A Node Pruning Algorithm Based on a Fourier Amplitude Sensitivity Test Method

Philippe Lauret, Eric Fock, Thierry A. Mara

A node pruning algorithm based on a Fourier amplitude sensitivity test method

Philippe Lauret, Eric Fock, *IEEE Member* and Thierry Alex Mara

Abstract

In this paper, we propose a new pruning algorithm to obtain the optimal number of hidden units of a single layer of a fully connected neural network. The technique relies on a global sensitivity analysis of model output (SAMO). The relevance of the hidden nodes is determined by analysing the Fourier decomposition of the variance of the model output. Each hidden unit is assigned a ratio (the fraction of variance which the unit accounts for) that allows their ranking. This quantitative information therefore leads to a suggestion of the most favourable units to eliminate. Experimental results suggest that the method can be seen as an effective tool available to the user in controlling the complexity in neural networks.

Index Terms

Pruning, Global sensitivity analysis, Variance decomposition, Fourier analysis, Feedforward neural networks

Manuscript received April 15, 2003

Philippe Lauret and Thierry Alex Mara are Assistant Professors at the University of La Reunion and researchers in the Industrial Engineering Laboratory

Eric Fock is a PhD Candidate in the same laboratory

Corresponding Author: eric.fock@univ-reunion.fr

A node pruning algorithm based on a Fourier amplitude sensitivity test method

I. Introduction

According to Bishop [1] , a central issue in the application of feed-forward neural networks is the determination of the appropriate level of complexity. The latter is governed by the number of coefficients or weights of the neural network. This search of the optimal model is of vital importance for the generalization property of the neural network (NN). The main techniques used to control this complexity are [1]–[3]: architecture selection, regularization, early stopping and training with noise, the last three being closely related. Bishop [1] argues that for most applications, techniques based on regularization should be preferred. One of the most popular regularization terms is the so-called weight decay term that consists in the sum of the squares of the parameters. Unfortunately, the simple weight decay term is inconsistent with known scaling properties of network mappings (see [4] for details). A consistent regularizer can be obtained by assigning separate regularizers to the first-layer weights and to the second-layer weights.The optimal weight decay term (the one that gives the best trade-off between bias and variance) can be determined through cross-validation. However, this procedure would be computationally expensive, especially if regularization schemes with multiple weight decay terms are to be considered. The bayesian approach [5] allows the values of regularization coefficients to be automatically tuned during the training process without the need to use cross-validation. Nonetheless, bayesian techniques are based on some simplifying assumptions. The most important one is that a gaussian approximation of the posterior distribution of the weights is needed in order to make the integrations over the weight space analytically tractable. Indeed, this approximation does not take into account the problem of multiple minima of the error function (although some techniques tend to moderate this statement [6]).

As for the technique of architecture selection, one of the simplest ways involves the use of networks with a single hidden layer, in which the number of free parameters is controlled by adjusting the number of hidden units. Practically, a set of networks ranging from 1 to p hidden units is trained. The performance of the networks is evaluated on a test set. The network that exhibits the best generalization performance is selected. However, this technique is computationally demanding and therefore usually restricted to networks having a single hidden layer. Other approaches consist in growing or pruning the network structure during the training process. The approach taken by the pruning methods is to start with a relatively large network and gradually remove either connections or complete units [4]. Nonetheless, one may notice that network architecture selection changes the actual number of adaptative parameters in the NN while regularization controls the effective number of parameters. Different methods to pruning have been developped. For a review covering the pruning methods, see [3]. The most popular ones of these are Optimal Brain Damage (OBD) [7] and Optimal Brain Surgeon OBS [8]. There exists an extension of OBD for pruning irrelevant hidden units and input units called Optimal Cell Damage (OCD) [9].

By considering the change in the error function due to small changes in the values of the weights, a measure of

the relative importance of the different weights or saliency can be computed. The weights with low saliencies are deleted. More precisely, both methods (OBD and OBS) use a second order Taylor expansion of the error function to estimate how the training error will change as the weights are perturbed. These methods are based on assumptions in order to reduce complexity in calculating the weights saliencies [4], [7], [8], [10]. First, both methods make the assumption that the pruning will be performed after training has converged to a minimum i.e. the gradient is zero (”extremal” assumption). Second, they assume that the error function is nearly quadratic in order to neglect the last term of the Taylor expansion (”quadratic” approximation). The OBD method additionally assumes that the off-diagonal terms of the Hessian matrix are zero.

Engelbrecht [10] proposed a new pruning algorithm based on output sensitivity analysis that consists in a first-order Taylor expansion of the NN output function. He showed that OBD (which is an objective function sensitivity analysis) and output sensitivity analysis are conceptually the same under the assumptions of OBD. The method is based on variance analysis of sensitivity information given by the derivatives of the NN output with respect to the parameters. It is quite a powerful method since the neural structure inherently contains all the information to compute efficiently these derivatives [11]. The basic idea of the technique is that a parameter with a low average sensitivity and with a variance in sensitivity which is not significantly different from zero over all patterns has little or no effect on the output of the NN considered. The method called Variance Nullity Pruning (VNP) [11] is not based on any assumptions to reduce the complexity in calculating the saliencies of the parameter. However, since the sensitivity information is given by the derivatives of the NN output with respect to the parameters, the network should be well trained to accurately approximate the true derivatives [10]. Indeed, it has been proven that as the NN converges to the underlying function so all derivatives also converge to the true derivatives [12]. The VNP algorithm has been also used to prune irrelevant input units. Prior to the VNP algorithm, Zurada [13] used a perturbation-based sensitivity method for inputs’ pruning.

The above approaches are derivative-based methods. The output sensitivity analysis developed by Engelbrecht [10] can be grouped in the so-called *local methods* of sensitivity analysis of model output (SAMO) [14]. But the analysis remains inherently local. Small variations in the parameter values do not change the local sensitivities but a significantly different parameter set may result in a completely different sensitivity pattern. Moreover, the quality and reliability of the results of this type of analysis depends on how well the Taylor expansion approximates the original model.

There exists a second sensitivity analysis (SA) school, the *global* sensitivity analysis of model output, which is more ambitious in two aspects: first, the space of the parameters (also called *factors* or *input factors* in the SA terminology) is explored within a finite region and, second, the variation of the output induced by a factor is taken globally - that is averaged over the variation of all the factors [14].

In this paper, we propose a new technique to obtain the optimal number of hidden units of a single layer of a fully connected network. This technique relies on a global Sensitivity Analysis of Model Output (SAMO). A global SA method, the EFAST method [15] (which stands for Extended Fourier Amplitude Sensitivity Test), is used to quantify the relevance of the hidden units. Thus, in our study, the output of hidden units are the factors of interest.

Sensitivity analysis of large dimensional overtrained neural networks are conducted in order to assess the relative importance of each hidden unit on the NN output. This is made possible by computing the contribution of each hidden unit on the NN output.

EFAST is an extension of the FAST method [14], [16]. The key idea of FAST is that all the factors are oscillated around their nominal value from one simulation run to another. The importance of a factor is determined by analysing the Fourier decomposition of the model response. The FAST method computes a ratio, which is called the main effect or first-order sensitivity index (S_h), that ranks quantitatively the different input factors. The FAST method is independent of any assumptions about the model and works for monotonic and non-monotonic models.

Developments and improvements of the FAST derivative methods are recent. Saltelli [15] extended the classical approach (thus giving the EFAST method) to perform the total sensitivity index (ST_h). The term "total" here means that the factor's main effect, as well as all the interactions involving that factor, are included in the ratio. In this paper, we will use this method. The ranking of the hidden units could lead to a suggestion of the most favourable ones to eliminate.

Section II proposes a brief introduction to SAMO. In section III, we will describe the FAST and the EFAST methods. Section IV illustrates the application of the EFAST method to the area of NN. Section V describes the experimental setup while section VI reports the results. The performance of the method has been evaluated (through extensive experimentation) on nine real-world problems issued mainly from the international benchmark Proben1 [17]. Section VII will draw the conclusions.

II. Introduction to SAMO

According to Saltelli [14], in the context of numerical modelling, SA means very different things to different people. Helton [18] proposed a review of the different techniques. But all these approaches have in common the aim to investigate how a given computational model responds to variation in its input factors. The term *input factor* must be interpreted in a very broad sense: a factor is a quantity that can be changed in the specification of the model prior to its execution. A factor can be an initial condition, a parameter, etc .... By considering, without loss of generality, a model $f(\bullet)$ such that $Y = f(Z_1, Z_2, \ldots, Z_p)$, SA estimates the effects of the p input factors $Z_1, Z_2, \ldots, Z_p$ on the output Y. The effect of a factor is the change in the response obtained by changing the value assumed by that factor.

Different types of analysis are possible with SA. The interested reader may refer to Saltelli [14] for more details. For instance, modellers may conduct SA in order to determine insignificant model parameters which can thus be eliminated, in other words parameters not affecting the variation of the output. In this way, irrelevant parts of the model can be dropped, or a simpler model can be built or extracted from a more complex one (model lumping).

The purpose of SA is manifold. SA either local or global have been used in numerous fields:

1) as a tool to understand mechanisms in complex chemical kinetics reaction schemes [19]
2) as a means to analyse fish population dynamics [20]
3) investigating the structure of an environmental numerical model related to climatic change studies [21]

4) analyzing a complex geological waste disposal system [18], etc.

A. How to perform SA

There are several procedures to conduct sensitivity analysis. The most common SA is sampling-based. Fig. 1 represents a sampling-based sensitivity analysis in which the model is executed repeatedly for combinations of values sampled from the distribution (assumed known) of the input factors. The following steps can be identified [14]:

1) define the model; its input factors and output variable(s)
2) assign probability density functions or ranges of variation to each input factor
3) generate an input matrix through sampling design
4) evaluate the output
5) assess the influences or relative importance of each input factor on the output variable

At step 4, an empirical probability distribution for the output can be created which may lead to a first step of uncertainty analysis. Mean, standard deviation, confidence bounds etc. can be estimated. After quantifying the variation of the output, the next step, sensitivity analysis, consists in apportioning the variance of the output according to the input factors. A possible representation of the results can be a pie chart that decomposes the variance (D_y) of the output into the percentages that each factor is accounting for. So, the variance decomposition may allow the identification of the most influential factors.

Fig. 1: General scheme of a quantitative Sensitivity Analysis method. The total variance is apportioned to the various input factors, as shown by the pie diagram.

B. Different sensitivity indices

There are different methods to perform sensitivity analysis of model output [14]. They all rely on the estimation of a sensitivity index. Consider again a $p-$factor model $Y = f(Z_1, Z_2, \ldots, Z_p)$.

In the following, let us denote by z_i the standardized factor (mean 0 and variance 1) relative to Z_i. To introduce the different sensitivity indices, it is convenient to consider, without loss of generality, that the model response under interest can be expressed in the form of the following polynomial expansion :

$$Y = Y_0 + \sum_{i=1}^{p} \beta_i z_i + \sum_{i=1}^{p}\sum_{j=1}^{p} \beta_{ij}\ z_i z_j + \sum_{i=1}^{p}\sum_{j=1}^{p}\sum_{k=1}^{p} \beta_{ijk} z_i z_j z_k + \ldots, \tag{1}$$

where,

Y is the model response,

the β_i's are the first order regression coefficients,

the β_{ij}'s are the second order regression coefficients, and so on ....

$z_i z_j$ represents the first-order interaction between the factors i and j....

Quantitative SA methods are usually based on the estimation of one of the three following sensitivity indices:

- the linear effect, a sensitivity index based on the β_i's alone
- the main effect, a sensitivity index based on the β_i's (linear effect), β_{ii}'s (quadratic effect), β_{iii}'s (cubic effect) and so on ...
- the total effect which is based on the β_i's, β_{ij}'s, β_{ijk}'s ... for a given i and all $(j, k, \ldots)$ i.e. all the coefficients involving the factor Z_i

Generally, the contribution of an interaction to the response variation is less than a least-order interaction and linear effect. However, the entire non-linear effects may have an important contribution to the model response variation. If we develop (1) up to an M-order polynomial (therefore we expect the high-order coefficients to have a negligible influence on the variation of the output), we can write, that

$$Y = Y_0 + \sum_{i_1=1}^{p} \beta_{i_1} z_{i_1} + \sum_{i_1=1}^{p} \sum_{i_2=1}^{p} \beta_{i_1 i_2} z_{i_1} z_{i_2} + \ldots + \sum_{i_1=1}^{p} \sum_{i_2=1}^{p} \cdots \sum_{i_M=1}^{p} \beta_{i_1 i_2 \ldots i_M} z_{i_1} z_{i_2} \ldots z_{i_M} + \varepsilon \tag{2}$$

M is called the interference factor (usually set to 4 or 6 in the SA community) and ε is the error term.

On one hand, if the model is non-linear but the factors are varied in a small range, the first-order regression coefficients is an adequate index for SA because non-linearities can be neglected. In that case, the method employed to estimate the β_i's belongs to the local SA methods. On the other hand, when factors are strongly varied over order of magnitude, then, the entire non-linearities cannot be neglected anymore and must be accounted for into the sensitivity index. For this purpose, a global SA variance-based method is employed.

C. Variance based-methods

Among the global methods, one may distinguish two variance-based methods: the Sobol' method [22] and the FAST method. Variance-based methods aim to estimate the quantity

$$S_h = \frac{Var_{Z_h}\left[E\left(Y|Z_h = z_h\right)\right]}{Var(Y)} = \frac{Amount\ of\ the\ model\ response\ variance\ due\ to\ factor\ Z_h\ only}{The\ model\ response\ variance} \tag{3}$$

where Z_h denotes an input factor, Y the model response, $E(Y|Z_h = z_h)$ the expectation of Yconditional on a fixed value of Z_h and the variance Var_{Z_h} is taken over all the possible values of Z_h. This ratio (S_h) represents the main effect. It is called the first-order index in the SA terminology. Thus, the main effect of a factor represents the average effect of that factor on the response or conversely these methods allow the computation of that fraction of the variance of a given model output which is due to each input factor.

In addition to the computation of the first-order indices, Sobol' method as well as the Extended FAST (EFAST) method also provide an estimation of the total sensitivity index (ST_h). The total effect includes the main effect as well as all the interaction terms involving that factor. The total effect is defined by :

$$ST_h = \frac{Amount\ of\ the\ model\ response\ involving\ factor\ Z_h}{The\ model\ response\ variance} \tag{4}$$

A model is said additive when the response is non-linear but interactions are negligible. In that case, the main effects are the suitable indices for SAMO because $\sum_{h=1}^{p} S_h \approx 1$. Otherwise, the total effects are the appropriate indices to rank the factors by order of importance and $\sum_{h=1}^{p} ST_h > 1$ that is $\sum_{h=1}^{p} S_h < 1$.

Sobol' method is a Monte-Carlo based method that consists on performing multiple model evaluations with randomly selected input factors. FAST is based on the Fourier decomposition of the variance in the frequency domain. Both methods are especially suited for a quantitative model-independent global sensitivity analysis. The computational cost of these methods is the number of models evaluations required and is a function of the number of input factors and the complexity of the model. The ever-increasing power of computers tend to make these global methods affordable for a large class of models.

III. Variance-based methods in the spectral domain : the FAST and EFAST methods

A. Introduction

To introduce the FAST and EFAST methods, we consider again the polynomial expansion.

Let $[a_h, b_h]$ be the range of variation of the factor Z_h. Let us suppose that N simulation runs are performed by varying each factor as follows: $Z_h^{(n)} = \frac{b_h+a_h}{2} + \frac{b_h-a_h}{2} \sin\left(\omega_h s^{(n)}\right)$ with $s(n) = 2\pi n/N$, ω_h the (integer) frequency assigned to factor Z_h and n the simulation number.

It is straightforward to note that $z_h^{(n)} = \sin\left(\omega_h s^{(n)}\right)$ and that (1) becomes :

$$\begin{aligned} Y^{(n)} &= Y_0 + \sum_{i=1}^{p} \beta_i \sin\left(\omega_i s^{(n)}\right) + \sum_{i=1}^{p}\sum_{j=1}^{p} \beta_{ij} \sin\left(\omega_i s^{(n)}\right) \sin\left(\omega_j s^{(n)}\right) \\ &+ \sum_{i=1}^{p}\sum_{j=1}^{p}\sum_{k=1}^{p} \beta_{ijk} \sin\left(\omega_i s^{(n)}\right) \sin\left(\omega_j s^{(n)}\right) \sin\left(\omega_k s^{(n)}\right) + \dots \end{aligned} \tag{5}$$

The previous relationship leads to the following conclusions :

- the linear effect of Z_h corresponds to the Fourier amplitude at the fundamental frequency ω_h.
- S_h is obtained by considering the Fourier amplitudes at the fundamental frequency ω_h (linear effect), the first harmonic (quadratic effect), the second harmonic (cubic effect) and so on .... This is the basic idea of the FAST method.
- Interactions induce new frequencies that are linear combinations of interacting factors' frequencies. Consequently, ST_h can be computed by considering all the Fourier amplitudes involving Z_h. One way to isolate these frequencies in the spectral domain, is to choose ω_h very high as compared to the other frequencies (denoted by $\omega_{\sim h}$) so that all the spectral components involving Z_h do not overlap in the low frequency region (where the spectral components do not concern Z_h). Such an approach reminds the frequency modulation technique and is called the EFAST method in SA (Extended FAST).

B. The FAST method

FAST enables the estimation of the total output variance (D_y) and the contribution of individual input factors to this variance, that is, the first order sensitivity indices. In FAST, each input factor Z_h is related to a frequency ω_h and a set of suitably defined parametric equations

$$Z_h(s) = G_h(\sin(\omega_h s)) \qquad \forall\ h = 1, 2, \ldots, p \tag{6}$$

allows each factor to vary in range, as the new parameter s is varied (where s is a scalar variable varying in the range $-\infty < s < \infty$) . The parametric equations define a curve that systematically explores the input factors' space. As s varies, all the factors oscillates at the corresponding driving frequency ω_h and their range is systematically explored.

Different transformation functions have been proposed [15], [16]. For the FAST method (and EFAST method), a parametric representation of the form

$$Z_h(s) = \frac{1}{2} + \frac{1}{\pi} \arcsin\left(\sin\left(\omega_h s\right)\right) \tag{7}$$

is often used. This transformation allows a better coverage of the factors' space since it generates samples that are uniformly distributed in the range $[0, 1]$ (see Fig. 2).

Fig. 2 : Plot of the transformations function (defined by (7)) and its respective empirical distribution.

Notice however that, if $[a_h, b_h]$ is the range of variation of the factor Z_h, each factor oscillates in the range $[a_h, b_h]$ along the curve defined by,

$$Z_h(s) = \frac{b_h + a_h}{2} + \frac{b_h - a_h}{\pi} \arcsin\left(\sin\left(\omega_h s\right)\right) \tag{8}$$

In the present application, the output of the hidden nodes will be varied according (8).

As each factor Z_h oscillates periodically between $[a_h, b_h]$ at the corresponding frequency ω_h, the model output Y exhibits different periodicities that result from the combination of the different frequencies $\omega_{i=1,\cdots,p}$, whatever the model f is. As stated by [15], if the h^{th} factor has strong influence on the output, the oscillations of Y at frequency ω_h shall be of high amplitude. This is a basis for computing a sensitivity measure for the factor Z_h based on the evaluation of the Fourier amplitudes at the corresponding frequency ω_h and its harmonics. In other words, large Fourier amplitudes at the fundamental frequency ω_h and and its harmonics indicates that the output is sensitive to the input factorZ_h.

Cukier [16] showed that, if an appropriate set of integer frequencies $\omega_{i=1,\ldots,p}$ is chosen, then

$$f(s) = f(Z_1(s), Z_2(s), \ldots, Z_h(s), \ldots, Z_p(s))$$

is 2π-periodic $(-\pi < s < \pi)$. So, $f(s)$ may be expanded in a Fourier series of the form:

$$f(s) = \sum_{j=-\infty}^{+\infty} (A_j \cos \omega_j s + B_j \sin \omega_j s) \tag{9}$$

where the Fourier coefficients are defined as

$$A_j = \frac{1}{2\pi} \int_{-\pi}^{\pi} f(s) \cos(\omega_j s) ds \tag{10}$$

$$B_j = \frac{1}{2\pi} \int_{-\pi}^{\pi} f(s) \sin(\omega_j s) ds \tag{11}$$

and $-\pi < s < \pi$.

So, N equally spaced sample points are required to perform the Fourier analysis. N represents the sample size and coincides with the number of model evaluations (that is the number of simulation runs).

One major advantage in shifting the analysis into the frequency domain is that the spectral decomposition is equivalent to a variance decomposition. An analysis of variance is possible because Parseval's theorem states that

$$D_y = Var(Y) = 2 \sum_{k=1}^{+\infty} \left(A_k^2 + B_k^2\right) \tag{12}$$

The portion of the variance of Y explained by Z_h alone is

$$D_h = Var_{z_h} \left[E\left(Y|Z_h\right)\right] = 2 \sum_{k=1}^{+\infty} \left(A_{k\omega_h}^2 + B_{k\omega_h}^2\right) \tag{13}$$

where $A_{k\omega_h}$ and $B_{k\omega_h}$ denote the Fourier coefficients for the fundamental frequency and its higher harmonics $k\omega_h$. Consequently, the expansion of the main effect is given by

$$S_h = \frac{D_h}{D_y} = \frac{Var_{z_h}\left[E\left(Y|Z_h\right)\right]}{Var\left(Y\right)} = \frac{2 \sum\limits_{k=1}^{+\infty} A_{k\omega_h}^2 + B_{k\omega_h}^2}{Var\left(Y\right)} \tag{14}$$

We stated above that in order to evaluate the main effect of Z_h, one must calculate the Fourier coefficients at the fundamental frequency ω_h and all the harmonics. As mentioned earlier (see section II. B and (2)) , only the first $(M-1)$ harmonics are considered so that the first-order sensitivity index is approximated by :

$$S_h = \frac{Var_{x_h}\left[E\left(Y|X_h\right)\right]}{Var\left(Y\right)} \simeq \frac{2 \sum\limits_{k=1}^{M} A_{k\omega_h}^2 + B_{k\omega_h}^2}{Var(Y)} \tag{15}$$

where M is called the interference factor (usually set to 4 or 6 in the SA community).

In the FAST approach, the number of simulation runs represents the sampling frequency and, to satisfy the Nyquist criterion, must be equal (at least) to $N = 2M\omega_{\max} + 1$ where $\omega_{\max} = \max\limits_{i \in [1,p]} (\omega_i)$.

Notice that the variance $Var(Y)$ can be evaluated in the frequency domain through the following relationship:

$$Var(Y) = 2 \sum_{\omega=1}^{(N-1)/2} A_\omega^2 + B_\omega^2, \tag{16}$$

where A_ω and B_ω denote the Fourier coefficients at frequency ω.

Thus, to estimate the main effects, N model evaluations are required. At this point, some comments must be made. First, the set of integer frequencies $\omega_{i=1,\cdots,p}$ must be properly chosen in order to avoid interferences up to order M. Let us recall that the Fourier coefficients evaluated at the input frequency ω_h and its multiples $k\omega_h$ give the sensitivity of the output to the h^{th} factor. If interferences occur at a given frequency, then the analysis becomes irrelevant by overestimating the main effect. Put differently, the difficulty with such an approach is to choose the frequency set so that the frequencies generated by the $M^{th}-$ order non-linearities do not equal $n\omega_i, n = 1, \ldots, M$ and $i = 1, \ldots, p$. Second, N is also constrained by the number p of inputs factors, given that, as the number of factors increases, it is necessary to choose higher $\omega_{\max}$ in order to obtain a set of frequencies free of interferences. Thus, even for a relatively small number of parameters (say 20), the choice of the set of frequencies will not be easy. This fact may render the method difficult to use in practice.

C. The EFAST method

Saltelli [15] proposed an extension of the FAST method that allows to cope more easily with this problem of interferences. Moreover, the new method computes both the main effect $(S_h)$and total effect(ST_h) using the same set of models evaluations. This is made possible by assigning the factor of interest h a "high" value for its frequency ω_h and a set of "low" frequency values to the remaining set of factors $Z_{\sim h}$ (in the following, we set $Z_{\sim h} = Z_1, Z_2, \ldots, Z_{h-1}, Z_{h+1}, \ldots, Z_p$, i.e. all the factors except the h_{th} factor). More precisely, the spectrum of a model response is divided into two areas (see Fig. 3).

Indeed, if we set $\omega_h = 2M\ max(\omega_{\sim h})$ where $max(\omega_{\sim h})$ is the highest frequency assigned to the set of factors $Z_{\sim h}$, then it will ensure that the frequencies generated by the $M-$order interactions involving Z_h will not interfere with the frequencies induced by the $M-$order non-linearities involving $Z_{\sim h}$. Then, the estimation of the total sensitivity index by the EFAST approach can be expressed as follows :

$$ST_h \simeq \frac{\sum_{\omega=Mmax(\omega_{\sim h})+1}^{(N-1)/2} \left(A_\omega^2 + B_\omega^2\right)}{\sum_{\omega=1}^{(N-1)/2} \left(A_\omega^2 + B_\omega^2\right)} \tag{17}$$

with $N = 2M\omega_h + 1$ as ω_h is the highest frequency assigned.

Conversely, the first order S_h is obtained as in classical FAST (see (15)).

One may see that the problem of interference is easier to manage than in the classical FAST since it may be easier to find a couple of frequencies (ω_h and $max\left(\omega_{\sim h}\right)$) that do not interfere up to an arbitrary high M. Interferences are avoided as long as $\omega_h \geq 2M\ max\left(\omega_{\sim h}\right)$.

Saltelli [15] proposed an algorithm to select ω_h (consequently $max\left(\omega_{\sim h}\right)$) and the frequencies in the complementary set $[1,\ max\left(\omega_{\sim h}\right)]$ for a given number of simulation runs N (see section IV.B). In order to obtain a better coverage of the input factors' space, one must assign distinct frequencies to the factors of the complementary set. However, to limit the number of model evaluations, it is possible (to some extent) to assign the same frequency to

two (or more) different factors of the complementary set.

Fig. 3 : The spectrum of a model response using the EFAST approach . The spectrum is divided into two regions: the first region $\left[1, \frac{\omega_h}{2} = Mmax(\omega_{\sim h})\right]$ *contains the frequencies involving all the factors except those of factor* Z_h *and the second region* $[Mmax(\omega_{\sim h}) + 1, (N-1)/2]$ *contains the effects of factor* Z_h *located in the high frequencies.*

As stated above, the total number of simulation runs required to compute the total effect of factor Z_h alone is $2M\omega_h + 1$ as h is the highest frequency assigned. To estimate the sensitivity index for another factor, a permutation of the frequencies is necessary, because the "high" frequency must be assigned to the factor of interest. Hence, to compute the entire p total sensitivity indices $p(2M\omega_h + 1)$ simulation runs are necessary.

Among the SA methods, the total sensitivity index is undoubtedly the best guide to rank quantitatively the factors by order of importance. Indeed, even if this occurs rarely, interaction effects on a model response may be more predominant than the main effects. So, whether the interaction effects are taken into account or not, the analysis may result in a different ranking of the factors' importance.

The results of the analysis can be displayed in an intuitive graphical way by normalizing each ST_h by the sum of $ST_i,\ i = 1, \ldots, p$. The normalized indices (S_n) can be plotted in the form of a pie chart, hence showing the fraction of variance which the factor accounts for. However, when dealing with complex models with a large number of parameters and for which the cost of one model evaluation is high, estimation of the total sensitivity indices may require a very high computational effort.

IV. USING THE EFAST METHOD TO OBTAIN THE OPTIMAL ARCHITECTURE

A. The Method

Regarding the intrinsic structure of an single output NN, one may decompose it into two sub-models. The first one (SM1) is the multi-response relationship between the inputs of the NN ***(x)*** and the output of the hidden units (Z). The second sub-model (SM2) is the single response relationship between the output of the hidden units (Z) and the output of the NN (Y). We state that the relevance of a hidden unit is related to its influence on the NN response. This is the key idea of the method proposed in this paper to determine the optimal architecture of an NN. In our approach, the model is SM2 and the factors are the output of the hidden units (Z).

The different steps of the proposed approach are :

1) Train a "reasonably large" network for some epochs
2) For each factor Z_h (output of the hidden node h), retain its minimal and maximal values a_h and b_h respectively
3) Set the interference factor to $M = 4$ and choose the number of simulation runs N
4) Given M and N, compute the frequency $\omega_h = (N-1)/2M$ to be assigned to the factor Z_h and the frequencies assigned to the other factors in order to perform the EFAST method
5) For each factor Z_h,

- Assign the frequency ω_h to the factor Z_h
- By only considering the SM2 model, perform N simulation runs. The factors are varied according to the curve defined by (8), compute the total effect (ST_h) of the factor Z_h using (17)

6) Given all the total effects $(ST_i (i = 1, 2, \ldots, p)$, compute the percentage contribution (i.e. the normalized indices $S_n = ST_h \Big/ \left(\sum_{i=1}^{p} ST_i \right)$ of each hidden unit to the variation of the output
7) Delete the hidden units that accounts for less of 5% of the output variance

Fig. 4 : The EFAST method applied to pruning of hidden units. Each output of the hidden units constitutes an input factor. All input factors oscillates (each with its own frequency ω_i) according to the curve defined by (8). N samples of the output are evaluated that enable the computation of the percentage contribution of each hidden unit (through the Fourier decomposition of the variance of the output).

At this stage, two points have to be highlighted. First, usually, pruning occurs when the NN has been trained into a minimum of the error function [7], [8] or when overfitting begins (a pruning indicator is detected through the monitoring of the error on a validation set) [10] [23]. It will be shown that for the EFAST pruning method these pre-requisites are not necessary. In other words, in the EFAST method, pruning starts when the NN has been trained for some epochs (and this latter parameter has not to be carefully tuned).

Second, it is also important to note that step 6 of the above procedure exhibits in a quantitative way the relevant units; those that accounts for at least 5% of the variation of the output. Indeed, it will be shown that the EFAST pruning method answers quite satisfactorily to the question of ”how much to prune”.

B. Parameters of the EFAST pruning algorithm

For a given number of hidden units, the parameters of the EFAST pruning algorithm are: N the number of models evaluations, M the interference factor and the set of frequencies assigned to the hidden units (factors). Actually, the choice of M and N determines the set of frequencies assigned to the factors.

First, we set $M = 4$. As discussed earlier, it is common practice in the SA community to set M to 4 or 6. Indeed, the spectral information rapidly decreases when frequency increases. Notice that experiments have been conducted with $M = 6$. But, even if the estimates of the partial variances were more accurate, this setting had no influence on the experimental results.

Second, the choice of N is dictated by the following consideration. As mentioned above, in order to have a better coverage of the factors' space, the frequencies of the complementary set must be distinct from each other. For instance, for a NN with 32 hidden units, it is recommended to choose (at least) $N = 2049$ then leading to $\omega_h = 256,\ max(\omega_{\sim h}) = 32$ and the resulting complementary set of frequencies $\omega_{\sim h} = [1, 2, 3, \cdots, 31, 32]$. So, each factor is assigned a distinct frequency. In the same way, for a NN with 128 hidden nodes, we have (at least) $N = 8193$ (leading to $\omega_h = 1024,\ max(\omega_{\sim h}) = 128$ and $\omega_{\sim h} = [1, 2, 3 \cdots, 127, 128]$ for the complementary set).

However, other values for N are allowed. We chose $N = 1025$ in order to obtain a good trade-off between between computational cost and accuracy of the method. 1025 model evaluations lead to $\omega_h = 128,\ max\left(\omega_{\sim h}\right) = 16$ and the resulting set of frequencies assigned to the other factors $\omega_{\sim h} = [1, 2, \cdots, 15, 16, \cdots\cdots 1, 2, \cdots, 15, 16]$. The pattern $[1, 2, \cdots, 15, 16]$ is duplicated in order to cover the whole range of factors. So, the same frequency is assigned to two (or more) different factors but experiments (see section VI.A.3) show that this choice of $N = 1025$ appears to be consistent when pruning NN having 128 or 32 hidden units.

Table I illustrates the different possibilities given the number of hidden units and the number of simulations runs when the assumed factor of interest is the third.

Table I Frequencies assigned to the input factors given the number of factors and number of simulation runs (the third factor is the factor of interest).

C. The computational cost of the method

For p hidden units, the NN output is given by the following equation: $Y = f(\sum_{j=1}^{p} w_j z_j)$. A single evaluation of the output of the NN output requires $\mathcal{O}(p)$ operations: each term in the sum necessitates one multiplication and one addition while the evaluation of the output activation function represents a small overhead. Thus, the computational cost of the EFAST pruning method is $N \times p \times \mathcal{O}(p)$ with N the number of simulations runs required by the EFAST method.

V. Experimental setup

A. Datasets

Extensive benchmark experiments have been made on nine real-world problems. All these datasets (except EES dataset [24]) are part of Proben1 [17]. The Proben1 benchmark set is a collection of classification and function approximation problems. The latter have between 8 and 120 inputs and between 303 and 7200 examples. The data in Proben1 are encoded for direct neural network use. Three suggested partitioning of the data into training, validation and test sets are given in Proben1. We chose the first pre-partitioning as it is. Table II lists the datasets.

Table II: The datasets, where the type is either c (classification) or a (approximation)

We used only a single output for classification problems while for approximation ones with more than one output (e.g. building), we handled separately each output with a single output NN. For further information on the Proben1 datasets, the interested reader should consult [17].

B. Pruning algorithms

SNNS [25] (Stuttgart Neural Network Simulator) is a simulator for neural networks developed at the Institute for Parallel and Distributed High Performance Systems at the University of Stuttgart. The simulator offers a flexible

and open environment for developing applications on neural networks. This open feature allowed us to implement the EFAST pruning method in SNNS.

Furthermore, five pruning functions are available in SNNS: Optimal Brain Surgeon (OBS), Optimal Brain Damage (OBD), Magnitude Based Pruning (MBP) [26], Skeletonization (SKEL) [27] and Non-Contributing units (NC) [28].

OBS, OBD and MBP are weight pruning methods whereas SKEL and NC are node pruning algorithms. . Rigorously, when pruning hidden units, we cannot compare the weight-oriented pruning methods (OBS, OBD and MBP) with the node pruning algorithms (SKEL, NC and EFAST). However, we have followed the same approach proposed by Engelbrecht [10] who compared its VNP pruning algorithm with MAG, OBS and OBD. For the weight-oriented pruning methods (OBS, OBD and MBP), an hidden unit is deleted if all incoming or all outgoing links to that unit are removed. Obviously, these methods necessitate more pruning steps (than the node pruning algorithms) as one link is deleted per pruning step. This specific treatment led to the computation of an effective number of pruning steps[1] (see formula below). The CPU time is also updated in the same way.

These standard algorithms computes the relevance of each element in order to prune the one with the smallest saliency.

Among these methods, MBP is the simplest one. The saliency of a weight is given by its absolute value and the algorithm eliminates the weight that has the smallest magnitude.

OBD estimates the change in the error function when pruning a certain weight. The saliency of a weight is given by $s_i = \frac{1}{2} h_{ii} w_i^2$ where h_{ii} is the i^{th} element of the hessian matrix (second derivatives of each parameter) and w_i^2 the value of the weight at the minimum of the error function.

For OBS, the saliency of the weight is the quantity $s_i = \frac{1}{2} \frac{w_i^2}{[H^{-1}]_{ii}}$ where H^{-1} is the inverted Hessian. OBS also computes a correction to the remaining weights after the deletion of a parameter in order to minimize the increase in error.

As mentioned above, the popular methods OBD and OBS are based on some assumptions (training to the error minimum, quadratic approximation, zero off-diagonal elements for OBD).

SKEL prunes units by estimating the change of the error function E when the unit is removed. The saliency of a unit is given by $s_j = -\frac{\partial E}{\partial \alpha_j}\Big|_{\alpha_j=1}$ where α_j is called the attentional strength (see [25] and [27] for details).

The NC method uses statistical means to find units that do not contribute to the net's behavior. The output of each unit is observed for the whole pattern set. The units that are removed are the ones that don't vary their output, always show the same output as another unit or always show the opposite output of another unit.

Notice that these methods do not really answer to the question 'how much to prune'. For instance, the authors of OBD suggest to prune 'some' low saliencies. So, these methods operate in a somewhat conservative way in the sense that only one parameter is removed per pruning step. This is the main drawback of these methods. In order to speed up the pruning process, one could remove parameters that are below a given threshold. But the latter must be chosen in an ad hoc fashion or set by some specific rules of thumb.

[1] $Effective\ number\ of\ (node)\ pruning\ steps\ =\ actual\ number\ of\ (weight)\ pruning\ steps * \frac{\#\ of\ units\ removed}{\#\ of\ weights\ deleted}$

C. Training algorithm

All runs were performed using the RPROP algorithm [29] available in SNNS. The three RPROP parameters are set to the following values: 0.1, 0.1 and 50 (See [29] for the meaning of these parameters).

D. The NN architectures

All the experiments were made with networks with one hidden layer of hyperbolic tangent (*tanh*) activation function. The activation function for the NN output was set to the standard sigmoid for classification problems and to the identity function for approximation problems. The pruning methods were compared on NN having 32 and 128 hidden nodes.

For the additive procedure (see section VI B), seven sizes of hidden layer were used: $2, 4, 8, 16, 32, 64, 128$ hidden units.

E. The benchmark procedure

For benchmarking comparisons purpose[2], we have evaluated the performance of the different pruning methods by using the following procedure:

1) Choose a ”reasonably large” NN architecture.
   Some tools are proposed in [30] that may help to shed some light on the term ”reasonably large” .
2) Train the NN for some epochs (100, 500, 1000)
3) Apply the pruning method e.g.:
   a) For MAG, OBS, OBD, SKEL and NC : compute the saliency of each element and delete the element with the smallest saliency.
   b) For EFAST : delete the units that account for less of 5% of the variation of the NN output
4) Retrain the NN for 10% of the first amount of training epochs (e.g. $10, 50, 100$ epochs)
5) Test the reduced NN on a validation set.
   If the validation error deteriorates by more than 10% from the previous iteration or no more hidden units can be deleted at the end of three (unsuccessful) pruning process, go to step 6 otherwise iterate to step 3
6) Test the NN on a test set
7) End of benchmarking procedure

VI. Experimental results

A. Results and discussion for the cancer problem

1) Pruning results: The pruning methods have been compared extensively on the cancer problem [31]. The pruning procedure depends on two parameters. The first one is the number of training epochs which governs

[2]The benchmark procedure has been implemented on a IBM eServer p690 : A computer having 32 processors Power 4+ 1,7 Ghz and developing a computing power of 220 Giga flops

the start of the pruning process. This is an important element especially for the above standard methods (MAG, OBD, OBS, SKEL, NC) as the NN needs to be ”well-trained”. The second one is the overall stopping criterion of the pruning process. Usually, this stopping criterion is not precisely defined (see [7] or [8] for instance) or vary according the different implementation of the pruning schemes. For our benchmark experiments, this stopping criterion is reached when the error on a validation set deteriorates by more than 10%. So, the behavior of the algorithms was assessed for three training epochs (100, 500 and 1000 epochs) and for two validation sets (by exchanging the original validation and test sets proposed in Proben1). Comparisons for NN having 32 and 128 hidden units were made according to the CPU time, the mean squared error (mse) obtained on the test set and the remaining number of hidden nodes.

Tables III to VI and Fig. 5 to 7 give the results of the benchmark procedure. For convenience, we named the validation set and the test set (provided by Proben1) respectively *cancer1vl.pat* and *cancer1ts.pat*.

Table III : Pruning results obtained from an original NN of 32 hidden units. The validation-set used to stop the pruning procedure is: cancer1vl.pat*. Note that the pruning steps computed for MAG, OBS and OBD are the number of effective pruning steps (see footnote in section V.B)*

Table IV : Pruning results obtained from an original NN of 32 *hidden units. The validation-set used to stop the pruning procedure is:* cancer1ts.pat*.*

We remind that the number of pruning steps computed for MAG, OBS and OBD are the number of effective (node) pruning steps (see footnote in section V.B).

The results of table III and IV are presented in a more synthetic way through the Fig. 5 to 7.

Fig. 5 : Remaining number of hidden units obtained from an original NN of 32 *hidden units. The validation-set used to stop the pruning procedure is:* cancer1vl.pat *(a) or* cancer1ts.pat *(b).*

Fig. 6 : Test mean squared error obtained from an original NN of 32 *hidden units. The validation-set used to stop the pruning procedure is:* cancer1vl.pat *(a) or* cancer1ts.pat *(b).*

Fig. 7 : CPU Time when pruning an original NN of 32 *hidden units. The validation-set used to stop the pruning procedure is:* cancer1vl.pat *(a) or* cancer1ts.pat *(b).*

Table V : Pruning results obtained from an original NN of 128 *hidden units. The validation-set used to stop the pruning procedure is:* cancer1vl.pat

Table VI : Pruning results obtained from an original NN of 128 *hidden units. The validation-set used to stop*

the pruning procedure is: cancer1ts.pat

Fig. 8 to 10 display the results of tables V and VI.

Fig. 8 : Remaining number of hidden units obtained from an original NN of 128 *hidden units. The validation-set used to stop the pruning procedure is:* cancer1vl.pat *(a) or* cancer1ts.pat *(b).*

Fig. 9 : Test mean squared error obtained from an original NN of 128 *hidden units. The validation-set used to stop the pruning procedure is:* cancer1vl.pat *(a) or* cancer1ts.pat *(b).*

Fig. 10 : CPU Time when pruning an original NN of 128 *hidden units. The validation-set used to stop the pruning procedure is:* cancer1vl.pat *(a) or* cancer1ts.pat *(b).*

The following remarks can be made :

1) Under the different pruning conditions (i.e. training epochs and validation set used to stop the procedure), the EFAST pruning method exhibits globally a better mean squared error (apart two exceptions, see Fig. 9a). Furthermore, the mse performance is quite stable whatever the pruning conditions.
2) When using the EFAST approach, the number of hidden units remains practically the same whatever the pruning conditions. It is not the case for the standard pruning methods. Indeed, they show quite a fluctuating performance. Moreover, these algorithms experience difficulties when pruning the NN with 128 hidden units. For instance, methods like MAG, OBS and OBD do not even prune the NN.
3) The EFAST CPU time is of the same order of magnitude or sometimes better than the other pruning methods.

When dealing with the standard methods, the above results show how it is important to correctly answer the question ”when should the pruning process start”. Clearly, the standard methods behave differently under different learning conditions (i.e given here by the number of training epochs). This behavior may question the results obtained with the methods that require specific conditions before pruning occurs. For instance, popular methods such as OBD or OBS require training to the (absolute) error minimum. For the cancer problem, it is supposed that this criterion is reached for 1000 epochs.

But, as also pointed out by [23], this introduces massive overfitting which cannot be repaired by subsequent pruning. This phenomenon is reinforced when pruning the NN with 128 nodes. To prevent this overfitting, one can use a pruning indicator through the monitoring of the error on a validation-set to trigger the pruning session. Nonetheless, starting the pruning process before a minimum is reached on the training set may be questionable for methods like OBS and OBD (since the results of the methods are valid provided an absolute minimum is reached).

In conclusion, the standard methods are highly sensitive to changes in the learning and pruning parameters. Therefore, the parameters of a pruning process for the standard methods must be carefully tuned.

As shown by the previous results, the EFAST pruning method is less or not sensible to these pruning parameters.

Actually, the EFAST algorithm relies only on information obtained during the training phase i.e. the variation of the output of the hidden node Z_h between its minimal and maximal values a_h and b_h. Consequently, the pruning process may occur when the NN has been trained for some epochs. This latter parameter has not to be carefully tuned. Thus, pruning with the EFAST method is possible before a minimum of the training error has been reached.

One another interesting feature of the EFAST algorithm is its stability when pruning NN of different original size of hidden layer. Indeed, whatever the original number of hidden nodes (32 or 128 units), the method leads practically to same number of hidden units.

Last but not least, the CPU time appears to be not a constraint as the EFAST method exhibits in a quantitative way the relevant units in a very few pruning steps.

2) Development of the validation error during the pruning process: Fig. 11 to 13 plot the evolution of the validation error during the benchmark procedure when pruning the original NN of 32 hidden units for 1000 epochs. For the EFAST method, the number of hidden nodes removed at each pruning step is displayed. Notice that (when using the EFAST method), the benchmark procedure always stops at the end of three unsuccessful pruning iterations (since there is no more units to be deleted).

Fig. 11: Evolution of the training and validation error during the benchmark experiment for (a) EFAST

As stated above, unlike the standard algorithms that delete one parameter per pruning step, the EFAST method yields the relevant units in a very few pruning step (practically, two or three pruning steps are necessary) and therefore answers quite satisfactorily to the question ”how much to prune”.

Fig. 11: Evolution of the training and validation error during the benchmark experiment for (b) MAG

Fig. 12: Evolution of the training and validation error during the benchmark experiment for (a) OBS and (b) OBD

Fig. 13: Evolution of the training and validation error during the benchmark experiment for (a) NC end (b) SKEL

3) Evaluation of the EFAST method for different number of simulation runs: Table VII lists illustrates section IV.B and concerns the effect of assigning the same frequency to more than one factor in the complementary set of frequencies. The following results have been obtained when pruning the NN for 1000 epochs.

Table VII : Influence of the number of simulation runs. The NN are trained for 1000 epochs.

As shown by table VII, regarding the number of remaining units, there is no difference when pruning the NN with 32 units. Therefore, assigning the same frequency to two factors has no effect on the pruning results. A difference of two units is observed when pruning the NN with 128 nodes but the better accuracy obtained with

8193 model evaluations is counterbalanced by the higher computational cost. Moreover, a difference of two units have practically no influence on the generalization performance of the NN.

B. Architecture selection by increasing the number of hidden nodes

Experiments with the standard technique of selecting the number of hidden units using a validation set (provided by Proben1) were performed. The selection of the model was based on the performance measure estimated by the mean squared error (mse) on the validation set (hold-out method)[3]. In order to obtain a better estimation of this measure, the NN were trained five times using different initializations and the mean of the mse was used as estimator.

Again, we evaluate the technique for two validation data sets (by exchanging the validation and test sets proposed by Proben1). The following procedure was used:

1) Start with a NN with one hidden unit
2) Train 5 times the NN for 1000 epochs using different initializations
3) Compute the mean of the mse on a validation set
   If the validation error deteriorates by more than 10% from the previous iteration, go to step 5 otherwise go to step 4
4) Increase (using a non linear scale : 1, 2, 4, 8, 16 , ...) the number of hidden units and proceed to step 2
5) end of procedure

Fig. 14 and table VIII show the results :

When the validation set is *cancer1vl.pat*, the procedure stops when the NN has a layer of 32 hidden units whereas with *cancer1ts.pat*, it stops when there are 2 hidden nodes. It can be seen (table VIII and Fig. 17) that this additive procedure nor is faster nor is better than the EFAST method (see table III) . For selecting the number of hidden nodes, such experiment seems to be useless as it uses a noisy performance measure (i.e. the validation error) and is validation-set dependent.

Furthermore, it has been shown that cross-validation scores are biaseds and do not lead to the optimal model [32].

Table VIII : Results for the growing phase

Fig. 14: Results obtained for the additive (or growing phase)

C. Other experimental results

We also evaluate the performance of the pruning methods over the range of significant datasets provided by Proben1 [17] . The experiments deal with the pruning of two original NN (128 and 32 nodes). For these problems,

[3]If there is not enough data and the whole data is used for training, one could perform a ten-fold or leave-one-out (computationally demanding) cross-validation experiments

we have not shown the results obtained with the additive phase as the same conclusion drawn in the previous section remains unchanged. Fig. 15 to 24 show the results:

Fig. 15: Card (mse, hidden units and cpu time)

Fig. 16: Diabetes (mse, hidden units and cpu time)

Fig. 17: Horse (mse, hidden units and cpu time)

Fig. 18: Thyroid (mse, hidden units and cpu time)

Fig. 19: Building1 (mse, hidden units and cpu time)

Fig. 20: Building2 (mse, hidden units and cpu time)

Fig. 21: Flare1 (mse, hidden units and cpu time)

Fig. 22: Flare2 (mse, hidden units and cpu time)

Fig. 23: Heart (mse, hidden units and cpu time)

Fig. 24: EES (mse, hidden units and cpu time)

Notice for the Card, Horse, Thyroid and EES problems, OBS failed (and exited with an error message of insufficient memory) when pruning the original NN with 128 nodes.

The results confirm that the EFAST method outperforms the other pruning algorithms when focusing on the couple mean squared error and number of remaining units i.e. the NN obtained with the EFAST algorithm are more parsimonious while yielding a test mse which is of the same order of magnitude. Indeed, even if in some cases, the EFAST mse is not the best, it is close to the best. Considering the remaining number of hidden units, the EFAST method always lands practically on the same number whatever the original NN. Again, the CPU time appears to be very affordable.

VII. Conclusion

In this paper, we have proposed a new method to prune hidden units of oversized neural networks. The procedure is based on the EFAST method, a quantitative model-independent method for global sensitivity of model output. The method delivers quantitative information about the relative importance of the hidden units.

The new pruning algorithm offers several advantages:

1) It is a robust, stable and consistent method that exhibits good performance whatever the original structure.
2) The method exhibits in a quantitative way the relevant units and therefore answers quite satisfactorily to the question ”how much to prune”.
3) The method does not necessitate a fine-tuning of the learning parameters.
4) Consequently, as convergence to a minimum of the criterion is not a prerequisite, it is possible to prune before the network is at the minimum of the cost function.
5) The results obtained with the EFAST method is only dependent on the training phase. This feature is very appealing when dealing with finite dataset. So, in practice, additional data such as a validation set is useless. In other words, the method is able to deal with the problem of model complexity without the need of cross-validation or the need to optimally tune a specific parameter during the pruning process.
6) Moreover, the CPU time is not a constraint as the method prunes several units per pruning step.

Experiments that consist in selecting the number of hidden nodes using a validation set seem to be inappropriate as the performance measure is highly biased and validation set dependent. This standard technique is also not faster nor as efficient as the proposed one.

Finally, on the basis of the results, we feel that the EFAST algorithm provide a useful and efficient method to prune hidden nodes of relatively large NN and we propose the following EFAST pruning recipe:

1) Train a NN that is larger than necessary
2) Apply the EFAST pruning algorithm for two or three steps.
3) Train the NN with the number of hidden nodes identified with the EFAST method
4) Test the NN

Application of the EFAST pruning method to NN having more than one layer of hidden nodes will be straightforward. Future work will aim to apply this new technique to pruning of inputs of the NN. It would be also interesting to examine the behavior of the method on recurrent networks or Elman networks where possibly some outputs of hidden units are fed back as inputs to the network. Would this method exhibit the effect of interactions?

REFERENCES

[1] C. M. Bishop, "Regularization and complexity control in feed-forward networks," Neural Computing Research Group, Aston University, Birmingham, UK, Tech. Rep. NCRG 95/022, 1995. [Online]. Available: http://neural-server.aston.ac.uk/

[2] J. Sjoberg and L. Ljung, "Overtraining, regularization, and searching for minimum in neural networks," in *Preprint* 4^{th} *IFAC Symposium on Adaptive Systems in Control and Signal Processing, Grenoble, France*, 1996, pp. 669–674.

[3] R. Reed, "Pruning algorithms - a survey," *IEEE Transactions on Neural Networks*, vol. 4, no. 5, pp. 740–747, 1993.

[4] C. M. Bishop, *Neural Networks for Pattern Recognition*. Oxford University Press, 1995.

[5] D. J. C. MacKay, "Bayesian interpolation," *Neural Computation*, vol. 4, no. 3, pp. 415–447, 1992.

[6] ——, "A practical bayesian framework for back-propagation networks," *Neural Computation*, vol. 4, no. 3, pp. 448–472, 1992.

[7] Y. Le Cun, J. S. Denker, and S. A. Solla, "Optimal brain damage," *Advances in Neural Information Processing systems*, vol. 2, pp. 598–605, 1990.

[8] B. Hassibi and D. G. Stork, "Second order derivatives for network pruning: Optimal brain surgeon," in *Advances in Neural Information Processing systems*, C. Lee, S. Hanson, and J. Cowan, Eds. Morgan Kaufmann, San Mateo, CA, 1993, vol. 5, pp. 164–171.

[9] T. Cibas, F. Fogelman Soulié, P. Gallinari, and S. Raudys, "Variable selection with neural networks," *Neurocomputing*, vol. 12, pp. 223–248, 1996.

[10] A. P. Engelbretch, "A new pruning heuristic based on variance analysis of sensitivity information," *IEEE Transactions on Neural Networks*, vol. 12, no. 6, pp. 1386–1399, 2001.

[11] M. E. Ricotti and E. Zio, "Neural network approach to sensitivity and uncertainty analysis," *Reliability Engineering and System Safety*, vol. 64, pp. 59–71, 1999.

[12] K. Hornik, M. Stinchcombe, and H. White, "Universal approximation of an unknown mapping and its derivatives using multilayer feedforward networks," *Neural Networks*, vol. 3, no. 5, pp. 551–560, 1990.

[13] J. M. Zurada, A. Malinowski, and S. Usui, "Perturbation method for deleting redundant inputs of perceptron networks," *Neurocomputing*, vol. 14, pp. 177–193, 1997.

[14] A. Saltelli, K.-S. Chan, and E. M. Scott, *Sensitivity Analysis*. New York: Wiley, 2000.

[15] A. Saltelli, S. Tarantola, and K.-S. Chan, "A quantitative model-independant method for global sensitivity analysis of model output," *Technometrics*, vol. 41, no. 1, pp. 39–56, 1999.

[16] R. I. Cukier, C. M. Fortuin, K. E. Shuler, A. G. Petscheck, and J. H. Schaibly, "Study of the sensitivity of coupled reaction systems to uncertainties in rate coefficients, part i theory," *Journal of Chemical Physics*, vol. 59, pp. 3873–3878, 1900.

[17] L. Prechelt, "Proben1 - a set of neural networks benchmark problems and benchmarking rules," Univ. Karlsruher, Tech. Rep. 21/94, 1994.

[18] J. C. Helton, "Uncertainty and sensitivity analysis techniques for use in performance assessment for radioactive waste disposal," *Reliabily Engineering and System Safety*, vol. 42, pp. 327–367, 1993.

[19] F. Campolongo and A. Saltelli, *Comparing Different Sensitivity Analysis Methods on a Chemical Reactions Model*. New York: Wiley, 2000, ch. 18 *in Sensitivity Analysis*, pp. 335–364.

[20] J. M. Zaldivar and F. Campologo, *An Application of Sensibility Analysis to Fish Population Dynamics*. New York: Wiley, 2000, ch. 19 *in Sensitivity Analysis*, pp. 367–382.

[21] F. Campolongo and A. Saltelli, "Sensitivity analysis of an environmental model: an application of different analysis method," *Reliabily Engineering and System Safety*, vol. 57, pp. 49–69, 1997.

[22] I. M. Sobol', "Sensitivity analysis for non linear mathematical models," *Math. Model. Comput. Exp.*, vol. 1, pp. 407–414, 1993.

[23] L. Prechelt, "Connection pruning with static and adaptive pruning schedules," *Neurocomputing*, vol. 16, pp. 49–61, 1997.

[24] C. Riviere, P. Lauret, Y. Page, T. Mara, E. Fock, J.-C. Gatina, and J. LeCoz, "Modelling the energy equivalent speed with an artificial neural network," in *FISITA 2004*, Barcelona, Spain, 2004.

[25] *SNNS: Stuttgart Neural Networks Simulator*. [Online]. Available: http://www-ra.informatik.uni-tuebingen.de/SNNS/

[26] M. Hagiwara, "Removal of hidden units and weights for backpropagation networks," in *International Joint Conference on Neural Networks*. IEEE, 1993, pp. 351–354.

[27] M. Mozer and P. Smolensky, "Skeletonization: a technique for trimming the fat from network via relevance assessment," in *Advances in Neural Information Processing systems*, D. Touretzky, Ed. Morgan Kaufmann, San Mateo, CA, 1991, vol. 1, pp. 107–115.

[28] J. Sietsma and R. Dow, "Creating artificial neural networks that generalize," *Neural Networks*, vol. 4, no. 1, pp. 67–79, 1991.

[29] M. Riedmiller and H. Braun, “A direct adaptive method for faster backpropagation learning: the rprop algorithm,” in *Proceedings of the IEEE International Conference on Neural Networks*, 1993.

[30] I. Rivals and L. Personnaz, “Neural network construction and selection in nonlinear modeling,” *IEEE Transactions on Neural Networks*, vol. 14, no. 4, pp. 804–819, 2003.

[31] W. H. Wolberg and O. L. Mangasarian, “Multisurface method of pattern separation for medical diagnosis applied to breast cytology,” in *Proceedings of the National Academy of Sciences*, vol. 87, 1990, pp. 9193–9196.

[32] I. Rivals and L. Personnaz, “On cross-validation for model selection,” *Neural Computation*, vol. 11, pp. 863–870, 1998.

LIST OF TABLES

TABLE I

FREQUENCIES ASSIGNED TO THE INPUT FACTORS GIVEN THE NUMBER OF FACTORS AND NUMBER OF SIMULATION RUNS (THE THIRD FACTOR IS THE FACTOR OF INTEREST)

Number of hidden units	N	ω_h	$max(\omega_{\sim h})$	**Set of frequencies assigned to the factors**
32	1025	*128*	16	[1,2,**128**,4,5,6,7,8,9,10,11,12,13,14,15,16,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16]
32	2049	*256*	32	[1,2,**256**,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32]
128	1025	*128*	16	[1,2,**128**,4,5,6,7,8,9,10,11,12,13,14,15,16,1, 2,...,15,16,1,2, ...,15,16]
128	8193	*1024*	128	[1,2,**1024**,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,33,...,127,128]

TABLE II

THE DATASETS, WHERE THE TYPE IS EITHER C (CLASSIFICATION) OR A (APPROXIMATION)

Dataset	**Description**	**Type**	**Inputs**	**Outputs**	**Number of examples**			
					training	*validation*	*test*	*total*
Cancer	Diagnosis of breast cancer	c	9	2	350	175	174	699
Card	Predict the approval or non-approval of credit card to a customer	c	51	2	345	173	172	690
Diabetes	Diagnosis of diabetes	c	8	2	384	192	192	768
Horse	Predict the fate the horse has a colic	c	58	3	182	91	91	364
Thyroid	Diagnose thyroid hyper or hypofunction	c	21	3	3600	1800	1800	7200
Building	Prediction of energy consumption in a building	a	14	3	2104	1052	1052	4208
Flare	Prediction of solar flares	a	24	3	533	267	266	1066
Heart	Predict heart disease	a	35	1	152	76	75	303
EES	Predict deformation energy in a frontal car crash	a	90	1	1150	100	500	1750

TABLE III

PRUNING RESULTS OBTAINED FROM AN ORIGINAL NN OF *32* HIDDEN UNITS. THE VALIDATION-SET USED TO STOP THE PRUNING PROCEDURE IS: *cancer1vl.pat*. NOTE THAT THE PRUNING STEPS COMPUTED FOR MAG, OBS AND OBD ARE THE NUMBER OF EFFECTIVE PRUNING STEPS (SEE FOOTNOTE IN SECTION V.B)

100 epochs	**MAG**	**OBS**	**OBD**	**NC**	**SKEL**	**EFAST**
Remaining units	23	7	3	1	1	10
mse	0,019772	0,052966	0,022105	0,015224	0,02005	0,016157
CPU Time (s)	0,25	2,79	0,64	8,39	1,26	1,42
Pruning Steps	6	18	15	31	31	7
500 epochs	**MAG**	**OBS**	**OBD**	**NC**	**SKEL**	**EFAST**
Remaining units	20	11	8	8	13	7
mse	0,033348	0,036446	0,03063	0,018723	0,012128	0,014704
CPU Time (s)	1,53	4,62	2,51	12,53	5,18	3,18
Pruning Steps	7	15	13	24	19	5
1000 epochs	**MAG**	**OBS**	**OBD**	**NC**	**SKEL**	**EFAST**
Remaining units	13	13	12	7	23	10
mse	0,030137	0,039323	0,030228	0,037333	0,033272	0,017552
CPU Time (s)	4,85	6,94	4,32	18,21	7,75	5,8
Pruning Steps	14	16	11	25	9	4

TABLE IV

PRUNING RESULTS OBTAINED FROM AN ORIGINAL NN OF *32* HIDDEN UNITS. THE VALIDATION-SET USED TO STOP THE PRUNING PROCEDURE IS: *cancer1ts.pat*.

100 epochs	**MAG**	**OBS**	**OBD**	**NC**	**SKEL**	**EFAST**
Remaining units	20	19	3	2	1	10
mse	0,032293	0,029819	0,024007	0,019215	0,025124	0,019227
CPU time (s)	0,32	1,73	0,65	8,37	1,25	1,41
Pruning steps	9	9	15	30	31	7
500 epochs	**MAG**	**OBS**	**OBD**	**NC**	**SKEL**	**EFAST**
Remaining units	30	30	32	12	12	7
mse	0,030459	0,030226	0,022761	0,031213	0,027922	0,017189
CPU time (s)	0,34	0,7	9,12	11,82	5,28	3,18
Pruning steps	1	1	0	20	20	5
1000 epochs	**MAG**	**OBS**	**OBD**	**NC**	**SKEL**	**EFAST**
Remaining units	32	32	32	25	26	10
mse	0,030478	0,03046	0,021553	0,028668	0,030621	0,021684
CPU time (s)	6,99	9,28	9,22	10,45	6,73	5,8
Pruning steps	0	0	0	7	6	4

TABLE V

PRUNING RESULTS OBTAINED FROM AN ORIGINAL NN OF *128* HIDDEN UNITS. THE VALIDATION-SET USED TO STOP THE PRUNING PROCEDURE IS: *cancer1vl.pat*

100 epochs	**MAG**	**OBS**	**OBD**	**NC**	**SKEL**	**EFAST**
Remaining units	110	110	128	22	3	9
mse	0,014326	0,017414	0,020443	0,027227	0,020272	0,014135
CPU Time (s)	1,6	135,21	39,6	351,54	14,61	6,83
Pruning Steps	9	9	0	106	125	5
500 epochs	**MAG**	**OBS**	**OBD**	**NC**	**SKEL**	**EFAST**
Remaining units	126	126	128	104	107	6
mse	0,014416	0,018459	0,036799	0,03136	0,03406	0,031294
CPU Time (s)	1,41	24,49	39,68	180,31	25,76	13,68
Pruning Steps	2	1	0	24	21	7
1000 epochs	**MAG**	**OBS**	**OBD**	**NC**	**SKEL**	**EFAST**
Remaining units	128	128	128	125	125	7
mse	0,020574	0,020719	0,017379	0,020584	0,020731	0,02671
CPU Time (s)	22,42	90,26	29,46	44,78	22,42	22,28
Pruning Steps	0	0	0	3	3	3

TABLE VI

PRUNING RESULTS OBTAINED FROM AN ORIGINAL NN OF *128* HIDDEN UNITS. THE VALIDATION-SET USED TO STOP THE PRUNING PROCEDURE IS: *cancer1ts.pat*

100 epochs	**MAG**	**OBS**	**OBD**	**NC**	**SKEL**	**EFAST**
Remaining units	110	110	128	63	18	9
mse	0,033361	0,024712	0,021125	0,02166	0,017033	0,017802
CPU Time (s)	1,61	149,86	33,99	305,85	14,34	6,83
Pruning steps	9	14	0	65	110	5
500 epochs	**MAG**	**OBS**	**OBD**	**NC**	**SKEL**	**EFAST**
Remaining units	126	126	128	104	107	8
mse	0,021172	0,025816	0,033185	0,033621	0,037907	0,017743
CPU Time (s)	1,42	25,16	39,74	180,19	25,76	13,29
Pruning steps	1	1	0	24	21	3
1000 epochs	**MAG**	**OBS**	**OBD**	**NC**	**SKEL**	**EFAST**
Remaining units	127	127	128	120	120	7
mse	0,037512	0,037809	0,018697	0,03437	0,033327	0,021557
CPU Time (s)	1,82	11,71	29,49	88,14	30,76	22,24
Pruning steps	1	1	0	8	8	3

TABLE VII

INFLUENCE OF THE NUMBER OF SIMULATION RUNS. THE NN ARE TRAINED FOR 1000 EPOCHS

Original NN	**32**	**32**	**128**	**128**
# model evaluations	2049	1025	8193	1025
Remaining units	10	10	9	7
Test mse	0,017646	0,017552	0,026478	0,02671
CPU Time (s)	7,43	5,77	97,07	22,23

TABLE VIII

RESULTS FOR THE GROWING PHASE

Validation Set	**cancer1vl.pat**	**cancer1ts.pat**
# of hidden units	32	2
CPU Time (s)	91,28	7,66
Test mse	0,028859	0,028384

LIST OF FIGURES

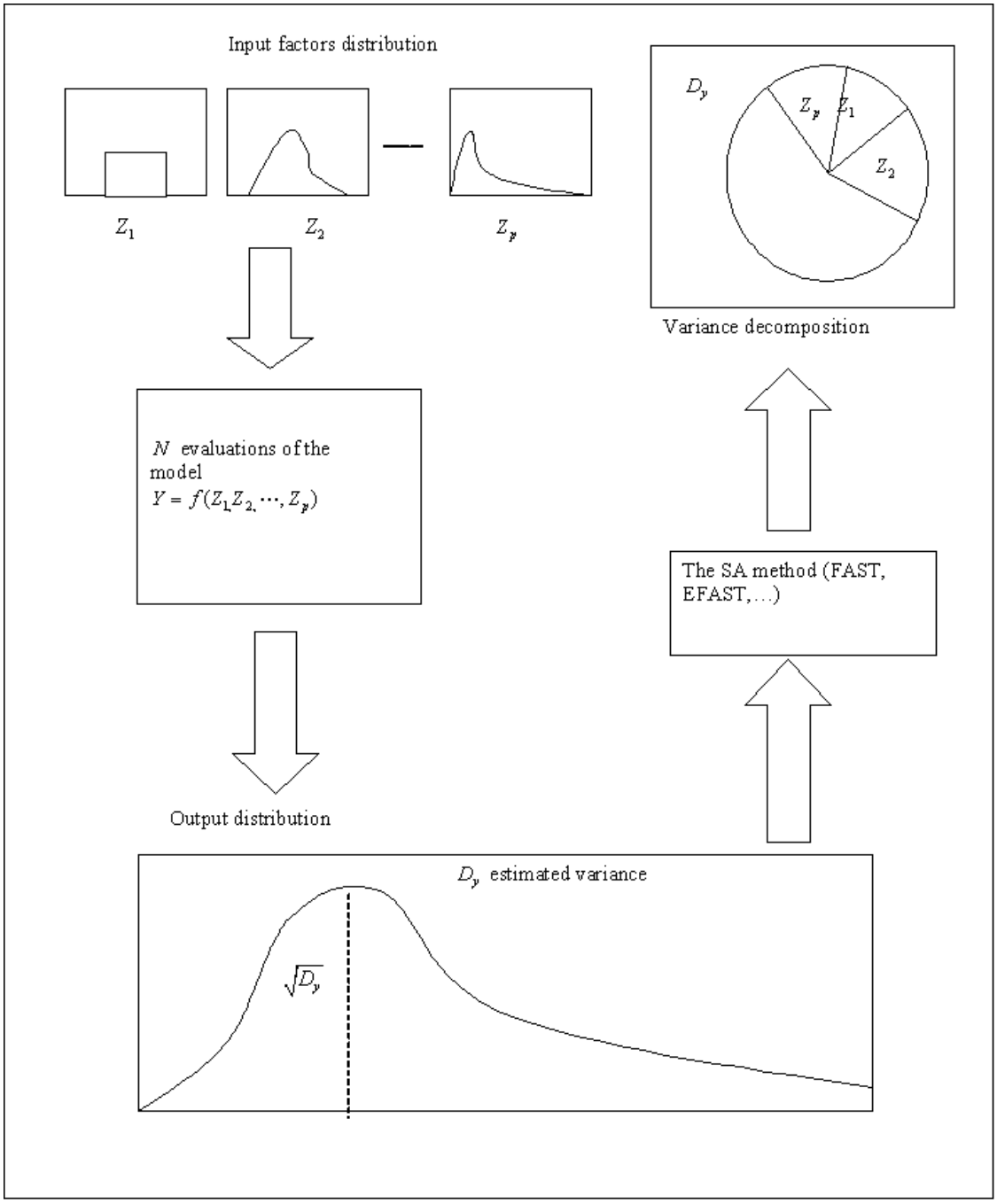
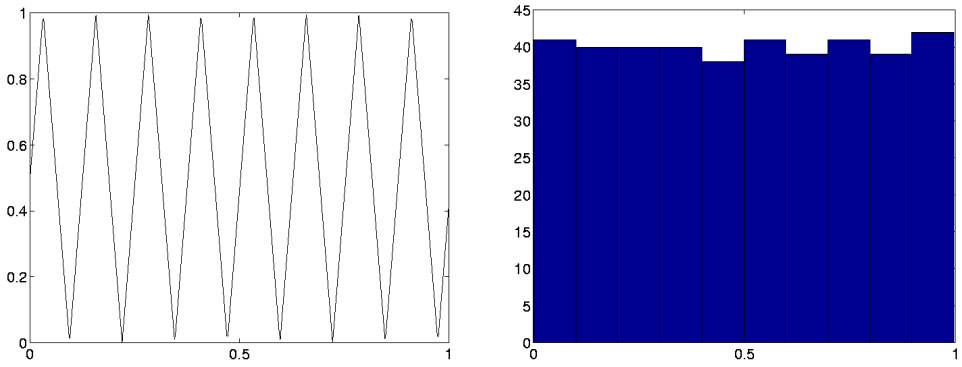


Fig. 1. General scheme of a quantitative Sensitivity Analysis method. The total variance is apportioned to the various input factors, as shown by the pie diagram.



(a) (b)

Fig. 2. (a) Plot of the transformations function (defined by (7)) and (b) its respective empirical distribution

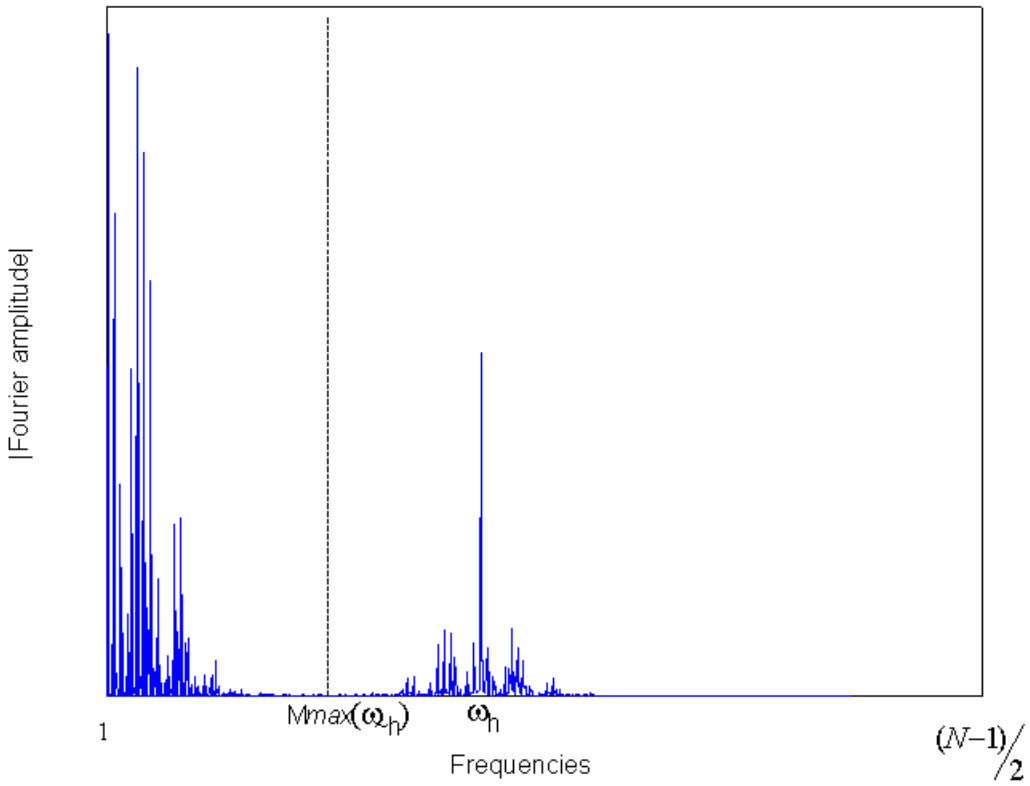


Fig. 3. The spectrum of a model response using the EFAST approach . The spectrum is divided into two regions: the first region $\left[1, \frac{\omega_h}{2} = Mmax(\omega_{\sim h})\right]$ contains the frequencies involving all the factors except those of factor Z_h and the second region $[Mmax(\omega_{\sim h}) + 1, (N-1)/2]$ contains the effects of factor Z_h located in the high frequencies.

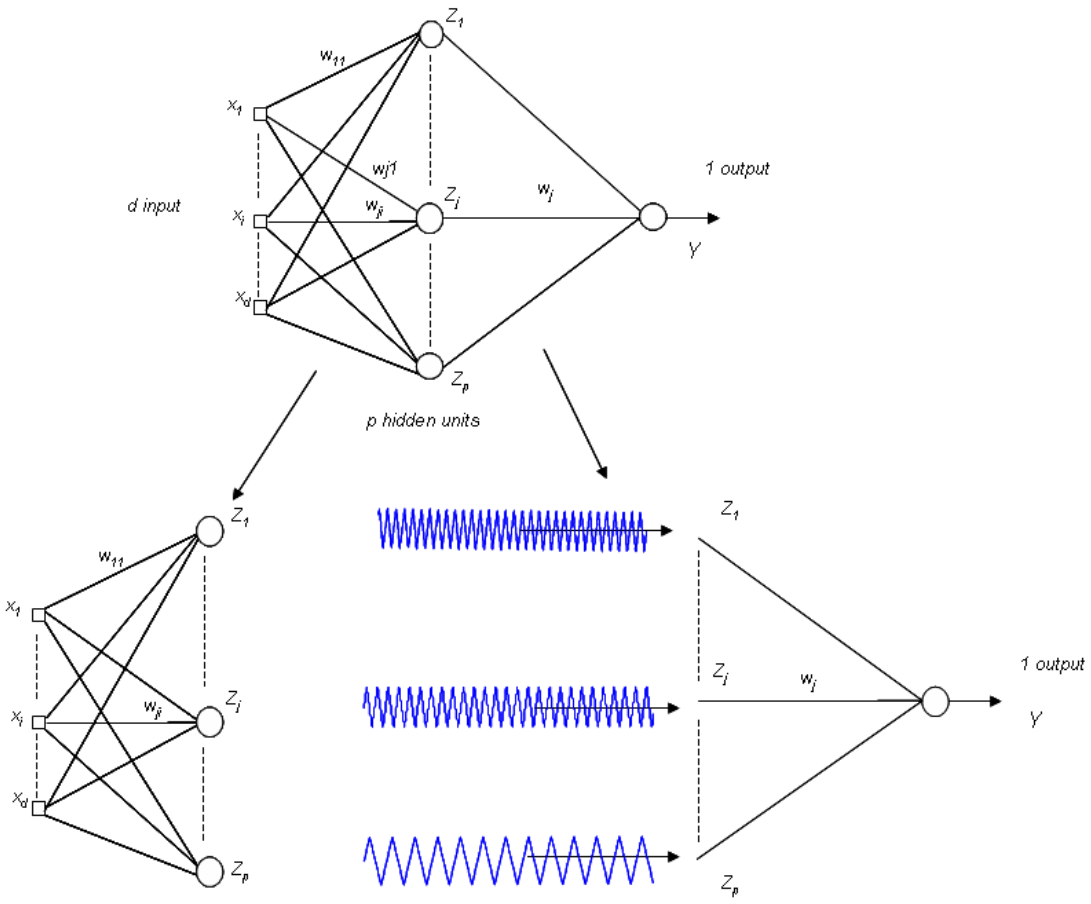
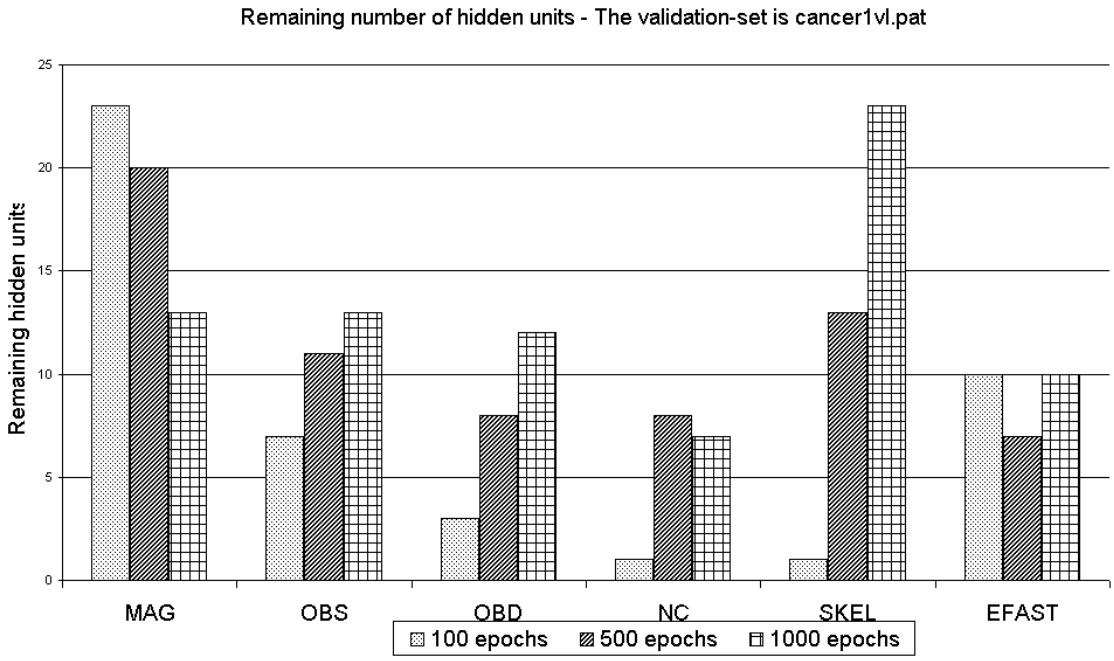
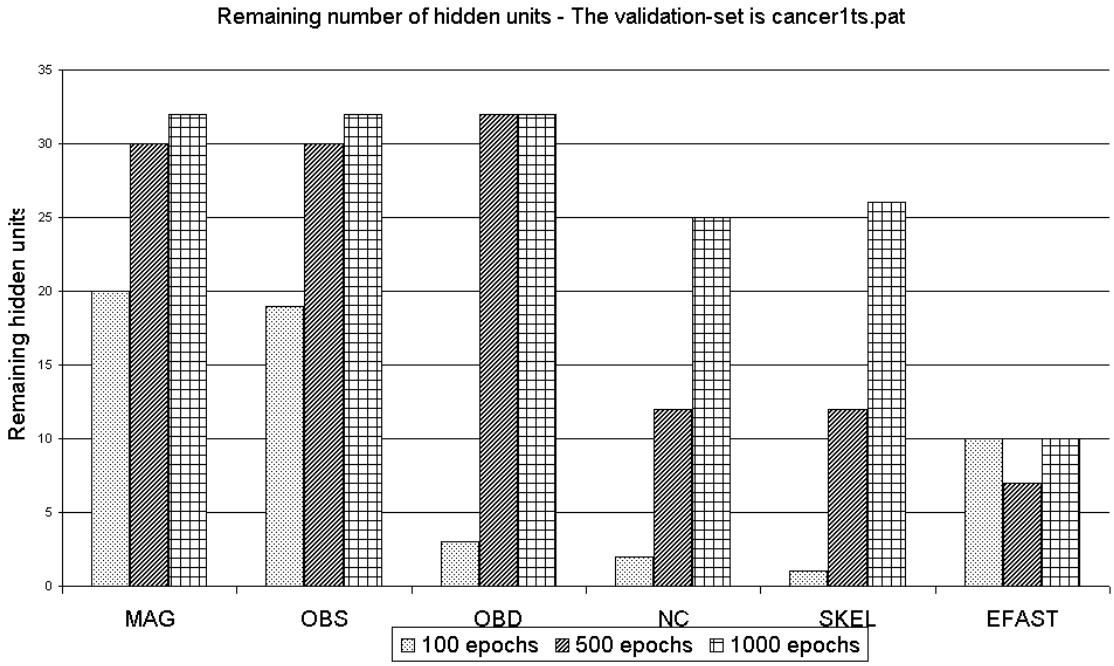


Fig. 4. The EFAST method applied to pruning of hidden units. Each output of the hidden units constitutes an input factor. All input factors oscillates (each with its own frequency ω_i) according to the curve defined by (8). N samples of the output are evaluated that enable the computation of the percentage contribution of each hidden unit (through the Fourier decomposition of the variance of the output).

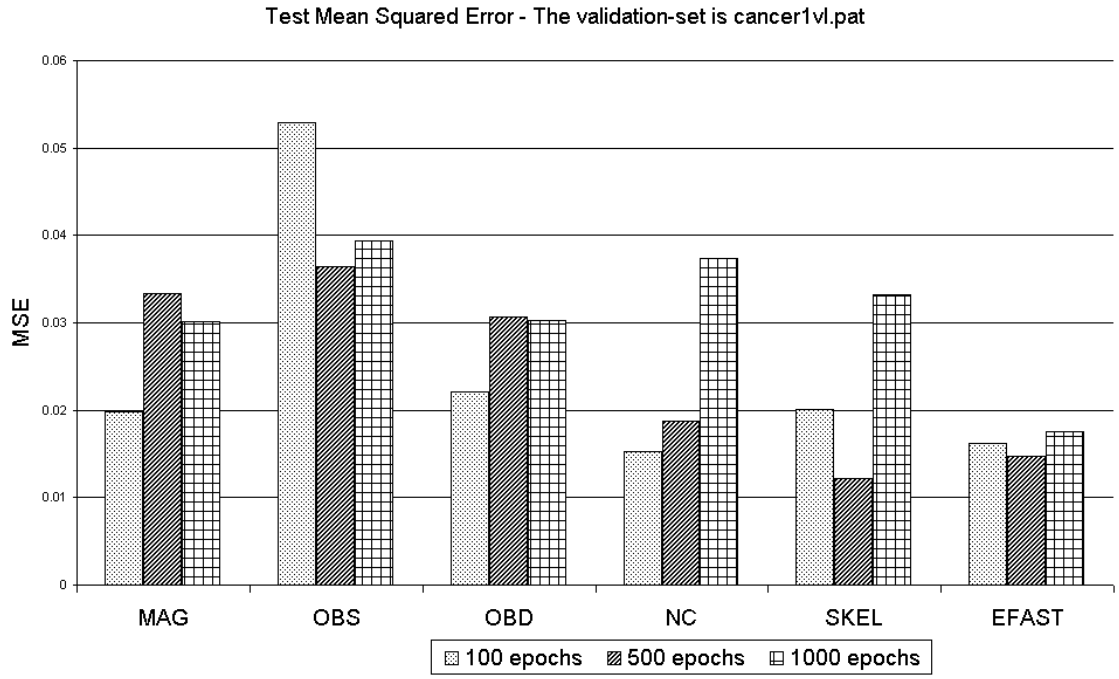


(a)

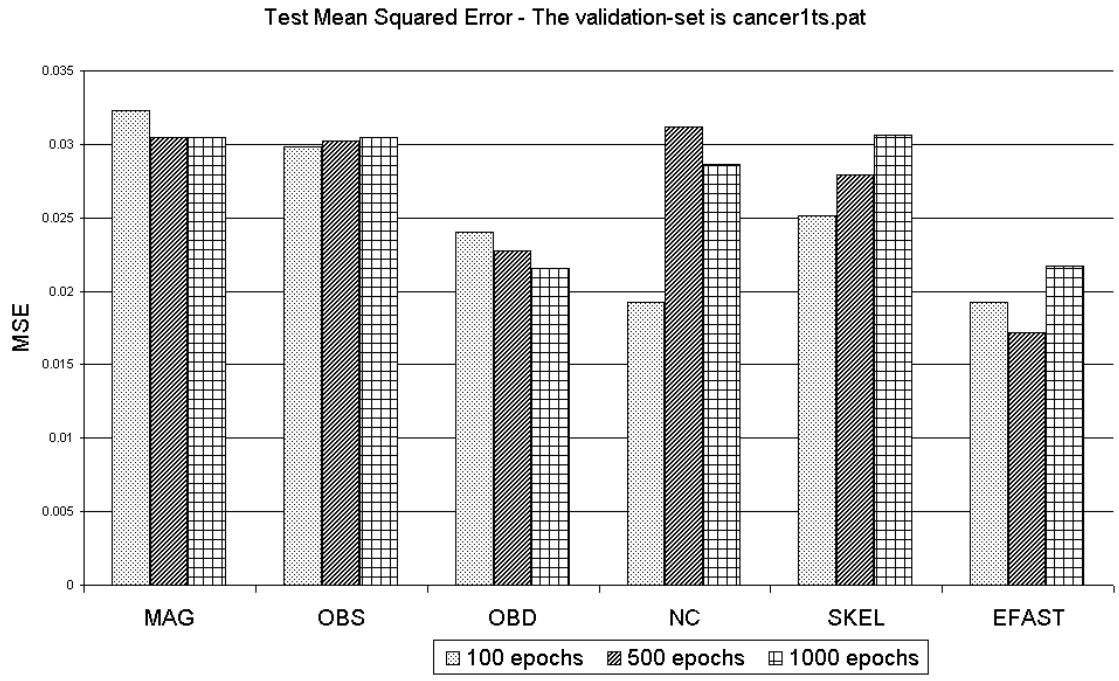


(b)

Fig. 5. Remaining number of hidden units obtained from an original NN of *32* hidden units. The validation-set used to stop the pruning procedure is: *cancer1vl.pat* (a) or *cancer1ts.pat* (b).

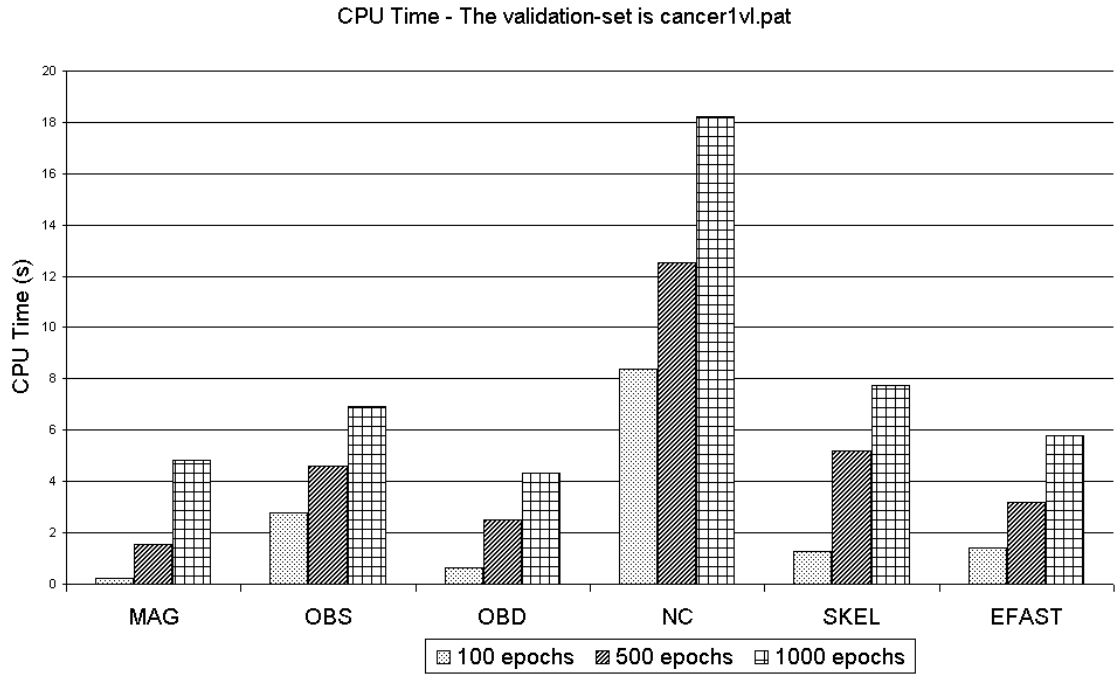


(a)

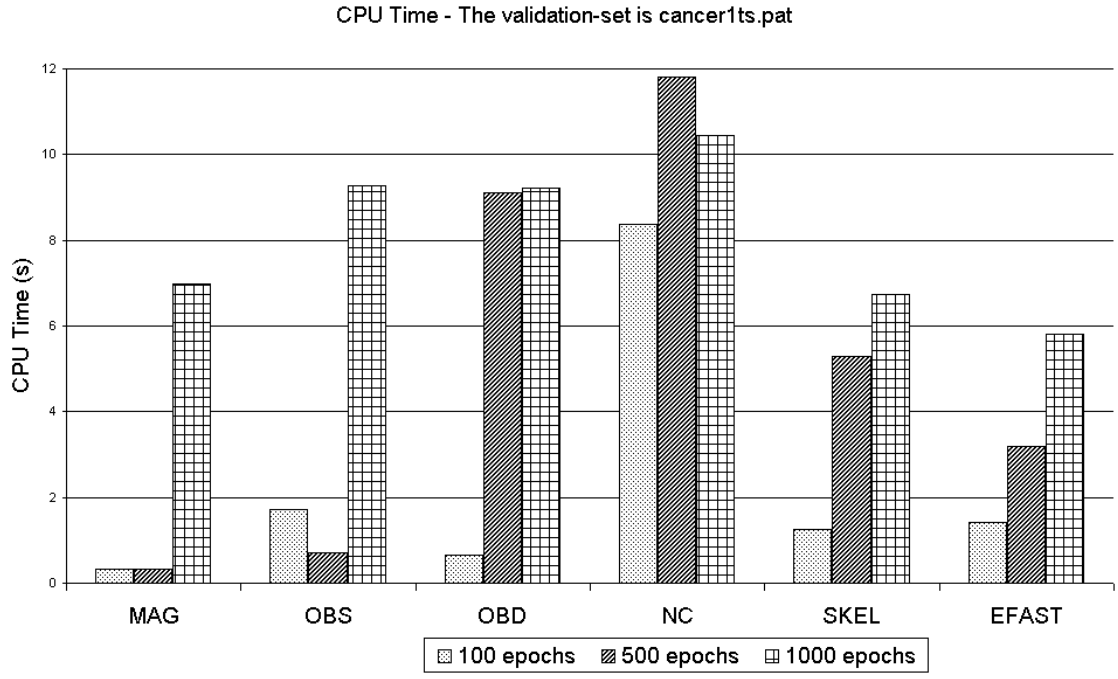


(b)

Fig. 6. Test mean squared error obtained from an original NN of *32* hidden units. The validation-set used to stop the pruning procedure is: *cancer1vl.pat* (a) or *cancer1ts.pat* (b).

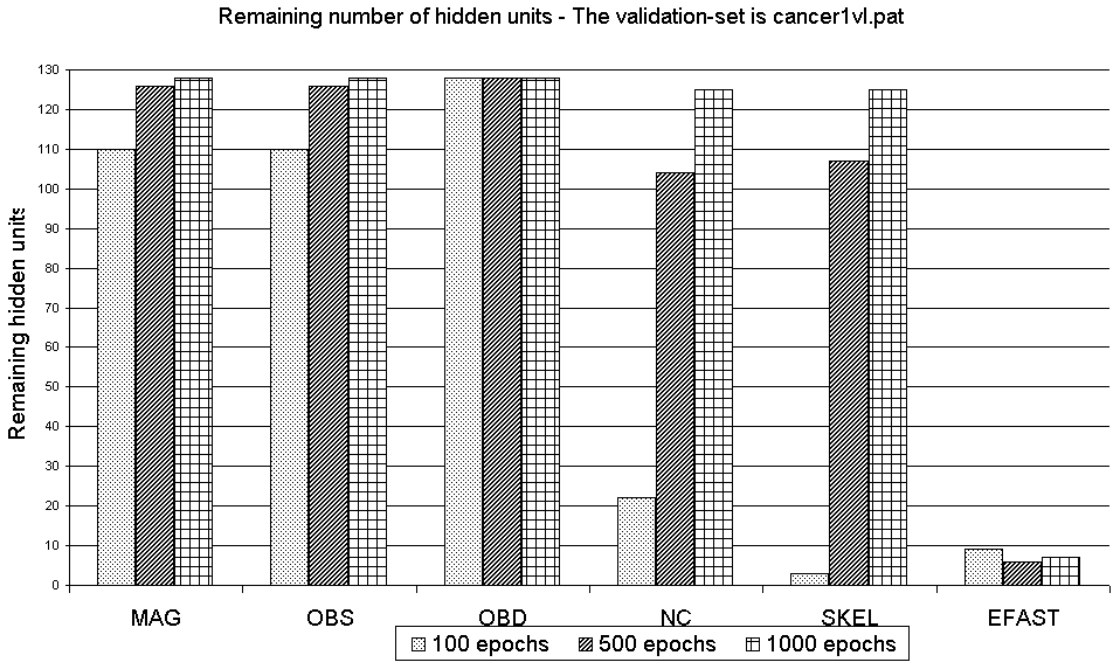


(a)

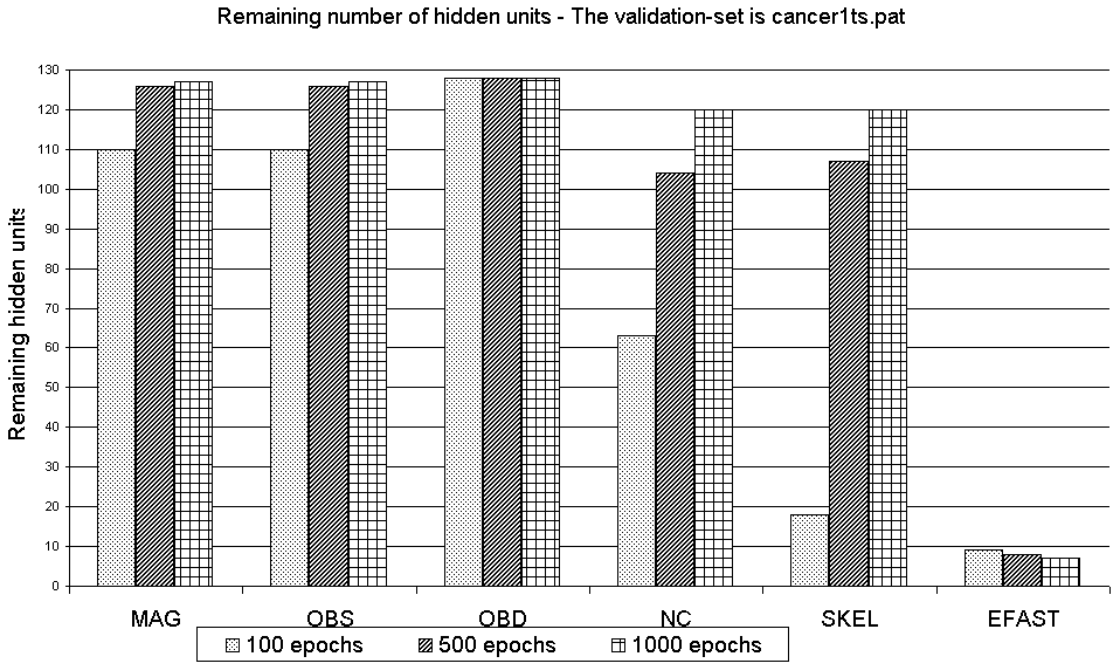


(b)

Fig. 7. CPU Time when pruning an original NN of *32* hidden units. The validation-set used to stop the pruning procedure is: *cancer1vl.pat* (a) or *cancer1ts.pat* (b).

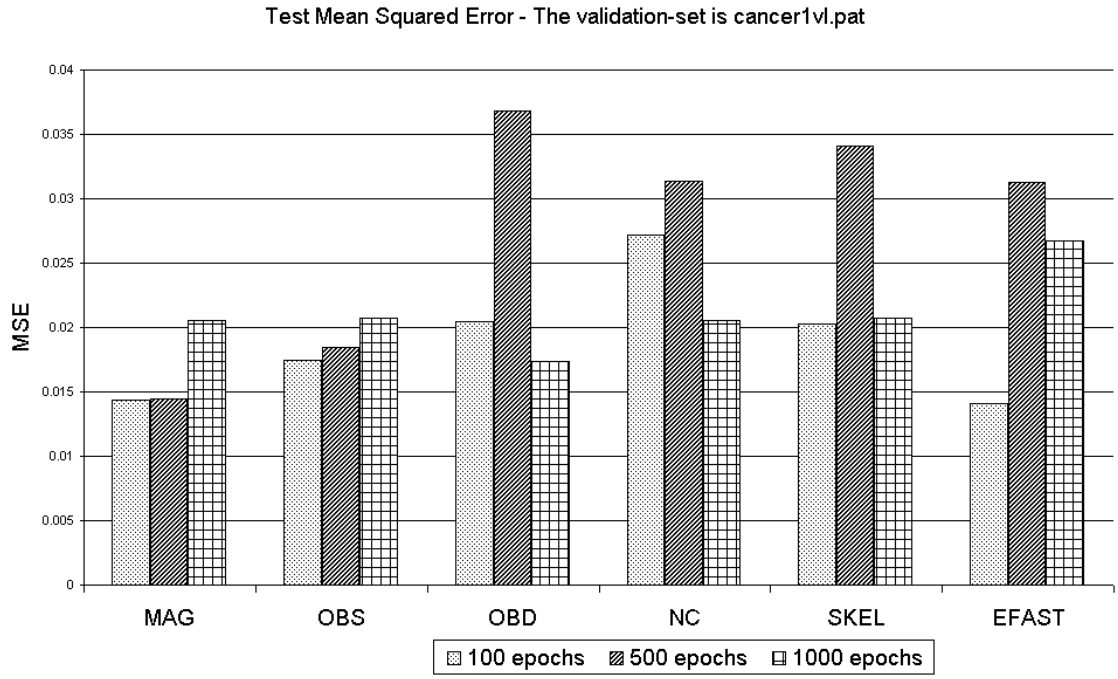


(a)

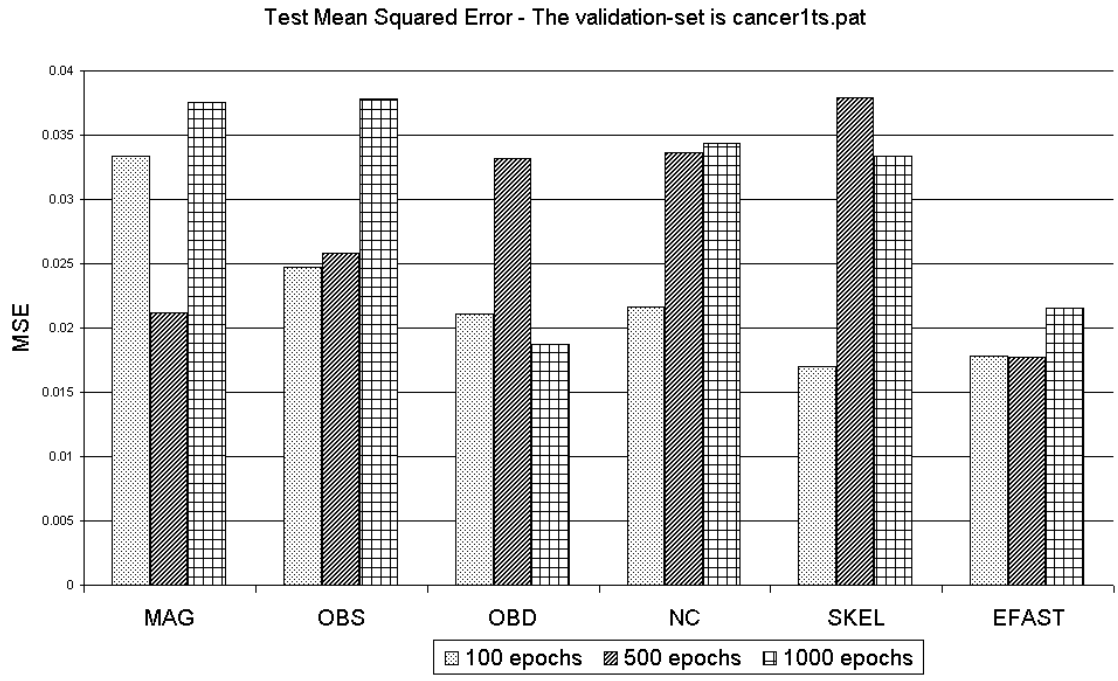


(b)

Fig. 8. Remaining number of hidden units obtained from an original NN of *128* hidden units. The validation-set used to stop the pruning procedure is: *cancer1vl.pat* (a) or *cancer1ts.pat* (b).

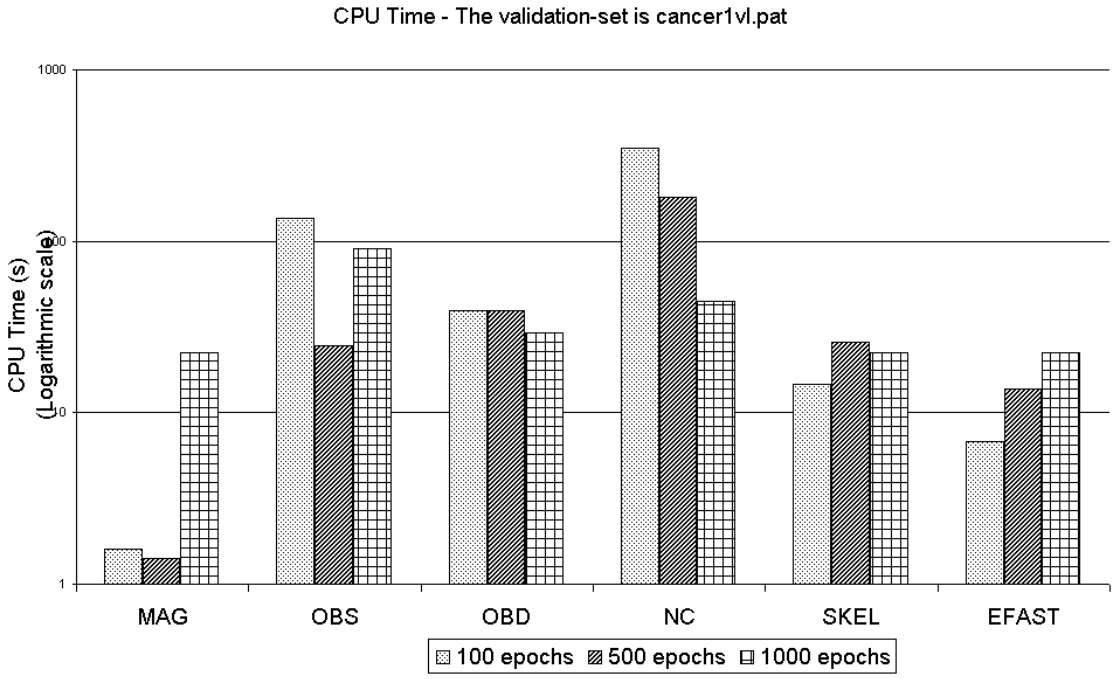


(a)

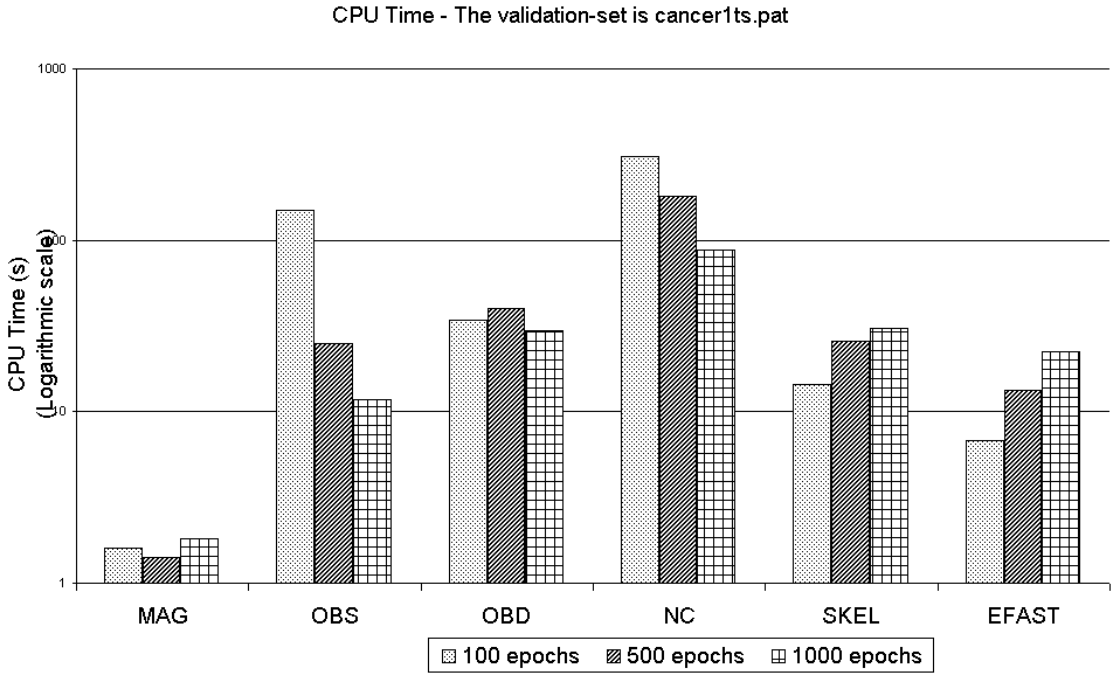


(b)

Fig. 9. Test mean squared error obtained from an original NN of *128* hidden units. The validation-set used to stop the pruning procedure is: *cancer1vl.pat* (a) or *cancer1ts.pat* (b).



(a)



(b)

Fig. 10. CPU Time when pruning an original NN of *128* hidden units. The validation-set used to stop the pruning procedure is: *cancer1vl.pat* (a) or *cancer1ts.pat* (b).

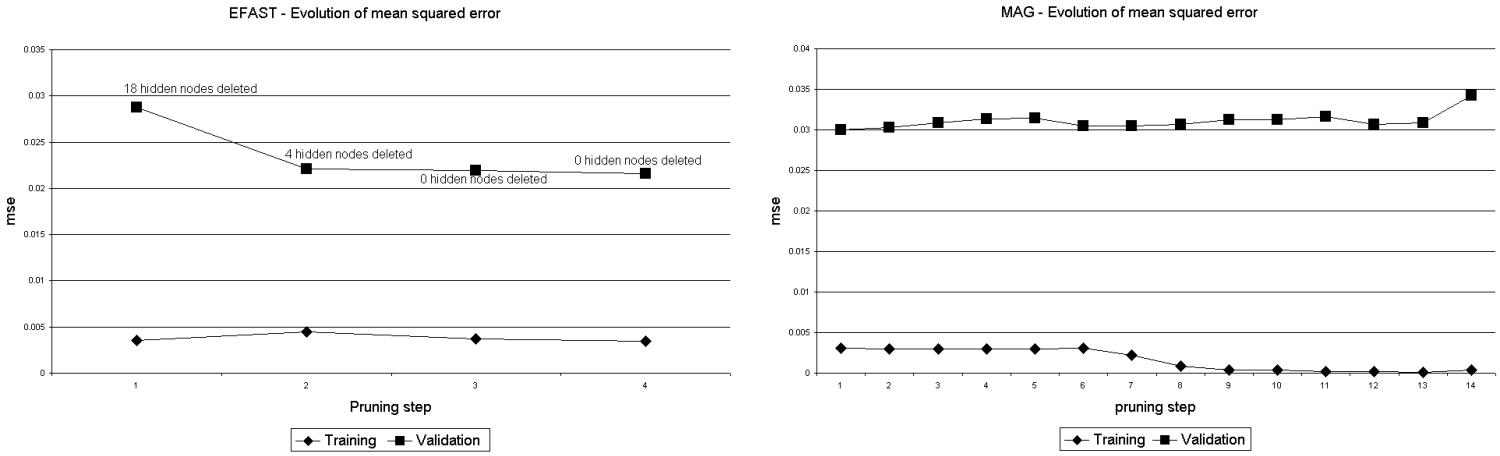


Fig. 11. Evolution of the training and validation error during the benchmark experiment for (a) EFAST and (b) MAG

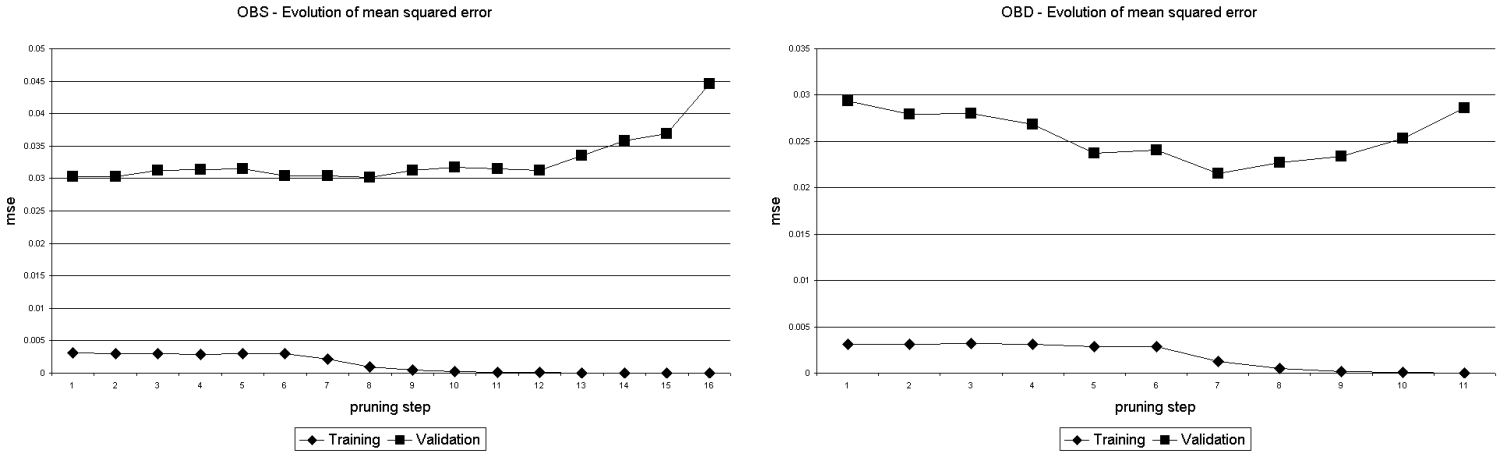


Fig. 12. Evolution of the training and validation error during the benchmark experiment for (a) OBS and (b) OBD

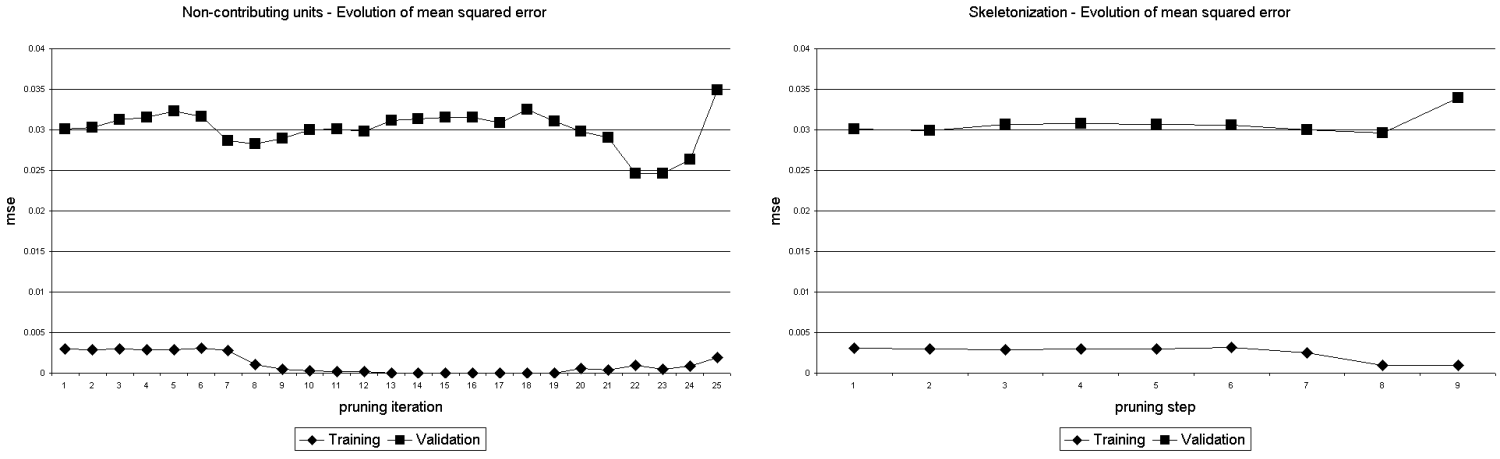


Fig. 13. Evolution of the training and validation error during the benchmark experiment for (a) NC and (b) SKEL

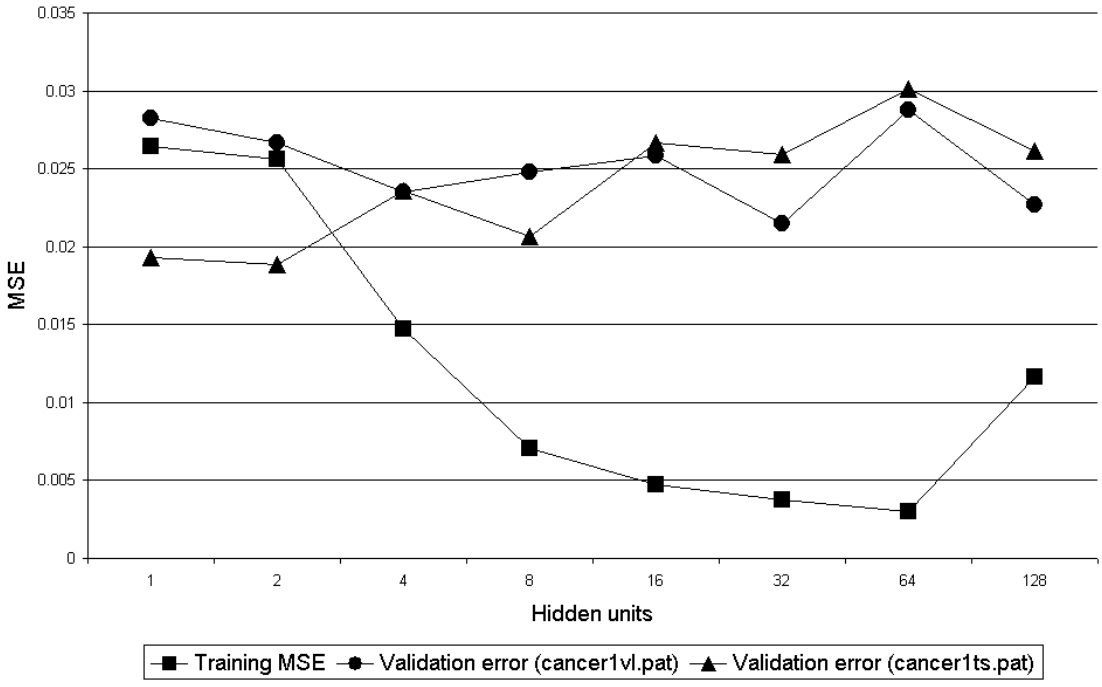
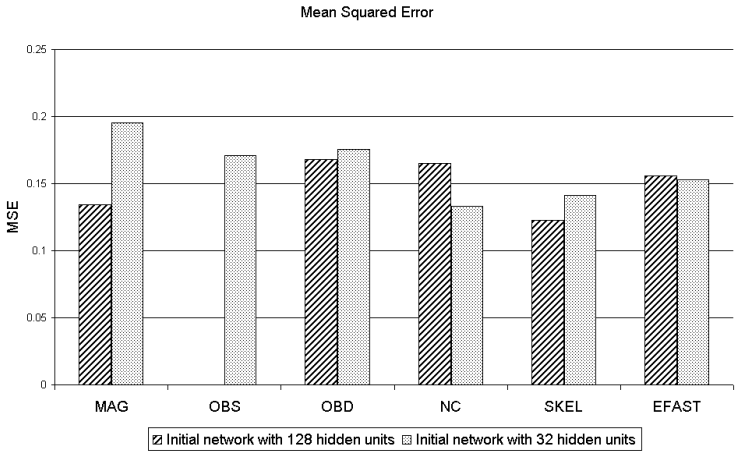
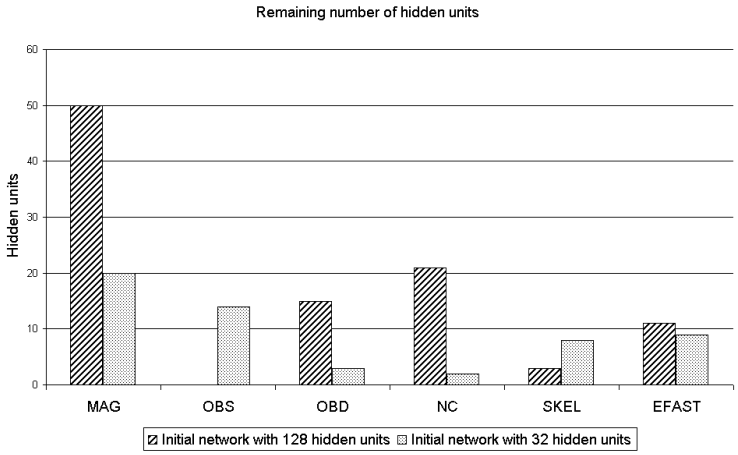


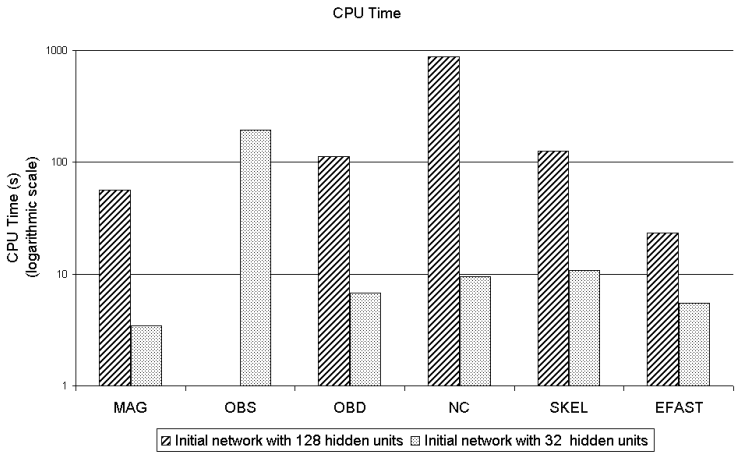
Fig. 14. Results obtained for the additive (or growing phase)



(a)

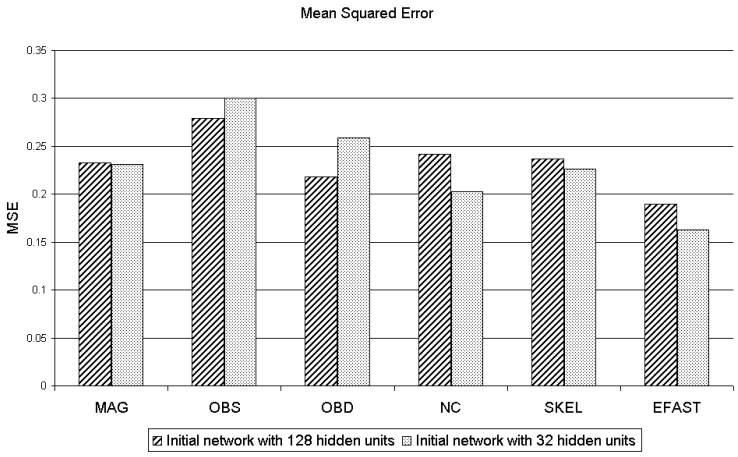


(b)

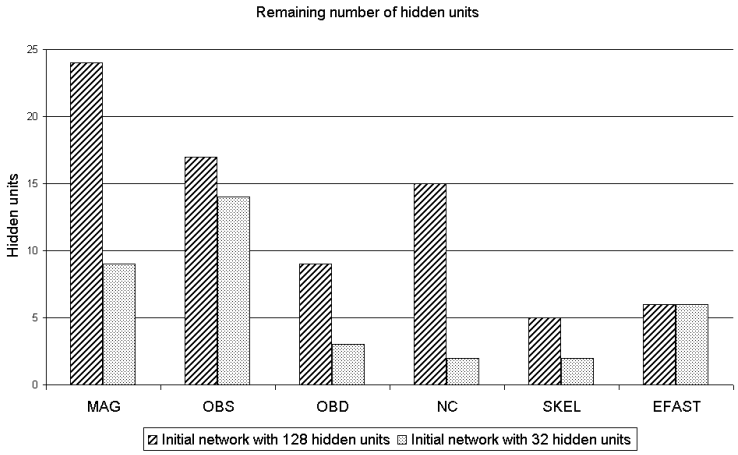


(c)

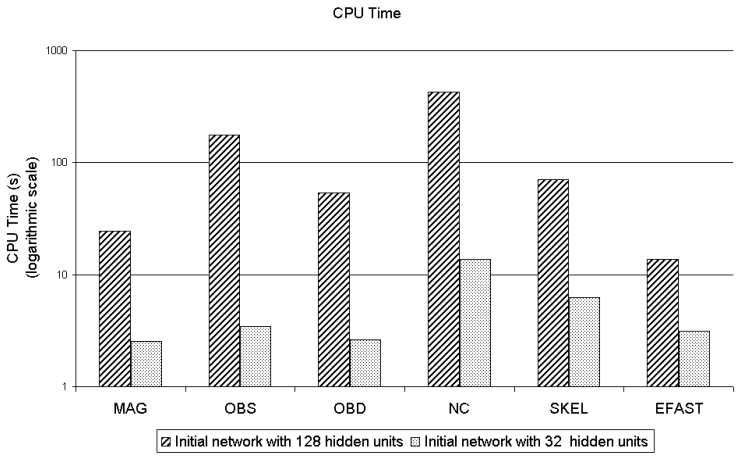
Fig. 15. Card: (a) mse, (b) hidden units and (c) cpu time



(a)

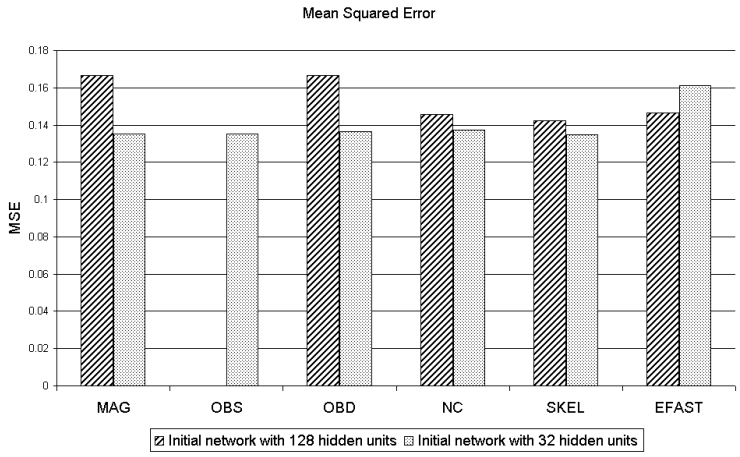


(b)

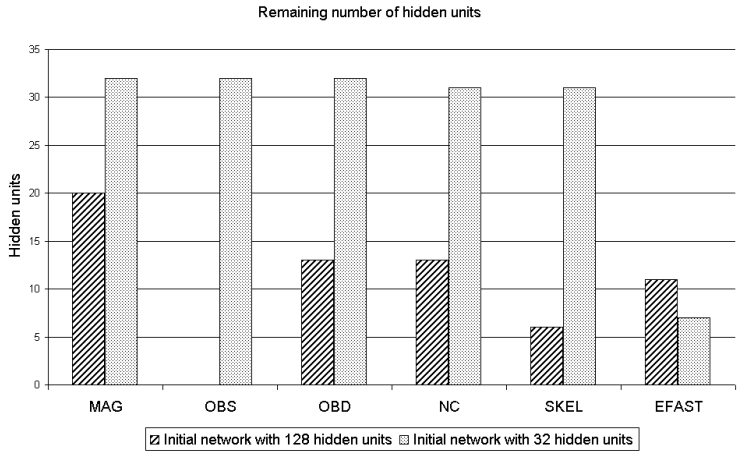


(c)

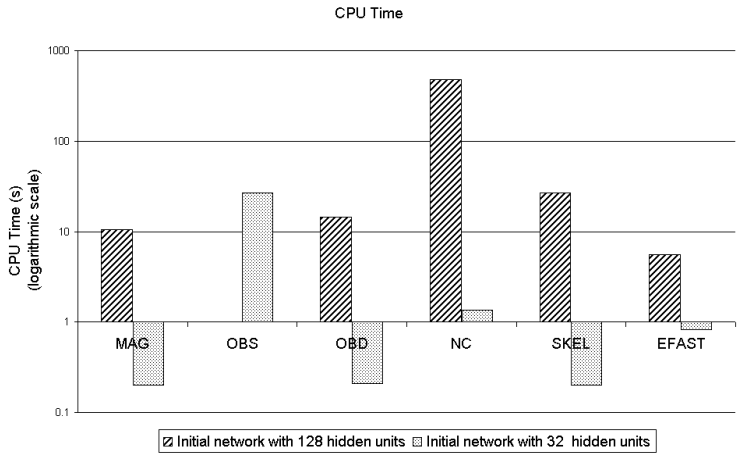
Fig. 16. Diabetes: (a) mse, (b) hidden units and (c) cpu time



(a)

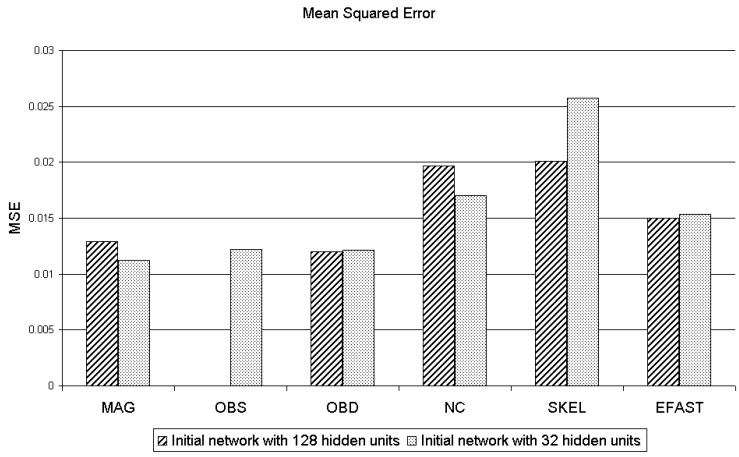


(b)

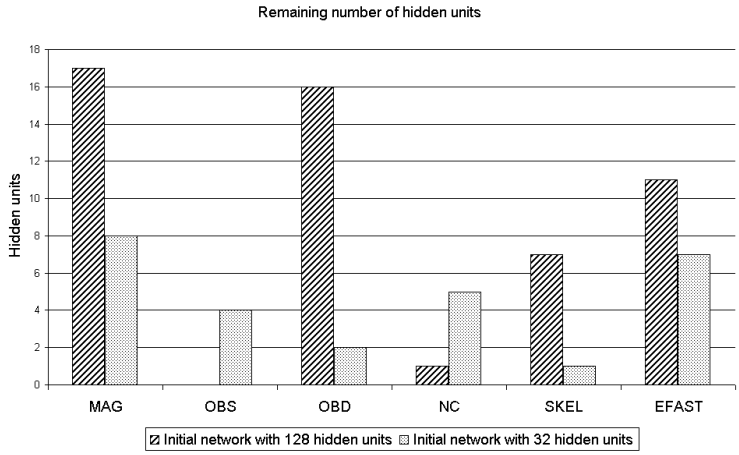


(c)

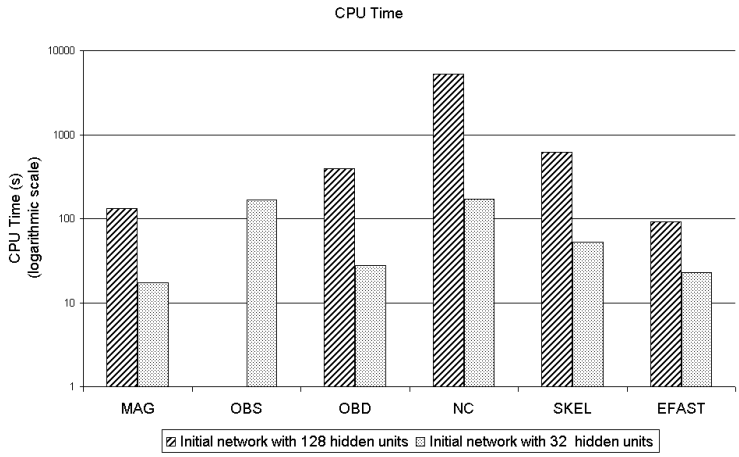
Fig. 17. Horse: (a) mse, (b) hidden units and (c) cpu time



(a)

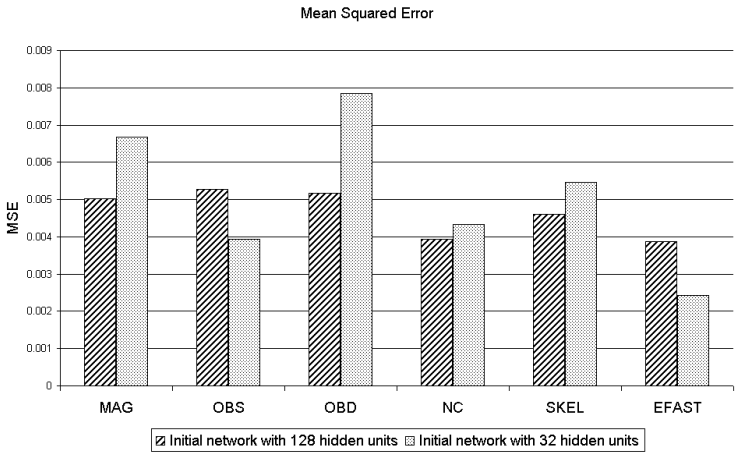


(b)

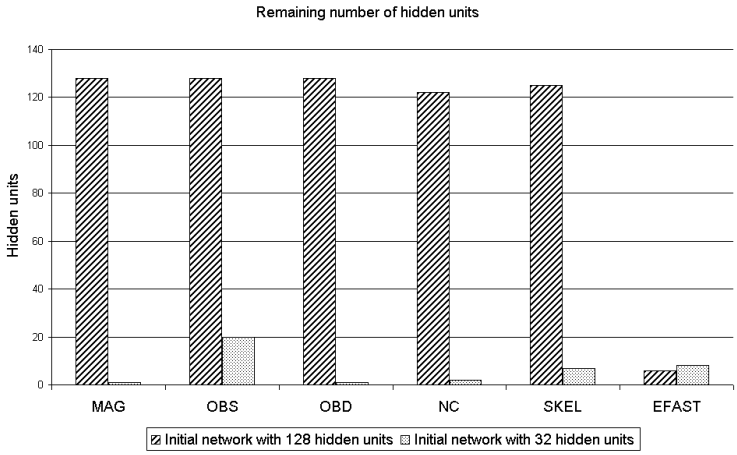


(c)

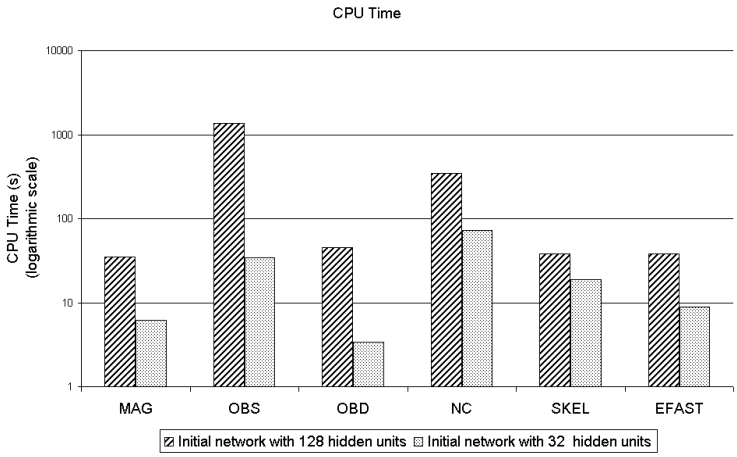
Fig. 18. Thyroid: (a) mse, (b) hidden units and (c) cpu time



(a)

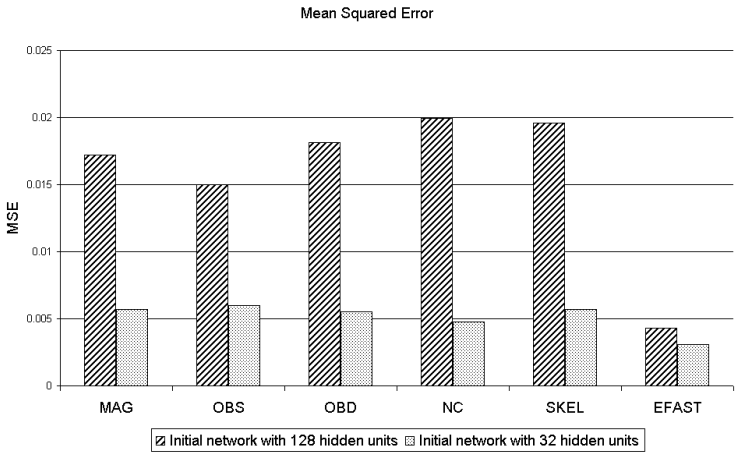


(b)

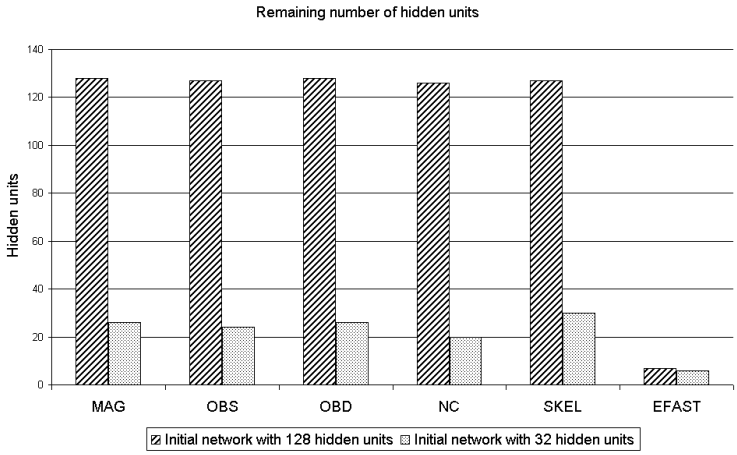


(c)

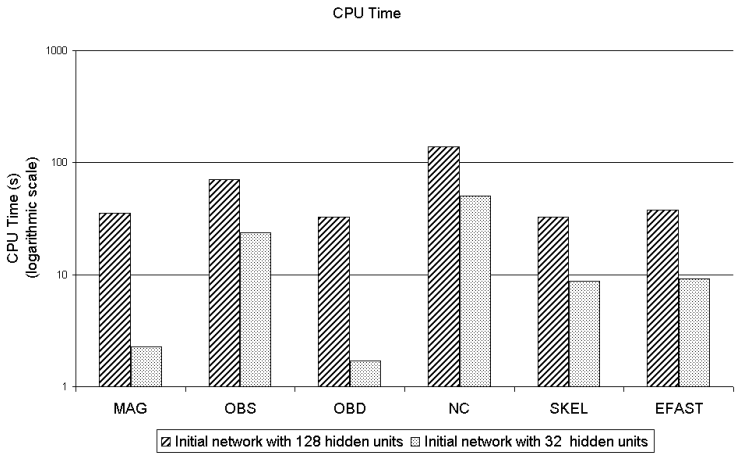
Fig. 19. Building1: (a) mse, (b) hidden units and (c) cpu time



(a)

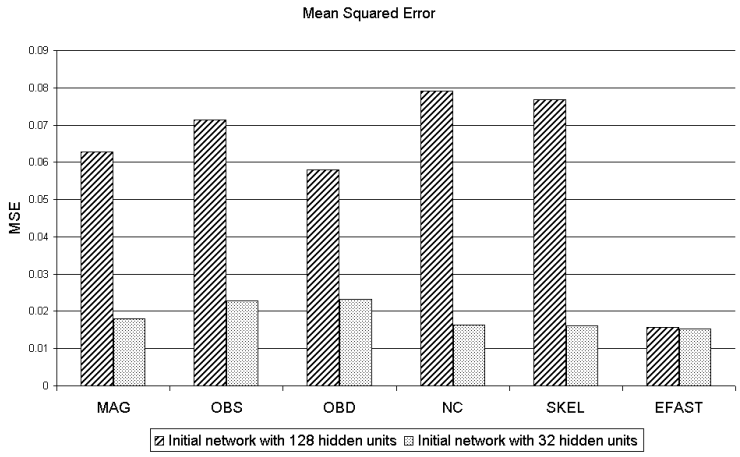


(b)

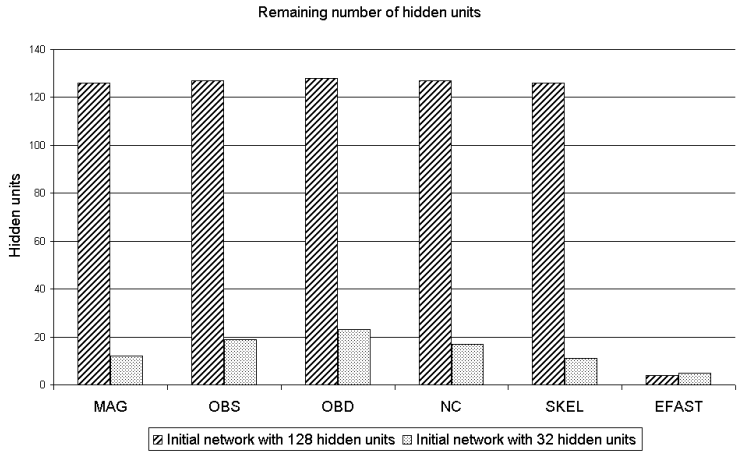


(c)

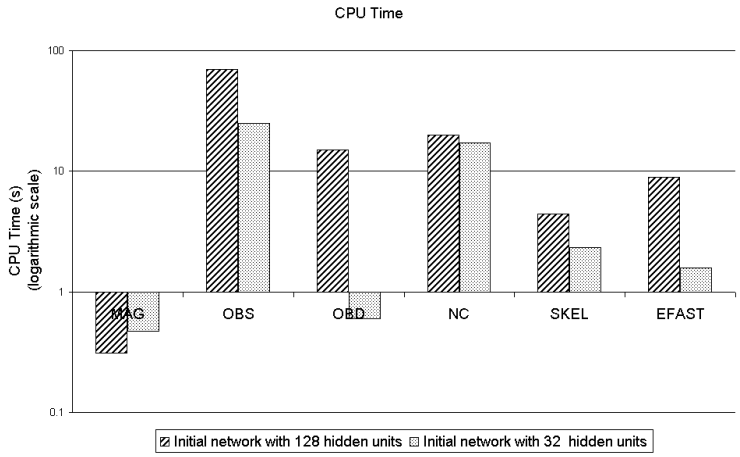
Fig. 20. Building2: (a) mse, (b) hidden units and (c) cpu time



(a)

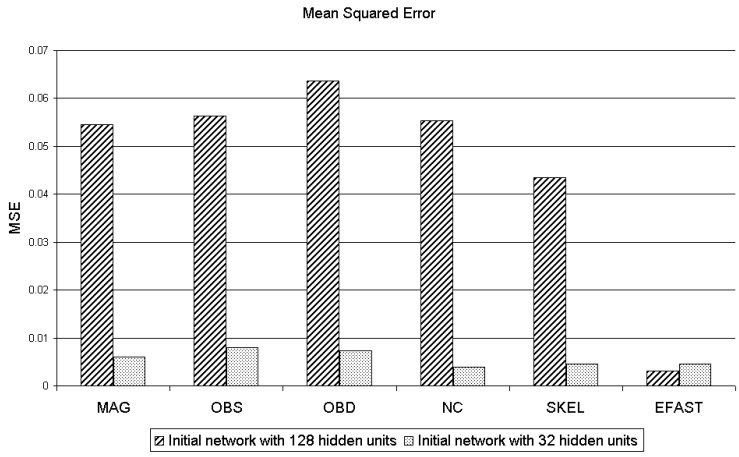


(b)

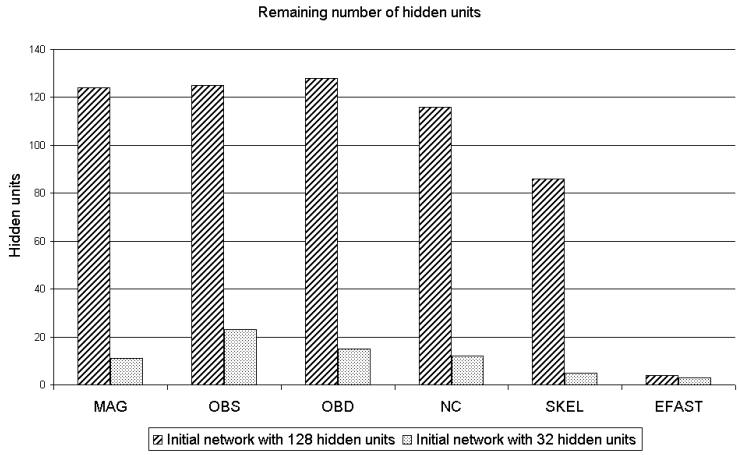


(c)

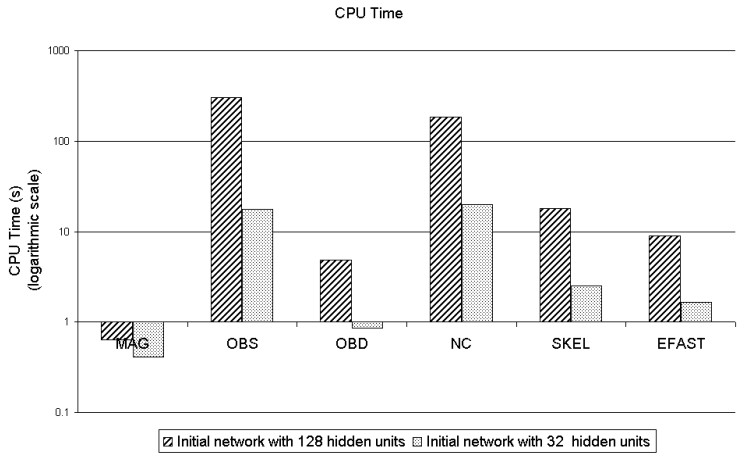
Fig. 21. Flare1: (a) mse, (b) hidden units and (c) cpu time



(a)

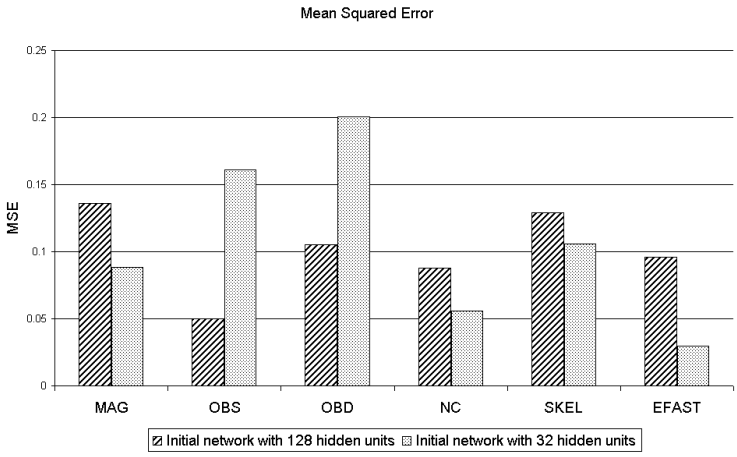


(b)

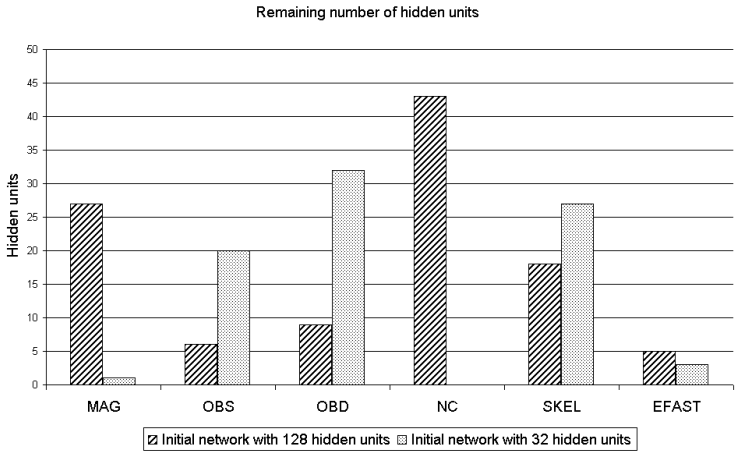


(c)

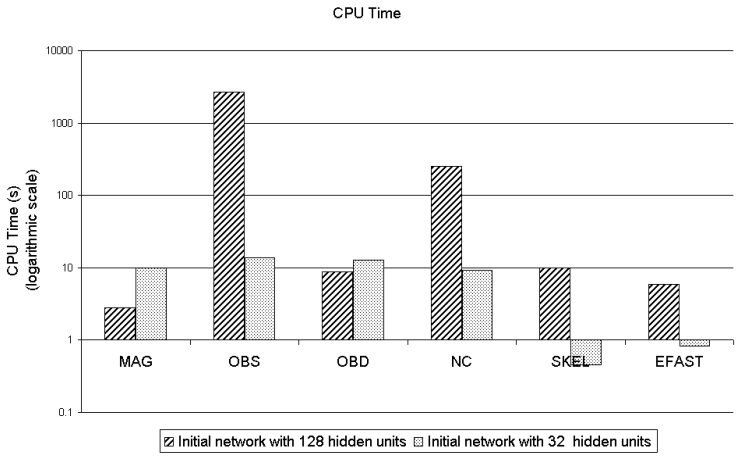
Fig. 22. Flare2: (a) mse, (b) hidden units and (c) cpu time



(a)

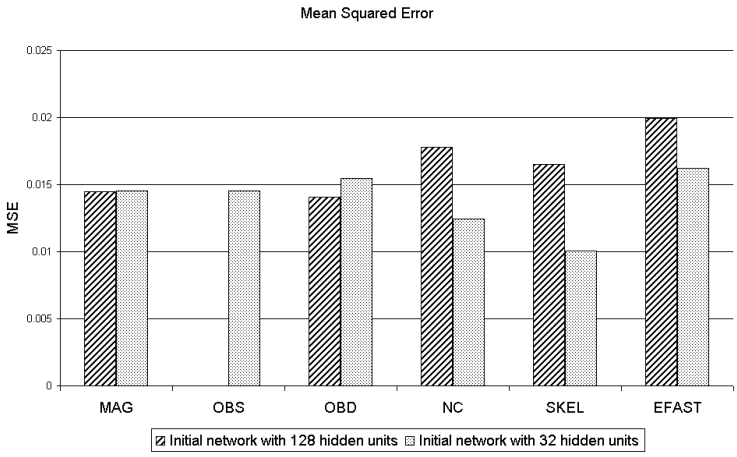


(b)

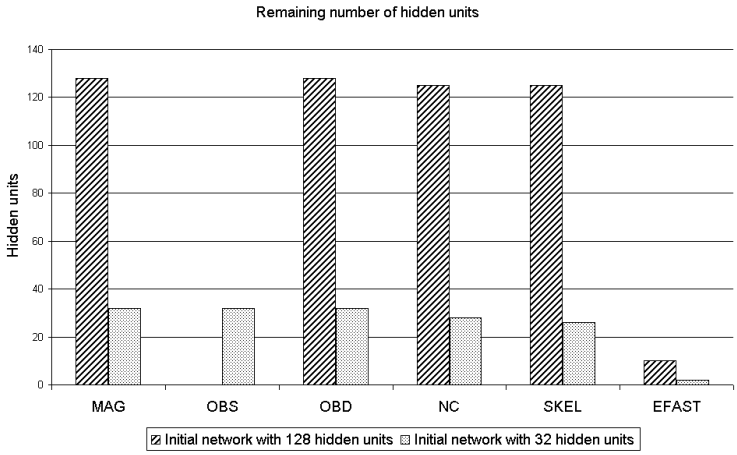


(c)

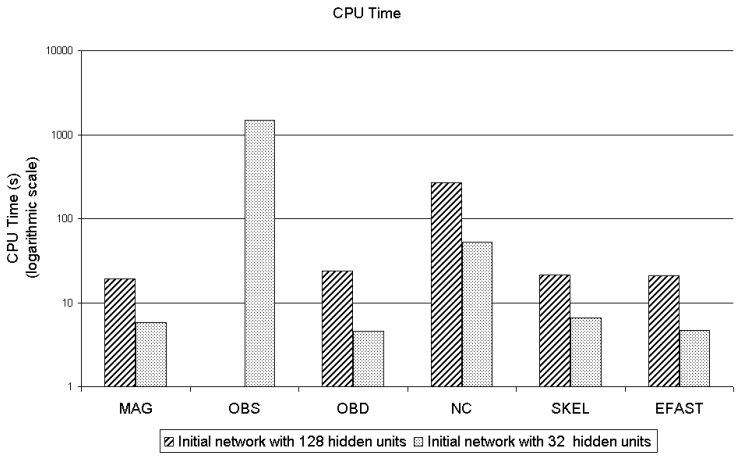
Fig. 23. Heart: (a) mse, (b) hidden units and (c) cpu time



(a)



(b)



(c)

Fig. 24. EES: (a) mse, (b) hidden units and (c) cpu time