.G.T., Holmes C.C., Mallick B.K. and Smith A.F.M. (2002): *Bayesian Methods for Nonlinear Classification and Regression*. — New York: Wiley.

Dudzinski M.L. and Mykytowycz R. (1961): *The eye lens as an indicator of age in the wild rabbit in Australia*. — CSIRO Wildlife Res., Vol. 6, No. 1, pp. 156–159.

Flake G.W. (1993): *Nonmonotonic activation functions in multilayer perceptrons*. — Ph.D. thesis, Dept. Comput. Sci., University of Maryland, College Park, MD.

Fletcher R. (1987): *Practical Methods of Optimization*. — New York: Wiley.

Goldberg D.E. (1989): *Genetic Algorithms in Search, Optimization, and Machine Learning*. Reading, MA: Addison-Wesley.

Gull S.F. (1989): *Developments in maximum entropy data analysis*, In: Maximum Entropy and Bayesian Methods, Cambridge 1998 (J. Skilling, Ed.). — Dordrecht: Kluwer, pp. 53–71.

Hancock P.J.B. (1992): *Genetic algorithms and permutation problems: A comparison of recombination operators for neural net structure specification*. — Proc. COGANN Workshop, Int. Joint Conf. *Neural Networks*, Piscataway, NJ, IEEE Computer Press, pp. 108–122.

Hashem S. (1997): *Optimal linear combinations of neural networks*. — Neural Netw., Vol 10, No. 4, pp. 599–614.

Hassibi B. and Stork D.G. (1992): *Second order derivatives for network pruning: Optimal Brain Surgeon*, In: Advances in Neural Information Processing Systems (S.J. Hanson, J.D. Cowan and C. Lee Giles, Eds.). — San Matteo, CA: Morgan Kaufmann, Vol. 5, pp. 164–171.

Hastings W.K. (1970): *Monte Carlo sampling methods using Markov chains and their applications*. — Biometrika, Vol. 57, pp. 97–109.

Haykin S. (1999): *Neural Networks. A Comprehensive Foundation (2nd Edition)*. — New Jersey: Prentice Hall.

Hoeting J., Madigan D., Raftery A. and Volinsky C. (1998): *Bayesian model averaging*. — Tech. Rep. No. 9814, Department of Statistics, Colorado State University.

Hornik K.M., Stinchcombe M. and White H. (1989): *Multilayer feedforward networks are universal approximators*. — Neural Netw., Vol. 2, No. 5, pp. 359–366.

Liu Y. and Yao X. (1996): *A population-based learning algorithm which learns both architectures and weights of neural networks*. — Chinese J. Adv. Softw. Res., Vol. 3, No. 1, pp. 54–65.

Lovell D. and Tsoi A. (1992): *The performance of the neocognitron with various s-cell and c-cell transfer functions*. — Tech. Rep., Intelligent Machines Laboratory, Department of Electrical Engineering, University of Queensland.

MacKay D.J.C. (1992): *A practical Bayesian framework for backpropagation networks*. — Neural Comput., Vol. 4, No. 3, pp. 448–472.

MacKay D.J.C. (1995): *Probable networks and plausible predictions — a review of practical Bayesian methods for supervised neural networks*. — Netw. Comput. Neural Syst., Vol. 6, No. 3, pp. 469–505.

MacKay D.J.C. (1999): *Comparison of approximate methods for handling hyperparameters*. — Neural Comput., Vol. 11, No. 5, pp. 1035–1068.

Mani G. (1990): *Learning by gradient descent in function space*. — Tech. Rep. No. WI 52703, Computer Sciences Department, University of Winsconsin, Madison, WI.

Matteucci M. (2002a): *ELeaRNT: Evolutionary learning of rich neural network topologies*. — Tech. Rep. No. CMU–CALD–02–103, Carnegie Mellon University, Pittsburgh, PA.

Matteucci M. (2002b): *Evolutionary learning of adaptive models within a Bayesian framework*. — Ph.D. thesis, Dipartimento di Elettronica e Informazione, Politecnico di Milano.

Montana D.J. and Davis L. (1989): *Training feedforward neural networks using genetic algorithms*. — Proc. 3rd Int. Conf. *Genetic Algorithms*, San Francisco, CA, USA, pp. 762–767.

Pearlmutter B.A. (1994): *Fast exact multiplication by the Hessian*. — Neural Comput., Vol. 6, No. 1, pp. 147–160.

Press W.H., Teukolsky S.A., Vetterling W.T. and Flannery B.P. (1992): *Numerical Recipes in C: The Art of Scientific Computing*. — Cambridge, UK: University Press.

Ronald E. and Schoenauer M. (1994): *Genetic lander: An experiment in accurate neuro-genetic control*. — Proc. 3rd Conf. *Parallel Problem Solving from Nature*, Berlin, Germany, pp. 452–461.

Rumelhart D.E., Hinton G.E. and Williams R.J. (1986): *Learning representations by back-propagating errors*. — Nature, Vol. 323, pp. 533–536.

Stone M. (1974): *Cross-validation choice and assessment of statistical procedures*. — J. Royal Stat. Soc., Series B, Vol. 36, pp. 111–147.

Tierney L. and Kadane J.B. (1986): *Accurate approximations for posterior moments and marginal densities*. — J. Amer. Stat. Assoc., Vol. 81, pp. 82–86.

Tikhonov A.N. (1963): *Solution of incorrectly formulated problems and the regularization method*. — Soviet Math. Dokl., Vol. 4, pp. 1035–1038.

Wasserman L. (1999): *Bayesian model selection and model averaging*. — J. Math. Psych., Vol. 44, No. 1, pp. 92–107.

Weigend A.S., Rumelhart D.E. and Huberman B.A. (1991): *Generalization by weight elimination with application to forecasting*, In: Advances in Neural Information Processing Systems, Vol. 3 (R. Lippmann, J. Moody and D. Touretzky, Eds.). — San Francisco, CA: Morgan-Kaufmann, pp. 875–882.

Williams P.M. (1995): *Bayesian regularization and pruning using a Laplace prior*. — Neural Comput., Vol. 7, No. 1, pp. 117–143.