Subsymbolic Computation Theory for the Human Intuitive Processor

Paul Smolensky

Johns Hopkins University, Department of Cognitive Science

Abstract. The classic theory of computation initiated by Turing and his contemporaries provides a theory of effective procedures — algorithms that can be executed by the human mind, deploying cognitive processes constituting the *conscious rule interpreter*. The cognitive processes constituting the human *intuitive processor* potentially call for a different theory of computation. Assuming that important functions computed by the intuitive processor can be described abstractly as symbolic recursive functions and symbolic grammars, we ask which symbolic functions can be computed by the human intuitive processor, and how those functions are best specified — given that these functions must be computed using neural computation. Characterizing the automata of neural computation, we begin the construction of a class of recursive symbolic functions computable by these automata, and the construction of a class of neural networks that embody the grammars defining formal languages.

1 Effective Procedures vs. Intuitive Cognitive Processes

What is the set of functions computable by human intuition? Intuitive cognitive processes contrast with conscious rule interpretation [9], the mental processes enabling a clerk to take a list of English instructions and carry out a complex mathematical calculation by consciously interpreting those instructions one after another. Less mundanely, the *Entscheidungsproblem* also concerns the capabilities of conscious rule interpretation: Turing's 1936 paper [14] and other classic works provide a theory of the set of functions computable by human conscious rule interpretation, i.e., by effective procedures. Other notions of computation yielding different sets of computable functions derive from alternative conceptions of 'machine' in the sense of physical artifact (e.g., differential analyzer, optical or quantum computer).

But it is a natural biological system, the brain, that motivates the conception of computation of interest here. Within one stream of cognitive science [10], models of the brain and models of mental processes converged around 30 years ago upon a class of computational systems for modeling intuition, a class known variously as 'connectionist', 'parallel distributed processing', or 'abstract neural network' computational architectures [7].

Since the dawn of modern cognitive science in the middle of the 20th century, many cognitive scientists have treated intuition as formally identical with conscious rule interpretation, but inaccessible to consciousness [10]. Within this

conception, all higher mental processes, and not just conscious rule interpretation, can be modeled as effective procedures using classical computation. Newell [4], for example, famously identified the architecture of the human mind with that of a universal classical computer: what he called a 'physical symbol system'. This mainstream approach can be called the *symbolic paradigm*: such computation centers on symbols which individually serve syntactically as the tokens of manipulation and semantically as the elements interpretable in terms of the contents of consciousness.

The connectionist view, in contrast, models intuitive cognition within a different class of computational system, which we call *subsymbolic* [9] (after [**?**]). Section 2 characterizes subsymbolic computation via a class of automata. Sections 4 and 5 consider classes of subsymbolically-computable functions respectively derived from recursive equations and from formal languages.

The research program we consider (formally synopsized in [11]) is called *sub*symbolic rather than *non*symbolic because, unlike other research exploring connectionist computation (e.g. [2]), the working hypothesis is that — at a less fine-grained, more abstract level of description — the functions computed by intuitive processes are often well approximated by symbolic descriptions. The intuitive mental processes that actually *compute* these functions do not however admit a symbolic description: these processes require subsymbolic, connectionist descriptions. A central question then is: *which* ***symbolic*** *functions can the human intuitive processor compute?* The formal foundation for pursuing this question is laid in Section 3 and revisited in Section 6.

We return briefly in Section 7 to implications for cognitive science of some of the results we present, but until then we put aside the psychological motivation for connectionist computation — from theories of intuitive cognitive processes — and focus on the biological motivation. Specifically, we consider the formal capabilities of (a certain conception of) neural computation.

An important part of the mind-body problem is to connect the mental to the physical brain, and computational reduction offers the first prospect for carrying this out rigorously. This requires, however, that the computational architecture deployed has primitive operations, data, and combinators that a brain can provide. And this is what the connectionist architectures employed in the subsymbolic paradigm achieve, according to our current best understanding of the appropriate level of neural organization. To summarize the subsequent discussion: a mental concept is encoded in the activity of many neurons, not a single one; the activity of a neuron is a continuously varying quantity at the relevant level of analysis; combination of information from multiple neurons has an approximately linear character, while the consequences of this combination for neural activation has a non-linear character that imposes minimum and maximum levels. The primitives provided by the continuous architectures of the subsymbolic paradigm are within the capabilities of the brain—that the brain has *greater* complexity than assumed in these architectures does not compromise the subsymbolic paradigm's reduction of mental to neural computation.

While brain theory is far from achieving a settled state, this conception of continuous neural processing has generally displaced earlier notions according to which the relevant level of analysis was taken to be one where neural activations are binary (firing/not-firing), as assumed by the early discrete-computational network descriptions of the brain developed by Turing [15] as well as McCulloch and Pitts [3]. Similarly, early on, the search for the meaning of neural activation targeted individual cells, but recent years have seen an explosion of research in which neural meaning is sought by recording 'population codes' over hundreds of neurons, or patterns of aggregated activity over hundreds of thousands of neurons (using functional Magnetic Resonance Imaging).

2 Subsymbolic Automata

From an automata-theoretic perspective, computation is characterized by four fundamental properties [7, and references therein].

1. The tokens manipulated by primitive operations are 'subsymbolic': they reside at a level lower than that of the symbols of the symbolic paradigm. Each token is called the 'activation value' of a 'unit' (or 'neuron'). Tokens are related many-to-many to consciously accessible concepts: a single concept is realized by a particular combination of many subsymbolic tokens — a 'pattern of activity' — and a single token is simultaneously part of the realization of multiple concepts. This crucial property is called *distributed representation* of concepts.

2. The tokens are real numbers, of which a given automaton has a fixed number n: a state is an element of $\mathbb{R}^n$. The primitive operations are those of continuous numerical (as opposed to discrete symbolic) computation. These primitives include multiplication and exponentiation, but not determination of whether two tokens are of the same type, nor binding a value to a variable.

3. Each automaton is a continuously evolving dynamical system: differential equations play the role of algorithms. The primitive operations combine not sequentially, but in parallel, as all activation values simultaneously evolve in time according to the dynamical equations.

4. The dynamical equations are quasi-linear: if $a_k(t)$ denotes the k^{th} activation value at time t, then the dynamics is given by $da_k/dt = \sigma([\mathbf{W}\cdot\mathbf{a}(t)]_k) - a_k(t)$ where $\mathbf{a}(t)$ is the collection of all activation values at time t, $\{a_k(t)\}_{k=1}^n$. The state space of all possible $\mathbf{a}$ is assumed to be vector space, and $\mathbf{W}$ is a matrix of values $\{W_{kj}\}_{j,k=1}^n$ called 'connection weights' (or 'synaptic strengths'): W_{kj} is the weight of the 'connection' from unit j to unit k. $\mathbf{W}\cdot\mathbf{a}$ is the matrix-vector product: $[\mathbf{W}\cdot\mathbf{a}(t)]_k \equiv \sum_j W_{kj}a_j(t)$ is called the 'net input to unit k'. The possibly non-linear function $\sigma : \mathbb{R} \to \mathbb{R}$ can serve to keep activation values within certain limits; a common choice is the logistic function $\sigma(z) = 1/[1+e^{-z}]$.

The space of dynamical equations available to subsymbolic models is of course greater than that given here; what is most crucial for our purposes is that the representational states of the computational system form a vector space.[1]

[1] A fifth property—learning—is key to much connectionist modeling; the connection W_{kj} is typically 'learned' from statistical inference concerning the co-activation of

Such a subsymbolic system is often depicted as a network, a graph with the units as nodes (labelled by activation values), and the connections as links (labelled by weights).

An important special case is the *linear associator*. First, $\sigma(z) \equiv z$; then the equilibrium state satisfies $\mathbf{W} \cdot \mathbf{a} = \mathbf{a}$. Second, the network is *two-layer, feed-forward*: the units split into a set of input units (with activation vector $\mathbf{i}$) and a set of output units ($\mathbf{o}$), with non-zero weights only from input to output units. Then the equilibrium state satisfies $\hat{\mathbf{W}} \cdot \mathbf{i} = \mathbf{o}$ and the network is equivalent to a linear transformation (given by the matrix $\hat{\mathbf{W}}$ of input-to-output weights) from the vector space of input vectors $\mathbf{i}$ to the vector space of output vectors $\mathbf{o}$.

The implications of distributed representation — of taking vectors in $\mathbb{R}^n$ as the basic data type of the computational architecture — are many [13]. The basic operations of continuous mathematics provided by vector space theory now do the fundamental computational work. Consider, for example, language processing, a system of intuitive processes that is critical for cognitive science. To preview the remainder of the article: Instead of stringing together two conceptual-level symbols to form $\texttt{Frodo}_{\texttt{subject}}\ \texttt{lives}_{\texttt{predicate}}$, we add together two vectors, the first encoding 'Frodo as subject' and the second 'lives as predicate'. The vector encoding 'Frodo as subject' results from taking a vector encoding Frodo and a vector encoding 'subject' and combining them with a vector operation, the tensor product. The basic mapping operation of vector space theory, linear transformation, provides the core of mental processing. Vector similarity, defined using the vector inner product, serves to model conceptual similarity, a central notion of psychological theory. And a notion from dynamical systems theory proves to have major implications as well: the differential equations governing many subsymbolic models have the property that as time progresses, a quantity called the *Harmony* steadily increases. Harmony can be interpreted as a numerical measure of the *well-formedness* of the network's activation vector with respect to the weights. The objective of network computation is to produce the representation (vector) that is maximally well-formed — *optimal*. And Harmony turns out to bear a deep relation to well-formedness as defined by a grammar.

We now examine some of the details behind this summary, and proceed to consider some of the implications for intuitive computation in the human mind.

3 From Symbolic to Vectorial Representations

At the foundation of the subsymbolic (but not the nonsymbolic) connectionist program is the reduction of symbolic conceptual-level mental representations to more fine-grained vectorial subconceptual representations. The vectorial embedding of a set of symbol structures S proceeds as follows [13, ch. 5]. Throughout, we use the example of binary trees with node labels from an alphabet A.

First, each symbol structure is mapped to a set of (symbolic) *filler/role bindings* via a *filler-role decomposition*. For example, a particular role r_x might be

units k and j during 'training', in which example input-output pairs of a target function are presented. However our concern here is not learnability but computability.

identified with a node position x (e.g., left child of right child of root, denoted by the bit string $x = 01$); among the bindings for a particular tree $s \in S$, this role would be bound to a particular symbol $f \in A$ just in case the symbol f labels the node position x in the tree s; this binding is written f/r_x and the set of all bindings of s is $\beta(s)$.

Next, a vector space V_F called the *filler (vector) space* is selected, along with an injection ψ_F from the set of symbolic fillers to V_F. Similarly, a *role (vector) space* V_R is selected, along with an injection ψ_R from the set of symbolic roles to V_R. Then V_F and V_R are combined by the tensor product to form the actual representational (vector) space for trees $V_S \equiv V_F \otimes V_R$.[2]

Finally, the filler-role decomposition and vectorial realizations are combined: the vector realization of s, $\psi_S(s) \in V_S$, is the *sum* of the vector realizations of the filler-role bindings of s, each of which uses the tensor product to bind together the encodings of the filler and the role:

$$\psi_S(s) = \sum_{f/r \in \beta(s)} \psi_F(f) \otimes \psi_R(r)$$

Henceforth, ψ_S will be abbreviated ψ.

For binary trees, we use a role realization obeying the recursive requirements

$$\psi_R(r_{0x}) = \psi_R(r_0) \otimes \psi_R(r_x), \quad \psi_R(r_{1x}) = \psi_R(r_1) \otimes \psi_R(r_x)$$

where $0x$ denotes the concatenation of the bit 0 with the bit-string x. Thus, e.g., $\psi_R(r_{010}) = \psi_R(r_0) \otimes \psi_R(r_1) \otimes \psi_R(r_0)$; ψ_R is entirely determined by $\psi_R(r_0)$ and $\psi_R(r_1)$, the vectors in $V_R^{(1)}$ realizing the roles 'left-' and 'right-child of root'. These two vectors need to be linearly independent, so $\dim(V_R^{(1)}) \geq 2$; for concreteness, we assume it equals 2. Roles for positions at tree depth d are realized with d-fold tensor products; the total vector space V_R is the direct sum of spaces $V_R^{(d)}$, $d = 0, \ldots, \infty$ for all tree depths: $V_R = \bigoplus_{d=0}^{\infty} V_R^{(d)}$, with $\dim(V_R^{(d)}) = 2^d$. Likewise, $V_S = \bigoplus_{d=0}^{\infty} V_S^{(d)} = \bigoplus_{d=0}^{\infty} \left[V_F \otimes V_R^{(d)} \right] = V_F \otimes V_R$. In the case $d = 0$, $V_R^{(0)}$ is a 1-dimensional vector space so $V_S^{(0)} \equiv V_F \otimes V_R^{(0)}$ is isomorphic to V_F. A tree s of depth $d = 0$ is simply a symbol in A; in this case $\mathtt{atom}(s) = \mathsf{T}$ (otherwise $\mathtt{atom}(s) = \mathsf{F}$). The identity transformations on infinite-dimensional V_S and V_R, 2-dimensional $V_R^{(1)}$, and 1-dimensional $V_R^{(0)}$ are respectively denoted $\mathbb{1}_S$, $\mathbb{1}_R$, $\mathbf{1}_R$ and 1_R. The identity transformation on finite-dimensional V_F is $\mathbf{1}_F$.

[2] Some definitions: Given bases $\{\hat{\mathbf{f}}_u\}_{u=1}^n$ for V_F and $\{\hat{\mathbf{r}}_v\}_{v=1}^m$ for V_R, $\{\hat{\mathbf{f}}_u \otimes \hat{\mathbf{r}}_v\}$ is a basis for the $n \times m$-dimensional space V_S. The tensor product of vectors $(\mathbf{f}, \mathbf{r}) \mapsto \mathbf{f} \otimes \mathbf{r}$ mapping $V_F \times V_R$ to V_S is bilinear, i.e., linear in each of $\mathbf{f}$ and $\mathbf{r}$ independently. So if $\{\mathrm{f}_u\}$ and $\{\mathrm{r}_v\}$ are respectively the elements of $\mathbf{f} \in V_F$ and $\mathbf{r} \in V_R$ with respect to bases $\{\hat{\mathbf{f}}_u\}$ and $\{\hat{\mathbf{r}}_u\}$, then the elements of $\mathbf{f} \otimes \mathbf{r} \in V_S$ with respect to the basis $\{\hat{\mathbf{f}}_u \otimes \hat{\mathbf{r}}_v\}$ are $\{\mathrm{f}_u \mathrm{r}_v\}$. The direct sum of vector spaces U and V, $U \oplus V$, is $U \times V$ with the obvious linear operations: $\alpha(\mathbf{u}, \mathbf{v}) + \beta(\mathbf{u}', \mathbf{v}') \equiv (\alpha\mathbf{u} + \beta\mathbf{u}', \alpha\mathbf{v} + \beta\mathbf{v}')$, $[\mathbf{A} \oplus \mathbf{B}] \cdot (\mathbf{u}, \mathbf{v}) \equiv (\mathbf{A} \cdot \mathbf{u}, \mathbf{B} \cdot \mathbf{v})$; $\dim(U \oplus V) = \dim(U) + \dim(V)$.

4 Linearly Computable Functions by Recursion

We will say that a function $f : S \to S$ is *linearly computable* (i.e., computable by a linear associator network) if there is a linear transformation $\mathbb{W}_f : V_S \to V_S$ such that $\psi_S \circ f = \mathbb{W}_f \circ \psi_S$, i.e., the following diagram commutes:

$$\begin{array}{ccc} S & \xrightarrow{f} & S \\ {\scriptstyle\psi}\downarrow & & \downarrow{\scriptstyle\psi} \\ V_S & \xrightarrow{\mathbb{W}_f} & V_S \end{array}$$

This section discusses work that begins to characterize the set of linearly computable symbolic functions over binary trees (cf. [1]). We proceed from the base case of functions over depth-0 trees (the symbols in A), to the closure under composition of the primitive tree-role construction and decomposition functions, to functions defined by a kind of primitive recursion.

We start then with the set $\mathcal{B}$ of functions f_g that map every symbol $\mathtt{A} \in A$ in a tree into the symbol $g(\mathtt{A})$, without changing which node it labels, where $g : A \to A$ is some (partial) function.

The functions in $\mathcal{B}$ are linearly computable. In the case of interest, the vectors in V_F realizing the symbols $\{\psi(a)|a \in A\}$ are linearly independent, hence form a basis for V_F (adding additional vectors if necessary); let its dual basis be $\{\psi^+(a)|a \in A\}$ (i.e., $[\psi^+(a)]^T\psi(a')$ is 1 if $a = a'$ otherwise 0). Then $f_g \in \mathcal{B}$ is realized by $\mathbb{W}_f \equiv \sum_{a\in A} \psi(g(a))[\psi^+(a)]^T \otimes \mathbb{1}_R$.

We next form $\mathcal{F}$, the smallest set of functions that (i) includes the functions in $\mathcal{B}$, (ii) is closed under composition with functions in $\mathcal{B}$, and (iii) is closed under composition with the primitive tree functions: $\mathtt{cons}$, which constructs a new binary tree $\mathtt{cons}(p, q) \equiv s$ with left sub-tree p and right sub-tree q; $\mathtt{ex}_0$ and $\mathtt{ex}_1$, which extract the left sub-tree $\mathtt{ex}_0(s) = p$ and right sub-tree $\mathtt{ex}_1(s) = q$ from $s = \mathtt{cons}(p, q)$.

The primitive tree functions are all linearly computable [13, ch. 8]. There exist linear transformations $\mathbb{W}_f$ on V_S satisfying, for $i = 0, 1$:

$$\psi \circ \mathtt{cons}(p, q) = \mathbb{W}_{\mathtt{cons}_0} \cdot \psi(p) + \mathbb{W}_{\mathtt{cons}_1} \cdot \psi(q) \qquad \psi \circ \mathtt{ex}_i(s) = \mathbb{W}_{\mathtt{ex}_i} \cdot \psi(s)$$

In fact any function f in $\mathcal{F}$ is linearly computable, by a linear transformation of the form

$$\mathbb{W}_f = \mathbb{1} \otimes \underline{\mathbf{W}}_f \quad (\text{e.g, } \mathbb{W}_{\mathtt{cons}_0} = \mathbb{1} \otimes \underline{\mathbf{W}}_{\mathtt{cons}_0})$$

where $\mathbb{1}$ satisfies the recursion relation $\mathbb{1} = 1_R + \mathbb{1} \otimes \mathbf{1}_R$ [13, ch. 8]. The solution is $\mathbb{1}_R$, which can be written (defining the sum-of-powers operator $\wp$):

$$\mathbb{1}_R = \wp(\mathbf{1}_R) \equiv \bigoplus_{d=1}^{\infty} (\mathbf{1}_R)^{\otimes d} \equiv 1_R \oplus \mathbf{1}_R \oplus (\mathbf{1}_R \otimes \mathbf{1}_R) \oplus (\mathbf{1}_R \otimes \mathbf{1}_R \otimes \mathbf{1}_R) \oplus \cdots$$

$\underline{\mathbf{W}}_f$ is a transformation dependent on f; the compositional map $f \mapsto \underline{\mathbf{W}}_f$ is recursively defined by requirements such as

$$f = \mathtt{cons}(f', f'') \Rightarrow \underline{\mathbf{W}}_f = \underline{\mathbf{W}}_{\mathtt{cons}_0} \cdot \underline{\mathbf{W}}_{f'} + \underline{\mathbf{W}}_{\mathtt{cons}_1} \cdot \underline{\mathbf{W}}_{f''}$$

An example of a simple function not in $\mathcal{F}$ is `reverse`, which reverses the left and right children of every node. This function is defined by the recursion

$$\texttt{reverse}(s) = \begin{cases} s & \text{if } \texttt{atom}(s) \\ \texttt{cons}(\texttt{reverse}(\texttt{ex}_1(s)), \texttt{reverse}(\texttt{ex}_0(s))) & \text{otherwise} \end{cases}$$

`reverse` is linearly computable by a transformation $\mathbf{1}_F \otimes \mathbb{P}$ where $\mathbb{P}$ satisfies the recursion

$$\mathbb{P} = 1_R \oplus \mathbb{W}_{\texttt{cons}_0} \cdot \mathbb{P} \cdot \mathbb{W}_{\texttt{ex}_1} + \mathbb{W}_{\texttt{cons}_1} \cdot \mathbb{P} \cdot \mathbb{W}_{\texttt{ex}_0}$$

A solution is

$$\mathbb{P} = \wp(\underline{\mathbf{W}}_{\texttt{reverse}}) \quad \text{where} \quad \underline{\mathbf{W}}_{\texttt{reverse}} \equiv \underline{\mathbf{W}}_{\texttt{cons}_0} \cdot \underline{\mathbf{W}}_{\texttt{ex}_1} + \underline{\mathbf{W}}_{\texttt{cons}_1} \cdot \underline{\mathbf{W}}_{\texttt{ex}_0}$$

More generally, consider the recursion

$$f(s) = \begin{cases} g(s) & \text{if } \texttt{atom}(s) \\ h(f(\texttt{ex}_0(s)), f(\texttt{ex}_1(s))) & \text{otherwise} \end{cases}$$

Suppose g and h are linearly computable:

$$\psi \circ g(s) = \mathbf{G} \cdot \psi(s) \quad \text{and} \quad \psi \circ h(p, q) = \mathbb{H}_0 \cdot \psi(p) + \mathbb{H}_1 \cdot \psi(q)$$

Then f will be linearly computable by a linear transformation $\mathbb{W}_f$ if a solution exists to the following recursion:

$$\mathbb{W}_f = \mathbf{G} \oplus \mathbb{H}_0 \cdot \mathbb{W}_f \cdot \mathbb{W}_{\texttt{ex}_0} + \mathbb{H}_1 \cdot \mathbb{W}_f \cdot \mathbb{W}_{\texttt{ex}_1}$$

A solution is

$$\mathbb{W}_f = \mathbf{G} \circledast \sum_{d=0}^{\infty} \mathbb{M}^{\circledast d} \quad \text{where} \quad \mathbb{M} \equiv \mathbb{H}_0 \circledast \mathbb{W}_{\texttt{ex}_0} + \mathbb{H}_1 \circledast \mathbb{W}_{\texttt{ex}_1}$$

Here $\circledast$ is a contracted tensor product operator the definition of which requires more detail concerning the direct-sum structure of V_S than is possible here.

5 Grammars and Optimization

Alongside recursive function theory is another classic approach to specifying functions: via formal languages, defined by grammars consisting of string rewriting rules (or productions). This approach has achieved considerable success in characterizing intuitive mental processing of natural language (including that underlying human performance in Turing's Imitation Game [16]); this work culminates in the theory of 'universal grammar', which formally characterizes what the grammatical systems of human languages share and exactly how they may differ. (We will discuss universal grammar in Section 7.)

The relation to subsymbolic computation arises because *grammatical* well-formedness can be realized as a kind of *connectionist* well-formedness called *Harmony* (or negative 'energy') [13, ch. 9, and references therein]. Harmony arises as

a quantity that is steadily increased by the dynamical processing within an important class of network which includes those quasi-linear networks having both a symmetric connection matrix ($W_{kj} = W_{jk}$) and a monotonically increasing function σ.[3] Network dynamics can be interpreted as computing a maximal-Harmony representation — *locally* maximal, that is: at an equilibrium of the dynamics, no infinitesimal state change can produce higher Harmony. To compute *global* Harmony maxima, stochastic differential equations can be used to define the network dynamics (achieving probabilistic convergence as $t \to \infty$).

Can the notion of well-formedness provided by grammars that are defined by string rewriting rules be realized as network Harmony? To investigate this question, a mediating notion — *Harmonic Grammar* — proves useful [13, ch. 6]. A Harmonic Grammar $H_\mathcal{G}$ assigns to any tree s a numerical well-formedness value $H_\mathcal{G}(s)$; the language specified by $H_\mathcal{G}$ is the set of trees with maximal Harmony: $\mathcal{L}_{H_\mathcal{G}} \equiv \text{argmax}_{s \in S} H_\mathcal{G}(s)$.

A Harmonic Grammar $H_\mathcal{G}$ is realized in a network $\mathcal{N}$ iff for all $s \in S$, $H_\mathcal{G}(s) = H_\mathcal{N}(\psi(s))$, where $H_\mathcal{N}$ is the connectionist Harmony of $\mathcal{N}$. It turns out that the language $\mathcal{L}_\mathcal{G}$ generated by any rewrite-rule grammar $\mathcal{G}$ can be specified by a Harmonic Grammar $H_\mathcal{G}$ (that is, $\mathcal{L}_\mathcal{G} = \mathcal{L}_{H_\mathcal{G}}$), and $H_\mathcal{G}$ can be constructed directly from the rules of $\mathcal{G}$ [13, ch. 10].

Further, when $\mathcal{G}$ is a context-free grammar in Chomsky Normal Form (hence with binary derivation trees), our vectorial realization of binary trees can be used to construct a network $\mathcal{N}_\mathcal{G}$ that realizes $H_\mathcal{G}$ [13, ch. 8]. This $H_\mathcal{G}$ has the form $H_\mathcal{G}(s) = \sum_{b,b' \in \beta(s)} H_{b,b'}$ where for each pair of filler/role bindings (b, b') of s, the number $H_{b,b'}$ encodes the grammatical well-formedness of b and b' co-existing in s. Typically in linguistics $H_{b,b'}$ is non-positive, with $|H_{b,b'}|$ interpreted as the strength in the grammar $H_\mathcal{G}$ of the *constraint*: 'do not combine b and b' in the same structure'.[4]

When $\mathcal{L}_\mathcal{G} = \mathcal{L}_{H_\mathcal{G}}$, any tree s_0 generated by $\mathcal{G}$ is realized by a vector $\psi(s_0)$ that has maximal network Harmony $H_{\mathcal{N}_\mathcal{G}}(\psi(s_0))$ *among the vectors* $\{\psi(s) | s \in S\}$ *realizing trees* (this discrete subset of V_S, the image of S under ψ, will be called "the grid").

Within V_S there are, however, vectors $\mathbf{v}^*$ off the grid which have higher Harmony than those grid states realizing grammatical trees: such a vector $\mathbf{v}^*$ does not realize a tree; rather it realizes a pseudo-tree with weighted blends of symbols bound to tree positions (e.g., instead of $\mathtt{A}/r_x$ there might be $[0.5\mathtt{A} - 0.1\mathtt{B}]/r_x$). Such non-grid states are optimal within the vector space because conflicts between the constraints encoded in H entail that optima constitute compromises

[3] The Harmony of the state vector $\mathbf{a}$ of a network $\mathcal{N}$ with weight matrix $\mathbf{W}$ is $H_\mathcal{N}(\mathbf{a}) \equiv \frac{1}{2}\sum_{kj} a_k W_{kj} a_j + \sum_k h(a_k)$, $h(a) \equiv -\int_0^a \sigma^{-1}(a')da'$

[4] E.g., if b is a singular subject (like 'Frodo') and b' a plural verb form (like 'live'), $h_{\text{agree}} = H_{b,b'} < 0$ is a grammatical penalty for a verb failing to agree in number with its subject. Adding $h_{\text{agree}} \psi(b)^+ [\psi(b')^+]^T$ to the weight matrix $\mathbf{W}$ causes the Harmony of any structure containing b and b' (e.g., 'Frodo live') to decrease by $|h_{\text{agree}}|$.

that interpolate between the alternative discrete states favored by the conflicting constraints [12].

6 Quantization

In order for the network to produce bona fide trees as output, another component can be added to the dynamics of $\mathcal{N}_{\mathcal{G}}$. This (deterministic) *quantization* dynamics produces an attractor at all and only the vectors on the grid. As the computation proceeds, this quantization dynamics comes to overpower the stochastic Harmony-optimization dynamics, so that the final output will be an attractor — a grid state [12]. In a range of (very simple) applications to modeling human language processing, simulations show these networks to be capable of reliably computing the vectorial realizations of symbolic states that are globally-optimal among grid states. Further, when forced to terminate too quickly, these simulations show that the probability of outputting some error E appropriately declines with its Harmony $H(E)$: not only do the global optima of the Harmonic Grammar model correct linguistic competence, the Harmonic Grammar's evaluation of sub-optimal states models the errors characteristic of human linguistic performance.

7 Optimization and Universal Grammar

The shift from a rewrite-rule grammar $\mathcal{G}$ to a Harmonic Grammar $H_{\mathcal{G}}$ for specifying a language may seem minor, but in fact it constitutes a fairly radical reconception: from grammars as generators to grammars as evaluators, from grammatical as generated to grammatical as optimal. Does this afford a better formal system for capturing the functions computed by human linguistic intuition? There is reason to believe that it does.

Soon after the appearance of Harmonic Grammar came the empirical discovery that in human grammars, it is often the case that the strength of each constraint exceeds the combined strengths of all weaker constraints. The constraints can then be considered to form a *strict-domination hierarchy*, with each constraint having absolute priority over all weaker constraints; while numerical strengths are needed for the reduction to neural computation, only the priority ranking is needed to determine optimality. This is *Optimality Theory* [5,6], which has proved to make a significant contribution to the theory of universal grammar via its central principle: the grammars of all human languages consist in the same constraints; grammars may differ only in how the constraints are ranked into a strict-domination hierarchy. This means that all languages share the same well-formedness criteria: they differ only in which criteria take priority in cases of conflict. The empirical successes are cases when what is universally shared by human languages are preferences that outputs of the grammar should respect, as opposed to processes that generate those outputs; since rewrite-rules characterize grammatical knowledge as generative procedures, for the purposes

of universal grammar, it proves advantageous that Optimality Theory characterizes grammatical knowledge as preferences over outputs [8]. In the case of language, at least, specifying computations via optimization has been shown to provide insight into the nature of the functions computed by human intuition.

References

1. Kimoto, M., Takahashi, M.: On computable tree functions. In: He, J., Sato, M. (eds.) Advances in Computing Science – ASIAN 2000 6th Asian Computing Science Conference, Lecture Notes in Computer Science, vol. 1961, pp. 273–289. Springer-Verlag, Penang, Malaysia (November 25-27 2000)
2. McClelland, J., Botvinick, M., Noelle, D., Plaut, D., Rogers, T., Seidenberg, M., Smith, L.: Letting structure emerge: connectionist and dynamical systems approaches to cognition. Trends in Cognitive Sciences 14(8), 348–356 (2010)
3. McCulloch, W., Pitts, W.: A logical calculus of the ideas immanent in nervous activity. Bulletin of Mathematical Biology 5(4), 115–133 (1943)
4. Newell, A.: Physical symbol systems. Cognitive Science 4(2), 135–183 (1980)
5. Prince, A., Smolensky, P.: Optimality Theory: Constraint interaction in generative grammar. Blackwell (1993/2004)
6. Prince, A., Smolensky, P.: Optimality: From neural networks to universal grammar. Science 275(5306), 1604–1610 (1997)
7. Rumelhart, D., McClelland, J., the PDP Research Group: Parallel distributed processing: Explorations in the microstructure of cognition. Vol. 1, Foundations. MIT Press: Cambridge, MA (1986)
8. Rutgers Optimality Archive: http://roa.rutgers.edu
9. Smolensky, P.: On the proper treatment of connectionism. Behavioral and Brain Sciences 11(01), 1–23 (1988)
10. Smolensky, P.: Cognition: Discrete or continuous computation? In: Cooper, S., van Leeuwen, J. (eds.) Alan Turing — His work and impact. Elsevier (2012)
11. Smolensky, P.: Symbolic functions from neural computation. Philosophical Transactions of the Royal Society – A: Mathematical, Physical and Engineering Sciences, in press (2012)
12. Smolensky, P., Goldrick, M., Mathis, D.: Optimization and quantization in gradient symbol systems: A framework for integrating the continuous and the discrete in cognition. Cognitive Science, in press (2012)
13. Smolensky, P., Legendre, G.: The harmonic mind: From neural computation to Optimality-Theoretic grammar, vol. 1: Cognitive architecture, vol. 2: Linguistic and philosophical implications. MIT Press: Cambridge, MA (2006)
14. Turing, A.M.: On computable numbers, with an application to the Entscheidungsproblem. Proceedings of the London Mathematical Society 42, 230–265 (1936)
15. Turing, A.M.: Intelligent machinery: A report by A. M. Turing. National Physical Laboratory (1948)
16. Turing, A.M.: Computing machinery and intelligence. Mind 59(236), 433–460 (1950)