4th WSEAS Int. Conf. on COMPUTATIONAL INTELLIGENCE, MAN-MACHINE SYSTEMS and CYBERNETICS Miami, Florida, USA, November 17-19, 2005 (pp242-247)

Feed-forward Neural Networks and Minimal Search Space Learning

Roman Neruda
Institute of Computer Science
Academy of Sciences of the Czech Republic
P.O.Box 5, 18207 Prague, Czech Republic

Abstract: A functional equivalence of feed-forward networks has been proposed to reduce the search space of learning algorithms. The description of equivalence classes has been used to introduce a unique parametrization property and consequently the so-called canonical parameterizations as representatives of functional equivalence classes. A novel genetic learning algorithm for RBF networks and perceptrons with one hidden layer that operates only on these parameterizations has been proposed. Experimental results show that our procedure outperforms the standard genetic learning. An important extension of theoretical results demonstrates that our approach is also valid in the case of approximation.

Key–Words: Search space, Feed-forward networks, Genetic algorithms.

1 Introduction

We consider a feed-forward network as a device for computing a real function of several real variables which depend on a set of network parameters (weights). A function realized by the network is referred to as an *input/output (or I/O) function* of the network. The logical question which functions can be approximated by a class of networks of a given type has been answered in recent years. A so called *universal approximation property* (the possibility to approximate any reasonable, e.g. continuous, function arbitrarily well) was examined. It has been proven that many common network architectures, including multilayer perceptrons and RBF networks which satisfy certain mild conditions on the activation or radial function, posses this property.

Thus, theoretically, for any reasonable function we can find a network of a given type that computes an arbitrarily close approximation of this function as its I/O function. It means that during the learning phase, the parameters of the network can be assigned in such a way that the desired function (usually described by a set of examples of input/output values) is approximated with arbitrary precision. In practice this typically requires to solve a non-linear optimization problem in the high-dimensional space of network parameters. This motivates one to search for possibilities to simplify this task. One of the approaches is to reduce the search space by identifying the classes of functionally equivalent networks and by selecting a single representative of each class. An algorithm which is able to restrict the learning only to these representatives operates on much smaller search space and thus may perform faster.

Hecht-Nielsen [2] pointed out that characterization of functionally equivalent network parameterizations might speed up some learning algorithms. Several authors studied functionally equivalent weight vectors for one hidden layer perceptron-type networks with various activation functions ([1], [4], [8]). In [5], [6] we have characterized the form of functional equivalence of Radial Basis Function networks with Gaussian radial function. The notion of unique parameterization property has been proposed by Kůrková to summarize results common to important non-trivial network architectures.

A very important question is whether the preceding results are also valid in the approximation case. In practice we usually deal with inexact values, either because of the nature of the problem, or because of the limited precision of machine computations. The "exact" formulation of results from section 2 does not suffice here. Fortunately, the original results presented in section 2 assure that the functional equivalence and unique parameterization property is still valid when we consider the approximation of functions.

Based on these results we are able to construct a factorization of the whole parameter space into the classes of functionally equivalent network parameterizations. A unique representative—called *canonical parameterization*—is selected from each equivalence

class by defining a certain condition that the network parameters have to satisfy. The set of all canonical parameterizations corresponds to minimal search set introduced by Hecht-Nielsen and in our case it forms a $1/k!$ part of the whole parameter space.

In order to take advantage of the description of canonical parameterization we propose a new learning algorithm of the genetic type. The basic idea is that the standard genetic operators are redefined such that they preserve the canonical property of parameterizations. This way, the algorithm is able to search only among canonical parameterizations, which results in much smaller search space and consequently faster learning.

To verify our theoretical considerations we have tested the proposed algorithm on several problems. The results for RBF networks confirm the expecting speedup of canonical genetic learning which is about 100% in comparison to the standard GA.

2 Theoretical results

From now on we consider two types of feed-forward network architectures. By an *RBF network* we mean the feed-forward network with one hidden layer containing radial-basis-function (RBF) units with a radial function $\psi : \mathcal{R}_+ \to \mathcal{R}$ and a metrics ρ on $\mathcal{R}^n$ (n is the number of input units) and with a single linear output unit. Such a network computes the function:

$$f(\mathbf{x}) = \sum_{i=1}^{k} w_i \psi \left(\frac{\rho(\mathbf{x}, \mathbf{c}_i)}{b_i} \right). \qquad (1)$$

Here, the *perceptron network* means a feed-forward network with n inputs, one hidden layer containing perceptron units and one linear output unit. This network computes the function $f : \mathcal{R}^n \to \mathcal{R}$ of the form:

$$f(\mathbf{x}) = \sum_{i=1}^{k} w_i \psi(\mathbf{v}_i \mathbf{x} + b_i), \qquad (2)$$

where $k \in \mathcal{N}$ is the number of hidden units, $w_i, b_i \in \mathcal{R}$, $\mathbf{v}_i \in \mathcal{R}^n$ and $\psi : \mathcal{R} \to \mathcal{R}$ is an activation function.

Definition 1 *A* radial-basis-function network parameterization with respect to (ψ, n, ρ) *is a sequence* $\mathbf{P} = (w_i, \mathbf{c}_i, b_i;\ i = 1, \ldots, k)$*, where k is the number of hidden units and for the i-th hidden unit the vector $\mathbf{c}_i \in \mathcal{R}^n$ describes the centroid while the real numbers b_i and w_i are widths and output weights, respectively (see(1)). If additionally, for every $i \in \{1, \ldots, k\}$ $w_i \neq 0$, and for every $i, j \in \{1, \ldots, k\}$, such that $i \neq j$ either $\mathbf{c}_i \neq \mathbf{c}_j$ or $b_i \neq b_j$, it is called* a reduced parameterization.

Definition 2 *Similarly, a* perceptron network parameterization with respect to (ψ, n) *is a sequence* $\mathbf{Q} = (w_i, \mathbf{v}_i, b_i;\ i = 1, \ldots, k)$ *with the meaning of symbols described by (2). Additionally, if for every $i \in \{1, \ldots, k\}$ $w_i \neq 0$, and for every $i, j \in \{1, \ldots, k\}$ $i \neq j$ implies that either $\mathbf{v}_i \neq \mathbf{v}_j$, or $b_i \neq b_j$ and there exists at least one i such that $\mathbf{v}_i = \mathbf{0}$, it is called* a reduced parameterization.

It is clear that a parameterization $\mathbf{P}$ (or $\mathbf{Q}$) determines a unique I/O function of an RBF network according to the formula (1) (or a perceptron network according to (2)).

Definition 3 *Two network parameterizations $\mathbf{P}$ and $\mathbf{P}'$ are* functionally equivalent *if they determine the same input/output function.*

Definition 4 *Two network parameterizations are called* interchange equivalent*, if $k = k'$ and there exists a permutation π of the set $\{1, \ldots, k\}$, such that for each $i \in \{1, \ldots, k\}$ $w_i = w'_{\pi(i)}$ and $b_i = b'_{\pi(i)}$ and $\mathbf{c}_i = \mathbf{c}'_{\pi(i)}$ for RBF network parameterizations, or $\mathbf{v}_i = \mathbf{v}'_{\pi(i)}$ for perceptron network parameterizations.*

We are interested in relationship between the functional equivalence and interchange equivalence. Clearly the later implies the former, so it is the non-trivial reverse implication that is in our focus.

Definition 5 *Let $n \in \mathcal{N}$. Function ψ has a* unique parameterization property *with respect to n, if for every two reduced parameterizations of perceptron networks w.r.t. (ψ, n) (or RBF networks w.r.t. (ψ, n, ρ)) functional equivalence implies interchange equivalence.*

The most general characterization of functions satisfying the unique parameterization property of perceptron networks is due to [4].

Theorem 6 *Let ψ be bounded, non-constant and asymptotically constant activation function, $n \in \mathcal{N}$. Then ψ has a unique parameterization property of perceptron networks with respect to n, if and only if it is neither self-affine, nor affinely recursive.*

Many popular activation functions, including logistic sigmoid or Gaussian, are not affinely recursive. On the contrary, polynomials are affinely recursive, so they do not posses the unique parameterization property. Self-affinity requires a finer analysis which is described in the original paper. Roughly speaking, trivial parameter changes such as sign flips also have to be taken into account.

In the case of RBF networks, the standard choice of a radial function is Gaussian and the most popular metrics are those induced by various inner products (such as Euclidean), or the maximum metrics. Our previous results [5] show that the unique parameterization property is satisfied in these cases.

Theorem 7 *Let n be a positive integer, ρ metrics on $\mathcal{R}^n$ induced by an inner product, or a maximum metrics. Then $\gamma(t) = \exp(-t^2)$ has a unique parameterization property of a corresponding RBF network.*

Preceding theorems enables to describe a canonical representation of a network computing a particular function easily. One of the possible choices is to impose a condition on a parameterization that weight vectors corresponding to hidden units are increasing in a lexicographic ordering on a parameterization. Represent a parameterization $\{w_i, k_i, \mathbf{c}_i; i = 1, \ldots, k\}$ or $\{w_i, k_i, \mathbf{v}_i; i = 1, \ldots, k\}$ as a vector $\mathbf{p} = \{\mathbf{p}_i, \ldots, \mathbf{p}_k\} \in \mathcal{R}^{k(n+2)}$, where $\mathbf{p}_i = \{w_i, b_i, c_{i1}, \ldots, c_{in}\} \in \mathcal{R}^{n+2}$, or $\mathbf{p}_i = \{w_i, b_i, v_{i1}, \ldots, v_{in}\} \in \mathcal{R}^{n+2}$, are weight vectors corresponding to the i-th hidden RBF, or perceptron, unit. Let $\prec$ denotes the lexicographic ordering on $\mathcal{R}^{n+2}$, i.e. for $\mathbf{p}, \mathbf{q} \in \mathcal{R}^{d+2}$ $\mathbf{p} \prec \mathbf{q}$ if there exists an index $m \in \{1, \ldots, d+2\}$ such that $p_j = q_j$ for $j < m$, and $p_m < q_m$.

Definition 8 *We call a network parameterization* $\mathbf{P}$ canonical *if* $\mathbf{p}_1 \prec \mathbf{p}_2 \prec \ldots \prec \mathbf{p}_k$.

In this terminology, theorems 6 and 7 guarantee that for every network parameterization a canonical parameterization determining the same input-output function exists. Thus, the set of canonical parameterizations corresponds to a minimal search set—weight space subset containing exactly one representative of each class of functionally equivalent weight vectors—proposed in [2].

An important extension of our previous results concerning the approximate version of functional equivalence exists. The strict functional equivalence, as was introduced, may not be a sufficient answer in many problems. Especially when dealing with approximation problems we are interested in the case where two functions realized by networks are not exactly the same but their difference is small. This case is also of a big importance in practical problems when we, by nature, operate with inexact values.

3 Genetic learning

In order to take advantage of the previous results a learning algorithm that can operate only on canonical parameterizations is needed. Neither back propagation nor more complicated three step learning algorithms of RBF networks (cf. [3]) are suitable, since the analytical solution obtained in each step of the iterative process cannot in principle be limited to a certain weight space subset. This is not so with genetic algorithm whose operations can be changed to preserve the property of being canonical.

Genetic algorithm

The core of canonical GA is the same as usual: In the beginning a population of m canonical parameterizations $\mathcal{P}_0 = \{\mathbf{P}_1, \ldots, \mathbf{P}_m\}$ is generated at random. Having population $\mathcal{P}_i$, the successive population $\mathcal{P}_{i+1}$ is generated by means of three basic genetic operations: *reproduction*, *crossover* and *mutation* that are proposed such that they generate only canonical parameterizations.

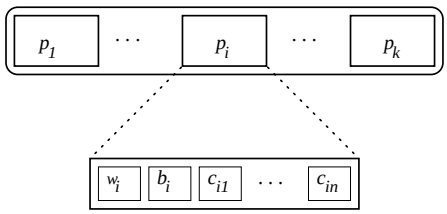

Figure 1: An individual: encoded parameters of a problem

For the overall GA scheme cf. Table 3.

Initial population

To generate the *initial population* of canonical parameterizations at random, one has to preserve the property that for each parameterization $\mathbf{P}$ it holds: $\mathbf{p}_s \prec \mathbf{p}_{s+1}$.

Reproduction

The *reproduction* operator represents a probabilistic selection of parameterization $\mathbf{P_l} \in \mathcal{P}_i$ according to the values of objective function $\mathcal{G}(\mathbf{P_l})$ which is computed by means of the error function (i.e. the sum of distances between the actual and desired output of the network over all of the patterns from the training set).

Thus, we have

$$\mathcal{G}(\mathbf{P}) = C - \mathcal{E} = C - \sum_{j=1}^{z} ||f_{\mathbf{P}}(\mathbf{x}_j) - y_j||^2,$$

```
Genetic learning
var Old, New: Population of N Individuals;
    Father, Mother, Son, Daughter: Individual;
    i: integer;
    p, max: real; begin
(*initialization*)
    for i:= 1 to N do
    begin
            Father:= random Individual;
            p:= objective-function(Father);
            put (Father,p) to Old
    end;
    repeat
(* next population *)
            max:= 0;
            select Father from Old;
            select Mother from Old;
            crossover(Father,Mother,Son,Daughter);
            mutation(Son);
            mutation(Daughter);
            p:= objective-function(Son);
            put (Son,p) to New;
            if p>max then max:= p;
            p:= objective-function(Daughter);
            put (Daughter,p) to New;
            if p>max then max:= p;
    until max<end-criterion
end.
```

Table 1: Scheme of a simple genetic algorithm

where y_j is the desired network output, $f(\mathbf{x}_j)$ represents the actual response of the network when the input $\mathbf{x}_j$ is presented, and C is the maximal error.

For each individual $\mathbf{P}_l$ the value of its objective function $\mathcal{G}(\mathbf{P}_l)$ is computed and then normalized by dividing by the sum of objective function values over all individuals in the population:

$$p_l = \frac{G(\mathbf{P}_l)}{\sum_{r=1}^{m} G(\mathbf{P}_r)}.$$

The number p_l then represents the probability with which the parameterization is selected at random. We use the roulette wheel selection together with the elitist mechanism.

Mutation

The *mutation* operates on two levels—first an element $\mathbf{p}_s$ is chosen randomly as a candidate for mutation. Its neighbors $\mathbf{p}_{s-1}$ and $\mathbf{p}_{s+1}$ then determine the lower and upper border of the range in which the $\mathbf{p}_s$ is changed at random.

Crossover

The *crossover* operator chooses two parameterizations $\mathbf{P} = (\mathbf{p}_1, \ldots, \mathbf{p}_k)$ and $\mathbf{Q} = (\mathbf{q}_1, \ldots, \mathbf{q}_k)$ in $\mathcal{P}_i$ and

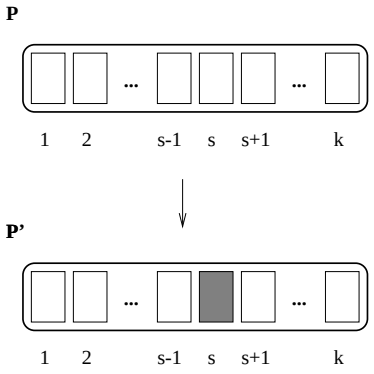

Figure 2: The mutation operator creates $\mathbf{P}' \in \mathcal{P}_{i+1}$ from $\mathbf{P} \in \mathcal{P}_i$.

generates a new offspring $\mathbf{P}' \in \mathcal{P}_{i+1}$. A position s is found at random such that the parameterization $\mathbf{P}' = (\mathbf{p}_1, \ldots, \mathbf{p}_s, \mathbf{q}_{s+1}, \ldots, \mathbf{q}_k)$ still satisfies the condition: $\mathbf{p}_s \prec \mathbf{q}_{s+1}$.

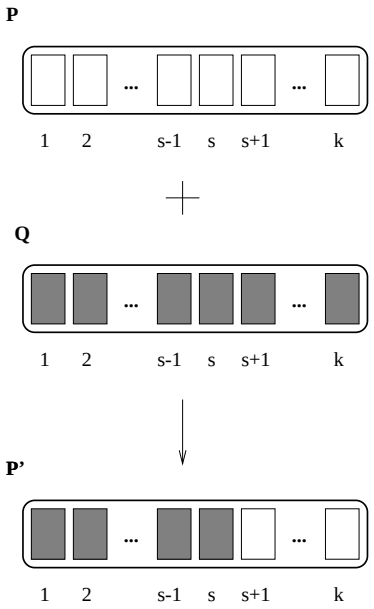

Figure 3: The crossover operator creates $\mathbf{P}' \in \mathcal{P}_{i+1}$ by combining $\mathbf{P}$ and $\mathbf{Q}$ from $\mathcal{P}_i$.

4 Experiments

In the following we describe our experiments testing the performance of canonical and standard learning algorithms of feed-forward networks on two problems: the XOR problem and the approximation of $\sin(x) \cdot \sin(y)$ function. The experiments were made on the cluster of PC with 1.4Ghz Pentium and 1GB RAM running Linux. The software system called Bang has been used to perform these tests.

The first task was a XOR problem defined by four training examples. We used 50 networks in the population and elitist selection for the two best networks. Error values for the first 500 iterations are plotted on Figure 4. Both algorithms were able to successfully learn the given task quite fast. (cf. Table 2); the canonical algorithm was about two times faster in terms of error decrease. Running times of both algorithms were roughly identical— about .4 seconds per 1000 iterations.

The second experiment was an approximation of the function $f(x, y) = \sin(x) \cdot \sin(y)$ given by a 10×10 mesh of points regularly taken from a $[0; 2\pi] \times [0; 2\pi]$ square. Again, both algorithms with 50 networks in population were used with the same elitist rate as in the previous experiment. The learning speed is shown on Table 2 and Figure 5. The performance was similar to the previous experiment: the canonical GA was again about twice faster. The average speed of 1000 iterations was 5 seconds.

XOR		
$\mathcal{E}$	canonical	standard
10^{-1}	50	76
10^{-2}	104	268
10^{-3}	443	705
10^{-4}	934	1553
10^{-5}	1075	1988
10^{-6}	1733	4485
10^{-8}	7623	>10000

Table 2: Number of iterations necessary to reach a particular error threshold for the XOR experiment.

Another set of experiments has been performed to explore the influence of different population and network sizes (i.e. numbers of networks in one population of GA, and number of units in the network). Figures 6, 7, and 8 illustrate the results for the XOR task. It shows that the population size does not play

$\sin(x) \cdot \sin(y)$		
$\mathcal{E}$	canonical	standard
10	177	200
5	313	765
3	489	1043
2	1268	—

Table 3: Number of iterations necessary to reach a particular error threshold for the $\sin(x) \cdot \sin(y)$ experiment

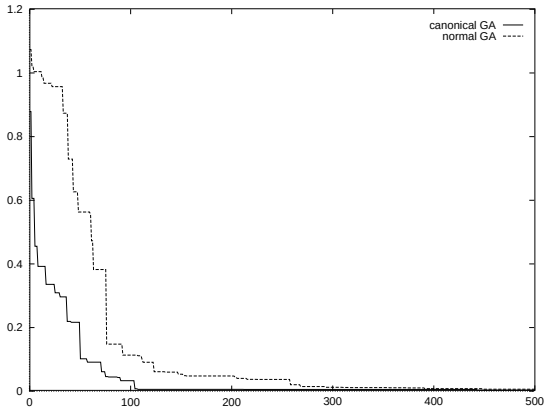

Figure 4: Comparison of error decrease for the XOR experiment.

an important role (as long as it stays within a reasonable range). On the other hand, number of units can improve the approximation in a relevant way.

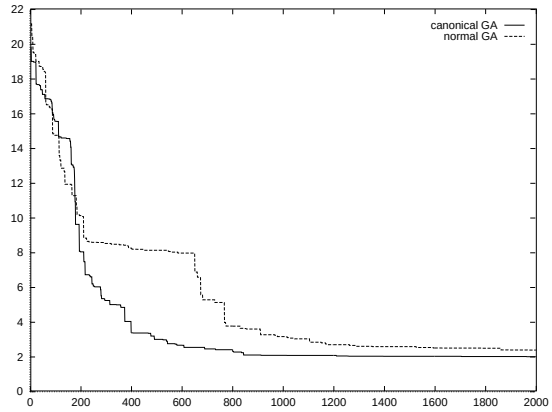

Figure 5: Comparison of error decrease for the $\sin(x) \cdot \sin(y)$ experiment.

5 Conclusions

We have presented results concerning functional equivalence property for the case of Gaussian RBF networks and one hidden layer perceptron networks. Based on these we have proposed the canonical genetic algorithm for learning feed-forward neural networks. The proposed algorithm has been realized and tested for the case of RBF networks with Gaussian units and perceptron networks with logistic sigmoid. It has been shown that for small/middle-size tasks the canonical GA is about twice faster in reaching the same error treshold. Moreover, the canonical GA does not show any relevant increase in time for one iteration

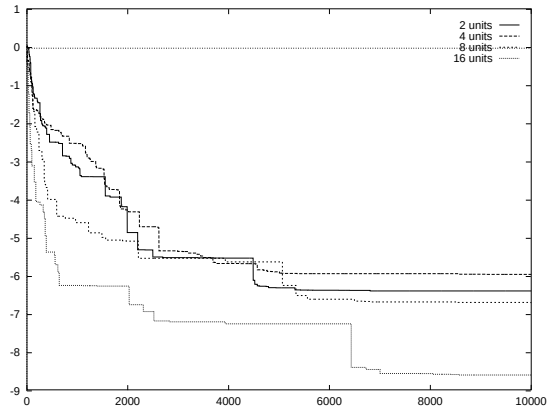

Figure 6: Comparison of error decrease for standard GA with various number of hidden units (y scale represents $\log_{10} \mathcal{E}$).

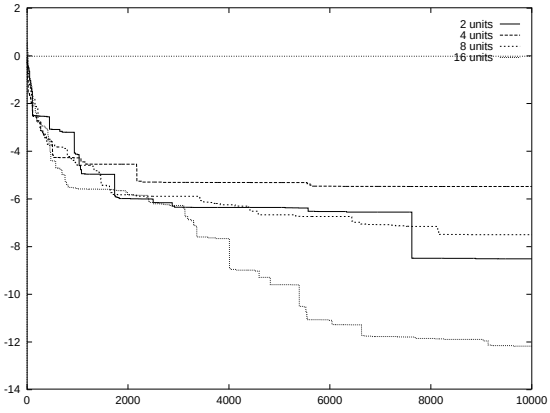

Figure 7: Comparison of error decrease for canonical GA with various number of hidden units (y scale represents $\log_{10} \mathcal{E}$).

in comparison to standard GA. Thus, the twice better times hold also in real time.

An interesting comparison would be against the standard learning algorithms of feed-forward networks, such as back propagation. This is one of the directions of our further work. A thorough comparison of RBF networks learning methods (including genetic, gradient-based and three-step learning), and their performace has been published in [7].

Acknowledgement

This research was supported by the Grant Agency of the Czech Republic under grant no. 201/05/0557.

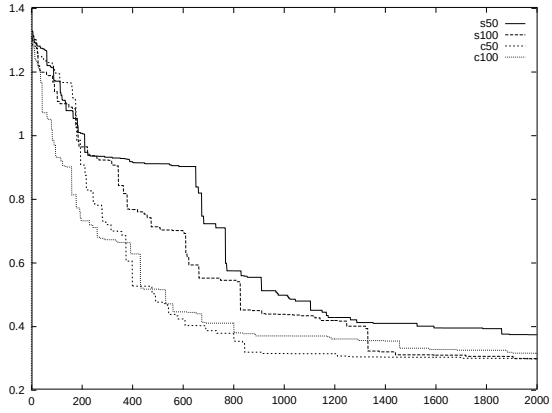

Figure 8: Comparison of error decrease for standard and canonical algorithm with population sizes of 50 and 100 networks (y scale represents $\log_{10} \mathcal{E}$).

References:

[1] F. Albertini and E.D. Sontag. For neural networks, function determines form. *Neural Networks*, 6:975–990, 1993.

[2] R. Hecht-Nielsen. On the algebraic structure of feedforward network weight spaces. In *Advanced Neural Computers*, pages 129–135. Elsevier, 1990.

[3] K. Hlaváčková and R. Neruda. Radial basis function networks. *Neural Network World*, (1):93–101, 1993.

[4] V. Kůrková and P. Kainen. Functionally equivalent feedforward networks. *Neural Computation*, 6:543–558, 1993.

[5] V. Kůrková and R. Neruda. Uniqueness of the functional representations for the Gaussian basis functions. In *Proceedings of the ICANN'94*, pages 474–477, London, 1994. Springer Verlag.

[6] R. Neruda. Functional equivalence and genetic learning of RBF networks. In *Proceedings of the ICANNGA'95*, pages 53–56, Vienna, 1995. Springer Verlag.

[7] Roman Neruda and Petra Kudová. Learning methods for radial basis functions networks. *Future Generation Computer Systems*, 21:1131–1142, 2005.

[8] H.J. Sussmann. Uniqueness of the weights for minimal feedforward nets with a given input-output map. *Neural Networks*, 5(4):589–594, 1992.