Interval Computations
No 3, 1993

Guaranteed Intervals for Kolmogorov's Theorem (and Their Possible Relation to Neural Networks)

Mitsumi Nakamura, Ray Mines, and Vladik Kreinovich[*]

In the article we prove a *constructive-mathematics* version of Kolmogorov's theorem. In 1957 a solution to one of Hilbert's problems was proved by Kolmogorov.

It actually shows that an arbitrary function can be implemented by a 3-layer neural network with appropriate activation functions. Such a solution has been transformed to the fast iterative algorithm, converges well, but does not provide a *guaranteed* approximation accuracy.

In 1991 Kurkova proposed another algorithm that ensures a given accuracy, however, by means of increasing the number of neurons while the accuracy $\varepsilon \to \infty$ and so does the complexity of the approximating network.

In this paper we describe algorithms that generate the activation functions with guaranteed accuracy and keep number of hidden neurons independent on ε.

Гарантированные интервалы для теоремы Колмогорова и их возможная связь с сетями нейронов

М. Накамура, Р. Майнс, В. Крейнович

В 1957 г. Колмогоров доказал теорему, являющуюся решением одной из проблем Гильберта. В этой статье мы доказываем *конструктивно-математический* вариант теоремы Колмогорова.

Нами доказано, что произвольная функция может быть реализована трехслойной сетью нейронов при подходящих функциях активации. Решение, преобразованное в быстрый итерационный алгоритм, хорошо сходится, но не обеспечивает достижения *заданной* точности приближения.

[*]This work was supported by NSF Grant No. CDA-9015006, NASA Research Grant No. 9-482, and a grant from the Materials Research Institute. We are also greatly thankful to Chitta Baral, Andrew Bernat, and to all the participants of the International Conference on Numerical Analysis with Automatic Result Verification, Lafayette, LA, February–March 1993, especially to Drs. V. Kurkova and S. Markov, and to the anonymous referees for valuable discussions.

В 1991 г. Куркова предложила другой алгоритм, который обеспечивает заданную точность, однако, при точности $\varepsilon \to \infty$, в нем возрастает количество нейронов, а вместе с ним и сложность сети.

В данной статье мы описываем алгоритмы, которые производят функции активации с заданной точностью и, вместе с тем, сохраняют количество скрытых нейронов независящим от ε.

1 A little bit of history, and a formulation of a problem

Before Hilbert [3]. Since ancient times, people know how to solve linear equations $a_0 + a_1 x = 0$. Starting from Babylonians, we can also solve quadratic equations $a_0 + a_1 x + a_2 x^2 = 0$. In the 16th century, Tartaglia, Cardan, and Ferrari showed how to solve cubic and quartic equations. Numerous attempts to find a general solution for algebraic equations

$$a_0 + a_1 x + \cdots + a_n x^n = 0 \tag{1}$$

of 5th and higher order resulted in a famous proof by Galois that there is no way to express such solutions in terms of basic algebraic operations $(+, -, \times, :, \sqrt[n]{\ })$.

In other words, if $n \geq 5$, then a function f that maps $(a_0, \ldots, a_n)$ into a solution of equation (1), cannot be represented as a superposition of basic algebraic operations.

1900: Hilbert's 13th problem. David Hilbert noticed that all basic algebraic operations are functions of one or two variables. So, he formulated a natural hypothesis: that not only one cannot express the solution of higher-order algebraic equations in terms of basic algebraic operations, but no matter what functions of one or two variables we add to these operations, we still won't be able to express the general solution.

Hilbert even included this hypothesis (under No. 13) into the list of 23 major problems that he formulated in 1900 as a challenge for the 20th century.

1957: Kolmogorov's solution. This problem remained a challenge until 1957, when (rather unexpectedly) Kolmogorov [9] proved that an arbitrary continuous function $f(x_1, \ldots, x_n)$ on an n-dimensional cube (of arbitrary dimension n) can be represented as a composition of addition and some functions of one variable.

Formulation of Kolmogorov's theorem. We will take it from [13], where the original Kolmogorov's theorem was simplified and improved (further improvement was later described in [14]).

Definition 1. A function $f(x)$ belongs to a Lipschitz class $\mathrm{Lip}[\alpha]$ if there exists a constant $C > 0$ such that $|f(x) - f(y)| \le C|x - y|^{\alpha}$.

Theorem [13]. *For every integer $N \ge 2$, there exists a monotonic increasing function $\psi : [0,1] \to [0,1]$ such that $\psi \in \mathrm{Lip}[\ln 2/\ln(2N+2)]$, and having the following property: For each $\delta > 0$, there exists a real number $\lambda > 0$ and a rational number ε, $0 < \varepsilon \le \delta$, such that for $2 \le n \le N$, every real continuous function $f : [0,1]^n \to R$, has a representation as*

$$f(x_1, \ldots, x_n) = \sum_{0 \le q \le 2n} \chi \left[\sum_{1 \le p \le n} \lambda^p \psi(x_p + \epsilon q) + q \right]$$

for some continuous function χ.

Kolmogorov's proof is not completely algorithmic in the following sense. It proves that an arbitrary function can be represented as the desired superposition, but it does not provide us with a ready-to-use algorithm for computing the corresponding functions ψ and χ.

To be more precise: this proof contains explicit formulas for the functions χ and ψ, but these formulas cannot be immediately implemented step-by-step on a computer. For example, in order to construct ψ, in [13], two sequences of intervals $E_k(i)$ and $H_k(i)$ are defined, and then, for every real number x, the value $\psi(x)$ is defined ([13], p. 350) as an intersection of all the intervals $H_{k_\nu}(j_{k_\nu}(i_\nu))$ for all infinite sequences $\{k_\nu\}$ and $\{i_\nu\}$ for which x belongs to the intersection

$$\bigcap_{\nu} E_{k_\nu}(i_\nu).$$

There is no way for a computer to handle infinite sequences. Therefore, in order to compute ψ and χ, we must somehow transform the definitions from [13] and [14] (that are not completely algorithmic) into computer algorithms.

1957–1987: interesting math, but of no use. This was a usual attitude to this theorem.

Hecht-Nielsen, 1987: Kolmogorov's theorem describes... neural networks. In 1987, R. Hecht-Nielsen noticed that this result has an interpretation in terms of *neural networks* [7].

A neural network is a way to perform computations using networks of interconnected computational units vaguely analogous to neurons simulating how our brain solves them. A *neuron* is a device with n real inputs $x_1, \ldots,$ x_n and an output $y = g(w_1 x_1 + \cdots + w_n x_n - w_0)$. Here, $g(x)$ is a function that is called an *activation function*, and parameters w_i are called *weights* (w_0 is also called a *threshold*). If we send the output of some neurons as inputs to others, we get a *neural network*.

This network is used as follows: we know several values of input signals and the desired output. So, we send the input signals to the network, and if the result is different from the desired output, we change the weights (*train* the network). After we are done with the training, we *freeze* (fix) the weights, and use a networks to generate an output from given inputs.

The fundamental question is: *can we train a network so that it would compute an arbitrary input-output function?*

Hecht-Nielsen noticed that the above Theorem describes the following neural network: its first layer consists of $n(2n+1)$ neurons N_{pq}, $1 \le p \le n$, $0 \le q \le 2n$, with an activation function $\psi(x)$ and weights $w_p = 1, w_0 = -\varepsilon q$. The next layer consists of $2n+1$ neurons N^1_q with activation function χ, weights λ^p, and thresholds $w_0 = -q$. Its 3rd layer consists of a linear neuron that just adds its inputs.

Therefore, this theorem actually proves that an arbitrary function can be represented by a 3–layer neural network.

1989: First algorithmic version of Kolmogorov's result. We have already noticed that Kolmogorov's proof is not completely algorithmic in the sense that it does not explicitly contain algorithms for computing ψ and χ. So, in order to apply this result to actual computations, we must provide such algorithms.

Such algorithms were proposed in [5] and turned out to be surprisingly fast (see also [12]). Namely, the authors of [5] presented an iterative procedure that converges to ψ and χ as the number of iterations N increases (i.e., as $N \to \infty$). Computer experiments show that this convergence is so fast that it can even help in solving practical problems.

The drawbacks of this algorithm (from the viewpoint of interval

computations). The main goal of interval computations is to provide computations results with guaranteed accuracy.

The algorithm from [5] converges when $N \to \infty$. So, if we want to represent f with a given accuracy ε, then after sufficiently many iterations the resulting expression will be $\varepsilon-$close to f. But this algorithm does not provide us with a *guaranteed* accuracy. In other words, we can run as many iterations as we want, and still there will be no guarantee that we have achieved the desired goal. So, this algorithm cannot be applied to the case when we need a *guaranteed result*, with guaranteed accuracy.

Comment. What we view as a drawback depends on what our goal is. From the viewpoint of interval computations, the main drawback of the first algorithmic version of Kolmogorov's theorem is that it does not guarantee any approximation accuracy. From the practical viewpoint ([6, 10]), another problem surfaces as a main drawback: the functions ψ and χ from Kolmogorov's theorem are highly non-smooth functions (their graphs are fractal [6, 10]) and therefore, they are difficult to compute and even more difficult to implement in hardware.

This is not the drawback of a particular proof or algorithm, but the inherent property of the approach itself: if we take smooth ψ and χ, then the input-output function f will also be smooth. So, if we want to represent non-smooth functions f as well, we must take non-smooth ψ and/or χ.

Second algorithmic version of Kolmogorov's result: main idea. In Kolmogorov's theorem, for every function f, there is a single design that fits f perfectly ($\varepsilon = 0$). Of course, in real life, we cannot manufacture the neurons with exactly the characteristics $\psi(x)$ and $\chi(x)$, but we know that the more precisely we manufacture, the closer is the resulting superposition to f. The design is the same, no matter how precisely we manufacture: the number of neurons in the hidden layer is the same, and the weights are the same.

So, for every function f, we have a single design (independent on the desired accuracy ε).

If we cannot efficiently generate a design that will serve for all ε, then maybe we can algorithmically generate separate designs for each ε?

In other words, we want to be able, given f and $\varepsilon > 0$, to generate a neural network for which the input-output function $\tilde{f}$ is $\varepsilon-$close to f, i.e., for which for all $x_i \in [0,1]$,

$$\tilde{f}(x_1, \ldots, x_n) \in \big[f(x_1, \ldots, x_n) - \varepsilon, f(x_1, \ldots, x_n) + \varepsilon\big].$$

The existence of a neural network that approximates any given function with a given precision, was proved by Hornik et al. [8]. This proof itself is non-algorithmic: it uses a Stone-Weierstrass theorem.

1989–92: Second algorithmic version of Kolmogorov's result. In principle, one can apply a constructive version of the Stone-Weierstrass theorem [1, 4, 2] and produce an algorithmic version of the result from [8]. The resulting algorithm will be, however, very complicated and thus impractical.

An efficient approximation algorithm has been proposed by V. Kurkova in [10, 11]. She has also given estimates for the number of hidden neurons. These estimates are reasonably low, and overall, her algorithm is very practical.

Remaining problem. In Kolmogorov's theorem, for every function f, there is a single design that fits f perfectly ($\varepsilon = 0$). In real life, as we have already mentioned, we cannot manufacture the neurons with exactly the characteristics $\psi(x)$ and $\chi(x)$, but we know that the more precisely we manufacture, the closer is the resulting superposition to f. The design is the same, no matter how precisely we manufacture, and the number of hidden neurons is the same for all ε.

Using Kurkova's algorithm, for different ε, we get different designs, and the number of hidden neurons in an approximating network increases when $\varepsilon \to 0$ (this number actually tends to ∞).

Switching to Kurkova's algorithm, we gain guaranteed approximation property, but we lose in elegance: namely, to get better approximation, we cannot just fine-tune the existing networks; we must add more and more hidden neurons. Is this necessary? Is it possible to find an algorithmic design with guaranteed approximation accuracy that will work for all ε?

What we are planning to do. Our answer is "yes". Crudely speaking, we will prove that we can algorithmically construct a single design that guarantees the approximation accuracy for all ε. We thus give a new algorithmic version of Kolmogorov's theorem.

It is necessary to give some definitions. In order to formulate our result, we must give definitions of what "algorithmic" means when we talk about functions and numbers that can be computed with a guaranteed accuracy. These definitions are more or less standard in the so-called *constructive mathematics* (see, e.g., [1, 2]), but we decided to give them, because these definitions are somewhat different from what we use in computer languages.

2 Definitions and the main result

In this paper, we assume that the readers already know what an algorithm is: crudely speaking, it is a computer program that transforms a finite sequence of symbols (e.g., an integer) into another finite sequence of symbols. We also assume that a reader is well acquainted with the notion of a *subroutine* (*procedure*). In computation theory, if a program $\mathcal{A}$ calls another program $\mathcal{B}$ as a subroutine, it is sometimes said that $\mathcal{A}$ uses $\mathcal{B}$ as an *oracle.*

Definition 2. We say that an algorithm $\mathcal{U}$ *computes* a real number x if for every natural number k, it generates a rational number r_k such that $|r_k - x| \leq 2^{-k}$. We say that we have a *computable* real number if we have an algorithm $\mathcal{U}$ that computes it.

Definition 3. We say that an algorithm $\mathcal{V}$ *computes* a function $f : R \to R$ if $\mathcal{V}$ includes calls to an (unspecified) algorithm $\mathcal{U}$ so that when we take as $\mathcal{U}$ an algorithm that computes a real number x, $\mathcal{V}$ will compute a real number $f(x)$. We say that we have a *computable* real function $f(x)$ if we have an algorithm that computes this function. We will also say that $f(x)$ is *computable* from x.

Comments.

1. This algorithm $\mathcal{V}$ takes k as input, and generates a rational number s_k such that $|s_k - f(x)| \leq 2^{-k}$. In course of computations, it may generate an auxiliary number l, and ask $\mathcal{U}$ for a value r_l that is 2^{-l}–close to x.

2. In a similar manner, one can define a constructive function of n real variables: it just calls n programs $\mathcal{U}_i$, $1 \leq i \leq n$.

Definition 4. We say that a computable function f is *constructively continuous* on a set S if there exists an algorithm, that for every $\varepsilon > 0$, generates $\delta > 0$ such that if $|x - y| \leq \delta$, then $|f(x) - f(y)| \leq \varepsilon$.

Comment. In Kolmogorov's theorem, a function ψ does not depend on f, so Definition 3 explains what we mean by computing it. As for χ, it depends on f, so we must explain what we mean by being able to compute it from f.

Definition 5. We say that an algorithm $\mathcal{W}$ *computes* a function $g : R \to R$ from a function $f : R \to R$ if $\mathcal{W}$ includes calls to (unspecified) algorithms $\mathcal{U}$ and $\mathcal{V}$ so that when we take as $\mathcal{U}$ an algorithm that computes a real number x, and as $\mathcal{V}$, an algorithm that computes f, $\mathcal{W}$ will compute a real number $g(x)$. We say that a real function $g(x)$ is *computable* from $f(x)$ if we have an algorithm that computes g from f.

Constructive–mathematics version of Kolmogorov's theorem. *There exists an algorithm $\mathcal{U}$ that for every integer $N \geq 2$, generates a monotonic increasing function $\psi : [0,1] \to [0,1]$ such that*

$$\psi \in \mathrm{Lip}[\ln 2/\ln(2N+2)]$$

and having the following property: For each $\delta > 0$, there exists a real number $\lambda > 0$ and a rational number ε, $0 < \varepsilon \leq \delta$ (both computable from δ), such that for $2 \leq n \leq N$, every real continuous function $f : [0,1]^n \to R$, has a representation as

$$f(x_1, \ldots, x_n) = \sum_{0 \leq q \leq 2n} \chi\left[\sum_{1 \leq p \leq n} \lambda^p \psi(x_p + \epsilon q) + q\right]$$

for some continuous function χ that is computable from f.

Comment. The algorithms that we construct in the proof are very complicated and not yet ready for practical usage. However, we believe that our result (proving that such algorithms are possible) is a necessary first step towards more practical future algorithms.

3 Proof

3.1 An algorithm that computes ψ

We will show how to compute the function defined in [13]. For that, we will analyze step-by-step how this function is constructed in [13], and show how to modify each step to make it algorithmic (in more precise terms, to make it algorithmic with guaranteed accuracy of the result). We will be thus referring to [13] a lot, so we will try to make our notations as close to the ones from [13] as possible.

In [13], ψ is defined in terms of two families of intervals: $E_k(i)$ and $H_k(i)$. So, let us first show how to compute endpoints of these intervals.

1. First, we compute $\gamma = 2n + 1$ ([13], p. 351) and $\lambda = \sqrt[n]{2}$ ([13], p. 348). Then, for every integers $k \geq 0$ and i, we can compute rational endpoints of an interval $E_k(i) = [e_k^-(i), e_k^+(i)]$ ([13], p. 347), where $e_k^-(i) = i\gamma^{-k}$, and

$$e_k^+(i) = e_k^-(i) + ((\gamma - 2)/(\gamma - 1))\gamma^{-k}.$$

For a fixed $k > 0$, these intervals follow the order of i, i.e.,

$$\cdots < e_k^-(i) < e_k^+(i) < e_k^-(i+1) < e_k^+(i+1) < \cdots$$

2. Now, we must describe an algorithm that computes β_k for given k ([13], p. 348). We take $\beta_1 = 1$. According to [13], if β_k is already chosen, we choose an integer β_{k+1} so that $\beta_{k+1} \geq n\beta_k + 1$ and

$$\gamma^{-\beta_{k+1}} < \gamma^{-\beta_k - 1} \min_{H_k} \left| \sum_{1 \leq p \leq n} h_p \lambda^p \right| \tag{2}$$

where H_k denotes the set of all non zero-vectors $(h_1, \ldots, h_n)$, with components h_p from the set $\mathcal{H}_k = \left\{-\gamma^{\beta_k}, \ldots, -\gamma, -1, 0, 1, \gamma, \ldots, \gamma^{\beta_k}\right\}$.

This value can be computed as follows: all the sums in the right-hand sides are constructive numbers (see, e.g., [1, 2]). The minimum of finitely many computable numbers is also computable, and hence, the right-hand side X of (2) is computable. So, there is an algorithm that for every m, generates a 2^{-m}-approximation r_m to X. We know ([13], p. 348) that $X > 0$. Therefore, $X > 2^{-m}$ for some integer m. Hence, from $|X - r_{m+1}| \leq 2^{-(m+1)}$, we conclude that $r_{m+1} > 2^{-(m+1)}$. Vice versa, if $r_l > 2^{-l}$ for some l, this means that $X \geq r_l - 2^{-l} > 0$.

So, to find β_{k+1}, we compute the approximations $r_1, r_2, \ldots,$ to X, and compare each approximation r_m with 2^{-m} (this comparison is algorithmic because both r_m and 2^{-m} are rational numbers). According to what we have just proved, there will be an m for which $r_m > 2^{-m}$. As soon as we get this m, stop. Now, we must find β_{k+1} for which $\beta_{k+1} \geq n\beta_k + 1$ and $\gamma^{-\beta_{k+1}} < r_m - 2^{-m}$ (since $r_m - 2^{-m} \leq X$, this inequality will guarantee that $\gamma^{-\beta_{k+1}} < X$). Such β_{k+1} can be obtained, e.g., by taking $p = n\beta_k + 1$, $n\beta_k + 2$, $n\beta_k + 3$, ... and comparing rational numbers γ^{-p} and $r_m - 2^{-m}$; as soon as we have $\gamma^{-p} < r_m - 2^{-m}$, we can stop and take this p as β_{k+1}.

3. Now, we must show how to compute ε_k ([13], p. 348). According to [13], formula (4.9),

$$\varepsilon_k = (\gamma - 2)\sum_{l=0}^{\infty} \gamma^{-\beta_{k+l}}.$$

Due to $\beta_{k+1} \geq n\beta_k + 1 > \beta_k + 1$, we have $\beta_{k+l} > \beta_k + l$. Therefore, $\gamma^{-\beta_{k+l}} \leq \gamma^{-\beta^k}\gamma^{-l}$. So, from the constructive convergence of the geometric progression γ^{-l} [1, 2], we conclude that this sum also converges constructively, so we can compute ε_k.

4. Now, let us find an algorithm that computes $j_k(i, t)$ ([13], p. 349) for all $k \geq 1, i$, and t. For $k = 1$, $j_k(i, t) = 1$. Formula (4.14) from [13] is already algorithmic, because it reduces computations of j_k to j_{k-1}, ...:

$$j_{k+1}(i,t) = \begin{cases} j_k(i,t)\gamma^{\beta_{k+1}-\beta_k} + t & \text{if } 0 \leq t \leq \gamma - 2 \\ \left\lfloor \frac{1}{2}[j_{k+1}(i,\gamma-2) + j_{k+1}(i+1,0)] \right\rfloor & \text{if } t = \gamma - 1 \end{cases}$$

where $\lfloor\ \rfloor$ denotes an integer part.

5. Let us now show how to compute $\tilde{H}_k(i) = \left[h_k^-(i), h_k^+(i)\right]$ (see [13], pp. 348–349, where this interval is denoted by $H_k(j_k(i))$): first, we compute $i' = \lfloor i/\gamma \rfloor$ and $t = i - i'\gamma$ (in PASCAL notations, $i' = i \text{ div } \gamma$, and $t = i \text{ mod } \gamma$). Then, we take $h_k^-(i) = j_k(i', t)\gamma^{-\beta_k}$ and $h_k^+(i) = h_k^-(i) + \varepsilon_k$. For fixed k, the order of these intervals also follows the order of i.

6. [13] defines $\psi(x)$ in terms of $E_k(i)$ and $H_k(i)$. However, as we have already notices in the main text, the formulas from [13] are not completely algorithmic. Let's design an algorithm.

Motivations. Part 1. According to ([13], (4.16)), if $x \in E_k(i)$, then $\psi(x) \in \tilde{H}_k(i)$. Since ψ is monotonic, this means that if $x \geq e_k^-(i)$, then $\psi(x) \geq h_k^-(i)$, and if $x \leq e_k^+(i)$, then $\psi(x) \leq h_k^+(i)$.

We cannot directly use this property to compute $\psi(x)$, because even for computable real numbers a and b, there is no way to check whether $a \leq b$ or not, and therefore, no way to check whether a given computable number belongs to an interval with computable endpoints [1, 2].

However, if $a < b$, then there exists an algorithm that for every computable c, generates 0 or 1 so that if 0 then $c < b$, and if 1 then $c > a$.

To explain how it works: compute the rational approximations c_l and r_l to c and $(a+b)/2$ with precision $2^{-l} < (b-a)/4$. Then, if $c_l \le r_l$, we generate 0. In this case, $c \le c_l + 2^{-l} \le r_l + 2^{-l} \le (a+b)/2 + 2 \cdot 2^{-l} < (a+b)/2 + 2(b-a)/4 = b$, and $c < b$. Likewise, if $c_l > r_l$, we generate 1, and conclude that $c > a$.

Algorithm. Part 1. If $a < b$ and c are given, we compute l such that $2^{-l} < (b-a)/4$, and compute 2^{-l}-approximations c_l and r_l to c and $(a+b)/2$. Then, we compare r_l and c_l. If $c_l \le r_l$, we generate 0, else 1.

Motivations. Part 2. We want to compute $\psi(x)$ for a given x. So, we take $c = x$, $a = e_k^-(i)$, $b = e_k^+(i)$, and apply this algorithm. If we fix x and k, then for $i \to \infty$, $x < e_k^-(i)$. So, for sufficiently small i, this algorithm cannot return 0. Likewise, for $i \to -\infty$, $x > e_k^+(i)$, so the algorithm cannot return 1. So, we arrive at the following algorithm.

Algorithm. Part 2. Assume that x is a computable real number, and $K > 0$ is an integer. We want to produce r_K such that $|\psi(x) - r_K| \le 2^{-K}$. Let's first take $k = 1$ and $i = 1$. We can apply the above algorithm to $c = x$, $a = e_k^-(i)$, $b = e_k^+(i)$. If this algorithm generates 0, then $x < e_k^+(i)$; in this case, repeat this procedure for $i = 0, -1, -2, \ldots$, until it generates 1. If for $i = 1$, this algorithm generates 1, meaning that $x > e_k^-(i)$, then try $i = 2$, $3, \ldots$ until we get 0.

As a result, we get two consequent values of i for which answers are 1 and 0, i.e., for which $x > e_k^-(i)$ and $x < e_k^+(i+1)$. Therefore, $h_k^-(i) \le \psi(x) \le h_k^+(i+1)$.

Motivations. Part 3. The difference $d(k) = h_k^+(i+1) - h_k^-(i)$ between two computable numbers is computable [1, 2]. Therefore, for each $k = 1, 2, 3, \ldots$, we can compute the $2^{-(K+2)}$-approximation $s_{K+2}(k)$ to this difference. When $k \to \infty$, $d(k) \to 0$ ([13], p. 349). This means, in particular, that for some k, $|d(k)| \le 2^{-(K+1)}$. For this k, $|s_{K+2}(k)| \le |d(k)| + 2^{-(K+2)} \le 2^{-(K+1)} + 2^{-(K+2)}$. Vice versa, if $|s_{K+2}(k)| \le 2^{-(K+1)} + 2^{-(K+2)}$, then

$$|d(k)| \le |s_{K+2}(k)| + 2^{-(K+2)} \le \left(2^{-(K+1)} + 2^{-(K+2)}\right) + 2^{-(K+2)} = 2^{-K}.$$

Likewise, one can easily check that if $|d(k)| \le 2^{-K}$, then a $2^{-(K+1)}$-approximation m_{K+1} to the midpoint $m = \left(h_k^-(i) + h_k^+(i+1)\right)/2$ satisfies the inequality $|\psi(x) - m_{K+1}| \le 2^{-K}$: indeed, since $\psi(x) \in \left[h_k^-(i), h_k^+(i+1)\right]$, we have $|\psi(x) - m| \le 2^{-(K+1)}$ and, therefore,

$$|\psi(x) - m_{K+1}| \le |\psi(x) - m| + |m - m_{K+1}| \le 2^{-(K+1)} + 2^{-(K+1)} = 2^{-K}.$$

So, we arrive at the following algorithm:

Algorithm. Part 3. After Part 2, compute a $2^{-(K+2)}$-approximation $s_{K+2}(k)$ to the difference $d(k) = h_k^+(i+1) - h_k^-(i)$, and check whether $|s_{K+2}(k)| \le 2^{-(K+1)} + 2^{-(K+2)}$. If this inequality is not true, repeat Part 2 for $k = 2, 3, \ldots$, until finally, we get this inequality. Then, compute a $2^{-(K+1)}$-approximation m_{K+1} to the midpoint $m = \big(h_k^-(i) + h_k^+(i+1)\big)/2$. This m_{K+1} is the desired 2^{-K}-approximation to $\psi(x)$.

7. This ψ is not only computable, but also constructively continuous, because one can easily extract an explicit formula for δ in terms of ε from [13, p. 351].

3.2 An algorithm that computes χ from f

Like in the previous subsection, we will show step-by-step how the construction from [13] can be modified so that each step will be algorithmic.

1. First [13, p. 342], we must find an integer k_0 such that

$$(\gamma - 1)^{-1}\gamma^{-k_0} \le \delta.$$

This inequality is equivalent to $k_0 \ge \ln\big(\delta(\gamma-1)\big)/\ln(\gamma)$. Since ln is a computable function, we can compute the 1/2-approximation r_1 to the right-hand side of this inequality, and take $k_0 = \lceil r_1 \rceil + 1$. After that, we compute $\varepsilon = (\gamma-1)^{-1}\gamma^{-k_0}$.

2. Then, for $r = 0, 1, 2, \ldots$, we must compute an integer k_r and a continuous function $\chi_r(x)$ ([13], pp. 344–345). We will also show that χ_r is constructively continuous.

2.1 For $r = 0$, $k_0 = 1$ and $\chi_0(x) = 0$. Assume now that we have already produced k_{r-1}, and generated an algorithm that computes χ_{r-1}. Let's show how to compute k_r and χ_r.

2.2 We know that ψ is computable and constructively continuous. It is also known that a superposition of constructively continuous functions is constructively continuous [1, 2]. Therefore, for $q = 0, \ldots, 2n$ (in this case, $m = n$ in [13]), the following functions are constructively continuous:

$$h^q(x_1, \ldots, x_n) = \sum_{1 \le p \le n} \lambda^p \psi(x_p + \epsilon q) + q$$

and

$$f_{r-1}(x_1,\ \ldots,\ x_n) = \sum_{0\le q\le 2n} \chi_{r-1}\big[h^q(x_1,\ \ldots,\ x_n)\big].$$

Both functions f and f_{r-1} are constructively continuous on the unit cube $[0,1]^n$, therefore, their difference $d_{r-1}(\vec{x}) = f(\vec{x}) - f_{r-1}(\vec{x})$ is also constructively continuous.

According to [13], to find k_r, we must know the least upper bound ($=$ supremum) of the function d_{r-1} on a special set. Let us start computing that set.

2.3 Computing a special set. First, for every $k \ge k_0$, and for every i and $q \le 2n$, we can compute an interval $E^q_k(i) = [e^-_k(i) - \varepsilon q, e^+_k(i) - \varepsilon q]$ ([13], (5.1)). Therefore, for each k, q, and for each set $(i_{1q},\ \ldots,\ i_{nq})$, we can compute the coordinates of a cube $S^q_k(i_{1q},\ \ldots,\ i_{pq}) = E^q_k(i_{1q}) \times E^q_k(i_{2q}) \times \cdots \times E^q_k(i_{nq})$ ([13], (5.5)). According to [13], Lemma 1, for each $i_1,\ \ldots,\ i_n$ such that $0 \le i_j \le \gamma^k$ ([13], (5.2)), the intersection

$$S_k(i_1,\ \ldots,\ i_n) = \bigcap_{q=0}^{2n} S^q_k(i_{1q},\ \ldots,\ i_{nq})$$

where $i_{jq} = i_j + \big(1 + \varepsilon(\gamma^k - 1)\big)q$, is non-empty ([13], p. 352). The coordinates of this intersection can be also easily computed: indeed, the intersection of Cartesian products is actually the Cartesian product of intersections $\cup E^q_k(i_{jq})$, and for each j, the intersection of intervals $E^q_k(i_{jq})$ with computable endpoints $e^-_k(i_{jq}) - \varepsilon q, e^+_k(i_{jq}) - \varepsilon q$ is an interval with computable endpoints $\max\big(e^-_k(i_{j1}),\ \ldots,\ e^-_k(i_{j,2n})\big) - \varepsilon q$ and $\min\big(e^+_k(i_{j1}),\ \ldots,\ e^+_k(i_{j,2n})\big) - \varepsilon q$.

There exist finitely many combinations $(i_1,\ \ldots,\ i_n)$, where $0 \le i_j \le \gamma^k$. Therefore, a union $S_k = \cup S_k(i_1,\ \ldots,\ i_n)$ ([13], (3.1)) of all such sets is a constructive compact in the sense of [1, 2].

The intersection of S_k with a unit cube is also a constructive compact: Indeed, S_k is a union of parallelepipeds, i.e., Cartesian products of intervals. The endpoints of all these intervals are rational; so, for them, we can algorithmically decide for each a and b, whether $a < b$ or not. Therefore, the intersection with a unit cube is a constructive operation.

2.4 Computing the supremum. It is known that we can compute a supremum of any constructively continuous function over any constructive compact [1, 2]. In particular, we can compute the supremum of a constructively continuous function $|d_{r-1}(\vec{x})|$ over $S_{k_{r-1}} \cap [0,1]^n$. Following [13, 3.3], we will denote this supremum by μ_{r-1}.

2.5 Computing k_r. By definition of constructive continuity, we can compute $\delta' > 0$ such that if $|\vec{x} - \vec{y}| \leq \delta'$ then

$$\big|d_{r-1}(\vec{x}) - d_{r-1}(\vec{y})\big| \leq \mu_{r-1}/(2n+1).$$

Now, according to [13], p. 345, we can choose k_r such that $\gamma^{-k_r} \leq \delta'$, or $k_r \geq |\ln(\delta')|/\ln(\gamma)$. This value can be computed using the same way as we computed k_0.

Let's now start computing χ_r.

2.6 Computing $\chi_r(x)$ for x from a union of intervals. For each q, and each $(i_{1q}, \ldots, i_{nq})$, the cube $S^q_{k_r}(i_{1q}, \ldots, i_{nq})$ is a constructive compact, and therefore, we can compute the supremum and infimum of h^q over this cube. In other words, the endpoints of an interval $h^q\big(S^q_{k_r}(i_{1q}, \ldots, i_{nq})\big)$ are computable.

Each set $S_{k_r}(i_1, \ldots, i_n)$ is a non-empty parallelepiped, namely, it is a Cartesian product of intervals with computable endpoints. For each of these intervals, a midpoint $x_j(i_1, \ldots, i_n)$ is thus also computable. Therefore, we have a computable point $\vec{x}(i_1, \ldots, i_n) = \big(x_1(i_1, \ldots), \ldots, x_n(i_1, \ldots)\big)$ in this set.

Now, for the values that belong to the intervals $h^q S^q_{k_r}(i_{1q}, \ldots, i_{nq})$, we define $\chi_r(x)$ as $\chi_r(x) = \chi_{r-1}(x) + d_{r-1}(\vec{x}(i_1, \ldots, i_n))$ ([13], (3.5)). This function is computable and constructively continuous on the union of these (non-intersecting) intervals, and for all such x, it satisfies the inequality ([13], (3.6)): $|\chi_r(x) - \chi_{r-1}(x)| \leq \mu_{r-1}/(2n+1)$.

2.7 Computing $\chi_r(x)$ for all x. Using the constructive version of the Tietze extension theorem ([4], Theorem 10.1), we can extend it to a computable function $\chi'(x)$ that is defined on the entire interval. For the values x outside the intervals, this extension can differ radically from χ_{r-1}. To avoid large positive differences, we take $\chi''(x) = \min\big(\chi_{r-1}(x) + \mu_{r-1}/(2n+1), \chi'(x)\big)$. This χ'' is also a computable and constructively continuous extension of χ_{r-1}, and it satisfies the inequality $\chi''(x) \leq \chi_{r-1} + \mu_{r-1}/(2n+1)$. To guarantee that there are no large negative differences, we take $\chi_r(x) = \max\big(\chi''(x), \chi_{r-1}(x) - \mu_{r-1}/(2n+1)\big)$. This function χ_r is a computable and constructively continuous extension of χ_{r-1}, and it satisfies the inequality (3.6) from [13] for all x.

2.8 Computing $\chi(x)$. According to [13], $\chi(x) = \lim \chi_r(x)$, where $|\chi_r(x) - \chi_{r-1}(x)| \le \nu_0(2n+1)^{-r}$ ([13], (3.9)), and ν_0 is a supremum of $|d_0(x)|$ over a unit cube ([13], (3.3)).

Since d_0 is computable and constructively continuous, so is $|d_0|$, so ν_0 is a computable real number.

For $s > r$, $\chi_s(x) - \chi_r(x) = \big(\chi_s(x) - \chi_{s-1}(x)\big) + \cdots + \big(\chi_{r+1}(x) - \chi_r(x)\big)$, therefore, $|\chi_s(x) - \chi_r(x)| \le |\chi_s(x) - \chi_{s-1}(x)| + \cdots + |\chi_{r+1}(x) - \chi_r(x)|$. When $s \to \infty$, the left-hand side tends to $|\chi - \chi_r|$. Using the inequality ([13], 3.9) to estimate the terms in the right-hand side, we conclude that

$$\big|\chi(x) - \chi_r(x)\big| \le \sum_{p=r+1}^{\infty} \frac{\nu_0}{(2n+1)^p}.$$

The right-hand sum is a sum of a geometric progression, and is thus easy to compute. So, $|\chi(x) - \chi_r(x)| \le (\nu_0/(2n))(2n+1)^{-r}$. This means that the sequence χ_r constructively uniformly converges to χ (i.e., for every ε', we can compute an N' such that for $r \ge N'$, $|\chi_r(x) - \chi(x)| \le \varepsilon'$). According to [1, 2], from this we can conclude that the limit $\chi(x)$ of this sequence is also a computable and constructively continuous function. Q.E.D.

References

[1] Bishop, E. *Foundations of constructive analysis.* McGraw-Hill, 1967.

[2] Bishop, E. and Bridges, D. S. *Constructive analysis.* Springer, N.Y., 1985.

[3] Boyer, C. B. and Merzbach, U. C. *A history of mathematics.* Wiley, N.Y., 1991.

[4] Bridges, D. S. *Constructive functional analysis.* Pitman, London, 1979.

[5] Frisch, H. L., Borzi, C., Ord, G., Percus, J. K., and Williams, G. O. *Approximate representation of functions of several variables in terms of functions of one variable.* Physical Review Letters **63** (9) (1989), pp. 927–929.

[6] Girosi, F. and Poggio, T. *Representation properties of networks: Kolmogorov's theorem is irrelevant.* Neural Computation **1** (1989), pp. 465–469.

[7] Hecht-Nielsen, R. *Kolmogorov's mapping neural network existence theorem.* In: "IEEE International Conference on Neural Networks", San Diego, SOS Printing **2** (1987), pp. 11–14.

[8] Hornik, K., Stinchcombe, M., and White, H. *Multilayer feedforward networks are universal approximators.* Neural Networks **2** (1989), pp. 359–366.

[9] Kolmogorov, A. N. *On the representation of continuous functions of several variables by superposition of continuous functions of one variable and addition.* Dokl. Akad. Nauk SSSR **114** (1957), pp. 369–373.

[10] Kurkova, V. *Kolmogorov's theorem is relevant.* Neural Computation **3** (1991), pp. 617–622.

[11] Kurkova, V. *Kolmogorov's theorem and multilayer neural networks.* Neural Networks **5** (1992), pp. 501–506.

[12] Ness, M. *Approximative versions of Kolmogorov's superposition theorem, proved constructively.* J. Comput. Appl. Math., 1993 (in printing).

[13] Sprecher, D. A. *On the structure of continuous functions of several variables.* Transactions Amer. Math. Soc. **115** (3) (1965), pp. 340–355.

[14] Sprecher, D. A. *An improvement in the superposition theorem of Kolmogorov.* Journal of Mathematical Analysis and Applications **38** (1972), pp. 208–213.

M. Nakamura
Department of Mathematics
University of Texas at Austin
Austin, TX 78712,
USA

R. Mines
Department of Mathematics
New Mexico State University
Las Cruces, NM 88003,
USA

V. Kreinovich
Department of Computer Science
University of Texas at El Paso
El Paso, TX 79968,
USA