mathematics

MDPI

Article

On Correspondences between Feedforward Artificial Neural Networks on Finite Memory Automata and Classes of Primitive Recursive Functions

Vladimir A. Kulyukin

Department of Computer Science, Utah State University, Logan, UT 84322, USA; vladimir.kulyukin@usu.edu

Abstract: When realized on computational devices with finite quantities of memory, feedforward artificial neural networks and the functions they compute cease being abstract mathematical objects and turn into executable programs generating concrete computations. To differentiate between feedforward artificial neural networks and their functions as abstract mathematical objects and the realizations of these networks and functions on finite memory devices, we introduce the categories of general and actual computabilities and show that there exist correspondences, i.e., bijections, between functions computable by trained feedforward artificial neural networks on finite memory automata and classes of primitive recursive functions.

Keywords: computability theory; theory of recursive functions; artificial neural networks; number theory

MSC: 03D32

Citation: Kulyukin, V.A. On Correspondences between Feedforward Artificial Neural Networks on Finite Memory Automata and Classes of Primitive Recursive Functions. *Mathematics* **2023**, *11*, 2620. https://doi.org/10.3390/math11122620

Academic Editor: Shamil Ishmukhametov

Received: 6 May 2023
Revised: 26 May 2023
Accepted: 5 June 2023
Published: 8 June 2023

1. Introduction

An offspring of McCollough and Pitts' research on foundations of cybernetics [1], artificial neural networks (ANNs) entered mainstream machine learning after the discovery of backpropagation by Rumelhart, Hinton, and Williams [2]. ANNs proved to be universal approximators of different classes of functions when no limits are imposed on the number of artificial neurons in any layer (*arbitrary width*) or on the number of hidden layers (*arbitrary depth*) and even with bounded widths and depths (e.g., [3–5]). ANNs cease being abstract mathematical objects when implemented in specific programming languages on computational devices with finite quantities of internal and external memory, to which we interchangeably refer in our article as *finite memory devices* (FMDs) and *finite memory automata* (FMA). To differentiate between functions computable by ANNs in principle and functions computable by ANNs realized on FMA, we introduce the categories of *general* and *actual* computabilities and show that there exist correspondences, i.e., bijections, between functions computable by trained feedforward ANNs (FANNs) on FMA and classes of primitive recursive functions.

Our article is organized as follows. In Section 2, we expound the terms, definitions, and notational conventions for functions and predicates espoused in this article and define the term *finite memory automaton*. In Section 3, we explicate the categories of general and actual computabilities and elucidate their similarities and differences. In Section 4, we formalize FANNs in terms of recursively defined functions. In Section 5, we present primitive recursive techniques to pack finite sets and Cartesian powers thereof into Gödel numbers. In Section 6, we use the set packing techniques of Section 5 to show that functions computable by trained FANNs implemented on FMA can be archived into natural numbers. In Section 7, we show how such archives can be used to define primitive recursive functions corresponding to functions computable by FANNs. In Section 8, we discuss theoretical and practical reasons for separating computability into the general and actual categories

and pursue some implications of the theorems proved in Section 7. In Section 9, we summarize our conclusions. For the reader's convenience, Appendix A gives supplementary definitions, results, and examples that are referenced in the main text when relevant.

2. Preliminaries

2.1. Functions and Predicates

If f is a function, $dom(f)$ and $codom(f)$ denote the domain and the co-domain of f, respectively. Statements such as $f : S \mapsto R$ abbreviate the logical conjunction $dom(f) = S \wedge codom(f) = R$. A function f is *partial* on a set S if $dom(f)$ is a proper subset of S, i.e., $dom(f) \subset S$. Thus, if $S = \mathbb{N} = \{0, 1, 2, \ldots\}$ and $f(x) = x^{1/3}$, then f is partial on S, because $dom(f) = \{i^3 | i \in \mathbb{N}\} \subset \mathbb{N}$. If S and R are sets, then $S = R$ is logically equivalent to the logical conjunction $S \subseteq R \wedge R \subseteq S$, i.e., S is a subset of R, and vice versa. If f is partial on S and $z \in S$, the following statements are equivalent: (1) $z \in dom(f)$; (2) f is defined on z; (3) $f(z)$ is defined; and (4) $f(z) \downarrow$. The following statements are also equivalent: (1) $z \notin dom(f)$; (2) f is undefined on z; (3) $f(z)$ is undefined; and (4) $f(z) \uparrow$. If f is partial on S and $dom(f) = S$, then f is *total* on S. Thus, $f(x) = x + 1$ is total on $\mathbb{N}$. When $f : S \mapsto R$ is a *bijection*, i.e., f is *injective* (one-to-one) and *surjective* (onto), f is a *correspondence* between S and R.

If S is a set, then $|S|$ is the cardinality of S, i.e., the number of elements in S. S is finite if and only if (iff) $|S| \in \mathbb{N}$. For $n > 0$, S^n is the n-th *Cartesian power* of S, i.e., $S^n = \{(s_0, \ldots, s_{n-1}) | s_i \in S, 0 \leq i \leq n-1\} = \{(s_1, \ldots, s_n) | s_i \in S, 1 \leq i \leq n\}$ Thus, if $f : \mathbb{R}^2 \mapsto \mathbb{N}$, $dom(f) = \{(x_1, x_2) | x_1, x_2 \in \mathbb{R}\}$. The symbol $\vec{x}$ is a sequence of numbers, i.e., a vector, from a set S, i.e., $\vec{x} = (x_0, x_1, \ldots, x_{n-1}) = (x_1, x_2, \ldots, x_n) \in S^n$; $()$ is the empty sequence. If $\vec{x} \in S^n$, its individual elements are $\vec{x}_0 = x_0$, $\vec{x}_1 = x_1$, $\ldots$, $\vec{x}_{n-1} = x_{n-1}$ or, equivalently, $\vec{x}_1 = x_1$, $\vec{x}_2 = x_2$, $\ldots$, $\vec{x}_n = x_n$. If $dom(f) \subseteq S^n$ and $\vec{x} \in S^n$, $f(\vec{x}) = f((\vec{x}_0, \ldots, \vec{x}_{n-1})) = f(x_0, \ldots, x_{n-1}) = f(x_1, \ldots, x_n)$. If $f : dom(f) \mapsto codom(f)$ is a bijection, the inverse of f is $f^{-1} : codom(f) \mapsto dom(f)$. When the arguments of f are evident, f or $f(\cdot)$ abbreviate $f(\vec{x})$, $f(x_0, \ldots, x_{n-1})$, or $f(x_1, \ldots, x_n)$

A total function $P : S^n \mapsto \{0, 1\}$ is a *predicate* if, for any $\vec{x} \in S^n$, $P(\vec{x}) = 1$ or $P(\vec{x}) = 0$, where 1 arbitrarily designates the logical truth and 0 designates a logical falsehood. The symbols $\neg$, $\wedge$, $\vee$, $\rightarrow$, respectively, refer to logical *not*, logical *and*, logical *or*, and logical *implication*. We abbreviate $P(\vec{x}) = 1$ to $P(\vec{x})$ and $P(\vec{x}) = 0$ to $\neg P(\vec{x})$. If P and Q are predicates, then $\neg P \vee Q$ is logically equivalent to $P \rightarrow Q$, i.e., $\neg P \vee Q \equiv P \rightarrow Q$. For clarity, sub-predicates of compound predicates may be included in matching pairs of $\{\}$. Thus, if a compound predicate P consists of predicates P_1, P_2, P_3, and P_4, it can be defined as $P \equiv \{\{P_1 \rightarrow P_2\} \wedge \{P_3 \vee P_4\}\}$. The symbols $\exists$ and $\forall$ refer to the logical *existential* (there exists) and *universal* (for all) quantifiers, respectively. Thus, the statement $(\exists \vec{x} \in S^n) P(\vec{x})$ is logically equivalent to the statement that $P(\vec{x})$ holds for at least one $\vec{x}$ in $dom(P)$, while the statement $(\forall \vec{x} \in S^n) P(\vec{x})$ is logically equivalent to the statement that $P(\vec{x})$ holds for all $\vec{x}$ in $dom(P)$.

2.2. Finite Memory Automata

A *finite memory device* D_j is a physical or abstract automaton with a finite quantity of internal and external memory and an automated capability of executing programs, i.e., finite sequences of instructions written in a formalism, e.g., a programming language for D_j, and stored in the finite memory of D_j. Since bijections exist between expressions over any finite alphabet, i.e., a finite set of symbols or *signs*, and subsets of $\mathbb{N}$ [6], we call the memory of D_j *numerical memory*. The numerical memory consists of *registers*, each of which is a sequence of numerical unit *cells*, e.g., digital array cells, mechanical switches, and finite state machine tape cells. The quantity of numerical memory is the product of the number of registers and the number of unit cells in each register, i.e., this quantity is a natural number.

A cell holds exactly one *elementary sign* from a finite alphabet, e.g., { ".", "0", "1", "2", "3", "4", "5", "6", "7", "8", "9" }, or is empty. The sign of the empty cell is unique and is not an elementary sign. A *number sign* is a sequence of elementary signs in consecutive

cells of a register with no empty cells to the left of the first elementary sign and possibly some empty cells to the right of the rightmost elementary sign. Thus, if " | " is the empty sign on D_j, the alphabet is { ".", "-", "0", "1", "2", "3", "4", "5", "6", "7", "8", "9" }, and each register on D_j has seven cells, then "3.1 | | | |", "3.14 | | |", "3.141 | |", "3.1415 |", and "3.14159" are number signs conventionally interpreted as the real numbers 3.1, 3.14, 3.141, 3.1415, and 3.14159, respectively. An arbitrary number sign interpretation is fixed a priori for a given alphabet and D_j and does not change from sign to sign. Thus, if the alphabet is { ",", "0", "f0", "ff0", "fff0", "ffff0", "fffff0", "ffffff0", "fffffff0", "ffffffff0", "fffffffff0" } and the interpretation is such that "," is interpreted as the decimal point, "0" as 0, "f0" as 1, "ff0" as 2, "fff0" as 3, etc., "*" is the empty sign, and each sign is read left to right, then, if each register on D_j has twenty three cells, the sign "f0,ffff0f0ffff0ff0f0***" is interpreted as 1.41421.

A real number x is *signifiable* on D_j iff a register on D_j can hold its sign. Put another way, a number is signifiable on D_j if, in a programming language L for D_j, the number's sign can be assigned to a variable, i.e., stored in a designated register. When x is signifiable on D_j, we say that x is simply signifiable. A set or a sequence of numbers is signifiable if each number in the set or sequence is signifiable.

$\Delta_j > 0$ is the smallest positive signifiable real number on D_j iff for any signifiable x, there is no signifiable y such that $x < y < x + \Delta_j$. The finite set of real numbers in the closed interval between 0 and 1 signifiable on D_j is

$$R^j_{0,1} \equiv \{x \in \mathbb{R} | x = i\Delta_j < 1\} \cup \{1\}, i \in \mathbb{N}. \tag{1}$$

We note, in passing, a notational convention in Equation (1) to which we adhere in our article: if D_j is an FMA, then the Latin letter j in subscripts or superscripts of symbols is used to emphasize that they are defined *with respect to* D_j. Thus, if D_j and D_k are two FMDs with different quantities of numerical memory, $\Delta_j \neq \Delta_k$.

Lemma 1. *If $z = i\Delta_j$ is a maximal element of $\{x \in \mathbb{R} | x = i\Delta_j < 1\}$ and $y = (i+1)\Delta_j$, then $y \geq 1$.*

Proof. If $y \in R^j_{0,1}$, then $y = 1$, because 1 is the only number in $R^j_{0.1}$ greater than z. If $y \notin R^j_{0,1}$, then $y > 1$ and $z < 1 < y$. □

A corollary of Lemma 1 is that if a, b are signifiable, $a < b$, then

$$R^j_{a,b} \equiv \{x \in \mathbb{R} | x = a + i\Delta_j < b\} \cup \{b\}, i \in \mathbb{N}, \tag{2}$$

is the finite set of signifiable numbers in the closed interval from a to b such that there exists no signifiable number between any two consecutive members of $R^j_{a,b}$ when the latter is sorted in non-descending order.

Lemma 2. *If a, b are signifiable and $b - a \geq \Delta_j$, there exists a bijection $\psi^j_{a,b}$: $R^j_{a,b} \mapsto Z^j_{a,b} = \{0, \ldots, z\} \subset \mathbb{N}$, $z > 0$, where $a + z\Delta_j \geq b$. If $a + z\Delta_j$ is signifiable, it is the smallest signifiable number $\geq b$.*

Proof. Let

$$\psi^j_{a,b}(x) = \begin{cases} k & \text{if } x = a + k\Delta_j < b, \\ z & \text{if } \{\{x = a + z\Delta_j = b\} \vee \\ & \quad \{a + (z-1)\Delta_j < x = b < a + z\Delta_j\}\}. \end{cases} \tag{3}$$

Let $r \in Z^j_{a,b}$. If $r = z$, then $\psi^j_{a,b}(x) = r$, for $x = a + z\Delta_j = b$ or $a + (z-1)\Delta_j < x = b < a + z\Delta_j$. If $r < z$, then $\psi^j_{a,b}(x) = r$, for $x = a + r\Delta_j$. Let $\psi^j_{a,b}(x) = \psi^j_{a,b}(y) = r$. If $r = z$, then $x = a + r\Delta_j = y = b$ or $a + (r-1)\Delta_j < x = b = y < a + r\Delta_j$. If $r < z$, then

$x = a + r\Delta_j = y$. Let $a + z\Delta_j$ be signifiable. If $a + z\Delta_j = b$, it is vacuously the smallest signifiable number $\geq b$. If $a + z\Delta_j > b$, then, since $a + (z-1)\Delta_j < b < a + z\Delta_j$, the assertion that $0 < b - (a + (z-1)\Delta_j) < \Delta_j$ or $0 < a + z\Delta_j - b < \Delta_j$ leads to a contradiction. □

A corollary of Lemma 2 is that ${\psi^j_{a,b}}^{-1} : Z^j_{a,b} \mapsto R^j_{a,b}$ is

$$
{\psi^j_{a,b}}^{-1}(k) = \begin{cases} x & \text{if } x = a + k\Delta_j < b, \\ b & \text{if } \{\{b = a + k\Delta_j\} \vee \\ & \quad \{a + (k-1)\Delta_j < b < a + k\Delta_j\}\}. \end{cases} \tag{4}
$$

Lemmas 1 and 2 draw on the empirically verifiable fact manifested by division underflow errors in modern programming languages: given an FMD D_j and two signifiable real numbers a and b, with $a < b$, the set of signifiable real numbers in the closed interval between a and b is a proper finite subset of the set of real numbers $\mathbb{R}$. Thus, bijections are possible between $R^j_{a,b}$ and finite subsets of $\mathbb{N}$. While these bijections may differ from FMA to FMA in that they depend on the exact quantity of memory on a given FMA, they differ only in terms of the cardinalities of their domains and co-domains: the larger the quantity of memory, the greater the cardinality. A constructive interpretation of Lemmas 1 and 2 is that if we take two signifiable real numbers a and b such that $b - a \geq \Delta_j$, we can effectively enumerate the elements of $Z^j_{a,b}$ by iteratively adding increasing integer multiples of Δ_j to a until we reach b, i.e., $a + z\Delta_j = b$, or go slightly above it, i.e., $a + (z-1)\Delta_j < b < a + z\Delta_j$, for $z > 0$.

To map the elements of $R^j_{a,b}$ to $\mathbb{N}^+ = \{1, 2, 3, \ldots\}$, we define the bijection $\mu^j_{a,b}(x) : R^j_{a,b} \mapsto I^j_{a,b} = \left\{z + 1 | z \in Z^j_{a,b}\right\}$ and its inverse ${\mu^j_{a,b}(x)}^{-1} : I^j_{a,b} \mapsto R^j_{a,b}$ as

$$
\begin{array}{rcl} \mu^j_{a,b}(x) & = & \psi^j_{a,b}(x) + 1; \\ {\mu^j_{a,b}}^{-1}(k) & = & {\psi^j_{a,b}}^{-1}(k-1), k > 0. \end{array} \tag{5}
$$

If we abbreviate $\mu^j_{0,1}$, ${\mu^j_{0,1}}^{-1}$, $\psi^j_{0,1}$, ${\psi^j_{0,1}}^{-1}$, $R^j_{0,1}$, and $Z^j_{0,1}$, $I^j_{0,1}$ to μ, μ^{-1}, ψ, ψ^{-1}, R, Z, and I, respectively, and let $\Delta_j = 0.2$, we have the following example.

Example 1.

$R = \{0, 0.2, 0.4, 0.6, 0.8, 1\}; Z = \{0, 1, 2, 3, 4, 5\}; I = \{1, 2, 3, 4, 5, 6\};$
$\psi(0) = 0, \psi(0.2) = 1, \psi(0.4) = 2, \psi(0.6) = 3, \psi(0.8) = 4, \psi(1) = 5;$
$\psi^{-1}(0) = 0, \psi^{-1}(1) = 0.2, \psi^{-1}(2) = 0.4, \psi^{-1}(3) = 0.6, \psi^{-1}(4) = 0.8, \psi^{-1}(5) = 1;$
$\mu(0) = 1, \mu(0.2) = 2, \mu(0.4) = 3, \mu(0.6) = 4, \mu(0.8) = 5, \mu(1) = 6;$
$\mu^{-1}(1) = 0, \mu^{-1}(2) = 0.2, \mu^{-1}(3) = 0.4, \mu^{-1}(4) = 0.6, \mu^{-1}(5) = 0.8, \mu^{-1}(6) = 1.$

For $\Delta_j = 0.3$, we have another example.

Example 2.

$R = \{0, 0.3, 0.6, 0.9, 1\}; Z = \{0, 1, 2, 3, 4\}; I = \{1, 2, 3, 4, 5\};$
$\psi(0) = 0, \psi(0.3) = 1, \psi(0.6) = 2, \psi(0.9) = 3, \psi(1) = 4;$
$\psi^{-1}(0) = 0, \psi^{-1}(1) = 0.3, \psi^{-1}(2) = 0.6, \psi^{-1}(3) = 0.9, \psi^{-1}(4) = 1;$
$\mu(0) = 1, \mu(0.3) = 2, \mu(0.6) = 3, \mu(0.9) = 4, \mu(1) = 5;$
$\mu^{-1}(1) = 0, \mu^{-1}(2) = 0.3, \mu^{-1}(3) = 0.6, \mu^{-1}(4) = 0.9, \mu^{-1}(5) = 1.$

3. Computability: General vs. Actual

Computability theory lacks a uniform, commonly accepted formalism for computable, partially computable, and primitive recursive functions. The treatment of such functions in our article is based, in part, on the formalism by Davis, Sigal, and Weyuker

(Chapters 2 and 3 in [7]), which has, in turn, much in common with Kleene's formalism (Chapter 9 in [8]). Alternative treatments include [9], where primitive recursive functions are formalized as loop programs consisting of assignment and iteration statements similar to DO statements in FORTRAN, and [10], where λ-calculus is used. These symbolically different treatments have one feature in common: computable, partially computable, and primitive recursive functions operate on natural numbers and the underlying automata, explicit or implicit, on which these functions can, in principle, be executed if implemented as programs in some formalism, have access to infinite numerical memory. To distinguish computability in principle from computability on finite memory automata, we introduce the categories of *general* and *actual* computabilities.

3.1. General Computability

As our formalism in this section, we use the programming language $\mathcal{L}$ developed in Chapter 2 in [7] and subsequently used in that book to define partially computable, computable, and primitive recursive functions and to prove various properties thereof. An $\mathcal{L}$ program $\mathcal{P}$ is a *finite* sequence of $\mathcal{L}$ instructions. The unique variable Y is designated as the output variable where the output of $\mathcal{P}$ on a given input is stored. $X_1, X_2, \ldots$ designate input variables, and $Z_1, Z_2, \ldots$ refer to internal variables, i.e., variables in $\mathcal{P}$ that are not input variables. No bounds are imposed on the magnitude of natural numbers assigned to variables. $\mathcal{L}$ has conditional dispatch instructions; line labels; elementary arithmetic operations on and comparisons of natural numbers; and macros, i.e., statements expandable into primitive $\mathcal{L}$ instructions.

A *computation* of $\mathcal{P}$ on some input $\vec{x} \in \mathbb{N}^m$, $m > 0$, is a finite sequence of *snapshots* $(s_1, \ldots, s_k)$, where each snapshot $s_{1 \leq i \leq k}$, $k > 0$, specifies the number of the instruction in $\mathcal{P}$ to be executed and the value of each variable in $\mathcal{P}$. The snapshot s_1 is the *initial* snapshot, where the values of all input variables are set to their initial values, the program instruction counter is set to 1, i.e., the number of the first instruction in $\mathcal{P}$, and the values of all the other variables in $\mathcal{P}$ are set to 0. The snapshot s_k in $(s_1, \ldots, s_k)$ is a *terminal* snapshot, where the instruction counter is set to the number of the instructions in $\mathcal{P}$ plus 1. Not all snapshot sequences are computations. If $(s_1, s_2, \ldots, s_k)$ is a computation of $\mathcal{P}$ on $\vec{x} \in \mathbb{N}^m$, i.e., $X_1 = x_1$, $X_2 = x_2$, ..., $X_m = x_m$, then there is a function that, given the text of $\mathcal{P}$ and a snapshot $s_{1 \leq i < k}$ in the computation, generates the next snapshot s_{i+1} of the computation. This function can verify if $(s_1, \ldots, s_k)$ constitutes the computation of $\mathcal{P}$ on $\vec{x}$. The existence of such functions implies that each instruction in $\mathcal{L}$ is interpreted unambiguously. If some program $\mathcal{P}$ in $\mathcal{L}$ takes m inputs and the values of the input variables are $X_1 = x_1$, $X_2 = x_2$, ..., $X_m = x_m$, then

$$\Psi_{\mathcal{P}}^{(m)}(x_1, x_2, \ldots, x_m) = \begin{cases} Y \text{ in } s_k & \text{if } \exists \text{ a computation } (s_1, \ldots, s_k), k \geq 1, \\ \uparrow & \textit{otherwise} \end{cases} \tag{6}$$

denotes the value of Y in the terminal snapshot s_k if there exists a computation $(s_1, \ldots, s_k)$ of $\mathcal{P}$ on $(x_1, x_2, \ldots, x_m)$ and is undefined otherwise.

Definition 1. *A function $f : \mathbb{N}^m \mapsto \mathbb{N}$, $m \in \mathbb{N}^+$, is partially computable if f is partial and there is an $\mathcal{L}$ program $\mathcal{P}$ such that Equation (7) holds.*

$$(\forall \vec{x} \in \mathbb{N}^m) f(\vec{x}) = \Psi_{\mathcal{P}}^{(m)}(\vec{x}) \tag{7}$$

Equation (7) is interpreted so that $f(\vec{x}) \downarrow$ iff $\Psi_{\mathcal{P}}^{(m)}(\vec{x}) \downarrow$ and $f(\vec{x}) \uparrow$ iff $\Psi_{\mathcal{P}}^{(m)}(\vec{x}) \uparrow$.

Definition 2. *A function $f : \mathbb{N}^m \mapsto \mathbb{N}$, $0 < m \in \mathbb{N}$, is computable if it is total, i.e., $(\forall \vec{x} \in \mathbb{N}^m) f(\vec{x}) \downarrow$, and partially computable.*

Let $f : \mathbb{N}^k \mapsto \mathbb{N}$ and $g_i : \mathbb{N}^n \mapsto \mathbb{N}, 1 \leq i \leq k, n \in \mathbb{N}^+$. Then, $h : \mathbb{N}^n \mapsto \mathbb{N}$ is obtained by *composition* from $f, g_1, \ldots, g_k$ if

$$h(x_1, \ldots, x_n) = f(g_1(x_1, \ldots, x_n), \ldots, g_k(x_1, \ldots, x_n)). \tag{8}$$

Let $k \in \mathbb{N}$, $n \in \mathbb{N}^+$, and $\phi : \mathbb{N}^2 \mapsto \mathbb{N}$, $f : \mathbb{N}^n \mapsto \mathbb{N}$, $g : \mathbb{N}^{n+2} \mapsto \mathbb{N}$ be total. If h is obtained from ϕ by the recurrences in (9) or from f and g by the recurrences in (10), then h is obtained from ϕ or from f and g by *primitive recursion* or simply by *recursion*. The recurrences in (10) are isomorphic to Gödel's recurrences (Section 2, Equation (2) in [6]) where he introduces the concept of *recursively defined number-theoretic function*. The three functions in (11) are the *initial* functions.

$$\begin{array}{rcl} h(0) & = & k, \\ h(t+1) & = & \phi(t, h(t)) \end{array} \tag{9}$$

$$\begin{array}{rcl} h(x_1, \ldots, x_n, 0) & = & f(x_1, \ldots, x_n), \\ h(x_1, \ldots, x_n, t+1) & = & g(t, h(x_1, \ldots, x_n, t), x_1, \ldots, x_n) \end{array} \tag{10}$$

$$\begin{array}{l} s(x) = x + 1; \\ n(x) = 0; \\ u_i^n(x_1, \ldots, x_n) = x_i, 1 \leq i \leq n, \end{array} \tag{11}$$

Definition 3. *A function is primitive recursive if it can be obtained from the initial functions by a finite number of applications of composition and recursion in (8)–(10).*

An implication of Definition 3 is that if f is a primitive recursive function, then there is a sequence of functions $(f_1, \ldots, f_n = f), n > 0$, where every function in the sequence is an initial function or is obtained from the previous functions in the sequence by composition or recursion.

A class $\mathcal{C}$ of total functions is *primitive recursively closed* (PRC) if the initial functions are in it and any function obtained from the functions in $\mathcal{C}$ by composition or recursion is also in $\mathcal{C}$. It has been shown (Chapter 3 in [7]) that (1) the class of computable functions is PRC; (2) the class of primitive recursive functions is PRC; and (3) a function is primitive recursive iff it belongs to every PRC class. A corollary of (3) is that every primitive recursive function is computable.

If $\mathcal{C}$ includes all functions of a certain type, we refer to it as *the* class of those functions, e.g., the class of partially computable functions, the class of computable functions, the class of primitive recursive functions, etc. When we say that $\mathcal{C}'$ is *a* class of functions of a certain type, we mean that $\mathcal{C}' \subseteq \mathcal{C}$, where $\mathcal{C}$ is the class of functions of that type.

3.2. Actual Computability

In general, the FMA defined in Section 2.2 is different from the *finite state automata* of classical computability theory, because the latter, e.g., a Turing machine (TM), do not impose any limitations on memory. A TM becomes an FMA iff the number of cells on its tape where it reads and writes symbols is finite. Analogously, a finite state automaton (FSA) of classical computability is an FMA iff there is a limit, expressed as a natural number, on the length of the input tape from which the FSA reads sign sequences over a given alphabet.

As is the case with general computability, we let P_L^j be a L program, i.e., a finite sequence of unambiguous instructions in a programming language L for an FMD D_j. Thus, if D_j is a physical computer with an operating system, e.g., Linux, a programming language for D_j can be Lisp, C, Perl, Python, etc. If D_j is an abstract FMA, e.g., a TM with a finite number of cells on its tape, then D_j is programmed with the standard quadruple formalism (Chapter 6 in [7]). If D_j is a mechanical device, then we assume that there is a formalism that consists of instructions such as "set switch i to position p", "turn handle full circle clockwise t times", etc. A state of D_j while executing P_L^j on some input $\vec{x}$ includes the

number of the instruction in P_L^j to execute next and, depending on D_j, may include the contents of each register, the signs on the finite input tape, or the state of each mechanical switch. As we did with general computability, we call such a state a *snapshot* of D_j for $P_L^j(\vec{x})$ and define a *computation* of $P_L^j(\vec{x})$ on D_j to be a *finite* sequence of snapshots $(s_1, \ldots, s_k)$, $k \geq 1$, where each subsequent snapshot is computed from the previous snapshot, the initial snapshot s_1 has the values of all the variables in P_L^j appropriately specified and the instruction counter of P_L^j set to 1, and the terminal snapshot s_k has the instruction counter set to the number of the instructions in P_L^j plus 1. We let

$$\Psi_{P_L^j}^{(n)}(\vec{x})$$

denote the number sign corresponding to the output of $P_L^j(\vec{x})$ executed on D_j. It is irrelevant to our discussion where this number sign is stored (e.g., in a register, a section of a finite tape, or the sequence of the positions of the mechanical switches examined left to right or right to left, etc.) so long as it is understood that the output, whenever there is a computation, is unambiguously interpreted as a real number according to an interpretation fixed a priori.

Definition 4. *A partial function $f : \mathbb{R}^m \mapsto \mathbb{R}$, $m \in \mathbb{N}^+$, is actually partially computable on D_j if Equation (12) holds.*

$$(\forall \vec{x} \in \mathbb{R}^m) f(\vec{x}) = \Psi_{P_L^j}^{(m)}(\vec{x}). \tag{12}$$

Equation (12) of actual computability is interpreted so that $f(\vec{x}) \downarrow$ iff $\Psi_{\mathcal{P}}^{(m)}(\vec{x}) \downarrow$, i.e., $f(\vec{x}) = z$ iff $\Psi_{\mathcal{P}}^{(m)}(\vec{x}) = z$, for any $\vec{x} \in \mathbb{R}^m$ and $z \in \mathbb{R}$ signifiable on D_j, and $f(\vec{x}) \uparrow$ iff $\Psi_{\mathcal{P}}^{(m)}(\vec{x}) \uparrow$. However, unlike Equation (7) of general computability, which is defined only on natural numbers and every natural number is signifiable by implication, in actual computability, we have to make provisions for non-signifiable real numbers. Toward that end, we introduce the following inequality, which holds when a non-signifiable number is encountered during a computation of $P_L^j(\vec{x})$.

$$(\exists \vec{x} \in \mathbb{R}^m) f(\vec{x}) \neq \Psi_{P_L^j}^{(m)}(\vec{x}). \tag{13}$$

Inequality (13) can be illustrated with two examples. Let D_j have two cells per register, let $f : \mathbb{N}^2 \mapsto \mathbb{N}$ be $f(x_1, x_2) = x_1 + x_2$, and let $P_L^j(x_1, x_2)$ be a program that implements f, i.e., adds two number signs of x_1 and x_2 and puts the number sign of $x_1 + x_2$ in a designated output register. Let number signs be interpreted in standard decimal notation. Furthermore, if some number x is not signifiable on D_j, only the first two elementary signs of the number sign of x are placed into a register, i.e., number signs are truncated to fit into registers, as is common in many programming languages. Then, after "100" is truncated to "10",

$$f(99, 1) = 100 \neq \Psi_{P_L^j}^{(2)}(99, 1) = 10,$$

and

$$f(213, 13) = 226 \neq \Psi_{P_L^j}^{(2)}(213, 13) = 34,$$

because 213 is not signifiable on D_j and is truncated to "21." In both cases, $f(x_1, x_2)$, as a mathematical object, is total, and there is a computation of $P_L^j(x_1, x_2)$ on $x_1 = 99, x_2 = 1$ and $x_1 = 213, x_2 = 13$, but during both computations, non-signifiable numbers, i.e., 100 and 213, are encountered.

Definition 5. *A function $f : \mathbb{R}^m \mapsto \mathbb{R}$, $m \in \mathbb{N}^+$, is actually computable on D_j if it is total, i.e., $(\forall \vec{x} \in \mathbb{R}^m) f(\vec{x}) \downarrow$, and actually partially computable.*

A program P_L^j that implements an actually computable $f(\vec{x})$ is guaranteed to have a computation for any signifiable $\vec{x}$. However, Inequality (13) may still hold if a non-signifiable number is produced during a computation. Functions can be defined for a specific D_j so that they deal only with signifiable numbers, e.g., whose domains and codomains are, respectively, finite signifiable proper subsets of $\mathbb{R}^m$ and $\mathbb{R}$. The next definition characterizes these functions.

Definition 6. *A function $f : \mathbb{R}^m \mapsto \mathbb{R}$, $m \in \mathbb{N}^+$, is absolutely actually computable on D_j if it is actually computable and Inequality (13) holds for no computation of $P_L^j(\vec{x})$, where $\vec{x}$ is signifiable on D_j.*

An implication of Definitions 4–6 is that if $f : \mathbb{N}^m \mapsto \mathbb{N}$ satisfies Definition 4, it is partially computable according to Definition 1, and if it satisfies Definitions 5 or 6, it is computable according to Definition 2, because, if no memory limitations are placed on registers, every natural number is signifiable.

We call an FMD D_j *sufficiently significant* if three conditions are satisfied. First, a programming language L for D_j exists with the same control structures as the programming language $\mathcal{L}$ described in Section 3.1 such that L (1) is capable of signifying a finite subset of $\mathbb{R}$ and (2) capable of specifying the following operations on numbers: addition, subtraction, multiplication, division, assignment, i.e., setting the value of a register to a number sign, comparison, i.e., $a = b$, $a < b$, $a > b$, $a \leq b$, $a \geq b$, on any signifiable a and b, and the truncation of the signs of non-signifiable numbers to fit them into registers. Second, the finite memory of D_j suffices to hold L programs of length $\leq N \in \mathbb{N}^+$, where the length of the program is the number of instructions in it. Third, the finite memory of D_j suffices, in addition to holding a program of at most N instructions, to hold number signs in $K \in \mathbb{N}^+$ registers.

Lemma 3. *Let an FMA D_j be sufficiently significant with $K \geq 7$, a, b signifiable, $b - a \geq \Delta_j$, and let $a + z\Delta_j$, $z > 0$, be the smallest signifiable number greater than or equal to b. Let $\mu_{a,b}^j$: $R_{a,b}^j \mapsto I_{a,b}^j$ be the bijection in (5). Let $P_L^j(x)$, $x \in R_{a,b}^j$, be a program for D_j that iterates from a to $a + z\Delta_j \geq b$ in positive unit integer increments of Δ_j until k or z that satisfies the conditions in (3) is encountered, and the length of $P_L^j \leq N$. Then, $\mu_{a,b}^j$ is absolutely actually computable.*

Proof. Since a, b, and $a + z\Delta_j$ are signifiable, so are $dom(\mu_{a,b}^j)$ and $codom(\mu_{a,b}^j)$. The finite memory of D_j suffices to hold P_L^j, and P_L^j needs access to five signifiable numbers to iterate over $dom(\mu_{a,b}^j)$: a, b, i, Δ_j, $a + i\Delta_j$. Since $K \geq 7$, the signs of these numbers are placed in registers ρ_1, ρ_2, ρ_3, ρ_4, and ρ_5. After $x \in dom(\mu_{a,b}^j)$ is placed in register ρ_6, P_L^j sets ρ_3 to 0. If $x < b$, P_L^j goes into a while loop with the condition of $\rho_5 < \rho_2$, i.e., $a + i\Delta_j < b$. Inside the loop, when $\rho_5 = \rho_6$, ρ_3 is incremented by 1 and placed into the output register ρ_7, and P_L^j exits. Otherwise, the loop continues with ρ_3 incremented by 1. If $x = b$, P_L^j goes into a while loop with the condition of $\rho_5 \leq \rho_2$, i.e., $a + i\Delta_j \leq b$, and keeps incrementing ρ_3 by 1 inside the loop. After the loop terminates, ρ_3 is incremented by 1 and placed into the output register ρ_7, and P_L^j exits. □

A corollary of Lemma 3 is that ${\mu_{a,b}^j}^{-1}$ is absolutely actually computable.

4. A Recursive Formalization of Feedforward Artificial Neural Networks

A trained feedforward artificial neural network (FANN) N_z^j implemented in a programming language L on a sufficiently significant FMA D_j is a finite set of artificial *neurons*, each of which is connected to a finite number of the neurons in the same set through the *synapses*, i.e., directed weighted edges (See Figure 1). The neurons are organized into $k+1$ layers $E_0, E_1, \ldots, E_k$, with E_0 being the input layer; E_k being the output layer; and E_e, $0 < e < k$, being the hidden layers. We let E_z^j denote the number of layers in N_z^j and $n_{z,i}^{j,e}$ refer to the i-th neuron in layer E_e in N_z^j. We abbreviate $n_{z,i}^{j,e}$ to n_i^e, because n_i^e always refers to a unique neuron in N_z^j. The function $nn_z^j(e) : \mathbb{N} \mapsto \mathbb{N}^+$ specifies the number of neurons in layer E_e of N_z^j and is abbreviated $nn(e)$.

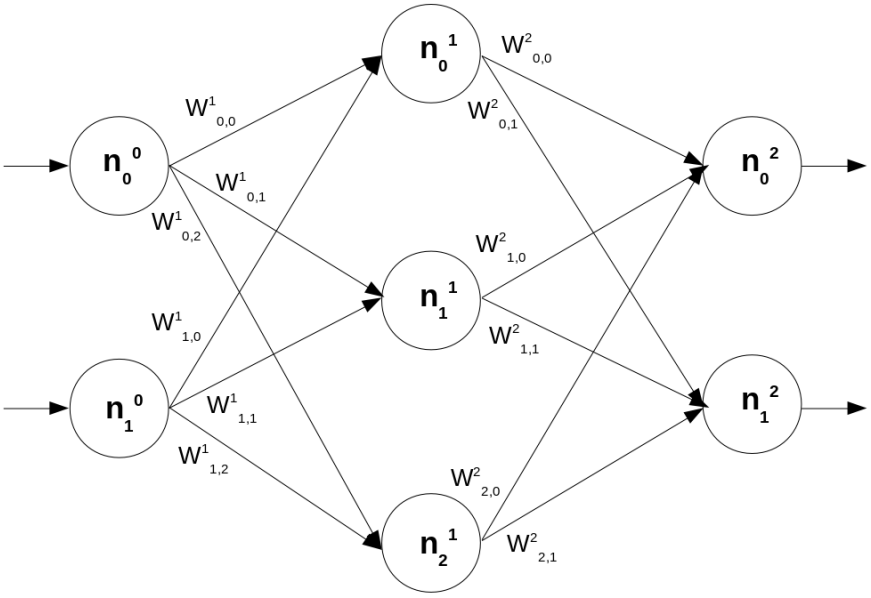

Figure 1. A 3-layer fully connected feedforward artificial neural network (FANN); layer 0 includes the neurons n_0^0 and n_1^0; layer 1 includes the neurons n_0^1, n_1^1, and n_2^1; layer 2 includes the neurons n_0^2 and n_1^2; the two arrows coming into n_0^0 and n_1^0 signify that layer 0 is the input layer; the two arrows going out of n_0^2 and n_1^2 signify that layer 2 is the output layer; $w_{i,j}^e$, $0 < e < 3$, is the weight of the synapse from n_i^{e-1} to n_j^e, e.g., $w_{0,0}^1$ is the weight of the synapse from n_0^0 to n_0^1 and $w_{2,1}^2$ is the weight of the synapse from n_2^1 to n_1^2.

We assume that N_z^j is trained, i.e., the synapse weights are fixed automatically or manually, and *fully connected*, i.e., there is a synapse from every neuron in layer E_{e-1} to every neuron in layer E_e. Each synapse has a weight, i.e., a signifiable real number, associated with it. We let $w_{i,j}^e$, $0 < e < E_z^j$, denote the weight of the synapse from n_i^{e-1} to n_j^e (see Figure 1) and $\vec{w}^e$ refer to a vector of all synaptic weights between E_{e-1} and E_e. We define $\vec{w}^0 = ()$. Thus, for the FANN N_z^j in Figure 1, $\vec{w}^1 = \left(w_{0,0}^1, w_{0,1}^1, w_{0,2}^1, w_{1,0}^1, w_{1,1}^1, w_{1,2}^1\right)$ and $\vec{w}^2 = \left(w_{0,0}^2, w_{0,1}^2, w_{1,0}^2, w_{1,1}^2, w_{2,0}^2, w_{2,1}^2\right)$. We assume, without loss of generality, that all numbers in $\vec{w}^e$ are in $R_{0,1}^j$ defined in (1), because, if that is not the case, they can be so scaled, nor is there any loss of generality associated with the assumption of full connectivity, because partial connectivity can be defined by setting the weights of the appropriate synapses to 0.

If $R_{0,1}^j$ is abbreviated to $R_{0,1}$, each n_i^e in N_z^j, $e > 0$, computes an activation function

$$\alpha_i^e\left(\vec{a}^{e-1}, \vec{w}^e\right) : {R_{0,1}}^{nn(e-1)} \mapsto R_{0,1}, \tag{14}$$

where $\vec{a}^{e-1}$ is the vector of the activations, i.e., real signifiable numbers, of the neurons in layer E_{e-1}. For $e = 0$,

$$\alpha_i^0(\vec{x}, ()) = \vec{x}_i, \tag{15}$$

where $\vec{x} \in {R_{0,1}}^{nn(0)}$ and $\vec{x}_i \in R_{0,1}$, $0 \leq i < nn(0)$. Thus, if $nn(0) = 3$, as in Figure 1, then, given the input $\vec{x} = (x_0, x_1, x_2) = (0.0, 0.3, 0.6)$, $\alpha_0^0(\vec{x}, ()) = \vec{x}_0 = x_0 = 0.0$, $\alpha_1^0(\vec{x}, ()) = \vec{x}_1 = x_1 = 0.3$, $\alpha_2^0(\vec{x}, ()) = \vec{x}_2 = x_2 = 0.6$. Since N_z^j is implemented on a sufficiently significant D_j, all activation functions $\alpha_i^e(\cdot)$ are absolutely actually computable. It is irrelevant to our discussion whether the activation functions are the same, e.g., sigmoid, for all or some neurons, or each neuron has its own activation function.

The term *feedforward* means that the activations of the neurons are computed layer by layer from the input layer to the output layer, because the activation functions of the neurons in the next layer require only the weights of the synapses connecting the next layer with the previous one and the activation values, i.e., the outputs of the activation functions of the neurons in the previous layer. To define the activation vectors of individual layers, let

$$\begin{array}{rcl} \vec{a}^0 & = & \left(\alpha_0^0\left(\vec{x}, ()\right), \ldots \alpha_{nn(0)-1}^0\left(\vec{x}, ()\right)\right), \\ \vec{a}^e & = & \left(\alpha_0^e\left(\vec{a}^{e-1}, \vec{w}^e\right), \ldots, \alpha_{nn(e)-1}^e\left(\vec{a}^{e-1}, \vec{w}^e\right)\right), \end{array} \tag{16}$$

where $0 < e < E_z^j$ and $\vec{x}$ is an input vector. For each N_z^j, we define the absolutely actually computable function that N_z^j computes as

$$\begin{array}{rcl} f_z^j(\vec{x}, 0) & = & \vec{x}, \\ f_z^j(\vec{x}, e+1) & = & \left(\alpha_0^{e+1}\left(f_z^j\left(\vec{x}, e\right), \vec{w}^{e+1}\right), \ldots, \alpha_{nn(e+1)-1}^{e+1}\left(f_z^j\left(\vec{x}, e\right), \vec{w}^{e+1}\right)\right). \end{array} \tag{17}$$

If $e > E_z^j - 1$, let $f(\vec{x}, e) = ()$. The function f_z^j in (17) computes the feedforward activation of N_z^j layer by layer, i.e., $f(\vec{x}, 0) = \vec{a}^0, f(\vec{x}, 1) = \vec{a}^1, \ldots, f(\vec{x}, E_z^j - 1) = \vec{a}^{E_z^j - 1}$. For example, if $\vec{x} = (x_0, x_1) \in {R_{0,1}}^2$ is the input to N_z^j in Figure 1,

$$\begin{array}{rcl} f_z^j(\vec{x}, 0) & = & \vec{a}^0 = \vec{x}; \\ f_z^j(\vec{x}, 1) & = & \left(\alpha_0^1\left(f_z^j\left(\vec{x}, 0\right), \vec{w}^1\right), \alpha_1^1\left(f_z^j\left(\vec{x}, 0\right), \vec{w}^1\right), \alpha_2^1\left(f_z^j\left(\vec{x}, 0\right), \vec{w}^1\right)\right) \\ & = & \left(\alpha_0^1\left(\vec{a}^0, \vec{w}^1\right), \alpha_1^1\left(\vec{a}^0, \vec{w}^1\right), \alpha_2^1\left(\vec{a}^0, \vec{w}^1\right)\right) \\ & = & \vec{a}^1 \in {R_{0,1}^j}^3; \\ f_z^j(\vec{x}, 2) & = & \left(\alpha_0^2\left(f_z^j\left(\vec{x}, 1\right), \vec{w}^2\right)\right), \alpha_1^2\left(f_z^j\left(\vec{x}, 1\right), \vec{w}^2\right)\right) \\ & = & \left(\alpha_0^2\left(\vec{a}^1, \vec{w}^2\right), \alpha_1^2\left(\vec{a}^1, \vec{w}^2\right)\right) \\ & = & \vec{a}^2 \in {R_{0,1}^j}^2. \end{array}$$

5. Finite Sets as Gödel Numbers

Our primitive recursive techniques to pack finite sets and Cartesian powers thereof into Gödel numbers in this section rely, in part, on our previous work on primitive recursive characteristics of chess [11], which, in turn, was based on several functions shown to be primitive recursive in [7]. For the reader's convenience, Appendix A.1 in Appendix A gives the functions shown to be primitive recursive in [7] and gives the necessary auxiliary definitions and theorems. Appendix A.2 in Appendix A gives the functions or variants thereof shown to be primitive recursive in [11]. When we use the functions from [7,11] in this section, we refer to their definitions in the above two sections of Appendix A as necessary.

Let G be a Gödel number (G-number) as defined in (A8). The primitive recursive predicate GP in (18) uses the bounded existential quantification of a primitive recursive predicate defined in (A2) and the primitive recursive functions $(x)_i$ and $Lt(x)$, respectively, defined in (A9) and (A10).

$$\begin{array}{rcl} GP(G) & \equiv & \{Lt(G) > 0\} \wedge \{\{Lt(G) = 1 \wedge Lt((G)_1) > 0\} \vee \\ & & \{(\forall t)_{\leq Lt(G)}\{\{t > 1\} \rightarrow \{\{Lt((G)_t) = Lt((G)_1)\} \wedge \{Lt((G)_t) > 0\}\}\}\}\} \end{array} \tag{18}$$

The logical structure of GP is $GP_1 \wedge \{GP_2 \vee GP_3\}$, where GP_1, GP_2, and GP_3 are

$$\begin{array}{rcl} GP_1 & \equiv & \{Lt(G) > 0\}; \\ GP_2 & \equiv & \{Lt(G) = 1 \wedge Lt((G)_1) > 0\}; \\ GP_3 & \equiv & (\forall t)_{\leq Lt(G)}\{\{t > 1\} \rightarrow \{\{Lt((G)_t) = Lt((G)_1)\} \wedge \{Lt((G)_t) > 0\}\}\}. \end{array}$$

The predicate GP holds for G-numbers with at least one element and whose elements themselves have the same length, i.e., the same number of elements, greater than 0. Thus, $GP([[1]])$, $GP([[1],[2],[3]])$, and $GP([[1,2],[3,4],[5,6]])$, but $\neg GP([[0]])$ and $\neg GP([[1],[3,4,5],[11,10]])$.

Let G be a G-number, the predicate $\in_g$ be as defined in (A13), the function $s(t)$ be as defined in (11), and the function $x \otimes_l y$ be as defined in (A15), and let

$$\begin{array}{rcl} \tau\chi_0(G,0) & = & 1, \\ \tau\chi_0(G,t+1) & = & [[(G)_{s(t)}]] \otimes_l \tau\chi_0(G,t). \end{array}$$

Then, the primitive recursive function

$$\tau_0(G) = \begin{cases} \tau\chi_0(G, Lt(G)) & \text{if } Lt(G) > 0 \wedge 0 \notin_g G, \\ 0 & \text{otherwise} \end{cases} \tag{19}$$

turns a G-number into another G-number whose elements are the elements of the original G-number G, each of which is placed into a G-number whose length is 1. Thus, $\tau_0([11,13]) = [[11],[13]]$. In general, if $G = [g_1, \ldots, g_n]$, $Lt(G) > 0$, $0 \notin_g G$, i.e., $g_i \neq 0$, for $1 \leq i \leq n$, then $\tau_0(G) = [[g_1], \ldots, [g_n]]$.

Let $g \in \mathbb{N}$, G be a G-number, the function $x \otimes_r y$ be defined in (A16), and

$$\begin{array}{rcl} \tau\chi_1(g,G,0) & = & 1, \\ \tau\chi_1(g,G,t+1) & = & [[g] \otimes_r [(G)_{s(t)}]] \otimes_l \tau\chi_1(g,G,t). \end{array}$$

Then, the primitive recursive function

$$\tau_1(g,G) = \begin{cases} \tau\chi_1(g,G,Lt(G)) & \text{if } g > 0 \wedge GP(G), \\ 0 & \text{otherwise} \end{cases} \tag{20}$$

adds g to each element of G. Thus, $\tau_1(1,[[2],[3]]) = [[1,2],[1,3]]$ and $\tau_1(3,[[1,2],[4,5]]) = [[3,1,2],[3,4,5]]$.

Let G_1 and G_2 be two G-numbers, and let

$$\begin{array}{rcl} \tau\chi_2(G_1,G_2,0) & = & 1, \\ \tau\chi_2(G_1,G_2,t+1) & = & \tau_1((G_1)_{s(t)},G_2) \otimes_l \tau\chi_2(G_1,G_2,t). \end{array}$$

Then, the primitive recursive function

$$\tau_2(G_1,G_2) = \begin{cases} \tau\chi_2(G_1,G_2,Lt(G_1)) & \text{if } 0 \notin_g G_1 \wedge GP(G_2) \wedge Lt(G_1) > 0, \\ 0 & \text{otherwise} \end{cases} \tag{21}$$

adds each element of G_1 to each element of G_2. Thus,

$$\begin{array}{lcl}\tau_2([1],[[2],[3]]) & = & [[1,2],[1,3]];\\ \tau_2([1,2],[[4,5],[6,7]]) & = & [[1,4,5],[1,6,7],[2,4,5],[2,6,7]].\end{array}$$

Let G be a G-number, and let

$$\begin{array}{lcl}\tau\chi_3(G,0) & = & \tau_0(G),\\ \tau\chi_3(G,t+1) & = & \tau_2(G,\tau_3(G,t)).\end{array}$$

Then, the primitive recursive function

$$\tau_3(G,t)=\begin{cases}\tau\chi_3(G,t) & \text{if } 0\notin_g G\wedge Lt(G)>0,\\ 0 & \text{otherwise}\end{cases} \tag{22}$$

computes, for $t\in\mathbb{N}^+$, a Gödel number whose components are Gödel numbers representing all sequences of $t+1$ elements of G. Thus,

$$\tau_3([1,2],1) \quad = \quad [[1,1],[1,2],[2,1],[2,2]].$$

Let $S=\{a_1,a_2,\ldots,a_n\}\subset\mathbb{N}^+$, $S\neq\emptyset$, and $G=[a_1,\ldots,a_n]$. An induction on t shows that, for $t>0$, $\tau_3(G,t-1)$ is a G-number representation of S^t in the sense that $(a_{i_1},\ldots,a_{i_t})\in S^t$ iff $[a_{i_1},\ldots,a_{i_t}]\in_g\tau_3(G,t-1)$.

If D_j is an FMA, we let

$$G^j_{a,b}=ggn\left(1,\left|R^j_{a,b}\right|,1\right), \tag{23}$$

where $R^j_{a,b}$ is defined in (2) and $ggn(\cdot)$ is defined in (A17). If we recall from Lemma 2 and (5) that $\mu^j_{a,b}\colon R^j_{a,b}\mapsto I^j_{a,b}=\{1,\ldots,z+1\}$, where $a+z\Delta_j$ is the smallest signifiable real number $\geq b$ on D_j, we observe that $G^j_{a,b}$ is a G-number representation of $I^j_{a,b}$. Thus, if we return to Example 2 and use the accessor function $(x)_i$ in (A9), then for $G^j_{0,1}=[1,2,3,4,5]$, we have

$$\begin{array}{lcccl}\mu(0) & = & 1 & = & \left(G^j_{0,1}\right)_1;\\ \mu(0.3) & = & 2 & = & \left(G^j_{0,1}\right)_2;\\ \mu(0.6) & = & 3 & = & \left(G^j_{0,1}\right)_3;\\ \mu(0.9) & = & 4 & = & \left(G^j_{0,1}\right)_4;\\ \mu(1) & = & 5 & = & \left(G^j_{0,1}\right)_5.\end{array}$$

In general, for $x\in R^j_{a,b}$,

$$\begin{array}{lcl}\mu^j_{a,b}\left(x\right) & = t = & \left(G^j_{a,b}\right)_t\in I^j_{a,b}\\ {\mu^j_{a,b}}^{-1}\left(\left(G^j_{a,b}\right)_t\right) & = & x.\end{array}$$

Let, for $t>1$, τ_3 in (22), and $x\dot{-}y$ in (A4),

$$G^{t,j}_{a,b}=\tau_3\left(G^j_{a,b},t\dot{-}1\right), \tag{24}$$

and, in particular, for $a=0$ and $b=1$, let

$$G^{t,j}_{0,1}=\tau_3\left(G^j_{0,1},t\dot{-}1\right). \tag{25}$$

Then, $G_{0,1}^{t,j}$ is a G-number representation of ${I_{0,1}^{j}}^{t}$, i.e., the t-th Cartesian power of $I_{0,1}^{j}$. Since both τ_3 and $\dot{-}$ are primitive recursive functions, $G_{a,b}^{t,j} \in \mathbb{N}$ and $G_{0,1}^{t,j} \in \mathbb{N}$ are primitive recursively computable.

Example 3. *Let* $R = \{0, 0.3, 0.6, 0.9, 1\}$, $I = \{1, 2, 3, 4, 5\}$ *and* $t = 2$. *Then,*

$$\begin{array}{rcl} G_{0,1}^{2,j} & = & \tau_3(G_{0,1}^{2}, 2 \dot{-} 1) \\ & = & \tau_3(ggn(1, |R|, 1), 1), 1) \\ & = & \tau_3([1, 2, 3, 4, 5], 1) \\ & = & [[1,1],[1,2],[1,3],[1,4],[1,5], \\ & & [2,1],[2,2],[2,3],[2,4],[2,5], \\ & & [3,1],[3,2],[3,3],[3,4],[3,5], \\ & & [4,1],[4,2],[4,3],[4,4],[4,5], \\ & & [5,1],[5,2],[5,3],[5,4],[5,5]]. \end{array}$$

We note that $(x, y) \in I^2$ *iff* $[x, y] \in G_{0,1}^{2,j}$.

Let $\vec{x} \in {R_{a,b}^{j}}^{t}$, $t > 0$, $\tilde{x} \in_g G_{a,b}^{t,j}$, and let $\eta_{a,b}^{t,j} : {R_{a,b}^{j}}^{t} \mapsto \mathbb{N}$ and $\zeta_{a,b}^{t,j} : \mathbb{N} \mapsto {R_{a,b}^{j}}^{t}$ be defined as

$$\begin{array}{rcl} \eta_{a,b}^{t,j}\left(\vec{x}\right) & = & \left[\mu_{a,b}^{j}(\vec{x}_0), \ldots \mu_{a,b}^{j}(\vec{x}_{t-1})\right] = \tilde{x}; \\ \zeta_{a,b}^{t,j}\left(\tilde{x}\right) & = & \left({\mu_{a,b}^{j}}^{-1}\left(\left(\tilde{x}\right)_1\right), \ldots, {\mu_{a,b}^{j}}^{-1}\left(\left(\tilde{x}\right)_t\right)\right). \end{array} \tag{26}$$

If $R_{a,b}^{j}$ is signifiable, $\eta_{a,b}^{t,j}(\vec{x}) = \tilde{x}$ iff $\zeta_{a,b}^{t,j}(\tilde{x}) = \vec{x}$, for any $\vec{x} \in {R_{a,b}^{j}}^{t}$. If $\tilde{x}$ is not signifiable, $\eta_{a,b}^{t,j}$ and $\zeta_{a,b}^{t,j}$ are actually computable; if $\tilde{x}$ is signifiable, the functions are absolutely actually computable.

Example 4. *To continue with Example 3, if* $\vec{x} = (0.9, 0.6) \in {R_{0,1}}^2$ *and* $\tilde{x} = [4, 3] \in G_{0,1}^{2,j}$, *then, if we abbreviate* $\eta_{0,1}^{2}$, $\zeta_{0,1}^{2}$ *to* η^2, ζ^2, *we have*

$$\begin{array}{rcccccl} \eta^2(\vec{x}) & = & (\mu(0.9), \mu(0.6)) & & & = & [4, 3]; \\ \zeta^2(\tilde{x}) & = & (\mu^{-1}((\tilde{x})_1), \mu^{-1}((\tilde{x}_2))) & = & (\mu^{-1}(4), \mu^{-1}(3)) & = & (0.9, 0.6). \end{array}$$

6. Numbers $\Omega_{z,i}^{j,e}$ and Ω_{z}^{j}: Packing FANNs into Natural Numbers

Let us assume that $\mu_{0,1}^{j}$ is absolutely actually computable on a sufficiently significant FMA D_j and abbreviate $\mu_{0,1}^{j}$ to μ, $\zeta_{0,1}^{t,j}$ to ζ^t, and $G_{0,1}^{t,j}$ in (25) to G^t. Let $\langle x, y \rangle$ be as defined in (A5) and $Lt(x)$ be as defined in (A10). Then, for each input neuron n_i^0 in an FANN N_z^j, let

$$\begin{array}{l} \Omega_{z,i}^{j,0} = \Omega_i^0 = \\ \left[\left\langle \left(G^{nn(0)}\right)_1, \mu\left(\alpha_i^0\left(\zeta^{nn(0)}\left(\left(G^{nn(0)}\right)_1\right), ()\right)\right)\right\rangle, \\ \ldots, \\ \left\langle \left(G^{nn(0)}\right)_{Lt\left(G^{nn(0)}\right)}, \mu\left(\alpha_i^0\left(\zeta^{nn(0)}\left(\left(G^{nn(0)}\right)_{Lt\left(G^{nn(0)}\right)}\right), ()\right)\right)\right\rangle\right]. \end{array} \tag{27}$$

We recall that $E_z^j > 0$ is the number of layers in N_z^j. Then, for a hidden or output neuron n_i^e, $0 < e < E_z^j$, let

$$\begin{aligned}
&\Omega_{z,i}^{j,e} = \Omega_i^e = \\
&\left[\left\langle \left(G^{nn(e-1)}\right)_1, \mu\left(\alpha_i^e\left(\zeta^{nn(e-1)}\left(\left(G^{nn(e-1)}\right)_1\right), \vec{w}^e\right)\right)\right\rangle, \\
&\ldots, \\
&\left\langle \left(G^{nn(e-1)}\right)_{Lt\left(G^{nn(e-1)}\right)}, \mu\left(\alpha_i^e\left(\zeta^{nn(e-1)}\left(\left(G^{nn(e-1)}\right)_{Lt\left(G^{nn(e-1)}\right)}\right), \vec{w}^e\right)\right)\right\rangle\right].
\end{aligned} \tag{28}$$

For an FANN N_z^j on D_j and $E = E_z^j - 1$, let

$$\Omega_z^j = \left[\left\langle 0, \left[\Omega_0^0, \ldots, \Omega_{nn(0)-1}^0\right]\right\rangle, \ldots, \left\langle E, \left[\Omega_0^E, \ldots, \Omega_{nn(E)-1}^E\right]\right\rangle\right]. \tag{29}$$

An implication of the definitions of $\langle x, y\rangle$ in (A5) and the G-number in (A8) is that Ω_z^j is unique for N_z^j, because the only way for another FANN N_k^j on D_j to have $\Omega_k^j = \Omega_z^j$ is for N_k^j to have the same number of layers, the same number of neurons in each layer, the same activation function in each neuron, and the same synapse weights between the same neurons, i.e., $N_k^j = N_z^j$. Appendix A.3 in Appendix A gives several examples of how the Ω numbers are computed for N_z^j in Figure 1.

Lemma 4. *Let $\mu_{0,1}^j$ be absolutely actually computable on a sufficiently significant FMA D_j and let N_z^j be an FANN implemented on D_j. Let $0 \le i < nn(0)$, $0 \le k < nn(e)$, $0 < e < E_z^j$, and $G_{0,1}^{t,j}$ in (25) be signifiable on D_j. Then, $\Omega_{z,i}^{j,0} = \Omega_i^0 \in \mathbb{N}$ and $\Omega_{z,i}^{j,e} = \Omega_i^e \in \mathbb{N}$.*

Proof. We abbreviate $\mu_{0,1}^j$ to μ, $\zeta_{0,1}^t$ to ζ^t, and $G_{0,1}^{t,j}$ to G^t, and let

$$\begin{aligned}
z_0 &= \mu\left(\alpha_i^0\left(\zeta^{nn(0)}\left(\left(G^{nn(0)}\right)_{t_0}\right), ()\right)\right); \\
z_e &= \mu\left(\alpha_k^e\left(\zeta^{nn(e-1)}\left(\left(G^{nn(e-1)}\right)_{t_{e-1}}\right), \vec{w}^e\right)\right),
\end{aligned}$$

where $0 < t_0 \le nn(0)$ and $0 < t_{e-1} \le nn(e-1)$. Since μ is absolutely actually computable and G^t signifiable, $\zeta^{nn(0)}, \zeta^{nn(1)}, \ldots, \zeta^{nn(e-1)}$ are absolutely actually computable. Thus, $z_0, z_e \in \mathbb{N}$. The statement of the lemma then follows from the definitions of $\langle x, y\rangle$ in (A5) and the G-number in (A8). □

7. FANNs and Primitive Recursive Functions

For $0 \le e < E_z^j$, $0 \le i < nn(e)$, $x \in \mathbb{N}$, let

$$\tilde{\alpha}_i^e(x) = r\left(asc\left(x, \left(r\left(asc\left(e, \Omega_z^j\right)\right)\right)_{i+1}\right)\right), \tag{30}$$

where $r(\cdot)$ and $asc(\cdot)$ are defined in (A6) and (A19), respectively. An example of computing $\tilde{\alpha}_i^e$ is given at the end of Appendix A.3 in the Appendix A.

Lemma 5. *Let $\mu_{0,1}^j$, abbreviated as μ, be absolutely actually computable on a sufficiently significant FMA D_j and let N_z^j be an FANN implemented on D_j. Let $G_{0,1}^t$ in (25), abbreviated as G^t, be signifiable. Let $0 \le e < E_j^z$, $\eta_{0,1}^{t,j}(\vec{x}) = \eta^t(\vec{x}) = \tilde{x} = \left[\mu\left(\vec{a}_0^e\right), \ldots, \mu\left(\vec{a}_{nn(e)-1}^e\right)\right] \in \mathbb{N}$, where $\vec{a}^e$ is defined in (16). Then,*

$$\tilde{\alpha}_i^e(\tilde{x}) = \begin{cases} \mu\left(\alpha_i^0\left(\zeta^{nn(0)}\left(\left(G^{nn(0)}\right)_t\right), \vec{w}^0\right)\right) & \text{if } e = 0, \\ \mu\left(\alpha_i^e\left(\zeta^{nn(e-1)}\left(\left(G^{nn(e-1)}\right)_t\right), \vec{w}^e\right)\right) & \text{if } e > 0, \end{cases}$$

where $t = asx\left(\tilde{x}, G^{nn(0)}\right)$, *for* $1 \leq t \leq Lt\left(G^{nn(0)}\right)$ *and* $e = 0$; $t = asx\left(\tilde{x}, G^{nn(e-1)}\right)$, *for* $1 \leq t \leq Lt\left(G^{nn(e-1)}\right)$ *and* $e > 0$; *and asx is as defined in (A18).*

Proof. By (28)–(30) and (A18), we have

$$\begin{aligned} \tilde{\alpha}_i^e(\tilde{x}) &= r\left(asc\left(\tilde{x}, \left(r\left(asc\left(e, \Omega_z^j\right)\right)\right)_{i+1}\right)\right) \\ &= r\left(asc\left(\tilde{x}, \left(r\left(\left\langle e, \left[\Omega_0^e, \ldots, \Omega_{nn(e)-1}^e\right]\right\rangle\right)\right)_{i+1}\right)\right) \\ &= r\left(asc\left(\tilde{x}, \left(\left[\Omega_0^e, \ldots, \Omega_{nn(e)-1}^e\right]\right)_{i+1}\right)\right) \\ &= r\left(asc\left(\tilde{x}, \Omega_i^e\right)\right) \end{aligned}$$

If $e = 0$, then $t = asx\left(\tilde{x}, G^{nn(0)}\right)$, for $1 \leq t \leq Lt\left(G^{nn(0)}\right)$. Thus,

$$\tilde{\alpha}_i^e(\tilde{x}) = r\left(asc\left(\tilde{x}, \Omega_i^e\right)\right) = \mu\left(\alpha_i^0\left(\zeta^{nn(0)}\left(\tilde{x}\right)\right), ()\right).$$

If $e > 0$, then $t = asx\left(\tilde{x}, G^{nn(e-1)}\right)$, for $1 \leq t \leq Lt\left(G^{nn(e-1)}\right)$. Thus,

$$\tilde{\alpha}_i^e(\tilde{x}) = r\left(asc\left(\tilde{x}, \Omega_i^e\right)\right) = \mu\left(\alpha_i^e\left(\zeta^{nn(e-1)}\left(\tilde{x}\right)\right), \vec{w}^e\right).$$

□

If $e = 0$ and $\left(G^{nn(0)}\right)_t = \tilde{x}$, for $0 < t \leq Lt\left(G^{nn(0)}\right)$, let

$$\tilde{a}^0 = \left[\tilde{\alpha}_0^0\left(\tilde{x}\right), \ldots, \tilde{\alpha}_{nn(0)-1}^0\left(\tilde{x}\right)\right]. \tag{31}$$

If $0 < e < E_z^j$ and $\left(G^{nn(e-1)}\right)_t = \tilde{x}$, for $0 < t \leq Lt\left(G^{nn(e-1)}\right)$, let

$$\tilde{a}^e = \left[\tilde{\alpha}_0^e\left(\tilde{x}\right), \ldots, \tilde{\alpha}_{nn(e)-1}^e\left(\tilde{x}\right)\right]. \tag{32}$$

Theorem 1. *Let* N_z^j *be an FANN with* $E_z^j > 0$ *layers on a sufficiently significant FMA* D_j, *and let* $f_z^j(\vec{x}, e)$ *in (17) be absolutely actually computable. Let* $\mu_{0,1}^j(\cdot)$ *be absolutely actually computable and* $G_{0,1}^{t,j}$, *for* $t \in \{nn(e) | 0 \leq e < E_z^j\}$, *be signifiable. Then, if* $\tilde{x} = \left[\mu_{0,1}^j\left(\vec{x}_0\right), \ldots, \mu_{0,1}^j\left(\vec{x}_{nn(0)-1}\right)\right] = \tilde{a}^0 = \eta_{0,1}^{nn(0),j}(\vec{x})$, *where* $\eta_{0,1}^{t,j}$ *is defined in (26), there exists a primitive recursive function* $\tilde{f}_z^j(\tilde{x}, e)$ *such that*

$$f_z^j(\vec{x}, e) = \vec{a}^e \quad \textit{iff} \quad \tilde{f}_z^j(\tilde{x}, e) = \tilde{a}^e.$$

Proof. Let us abbreviate f_z^j to f, $\mu_{0,1}^j$ to μ, ${\mu_{0,1}^j}^{-1}$ to μ^{-1}, $\eta_{0,1}^{t,j}$ to η^t, $\zeta_{0,1}^{t,j}$ to ζ^t, and $G_{0,1}^{t,j}$ to G^t. Since G^t is signifiable, ζ^t and η^t are absolutely actually computable. Let

$$\begin{aligned} \tilde{f}_z^j(\tilde{x}, 0) &= \tilde{x}, \\ \tilde{f}_z^j(\tilde{x}, e+1) &= \left[\tilde{\alpha}_0^{e+1}\left(\tilde{f}_z^j\left(\tilde{x}, e\right)\right), \ldots, \tilde{\alpha}_{nn(e+1)-1}^{e+1}\left(\tilde{f}_z^j\left(\tilde{x}, e\right)\right)\right]. \end{aligned}$$

Let us abbreviate $\tilde{f}_z^j$ to $\tilde{f}$, and let $e = 0$. Then $f(\vec{x}, 0) = \vec{a}^0 = \vec{x}$ and $\tilde{f}(\tilde{x}, 0) = \tilde{x}$. We observe that

$$\begin{aligned}\tilde{x} &= \left[\mu\left(\vec{x}_0\right),\ldots,\mu\left(\vec{x}_{nn(0)-1}\right)\right]\\ &= \left[\mu\left(\alpha_0^0\left(\vec{x},\left(\right)\right)\right),\ldots,\mu\left(\alpha_{nn(0)-1}^0\left(\vec{x},\left(\right)\right)\right)\right]\\ &= \eta^{nn(0)}(\vec{x}).\end{aligned}$$

Since μ is an absolutely actually computable bijection,

$$\begin{aligned}\vec{x} &= \left(\mu^{-1}\left(\mu\left(\vec{x}_0\right)\right),\ldots,\mu^{-1}\left(\mu\left(\vec{x}_{nn(0)-1}\right)\right)\right)\\ &= \left(\mu^{-1}\left(\left(\tilde{x}\right)_1\right),\ldots,\mu^{-1}\left(\left(\tilde{x}\right)_{nn(0)}\right)\right)\\ &= \zeta^{nn(0)}(\tilde{x}).\end{aligned}$$

By (26), $\eta^{nn(0)}(\vec{x}) = \tilde{x}$ iff $\zeta^{nn(0)}(\tilde{x}) = \vec{x}$. Thus, $f(\vec{x},0) = \vec{a}^0$ iff $\tilde{f}(\tilde{x},0) = \tilde{x}$. Let $e = 1$. Then,

$$\begin{aligned}f(\vec{x},1) &= \vec{a}^1\\ &= \left(\alpha_0^1\left(\vec{a}^0,\vec{w}^1\right),\ldots,\alpha_{nn(1)-1}^1\left(\vec{a}^0,\vec{w}^1\right)\right)\\ &= \left(\alpha_0^1\left(\vec{x},\vec{w}^1\right),\ldots,\alpha_{nn(1)-1}^1\left(\vec{x},\vec{w}^1\right)\right).\end{aligned}$$

By Lemma 5,

$$\begin{aligned}\tilde{f}(\tilde{x},1) &= \left[\tilde{\alpha}_0^1\left(\tilde{f}\left(\tilde{x},0\right)\right),\ldots,\tilde{\alpha}_{nn(1)-1}^1\left(\tilde{f}\left(\tilde{x},0\right)\right)\right]\\ &= \left[\tilde{\alpha}_0^1\left(\tilde{x}\right),\ldots,\tilde{\alpha}_{nn(1)-1}^1\left(\tilde{x}\right)\right]\\ &= \left[\mu\left(\alpha_0^1\left(\vec{x},\vec{w}^1\right)\right),\ldots,\mu\left(\alpha_{nn(1)-1}^1\left(\vec{x},\vec{w}^1\right)\right)\right]\\ &= \left[\mu\left(\alpha_0^1\left(\vec{a}^0,\vec{w}^1\right)\right),\ldots,\mu\left(\alpha_{nn(1)-1}^1\left(\vec{a}^0,\vec{w}^1\right)\right)\right]\\ &= \left[\mu\left(\vec{a}_0^1,\right),\ldots,\mu\left(\vec{a}_{nn(1)-1}^1\right)\right]\\ &= \tilde{a}^1 = \eta^{nn(1)}(\vec{a}^1).\end{aligned}$$

Since μ is an absolutely actually computable bijection,

$$\vec{a}^1 = \left(\mu^{-1}\left(\mu\left(\left(\tilde{a}^1\right)_1\right)\right),\ldots,\mu^{-1}\left(\mu\left(\left(\tilde{a}^1\right)_{nn(1)}\right)\right)\right),$$

whence, since $\zeta^{nn(1)}(\tilde{a}^1) = \vec{a}^1$ iff $\eta^{nn(1)}(\vec{a}^1) = \tilde{a}^1$, $f(\vec{x},1) = \vec{a}^1$ iff $\tilde{f}(\tilde{x},1) = \tilde{a}^1$.

Let us assume $f(\vec{x},e) = \vec{a}^e$ iff $\tilde{f}(\tilde{x},e) = \tilde{a}^e$ for $e \geq 1$. Then,

$$\begin{aligned}f(\vec{x},e+1) &= \vec{a}^{e+1}\\ &= \left(\alpha_0^{e+1}\left(f\left(\vec{x},e\right),\vec{w}^{e+1}\right),\ldots,\alpha_{nn(e+1)-1}^{e+1}\left(f\left(\vec{x},e\right),\vec{w}^{e+1}\right)\right.\\ &= \left(\alpha_0^{e+1}\left(\vec{a}^e,\vec{w}^{e+1}\right),\ldots,\alpha_{nn(e+1)-1}^{e+1}\left(\vec{a}^e,\vec{w}^{e+1}\right)\right),\end{aligned}$$

and

$$\begin{aligned}\tilde{f}(\tilde{x},e+1) &= \left[\tilde{\alpha}_0^{e+1}\left(\tilde{f}\left(\tilde{x},e\right)\right),\ldots,\tilde{\alpha}_{nn(e+1)-1}^{e+1}\left(\tilde{f}\left(\tilde{x},e\right)\right)\right]\\ &= \left[\tilde{\alpha}_0^{e+1}\left(\tilde{a}^e\right),\ldots,\tilde{\alpha}_{nn(e+1)-1}^{e+1}\left(\tilde{a}^e\right)\right]\\ &= \left[\mu\left(\alpha_0^{e+1}\left(\vec{a}^e,\vec{w}^{e+1}\right)\right),\ldots,\mu\left(\alpha_{nn(e+1)-1}^{e+1}\left(\vec{a}^e,\vec{w}^{e+1}\right)\right)\right]\\ &= \eta^{nn(e+1)}(\vec{a}^{e+1}).\end{aligned}$$

Then,

$$\vec{a}^{e+1} = \left(\mu^{-1}\left(\mu\left(\left(\tilde{a}^{e+1}\right)_1\right)\right),\ldots,\mu^{-1}\left(\mu\left(\left(\tilde{a}^{e+1}\right)_{nn(e+1)}\right)\right)\right),$$

whence, by induction, since $\zeta^{nn(e+1)}(\tilde{a}^{e+1}) = \vec{a}^{e+1}$ iff $\eta^{nn(e+1)}(\vec{a}^{e+1}) = \tilde{a}^{e+1}$, $f(\vec{x}, e+1) = \vec{a}^{e+1}$ iff $\tilde{f}(\tilde{x}, e+1) = \tilde{a}^{e+1}$. □

Let, for $\vec{x} \in R_{0,1}^{j\ nn(0)}$ and $E_z^j > 0$,

$$A_z^j(\vec{x}) = f_z^j(\vec{x}, E_z^j - 1), \tag{33}$$

and, for $\tilde{x} = \eta^{nn(0)}(\vec{x})$, let

$$\tilde{A}_z^j(\vec{x}) = \tilde{f}_z^j(\tilde{x}, E_z^j \dot{-} 1). \tag{34}$$

Then, $A_z^j(\vec{x})$ is the absolutely actually computable function computed by N_z^j and, by Theorem 1, $\tilde{A}_z^j$ is primitive recursive. We are now in a position to prove the final theorem of this article.

Theorem 2. *Let*

$$\mathcal{N}_j = \left\{N_1^j, N_2^j, \ldots, N_k^j\right\}, k \in \mathbb{N}^+, \tag{35}$$

be the set of FANNs implemented on a sufficiently significant FMA D_j, and let

$$\mathcal{A}_j = \left\{A_1^j, A_2^j, \ldots, A_k^j\right\}, k \in \mathbb{N}^+, \tag{36}$$

be the set of corresponding absolutely actually computable functions of the FANNs in $\mathcal{N}_j$, as defined in (33). There exists a bijection between $\mathcal{N}_j$ and a class of primitive recursive functions.

Proof. Let

$$\mathcal{O}_j = \left\{\Omega_1^j, \Omega_2^j, \ldots, \Omega_k^j\right\}, k \in \mathbb{N}^+, \tag{37}$$

be the set of the numbers Ω_z^j defined in (28), each of which uniquely corresponds to $N_z^j \in \mathcal{N}_j$. Let

$$\mathcal{F}_j = \left\{\tilde{A}_1^j, \tilde{A}_2^j, \ldots, \tilde{A}_k^j\right\}, k \in \mathbb{N}^+, \tag{38}$$

be a class of primitive recursive functions, one function per each $\Omega_z^j \in \mathcal{O}_z$, as defined in (34). We observe that

$$\left|\mathcal{N}_j\right| = \left|\mathcal{A}_j\right| = \left|\mathcal{O}_j\right| = \left|\mathcal{F}_j\right| = k.$$

Let $\lambda_1 : \mathcal{N}_j \mapsto \mathcal{A}_j$, $\lambda_2 : \mathcal{A}_j \mapsto \mathcal{O}_j$, and $\lambda_3 : \mathcal{O}_j \mapsto \mathcal{F}_j$ be defined as

$$\begin{array}{rcl} \lambda_1^j(N_j) & = & A_z^j; \\ \lambda_2^j(A_z^j) & = & \Omega_z^j; \\ \lambda_3^j(\Omega_z^j) & = & \tilde{A}_z^j. \end{array}$$

Then, $\lambda_j : \mathcal{N}_j \mapsto \mathcal{F}_j$, defined as

$$\lambda_j(N_z^j) = \lambda_3^j\left(\lambda_2^j\left(\lambda_1^j\left(N_z^j\right)\right)\right), \tag{39}$$

is a bijection. □

8. Discussion

The definition of the finite memory device or automation (FMD or FMA) in Section 2.2 has four main implications. First, a physical or abstract automaton is an FMD when its memory amount is quantifiable as a natural number. Second, characters and strings are not necessary, because bijections exist between any finite alphabet of symbols and natural numbers and, through Gödel numbering, between any strings over a finite alphabet and

natural numbers, hence the term *numerical memory* used in the article. Third, an FSA of classical computability becomes an FMA when the quantity of its internal and external memory is finite, i.e., there is an upper bound in the form of a natural number on the quantity of the machine's memory. It is irrelevant for the scope of this investigation whether the input tape of an FSA, the input and output tapes of such FSA modifications as the Mealy and Moore machines (Chapter 2 in [12]) or the finite state transducers (Chapter 3 in [13]), and the input tape and the stack of a pushdown automaton (PDA) (Chapter 5 in [12]) are considered internal or external memory. Fourth, a universal Turing machine (UTM) (Chapter 6 in [7]) is an FMA when the number of its tape cells is bounded by a natural number, which a fortiori makes any physical computer an FMA. Thus, only one type of universal computer is needed to define all FMA it can simulate.

Consider a universal computer UC capable of executing the universal L program U_1 constructed to prove the Universality Theorem (Theorem 3.1, Chapter 3 in [7]). The computer UC, equivalent to a UTM, takes an arbitrary L program P, an input to that program in the form of a natural number stored in its input register X_1, which can be a Gödel number encoding an array of numbers, executes P on X_1 by encoding the memory of P as another Gödel number and returns the output of P as a natural number, which can also be a Gödel number encoding a sequence of natural numbers, saved in its output register Y. Since characters and character sequences can be bijectively mapped to natural numbers, UC can simulate any FSA or a modification thereof, e.g., a Mealy machine, a Moore machine, a finite state transducer, or a PDA. Technically speaking, there is no need to distinguish between the Mealy and Moore machines, because they are equivalent (Theorems 2.6, 2.7, Chapter 2 in [12]). When a limit is placed on the numerical memory of UC by way of the number of registers it can use and the size of the numbers signifiable in them, the input and output registers included, UC immediately becomes an FMD and so a fortiori any device that UC is capable of simulating.

The separation of computability into the two overlapping categories, general and actual, is necessary for theoretical and practical reasons. A theoretical reason, generally accepted in classical computability theory, is that it is of no advantage to put any memory limitations on automata or on the a priori counts of unit time steps that automata may take to execute programs that implement functions in order to show that those functions are computable. Were it not the case, we would not be able to investigate what is computable in principle. Rogers [10] succinctly expresses this point of view:

> "[w]e thus require that a computation terminate after some finite number of steps; we do not insist on an a priori ability to estimate this number."

An implication of the above assumption is that an automaton, explicit or implicit, on which the said computation is executed has access to, literally, astronomical quantities of numerical memory. For a thought experiment, consider an automaton programmable in $\mathcal{L}$ of Chapter 2 of [7] that we used in Section 3.1, and let a program $P^j_{\mathcal{L}}(n)$, $n \in \mathbb{N}^+$, compute the G-number of the sequence $(1, \ldots, n)$, i.e., the function computed by $P^j_{\mathcal{L}}$ is $f(n) = [1, \ldots, n]$, as defined in (A8). Then, $f(n)$ is a primitive recursive function and, hence, computable in the general sense of Definition 2. Thus, $f(n)$ is signifiable for any $n \in \mathbb{N}^+$ on the automaton. In particular, if n is the Eddington number, i.e., $n = 10^{80} \in \mathbb{N}^+$, estimating the number of hydrogen atoms in the observable universe [14], there is a computation and, by implication, a variable in $P^j_{\mathcal{L}}$ to which the G-number of $(1, 2, \ldots, 10^{80})$ can be assigned.

The foregoing paragraph brings us to a practical reason for separating computability into the general and actual categories: it is of little use for an applied scientist who wants to implement a number-theoretic function f in a programming language L for an FMA D_j to know that f is *generally computable* and the L program can, therefore, compute, in principle, some characteristic of arbitrarily large natural numbers, e.g., the Eddington number. If no natural number greater than some $n \in \mathbb{N}$ is signifiable on D_j, the scientist must make

provisions in the program for the non-signifiable numbers in order to achieve feasible results with absolutely actually computable functions.

Theorem 1 shows that the computation of a trained FANN on a finite memory device can be packed into a unique natural number. Once packed, the natural number can be used as an archive, after a fashion, to look up natural numbers that correspond, in the bijective sense of the term, to the real vectors computed by the function A_z^z of an FANN N_z^j implemented on the device. The correspondence is such that for any signifiable $\vec{x}$, the output of N_z^j, i.e., $A_z^j(\vec{x}) = \vec{a}$, corresponds to the natural number $\tilde{a}$ computed by the primitive recursive function $\tilde{A}_z^j$, i.e., $\tilde{A}_z^j(\tilde{x}) = \tilde{a}$, and the input $\vec{x}$ corresponds to the natural number $\tilde{x}$. Thus, $A_z^j(\vec{x}) = \vec{a}$ iff $\tilde{A}_z^j(\tilde{x}) = \tilde{a}$. Furthermore, the function $\tilde{A}_z^j$ is computable in the general sense and is absolutely actually computable on any FMA where the natural number Ω_z^j is signifiable.

A correspondence established in Theorem 2 should be construed so that the uniqueness of Ω_z^j does not imply the uniqueness of A_z^j because the same function can be computed by different FANNs. What it implies is that, for any two different FANNs N_n^j and N_m^j, $n \neq m$ (e.g., different numbers of layers or different numbers of nodes in a layer or different activation functions or different weights), implemented on the same FMA D_j, $\Omega_n^j \neq \Omega_m^j$. However, it may be the case that $A_m^j(\vec{x}) = A_n^j(\vec{x})$ for any signifiable $\vec{x}$, and consequently, $\tilde{A}_m^j(\tilde{x}) = \tilde{A}_n^j(\tilde{x})$.

9. Conclusions

To differentiate between feedforward artificial neural networks and their functions as abstract mathematical objects and the realizations of these networks and functions on finite memory devices, we introduced the categories of general and actual computability. We showed that correspondences are possible between trained feedforward artificial neural networks on finite memory devices and classes of primitive recursive functions. We argued that there are theoretical and practical reasons why computability should be separated into these categories. The categories are overlapping in the sense that some functions belong in both categories.

Funding: This research received no external funding.

Data Availability Statement: No additional data are provided for this article.

Conflicts of Interest: The author declares no conflict of interest with himself.

Abbreviations

The following abbreviations are used in this article:

ANN	Artificial Neural Network
FANN	Feedforward Artificial Neural Network
FMA	Finite Memory Automaton or Automata
FMD	Finite Memory Device
G-number	Gödel Number
TM	Turing Machine
UTM	Universal Turing Machine
FSA	Finite State Automaton or Automata
PDA	Pushdown Automaton or Automata

Appendix A

Appendix A.1. Primitive Recursive Functions and Predicates

In this section, we define several functions shown to be primitive recursive in [7]. All smallcase variables in this section, e.g, x, y, z, t, n, and m, with and without subscripts, refer to natural numbers and the term *number* is synonymous with the term *natural number*.

The expression

$$(\exists t)_{\leq z} P(t, x_1, \ldots, x_n) \tag{A1}$$

is called the *bounded existential quantification* of the predicate P and holds iff $P(t, x_1, \ldots, x_n) = 1$ for at least one t such that $0 \leq t \leq z$. The expression

$$(\forall t)_{\leq z} P(t, x_1, \ldots, x_n) \tag{A2}$$

is called a *bounded universal quantification* of P and holds iff $P(t, x_1, \ldots, x_n) = 1$ for every t such that $0 \leq t \leq z$. If $P(t, x_1, \ldots, x_n)$ is a predicate and z is a number, then

$$x = \min_{t \leq z} \{P(t, x_1, \ldots, x_n)\} \tag{A3}$$

is called the *bounded minimalization* of P and defines the smallest number t for which P holds or 0 if there is no such number. It is shown in [7] that (1) the predicates $x = y$, $x \neq y$, $x < y$, $x > y$, $x \leq y$, $x \geq y$, and $x|y$, i.e., x divides y, are primitive recursive; (2) a finite logical combination of primitive recursive predicates is primitive recursive; and (3) if a predicate $P(\cdot)$ is primitive recursive, then so are its negation, its bounded minimalization, and its bounded universal and existential quantifications.

Let

$$x \dot{-} y = \begin{cases} x - y & \text{if } x \geq 0, \\ 0 & \text{if } x < y. \end{cases} \tag{A4}$$

The pairing function of natural numbers x and y, $\langle x, y \rangle : \mathbb{N} \to \mathbb{N}$, is

$$\langle x, y \rangle = z, \tag{A5}$$

where

$$\begin{aligned} &z = 2^x(2y + 1) \dot{-} 1; \\ &\gamma(d) \equiv \{2^d | (z + 1) \wedge (\forall c)_{\leq z+1} \{2^c \nmid (z + 1) \vee c \leq d\}\}; \\ &x = \min_{d \leq z+1} \gamma(d); \\ &y = \tfrac{1}{2}\left(\tfrac{z+1}{2^x} \dot{-} 1\right). \end{aligned}$$

For any number z, there are unique x and y such that $\langle x, y \rangle = z$. For example, if $z = 27$, then

$$\begin{aligned} x &= \min_{d \leq 28} \gamma(d) = 2; \\ y &= \tfrac{1}{2}\left(\tfrac{28}{2^2} \dot{-} 1\right) = 3; \\ \langle 2, 3 \rangle &= 2^2(2 \cdot 3 + 1) \dot{-} 1 = 27. \end{aligned}$$

The functions $l(z)$ and $r(z)$

$$\begin{aligned} l(z) &= \min_{x \leq z} \{(\exists y)_{\leq z} \{z = \langle x, y \rangle\}\} \\ r(z) &= \min_{y \leq z} \{(\exists x)_{\leq z} \{z = \langle x, y \rangle\}\} \end{aligned} \tag{A6}$$

return the left and right components of any number z so that $\langle l(z), r(z)\rangle = z$. Thus, if $z = 27 = \langle 2, 3\rangle$, then $l(z) = 2$, $r(z) = 3$.

The symbol p_n refers to the n-th prime, i.e., $p_1 = 2$, $p_2 = 3$, $p_3 = 5$, etc., and $p_0 = 0$, by definition. The primes are computed by the following primitive recursive function.

$$\pi(i) = p_i. \tag{A7}$$

Thus, $\pi(0) = 0$, $\pi(1) = 2$, $\pi(2) = 3$, $\pi(3) = 5$, $\pi(4) = 7$, $\pi(5) = 11$, etc. If $(a_1, \ldots, a_n)$ is a sequence of numbers, the function

$$[a_1, \ldots, a_n] = \prod_{i=1}^{n} \pi(i)^{a_i} \tag{A8}$$

computes the Gödel number (G-number) of this sequence. The G-number of the empty number sequence $()$ is 1. Thus, the G-number of $(3, 101, 7891, 1, 43)$ is $[3, 101, 7891, 1, 43] = 2^3 \cdot 3^{101} \cdot 5^{7891} \cdot 7^1 \cdot 11^{43}$.

If $x = [a_1, \ldots, a_n]$, the accessor function

$$(x)_i = \min_{t \le x}\{\neg\{\pi(i)^{t+1} | x\}\} \tag{A9}$$

returns the i-th element of x. Thus, if $x = [1, 7, 13]$, then $(x)_1 = 1$, $(x)_2 = 7$, $(x)_3 = 13$, and $(x)_j = 0$ for $j = 0$ or $j > 3$.

The length of a Gödel number x is the position of the last non-zero prime power in x. Specifically, if $x = [a_1, a_2, \ldots, a_n]$, its length is computed by the function $Lt(\cdot)$ defined as

$$Lt(x) = \min_{i \le x}\{(x)_i \neq 0 \wedge (\forall j)_{\le x}\{\{j > i\} \to \{(x)_j = 0\}\}\}. \tag{A10}$$

Thus, $Lt(540) = Lt([2, 3, 1]) = 3$. $Lt([a_1, \ldots, a_n]) = n$ iff $a_n \neq 0$, $[(x)_1, \ldots, (x)_n] = x$ when $Lt(x) = n$, and $Lt(0) = Lt(1) = 0$. $Lt([x_1, x_2, \ldots, x_n]) = Lt([x_1, x_2, \ldots, x_n, 0, \ldots, 0])$, where $x_n \neq 0$.

The function $\lfloor x/y \rfloor$ returns the integer part of the quotient x/y. Thus, $\lfloor 7/2 \rfloor = 3$, $\lfloor 2/5 \rfloor = 0$, $\lfloor 8/5 \rfloor = 1$, and $\lfloor x/0 \rfloor = 0$ for any number x.

Appendix A.2. Gödel Number Operators

The functions in this section or variants thereof were shown to be primitive recursive in [11]. The function

$$set(b, i, v) = \begin{cases} \left\lfloor \frac{b}{\pi(i)^{(b)_i}} \right\rfloor \cdot \pi(i)^v & \text{if } 1 \le i \le Lt(b) \wedge b > 1 \wedge v > 0, \\ 0 & \text{otherwise} \end{cases} \tag{A11}$$

assigns the value of the i-th element of the G-number b to v. Thus, if $b = [1, 2] = 2^1 3^2 = 18$, $i = 1$, and $v = 3$, then

$$set([1, 2], 1, 3) = \left\lfloor \frac{b}{\pi(1)^{(b)_1}} \right\rfloor \cdot \pi(1)^3 = \left\lfloor \frac{[1, 2]}{2^{([1,2])_1}} \right\rfloor \cdot 2^3 = \left\lfloor \frac{2^1 \cdot 3^2}{2^1} \right\rfloor \cdot 2^3 = [3, 2] = 72.$$

The function $cnt(\cdot)$ in (A12), where $s(t) = t + 1$ is one of the three initial functions defined in (11) and $(x)_i$ is defined in (A9), returns the count of occurrences of x in y. Thus, if $y = [1, 2, 1, 3]$, then $\mathtt{cnt}(1, y) = 2$. A convention in (A12) and other equations in this section is that the name of auxiliary functions end in "x".

$$cnt(x, y) = \begin{cases} cntx(x, y, Lt(y)) & \text{if } y > 1, \\ 0 & \text{otherwise.} \end{cases} \tag{A12}$$

$$\begin{array}{lcl} cntx(x,y,0) & = & 0, \\ cntx(x,y,t+1) & = & cntxx(x,y,t,cntx(x,y,t)). \end{array}$$

$$cntxx(x,y,t,c) = \begin{cases} 1+c & \text{if } (y)_{s(t)} = x, \\ c & \text{otherwise.} \end{cases}$$

If y is a G-number, then the predicate

$$x \in_g y \equiv cnt(x,y) \neq 0 \tag{A13}$$

holds if x is an element of y. Thus, $1 \in_g [3,4,1,5]$, but $1 \notin_g [3,4,2,5]$. The function

$$rap(x,y) = \begin{cases} y \cdot \{\pi(\mathtt{Lt}(y)+1)\}^x & \text{if } x > 0 \wedge y > 1 \wedge 0 \notin_g y, \\ 0 & \text{otherwise} \end{cases} \tag{A14}$$

appends x to the right of the rightmost element of y. Thus,

$$\begin{array}{lcl} rap(1,[1]) & = & [1] \cdot \{\pi(Lt([1])+1)\}^1 = [1] \cdot \{\pi(2)\}^1 \\ & = & 2^1 \cdot 3^1 = [1,1]; \\ rap(8,[2,3,5]) & = & [2,3,5] \cdot \{\pi(4)\}^8 \\ & = & [2,3,5,8]; \\ rap(5,set([10,3],1,2)) & = & rap(5,[2,3]) \\ & = & [2,3,5]. \end{array}$$

Let

$$\begin{array}{lcl} lc(x_1,x_2,0) & = & x_2, \\ lc(x_1,x_2,t+1) & = & rap((x_1)_{s(t)}, lc(x_1,x_2,t)). \end{array}$$

Then, the function

$$x \otimes_l y = \begin{cases} lc(x,y,Lt(x)) & \text{if } x > 1 \wedge y > 1 \wedge 0 \notin_g x \wedge 0 \notin_g y, \\ x & \text{if } x > 1 \wedge y = 1 \wedge 0 \notin_g x, \\ 0 & \text{otherwise} \end{cases} \tag{A15}$$

places all numbers in y, in order, to the left of the first number in x, while the function

$$x \otimes_r y = \begin{cases} y \otimes_l x & \text{if } x > 1 \wedge y > 1 \wedge 0 \notin_g x \wedge 0 \notin_g y, \\ x & \text{if } x > 1 \wedge y = 1 \wedge 0 \notin_g x, \\ 0 & \text{otherwise} \end{cases} \tag{A16}$$

places all numbers of y, in order, to the right of the rightmost number in x. We refer to the function in (A15) as *left concatenation* and to the function in (A16) as *right concatenation*. Thus, $[3,5] \otimes_l [7,11] = [7,11,3,5]$; $[3,5] \otimes_r [7,11] = [3,5,7,11]$; $[2,3] \otimes_l [1] = [1,2,3]$; $[2,3] \otimes_r [1] = [2,3,1]$.

Let

$$\begin{array}{lcl} gnx(l,u,k,0) & = & [l], \\ gnx(l,u,k,t+1) & = & gnxx(l,u,k,gnx(l,u,k,t),t); \end{array}$$

$$gnxx(l,u,k,z,t) = \begin{cases} z \otimes_r [l+s(t)k] & \text{if } l+s(t)k \leq u, \\ z & \text{otherwise.} \end{cases}$$

Then, for $l > 0$ and $u > 0$, the function

$$ggn(l,u,k) = \begin{cases} gnx(l,u,k,s(u \dot{-} l)) & \text{if } k > 0 \wedge (\exists t)_{\leq u}\{l + tk = u \wedge t > 0\}, \\ 0 & \text{otherwise.} \end{cases} \tag{A17}$$

generates a G-number whose numbers start at l and go to u in positive integer increments of k. Thus, $ggn(1,2,1) = [1,2]$; $ggn(1,2,2) = 0$; $ggn(1,3,1) = [1,2,3]$; $ggn(1,3,2) = [1,3]$; $ggn(1,3,3) = 0$. The abbreviation ggn stands for *generator of Gödel numbers*.

The function

$$asx(x,y) = \min_{t \leq Lt(y)} \{t > 0 \wedge x = l((y)_t)\} \tag{A18}$$

returns the smallest index t of $\langle i,j \rangle \in y$ such that $x = i$. Thus, if

$$y = [\langle 10,100 \rangle, \langle 20,200 \rangle, \langle 30,300 \rangle],$$

then $asx(10,y) = 1$, $asx(20,y) = 2$, $asx(30,y) = 3$. The function

$$\mathtt{asc}(x,y) = (y)_{\mathtt{asx}(x,y)} \tag{A19}$$

returns the pair from y at the index t returned by $asx(\cdot)$. Thus, if

$$y = [\langle 10,100 \rangle, \langle 20,200 \rangle, \langle 30,300 \rangle],$$

then

$$\begin{array}{rcccccl} asc(10,y) & = & (y)_{asx(10,y)} & = & (y)_1 & = & \langle 10,100 \rangle; \\ asc(20,y) & = & (y)_{asx(20,y)} & = & (y)_2 & = & \langle 20,200 \rangle; \\ asc(30,y) & = & (y)_{asx(30,y)} & = & (y)_3 & = & \langle 30,200 \rangle; \\ asc(13,y) & = & (y)_{asx(13,y)} & = & (y)_0 & = & 0. \end{array}$$

Appendix A.3. Examples of Ω *Numbers*

Let us abbreviate $G^{t,j}_{0,1}$ in (25) to G^t and consider the FANN in Figure 1. Let us assume that, as in Example 3, $R = \{0, 0.3, 0.6, 0.9, 1\}$, $I = \{1,2,3,4,5\}$ and $t = 2$, and

$$\begin{array}{rcl} G^{2,j}_{0,1} & = & G^2 \\ & = & [[1,1],[1,2],[1,3],[1,4],[1,5], \\ & & [2,1],[2,2],[2,3],[2,4],[2,5], \\ & & [3,1],[3,2],[3,3],[3,4],[3,5], \\ & & [4,1],[4,2],[4,3],[4,4],[4,5], \\ & & [5,1],[5,2],[5,3],[5,4],[5,5]]. \end{array}$$

In other words, G^2 is a G-number such that $[x_1,x_2] \in_g G^2$ iff $(x_1,x_2) \in I^2$. G^3, whose definition we omit for space reasons, is a G-number whose length is 125 such that $[x_1,x_2,x_3] \in_g G^3$ iff $(x_1,x_2,x_3) \in I^3$, e.g., $[1,2,3] \in_g G^3$ iff $(1,2,3) \in I^3$. We can compute Ω^e_i for the FANN N^j_z in Figure 1 as follows.

$$\Omega_0^0 = \left[\left\langle \left(G^2\right)_1, \mu\left(\alpha_0^0\left(\zeta^2\left(\left(G^2\right)_1\right), ()\right)\right)\right\rangle, \ldots, \left\langle \left(G^2\right)_{25}, \mu\left(\alpha_0^0\left(\zeta^2\left(\left(G^2\right)_{25}\right), ()\right)\right)\right\rangle\right];$$
$$\Omega_1^0 = \left[\left\langle \left(G^2\right)_1, \mu\left(\alpha_1^0\left(\zeta^2\left(\left(G^2\right)_1\right), ()\right)\right)\right\rangle, \ldots, \left\langle \left(G^2\right)_{25}, \mu\left(\alpha_1^0\left(\zeta^2\left(\left(G^2\right)_{25}\right), ()\right)\right)\right\rangle\right];$$
$$\Omega_0^1 = \left[\left\langle \left(G^2\right)_1, \mu\left(\alpha_0^1\left(\zeta^2\left(\left(G^2\right)_1\right), \vec{w}^1\right)\right)\right\rangle, \ldots, \left\langle \left(G^2\right)_{25}, \mu\left(\alpha_0^1\left(\zeta^2\left(\left(G^2\right)_{25}\right), \vec{w}^1\right)\right)\right\rangle\right];$$
$$\Omega_1^1 = \left[\left\langle \left(G^2\right)_1, \mu\left(\alpha_1^1\left(\zeta^2\left(\left(G^2\right)_1\right), \vec{w}^1\right)\right)\right\rangle, \ldots, \left\langle \left(G^2\right)_{25}, \mu\left(\alpha_1^1\left(\zeta^2\left(\left(G^2\right)_{25}\right), \vec{w}^1\right)\right)\right\rangle\right];$$
$$\Omega_2^1 = \left[\left\langle \left(G^2\right)_1, \mu\left(\alpha_2^1\left(\zeta^2\left(\left(G^2\right)_1\right), \vec{w}^1\right)\right)\right\rangle, \ldots, \left\langle \left(G^2\right)_{25}, \mu\left(\alpha_2^1\left(\zeta^2\left(\left(G^2\right)_{25}\right), \vec{w}^1\right)\right)\right\rangle\right];$$
$$\Omega_0^2 = \left[\left\langle \left(G^3\right)_1, \mu\left(\alpha_0^2\left(\zeta^3\left(\left(G^3\right)_1\right), \vec{w}^2\right)\right)\right\rangle, \ldots, \left\langle \left(G^3\right)_{125}, \mu\left(\alpha_0^2\left(\zeta^3\left(\left(G^3\right)_{125}\right), \vec{w}^2\right)\right)\right\rangle\right];$$
$$\Omega_1^2 = \left[\left\langle \left(G^3\right)_1, \mu\left(\alpha_1^2\left(\zeta^3\left(\left(G^3\right)_1\right), \vec{w}^2\right)\right)\right\rangle, \ldots, \left\langle \left(G^3\right)_{125}, \mu\left(\alpha_1^2\left(\zeta^3\left(\left(G^3\right)_{125}\right), \vec{w}^2\right)\right)\right\rangle\right].$$

We can compute individual elements of Ω_i^e. For example, since $(G^2)_{17} = [4,2]$,

$$\begin{aligned}
\left(\Omega_0^0\right)_{17} &= \left\langle \left(G^2\right)_{17}, \mu\left(\alpha_0^0\left(\zeta^2\left(\left(G^2\right)_{17}\right), ()\right)\right)\right\rangle \\
&= \left\langle [4,2], \mu\left(\alpha_0^0\left(\zeta^2\left([4,2]\right), ()\right)\right)\right\rangle \\
&= \left\langle [4,2], \mu\left(\alpha_0^0\left((0.9, 0.3), ()\right)\right)\right\rangle \\
&= \left\langle [4,2], \mu\left(0.9\right)\right\rangle \\
&= \left\langle [4,2], 4\right\rangle \in \mathbb{N}.
\end{aligned}$$

Since $(G^2)_{12} = [3,2]$,

$$\begin{aligned}
\left(\Omega_0^1\right)_{12} &= \left\langle \left(G^2\right)_{12}, \mu\left(\alpha_0^1\left(\zeta^2\left(\left(G^2\right)_{12}\right), \vec{w}^1\right)\right)\right\rangle \\
&= \left\langle [3,2], \mu\left(\alpha_0^1\left(\zeta^2\left([3,2]\right), \vec{w}^1\right)\right)\right\rangle \\
&= \left\langle [3,2], \mu\left(\alpha_0^1\left((0.6, 0.3), \vec{w}^1\right)\right)\right\rangle \\
&= \left\langle [3,2], z\right\rangle \in \mathbb{N},
\end{aligned}$$

where $z = \mu\left(\alpha_0^1\left((0.6, 0.3), \vec{w}^1\right)\right) \in I$. We know that $[2,3,4] \in_g G^3$ because $(2,3,4) \in I^3$. Thus, $(G^3)_t = [2,3,4]$, for $1 \leq t \leq 125$. Let us therefore assume, for the sake of this example, that $(G^3)_{35} = [2,3,4]$. Then,

$$\begin{aligned}
\left(\Omega_1^2\right)_{35} &= \left\langle \left(G^3\right)_{35}, \mu\left(\alpha_1^2\left(\zeta^3\left(\left(G^3\right)_{35}\right), \vec{w}^2\right)\right)\right\rangle \\
&= \left\langle [2,3,4], \mu\left(\alpha_1^2\left(\zeta^3\left([2,3,4]\right), \vec{w}^2\right)\right)\right\rangle \\
&= \left\langle [2,3,4], \mu\left(\alpha_1^2\left((0.3, 0.6, 0.9), \vec{w}^2\right)\right)\right\rangle \\
&= \left\langle [2,3,4], \mu\left(\alpha_1^2\left((0.3, 0.6, 0.9), \vec{w}^2\right)\right)\right\rangle \\
&= \left\langle [2,3,4], z\right\rangle \in \mathbb{N},
\end{aligned}$$

where $z = \mu\left(\alpha_1^2\left((0.3, 0.6, 0.9), \vec{w}^2\right)\right) \in I$.

Using (29), we can compute Ω_z^j for the FANN N_z^j in Figure 1 with the Ω numbers as

$$\Omega_z^j = \left[\left\langle 0, \left[\Omega_0^0, \Omega_1^0\right]\right\rangle, \left\langle 1, \left[\Omega_0^1, \Omega_1^1, \Omega_2^1\right]\right\rangle, \left\langle 2, \left[\Omega_0^2, \Omega_1^2\right]\right\rangle\right].$$

From Ω_z^j above, we can compute all $\tilde{\alpha}_i^e$ defined in (30) for N_z^j in Figure 1. For example, since $(G^2)_{12} = [3,2]$,

$$\begin{aligned}
\tilde{\alpha}_1^1([3,2]) &= r\left(asc\left(x,\left(r\left(asc\left(1,\Omega_z^j\right)\right)\right)_2\right)\right)\\
&= r\left(asc\left([3,2],\Omega_1^1\right)\right)\\
&= r\left(\left\langle\left(G^2\right)_{12},\mu\left(\alpha_1^1\left(\zeta^2\left(\left(G^2\right)_{12}\right),\vec{w}^1\right)\right)\right\rangle\right)\\
&= \mu\left(\alpha_1^1\left(\zeta^2\left(\left(G^2\right)_{12}\right),\vec{w}^1\right)\right)\\
&= \mu\left(\alpha_1^1\left(\zeta^2\left([3,2]\right),\vec{w}^1\right)\right)\\
&= \mu\left(\alpha_1^1\left(\left(0.6,0.3\right),\vec{w}^1\right)\right) \in I.
\end{aligned}$$

References

1. McCulloch, W.S.; Pitts, W. A logical calculus of the ideas immanent in nervous activity. *Bull. Math. Biophys.* **1943**, *5*, 115–133. [CrossRef]
2. Rumelhart, D.E.; Hinton, G.E.; Williams, R.J. Learning representations by back-propagating errors. *Nature* **1986**, 323, 533–536. [CrossRef]
3. Hornik, K. Approximation capabilities of multilayer feedforward networks. *Neural Netw.* **1991**, *4*, 251–257. [CrossRef]
4. Gripenberg, G. Approximation by neural networks with a bounded number of nodes at each level. *J. Approx. Theory* **2003**, *122*, 260–266. [CrossRef]
5. Guliyev, N.; Ismailov, V. On the approximation by single hidden layer feedforward neural networks with fixed weights. *Neural Netw.* **2019**, *98*, 296–304. [CrossRef] [PubMed]
6. Gödel, K. On formally undecidable propositions of *Principia Mathematica* and related systems I. In *Kurt Gödel Collected Works Volume I Publications 1929–1936*; Feferman, S., Dawson, J.W., Kleene, S.C., Moore, G.H., Solovay, R.M., van Heijenoort, J., Eds.; Oxford University Press: Oxford, UK, 1986.
7. Davis, M.; Sigal, R.; Weyuker, E. *Computability, Complexity, and Languages: Fundamentals of Theoretical Computer Science*, 2nd ed.; Harcourt, Brace & Company: Boston, MA, USA, 1994.
8. Kleene, S.C. *Introduction to Metamathematics*; D. Van Nostrand: New York, NY, USA, 1952.
9. Meyer, M.; Ritchie, D. The complexity of loop programs. In Proceedings of the ACM National Meeting, Washington, DC, USA, 14–16 November 1967; pp. 465–469.
10. Rogers, H., Jr. *Theory of Recursive Functions and Effective Computability*; The MIT Press: Cambridge, MA, USA, 1988.
11. Kulyukin, V. On primitive recursive characteristics of chess. *Mathematics* **2022**, *10*, 1016. [CrossRef]
12. Hopcroft, J.E.; Ullman, J.D. *Introduction to Automata Theory, Languages, and Computation*; Narosa Publishing Hourse: New Delhi, India, 2002.
13. Jurafsky, D.; Martin, J.H. *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*; Prentice-Hall, Inc.: Upper Saddle River, NJ, USA, 2000.
14. Eddington, A.S. The constants of nature. In *The World of Mathematics*; Newman, J.R., Ed.; Simon and Schuster: New York, NY, USA, 1956; Volume 2, pp. 1074–1093.