Neural Networks 59 (2014) 16–22

Contents lists available at ScienceDirect

Neural Networks

journal homepage: www.elsevier.com/locate/neunet

On computational algorithms for real-valued continuous functions of several variables

David Sprecher*

Department of Mathematics, University of California, Santa Barbara, CA 93106, United States

ARTICLE INFO

Article history:
Received 11 December 2013
Received in revised form 20 May 2014
Accepted 26 May 2014
Available online 26 June 2014

Keywords:
Algorithm
Space-filling curves
Hilbert curve
Kolmogorov superpositions
Superpositions

ABSTRACT

The subject of this paper is algorithms for computing superpositions of real-valued continuous functions of several variables based on space-filling curves. The prototypes of these algorithms were based on Kolmogorov's dimension-reducing superpositions (Kolmogorov, 1957). Interest in these grew significantly with the discovery of Hecht-Nielsen that a version of Kolmogorov's formula has an interpretation as a feedforward neural network (Hecht-Nielse, 1987). These superpositions were constructed with devil's staircase-type functions to answer a question in functional complexity, rather than become computational algorithms, and their utility as an efficient computational tool turned out to be limited by the characteristics of space-filling curves that they determined. After discussing the link between the algorithms and these curves, this paper presents two algorithms for the case of two variables: one based on space-filling curves with worked out coding, and the Hilbert curve (Hilbert, 1891).

1. Introduction

This paper follows the research in Sprecher (2002, 2013), exploring further the link between Kolmogorov's superpositions:

$$f(x_1,\dots,x_n)=\sum_{q=1}^{2n+1}\Phi^q\circ\sum_{p=1}^{n}\psi_p^q(x_p). \tag{1}$$

(Kolmogorov, 1957) and space-filling curves from the point of view of computational algorithms for real-valued continuous functions. Hilbert's name is inextricably connected with this line of research on two counts: it was his Problem 13 out of 23 problems that he proposed in 1900 that led to Formula (1), and nine years earlier he was the first to construct a space-filling curve, using geometric intuition (Hilbert, 1891). This guided later mathematicians, including this author, to construct a variety of such curves.

Formula (1) was implemented as a computational algorithm through a linear ordering of families of pairwise disjoint n-cubes of diminishing diameters, order-coded recursively by the inner functions

$$y^q=\sum_{p=1}^{n}\psi_p^q(x_p) \tag{2}$$

in Formula (1). The functions ψ_p^q result from an iteration of devil's staircase-type functions for fixed q, and the functions (2) determine space-filling curves on which target functions $f : E^n \to R$ are computed. The implicit presence of the space-filling curves in the implementation of Formula (1) can be seen from the following observations.

Let us arrange the rational coordinate points $(d_{1,k},\dots,d_{n,k})\in E^n$, $d_k=\sum_{r=1}^{k}\frac{i_r}{10^r}$, in order of increasing values $y^q=\sum_{p=1}^{n}\psi_p^q(d_{pk})$ for fixed q and k. They can be connected pairwise with a serpentine polygonal curve that does not intersect itself (see Fig. 5 for an example). Generally speaking, these curves can be associated with continuous mappings: $\xi_k^q : E \xrightarrow{\text{into}} E^n$ of the unit interval $E=[0,1]$ into the n-cube $E^n=[0,1]^n$. As $k\to\infty$ they converge for each value of q to a space-filling curve Ξ^q. The curves are the method by which the functions (2) store and locate grid-points, and how neighboring points of a given point are located. The latter is determined by their clustering properties. The curves generated by the functions (2), and by implication the functions themselves, are not suited for efficient implementation.

Researches have approached implementation in a number of ways. In two recent examples, computations were carried out with approximate methods, using cubic splines (Igelnik & Parikh, 2003; Leni, Fougerolle, & Truchete, 2013). Issues of implementation are also discussed in connection with algorithms derived from Formula (1) in Braun (2009), Braun and Griebel (2007) and Leni (2011), among others. The applicability of Formula (1) to neural networks (Hecht-Nielse, 1987) is discussed in Kůrková (1991).

* Tel.: +1 8054527014.
E-mail addresses: sprecher@math.ucsb.edu, sprecher2@me.com.

http://dx.doi.org/10.1016/j.neunet.2014.05.015
0893-6080/© 2014 Elsevier Ltd. All rights reserved.

Consider now one of the inner functions (2), the space-filling curve $\xi^q : E \xrightarrow{\text{into}} E^n$ that it defines, and its parametric representation $y^q = \zeta^q(t)$, with components

$$\begin{cases} x_1 = c_1^q(t) \\ x_2 = c_2^q(t) \\ \cdots \\ x_n = c_n^q(t) \end{cases} \quad t \in E.$$

These are continuous vector-valued functions, and a direct substitution in the right side of (1) gives the representation

$$f(x_1, \ldots, x_n) = \sum_{q=1}^{2n+1} \Phi^q \circ \sum_{p=1}^{n} \psi_p^q(c_p^q(t)),$$

also expressible in the form

$$f(x_1, \ldots, x_n) = \sum_{q=1}^{2n+1} \Phi^q \circ \Psi^q(\zeta^q(t)). \tag{3}$$

This shows the inevitable effect of the space-filling curves on the implementation of Formula (1), since any computational difficulty of the curves is transmitted to the computation of f. We observe that f in these two formulas is representable as a continuous function of the single variable t. Indeed, we have the following general statement.

Theorem 1. *Let Ξ be an arbitrary space-filling curve generated with a continuous (surjective) mapping $\xi : E \xrightarrow{\text{onto}} E^n$ whose parametric representation $y = \zeta(t)$ has components*

$$\begin{cases} x_1 = c_1(t) \\ x_2 = c_2(t) \\ \cdots \\ x_n = c_n(t) \end{cases} \quad t \in E.$$

Then every real-valued continuous function $f : E^n \to R$ has a representation $f(x_1, \ldots, x_n) = f(c_1(t), \ldots, c_n(t))$ as a continuous function of one variable in the topology of space-filling curves.

This theorem is an immediate consequence of the fact that a composition of continuous functions is continuous. One wonders how Hilbert might have phrased Problem 13 had he applied space-filling curves to analysis. It needs to be noted that analytic properties that the function $f(x_1, \ldots, x_n)$ might possess do not transfer as a rule to $f(c_1(t), \ldots, c_n(t))$ and, in fact, there is an inevitable descent in smoothness, because the mappings $y = \zeta(t)$ are nowhere differentiable. We also know that the relationship between the parameter $t \in E$ and the points $(x_1, \ldots, x_n) \in E^n$ cannot be one–one, and while every point $t \in E$ defines a unique point in E^n, a point in the n-cube may have more than one pre-image.

This paper presents in the case of two variables computational algorithms based on formulas of the form (3), with fixed inner functions defined through a priori space-filling curves: a curve Ξ with a coding algorithm based on Sprecher (2013), and the Hilbert curve. The latter is among the most studied and applied space-filling curves, and the literature contains numerous coding and implementation schemes. The algorithm developed here can utilize any of the existing schemes, such as Chung, Huang, and Liu (2007). With either algorithm, the curves per se are not involved in computations beyond coding vertices of grid points. It is a reasonable conjecture that the two algorithms can be extended to continuous functions of more than 2 variables.

2. The first algorithm

This algorithm is derived from the following representation theorem:

Theorem 2. *There are space-filling curves Ξ^q and continuous functions $\xi^q : E^2 \to R$ determined by them, such that every real-valued continuous function $f : E^2 \to R$ has a representation*

$$f(x_1, x_2) = \sum_{q=1}^{3} \Phi^q \circ \xi^q(x_1, x_2) \tag{4}$$

with continuous functions Φ^q.

The algorithm for implementing Formula (4) follows the construction of a space-filling curve Ξ^1 from which the other two curves are obtained as translates. The proof of the theorem is given in the Appendix.

2.1. Coding squares

This section provides the computational basis for the algorithm based on Theorem 2. The coding of grid points is achieved through the space-filling curve constructed below (see Sprecher, 2013). We begin with families of pairwise disjoint Cartesian-product squares

$$s_k(d_{1,k}, d_{2,k}) = \left[d_{1,k}, d_{2,k} + \frac{8}{9.10^k}\right]^2, \quad d_k = \sum_{r=1}^{k} \frac{i_r}{10^r}, \tag{5}$$

where $i_1 = 0, 1, \ldots, 10$ for $k = 1$, and $i_r = 0, 1, \ldots, 9$ for $k > 1$. The 121 squares for $k = 1$ are ordered linearly through their lower left vertices with the coding index $n_1 = 11i_{1,1} + i_{2,1}$ (see Fig. 1):

$$\mathbf{S}(n_1) = \begin{cases} S_1\left(\dfrac{i_{1,1}}{10}, \dfrac{i_{2,1}}{10}\right) \\ \quad i_{1,1} = 0, 2, \ldots, 10 \\ S_1\left(\dfrac{i_{1,1}}{10}, \dfrac{10 - i_{2,1}}{10}\right) \\ \quad i_{1,1} = 1, 3, \ldots, 9 \end{cases} \quad t_{2,1} = 0, 1, \ldots, 10.$$

2.2. Configurations $\sigma_m(n_r)$, $m = 1, 2, \ldots, 7$, $r = 2, 3, \ldots$

The ordering scheme for values $k > 1$ utilizes seven configurations (see Fig. 2). Configuration $\sigma_1(n_r)$ is patterned after $\mathbf{S}(n_1)$ on 81 generic squares with coding index $n_1 = 9i_{1,1} + i_{2,1}$:

$$\sigma_1(n_r) = \begin{cases} S_1\left(\dfrac{i_{1,r}}{10^r}, \dfrac{i_{2,r}}{10^r}\right) \\ \quad i_{1,1} = 0, 2, 4, 6, 8 \\ S_1\left(\dfrac{i_{1,r}}{10^r}, \dfrac{8 - i_{2,r}}{10^r}\right) \\ \quad i_{1,1} = 1, 3, 5, 7, 9 \end{cases} \quad i_{2,1} = 0, 1, \ldots, 9$$

$$\sigma_2(n_r) = \begin{cases} \sigma_1(n_r) & n_r = 0, 1, \ldots, 80 \\ \downarrow (9, 89 - n_r) & n_r = 81, \ldots, 89 \end{cases}$$

$$\sigma_3(n_r) = \begin{cases} \sigma_1(n_r) & n_r = 0, 1, \ldots, 80 \\ \leftarrow (89 - n_r, 9) & n_r = 81, \ldots, 89 \end{cases}$$

$$\sigma_4(n_r) = \begin{cases} \sigma_2(n_r) & n_r = 0, 1, \ldots, 89 \\ \leftarrow (99 - n_r, -1) & n_r = 90, \ldots, 99 \\ \downarrow (-1, 109 - n_r) & n_r = 100, \ldots, 109 \end{cases}$$

$$\sigma_5(n_r) = \begin{cases} \sigma_2(n_r) & n_r = 0, 1, \ldots, 89 \\ \leftarrow (99 - n_r, -1) & n_r = 90, \ldots, 109 \end{cases}$$

$$\sigma_6(n_r) = \begin{cases} \sigma_3(n_r) & n_r = 0, 1, \ldots, 89 \\ \downarrow (-1, 99 - n_r) & n_r = 90, \ldots, 99 \\ \leftarrow (109 - n_r, -1) & n_r = 100, \ldots, 109 \end{cases}$$

$$\sigma_7(n_r) = \begin{cases} \sigma_3(n_r) & n_r = 0, 1, \ldots, 89 \\ \leftarrow (-1, 99 - n_r) & n_r = 90, \ldots, 109. \end{cases}$$