

Neural Computation as an Extension of Information Theory

Robert Sneddon

Part 1: Noiseless, Deterministic Computation

The concept of a "Neural Code" is based on the assumption that the brain performs computations. However, this notion of neural computation is generally stated in a vague manner or not at all. In the present work, this assumption is stated in a formal, quantitative manner. This yields an implicit definition of the neural code. Specifically, any form of the neural code must satisfy basic quantitative conditions. These conditions take the form of functional equations which relate the input of a neural circuit to its output. These functional equations are applied to basic neural circuitry, i.e., an integrate and fire neuron as well as a thalamic relay neuron. The predicted behavior of thalamic relay neurons was confirmed, as well as that of an integrate and fire neuron. This suggests that this theory is an accurate quantitative formulation of basic neural computation.

Introduction

In modern times, the paradigm of the brain as a biological computer is almost universal, see e.g., Churchland and Sejnowski (1992). This approach is characterized by two basic forms of analysis: information theory and neural coding. Together, these approaches seek to answer the questions: “How much information is transmitted,” and “What is the code that the information is transmitted in?”

Neural Computation as an Extension of Information Theory

In the present study the concept of information encoding and transmission will be extended to computational coding and processing. Thus, we will extend the question “How much information is transmitted,” to “And what has been derived from this information?” We postulate that the nervous system processes information in a manner that aids the organism. Specifically, the informative portions of the information are computed and transponded. They may be compared to previous experience and they might be encoded into memories for future reference. However, in all cases, the information is not just transmitted, it is *transformed*. This transformation is the essence of what we call computation. However, to tie this formulation of computation in with traditional computational theories, traditional definitions of computation must be examined.

Defining computation

The original concept of computation can be traced back to Hilbert’s notion of formal operations (Detlefsen, 1986). A formal operation or the use of “mathematical instrumentalism” is a rule-based procedure conducted on tokens, e.g., mathematical symbols. This notion of formal operation was extended by Turing to the ‘Turing machine,’ (Hennie, 1977). The Turing machine conducts formal operations according to a memory (or program input) tape and writes output onto another tape. This set of formal operations is an instantiation of an algorithm which in principle, will perform a desired computation.

Thus, in terms of a Turing machine, computation consists of three basic parts. They are the input codes/commands, the orderly processing of the codes by some mechanism such as the brain, and output codes/messages. This basic computational approach has burgeoned into a rich and deep theory. A theory that assumes knowledge of the computing mechanism.

Thus, Turing and other similar computational theorists concentrated their research on the question, “What will the output be, given our knowledge of the computational mechanism and the input?” A classic example of this is the unsolved halting problem (Davis, 1967).

At a more practical level, many professionals concern themselves with the question, “What must the input code be if we know what the computational mechanism is and we know what output we want?” In other words, many people attempt to write an appropriate program. This program, in principle, will yield the desired output. Thus, there are two basic approaches to understanding computation at this time. They are:

1. Assuming knowledge of the input and the computation mechanism, what is the output?

2. Assuming knowledge of the computation mechanism and the output, what is the input?

This leaves a third possible question:

3. Assuming knowledge of the input and the output, what can we say about the computation mechanism?

This question is rarely if ever asked because in almost all situations we know what the computation mechanism is. It may be something simple like a light meter or a speedometer or something quite complex like a calculator or a computer. In almost all cases, we know, in principle, exactly what the computation mechanism is. However, there is one important exception. That exception is the brain.

An Input-to-Output Computational Analysis of the Brain

A basic input to output analysis of neural processing is not a new idea. For example, Rieke, Warland, Steveninck, and Bialek (1997) has performed an excellent information-theoretic analysis of physical stimuli. Others have performed a Wiener series analysis of the early stage processing of auditory stimuli see, e.g., Eggermont (1993). However, these studies do not explicitly examine output as a *computational* function of input. To do so, computation, in its most fundamental form, must be defined.

A Basic Definition of Computation

1. An input code.
2. Classification/Categorization of the input code, equivalently, a look-up table of the input code.
3. An output code denoting the categorized input.

There are at least two possible additions to this definition. However they are not necessary for computation:

4. Translation, i.e., changing the “language” of the input code to a different one for the output code.
5. Memory/Adaptation. For the purposes of this exposition it is worthwhile to examine examples of each of these parts of the definition of computation.

1. An input code. A code that the computer (correctly) performs categorization on. A simple, but non-obvious example is a light intensity meter. Here, the input code is the time-varying function of light intensity. Some neural examples are a spiking “rate” code, and a timing code. Non-neural examples are computer machine code, Morse code etc.

2. Categorization/Classification. A mapping which classifies input into categories. Stated in another way, a look-up table. It is important to note that the actual computation generally doesn’t use a look up table. However the algorithm

used performs the equivalent of a look-up table. A simple example is a speedometer which maps the rotational speed of the front axle of a vehicle to a number that denotes the speed of the vehicle. The speed of the car is “looked-up” in a table relating angular velocity of an axle to linear speed. Other forms of classification are attractor networks, hierarchical clustering etc. This categorization/classification may or may not use a memory.

Categories (look-up table entries) are generally ordered, either explicitly or implicitly. This ordering can be numerical. In the speedometer example, a 70 mph speed is higher than a 60 mph speed. This ordering is explicit: the pointer on a speedometer dial rotates to a larger angle for a larger speed. A useful example in neuroscience is a mean rate code. A spike train with a mean rate of 10 Hz is higher than one with a mean rate of 5 Hz. This sort of ordering is implicit - the use of numerical categories immediately yields a greater than/less than order.

3-4. Output codes and translations. Sometimes the output codes are a translation of the input code. For example, Morse code is translated into letters from the Latin alphabet. Another example is a speedometer. A speedometer receives input from a car’s axle. However, its output is a number that denotes miles per hour. On the other hand, the input and output code are often in the same language. For example, neural circuits have spike trains as both input and output. (Other considerations are neurotransmitters etc.) A tape recorder/player takes sound as input, categorizes sound intensity and pitch and translates it into patterns of magnetization. However, when the tape is played, the output is once again sound.

5. Memory/Adaptation. Adaptation can be thought of as a changing of computation that depends on a specific input history. Examples from neuroscience are long term potentiation (LTP) and long term depression (LTD). Quantitatively, this is modeled as a history dependent function. On the other hand, computation which uses an explicit memory, poses a very complex problem which is beyond the scope of this paper.

Discrete, Deterministic Computation

This example can be easily extended to computation. Consider the speedometer common to every car. This machine takes as its “input code” the rotational speed of the front axle of a car. It uses this rotation speed to compute a linear speed of the car. This speed is denoted by the angle of a pointer on the speedometer dial. Each angle has a number associated with it. This number denotes the speed of the car.

This simple computation is highly similar to information transmission. We have an input code, an output code which relates directly to the input code and a simple one-to-one computation relating axle rotation speed to the car’s speed. An interesting difference is the mode of input and output in this mechanism. The mode changes: the input is axle rotation, the output is the angle of a pointer. This differs from information which often has the same modes of input and output.

Are there any computations where the mode of input and output are the same? A simple example is the “even or odd” computation used in binary computers. A binary computer uses zeroes and ones to denote both input and output. The “even or odd” computation in particular, takes an even number and returns a zero and

takes an odd number and returns a one: Some results of this computation are tabulated below:

The 'Even or Odd' Computation	
Input	Output
0	0
1	1
10	0
11	1
110	0
111	1

Notice that the input codes are any binary number. However, all outputs are either zero or one. Suppose that we only had access to the computer's outputs and thought that this computer was merely transmitting information. In that case, the computer/information transmitter would appear to have the following properties:

The 'Even or Odd' Computation Restricted to Output Codes	
Input	Output
0	0
1	1

As expected, the input code is exactly the same as the output code. This means that if we treat this simple computation as if it were an instance of information transmission, inputs and outputs are expected to match.

This statement is formalized as follows.

Definition: *Discrete, Deterministic Computation*

Consider a function C_0 . This function operates on a finite number of input elements and does the equivalent of using a look-up table to classify these input elements, I_j . Here, I denotes an input code and the subscript j denotes the j th input code where j is an integer. The output codes are a subset of the input codes and denoted as O_j . Furthermore, it is assumed that outputs are congruent with

inputs. Here, congruence is defined as a matching of type between input and output. In the “even or odd” computation described above, this means that an even input yields an even output and an odd input yields an odd output. With these assumptions, the following proposal can be made:

Proposition 1.

Let C_0 denote a function which satisfies the definition of discrete, deterministic computation. Then output codes O_j satisfy the following identity:

$$\mathbf{C}_0(O_j) = O_j \tag{1}$$

Proof: See the Appendix, Proposition 1. A formal exposition of discrete, deterministic computation is given in the appendix, definitions 1-3.

Eq. (1) as an optimal information channel.

The function \mathbf{C}_0 is a generalization of a noise-free, deterministic information channel. Consider an ‘alphabet’ of inputs restricted to any value of O_j . If O_j is the input code, then the output code is, of course, O_j . Thus, Eq. (1) treats these symbols in the manner of a noiseless information channel. However, it has the additional property of classifying any input of the form I_j . This is why we consider it to be an extension of the notion of an information channel.

Dynamical, Numerically Defined Computation

The nature of inputs and outputs in a neuron or neural circuit is often defined in terms of spiking voltages, spike rates etc. Thus, input and output codes are allowed to be (approximately) continuous over time instead of taking on a discrete form. The inputs and outputs have the nice property that at each point in time they can be assigned a numerical value, e.g., 70 mV, or 20 Hz. Furthermore, like discrete, deterministic computation, the basic form of the output is the same as the input, i.e., voltage over time, spiking frequency etc. This fact allows one to relate the input to the output in terms of the transfer function.

More importantly, this fact also allows one to generalize the notion of output codes to one that is a *translation* of input codes. A translation of input code to output code is often the case. For example, a car’s speedometer translates the angular velocity of an axle to the angle of a pointer on the speedometer screen. Thus, understanding a computation presumes an understanding of what the translation is. In the case of neural computation, this translation sometimes equates with the transfer function.

This occurs when an input is a *canonical input*. A canonical input is defined as an input whose attendant output has 100% of the information of the input. In other words, the output is equal to the input up to a linear transformation. Descriptively, this sort of input exemplifies the attribute which is being computed. For example, consider a light intensity meter which measures the mean light intensity of a region of light. Here, the input is the light impinging on each point of the meter. Numerically, this is the light amplitude at each point of the meter. The output is a number denoting average intensity. Imagine a situation where the amplitude, and hence,

the intensity of light is constant at each point on the meter. Then the output - a number denoting average intensity - is equal to the amplitude of the input up to a multiplicative constant. Thus, a constant intensity input is a “canonical input.”

Definition: *Dynamical, Numerically Defined Computation* \mathbf{C}_1

Time-varying, numerically defined computation consists of inputs and outputs which vary in time and value. Pragmatically speaking, this means that for a given time increment, Δt , the inputs and outputs have a fixed value, e.g., -40 mV, 100 Hz etc. Different time increments can have different values. Furthermore, the resolution of the input and output values, e.g., mV, is also assumed to be finite. (Note that these assumptions force the number of possible inputs and outputs to be finite.)

Proposition 2.

Let C_1 denote a function which satisfies the definition of dynamical, numerically defined computation. Furthermore, let T denote the transfer function which is defined by the equality: $x(t)T(x) = y(t)$ ¹. Here, x is any possible input to the computing mechanism and y is the output. Then output codes O_j satisfy the following identity:

$$\mathbf{C}_1 \left[\frac{O_j(t - \tau)}{T(O_j)} \right] = O_j(t) \quad (2)$$

Here, τ denotes a causal time delay and $T(O_j)$ must equal a constant over all values of $O_j(t)$. Note that by restricting $T(O_j)$ to a constant, no information is lost in this computation.

Proof: See the Appendix, Proposition 2. A formal exposition of time-varying numerically defined computation is given in the Appendix, Definition 4.

Eq. (2) as a noiseless, continuous information channel.

Like \mathbf{C}_0 , the function \mathbf{C}_1 is a generalization of a noise-free, deterministic information channel. However, in this case, its ‘alphabet’ consists of continuously valued functions, $O_j(t)$. These functions might be translated by a constant. However, their information content remains unchanged from input to output. Nonetheless, this possibility of translating the input, extends the behavior of Eq. (2) beyond just transmitting symbols. Furthermore, like Eq. (1), this function will also classify inputs of the form $I_j(t)$. Thus, Eq. (2) is an extension of a noiseless, continuous information channel.

Application of Eq.(2) to a Sum and Fire Neuron.

Consider a simple sum and fire neuron. This form of neuron sums voltages until a threshold is reached. Suppose that the threshold of this neuron is 90 mV. Each time this threshold is reached, the neuron fires a 60 mV action potential.

¹Note that the function $T(x)$ is a generalized transfer function defined by the identity $x(t)T(x) = y(t)$. It need not be a rate code transfer function.

Consider the situation where this sum and fire neuron has a 30 Hz spike train as its input. Furthermore, assume that the spikes in this input are 60 mV. The outgoing spike train would be 20 Hz spikes with each spike being 60 mV. In terms of Eq.(2) the transfer function, $T(O_j)$ becomes a *translation* function, i.e., the constant $2/3$. Thus, an application of Eq.(2) to the sum and fire neuron yields: $C_1 [(20 \text{ Hz}) / (2/3)] = 20 \text{ Hz}$. Therefore, a 20 Hz, 60 mV spike train is an output code. This makes sense: output spikes are at 60 mV and are at $2/3$ the rate of the input – indicating a 90 mV threshold. In principle, if the nature of the neuron had been unknown, we could have reached this conclusion by solving Eq.(2).

Application of Eq.(2) to the Thalamocortical Neuron

The thalamocortical neuron is a well-studied, well understood neuron; see De-schenes, Paradis, Roy , and Steriade (1984), Leresche, Lightowler, Soltesz, Jassik-Gerschenfeld , and Crunelli (1991), Nunez, Dossi, Contreras , and Steriade (1992) and Dossi, Nunez , and Steriade (1992). Within the context of biophysics, the followin possible outputs of this neuron can occur at the delta frequency level (1-4 Hz).

Biophysically Possible Outputs of the Thalamocoritcal Relay Neuron

Basic Frequencies	Changing Frequencies
1 Hz	$1 \leftrightarrow 2 \text{ Hz}$
2 Hz	$1 \leftrightarrow 3 \text{ Hz}$
3 Hz	$1 \leftrightarrow 4 \text{ Hz}$
4 Hz	$2 \leftrightarrow 3 \text{ Hz}$
	$2 \leftrightarrow 4 \text{ Hz}$
	$3 \leftrightarrow 4 \text{ Hz}$

An accurate biophysical model of the thalamocortical neuron has been described by Wang (1994). He carefully measured the transfer function of delta frequency spiking as a function of input. This transfer function was used to solve Eq.(2)². The possible output codes solutions (O_j) are 1, 2, 3 or 4 Hz as well as outputs with the changing frequencies: $1 \leftrightarrow 2 \text{ Hz}$, $2 \leftrightarrow 3 \text{ Hz}$ and $2 \leftrightarrow 4 \text{ Hz}$. Other outputs with changing frequencies, in particular $1 \leftrightarrow 3 \text{ Hz}$, $1 \leftrightarrow 4 \text{ Hz}$ and $3 \leftrightarrow 4 \text{ Hz}$ are biophysically possible but *computationally impossible*. Thus, if the thalamocortical

²For the sake of tractability, only delta frequency outputs were considered. Furthermore, only solutions where input and output spikes had equal voltages were calculated.

neuron is actually performing computations, these frequency transitions should not be found *in vivo* and probably not *in vitro*. This hypothesis was tested by the use of data published in Deschenes et al. (1984), Leresche et al. (1991), Nunez et al. (1992) and Dossi et al. (1992). The results are tabulated below.

Actual Outputs

Predicted output codes	Found in actual data	Non-output codes	Found in actual data
1 Hz	Yes	1 ↔ 3 Hz	No
2 Hz	Yes	1 ↔ 4 Hz	No
3 Hz	Yes	3 ↔ 4 Hz	No
4 Hz	Yes		
1 ↔ 2 Hz	Yes		
2 ↔ 3 Hz	Yes		
2 ↔ 4 Hz	Yes		

Note that the actual outputs and more importantly, the *outputs which were not found* correspond exactly to what was predicted.

Computation with Implicit Memory:

History Dependent, Time-Varying, Numerically Defined Computation

In neurobiology, the computation performed by a neuron/neural circuit is often a function of its input history. In other words, a neuron has a ‘memory.’ A simple example would be a sum and fire neuron which has a refractory period. Thus, if an incoming spike had caused this neuron to fire, a second incoming spike would not cause anything to happen if it occurred during the refractory period. Suppose again, that this neuron had a 90 mV threshold. Again, the output code from such a neuron would be spikes with a voltage of 60 mV. However, each pair of spikes in this output code would be separated by at least the refractory period. Thus, this unavoidable spike spacing in the output code would be indicative of some basic property of the computation mechanism. This notion of history dependent computation is formalized as follows.

Definition:

History Dependent, Time-Varying Numerically Defined Computation \mathbf{C}_2

A history dependent computation means that two versions of the same computation will have different results if they have different histories. That is, if the history over a time t_0 is $x_1(t_0)$ in one case and $x_2(t_0)$ in a second case, then the two different versions of \mathbf{C}_2 will compute differently *with the same inputs*.

Proposition 3.

Let C_2 denote a function which satisfies the definition of history dependent, time-

varying, numerically defined computation. Then output codes O_j satisfy the following identity:

$$\mathbf{C}_2 \left[\int_0^\infty \frac{O_j(t - \tau)}{T[O_j(\tau)]} d\tau \right] = O_j(t) \quad (3)$$

Proof: See the appendix, Proposition 3. A formal exposition of history dependent, time-varying numerically defined computation is given in the appendix, definition 5.

Discussion

The Implicit Definition of the Neural Code

Traditional coding theories define a neural code explicitly. For example, a “mean rate code” asserts that the important aspect of brain computation is the spiking rate. Similarly, a “timing code” asserts that actual spike times are the pertinent facts of neural computation. In both of these cases, the neural code is defined in an explicit manner. This is contrasted by the use of implicit definitions.

An implicit definition is a definition given by examples, i.e., exemplars of the neural code. This study uses implicit definitions. Specifically, examples of the neural code are given in the following manner. A group (or set) of inputs is defined as a group of exemplars when each member of the group yields the same output. A different output will result from a different set of inputs. Thus, each group of inputs have the same computational attribute. However, this attribute takes on a different value in a different group of inputs.

As an example, suppose that a neural circuit uses an average frequency code as its neural code. In this case, all spike trains with the same mean frequency will map to the same output. A second group of spike trains with a different mean frequency will yield a different output. Thus, each spike train in a given group of inputs has something in common with the other members of that set - they all have the same average spiking frequency. At the same time, they all have a different mean spiking frequency from members of other groups of inputs. Thus, grouping inputs in accordance to the rule that each member of the group has the same output yields an implicit definition of the neural code. Another example of this implicit definition of computational coding is given below.

An Implicit Definition of the Even or Odd Computation

By solving Eq. (1) for the “Even or Odd” computation we would discover that the output codes are 0 and 1. To fully characterize this computation, we need to classify inputs such that they yield either a 0 or 1. That is, find all inputs, I_k such that

$$\mathbf{C}_0(I_k) = O_j \quad (4)$$

The results are tabulated below:

The ‘Even or Odd’ Computation with Input Solutions

Inputs which yield the output code.

Possible inputs	Output
0, 10, 100, 110, ...	0
1, 11, 101, 111, ...	1

These results are a functional representation of the “Even or Odd” computation. Thus, in principle, we could implicitly define the computation of an “Even or Odd” computer by first solving Eq. (1) and then solving Eq. (4) for the output codes 0 and 1. If we are clever, we will notice that all even number inputs yield a 0 and all odd number inputs yield a 1. This would allow one to explicitly define this computation as an “even or odd” computation.

Appendix

Mathematical Definition of Computation

This initial definition is a noise-free idealization. It uses a single input and a single output. There are three parts to this definition:

Definition 1. Inputs and Outputs.

a. The set of all possible inputs \mathbf{X} and the set of all possible outputs \mathbf{Y} . These inputs and outputs are denoted as x and y respectively. For the purposes of this exposition, a basic one input, one output circuit model will be used first. This model has no memory. The union of the sets of all possible inputs and output will be denoted as \mathbf{S} . Thus we write $x, y \in \mathbf{S}$.

b. The subset of possible inputs and outputs over which the computation occurs. Denote inputs and outputs as I and O respectively. Denote the set of all possible I as \mathbf{I} and the set of all possible O as \mathbf{O} . Clearly $\mathbf{I}, \mathbf{O} \subset \mathbf{S}$.

Definition 2. Input and Output Code Partitions.

a. Denote each partition of \mathbf{I} as \mathbf{I}_j , then $\mathbf{I} = \bigcup_{j=1}^M \mathbf{I}_j$,

b. The output code partition. At its most fundamental level, computation is categorization. Thus, the set of all possible output codes \mathbf{O} is partitioned into categories, \mathbf{O}_k and $\mathbf{O} = \bigcup_{k=1}^N \mathbf{O}_k$. Each partition has only *one* member, O_k .

c. Partitions are ordered. Let \sim denote congruence, \succ denote precedence and \prec succession which, in fact, is an equivalence relation (i.e., reflexive, symmetric, and transitive). Then:

$$\text{If } j = k, \text{ then } \mathbf{I}_j \sim \mathbf{I}_k, \text{ else if } j \neq k, \mathbf{I}_j \succ \mathbf{I}_k \text{ or } \mathbf{I}_j \prec \mathbf{I}_k. \quad (5)$$

Similarly,

$$\text{If } j = k, \text{ then } \mathbf{O}_j \sim \mathbf{O}_k, \text{ else if } j \neq k, \mathbf{O}_j \succ \mathbf{O}_k \text{ or } \mathbf{O}_j \prec \mathbf{O}_k. \quad (6)$$

This ordering is transitive:

$$\text{If } \mathbf{I}_j \succ \mathbf{I}_k \text{ and } \mathbf{I}_k \succ \mathbf{I}_l \text{ then } \mathbf{I}_j \succ \mathbf{I}_l \quad (7)$$

Elements of the same partition are congruent, otherwise one has precedence over the other:

$$\text{If } I_1 \in \mathbf{I}_j \text{ and } I_2 \in \mathbf{I}_k \text{ and } j = k \text{ then } I_1 \sim I_2 \text{ else } j \neq k, \text{ and } I_j \succ I_k \text{ or } I_j \prec I_k. \quad (8)$$

A similar relation holds for output codes. All elements also satisfy transitivity:

$$\text{If } I_j \succ I_k \text{ and } I_k \succ I_l \text{ then } I_j \succ I_l \quad (9)$$

Definition 3. The discrete, deterministic and historyless computing function \mathbf{C}_0 .

This function is said to *compute* iff:

- a. The domain of \mathbf{C}_0 is the set of all possible inputs, \mathbf{X} and the range is the set of all possible outputs, \mathbf{Y} , where $\mathbf{X}, \mathbf{Y} \in \mathbf{S}$.
- b. $\mathbf{I} \subset \mathbf{X}$ and $\mathbf{O} \subset \mathbf{Y}$.
- c. \mathbf{I} and \mathbf{O} are partitioned such that $N \geq M$,
- d. If $x_1 \in \mathbf{I}_j$ and $x_2 \in \mathbf{I}_j$ then $\mathbf{C}_0(x_1) = \mathbf{C}_0(x_2)$.
- e. If $x \in \mathbf{I}$ then $\mathbf{C}_0(x) \in \mathbf{O}$.

This function is called *order-preserving* when:

$$I_1 \succ I_2 \text{ iff } \mathbf{C}_0(I_1) \succ \mathbf{C}_0(I_2). \quad (10)$$

Proposition 1.

Let \mathbf{C}_0 be a discrete, deterministic and historyless computing function which is order-preserving. Furthermore, let $\mathbf{O} \subset \mathbf{I}$. Then all output codes O_j satisfy the equation:

$$\mathbf{C}_0(O_j) = O_j \quad (11)$$

Proof by induction:

Proof for the first output, O_1 . Without loss of generality, define the preference relation of output codes as $O_M \succ O_{M-1} \dots O_3 \succ O_2 \succ O_1$. Since each input code partition contains only congruent inputs and no two different output codes are congruent, the assumption of $\mathbf{O} \subset \mathbf{I}$, forces the number of output codes in each input partition to be either zero or one. Denote the first input code partition that contains an output code as \mathbf{I}_1 .

Now, consider only the input code partitions, \mathbf{I}_k which contain an output code. These partitions are ordered as $\mathbf{I}_M \succ \mathbf{I}_{M-1} \dots \mathbf{I}_2 \succ \mathbf{I}_1$. Denote the output code in partition \mathbf{I}_1 as O_j . Let \mathbf{I}_m , $m > 1$ be all the other input code partitions which contain an output code O_k . Then for all $m > 1$, $\mathbf{C}(O_m) \succ \mathbf{C}(O_j)$. Thus, by order-preservation, for all $m > 1$, $O_m \succ O_j$. There is only one output code that satisfies this condition, it is O_1 . Thus $O_j = O_1$. Hence, $\mathbf{C}_0(O_1) = O_1$.

Generalization to O_n . Let $O_n \in \mathbf{I}_n$, and $O_n = \mathbf{C}(O_n)$. Then for all \mathbf{I}_m , $m > n$, $\mathbf{C}_0(O_m) \succ \mathbf{C}_0(O_n)$ and by order preservation, $O_m \succ O_n$. Assume $O_j \in \mathbf{I}_{n+1}$, then for all $k > j$, $O_k \succ O_j \succ O_n$. There is only one output code that satisfies this condition, it is O_{n+1} . Therefore, $O_j = O_{n+1}$. Thus, if $\mathbf{C}_0(O_n) = O_n$, then $\mathbf{C}_0(O_{n+1}) = O_{n+1}$ which completes the proof.

Dynamical, Numerically Defined Computation \mathbf{C}_1

Assume that all input and output codes are time dependent functions, i.e., $I_i(t)$ and $O_j(t)$ respectively. At each small interval in time Δt , assume that each $I_i(\Delta t)$ is equal to a constant k_i . Also, assume that values of k_i are defined on the interval

$[a, b]$ of the real number line. Similarly, at each time interval all $O_j(\Delta t) = L_i$ are defined on a real number line interval $[c, d]$.

Definition 4. The Dynamical, Numerical Computing Function.

a. For each small time interval, Δt , $I_i(n\Delta t)$ and $O_j(n\Delta t)$ satisfy the definition of input and output codes. Here, n is an integer.

b. $\mathbf{C}_1 [I_i(n\Delta t - \tau)] = O_j(n\Delta t)$ is a causal, order-preserving function which computes (Definition 1). Here, τ denotes the causal time delay

c. A time-dependent input code partition is defined as: $\mathbf{I}_j(t) = \bigcup_{n=1}^L \mathbf{I}_j(n\Delta t)$.

Here, n and L are integers and $L\Delta t$ is the time length over which the partition and input codes are defined.

d. A time-dependent output code partition is defined as: $\mathbf{O}_j(t) = \bigcup_{n=1}^L \mathbf{O}_j(n\Delta t)$.

e. For each small time interval Δt , $I_i(n\Delta t) = k_{i,n}$. Here, $k_{i,n}$ is a constant and for each n , all $k_{i,n}$ are in the real number interval $[a_n, b_n]$.

f. Similarly, for each time interval Δt , $O_j(n\Delta t) = l_{j,n}$. Here, $l_{j,n}$ is a constant and for each n , all $l_{j,n}$ are in the real number interval $[c_n, d_n]$.

Note that if $\mathbf{O}(t) \subset \mathbf{I}(t)$ and \mathbf{C}_1 is order-preserving, then by Proposition 1:

$$\mathbf{C}_1 [O_j(t - \tau)] = O_j(t) \quad (12)$$

Proposition 2.

Let \mathbf{C}_1 be a time-dependent, numerical computing function which is order-preserving. Then all output codes O_j satisfy the functional equation on $O_j(t)$:

$$\mathbf{C}_1 \left[\frac{O_j(t - \tau)}{T[O_j]} \right] = O_j(t) \quad (13)$$

Here, $T[O_j]$ is the input *transfer* function, defined as $T(x)x(t) = y(t)$. In this equation, $T(x)$ is constrained to the set of x such that $T(x) = k$, where k is a constant.

Proof.

Case 1. $O_j(t) \notin \mathbf{I}(t)$. Choose $x_i(t) \notin \mathbf{I}(t)$ such that $x_i(t) \sim I_i(t)$. Then $\mathbf{C}_1 [x_i(t - \tau)] = x_i(t)T(x_i) = O_j(t)$. No computation is performed on all $x_i(t)$ when $x_i(t) \notin \mathbf{I}(t)$. Hence, for all $x_i(t)$, $x_j(t) \notin \mathbf{I}(t)$, $T(x_i) = T(x_j)$. That is, the transfer function is independent of x and becomes a *translation* function. Thus, $T(x_i) = T[O_j]$ which yields: $x_i(t)T(O_j) = O_j(t)$. Hence, $x_i(t - \tau) = \frac{O_j(t - \tau)}{T(O_j)}$. Thus, $\mathbf{C}_1 [x_i(t - \tau)] = \mathbf{C}_1 \left[\frac{O_j(t - \tau)}{T(O_j)} \right] = O_j(t)$.

Proof of $\frac{O_j(t)}{T(O_j)} \sim I_i(t)$. $T(O_j)$ is restricted to a *translation* function. Denote its translation value as k (it is assumed that $\mathbf{C}_1 [0] = 0$). Examining the Taylor series

expansion of $\mathbf{C}_1 \left[\frac{O_j(t)}{k} \right]$:

$$\mathbf{C}_1 \left[\frac{O_j(t)}{k} \right] = \sum_{j=0}^{\infty} \left(\frac{d^j \mathbf{C}_1 [x]}{dx^j} \right)_{x=O_j(t)} \frac{\left[\frac{O_j(t)}{k} - O_j(t) \right]^j}{j!} \quad (14)$$

$$= kO_j(t) + k \left[\frac{O_j(t)}{k} - O_j(t) \right] \quad (15)$$

$$= O_j(t) \quad (16)$$

Hence, $\mathbf{C}_1 \left[\frac{O_j(t)}{k} \right] = \mathbf{C}_1 [I_i(t)]$, so by the order-preserving property, $\frac{O_j(t)}{T(O_j)} \sim I_i(t)$.
 Case 2. $O_j(t) \in \mathbf{I}(t)$. Proposition 1 yields: $\mathbf{C}_1 [O_j(t - \tau)] = O_j(t)$. Since $T [O_j] = 1$, this can be rewritten as $\mathbf{C}_1 \left[\frac{O_j(t-\tau)}{T(O_j)} \right] = O_j(t)$.

Definition 5. The Time and History Dependent Numerical Computing Function \mathbf{C}_2

a. The time and history dependent numerical computing function satisfies Definition II. However it is history dependent. Specifically, let $\mathbf{C}_{2a}(t)$ be the function with an input, $I_a(t)$ ending at $-\tau$ and $\mathbf{C}_{2b}(t)$ be the function with a different input, $I_b(t)$ of equal time length ending at $-\tau$, In general, $\mathbf{C}_{2a}(t) \neq \mathbf{C}_{2b}(t)$.

Proposition 3.

Let $\mathbf{C}_2 [I(t - \tau)]$ be a time and history dependent numerical computing function with the zero point $\mathbf{C}_2(0) = 0$. Then all output codes O_j satisfy the functional equation on $I_i(t)$:

$$\mathbf{C}_2 \left[\int_0^{\infty} \frac{O_j(t - \tau)}{T [O_j(\tau)]} d\tau \right] = O_j(t) \quad (17)$$

Proof: Let $p(t + k\Delta\tau)$ denote a rectangular pulse, where $\Delta\tau$ is a small increment of time. Then an output, $O(t)$ can be approximated as:

$$O(t + \tau) \approx \sum_{k=0}^{\infty} O(k\Delta\tau)p(t + k\Delta\tau)\Delta\tau \quad (18)$$

This approximation is applied to an inversion of \mathbf{C}_2 :

$$\mathbf{C}_2^{-1} [O_j(t + \tau)] = I_i(t) = \frac{O_j(t - \tau)}{T [O_j(t - \tau)]} \quad (19)$$

Here, the domain of \mathbf{C}_2 is restricted to a single $I_i(t)$ for each $\mathbf{I}(t)_j$, thus making the function invertible. Also note that the O_j notation in $T [O_j(t - \tau)]$ will be suppressed, so that it will be written as $T(t - \tau)$. Then, to compute a history dependent value for $\mathbf{C}_2^{-1} [O(t)]$ we treat each $O(k\Delta\tau)$ as a separate variable and

apply a multi-variable Taylor series expansion around zero:

$$\mathbf{C}_2^{-1}[O(t+\tau)] \approx \mathbf{C}_2^{-1}(0) + \sum_{j=1}^{\infty} \left(\frac{1}{j!} \left(\sum_{k=0}^{\infty} [O(k\Delta\tau) - 0] \frac{\partial}{\partial O(k\Delta\tau)} \right)^j \mathbf{C}_2^{-1}[O(t)] \right) \quad (20)$$

Using Eqs. (13, 18) and yields

$$= \mathbf{C}_2^{-1}(0) + \sum_{k=0}^{\infty} \frac{O_j(k\Delta\tau)}{T(t+k\Delta\tau)} \Delta\tau \quad (21)$$

and taking the limit as $\Delta\tau$ goes to $d\nu$ and N goes to infinity yields:

$$= \mathbf{C}_2^{-1}(0) + \int_0^{\infty} \frac{O_j(\nu)}{T(t+\nu)} d\nu \quad (22)$$

Assuming the $O(t)$ is zero for $t < 0$ and changing variables $t + \nu = -\tau$ yields:

$$= \mathbf{C}_2^{-1}(0) + \int_0^{-\infty} \frac{O_j(t+\tau)}{T(-\tau)} d\tau = \mathbf{C}_2^{-1}[O_j(t)] \quad (23)$$

In general, $\mathbf{C}_2^{-1}(0) = 0$. Thus, inverting \mathbf{C}_2^{-1} and noting that this inversion changes the sign of the causal time lag yields:

$$\mathbf{C}_2 \left[\int_0^{\infty} \frac{O_j(t-\tau)}{T(\tau)} d\tau \right] = O_j(t) \quad (24)$$

*References

- Churchland, P. & Sejnowski, T. (1992). *The computational brain*. Cambridge: MIT Press.
- Davis, M. (Ed.). (1967). *The undecidable : basic papers on undecidable propositions, unsolvable problems and computable functions*. Hewlett, NY: Raven Press.
- Deschenes, M., Paradis, J., Roy, J. P. , & Steriade, M. (1984). Electrophysiology of neurons of lateral thalamic nuclei in cat: resting properties and burst discharges *Journal of Neurophysiology*, *51*, 1196-1219.
- Detlefsen, M. (1986). *Hilbert's program : an essay on mathematical instrumentalism*. Boston: Dordrecht.
- Dossi, R., Nunez, A. , & Steriade, M. (1992). Electrophysiology of a slow (0.5-4 hz) intrinsic oscillation of cat thalamocortical neurones in vivo *Journal of Physiology*, *447*, 215-234.
- Eggermont, J. J. (1993). Wiener and volterra analyses applied to the auditory system *Hearing Research*, *66*, 177-201.
- Hennie, F. (1977). *Introduction to computability*. Reading, MA: Addison-Wesley.
- Leresche, N., Lightowler, S., Soltesz, I., Jassik-Gerschenfeld, D. , & Crunelli, V. (1991). Low-frequency oscillatory activities intrinsic to rat and cat thalamocortical cells. *Journal of Physiology*, *441*, 155-174.
- Nunez, A., Dossi, R. C., Contreras, D. , & Steriade, M. (1992). Intracellular evidence for incompatibility between spindle and delta oscillations in thalamocortical neurons of cat *Neuroscience*, *48*, 75-85.
- Rieke, F., Warland, D., Steveninck, R. de Ruyter van , & Bialek, W. (1997). *Spikes: Exploring the neural code*. Cambridge, MA: MIT Press.
- Wang, X.-J. (1994). Multiple dynamical modes of thalamic relay neurons: rhythmic bursting and intermittent phase-locking *Neuroscience*, *59*, 21-31.