Variants of Learning Algorithm Based on Kolmogorov Theorem

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. A thorough analysis of theoretical and computational properties of kolmogorov learning algorithm for feedforward neural networks lead us to proposal of efficient sequential and parallel implementation. A novel approach to parallelization which combines our previous results in order to achieve higher parallel speedup is presented.

1 Introduction

In 1957 Kolmogorov [4] 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 [1]. Kůrková [2] 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 [5] we have proposed a serial implementation of the Kolmogorov algorithm. Both the theoretical results and practical experiments have shown that the time complexity is relatively big. On the other hand, the algorithm has been suitable for parallelization. In [6] our first parallel implementation has been proposed. This quite efficient parallel algorithm with very low communication is suitable to be implemented on the cluster of workstation architecture. Its drawback is a limited number of processors which can be employed.

In this contribution we propose a different parallel implementation that is based on theoretical properties of the algorithm. This implementation can achieve higher degree of parallelism while still maintaining good efficiency. Communicational demands of the processors are very small in comparison to computations. Moreover, this implementation can make use of the efficient serial implementation we have developed earlier. The time complexity analysis of this algorithm is provided based on the BSP (Bulk Synchronous Parallel) model of a realistic distributed memory parallel machine. Our first experiments have been done in the PVM (Parallel Virtual Machine) environment.

* This research has been partially supported by Grant Agency of the Academy of Sciences of the Czech Republic under grants B1030006 and A2030801.

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.

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 $\sigma : \mathcal{R} \rightarrow \mathcal{I}$ with the following limits: $\lim_{t \rightarrow -\infty} \sigma(t) = 0 \quad \lim_{t \rightarrow \infty} \sigma(t) = 1$

Definition 3. A function $\omega_f : (0, \infty) \rightarrow \mathcal{R}$ is called a modulus of continuity of a function $f : \mathcal{I}^n \rightarrow \mathcal{R}$ if $\omega_f(\delta) = \sup\{|f(x_1, \dots, x_n) - f(y_1, \dots, y_n)|; (x_1, \dots, x_n), (y_1, \dots, y_n) \in \mathcal{I}^n \text{ with } |x_p - y_p| < \delta \text{ for every } p = 1, \dots, n\}$.

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 ψ_{pq} with the following property: For every real-valued continuous function $f : \mathcal{I}^n \rightarrow \mathcal{R}$ there are continuous functions ϕ_q such that

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

Further improvements by Sprecher provide a form that is more suitable for computational algorithm. Namely, the set of functions ψ_{pq} is replaced by shifts of a fixed function ψ 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 λ_p, β , etc.

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

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

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

3 Algorithm Proposal

Based on Theorem 2 Sprecher in [7] proposed a numerical algorithm whose core consists of four steps:

Let n be the input dimension. Let f is a known continuous function, $e_0 \equiv f$. For each iteration r the number $k = k(r)$ is determined according to previous approximation f_{r-1} . Note that the number $k(r)$ determines the dissection of the unit cube $\mathcal{I}^n$ and plays an important role for convergence of the algorithm.

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

$$\mathcal{Q}^n = \{\mathbf{d}_k = (d_{k1}, \dots, d_{kn}); d_{kp} \in \mathcal{Q}, p = 1, \dots, n\}, \quad (4)$$

where $\mathcal{Q} = \{d_{kp} = \sum_{s=1}^k i_s \gamma^{-s}; i_s \in \{0, 1, \dots, \gamma - 1\}, k \in \mathcal{N}\}$ and integers m, γ are such that $m \geq 2n$ and $\gamma \geq m + 2$.

2. Construct monotonically increasing functions $\psi : \mathcal{Q} \rightarrow \mathcal{I}$ whose linear combination defines the functions ξ in the points $\mathbf{d}_k^q$. The functions $\xi(\mathbf{d}_k^q)$ are expressed by equation

$$\xi(\mathbf{d}_k^q) = \sum_{p=1}^n \lambda_p \psi(d_{kp}^q) \quad (5)$$

where $\psi(d_{kp}^q)$ are defined by means of representation of 6 and real coefficients λ_p are the results of infinite sums dependent on γ (for details cf. [5]).

For $q = 0, 1, \dots, m$ numbers $\mathbf{d}_k^q \in \mathcal{Q}^n$ are constructed as:

$$d_{kp}^q = d_{kp} + q \sum_{s=1}^k \gamma^{-s}. \quad (6)$$

Obviously $d_{kp}^q \in \mathcal{Q}$ for $p = 1, 2, \dots, n$.

3. Create sigmoidal steps θ by means of a pair of shifted sigmoidal functions σ

$$\begin{aligned} \theta(\mathbf{d}_k^q; y_q) &= \sigma(\gamma^{\beta(k+1)}(y_q - \xi(\mathbf{d}_k^q)) + 1) \\ &\quad - \sigma(\gamma^{\beta(k+1)}(y_q - \xi(\mathbf{d}_k^q) - (\gamma - 2)b_k), \end{aligned} \quad (7)$$

where real numbers b_k are results of infinite sums dependent on γ (for details cf. [5]).

Functions θ and previous iteration error function e_{r-1} are then used to compile the outer functions ϕ_q^r for $q = 0, \dots, m$

$$\phi_q^r(y_q) = \frac{1}{m+1} \sum_{\mathbf{d}_k^q} e_{r-1}(\mathbf{d}_k) \theta(\mathbf{d}_k^q; y_q), \quad (8)$$

Note that due to the definition of sigmoidal function σ (2) $\theta(\mathbf{d}_k; y_q) \in [0, 1]$, $\theta(\mathbf{d}_k; y_q) = 1$ for $y_q \in [\xi(\mathbf{d}_k^q); \xi(\mathbf{d}_k^q) + (\gamma - 2)b_k]$.

The functions $\phi_q(y_q)$ from theorem 2 are then expressed as: $\phi_q(y_q) = \lim_{r \rightarrow \infty} \sum_{j=1}^r \phi_q^j(y_q)$.

4. Construct the r -th approximation f_r of original function f , and the r -th approximation error e_r .

$$e_r(\mathbf{x}) = e_{r-1}(\mathbf{x}) - \sum_{q=0}^m \phi_q^r \circ \xi(\mathbf{x}_q), \quad (9)$$

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

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

$$f_r(\mathbf{x}) = f_{r-1} + \sum_{q=0}^m \phi_q^j \circ \xi(\mathbf{x}_q). \quad (10)$$

We conclude this section by several notes and observations mostly taken from [7].

The convergence of the above algorithm is assured if $k(r)$ is set such that the dissection of the mesh $\mathcal{Q}^n$ in each step r corresponds to the modulus of continuity $\omega_{f_{r-1}}$.

Another assumption used in the proof is that the uniform norm f as well as the values of f on an everywhere dense set in $\mathcal{I}^n$ are known. Then we can restrict ourselves to the mesh $\mathcal{Q}^n$ and no other values than $f(\mathbf{d}_k^q)$ are required.

In practice, function f is typically given only by a prescribed set of arguments and values $\{\mathbf{x}_t, f(\mathbf{x}_t)\}$, the so-called *training set*.

For each k and q denote the family of closed intervals

$$E_k^q(d_k^q) = \left[\frac{-q}{\gamma-1} \gamma^{-k} + d_k, \frac{\gamma-2-q}{\gamma-1} \gamma^{-k} + d_k \right]. \quad (11)$$

Note that for fixed q and k these intervals are separated by gaps. Cartesian product of these intervals

$$S_k^q(\mathbf{d}_k^q) = E_k^q(d_{k,1}^q) \times E_k^q(d_{k,2}^q) \times \dots \times E_k^q(d_{k,n}^q) \quad \text{for } q = 0, 1, \dots, m \quad (12)$$

is a closed cube.

If the training set is appropriate and the set $\mathcal{Q}^n$ is fine enough, i.e. for known value $f(\mathbf{x}_k^q)$ there exists surrounding $S_k^q(\mathbf{d}_k)$ of $\mathbf{d}_k$ such that $\mathbf{x}_k^q \in S_k^q(\mathbf{d}_k)$ then the algorithm converges.

In some cases, the set of known values can be too irregular. Then, some special techniques that involve, among others, the use of interpolating functions, can be applied; see [3]. The disadvantage of this approach is its high computational cost.

4 Serial Implementation

In the following, we consider our practical implementation of the above described algorithm, which introduces several additional computational improvements and simplifications. First, the infinite sums are replaced by sums up to the sufficiently big constant K . Since the sums vanish quickly, usually, $K = 8$ suffices to reach machine precision limit. For the sake of simplicity we also do not deal with modulus of continuity of the function f and keep the number $k(r)$ fixed to k . Again, small values of $k = 3, 4, 5$ show to be satisfiable.

Note that all computations of f are performed on numbers $\mathbf{d}_k^q$ taken from the dense support set $\mathcal{Q}^n$. Due to the nature of functions ψ and ξ , it shows advantageous to represent numbers $\mathbf{d}_k^q$ by means of ordinal values of their coordinates $d_k \in \mathcal{Q}$ instead of only their floating point decimal value. It allows quite efficient computation of $\xi(\mathbf{d}_k^q)$ and moreover, it enables caching.

Caching of already computed values is quite a straightforward technique that can save reasonable amount of computations compared to formulae presented above in our sketch of algorithm. In our serial implementation caching takes place in the computations of values of $\psi(d_k^q)$, and $\xi(\mathbf{d}_k^q)$.

Taking all these computational issues into account, we have analyzed the time complexity of this serial implementation. The following theorem 3 summarizes our estimation of the total amount of computational time for one iteration (cf. [5]).

Theorem 3. *The computational requirements in one iteration of the sequential algorithm is estimated as follows.*

$$T_S = \mathcal{O}(m n \gamma^{nk}(n^k + k^3)). \quad (13)$$

5 Parallel Implementation I

One possible way how to parallelize the above described algorithm is to take the outermost sum from 10 and distribute the computations of mutually independent terms $\Phi_0^r, \dots, \Phi_m^r$ on separate processors. Additionally, one master processor is used to compute e_r and f_r , respectively, and to distribute the computed values back to the nodes.

We have performed our analysis and experiments for parallel architecture of cluster of workstations. No special network topology is considered, and we assume that (within a reasonable range of data amounts) the cost of sending one floating point number over the network is constant. Let us denote t_c such a time needed to communicate one float, expressed w.r.t. the cost of one arithmetic operation. Note that, considering a common Pentium processors connected via fast Ethernet, $t_c \approx 10^1 - 10^2$.

An analysis of the parallel time complexity is summarized in the following result (cf. [6]).

Theorem 4. *The total time for one iteration of the parallel algorithm is estimated as follows.*

$$T_{P_1} = \mathcal{O}(n \gamma^{nk}(n^k + k^3) + m \gamma^{nk} t_c). \quad (14)$$

Comparison of theorems 3 and 4 shows that this method of parallelization is very efficient, yet limited. The efficiency means that, employing m additional processors, one achieves speedup in the order of m . Considering reasonable values of t_c , this holds even for quite small values of parameters n , m and k . In our practical experiments, the communication time was always only a tiny fraction of the overall computation.

The drawback of this approach is the limited degree of parallelism. One cannot utilize more than $m \approx 2n$ processors. This can be overcome by a different way of parallelization which is described in the following section.

6 Parallel Implementation - II

Another natural approach to parallelization of the algorithm is to distribute subsets of $\mathcal{Q}^n$ among processors. Functions ξ map the closed cubes $S_k^q(\mathbf{d}_k^q) = E_k^q(d_{k,1}^1) \times \dots \times E_k^q(d_{k,n}^1)$ into disjunct closed intervals $T_k^q(\mathbf{d}_k^q)$. Therefore, we can provide computations on $S_k^q(\mathbf{d}_k^q)$ in a mutually independent way. One master process dissects $\mathcal{T}^n$ into $S_k^q(\mathbf{d}_k^q)$ and distributes these subdomains to slave processors. They compute their corresponding contribution to approximation and deliver the result back to master.

This algorithm allows for much higher degree of parallelization — it can utilize up to γ^{nk} processors. The communication takes place only at the beginning and the end of the algorithm. The iterations can be done independently.

Theorem 5. *The total time for one iteration of the parallel algorithm is estimated as follows.*

$$T_{P_2} = \mathcal{O}(m n (n^k + k^3)). \quad (15)$$

It is not obvious that the computations on each $S_k^q(\mathbf{d}_k^q)$ can be done independently. Since the definitions of sigmoidal steps θ are global, it might seem that the opposite is true. But, a closer observation of equations 7 and 8 reveals that the global approximation is composed only of local steps. Although the sum in 8 is defined over all $\mathbf{d}_k^q$, the function $\theta(\mathbf{d}_k^q, y_q)$ gives non-zero values only on $S_k^q(\mathbf{d}_k^q)$.

Slave nodes can make use of a caching technique from the serial algorithm for their local computations. Each computation of one processor is repeated for $q = 0, \dots, m$. Between each inner cycle iteration nodes do not clear their cache. Because the cubes S_k^q intersect for any two $q \in \{0, \dots, m\}$ this precaution leads to computational saving.

In the following we present sketch of this parallel algorithm more thoroughly.

Parallel algorithm II

Let n be the input dimension. Let f is a known continuous function, $e_0 \equiv f$, k is fixed.

1. On the master node, construct the mesh $\mathcal{Q}^n$ of rational points $\mathbf{d}_k$ dissecting the unit cube $\mathcal{T}^n$ according to 4. Create the family of closed cubes $C(\mathbf{d}_k) = \cup_q S_k^q(\mathbf{d}_k^q)$ for all $\mathbf{d}_k^q \in \mathcal{Q}$, where $S_k^q(\mathbf{d}_k^q)$ is defined by 12.
2. Distribute cubes $C(\mathbf{d}_k)$ among processors. Note, that $S_k^q(\mathbf{d}_k^q) \in C(\mathbf{d}_k)$ for any $q \in \{0, 1, \dots, m\}$.
3. for $q = 1, \dots, m$
 - (a) On each slave node, perform sequential computation (cf. section 4) of 10 and 9 on $S_k^q(\mathbf{d}_k^q)$. Store the results, and compute local partial sum over $S_k^q(\mathbf{d}_k^q)$.
 - (b) Send the result back to the master node.
4. Master node computes the function f as a sum over all cubes $C(\mathbf{d}_k)$.

Fig. 1. Function ξ maps the closed cubes $S_k^q(\mathbf{d}_k^q) = E_k^q(d_{k,1}^1) \times \dots \times E_k^q(d_{k,n}^1)$ into disjunct closed intervals on $\mathcal{I}$.

Fig. 2. Schema of paralelization II. After the decomposition master node sends the domain to the slave nodes which provide the local computation over $S_k^q(\mathbf{d}_k^q)$.

7 Conclusion

Comparing the two above parallel implementations, one can see the pros and cons of both approaches. The first parallel algorithm is quite simple, and very efficient with low communication demands. The degree of parallelism is limited by $2 * n$, though. On the other hand, the second parallel algorithm achieves much better degree of parallelism. Moreover, for the local case, it can make use of the efficient serial implementation (with features like local cache). The disadvantage is in quite a complicated nature of the algorithm.

In our future work we would like to focus on these problems: What is the best way of expressing the algorithm for a general training set. So far, we still consider the data to be from the Q^n . It might be better to use some interpolation scheme and avoid long lasting iterations. We have also considered a fixed value of parameter k over one iteration. The second parallel algorithm allows for a different value of k even among processors, as long as they communicate the value they have used for computations.

In case of relatively large number of processors, one can consider even more dynamic version of parallel algorithm, where the processors are assigned their domains during the second or third iteration. A similar effect can be achieved by increasing the parameter γ . All this can be done independently for each processor, since the individual computations are truly local.

References

1. Robert Hecht-Nielsen. Kolmogorov's mapping neural network existence theorem. In *Proceedings of the International Conference on Neural Networks*, pages 11–14, New York, 1987. IEEE Press.
2. Věra Kůrková. Kolmogorov's theorem is relevant. *Neural Computation*, 3, 1991.
3. Hidefumi Katsuura and David A. Sprecher. Computational aspects of kolmogorov's superposition theorem. *neural Netowrks*, 7:455–461, 1994.
4. 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.
5. R. Neruda, A. Štědrý, and J. Drkošová. Implementation of kolmogorov learning algorithm for feedforward neural networks. In *Proceedings of the International Conference on Computational Science*, pages 986–995. Springer, LNCS 2074, 2001.
6. R. Neruda, A. Štědrý, and J. Drkošová. Kolmogorov learning for feedforward networks. In *Proceedings of the International Joint Conference on Neural Networks*, pages 986–995, Washington,DC, 2001. IEEE.
7. David A. Sprecher. A numerical implementation of Kolmogorov's superpositions II. *Neural Networks*, 10(3):447–457, 1997.