LETTER Communicated by Sara Solla

Architecture-Independent Approximation of Functions

Vicente Ruiz de Angulo
Carme Torras
Institut de Robòtica i Informàtica Industrial, (CSIC-UPC), 08034-Barcelona, Spain

We show that minimizing the expected error of a feedforward network over a distribution of weights results in an approximation that tends to be independent of network size as the number of hidden units grows. This minimization can be easily performed, and the complexity of the resulting function implemented by the network is regulated by the variance of the weight distribution. For a fixed variance, there is a number of hidden units above which either the implemented function does not change or the change is slight and tends to zero as the size of the network grows. In sum, the control of the complexity depends on only the variance, not the architecture, provided it is large enough.

1 Introduction

Neural networks are function approximators with a unique feature, their large number of adaptable parameters, that makes them general and powerful estimators. But when the number of samples is not high compared to the number of weights, this power should be restricted in some way to obtain useful approximations. The good results obtained almost from the very beginning of the revival of connectionism in the past decade, particularly with backpropagation networks, can be explained by the adoption of methods against overfitting, some of them first used in the context of neural networks, like early stopping. These methods can be divided in two groups.

The first group refers to architecture selection and manipulation, and consists basically of choosing networks with few parameters. For certain applications with known invariances, this can be accomplished by forcing some groups of weights to take the same value, that is unifying parameters (Lang, Waibel, & Hinton, 1990). But in general this is done by limiting the number of weights or hidden units. A popular method is to begin with a large network and then eliminate the least significant parameters for the fitting, during or after training (Le Cun, Denker, & Solla, 1990). In the latter case, it is usual to repeat the learning-pruning cycle several times.

The second group of methods adds a regularization term to the cost function in order to constrain the weights to minimize the regularizer also. The relative weight given to the regularizer is controlled by means of a regularization coefficient. The typical regularizer for neural networks is

a quadratic function of the weights, limiting their magnitude. The early stopping technique, which avoids the growth of the weights in the later stages of learning, has a similar working principle.

There is a third option: the use of random weights. The idea is that the information contained in a random weight depends on its variance and, thus, the fitting power of a network can be controlled through the amplitude of its weight distribution. The use of random weights was first suggested by Hanson (1990) to avoid poor minima. Later, An (1996) and Murray and Edwards (1993, 1994) showed the utility of the approach to improve generalization, especially for classification problems. Hinton and van Camp (1993a, 1993b) used individual variances for each weight that were adapted during learning to minimize a minimum description length cost.

Here we show that simple noise addition, in the sense indicated below, has very interesting properties. Being a single-variance version of the Hinton and van Camp framework, it is tantamount to a regularization approach, but at the same time, in practice, it also appertains to the architecture selection methods. Indeed we show that the variance determines the surviving architecture and the optimal weight vector, provided the network has enough hidden units. Thus, there exists a one-to-one correspondence between the regularization coefficient and the function implemented by the trained network, regardless of the architecture.

2 Learning with Noisy Weights

This article studies the weight configurations obtained by minimizing a cost function $E_P(W)$ that is the expectation of $E(W)$,

$$E_P(W) = \int E(W + R)\, P(R)\, dR, \tag{2.1}$$

where W is the weight vector and R is a vector of random variables, each obeying the same symmetric distribution $P(\cdot)$. In regression networks, the error function is taken as follows:

$$E(W) = \sum_p E^{p)}(W) = \frac{1}{2} \sum_p ||F(W; X^{p)}) - D^{p)}||^2, \tag{2.2}$$

$F(W, X^{p)})$ being the output of the network when the pth input pattern $X^{p)}$ is presented, and $D^{p)}$ is the pth output pattern.

A deterministic algorithm was developed by Ruiz de Angulo and Torras (1994) to minimize equation 2.1 without sampling the weight distribution for the purpose of mitigating weight perturbation effects. It is rather simple and has been shown to provide very accurate minima for all variances of P. For this reason, we will use it in all the experiments presented in this article.

This algorithm is based on the following equivalence,

$$E_P(W) = E(W) + \frac{\sigma^2}{2} \sum_{j,i} \frac{\partial^2 E}{\partial w_{ji}^2} + \mathcal{O}(\mu_4) \approx E(W) + \frac{\sigma^2}{2} \sum_{j,i} \left(\frac{\partial F}{\partial w_{ji}} \right)^2, \quad (2.3)$$

σ^2 being the variance of R, μ_4 its fourth-order moment, and w_{ji} the weight departing form unit i and impinging on unit j. $\frac{\sigma^2}{2}$ can be considered a regularization coefficient, which we call α.

The equality in equation 2.3 holds everywhere as a consequence of the symmetry of P, which cancels out the terms associated with the first and third derivatives, as well as the nondiagonal terms of the Hessian. When the variance is small, fourth- and higher-order moments are very small, and the terms associated with them can be neglected. Instead, when the variance is large, the minimization of $E_P(W)$ produces networks whose fourth and higher derivatives are small, so that those same terms also can be neglected. These phenomena complement one another in such a way that the $\mathcal{O}(\mu_4)$ terms are not important for any variance in the minimum of $E_P(W)$. Thus, any minimization algorithm can work asymptotically without these terms.

In the approximation step of equation 2.3, a term dependent on both the pattern misfits $F(W; X^{p)}) - D^{p)}$ and the second derivatives of F has been dropped. This term can also be neglected in the minimum of $E_P(W)$ due to similar reasons: when the variance is small, the misfits are small, and when the variance is large, the minimum tends to have small second derivatives of F. Observe that this argument requires that for $\sigma = 0$, the network is able to approximate all the training points fairly well. When the training set contains points with the same input and different outputs, we assume that the network is able to interpolate the average of these outputs (which is the result of the minimization of the plain sum-of-squares error).[1] If this condition is satisfied, the sum of the neglected terms dependent on the misfits is null.

The accuracy of an algorithm based on this approximation was demonstrated for all variances in Ruiz de Angulo (1996) and Ruiz de Angulo and Torras (1998) by means of comparisons with the exact minimization for a very simple form of P, namely, one giving equal probability to the extreme points of a cross centered at the origin, and zero probability elsewhere.

With this approximation, the derivatives of the expected error for pattern p, $E_P^p(W)$, for a two-layer regression network with the above-defined $E(W)$[2]

[1] Thus, we are assuming that the network is large enough to do this, and that the control of the complexity of the approximation is carried out by regulating σ and not the number of hidden units.

[2] The derivatives for a classification network using the relative entropy error function and whose output units' activation function is *tanh* are the same as in equation 2.4.

and linear output units, are

$$\frac{\partial E_P^{p)}}{\partial w_{ji}}(W) \approx \frac{\partial E^{p)}}{\partial w_{ji}} + \alpha \begin{cases} 2w_{ji}\left(y_i'\right)^2 Q^{p)} & \text{if } j \text{ is an output unit, } i \neq \text{bias} \\ 0 & \text{if } j \text{ is an output unit, } i = \text{bias} \\ x_i S_j & \text{if } j \text{ is a hidden unit} \end{cases} \quad (2.4)$$

where $Q^{p)} = ||X^{p)}||^2 + 1$ (we are considering a network with a bias unit connected to all hidden and output units), $S_j = 2y_j'(n_O y_j + y_j'' Q^{p)} \sum_m w_{mj}^2)$, n_O is the number of output units, y_j is the activation function value of the jth hidden unit, and y_j' and y_j'' are its first and second derivatives, respectively. S_j is the same for all the connections impinging on a hidden unit and thus must be calculated only once for each hidden unit and not for each input-hidden layer weight.

The deterministic algorithm consists of using the derivatives, equation 2.4, instead of the plain error derivatives, inside a backpropagation procedure.[3]

An alternative to this algorithm would be to add noise to the weights during learning (An, 1996), but an insurmountable computational time would be required to get the experimental results presented here. This is due to the high dimensionality of W, which imposes the use of very small learning rates to permit collecting enough statistics of $\bigtriangledown E_P(W)$, through samples of $E^{p)}(W + R)$, before the characteristics of $E_P(W)$ change too much.

3 Experimental Results

In this section we present two types of experimental results: qualitative comparison of the weight configurations resulting from training networks of different sizes under the same variance and numeric evaluation of the similarity of networks across a range of variances. The experiments are aimed at determining under what conditions using the same weight variance leads to equivalent or quasi-equivalent network configurations. Even if two networks in their global minima may have identical weight configurations, this is hard to visualize, because it is impossible to reach exactly those minima, and having different architectures, the networks follow different minimization paths. For this reason, we have chosen a convenient criterion for stopping learning: the quantity

$$AVG(W) = \sqrt{\frac{\sum_{j,i}\left(\frac{\partial E_P}{\partial w_{ji}}(W)\right)^2}{n_p}} \quad (3.1)$$

must be less than or equal to a fixed number, where n_p is the number of patterns in the training set.

[3] A similar algorithm based on equation 2.3 for radial basis function networks is also simple to derive and implement (Ruiz de Angulo, 1996).

In all the experiments, we will use symmetric activation functions for the network units. With this kind of unit, when all weights (except the biases of the output units) are zero, the derivatives of the network function F with respect to all the weights are zero. Therefore, the regularizer term $\frac{\sigma^2}{2} \sum_{j,i} \left(\frac{\partial F}{\partial w_{ji}} \right)^2$ has its minimum in this type of configuration. This implies that for infinite variance, the minimizer of E_P is that with all weights set to zero except the biases of the output units, which are set to the mean of the corresponding output pattern component.

3.1 Architecture Independence. We first illustrate the workings of the algorithm on the one-dimensional function $.3x^3+.3x^2-\frac{10}{3(x+3)^2}$ in the interval $[-1.5, 1.5]$, as plotted in Figure 1a. In Figure 1b the weights of a network with four hidden units (HU) resulting from applying the algorithm to 20 patterns randomly drawn from that interval, up to $AVG(W) = .0001$ and using $\alpha = .001$, are shown. Figure 1c displays the results obtained under the same conditions with a network of 8 HU. The visible result is that the number of surviving units is the same in both networks: two. Moreover, the weights of the first unit in Figure 1b are the same as those of the fifth unit in Figure 1c. The fourth unit in Figure 1b shows a direct correspondence with the first unit in Figure 1c. The weights have pairwise the same magnitude, and since the signs of both the input weights and the output weights are inverted, the two units have the same functionality. It can be concluded that the two networks implement the same approximation of the function $.3x^3 + .3x^2 - \frac{10}{3(x+3)^2}$, represented also in Figure 1a.

This is not a rare event but a common outcome of our algorithm: same number of surviving hidden units and exact (except for sign inversions) one-to-one weight correspondences, as can be seen in a more systematic experiment. This time the training set was 30 samples of the function $\sin(x_1+x_2)+\eta$ in the interval $[-\pi, \pi]$, η being a gaussian noise of standard deviation .5. Figure 2 displays this function for $\eta = 0$ and the noisy points used to train the networks in this experiment. Two networks, one with 6 HU and another with 12 HU, were randomly initialized and subsequently trained with these data up to $AVG(W) = .00005$. This process was repeated for a number of times with different α's. Figure 3 shows the results. Since, as a consequence of the minimization of $E_P(W)$, there are no useless weights alive in the network, the number of surviving hidden units can be determined by counting the non-null hidden-to-output weights. Due to the similar values taken by two hidden-to-output weights some times, the counting is not very clear in Figure 3a, but there are four surviving units when $\alpha = .0066$, three between .013 and .026, two between .033 and .046, and one for larger α's. On the other hand, again there is an exact correspondence between the absolute values of the weights of the surviving units in the 6 HU network and those in the 12 HU network for all $\alpha \geq .0066$ (except for $\alpha = .053$, where one of the networks gets trapped in a local minimum; see the next section). Note

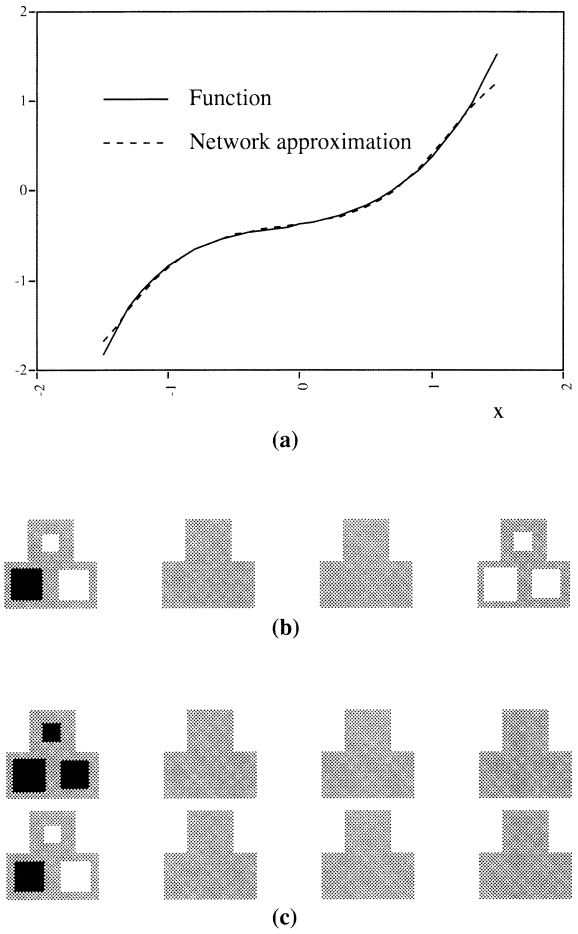

Figure 1: (a) Plot of the function $.3x^3 + .3x^2 - \frac{10}{3(x+3)^2}$ in the interval $[-1.5, 1.5]$. The results of training a (b) 4 HU network and an (c) 8 HU network are shown. Each of the blocks in the figure contains all the weights associated with a hidden unit. Weights are represented by squares—white if their value is positive and black if their value is negative. The size of the square is proportional to the magnitude of the weight. The top weight in the block is the weight from the hidden unit to the output unit. The weights in the bottom row correspond to the weights incoming to the hidden unit. The function implemented by these networks is represented in *a* with a dashed line.

that a network with only 3 HU will not be able to attain the same weight configuration as the 12 HU network when $\alpha = .0066$, because there are four surviving units in the latter network.

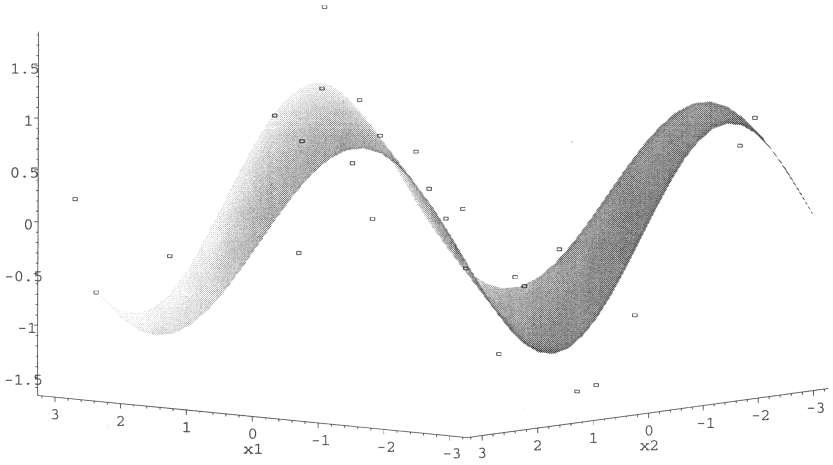

Figure 2: Plot of the function $\sin(x_1 + x_2)$ in the interval $[-\pi, \pi]$. The small squares constitute the training set used in the experiment of the next figure, obtained by sampling the function $\sin(x_1 + x_2) + \eta$, where η is a gaussian noise with standard deviation 0.5.

In general, given two networks with different number of units, there exists a value of α, which we call the confluence point, such that for any greater value, both networks implement identical functions. At the resolution level of Figure 3, the confluence point is .0066 (the first value greater than zero), but using much smaller steps results in a more accurate confluence point estimation of .002. (In section 3.3 we will see an example with a much larger confluence point.)

Another consequence of using the algorithm with $\alpha > 0$ is that above a certain number of iterations, the generalization error does not change with more training. Thus, overffiting is limited. Although the selection of the best variance (or regularizer coefficient) for generalization is not the purpose of this article, we show in Figure 4 the generalization error of the intensively trained networks used in the preceding figure. It can be seen that the networks with 6 HUs and 12 HUs generalize equally well for $\alpha > 0$, except in the case $\alpha = .053$, where the former falls in a local minimum.

In subsequent experiments with a fixed α, it has been observed that the number of surviving hidden units increases with the number of data items to cope with the increase in the amount of information available (Ruiz de Angulo, 1996).

3.2 Local Minima. A question that arises naturally after seeing the results of the preceding experiments is whether two networks with different

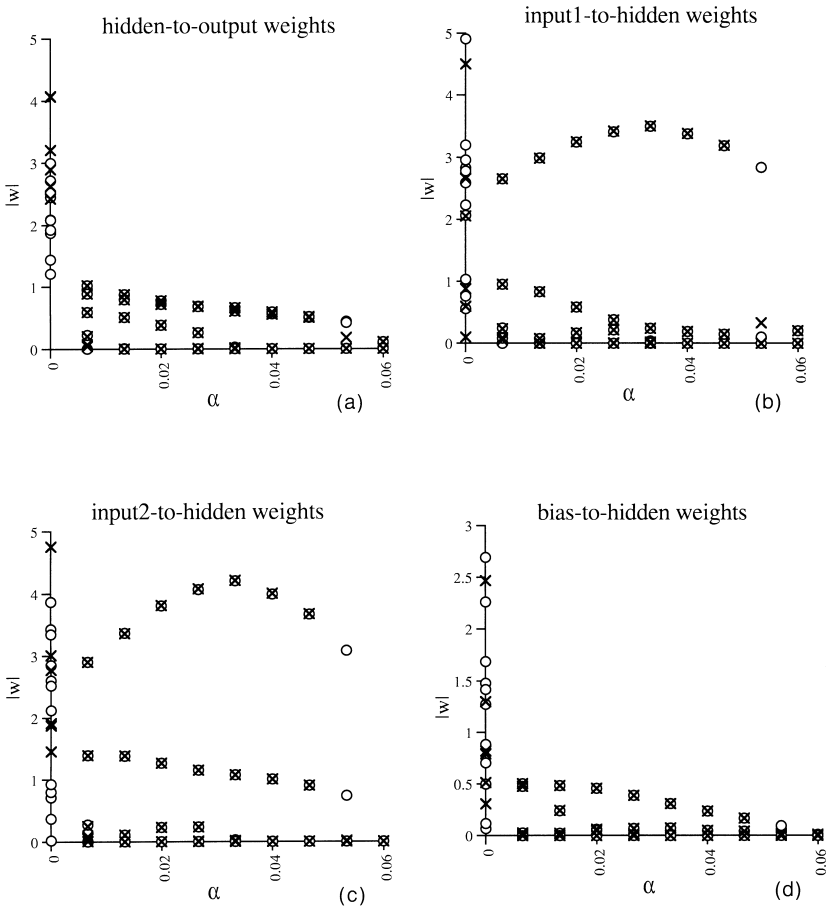

Figure 3: Representation of the absolute values of weights after training. Those of the 6 HU network are marked with a cross and those of the 12-HU network with a circle. Note that the weights lying on the axis of abcissas are null, and thus correspond to nonsurviving units.

number of neurons, trained with the same training set and α, will always produce equivalent approximations.

First, let us state that all the claims we make in this article refer to well-minimized networks. No strong claims can be made about weights resulting from incomplete optimizations. Thus, no ways to regulate complexity extrinsic to the cost function, such as early stopping, are applied.

Often two networks must be trained for a long time before their similarity is clearly noticed. It is not unusual (as in the case of minimizing the plain

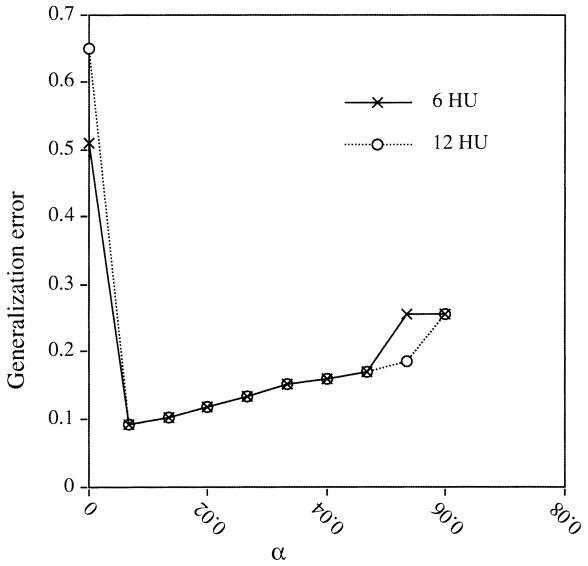

Figure 4: Generalization error of the networks displayed in Figure 3.

$E(W)$) that learning slows down in nearly flat surfaces of $E_P(W)$, sometimes leading to replicated units, which disappear after enough training.

From time to time, even after long training periods, one gets different configurations, including cases with different number of remaining hidden units. Two situations need to be distinguished here, depending on whether the number of surviving units in the larger network is greater than the number of available hidden units in the smaller network or not. In the first situation, there is no opportunity for confluence. In the second, if the configuration reached by the larger network is a global minimum, it must also be a global minimum for the smaller network. Consequently, in the second situation, we have always verified that at least one of the networks had been trapped in a local minimum.

Thus, our claim for architecture-independent approximation still holds, but must be stated in the context of global minimization of the regularized function.

The frequency of local minima is not too high. Results such as those in Figure 1 are common in a first trial. However, we must acknowledge that results such as those displayed in Figure 3, in which with fixed initial weights equal configurations are obtained for the entire range of α's, are uncommon. Experimental results indicate that it is more likely to fall in local minima when the number of surviving hidden units required by the global minimum is close to that of the network and also when this number is much larger.

A strategy to avoid local minima is to add a random perturbation several times in the middle and late stages of learning.

3.3 Quasi-Similar Approximations. So far we have seen only one of the two types of hidden units produced by the algorithm, which hereafter will be called nonreplicated units: those whose weight pattern appears only once in the network. But the algorithm often produces weight configurations in which several units are replicated. In these cases the number of surviving units is not indicative of the complexity of the network, and, indeed, it is not unusual that an increase of α produces a minimum with more surviving units, some of them being replicated. The replication of a hidden unit depends on the degree of linearity that its output exhibits in the range of the input training data.

The nonreplicated units are always nonlinear, whereas the replicated ones may be of either type. As will be shown next, when the replicated units are nonlinear, the global minimum contains a fixed number of them, irrespective of the size of the network. On the contrary, the number of linear replicated units grows with network size, tending to fill the whole network.

Suppose we have a network with a number $n > 0$ of nonsurviving hidden units, and assume that one of the surviving units is almost linear. Since the second derivatives associated with a hidden unit depend on only the squares of the hidden-to-output weights and on the square of the hidden unit activation, splitting a linear hidden unit keeps the same $E(W)$ while reducing the second derivatives part of $E_P(W)$. Thus, returning to our imaginary network, by splitting the linear hidden unit into $n+1$ equal units and replicating its weight pattern for all the nonsurviving units, the minimum value of $E_P(W)$ is obtained with the same $E(W)$. For this reason, we call these units replicated units of the fill-everything type. As an example, see Figure 5, where the weights resulting from training a 6 HU and a 12 HU network with 25 samples of the function $\sin(x_1 + x_2)$ up to $AVG(W) = .00005$, using $\alpha = .02$, are shown. One can see that in Figure 5a, there are five functionally identical units, the first and second of which have inverted sign patterns in relation to the others. In Figure 5b, that same replicated unit appears 11 times, but the magnitudes of its weights are smaller. Thus, when the resulting weight configuration contains linear replicated units, enlarging the number of hidden units causes a rescaling of these replicated hidden units. Because the hidden units cannot be completely linear, this rescaling imposes a slight adjustment on the remaining, nonlinear units. Figure 6 shows the weights of the same 6 HU and 12 HU networks used in Figure 5 for a complete range of α's. Here, for $0 < \alpha < .04$, the configuration in both networks consists of a nonreplicated unit and fill-everything linear units. When the number of hidden units tends to infinity, the rescaling tends to be completely linear, and the adjustment of the nonreplicated units tends to zero. Instead, when there are too few units, the rescaling makes the hidden units too nonlinear, and thus, a completely different solution is found. Fig-

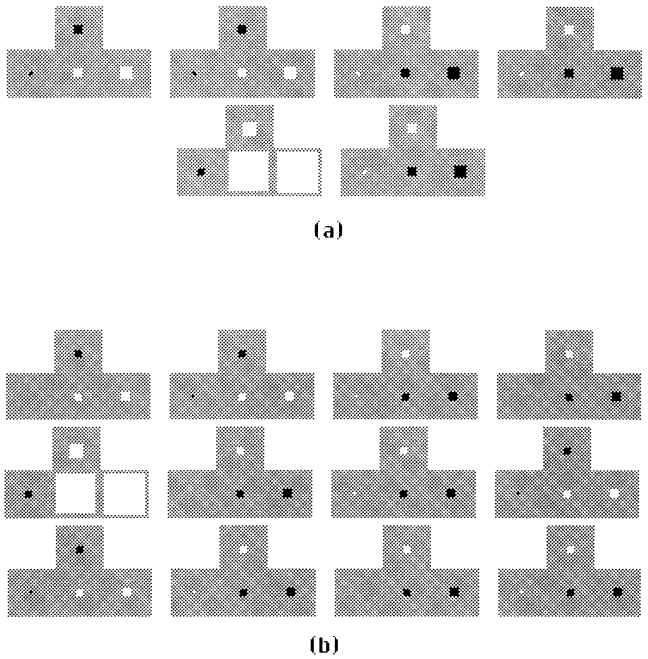

Figure 5: Results of training a 6 HU and a 12 HU network with 25 samples of the function $\sin(x_1 + x_2)$, using $\alpha = .02$. The resulting configurations contain replicated units of the fill-everything type.

ure 7 shows an example with the same settings as in Figure 5, but supplying more information for the network (30 patterns are used) and requiring more fidelity to the data ($\alpha = .0066$). Here, the absolute values of the weights of the replicated units are larger, and, thus, they do not admit a redistribution into just five units. Therefore, the 6 HU network converges to an entirely different configuration.

On the other hand, when the replicated units are significantly nonlinear, the scenario is much more restricted, because a fixed number of replicated units is involved in the global minimum. We call these units replicated units of the fixed number type. For instance, in Figure 6, $\alpha = .04$ produces a solution with four replicated units, in both the 12 HU and the 6 HU networks. Different initializations lead always to the same type of solution with the 12 HU network, but this solution is more difficult to find for the 6 HU network, because it requires almost all its hidden units, and approximately half of the times produces a solution with three replicated units. The behavior of the 6 HU network is also favored by another fact: if the replicated units are not highly nonlinear, one more and one less number of replications can

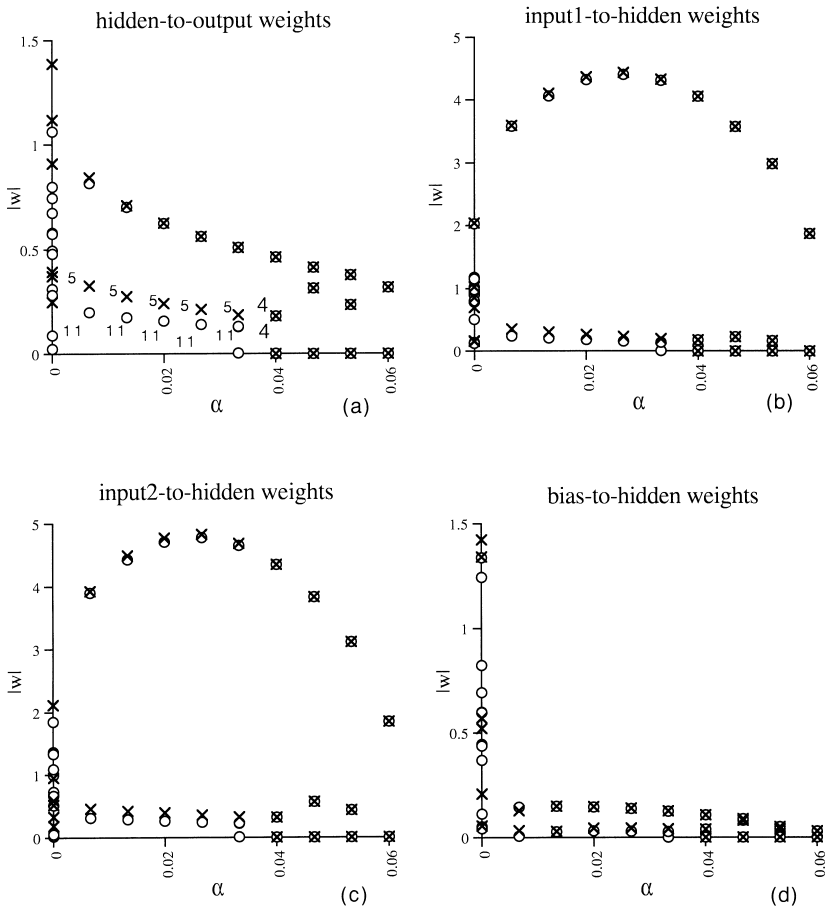

Figure 6: Representation of the weights resulting from training 6 HU and 12 HU networks under the same conditions as in Figure 5, but for a complete range of α's. In the top left plot, the number of weights with the same value is indicated besides the corresponding symbol.

be a solution with an $E_P(W)$ value almost indistinguishable from the global minimum.

A consequence of all this is that quasi-similar functions can be implemented by different networks, even if α is not in their confluence range. To measure the degree of similarity between two functions, we define the functional distance as the mean square distance between the outputs of the two networks in a grid of 10,000 points regularly distributed in the input domain. Figure 8 shows the functional distances between architectures with

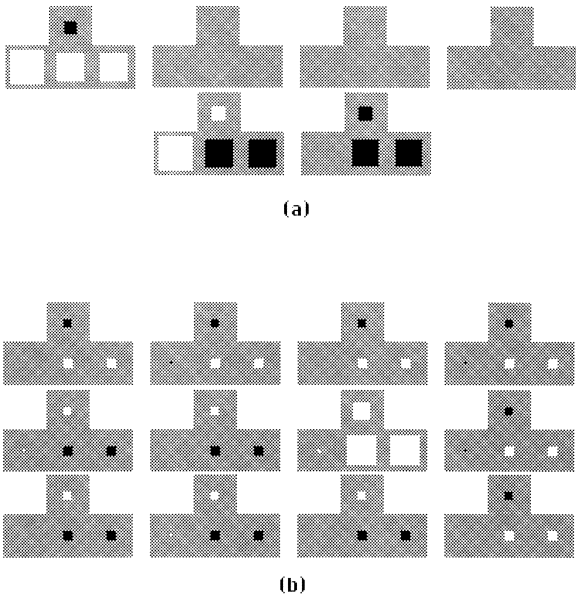

Figure 7: Results of training a 6 HU and a 12 HU network with 30 samples of the function $\sin(x_1 + x_2)$, using $\alpha = .0066$. Here, the 6 HU is too small to accommodate the rescaling of the fill-everything units, and different solutions are found by the two networks.

different number of hidden units, minimized for several α's. Since for the two highest α's, some of the networks fell in local minima, in these cases we used the best minimum selected from three different random trials. It is evident that as α grows, all the architectures tend to produce the same results. But the most interesting observation from this graph is that for any $\alpha > 0$, the distances between architectures decrease very quickly as the number of hidden units grows, and they are indistinguishable from zero above 50 units. Notice that the comparisons involve networks that differ the more in the number of hidden units, the larger are the architectures. The above observation agrees with the expectation of a tendency to closer similarity for larger nets. Of course, for each given α, all the architectures exhibit almost the same generalization error, especially those above 50 HUs. An interesting fact derived from Figure 8 is that as the sizes of the networks grow, their confluence occurs at lower α values. Thus, the chances that the value of α leading to the best generalization falls within the confluence region increase with the size of the networks. In other words, it seems that good generalization and confluence can be simultaneously achieved for large networks.

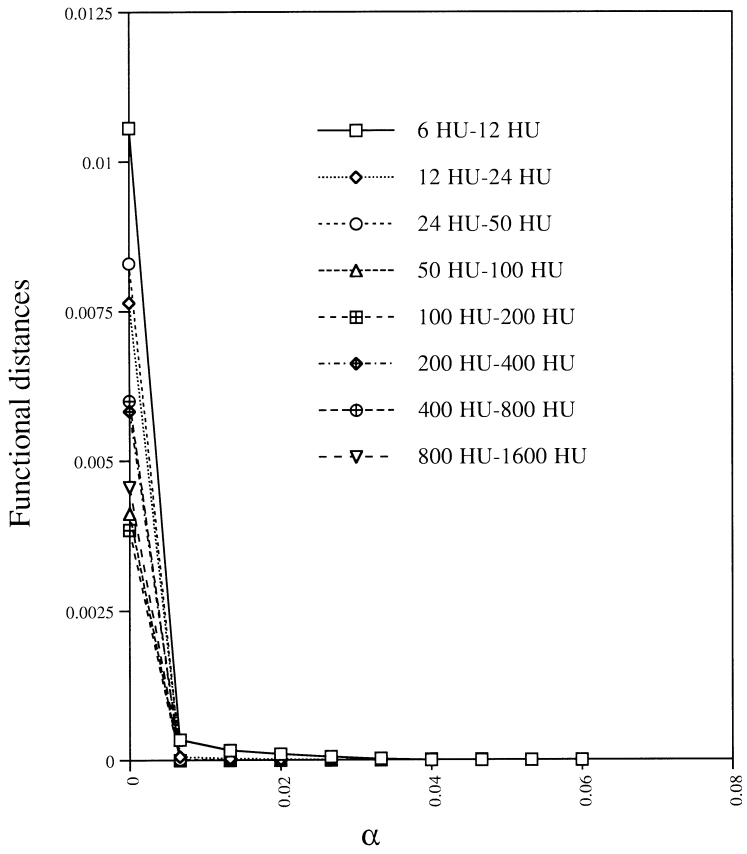

Figure 8: Evaluation of the similarity between the functions implemented by architectures with different numbers of hidden units. A complete range of α's is tested.

In conclusion, no matter what the value of α is, large enough networks make similar approximations. For that reason, we can disregard the network size selection problem and concentrate on the ideal regularizer coefficient to reduce the complexity of the network.

4 Discussion

We have presented a very simple neural network algorithm capable of producing quasi-similar approximations with different network architectures. This algorithm can be viewed as a way of minimizing the expectation of the standard error over a distribution of the weights with a common and fixed variance. Alternatively, it can be seen as the addition of a regular-

ization or penalty term to $E(W)$ to control the degree of adaptivity of the network.

For a given α, if the minimum in the limit of infinite HUs does not contain replicated units of the fill-everything type, there exists a threshold number of units above which the function implemented by any architecture is exactly the same. Otherwise, if the minimum admits replicated units, then there exists also a threshold number of HUs above which linear replicated units begin to appear. Above this point, the networks show a close functional similarity, which approaches quickly the identity as the number of HUs grows. If enough hidden units are provided, in either case the function implemented by a neural network depends only on α, it being independent of the architecture.

To make sure that this architecture-independence property is not shared by all weight-elimination procedures, we have run Optimal Brain Damage (Le Cun et al., 1990) on the same data and the same architectures employed in Figure 8. We have pruned the weights five at a time and have retrained all the networks until only the number of weights indicated in the abscisses is left.[4] The results displayed in Figure 9 are clear, in the sense that the function implemented by the network varies markedly all along the weight removal process; in other words, it does not stabilize beyond a given number of weights.

The results in this article can be put under a Bayesian light. Neal (1996) has advocated convincingly for the use of neural networks with as many hidden units as can be afforded computationally, regardless of the size of the training set. In a proper Bayesian approach, it makes more sense to allow the maximum information to be extracted from the data and let the prior reduce conveniently the complexity (if the prior is correct). From this point of view, the first problem is to look for priors over weights such that the corresponding prior over the function reaches a reasonable limit as the number of hidden units goes to infinity. In fact, not a prior but a family of priors suitably scaled according to the number of HUs is defined by Neal (1996) to obtain such convergence.

The usual type of priors, the gaussian one, converges to gaussian processes—a family of distributions over functions that can be handled more efficiently by a direct implementation. In addition, they produce networks where the contribution of individual HUs is negligible and consequently are unable to represent "hidden features" of the data. Another problem is that implementing Bayesian inference with methods based on the posterior may break down as the number of HUs becomes large. Instead, Monte Carlo methods are able to do the job, but being slow, they may fail to reach the true posterior in a reasonable time for a number of HUs close enough to the limit.

[4] Note that an initial pruning has been necessary to make the number of weights of all networks a multiple of five before beginning to prune the weights five at a time.

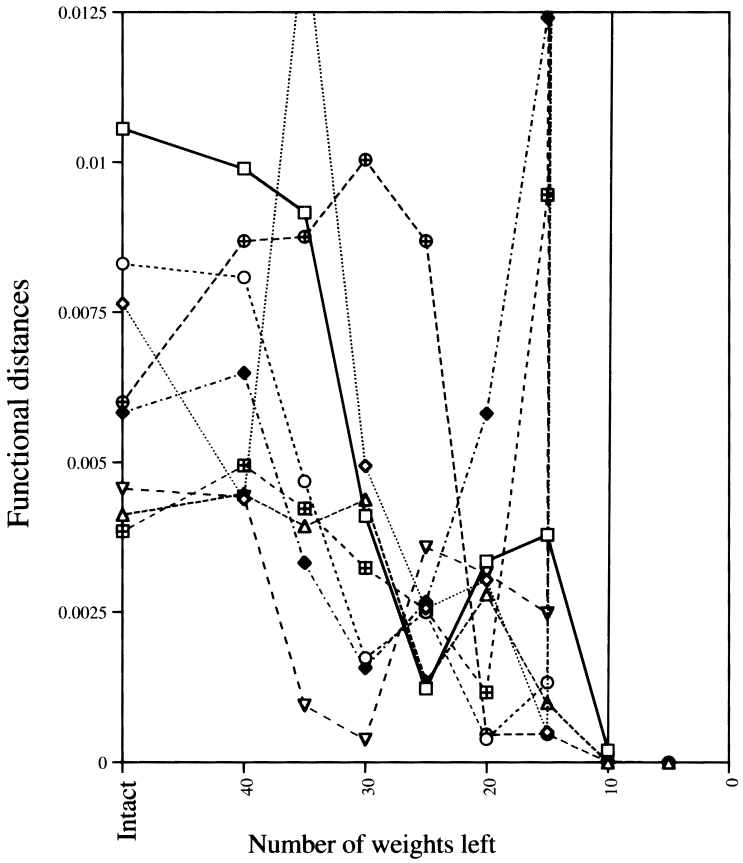

Figure 9: Evaluation of the similarity between the functions implemented by pruned networks (following Optimal Brain Damage) derived from initial architectures containing different numbers of hidden units. The different symbols correspond to the same pairs of networks as in Figure 8, and the scale is also the same as in that figure.

The algorithm presented in this article is based on a regularization term, $\alpha \sum_{j,i} \left(\frac{\partial F}{\partial w_{ji}} \right)^2$, that can be considered a prior on the weights for which, as we have seen, the posterior maximum reaches a limit as the size of the networks goes to infinity. This limit is reached abruptly if there are only nonreplicated and fix-number replicated units and more smoothly if there are fill-everything replicated units. In the former case, the limit is easily reached and identified; in the latter case, a good approximation can also be identified when adding a new neuron produces a rescaling of the replicated units. In addition, in the infinite limit, the contribution of some individual

units (nonreplicated and fixed-number replicated) is relevant, and that of the remaining units can be grouped in a linear unit. This means that the set of surviving units can represent hidden features, a fact that can be useful to link several outputs of the function or for knowledge transference between tasks.

It would be interesting to analyze the features that confer these properties on our prior, so that other priors with these or similar properties can be devised. One of such features is surely the fact that our prior treats the weights in groups associated with the hidden units, not independently.

References

An, G. (1996). The effects of adding noise during back-propagation training on generalization performance. *Neural Computation, 8*, 643–674.

Hanson, S. J. (1990). A stochastic version of the delta rule. *Physica D, 42*, 265–272.

Hinton, G. E., & van Camp, D. (1993a). Keeping neural networks simple. In *Proceedings of the International Conference on Artificial Neural Networks* (pp. 11–18). Amsterdam.

Hinton, G. E., & van Camp, D. (1993b). Keeping neural networks simple by minimizing the description length of the weights. In *Proceedings of the Sixth ACM Conf. Comp. Learning Theory* (pp. 5–13). Santa Cruz.

Lang, K. J., Waibel, A. H., & Hinton, G. E. (1990). A time-delay neural network architecture for isolated word recognition. *Neural Networks, 3*, 23–43.

Le Cun, Y., Denker, J. S., & Solla, S. A. (1990). Optimal brain damage. In D. Touretzky (Ed.), *Advances in neural information processing systems, 2*. San Mateo, CA: Morgan Kaufmann.

Murray, A. F., & Edwards, P. J. (1993). Synaptic weight noise during multilayer perceptron training: Fault tolerance and training improvements. *IEEE Trans. on Neural Networks, 4*, 722–725.

Murray, A. F., & Edwards, P. J. (1994). Enhanced multilayer perceptron performance and fault tolerance resulting from synaptic weight noise during training. *IEEE Trans. on Neural Networks, 5*, 792–802.

Neal, R. M. (1996). *Bayesian learning for neural networks*. New York: Springer-Verlag.

Ruiz de Angulo, V. (1996). *Interferencia catastrófica en redes neuronales: Soluciones y relación con otros problemas del conexionismo*. Unpublished doctoral dissertation, Basque Country University.

Ruiz de Angulo, V., & Torras, C. (1994). Random weights and regularization. In *Proceedings of the International Conference on Artificial Neural Networks* (pp. 1456–1459). Sorrento.

Ruiz de Angulo, V., & Torras, C. (1998). *Averaging over networks: Properties, evaluation and minimization*. (Tech. Rep. No. IRI-DT 9811). Barcelona: Universitat Politècnica de Catalunya.

Received May 11, 1999; accepted July 27, 2000.