# Implementation of Kolmogorov Learning Algorithm for Feedforward Neural Networks

Roman Neruda, Arnošt Štědrý, and Jitka Drkošová*

Institute of Computer Science, Academy of Sciences of the Czech Republic,
P.O. Box 5, 18207 Prague, Czech Republic
roman@cs.cas.cz

**Abstract.** We present a learning algorithm for feedforward neural networks that is based on Kolmogorov theorem concerning composition of $n$-dimensional continuous function from one-dimensional continuous functions. A thorough analysis of the algorithm time complexity is presented together with serial and parallel implementation examples.

## 1 Introduction

In 1957 Kolmogorov [5] has proven a theorem stating that any continuous function of $n$ variables can be exactly represented by superpositions and sums of continuous functions of only one variable. The first, who came with the idea to make use of this result in the neural networks area was Hecht-Nielsen [2]. Kůrková [3] has shown that it is possible to modify the original construction for the case of approximation of functions. Thus, one can use a perceptron network with two hidden layers containing a larger number of units with standard sigmoids to approximate any continuous function with arbitrary precision. In the meantime several stronger universal approximation results has appeared, such as [6] stating that perceptrons with one hidden layer and surprisingly general activation functions are universal approximators.

In the following we review the relevant results and show a learning algorithm based on Sprecher improved version of the proof of Kolmogorov's theorem. We focus on implementation details of the algorithm, in particular we proposed an optimal sequential version and studied its complexity. These results have also lead us to consider ways of suitable parallelization. So far, we have realized one parallel version of the algorithm running on a cluster of workstations.

## 2 Preliminaries

By $\mathcal{R}$ we denote the set of real numbers, $\mathcal{N}$ means the set of positive integers, $\mathcal{I} = [0, 1]$ and thus $\mathcal{I}^n$ is the $n$-dimensional unit cube. By $\mathcal{C}(\mathcal{I}^n)$ we mean a set of all continuous functions defined over $\mathcal{I}^n$.

* This research has been partially supported by GAASCR under grants B1030006 and A2030801, and by GACR under grant 201/99/0092.

V.N. Alexandrov et al. (Eds.): ICCS 2001, LNCS 2074, pp. 986–995, 2001.
© Springer-Verlag Berlin Heidelberg 2001

**Definition 1.** *The sequence* $\{\lambda_k\}$ *is* integrally independent *if* $\sum_p t_p\lambda_p \neq 0$ *for any finite selection of integers* $t_p$ *for which* $\sum_p |t_p| \neq 0$

**Definition 2.** *By* sigmoidal function *we mean any function* $\sigma : \mathcal{R} \to \mathcal{I}$ *with the following limits:* $\lim_{t\to-\infty} \sigma(t) = 0$ $\lim_{t\to\infty} \sigma(t) = 1$

**Definition 3.** *The set of* staircase-like functions of a type $\sigma$ *is defined as: the set of all functions* $f(x)$ *of the form* $f(x) = \sum_{i=1}^{k} a_i\sigma(b_ix + c_i)$, *and denoted as* $\mathcal{S}(\sigma)$.

**Definition 4.** *A function* $\omega_f : (0, \infty) \to \mathcal{R}$ *is called a* modulus of continuity *of a function* $f : \mathcal{I}^n \to \mathcal{R}$ *if* $\omega_f(\delta) = sup\{|f(x_1, \ldots, x_n) - f(y_1, \ldots, y_n)|; (x_1, \ldots, x_n), (y_1, \ldots, y_n) \in \mathcal{I}^n$ *with* $|x_p - y_p| < \delta$ *for every* $p = 1, \ldots, n\}$.

In the following we always consider (neural) network to be a device computing certain function dependent on its parameters. Without the loss of generality we limit ourselves only to networks with $n$ inputs with values taken from $\mathcal{I}$ and one real output. Thus we consider functions $f : \mathcal{I}^n \to \mathcal{R}$.

Moreover, we talk about two types of network architectures. One is the usual multilayer perceptron with two hidden layers. Perceptron units in each layer compute the usual affine combination of its inputs and weights (and bias) and then apply a sigmoidal activation function. An example can be the most common perceptron with logistic sigmoid. The (single) output unit computes just the linear combination.

The second type of network is a more general feedforward network where the units in different layers can compute different activation functions that can be more complicated than the logistic sigmoid. The description of a concrete form of such a network is subject to the section 4.

## 3 Kolmogorov Theorem

The original Kolmogorov result shows that every continuous function defined on $n$-dimensional unit cube can be represented by superpositions and sums of one-dimensional continuous functions.

**Theorem 1 (Kolmogorov).** *For each integer* $n \geq 2$ *there are* $n(2n + 1)$ *continuous monotonically increasing functions* $\psi_{pq}$ *with the following property: For every real-valued continuous function* $f : \mathcal{I}^n \to \mathcal{R}$ *there are continuous functions* $\phi_q$ *such that*

$$f(x_1, \ldots, x_n) = \sum_{q=0}^{2n} \phi_q \left[ \sum_{p=1}^{n} \psi_{pq}(x_p) \right]. \tag{1}$$

Further improvements by Sprecher provide a form that is more suitable for computational algorithm. Namely, the set of functions $\psi_{pq}$ is replaced by shifts of a fixed function $\psi$ which is moreover independent on a dimension. The overall quite complicated structure is further simplified by suitable parameterizations and making use of constants such as $\lambda_p, \beta$, etc.

**Theorem 2 (Sprecher).** *Let $\{\lambda_k\}$ be a sequence of positive integrally independent numbers. There exists a continuous monotonically increasing function $\psi : \mathcal{R}^+ \to \mathcal{R}^+$ having the following property: For every real-valued continuous function $f : \mathcal{I}^n \to \mathcal{R}$ with $n \geq 2$ there are continuous functions $\phi$ and a constant $\beta$ such that:*

$$\xi(\mathbf{x}_q) = \sum_{p=1}^{n} \lambda_p \psi(x_p + q\beta) \tag{2}$$

$$f(\mathbf{x}) = \sum_{q=0}^{2n} \phi_q \circ \xi(\mathbf{x}_q) \tag{3}$$

Another result important for computational realization is due to Kůrková who has shown that both inner and outer functions $\psi$ and $\phi$ can be approximated by staircase-like functions (cf. Definition 3) with arbitrary precision. Therefore, standard perceptron networks with sigmoidal activation functions can, in principle, be used in this approach. The second theorem of hers provides the estimate of units needed for approximation w.r.t. the given precision and the modulus of continuity of the approximated function.

**Theorem 3 (Kůrková).** *Let $n \in \mathcal{N}$ with $n \geq 2$, $\sigma : \mathcal{R} \to \mathcal{I}$ be a sigmoidal function, $f \in \mathcal{C}(\mathcal{I}^n)$, and $\varepsilon$ be a positive real number. Then there exists $k \in \mathcal{N}$ and functions $\phi_i$, $\psi_{pi} \in \mathcal{S}(\sigma)$ such that:*

$$|f(x_1, \ldots, x_n) - \sum_{i=1}^{k} \phi_i \left( \sum_{p=1}^{n} \psi_{pi} x_p \right)| \leq \varepsilon \tag{4}$$

*for every $(x_1, \ldots, x_n) \in \mathcal{I}^n$.*

**Theorem 4 (Kůrková).** *Let $n \in \mathcal{N}$ with $n \geq 2$, $\sigma : \mathcal{R} \to \mathcal{I}$ be a sigmoidal function, $f \in \mathcal{C}(\mathcal{I}^n)$, and $\varepsilon$ be a positive real number. Then for every $m \in \mathcal{N}$ such that $m \geq 2n+1$ and $n/(m-n)+v < \varepsilon/||f||$ and $\omega_f(1/m) < v(m-n)/(2m-3n)$ for some positive real $v$, $f$ can be approximated with an accuracy $\varepsilon$ by a perceptron type network with two hidden layers, containing $nm(m+1)$ units in the first hidden layer and $m^2(m+1)^n$ units in the second one, with an activation function $\sigma$.*

## 4 Algorithm Proposal

Sprecher sketched an algorithm based on Theorem 2 that also takes into account Theorem 4 by Kůrková. Here we present our modified and improved version that addresses crucial computational issues.

The core of the algorithm consists of four steps:

For each iteration $r$:

1. Construct the mesh $\mathcal{Q}^n$ of rational points $\mathbf{d}_k$ dissecting the unit cube (cf. (5)).

2. Construct the functions $\xi$ in the points $\mathbf{d}_k^q$ (see (6) and (11)).
3. Create sigmoidal steps $\theta$ described in (10).
4. Compile the outer functions $\phi_q^j$ (see (9)) based on $\theta$ and previous approximation error $e_r$.
5. Construct the $r$-th approximation $f_r$ of original function $f$ according to (14), and the $r$-th approximation error $e_r$ according to (13).

### 4.1 The Support Set $\mathcal{Q}$

Take integers $m \geq 2n$ and $\gamma \geq m+2$ where $n$ is the input dimension. Consider a set of rational numbers $\mathcal{Q} = \left\{ d_k = \sum_{s=1}^{k} i_s \gamma^{-s}; i_s \in \{0, 1, \ldots, \gamma - 1\}, k \in \mathcal{N} \right\}$. Elements of $\mathcal{Q}$ are used as coordinates of $n$-dimensional mesh

$$\mathcal{Q}^n = \{ \mathbf{d}_k = (d_{k1}, \ldots, d_{kn}); d_{kj} \in \mathcal{Q}, j = 1, \ldots, n \}. \tag{5}$$

Note that the number $k$ determines the precision of the dissection.

For $q = 0, 1, \ldots, m$ we construct numbers $\mathbf{d}_k^q \in \mathcal{Q}^n$ whose coordinates are determined by the expression

$$d_{kp}^q = d_{kp} + q \sum_{s=1}^{k} \gamma^{-s} \tag{6}$$

Obviously $d_{kp}^q \in \mathcal{Q}$ for $p = 1, 2, \ldots, n$. We will make use of $\mathbf{d}_k^q$ in the definition of functions $\xi$.

### 4.2 The Inner Function $\psi$

The function $\psi : \mathcal{Q} \to \mathcal{I}$ is then defined with the help of several additional definitions. For the convenience, we follow [11] in our notation.

$$\psi(d_k) = \sum_{s=1}^{k} \tilde{i_s} 2^{-m_s} \gamma^{-\rho(s-m_s)}, \tag{7}$$

$$\rho(z) = \frac{n^z - 1}{n - 1},$$

$$m_s = \langle i_s \rangle \left( 1 + \sum_{l=1}^{s-1} [i_l] \cdot \ldots \cdot [i_{s-1}] \right), \tag{8}$$

$$\tilde{i_s} = i_s - (\gamma - 2)\langle i_s \rangle.$$

Let $[i_1] = \langle i_1 \rangle = 1$ and for $s \geq 2$ let $[i_s]$ and $\langle i_s \rangle$ be defined as:

$$[i_s] = \begin{cases} 0 \text{ for } i_s = 0, 1, \ldots, \gamma - 3 \\ 1 \text{ for } i_s = \gamma - 2, \gamma - 1 \end{cases}$$

$$\langle i_s \rangle = \begin{cases} 0 \text{ for } i_s = 0, 1, \ldots, \gamma - 2 \\ 1 \text{ for } i_s = \gamma - 1 \end{cases}$$

Figure 1 illustrates the values of $\psi$ for $k = 1, 2, 3, 4; n = 2; \gamma = 6$.

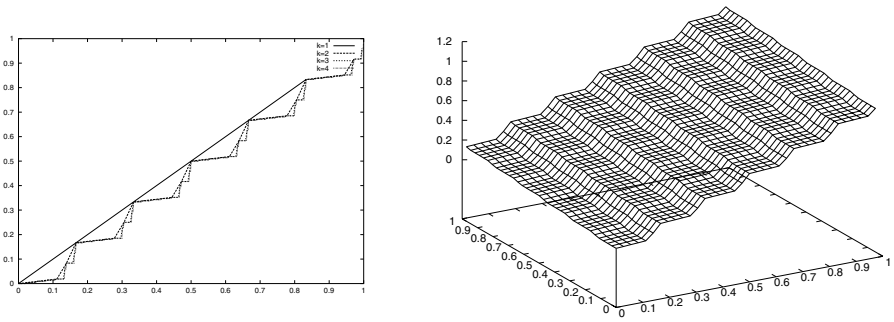


**Fig. 1.** a) Values of $\psi(d_k)$ for various $k$. b) Values of $\xi(\mathbf{d}_k)$ for $k = 2$.

### 4.3 The Outer Functions $\phi$

The functions $\phi_q$ in equation (3) are constructed iteratively as functions $\phi_q(y_q) = \lim_{r\to\infty} \sum_{j=1}^{r} \phi_q^j(y_q)$. Each function $\phi_q^j(y_q)$ is determined by $e_{j-1}$ at points from the set $\mathcal{Q}^n$. The construction is described in the following.

For $q = 0, 1, \ldots, m$ and $j = 1, \ldots, r$ we compute:

$$\phi_q^j \circ \xi(\mathbf{x}_q) = \frac{1}{m+1} \sum_{\mathbf{d}_k^q} e_{j-1}(\mathbf{d}_k)\ \theta(\mathbf{d}_k^q; \xi(\mathbf{x}_q)), \tag{9}$$

where $\mathbf{d}_k \in \mathcal{Q}$.

The real-valued function $\theta(\mathbf{d}_k; \xi(\mathbf{x}_q))$ defined for a fixed point $\mathbf{d}_k^q \in \mathcal{Q}^n$. The definition is based on a given sigmoidal function $\sigma$:

$$\begin{aligned} \theta(\mathbf{d}_k^q; y_q) = {} & \sigma(\gamma^{\beta(k+1)}(y_q - \xi(\mathbf{d}_k^q)) + 1) \\ & - \sigma(\gamma^{\beta(k+1)}(y_q - \xi(\mathbf{d}_k^q) - (\gamma - 2)b_k) \end{aligned} \tag{10}$$

where $y_q \in \mathcal{R}$, and $b_k$ is a real number defined as follows:

$$b_k = \sum_{s=k+1}^{\infty} \gamma^{-\rho(s)} \sum_{p=1}^{n} \lambda_p.$$

The functions $\xi(\mathbf{d}_k^q)$ are expressed by equation

$$\xi(\mathbf{d}_k^q) = \sum_{p=1}^{n} \lambda_p \psi(d_{kp}^q) \tag{11}$$

where $\psi(d_{kp}^q)$ are from (7), and coefficients $\lambda_p$ are defined as follows.

Let $\lambda_1 = 1$ and for $p > 1$ let

$$\lambda_p = \sum_{s=1}^{\infty} \gamma^{-(p-1)\rho(s)} \tag{12}$$

Figure 1 shows values of $\xi(\mathbf{d}_k)$ for $k = 2$.

### 4.4 Iteration Step

Let $f$ is a known continuous function, $e_0 \equiv f$. The $r$-th approximation error function $e_r$ to $f$ is computed iteratively for $r = 1, 2, \ldots$

$$e_r(\mathbf{x}) = e_{r-1}(\mathbf{x}) - \sum_{q=0}^{m} \phi_q^r \circ \xi(\mathbf{x}_q), \tag{13}$$

where $\mathbf{x} \in \mathcal{I}^n$, $\mathbf{x}_q = (x_1 + q\beta, \ldots, x_n + q\beta)$, and $\beta = \gamma(\gamma - 1)^{-1}$.

The $r$-th approximation $f_r$ to $f$ is then given by:

$$f_r(\mathbf{x}) = \sum_{j=1}^{r} \sum_{q=0}^{m} \phi_q^j \circ \xi(\mathbf{x}_q). \tag{14}$$

It was shown in [11] that $f_r \to f$ for $r \to \infty$.

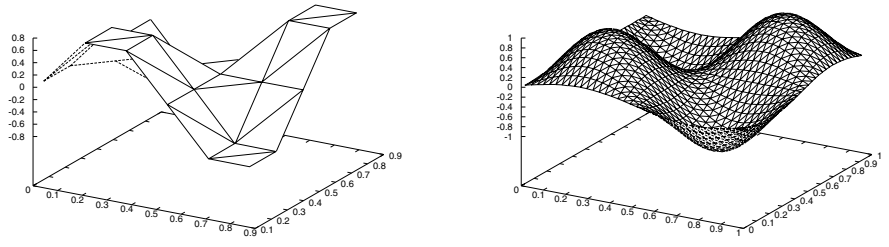

**Fig. 2.** Approximation of $\sin(6.28x) \cdot \sin(6.28y)$ for $k = 1$ and $k = 2$.

## 5 Time Complexity

In the following time complexity analysis we consider our practical implementation of the above described algorithm, which introduces few further simplifications. First, the infinite sums are replaced by sums up to the sufficiently big constant $K$. We also keep the number $k(r)$ fixed to constant $k$ during the iteration loop. Also note that all computations of $f$ are performed on numbers taken from the dense support set $\mathcal{Q}$.

The time complexity is derived with respect to the following basic operations: For time of the floating point multiplication and division the symbol $t_m$ is used, while for the time of floating point addition and subtraction $t_a$ is used.

In general, we first provide fine-grain analysis in the following lemmas, which is then summarized in the theorem 5 as an estimation of the total amount of computational time for one iteration step.

The following lemma summarizes the time needed for computing several terms that are independent of the iteration loop and can be performed in advance.

**Lemma 1.** *The times $T_\lambda, T_b, T_\gamma, T_{e_0}$ needed for computations of $\lambda_p, b_k, \gamma^{\rho(k+1)}$, $e_0$ are expressed by the following formulae:*

1. $T_\lambda = (2+K)t_a + [p+2+2K+(n^K-1)/(n-1)]t_m$,
2. $T_b = n \times T_\lambda + [2(K-k)+(n^K-n^k)/(n-1)]t_m + (K-k)t_a$,
3. $T_\gamma = (k+1+n^k)t_m$,
4. $T_{e_0} = \gamma^{nk} \times (2kt_m + (k-1)t_a)$.

*Proof.* It is easy to derive these results by examining the corresponding definitions.

1. We compute $\lambda_p = \sum_{s=1}^{\infty} \gamma^{-(p-1)\rho(s)}$. The value $\gamma^{1-p}$ is computed and saved. If we suppose that $\rho(s)$ is computed using $\rho(s-1)$ then it costs $2t_m + 1t_a$. Any entry in the sum consumes $(1+(\rho(s)-\rho(s-1))t_m$. The total number of operations is $2t_a + (p+2)t_m + \sum_{s=1}^{K}[(2+n^{s-1})t_m + t_a]$.
2. The term $b_k = \sum_{s=k+1}^{\infty} \gamma^{-\rho(s)} \sum_{p=1}^{n} \lambda_p$. To obtain the entries of the outer sum consumes $\sum_{s=k+1}^{K}[(2+n^{s-1})t_m + t_a]$ and the time for $\lambda_p$.
3. Obvious.
4. The initial error function $e_0$ is set to the function $f$, while only values $f(\mathbf{d}_k)$ are needed. It means that the values of $\mathbf{d}_k$ must be expressed in the form suitable as an input of $f$ (decadic form). Since they are originally stored via their ordinal values, we need $(2kt_m + (k-1)t_a)$ computations for any $\boldsymbol{d}_k$.

In order to compute time $T_S$ of one (serial) iteration of the algorithm, we need to express times for partial steps that are thoroughly described in section 4. The most inner terms in the computation are numbers $\mathbf{d}_k^q$ that are copmuted by means of their coordinates $d_{kp}^q = d_{kp} + q\sum_{s=1}^{k}\gamma^{-s}$.

**Lemma 2.** *Time $T_{\mathbf{d}_k^q}$ for computing $\mathbf{d}_k^q$ is $T_{\mathbf{d}_k^q} = (k+n)t_a + (k+1)t_m$.*

*Proof.* To compute the sum costs $kt_a + kt_m$. Then, $t_m + nt_a$ operations are needed to complete the computation.

Quite complicated algorithm for computing $\Psi$ is described in 4.2. Its complexity is estimated as follows.

**Lemma 3.** *For any coordinate $d_{kp}$ the function $\Psi$ requires $T_\Psi = (1/6k^3 + 3k^2 + 10k + 1)t_a + [(n^k-1)/(n-1) + (k^2+k)/2 + 1]t_m$.*

*Proof.* The definition of $m_s$ (cf. 8) plays the key role in computation of $\Psi(d_{kp}^q)$. The upper bound for the expression of this value is $1/2(s^2+3s+4)t_a + st_a$. To simplify the estimation we consider the worst case $s - m_s = s$ as an input into $\rho(s - m_s)$. This computation then requires $2t_a + st_m$. To estimate the time consumed to express the function $\Psi(d_{kp})$ means to add the entries $1/2(s^2+5s+14)t_a + (s + (n^s-1)/(n-1))t_m$ together.

The time $T_\xi$ needed for function $\xi(d_k^q)$ is straightforwardly computed by means of the three above mentioned quantities.

**Lemma 4.** *Function* $\xi(d_k^q) = \sum_{p=1}^{n} \lambda_p \Psi(d_{kp}^q)$ *consumes for any* $d_k^q$ *the amount* $T_\xi = n \times (T_\lambda + T_\Psi + T_{d_k^q})$

Next we analyze the time $T_\theta$ necessary to compute $\theta$.

**Lemma 5.** *To evaluate the function* $\theta$ *costs* $T_\theta = [4K - k + 6 + n^k + (2n^K - n^k)/(n-1) + n]t_m + [2K - k + 7]t_a + T_\sigma$.

*Proof.* $\theta(d_k^q, y_q) = \sigma(\gamma^{\rho(k+1)}(y_q - \xi(d_k^q)) + 1) - \sigma(\gamma^{\rho(k+1)}(y_q - \xi(d_k^q)) - (\gamma - 2)b_k))$ and according to our assumptions $y_q = \xi(d_k^q)$. The time $T_\theta$ is then the sum $T_\gamma + T_b + T_\sigma + 5t_a + 3t_m$. Using lemma 1 we directly obtain the result.

**Lemma 6.** *Time needed to compute the value of the outer function* $\Phi$ *for one particular* $d_k^q$ *is the following.* $T_\Phi = [n^2 + n^k + 3n + 7 - k + (2n+4)K + (n^K(n+2) - n^k - n)/(n-1)]t_m + [(2+n)K + 2n - k + 7]t_a + T_\Psi + T_{d_k^q} + T_\sigma$

*Proof.* $T_\Phi = T_\xi + T_\theta$. Separate the operations that are computed just once and do not depend on the iteration loop and use lemmas 4 and 5, which gives the estimation.

**Lemma 7.** *The time for evaluation of the error function* $e_r$ *in one particular* $d_k$ *is* $T_e = (m+1)t_a$.

*Proof.* If we assume that the values of $e_{r-1}(d_k)$ are saved and $\Phi_q^r$ have been already computed, then the computation of $e_r(d_k)$ costs $(m+1)t_a$.

**Lemma 8.** *To generate the* $r-th$ *approximation to the function* $f$ *in one particular value consumes* $T_f = mt_a$.

*Proof.* The values of $f_r$ are recurrently computed by means previous iteration $f_{r-1}$ and already computed $\Phi_q^r$, thus only $m$ additions is needed.

Our analysis of the serial case is summarized in the following theorem.

**Theorem 5.** *The computational requirements in one iteration of the sequential algorithm is estimated as follows.*

$$T_S = \mathcal{O}\left(m\ n\ \gamma^{nk}(n^k + k^3)\right). \tag{15}$$

*Proof.* Express $T_S$ according to the algorithm description:

$$T_S = \gamma^{nk} \left[ m T_\Phi + T_e + T_f \right].$$

The partial terms used have been specified by the preceeding lemmas. We do not take into account terms that can be pre-computed (cf. Lemma 1). Also, a particular sigmoidal function — step sigmoid — is considered, which can easily be computed by means of two comparisons. We have sacrificed the tightness of the bound to tractability of the result, and consider $t_a = t_m$ (cf. Discussion).

$$T_S \approx m\ n\ \gamma^{nk} \left[ k^3 t_a + (n^k + k^2) t_m \right],$$

which proves (15).

## 6 Discussion

The overall time of the serial algorithm consists of initial computations of the constant terms which take $T_\lambda + T_b + T_\gamma + T_{e_0} + T_\lambda$ and $r \times T_S$. The assumption that the additive and multiplicative operations take approximately the same time is not a strong simplification if one considers current microporcessors such as Pentiums. Note also that the complexity analysis considers quite optimal sequential realization of the algorithm described in section 4. In particular, some formulae have been reformulated for speedup. Also, the mesh $\mathcal{Q}$ is represented in a way which allows efficient access, and we cache once computed values wherever possible.

From the point of view of parallel implementation, there is one straightforward approach employing $m + 1$ processors for computations of $\Phi_0^r, \ldots, \Phi_m^r$, which are mutually independent. This can reduce the $T_S$ by a factor of $m$ on one hand, but it requires additional communication to complete each iteration by exchanging the computed values. Although a finer analysis of this aproach has not been performed yet, our first experiments show that the communication requirements are tiny compared to the total computation time, especially for problems of higher dimension. Our parallel implementation has been done on a cluster of workstations in the PVM environment. Figure 3 shows a typical behaviour of the program running on six machines for two iterations of the algorithm.

In the future work we plan to focus on the exact analysis of the parallel implementation, and on employing other means of parallelization, such as clever partitioning of the input space.

## References

1. F. Girosi and T. Poggio. Representation properties of networks: Kolmogorov's theorem is irrelevant. *Neural Computation*, 1:461–465, 1989.
2. Robert Hecht-Nielsen. Kolmogorov's mapping neural network existence theorem. In *Procceedings of the International Conference on Neural Networks*, pages 11–14, New York, 1987. IEEE Press.


**Fig. 3.** Illustration of a typical parallel behaviour of the algorithm running on cluster of Pentium workstations under PVM.


3. Věra Kůrková. Kolmogorov's theorem is relevant. *Neural Computation*, 3, 1991.
4. Věra Kůrková. Kolmogorov's theorem and multilayer neural networks. *Neural Networks*, 5:501–506, 1992.
5. A. N. Kolmogorov. On the representation of continuous function of many variables by superpositions of continuous functions of one variable and addition. *Doklady Akademii Nauk USSR*, 114(5):953–956, 1957.
6. M. Leshno, V. Lin, A. Pinkus, and S. Shocken. Multilayer feedforward networks with a non-polynomial activation function can approximate any function. *Neural Networks*, (6):861–867, 1993.
7. G.G Lorentz. *Approximation of functions*. Halt, Reinhart and Winston, New York, 1966.
8. David A. Sprecher. On the structure of continuous functions of several variables. *Transactions of the American Mathematical Society*, 115:340–355, 1965.
9. David A. Sprecher. A universal mapping for Kolmogorov's superposition theorem. *Neural Networks*, 6:1089–1094, 1993.
10. David A. Sprecher. A numerical construction of a universal function for Kolmogorov's superpositions. *Neural Network World*, 7(4):711–718, 1996.
11. David A. Sprecher. A numerical implementation of Kolmogorov's superpositions II. *Neural Networks*, 10(3):447–457, 1997.