Helsinki University of Technology Laboratory of Computational Engineering Publications
Teknillisen korkeakoulun Laskennallisen tekniikan laboratorion julkaisuja
Espoo 2000 REPORT B22

NEURAL SYSTEMS IN LIGHT OF DISTRIBUTION CODES AND INFORMATION THEORY

Seppo Pohja

TEKNILLINEN KORKEAKOULU
TEKNISKA HÖGSKOLAN
HELSINKI UNIVERSITY OF TECHNOLOGY
TECHNISCHE UNIVERSITÄT HELSINKI
UNIVERSITE DE TECHNOLOGIE D'HELSINKI

Helsinki University of Technology Laboratory of Computational Engineering Publications
Teknillisen korkeakoulun Laskennallisen tekniikan laboratorion julkaisuja
Espoo 2000 REPORT B 22

NEURAL SYSTEMS IN LIGHT OF DISTRIBUTION CODES AND INFORMATION THEORY

Seppo Pohja

Dissertation for the degree of Doctor of Technology to be presented with due permission for public examination and debate in Auditorium S4 at Helsinki University of Technology (Espoo, Finland) on the 3:rd of November, 2000 at 2 o'clock afternoon.

Helsinki University of Technology
Department of Electrical and Communications Engineering
Laboratory of Computational Engineering

Teknillinen korkeakoulu
Sähkö- ja tietoliikennetekniikan osasto
Laskennallisen tekniikan laboratorio

Distribution:
Helsinki University of Technology
Laboratory of Computational Engineering
P. O. Box 9400
FIN-02015 HUT
Fax.+358-9-451 4830

ISBN 951-22-5197-3
ISSN 1455-0474
Otamedia Oy
Espoo 2000

ABSTRACT

Distribution code is defined as a code that uses the joint probability distribution of a vector of binary-valued random variables to represent numeric values. When applied to neural systems, the values of the source variables (i.e., input variables) determine the relative frequencies of input vectors either deterministically or stochastically. Similarly, the relative frequencies of the output vectors determine the values of the target variables (i.e., output variables).

A mathematical model is proposed allowing unified treatment of biological neural systems and binary-valued artificial neural systems.

The distribution function of a neural system is defined as the mapping from the distribution of the input vectors to the distribution of output vectors. It is conceptually similar to the input-output function.

Information-theoretic properties of probabilistic and deterministic one-dimensional distribution codes are studied. One result is that the channel capacity of such codes has an upper limit with logarithmic asymptotic behaviour in terms of the size of the sample set and that there are codes whose channel capacity reaches the upper limit.

The distribution function of a deterministic memoryless neural system using a one-dimensional distribution code is shown to be monotonic. Several other kinds of neural systems are studied and shown not to have this limitation.

It is shown that, when the input weights are fixed, the distribution function of a binary neuron can be controlled to much finer degree – it is an adjustable linear combination of several parts – than the input-output function of a real-valued neuron, which can merely be translated.

Multidimensional distribution codes can be used to encode source variable values whose number greatly exceeds the number of the physical inputs of the neural system. There are multidimensional distribution codes that make it possible to change the output distribution of the system without changing any of the marginal distributions of the physical inputs. It can be argued that real-valued neurons do not have an analogue for this property.

The last part of this work focuses on experimental work and biological neural systems. A literature survey of information theoretic study on biological neural systems shows a need for new methods for generating data in a controlled and easy manner. A simulation-based methodology is proposed, implemented and evaluated. Two sets of simulations are carried out as a proof of concept. They focus on studying the information-theoretic implications of the topology of a pyramidal neuron.

AUTHOR'S CONTRIBUTION

This entire thesis is the contribution of the author with one exception.

Erik Fransén contributed to Chapter 12.3 by proposing that variations in the topology of simulated biological neurons could be easily modelled if the equivalent cylinder of the dendritic tree was maintained constant. This made it feasible to carry out the simulations. He is not a co-author and, therefore, any errors or omissions in the mathematical treatment or in the text are the fault of the author of this thesis.

Earlier forms of some of the ideas discussed in Part 2 have been published previously (Pohja, 1996). Other than that, this monograph comprises previously unpublished work.

ACKNOWLEDGMENTS

To my father's memory

Without the continuous support and understanding of my closest ones I could never have completed this work. I'm ever grateful to them.

This work has been done under the inspiring guidance of Pentti Kanerva and Kimmo Kaski. It has been a great privilege. Special thanks to them.

I would also like to thank the following people and organisations:

Part of the research was done in the NEAR project at Swedish Institute of Computer Science where I got valuable feedback from Gunnar Sjödin, Jan Kristoferson and Roland Karlsson among others of the great staff of the institute.

Anders Lansner of Kunliga Tekniska Högskolan gave advice and information on biological neurons.

Erik Fransén of Kunliga Tekniska Högskolan saved no trouble in teaching me the basics of simulating biological neurons.

Jorma Rissanen of IBM Almaden Research Center instructed me on various aspect of information theory while Seppo Kuusisto of Tampere University of Technology provided information on related tools. Their combined effort enabled me to rule out some potential lines of research.

Kari Kankaala, who worked at the time at Tampere University of Technology, spent considerable time and trouble in explaining the qualities and use of random number generators.

The NEAR project was partly funded by Real World Computing Partnership. Tampere University of Technology granted a stipend in support of writing this thesis. Nokia Telecommunications has provided technical resources and support services. Helsinki University of Technology has supported the publishing of this thesis.

TABLE OF CONTENTS

PART 2: NEURAL SYSTEMS

CHAPTER 1
WORK OF OTHERS VERSUS THIS THESIS

This chapter outlines the position of this thesis in respect to the work of others.

The roots of this thesis are planted in more than one of currently active areas neural research. The focus of this thesis could be said to be the intersection, rather than the union, of those research areas. This chapter positions this thesis against the back ground of current research.

The main tools used are statistics, information theory and simulation. The goal of this thesis is to use those tools to contribute to the various areas of research where this thesis is stemming from and to bring those areas a bit closer together by identifying and, hopefully, bridging some gaps between them.

1.1 TYPE OF NEURON

One research area where this thesis is rooted are the similarities and differences of real-valued neurons (i.e. sigmoidal or graded-response neurons), binary-valued neurons (i.e. McCulloch-Pitts neurons or threshold gates) and spiking neurons and the corresponding neural networks.

Gerstner et al. (1993) showed that Hebbian learning enables spiking neurons to store and retrieve spatio-temporal patterns with a time resolution of 1 ms (pp. 503, 513). They achieved this by using information on individual spikes and not just on the mean firing rates of neurons. They also argue that such improved time resolution is required in various biological contexts, e.g., a fly rushing around at high speed through an unknown environment (p. 513). Gerstner et al. draw a parallel between real-valued neurons and the use of mean firing rate (p. 503), which is a case of a priori averaging. While they do not directly criticise

the use of real-valued neurons, they do warn about the risk of missing the essentials if using a priori averaging (p. 514).

Similar biologically oriented motivation for the use of spiking neurons was given by Maass (1998) as he analysed the computational models from point of view of time and space. As he put it "… issue on which most neuroscientists seem to be able to agree, and which may point to a fruitful area for future contributions from theoretical computer science to this field: that time and space appear to play a different role for computations in biological neural systems than for computations in currently existing computational models in theoretical computer science" (p. 73). He also illustrates that a spiking neuron can carry out computational operations that are not at all reflected in the model of binary-valued or real-valued neuron (pp. 76-77).

In another paper Maass (1997a) gives the proof that the computational power of a network of spiking neurons is strictly larger than that of a network of real-valued neurons. In yet another article Maass (1997b) systematically compares the binary-valued, real-valued and spiking neurons in terms of their ability to approximate various types of functions. Again, he concludes that spiking neurons are computationally more powerful than other neural network models.

This thesis introduces a model that combines properties of binary-valued neurons and spiking neurons. Binary-valued neurons are dealt with as one of the simplest types of spiking neurons. This brings together what Maass (1997b) calls the first and the third generation of neural networks. Real-valued neurons are not the focus of this thesis, but the properties of binary-valued and spiking neurons are frequently compared to the properties of real-valued neurons.

1.2 NEURAL CODE

The question of how information is represented in networks of spiking neurons is another research area that has motivated this thesis. While this question is mostly presented by the biologically oriented part of the neural research community, the question is valid also for those interested in artificial neural networks – at least in the form of how the information *could* be represented in neural systems. For any given type of

neural systems – e.g., spiking or real-valued – there is a multitude of possible codes that could carry any given type of data.

Gerstner and van Hemmen (1992a) showed that "in a fully connected (recurrent) network that functions as an associative memory for stationary patterns, i.e., in generalized associative networks of the `Little-Hopfield´-type, the mean firing rate is the *only* relevant parameter of the network. Second order features of the spiking, such as the exact distribution or the variance of the spiking intervals, do not matter" (p. 195). They reached this conclusion using very general assumptions about the network. They did not, for instance, require any specific model of a neuron.

In contrast to the above-mentioned synthetic approach used by Gerstner and van Hemmen (1992a), several articles using an analytic approach to determine the nature of the neural code have been published. Often, information theoretic tools are used in the analysis as in (Bialek et al. 1993), (DeWeesa and Bialek 1995), (Johnson 1996), (Altman et al. 1997), (Deco and Schürmann 1998), (Strong et al. 1998) and (Johnson and Gruner 1998a), to name a few.

There is also a lot of literature focusing on specific aspects of the neural code. For instance, Bialek and Zee (1990) studied the encoding of continuously varying signals and showed that there is a trade-off between the context dependence of the code and its information capacity. Maass and Ruf (1995 and 1999) have shown how the spike shape affects the computational power of a neural system. Rudermann and Bialek (1992) studied stimuli whose frequency of change is above the Nyquist frequency of a single neuron. They showed that arrays of neurons can code those high frequencies; a result later supported by Spiridon and Gerstner (1999b). Arleo and Gerstner (1999) studied the coding of navigation maps. A large body of literature is dedicated to the study of the neural code of auditory, visual, tactile or some other given kind of information in the nervous system of a particular species. Chapter 11 gives some examples of this kind.

This thesis suggests a theoretical framework within which to represent, analyse and discuss various neural codes; namely, the distribution codes. That framework is then applied to various codes either to study the information-theoretical properties of the codes themselves, to study the neural systems carrying those codes or to demonstrate a proposed research methodology.

1.3 CAPABILITIES AND LIMITATIONS

Yet another research area influencing this thesis has been the inherent computational limits and capabilities of spiking neurons and networks of them. Most of the work done by the research community in this area is based on theoretic models rather than experiments.

Maass (1996) showed that networks of spiking neurons can simulate arbitrary given boolean circuits and finite automata. He also showed that this could be done with arbitrarily high reliability in the presence of noise and in "real-time". Maass and Natschläger (1997) showed that an arbitrary given Hopfield network can be emulated by a network of spiking neurons in temporal coding. The computational power of spiking was also discussed by Maass (1997c), who showed that networks of noisy spiking neurons are "universal approximators" and that arbitrary feedforward networks of real-valued neurons can be simulated by spiking neurons.

Quite many other capabilities and limitations of networks of spiking neurons have also been studied: The dynamics (Gerstner and van Hemmen 1992b; van Hemmen and Ritz 1995; Gerstner et al. 1996; Gerstner 2000; Eggert and van Hemmen 2000), the complexity of learning in temporal coding (Maass and Schmitt 1997), co-incidence detection abilities (Kempter et al. 1998), ability to tolerate and utilise noise and unreliable synaptic transmissions (Maass and Orponen 1998; Natschläger and Maass 1999; Spiridon and Gerstner 1999a), to name some.

This thesis focuses mostly on the limits in computational abilities of neural systems that use distribution codes. A specific object of study is how the topology of the network, the algorithms used and the specifics of the neural code affect those limits. Learning and training of networks are outside the scope of this thesis.

1.4 METHODOLOGY

The tools chosen for this thesis and used in the following chapters – statistics, information theory and simulation – allow us to combine these various research areas. The tools themselves or, more precisely, the study

of the methodologies needed to apply those tools in neural-net research has also been a research area inspiring this thesis. The development of the methodologies as well as some of the key results obtained by applying them are outlined in Chapter 11 in more detail.

Other relevant literature includes (Bialek et al., 1993; Strong et al. 1997; Strong et al. 1998; Johnson and Gruner 1998a; Johnson and Gruner 1998b; Johnson et al., 2000), most of which have already been mentioned in this chapter.

This thesis proposes a new methodology for the study of biological neurons. The methodology is based on the use of simulation, and on statistical and information theoretical analysis. As a case study, the methodology is applied to the study of the relationship between the information-transfer rate of the pyramidal neuron and the topology of its basal dendrite. In the case study, the methodology is applied in its simplest form but it can be combined with other methods proposed in the literature.

Part 1
Codes

CHAPTER 2
INTRODUCTION TO DISTRIBUTION CODES

It is frequently claimed that artificial neural systems are simplified models of their biological counterparts.

A similar claim could also be made concerning coding. The way data is coded when used as input and output to an artificial neural system is arguably mimicking the impulse discharges of biological neural systems.

One of the main dynamic attributes of a biological neuron is its firing rate or, in other words, how many impulse discharges per unit of time the neuron generates.

Many artificial neural networks accept real-valued vectors – that is, vectors whose elements are real numbers – as their input and produce output of the same kind. Each element of such a vector can be seen as an abstraction of the firing rate of a biological neuron (Beale and Jackson 1991, pp. 39-45; Wasserman 1989, pp. 12-13). Data is represented as real-valued vectors e.g., in Widrow-Hoff perceptrons (Widrow 1959; Widrow and Hoff 1960) and backpropagation networks (Rumelhart et al. 1986) and Kohonen's self-organising maps (Kohonen 1984).

Another main category of artificial neural systems besides the real-valued ones is binary neural systems. As the name implies, the input and output vectors have binary values. McCulloch and Pitts proposed that the activity of a biological neuron could be represented as a proposition of binary logic (1943, p. 117). In such an analogy, each binary value represents the presence or the absence of an impulse discharge. Binary representation of data is used e.g., in Hopfield networks (Hopfield 1982) and Kanerva's sparse distributed memory (Kanerva 1988).

There is a clear difference in how the coding of real-valued neural systems and that of binary neural systems relate to the coding of biological neural systems. In the case of real-valued systems the coding is a measure of the distribution of impulse discharges over time. In the case of binary systems the coding is an abstraction of individual impulse discharges.

These are, perhaps, the most commonly drawn analogies between the coding of biological and artificial neural systems. We will focus, however, on a third alternative, which combines the nature of both the previous analogies. We will call it *distribution code*.

As an example of a one-dimensional distribution code, let us look at a basic form of pulse-density modulation or PDM (see, e.g., Murray and Smith 1988; Tomberg and Kaski 1990). PDM represents the value of any single input or output of a neuron as either 0 or 1 at any given moment in time. In this respect PDM is not very different from the coding of any binary neural system. The difference is in encoding and decoding, which use not just the current value but previous values as well. Encoding controls the ratio of 0s and 1s and decoding observes that value.

To give an example, let us consider a very basic form of PDM which uses a time window of some predetermined length, say, 10 most recent values (see Figure 1). To encode a value between 0.0 and 1.0, say, 0.7 we produce a stream of 0s and 1s where there are exactly seven 1s in the time window at any time. To decode it, we just count the number of 1s (which is 7) and divide it by the width of the time window (which is 10).

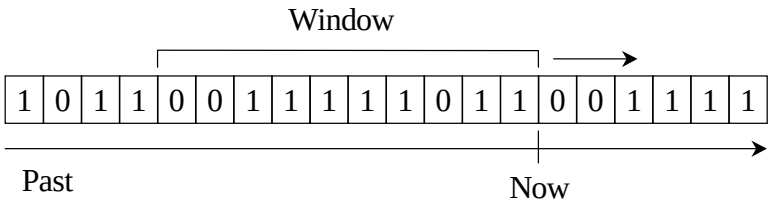

Figure 1. An example of defining the sample for a one-dimensional distribution code. The example is based on a pulse density modulation where a sliding fixed-width window is used.

In this one-dimensional example we can see the three main characteristics of a one-dimensional distribution code:

- Time is discrete.
- Momentary values are binary.
- There is mechanism for selecting a collection of momentary values. In this case, the mechanism is a sliding time window that is advanced one time step at a time.
- Encoding and decoding are based on the distribution of the momentary values. The value of a codeword is the probability of a randomly chosen bit in the codeword being 1.

In multidimensional cases we have several inputs (or outputs) and, thus, momentary values are binary vectors. We will write a sequence of such vectors as a matrix, whose columns identify with moments in time and rows with individual inputs (or outputs) of the neural system. See Figure 2.

In the one-dimensional case – often termed a spike frequency code – we are able to describe the distribution in just one number. In the multi-dimensional case we are dealing with a joint distribution of as many variables as there are rows in the matrix.

In the example, above, we have used the concept of a time window. A more general approach, which we will use, is a *sample*. Thus, we are interested in the distribution of the elements of a sample.

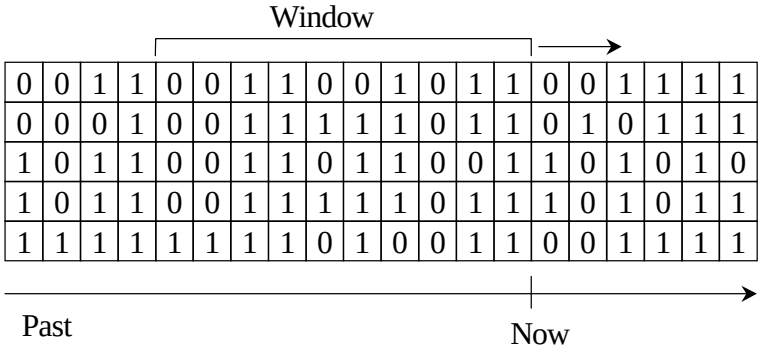

Figure 2. A multidimensional example of distribution code sampling.

Let us approach some issues related to the multidimensional cases by looking at the joint distribution in an example where a sample of a thousand binary vectors is used, each of which comprises three values. Each of the thousand vectors would correspond to one column such as those shown in Figure 2 except that there would only be three rows. Unlike in Figure 2 the columns may or may not comprise a single consecutive window. What can we say about such a distribution?

First, the vectors in the sample fall into eight different categories; one category for each value in $\{0,1\}^3$, e.g. $[1\,0\,1]$.

Second, associated with each category there is a *normalised sample frequency* or, in other words, the number of vectors in the category divided by the total number of vectors in the sample. These normalised sample frequencies define the *(normalised sample) distribution*.

Due to the relative-frequency interpretation of probability, we can often talk interchangeably about the normalised sample distribution and about the probability distribution. Therefore, we will often just talk about the distribution and, should the distinction matter, one can deduce the right kind of distribution from the context.

Consider a neuron with three inputs. As we saw before, there are eight categories of vectors and each category will have an associated normalised sample frequency. Each frequency must fall between 0.0 and 1.0 inclusive and their sum equals 1. Therefore, choosing the values of seven of those frequencies always defines the remaining one.

If the number of inputs (or outputs) of a neuron is n we have $d = 2^n - 1$ degrees of freedom remaining after requiring that the sum equals 1.

It is therefore possible to map a large number of source variables to a very small number of neural inputs (or target variables to outputs) using a distribution code. This can be achieved through distributed representation of the source variables across the inputs.

To give an idea what this could mean in practice, let us have 5 inputs and 31 source variables each falling between 0.0 and 1.0 inclusively. We associated each source variable with a unique five-bit codeword. Then we associated the one remaining five-bit codeword with the sum of all the source variables. To get the relative frequencies of each codeword, we divide the associated value by twice the sum of the source variables. This meets the requirement that the sum of the relative frequencies must equal 1.

There are, of course, other ways of distributing the representation of the source variables. This example was chosen mostly for its clarity.

We discussed the analogies often drawn between biological neural systems and either real-valued or binary-valued artificial neural systems on page 9. Returning to that discussion, one of the limiting factors in the analogy concerning real-valued systems is that they only have n degrees of freedom. They are limited to one source variable per input.

The difference between real-valued neural systems and binary-valued systems using distribution codes is striking in terms of degrees of freedom.

Comparing binary-valued systems using distribution codes and ones relying on more traditional codes – that is, codes where a single vector describing the momentary states of the inputs (or outputs) forms the codeword (see the discussion on page 9) – the difference is equally striking. Distribution codes can be used to represent $2^n - 1$ different real-valued source variables where as the more traditional codes can represent 2^n *discrete* values.

In practical terms, using all of the $2^n - 1$ degrees of freedom is often not sensible because the required size of the sample may become undesirably high. Thus, one might often consider adding a few more inputs or outputs to increase n and then using redundant coding.

A rather natural approach would be to use only n degrees of freedom and a redundant code that uses the marginal distributions of the inputs or outputs (instead of their joint distributions). This approach would seem to be analogous to using real-valued neurons and we will study it further in Chapter 7. Section 7.5 will, however, show that the analogy is only superficial and that this redundant code has very limited use.

Another kind of redundant codes will be discussed in Chapter 8. The codes have qualities closer to and, in some cases, exceeding those found in real-valued neurons.

In general, multidimensional distribution codes can be seen as extensions or generalisations of the concept of firing rate: Instead of the frequency of impulse discharges on a single input or output one counts the frequency of various bit patterns of several inputs or outputs.

However, taking an information-theoretic viewpoint, distribution codes are limited. Distribution codes deal with frequencies of bit patterns and ignore information carried in the ordering of those patterns. It is possible to extend the concept of distribution codes in such a way that

they cover all the available information – this idea will be discussed further in Section 10.5. – but our focus will be this limited case.

There is an obvious trade-off between the speed at which the neural system can operate and the size of the sample set used to compute the distribution: It takes longer to collect large sample sets than small ones. Gerstner et al. (1993) reached the same conclusion concerning the one-dimensional case. They also gave reasons to believe that this is a biological relevant factor.

One way to alleviate any problems this trade-off may cause is to add more inputs – that is, to increase n – and increase redundancy in the code. That way a smaller sample set can be used resulting in faster operation.

CHAPTER 3
BIOLOGICAL VERSUS ARTIFICIAL CODING

3.1 BIOLOGICAL CODING

Compared to biological neural systems, mathematical models of artificial neural systems are quite easy to deal with when applying distribution codes. If the mathematical model violates some of the assumptions underlying the distribution codes, we can simply say that the model is wrong and change it.

It is not so with biological neural systems. We cannot decide what the biological system is going to be like. Instead, we have to find out whether there is an acceptable mapping from the reality (that is, the biological neural system and its codes) onto our model of distribution codes and back. Without such a mapping we cannot either describe the reality in our model or interpret our model-based result and findings as properties of reality.

The situation is quite comparable to the difference between mathematics and physics. A theory may be flawless as far as mathematics is concerned but to turn it into a valid theory of physics one must show that it correctly reflects the reality.

There is no single physical phenomenon or property shared by all different types of biological neurons that could be considered the medium carrying the information nor is there any single kind of code. However, a considerable part of the research has been focused on spiking or, by another name, impulse discharges. We shall also focus on spiking, as it seems to be the principal means of passing information to and from several different types of neurons.

Mathematically, spikes are quite appealing: A spike either occurs or it does not. This can be represented using binary notation, say, 1 standing for the occurrence and 0 for the lack of it. (McCulloch and Pitts 1943, p. 117). Also, following each spike there is a period of time known as absolute refractory period, during which the neuron is unable to produce another spike. Thus, the code produced by a spiking neuron can be expressed as a list of those moments in time, when a spike occurs. Furthermore, if the observed period of time is limited, the length of the list is finite.

Maass and Ruf (1995) make an argument that the shape of the pulse is a relevant factor of the computational power of a neural system. They show that there are certain computational operations that cannot be carried out if spikes have a rectangular shape but can be carried out if the spikes have, say, a triangular shape.

As for our discussion, the important point is that the maximal computational power can be achieved without varying the shape of the spike. If the shape is fixed, all the available information is in the timing of the spikes and the shape can be ignored in information theoretical analysis, which is exactly what we will do in this model. It should be kept in mind that this model cannot be applied if the shape of the spike does carry information.

Figure 3 shows one possible function that can represent spiking. Before t_{i-1} the neuron is at rest and thus the value of the function is 0. At t_{i-1} a spike occurs and the function equals 1. Immediately thereafter, the neuron returns to its rest state, that is, the function returns to 0. At t_i another spike occurs. We will be assuming that any delay in returning to the rest-state after an impulse discharge has started does not carry any useful information. Therefore, a spike is described here as a single point in time.

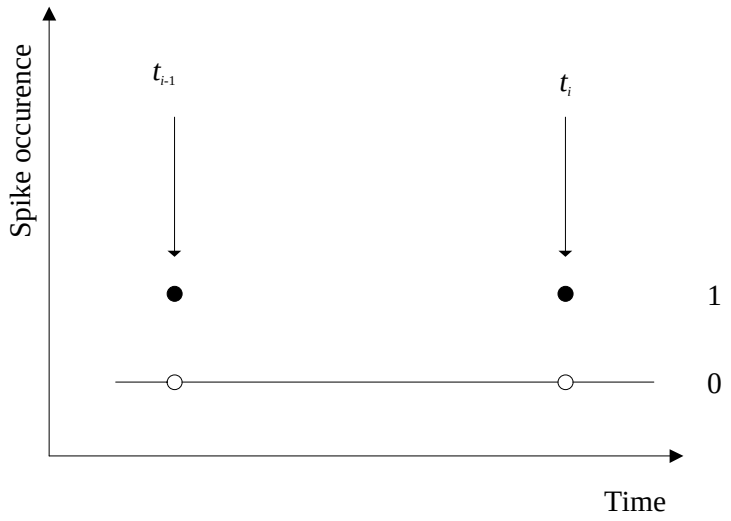

Figure 3. A model for describing spikes.

This kind of model does not correspond well to distribution codes. There is one element missing and it is the discrete time scale. Without it, each spike could carry infinite amount of information due to their point-like nature and, furthermore, a semi-continuous function could not be represented as a matrix.

We can solve these problems by replacing the continuous time scale with discrete time steps. If there is a spike during a time step, the value for that time step is 1 and otherwise it is 0. The time steps must not be longer than the absolute refractory period or there is a risk of loosing information due to two spikes occurring in one time step. Shorter time steps give a better approximation of the continuous time scale but for practical purposes, tend to increase the computational burden of most analyses.

An alternative approach, which was used, e.g., by Strong et al. (1998) would be to use time steps longer that the absolute refractory period and count the number of spikes occurring in each time step. Thus, we would get small non-negative values instead of 1 and 0. This alternative has the

draw-back of potentially losing some of the available information and is therefore discarded.

The result of using a discrete time scale is shown in Figure 4.

So far we have discussed how a physical reality could be described using a distribution code. The opposite is now rather obvious. If there is a 1 in a matrix associated with a distribution code, there is a spike occurring during that time step at the corresponding input (or output). Similarly, if there are ten 1s in a sample covering a single input, and if the sample has 1000 elements and a time step of 10 ms is used, there is an average of one impulse per second.

We can now use distribution codes for studying the nature of both artificial and biological neural systems and their codes.

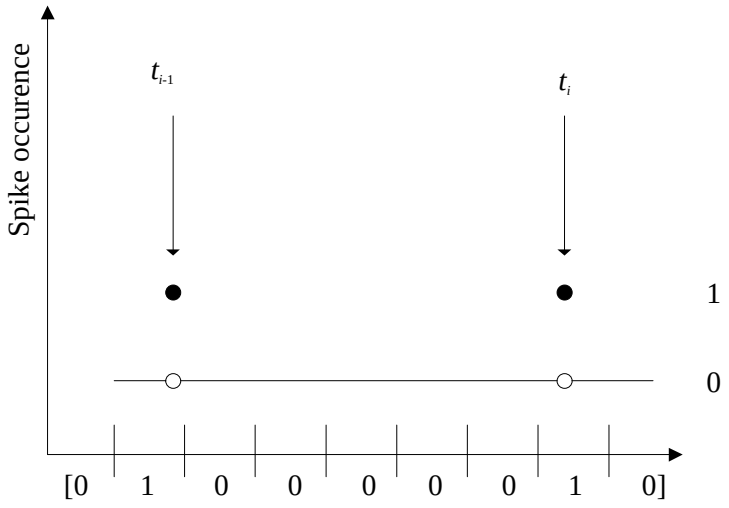

Figure 4. Generating a discrete time, binary vector representation of spiking.

CHAPTER 4
ONE-DIMENSIONAL DISTRIBUTION CODES

We will now study more closely the properties of one-dimensional distribution codes; particularly, their information-theoretic properties.

There is no single one-dimensional distribution code but a family of them. Some PDM codes – which are a family of codes themselves – are members of that family. PDM codes have been used as an implementation technique in both analog and digital VLSI artificial neural networks (see, e.g., Tomberg and Kaski 1990, p. 1277). Later, we will discuss two specific distribution codes. One of them is probabilistic and the other is deterministic.

For now, we shall focus on the properties that all one-dimensional distribution codes share.

4.1 BASIC DEFINITIONS

Let us now develop the basic notation and definitions to be used. This is mostly a task of formalising what we discussed earlier. We will give a special emphasis on the relationship between one-dimensional distribution codes and firing rates.

One of the basic measures of the state of a biological neuron is the firing rate, which is often expressed in Hertz or in spikes (that is, impulse discharges) per second.

Let us assume that we have taken a sample covering a time period t of the spike train produced by a biological neuron. Discrete time scale is being used. We have stored the sample as a binary vector, $\mathbf{x}$, which has l elements:

1) $$\mathbf{x} = [x_1\ x_2\ \ldots\ x_l]$$

Here, $\mathbf{x}$ is the *codeword* to be studied. To learn the firing rate $\hat{f}(\mathbf{x})$ of the neuron, all we need is to count the number of 1s and divide the result — which is denoted by $\text{ones}(\mathbf{x})$ below — by the length of the time period t.

2) $$\text{ones}(\mathbf{x}) = \left| \left\{ x_i = 1 \middle| \, i \in \{1,2,\ldots,l\} \right\} \right| = \sum_{i=1}^{l} x_i$$

3) $$\hat{f}(\mathbf{x}) = \frac{\text{ones}(\mathbf{x})}{t}$$

An alternative definition – one based on the number of sample elements – would be that firing rate is the mean of the codeword values: it is the count of 1s divided by the combined count of 1s and 0s in the vector, that is, the number of elements or l. To distinguish between these two definitions we will denote the latter by $f(\mathbf{x})$, which is

4) $$f(\mathbf{x}) = \frac{\text{ones}(\mathbf{x})}{l}$$

In most cases it is reasonable to assume that we know the length $\frac{t}{l}$ of a time step in discrete time scale. If so, the two definitions are freely interchangeable since

5) $$f(\mathbf{x}) = \frac{\text{ones}(\mathbf{x})t}{tl} = \hat{f}(\mathbf{x})\frac{t}{l}$$

We will be using the latter definition, as it also completely describes the distribution of any one-dimensional distribution code. To give this value a name – something more related to distribution codes and less to biological neurons – let us call $f(\mathbf{x})$ the *relative frequency* from now on.

Due to the definition, there typically are several binary vectors that have the same relative frequency, since only the number, but not the indexes, of 1s in the vector matters.

This is the reason why we can say that there is no single one-dimensional distribution code but a family of codes, whose members each have their own characteristics depending on the details of mechanism used to choose the binary vectors.

Encoding a source value is a two-step process. First the source variable is mapped onto a relative frequency and then the relative frequency is mapped onto a binary vector. For the sake of simplicity, we will always assume that a continuous source variable falls between 0.0 and 1.0, inclusive, and that the mapping onto the relative frequency is done by making the two equal.

Perhaps the most fundamental question about any code is how much information it can carry. Even though we are dealing with a family of codes, we can still find the upper limit for the amount of information. Then we can study the limit as a function of the length of the codeword.

We will approach this question through by studying how much mutual information the code and the value it represents can have at most. This is identical to observing the channel capacity of a memoryless channel, where a transition matrix $p(f(\mathbf{x})|s)$ defines the conditional probability of the relative frequency $f(\mathbf{x})$ of the vector (that is, sample) $\mathbf{x}$ given the source variable s. In terms of a memoryless channel, s is the value of a random variable $\mathbf{S}$ that is the input of the channel and $f(\mathbf{c})$ is the value of another random variable $\mathbf{T}$ that is the output of the channel.

We shall first assume that the number l of the elements in the output vector is fixed, in which case the channel capacity C is

6) $$C = \max_{p(w)}(I(\mathbf{T};\mathbf{S}))$$

where the maximum is taken over all possible distributions $p(w)$ and $I(\mathbf{T};\mathbf{S})$ is the mutual information of $\mathbf{T}$ and $\mathbf{S}$. This can also be written as a function of entropy $H(\mathbf{T})$ and conditional entropy $H(\mathbf{T}|\mathbf{S})$ as

7) $$C = \max_{p(w)} \left(H(\mathbf{T}) - H(\mathbf{T} \mid \mathbf{S}) \right)$$

We modify this equation by replacing the maximum of a difference by the difference of the maximum of the first term and the minimum of the second term to obtain an upper limit, that is,

8) $$C \leq \max_{p(w_1)} \left(H(\mathbf{T}) \right) - \min_{p(w_2)} \left(H(\mathbf{T} \mid \mathbf{S}) \right)$$

The first term is the maximum entropy of the relative frequency. Since the relative frequency is a discrete variable that can take $l+1$ different values, the maximum of the first term is $\log(l+1)$. The second term is a conditional entropy involving two discrete random variables. Thus, its minimum is zero. Thus the upper limit for the channel capacity is

9) $$C \leq \log(l+1) - 0 = \log(l+1)$$

Figure 5 visualises the upper limit as a function of the code length, that is, the number l of elements of the output vector.

The upper limit grows in a logarithmic fashion as the codeword length increases. Thus, the asymptotic growth of the channel capacity of any given one-dimensional distribution code must also be logarithmic if not slower.

This is not very good compared to the fact that the channel capacity could grow linearly as a function of the code length. Thus, it seems reasonable to say that one-dimensional distribution codes are best suited for situations where either only a small amount of information is necessary or where very long codewords are not a burden.

If we apply this to biological neurons, we must take into account that the length $\frac{t}{l}$ of the time step is an artefact introduced for the purpose of modelling and, therefore, so is also the number of time steps l. A more inherent property is the absolute refractory period and its relation to the length of the sample in units of time.

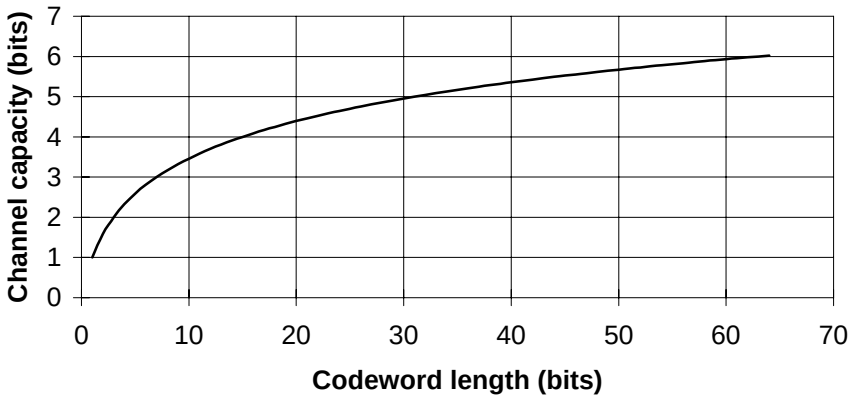

Figure 5. The upper limit for the channel capacity of any one-dimensional distribution code as a function of the codeword length, i.e., sample size.

Given the sample length t and the absolute refractory period t_r, the upper limit for the channel capacity is given by

10) $$C \leq \log\left(\frac{t}{t_r}+1\right)$$

assuming that the absolute refractory period equals the time step.

Figure 6 illustrates the upper limit for the channel capacity as a function of the absolute refractory period for some sample lengths. There is a near-linear relationship between the upper limit for the channel capacity and the logarithm of the absolute refractory period. The asymptotic coefficient is 1 (assuming a common logarithmic base). This means that, say, doubling the maximum firing rate of a biological neuron would in this model also double the upper limit for the channel capacity.

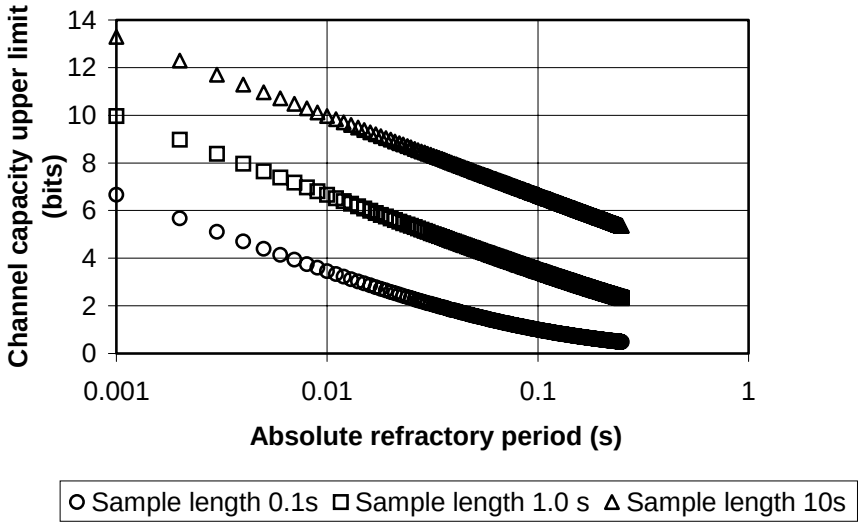

Figure 6. The upper limit of the channel capacity for distribution codes in terms of the absolute refractory period for three different sample lengths.

CHAPTER 5
DETERMINISTIC ONE-DIMENSIONAL DISTRIBUTION CODE

Let us now look at a specific distribution code. This code is one-dimensional and deterministic: In encoding, the same value of the source variable is always encoded using the same binary vector and, in decoding, the same binary vector always implies the same value of the target variable.

This particular code was chosen to be our case study on deterministic one-dimensional distribution codes because it minimises the error between the source variable and target variable – that is, the encoding-decoding error is minimised. This is true even when the codeword is built incrementally — by appending more and more bits to the codeword — without prior knowledge when to stop. On the other hand, the code is strongly oriented towards use in an artificial neural system rather than towards modelling biological neural systems.

5.1 CODING

All one-dimensional distribution codes are decoded the same way by computing the relative frequency, which is also the value of the final target variable. (Even if some other mapping between the relative frequency and the target variable is used, it is a choice independent of any other choices including which class of one-dimensional distribution codes is used. Hence, we can ignore this issue here). Therefore, one-dimensional distribution codes can be best distinguished from one another by describing how the encoding is done.

As for this particular code, let us assume that we have some number s that falls between zero and one. This is the number that we would like to encode.

11) $$s \in [0,1]$$

We shall represent the codeword $\mathbf{x}$ as a binary vector

12) $$\mathbf{x} = [x_1 x_2 \ldots x_l] \in \{0,1\}^l$$

whose length is l.

Encoding can be described as an incremental, step-by-step process where we first decide on the value of x_1 followed by the value of x_2 and so on. At each step, when deciding the value of element x_n, we compute what the relative frequency over $x_1, x_2, \ldots, x_n$ would be if x_n is zero and if its is one. For both cases, the absolute value of the difference between the relative frequency and the value s of the source variable is computed. Then, if for instance $x_n = 1$ would yield a smaller result than $x_n = 0$, that is if

13) $$\left| s - \frac{1 + \sum_{k=1}^{n-1} x_k}{n} \right| < \left| s - \frac{0 + \sum_{k=1}^{n-1} x_k}{n} \right|$$

we decide that x_n equals one[1]. If $x_n = 0$ would yield a smaller result, we decide that x_n equals zero. In the case where both possible values of x_n would yield the same absolute value, we always give x_n the same value. Whether that value is zero or one, does not matter.

This encoding yields an interesting result: Regardless of the number of steps taken, n, the error or, in other words, the absolute value of the difference between s and the relative frequency over $x_1, x_2, \ldots, x_n$ is always the smallest possible for that number of elements of $\mathbf{x}$. This may become handy in some applications where codewords are sent in a serial fashion, bit by bit. If necessary, one can start encoding and sending bits

[1] The $\sum_{k=1}^{n-1} y_k$ terms are taken to equal zero if $n - 1 = 0$.

of a codeword without prior knowledge on l. All that is needed is a signal when to stop working on that particular codeword. This gives a very flexible way of creating a stream of codewords of variable number of bits. On the other hand, if such an approach is taken, one must have some means of communicating the beginning or end of each codeword in the stream of bits.

As Figure 7 demonstrates, some values of the relative frequency correspond to more than one codeword of given length. For instance, when the length is three, the encoding algorithm may yield (among other values) $[010]$ and $[001]$ which both yield the same value for the relative frequency, $1/3$.

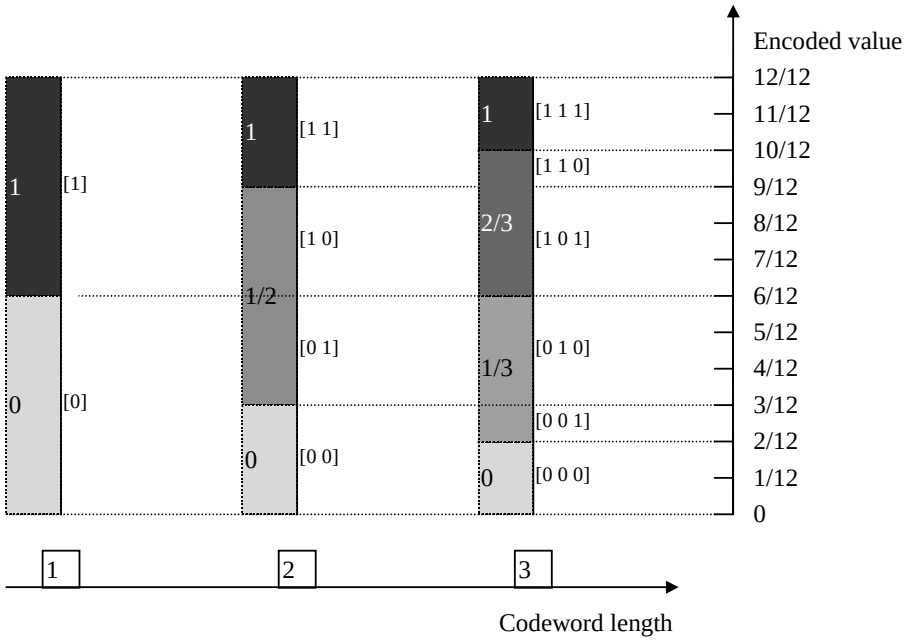

Figure 7. This figure shows all the possible codewords of length one, two or three bits. On the right hand side is an axis divided in twelfths. This axis represents the value of the source variable, s. The three shaded columns represent codeword lengths one, two and three. The relative frequency can be found by taking a horizontal line from the axis to one of the columns. Next to each column are the codewords. As you can see, there may be several codewords for a given codeword length and relative frequency. The dotted lines show the intervals covered by each codeword.

This suggests that we are losing information in decoding. We shall return to this question on page 33 but first, let us find out how much information the relative frequency carries about the encoded value s.

5.2 CHANNEL CAPACITY

While it is always enlightening to learn about the channel capacity, we now have an extra interest in doing that: In section 4.2 we discussed the upper limit of channel capacity of the family of one-dimensional distribution codes. At that time we did not show whether any member of the family could actually reach this upper limit.

If we can show that the channel capacity of this deterministic code reaches the upper limit we have also shown that the upper limit is also the supremum.

We shall approach the channel capacity problem in a bottom-up fashion by defining some useful concepts before deriving the actual formula for the channel capacity.

First, there are several potential values z_i for the relative frequency s. Assuming that the codeword has l bits, the number of 1s among those bits can be any integer between 0 and l inclusive. Therefore, we can write

14) $$z_i = \frac{i}{l}, \quad i \in \{0,1,\ldots,l\}$$

Second, there are $l+1$ intervals such that s belonging to a given interval always yields the same relative frequency z_i and, conversely, s belongs to the same interval when it yields a given relative frequency z_i. In Figure 7, those intervals are shown as areas of different shading in the columns. The first interval is

15) $$R_0 = \left[0, \frac{1}{2l}\right]$$

while the intermediate intervals can be written as

16) $$R_i = \left(\frac{2i-1}{2l}, \frac{2i+1}{2l} \right], \quad i \in \{1, 2, \ldots, l-1\}$$

and the last interval is

17) $$R_l = \left(\frac{2l-1}{2l}, l \right]$$

The intervals are here so ordered that, if s is a member of R_i, it yields z_i as the relative frequency.

To compute the channel capacity, we need an expression for the mutual information between the random variable $\mathbf{S}$ (that is, the source variable, whose value is s) and the random variable $\mathbf{T}$ (that is, the target variable, whose value is the relative frequency). By definition, the mutual information is

18) $$I(\mathbf{S};\mathbf{T}) = \sum_{i=0}^{l} \int_{0}^{1} p_{\mathbf{S},\mathbf{T}}(s, z_i) \log \frac{p_{\mathbf{S},\mathbf{T}}(s, z_i)}{p_{\mathbf{S}}(s) p_{\mathbf{T}}(z_i)} ds$$

Since s in R_i always yields z_i as the relative frequency, $p_{\mathbf{S},\mathbf{T}}(s, z_i)$ is zero unless s is in R_i while $\mathbf{T} = z_i$. This lets us write

19) $$I(\mathbf{S};\mathbf{T}) = \sum_{i=0}^{l} \int_{R_i} p_{\mathbf{S},\mathbf{T}}(s, z_i) \log \frac{p_{\mathbf{S},\mathbf{T}}(s, z_i)}{p_{\mathbf{S}}(s) p_{\mathbf{T}}(z_i)} ds$$

This has the effect of limiting the integration to areas where $p_{\mathbf{S},\mathbf{T}}(s, z_i) = p_{\mathbf{S}}(s)$. Therefore, we can derive an even simpler expression for the mutual information, first by replacement

20) $$I(\mathbf{S};\mathbf{T}) = \sum_{i=0}^{l} \int_{R_i} p_{\mathbf{S}}(s) \log \frac{p_{\mathbf{S}}(s)}{p_{\mathbf{S}}(s) p_{\mathbf{T}}(z_i)} ds$$

and then by simplification

21) $$I(\mathbf{S};\mathbf{T}) = -\sum_{i=0}^{l} \int_{R_i} p_{\mathbf{S}}(s) \log p_{\mathbf{T}}(z_i)\, ds$$

If we restructure the term and write

22) $$I(\mathbf{S};\mathbf{T}) = -\sum_{i=0}^{l} \log p_{\mathbf{T}}(z_i) \int_{R_i} p_{\mathbf{S}}(s)\, ds$$

we can take advantage of the fact that the probability of s being in R_i is the same as the probability of $\mathbf{T} = z_i$ by writing

23) $$I(\mathbf{S};\mathbf{T}) = -\sum_{i=0}^{l} p_{\mathbf{T}}(z_i) \log p_{\mathbf{T}}(z_i)$$

which happens to be the entropy of $\mathbf{T}$ or, in other words,

24) $$I(\mathbf{S};\mathbf{T}) = H(\mathbf{T})$$

We could have reached the same result simply by concluding that the relative frequency is a deterministic function of s. However, this longer approached helps us by underlining the role of the intervals R_i, which we will discuss in more detail shortly.

First, however, let us write the formula for that maximal value or, in other words, the formula for the channel capacity C. By definition, it is

25) $$C = \max_{p_{\mathbf{S}}(s)} I(\mathbf{S};\mathbf{T})$$

but, based on what we just learned, this can be given in terms of the entropy of $\mathbf{T}$. The code yields $l+1$ different values of $\mathbf{T}$, which is a discrete random variable. Therefore,

26) $$C = \max_{p_{\mathbf{T}}(z_i)} H(\mathbf{T}) = \log(l+1)$$

5.3 CHANNEL CAPACITY AND UNIFORM DISTRIBUTION

This maximum is reached whenever $\mathbf{T}$ has a uniform distribution. On the other hand, $\mathbf{T}$ is distributed identically to the way that $\mathbf{S}$ is distributed onto the intervals R_i. Therefore, any such distribution of $\mathbf{S}$ that gives s an equal probability of being in any of the intervals R_i would attain the channel capacity.

What would such a distribution be like? Looking at the widths of the intervals gives us some idea, since they are a limiting factor. The intervals that are outermost have width

27) $$w_0 = w_l = \frac{1}{2l}$$

which is one half of the width of all the intermediate intervals

28) $$w_i = \frac{1}{l}$$

Therefore, the average density within each intermediate interval must have the same value whereas the average density within the outermost intervals must be twice as high.

Taking an engineering point of view, such a probability density function may be hard to implement, in particular if the length l of the codeword varies, because the width of the intervals would change whenever l is changed. If it were not for those two outermost intervals, we could use a uniform distribution, which would be much easier from this point of view.

Let us find out how much of the channel capacity would be wasted if we did use uniform distribution. In that case, the mutual information would be

$$I(\mathbf{S};\mathbf{T}) = H(\mathbf{T})$$

29)
$$= -2\frac{1}{2l}\log\left(\frac{1}{2l}\right) - (l-1)\frac{1}{l}\log\left(\frac{1}{l}\right)$$

where the first term is the contribution of the outermost intervals and the second term is that of the intermediate intervals. This can be further reduced to

30)
$$I(\mathbf{S};\mathbf{T}) = \frac{1}{l}\log(2l) - \frac{l-1}{l}\log(l)$$

By using this information, we can compute how much of the channel capacity gets wasted if we use uniform distribution instead of an optimal one. Figure 8 shows the result for some short codewords.

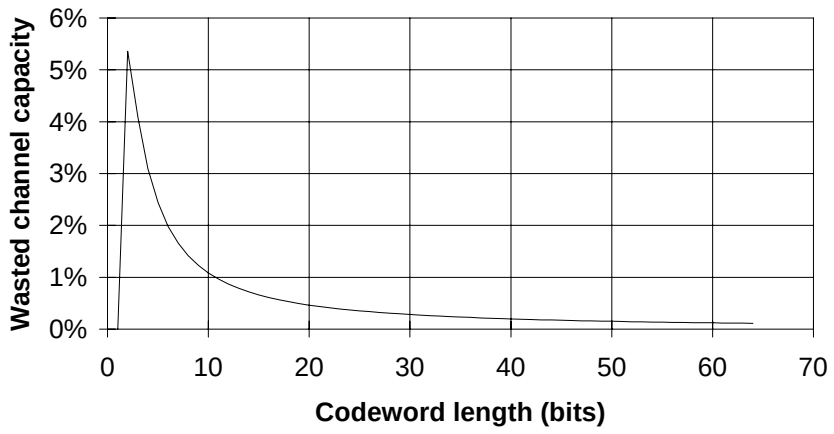

Figure 8. The wasted percentage of the channel capacity if uniform distribution is used. This is given as a function of the codeword length.

Looking at Figure 8, it would seem reasonable to say that for most implementations uniform distribution behaves quite well even at small codeword lengths.

At longer codeword lengths we should look at the asymptotic behaviour. Instead of looking at the proportion of the channel capacity that gets wasted as we did in Figure 8, let us study the absolute number of bits. For most purposes this probably makes more sense since the channel capacity itself grows towards infinity as the codewords get longer.

All we have to do is to look at the limes of the difference between the channel capacity and mutual information:

$$\begin{aligned} C - I(\mathbf{S};\mathbf{T}) &= \log(l+1) - \frac{1}{l}\log(2l) - \frac{(l-1)}{l}\log(l) \\ &\xrightarrow[l\to\infty]{} \log(l+1) - 0 - 1\times\log(l) \\ &\xrightarrow[l\to\infty]{} 0 \end{aligned} \tag{31}$$

Since the limes is 0, it seems obvious that the uniform distribution is close enough to the optimal distribution for any practical purposes, if the codewords are long enough.

5.4 CODEWORDS

Now that we have settled the issues related to the channel capacity as for the relative frequency of the codewords, let us focus on the codewords themselves.

Inherent to the concept of a one-dimensional distribution code is the idea that any information that is not available in the relative frequency is of no interest and *must not* be used in any form. If we do not follow this principle, we are no longer dealing with a one-dimensional distribution code but with some other code.

Let us now find out how much information we throw away by following this principle. To do that, we must entertain the idea of abandoning the principle. That way we can compare the channel capacity that we just computed to the channel capacity that is available by abandoning the principle. Since the former is computed based on the

relative frequency and the latter requires studying the codewords themselves, we shall call them the relative-frequency channel capacity and the codeword channel capacity, respectively.

As we can see in Figure 7, each possible value of the relative frequency defines an interval assuming that the length of codeword is given. In the figure, these intervals are shown using different shadings in the columns. The intervals are further divided where more than one codeword yields the same relative frequency.

Let us now find out the set P_l of points that separate intervals for two codewords when the codeword length is l. First, when the codeword length is one there is only one such point:

32)
$$P_1 = \left\{ \frac{1}{2} \right\}$$

By studying the algorithm used to produce the codewords, we soon realise that all points belonging to P_l separate two relative-frequency intervals for codewords of length l or more. (See Figure 7.)

We have already identified those points in Equation 16. Thus, we can express P_l recursively as

33)
$$P_l = \left\{ \frac{0+1}{2l}, \frac{1+2}{2l}, \ldots, \frac{2l-1}{2l} \right\} \cup P_{l-1}, l \in \{2,3,4,\ldots\}$$

Since P_l comprises the points that separate the codeword intervals, the number of such intervals is one greater than the number $\left|P_l\right|$ of points in P_l. Unfortunately, there is no known closed form equation for computing that number but we can compute it recursively by using Equation 33. The results are illustrated in Figure 9.

Analogously to what we did when computing the relative-frequency channel capacity C, we can reason that the codeword channel capacity $\hat{C}$ is equal to the maximum entropy of a discrete random variable. This time, the number of codeword intervals $\left|P_l\right|+1$ is the number of different values the random variable can take. Thus,

34) $$\hat{C} = \log\left(\left|P_l\right| + 1\right)$$

Figure 10 illustrates both channel capacities while Figure 11 shows how large a portion of the information that is available in a codeword 'is lost in translation' to the relative frequency. It would seem that the portion approaches one half asymptotically from below. There is no known way of confirming that.

The two channel capacities can not be achieved by the same source variable distribution, which is obvious based on Figure 7.

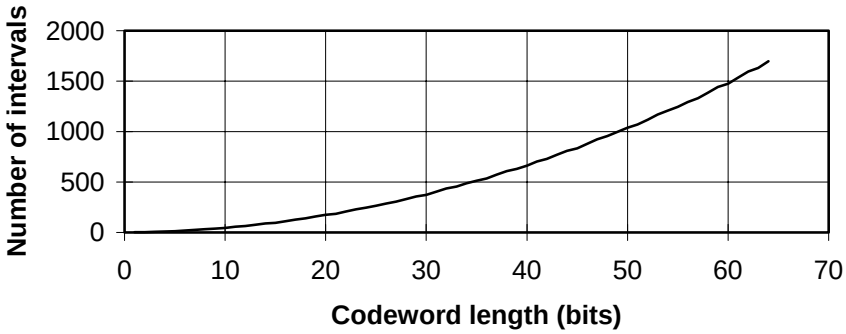

Figure 9. The number of intervals yielding a unique codeword as a function of the codeword length.

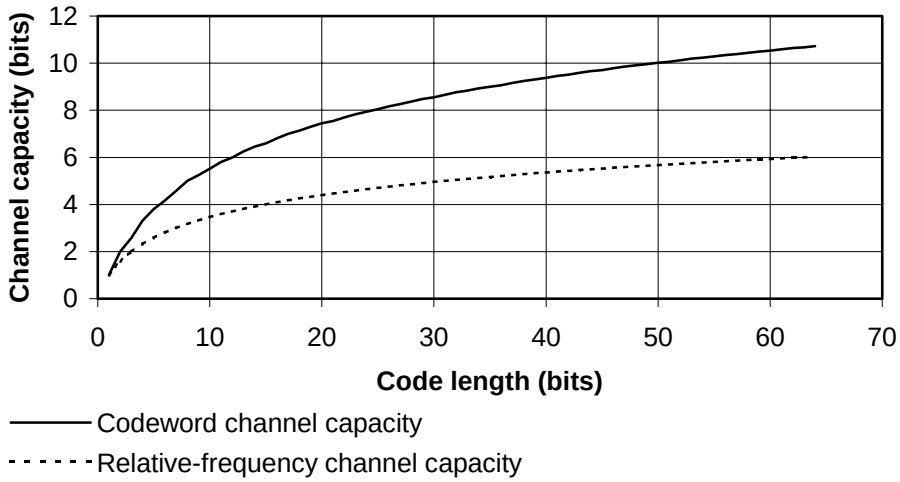

Figure 10. The codeword channel capacity and the relative-frequency channel capacity as functions of the codeword length.

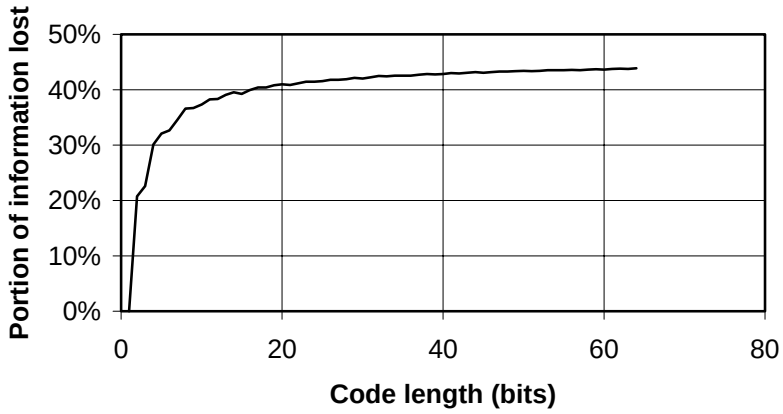

Figure 11. This figure shows how large a portion of information is lost as the relative frequency of the codeword is used instead of the codeword itself, as a function of the codeword length.

CHAPTER 6
PROBABILISTIC ONE-DIMENSIONAL DISTRIBUTION CODE

We will now study a probabilistic one-dimensional distribution code.

This particular code could be seen either as a simple model for the codes found in biological neural systems. Its simplicity makes it also ideal for artificial neural system implementations. Being probabilistic, this code also complements our previous case study on the deterministic one-dimensional distribution code.

6.1 CODING

Once again, we define the code in terms of the way the encoding is done.

Let us assume that we have some number s that falls between 0 and 1. This is the number that we would like to encode.

35) $$s \in [0,1]$$

This code is called probabilistic because the codeword $\mathbf{x}$ for a given number s is chosen randomly. In other words, we do not know what the codeword is going to be, based on s. We only know the distribution from which the codeword is to be drawn. Let us assume the $\mathbf{x}$ comprises l elements:

36) $$\mathbf{x} = [x_1 x_2 \ldots x_l] \in \{0,1\}^l$$

Each of those elements is drawn randomly from identical but independent distributions where the probability of the element being equal to 1 is s.

37) $$\forall\left(i \in \{1,2,\ldots,l\}\right) \quad \Pr\{x_i = 1\} = s$$

Obviously, the expected value of the relative frequency is s.

6.2 MUTUAL INFORMATION

In section 4.2 we found an upper limit for the channel capacity of any one-dimensional distribution code and we have already shown that the channel capacity of the deterministic one-dimensional distribution code meets that upper limit.

It would be interesting to find out whether this particular probabilistic one-dimensional distribution code can also meet the upper limit. Unfortunately, we do not have an analytic solution for the channel capacity of this code, so far. Therefore, we must settle for less and study the amount of mutual information between s and the relative frequency of $\mathbf{x}$. We shall assume uniform distribution for s so that we can make comparisons to the previous study on the deterministic code.

We shall use random variables $\mathbf{S}$ and $\mathbf{T}$ corresponding to s and the relative frequency of $\mathbf{x}$. Thus, we are trying to learn the mutual information $I(\mathbf{S};\mathbf{T})$ between the two random variables.

To compute $I(\mathbf{S};\mathbf{T})$ we shall start with the standard definition of mutual information of two continuous random variables or

38) $$I(\mathbf{S};\mathbf{T}) = h(\mathbf{S}) + h(\mathbf{T}) - h(\mathbf{S},\mathbf{T})$$

The problem here is that the definition would require not only $\mathbf{S}$ but also $\mathbf{T}$ to be continuous. To do that, we will denote the values that $\mathbf{T}$ can take by

39) $$z_i = \frac{i}{l}, \quad i \in \{0,1,\ldots,l\}$$

Using this, we can write a probability density function of $\mathbf{T}$ as a continuous function:

40) $$p_{\mathbf{T}}(t) = \begin{cases} \Pr\{\mathbf{T} = z_i\}\delta_{z_i}(t), & \{\forall t, i \,|\, t = z_i, i \in \{0,1,\ldots,l\}\} \\ 0 \quad , & \textit{otherwise} \end{cases}$$

where $\delta_{z_i}(\ )$ is the Dirac function.

Now that we have a better understanding of the nature of $\mathbf{T}$, let us look at Equation 38 again. It comprises three different entropies. The first one, $h(\mathbf{S})$, is easy to solve since we have already agreed that $\mathbf{S}$ is uniformly distributed between 0 and 1. Therefore, it is 0:

41) $$h(\mathbf{S}) = -\int_0^1 1\log(1)ds = 0$$

For joint entropy $h(\mathbf{S},\mathbf{T})$, we must first observe that the encoding algorithm and the joint probability distribution it produces. The algorithm starts with the value s of $\mathbf{S}$, which is uniformly distributed between 0 and 1. Once s is fixed, the algorithm obeys the binary distribution function when randomly drawing the bits of the codeword. Keeping with the approach used in Equation 40, we can write the joint probability function $p_{\mathbf{S},\mathbf{T}}(s,t)$ as

42) $$p_{\mathbf{S},\mathbf{T}}(s,t) = \begin{cases} \binom{l}{i} s^i (1-s)^{l-1} & , \quad \{t, i \,|\, t = z_i, i \in \{0,1,\ldots,l\}\} \\ 0 & , \quad \textit{otherwise} \end{cases}$$

Somewhat exceptionally for a binary distribution function, s is also an argument of the function here.

This result lets us now write the joint entropy $h(\mathbf{S},\mathbf{T})$ in terms of probability densities

43) $$h(\mathbf{S},\mathbf{T}) = -\int_0^1 \sum_{i=0}^{l} p_{\mathbf{S},\mathbf{T}}(s,z_i)\log\left(p_{\mathbf{S},\mathbf{T}}(s,z_i)\right)ds$$

from which we can compute numerical values.

One more entropy remains, $h(\mathbf{T})$, and we can find a quite simple expression for it but only after considerable more steps. We shall start by expressing the nonzero terms of the probability density function $p_{\mathbf{T}}(t)$ in Equation 40 in terms of the joint probability function in Equation 42:

44) $$p_{\mathbf{T}}(z_i) = \Pr\{\mathbf{T} = z_i\}\delta_{z_i}(z_i) = \int_0^1 p_{\mathbf{S},\mathbf{T}}(s,z_i)ds$$

or, expanded,

45) $$p_{\mathbf{T}}(z_i) = \int_0^1 \binom{l}{i} s^i (1-s)^{l-i}\, ds$$

That, on the other hand can be written using Euler gamma functions so that

46) $$p_{\mathbf{T}}(z_i) = \binom{l}{i} \frac{\Gamma(1+l-i)\Gamma(1+i)}{\Gamma(l+2)}$$

The arguments of the Euler gamma are, in this case, integers. Therefore, it is possible to replace the gammas with factorials. As we do that, we get the equation

47) $$p_{\mathbf{T}}(z_i) = \binom{l}{i} \frac{(l-i)!\,i!}{(l+1)!}$$

and, further simplified

48) $$p_{\mathbf{T}}(z_i) = \binom{l}{i} \frac{(l-i)!i!}{l!} \frac{1}{l+1} = \frac{1}{l+1}$$

Finally, we have the probability distribution $\mathbf{T}$ in a form that is easy to use:

49) $$p_{\mathbf{T}}(t) = \begin{cases} \dfrac{1}{l+1} & , \quad \{\forall t, i \,|\, t = z_i, i \in \{0,1,\ldots,l\}\} \\ 0 & , \quad \textit{otherwise} \end{cases}$$

and we can write the formula for the entropy of $\mathbf{T}$, which is

50) $$h(\mathbf{T}) = -\sum_{i=0}^{l} \frac{1}{l+1} \log\left(\frac{1}{l+1}\right) = \log(l+1)$$

All that remains is writing out the formula for mutual information, which is

51) $$I(\mathbf{S};\mathbf{T}) = \log(l+1) + \int_0^1 \sum_{i=0}^{l} \binom{l}{i} s^i (1-s)^{l-i} \log\left(\binom{l}{i} s^i (1-s)^{l-i}\right) ds$$

Figure 12 plots out the mutual information $I(\mathbf{S};\mathbf{T})$ as a function of the codeword length l. We do not know the channel capacity of this particular one-dimensional distribution code but, instead, Figure 12 shows the upper limit of the channel capacity, as of the entire family of one-dimensional distribution codes.

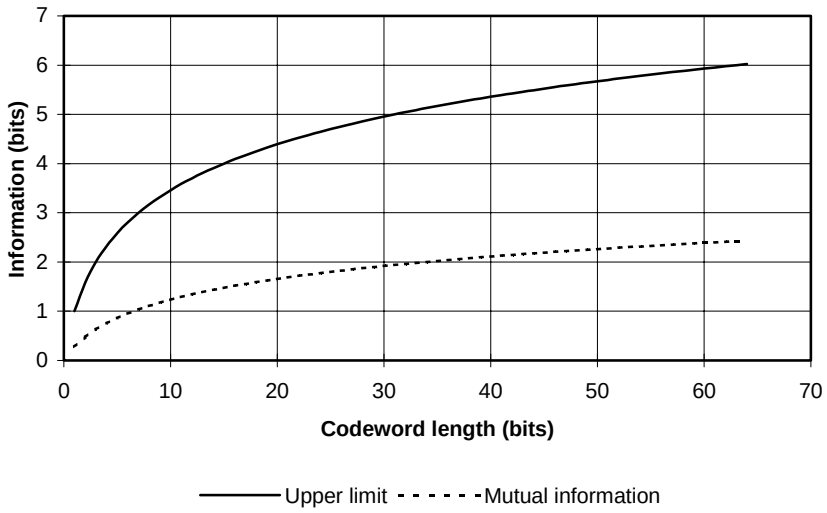

Figure 12. This figure shows the mutual information between the value of the source variable, which is drawn from uniform distribution, and the relative frequency of the codewords. For comparison, this figure also shows what is the upper limit of the channel capacity of any one-dimensional distribution code. These both are given as functions of the codeword length.

Assuming uniform distribution of the values that are encoded, the probabilistic one-dimensional distribution code is not as efficient as the deterministic one. However, we do not know what the channel capacity of the probabilistic code is. Therefore, we do not know whether the probabilistic is inherently less efficient than the deterministic code or whether the apparent inefficiency is only due to unfavourable choice of distribution. After all, uniform distribution utilises the channel capacity of the deterministic code almost completely.

6.3 CODEWORDS

When we discussed the deterministic one-dimensional distribution code, we noticed that the code could produce several codewords having the same relative frequency. We asked the question of whether information was lost in the process of computing the relative frequency

and using it instead of the codewords themselves. The answer was positive and we were able to compute the amount of information that was lost.

Once again, we have a code — the probabilistic code this time — which can produce several codewords having the same relative frequency. Hence, it seems reasonable to ask the same question again. Furthermore, there are also some differences, which makes the question even more interesting. This time, the relation between the source variable and the codeword is not a function but a many-to-many relation.

There are several ways to approach this question. One of them we saw in section 5.4. Another way is to study the conditional mutual information $I(\mathbf{S};\mathbf{X}|\mathbf{T})$, where $\mathbf{S}$ is the random variable whose value is encoded, $\mathbf{X}$ is a random variable for the codeword and $\mathbf{T}$ for the relative frequency. In more casual terms, we want to know that, if we already know the relative frequency, can we learn anything more about the value of the source variable if someone tells us the codeword. If the conditional mutual information is zero, we cannot; thus, no information is lost.

One way to express the conditional mutual information $I(\mathbf{S};\mathbf{X}|\mathbf{T})$ is in terms of two different conditional entropies.

52) $$I(\mathbf{S};\mathbf{X}|\mathbf{T}) = H(\mathbf{X}|\mathbf{T}) - H(\mathbf{X}|\mathbf{S},\mathbf{T})$$

Assuming l fixed, $H(\mathbf{X}|\mathbf{T})$ is the conditional entropy of $\mathbf{X}$ if we have been told what the relative frequency is. In other words, we have been told how many 1s and 0s there are in the codeword. Could we learn anything more about the codeword if someone told us the encoded value? No, we could not, since all codewords in the set of codewords that have the given number of 1s and 0s are equally probable. We already know that the codeword is in that set, but knowing the encoded value does not give us any clues for choosing one member of the set over another.

Thus,

53) $$H(\mathbf{X}|\mathbf{T}) = H(\mathbf{X}|\mathbf{S},\mathbf{Z})$$

from which it follows that

54) $$I(\mathbf{S};\mathbf{X}|\mathbf{T}) = 0$$

Note that no assumptions were made about the distribution of $\mathbf{S}$. Therefore this result holds regardless of the distribution.

One could argue that the probabilistic code is inherently a one-dimensional distribution code, unlike the deterministic code. This is because the probabilistic code does not carry any additional information about the encoded value in its codewords besides the information found in the relative frequency.

The probabilistic one-dimensional distribution code is more like the codes found in biological neural systems than the deterministic code. However, it would be an error to conclude that the biological codes rely solely on the relative frequency — that is, to conclude that biological neural systems rely solely on the firing rate. The probabilistic one-dimensional distribution code is not, after all, compatible with even those quite simple requirements that we set forth previously when discussing biological codes earlier. It does not, for instance, allow for the existence of the absolute refractory period except in special cases.

If anything, the probabilistic one-dimensional distribution code is a tool for modelling what could happen *assuming* that biological neural systems use only the firing rate for passing information. In the chapters that follow, we shall use it as such a tool.

Part 2
Neural Systems

CHAPTER 7
DETERMINISTIC MEMORYLESS NEURAL SYSTEMS

Many memoryless artificial neural systems are deterministic in the sense that, once the network has been trained, a given input vector $\mathbf{x}$ yields always the same output vector $\mathbf{y}$. In other words, there is a well-defined *input–output function*:

55) $$\mathbf{y} = f(\mathbf{x})$$

We will analyse how such neural systems process distribution codes.

Since multidimensional distribution codes deal with a matrix, not a vector, it is worth discussing how to combine input–output functions and distribution codes. We previously agreed that each column in the matrix corresponds to the input or the output vector of a neural system at a given time (see Figure 2). On the other hand, the input–output function applies to a single input vector and a single output vector. Therefore, the most natural combination would be mapping a sequence of input vectors $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_l$ to an equally long sequence of output vectors $\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_l$ so that

56) $$\mathbf{y}_i = f(\mathbf{x}_i)$$

is satisfied. This equation also guarantees that for any given permutation of the sequence $\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_l$ of output vectors there is a corresponding permutation of the sequence $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_l$ of input vectors and vice versa.

When the sequence of output vectors is decoded the result does not depend on which permutation is being used. This is because the encoding

always treats the sequence as a statistical sample, that is, as an unordered set.

Some distribution codes, such as the deterministic code in Chapter 5, are based on decoding which always produces the same permutation of the same sequence of input vectors. Obviously, this has no effect on the outcome of decoding the output vectors as the neural system processes each input vector independently of all others.

All this enables us to substitute the sequence of input and output vectors with the input distribution $\tilde{\mathbf{X}}$ and output distribution $\tilde{\mathbf{Y}}$, respectively, without loosing any aspects of the distribution codes. In this case, $\tilde{\mathbf{X}}$ and $\tilde{\mathbf{Y}}$ are normalised sample distributions, which give the relative frequencies of the vectors in the samples or, in other words, in the sequence.

Thus we can express the combination of a distribution code and a deterministic neural system as:

$$\mathbf{s} \xrightarrow{\text{encoding}} \tilde{\mathbf{X}} \xrightarrow{\text{input - output function}} \tilde{\mathbf{Y}} \xrightarrow{\text{decoding}} \mathbf{t} \tag{57}$$

Here, $\mathbf{s}$ is a vector of source variables. Let us call it the source vector. By encoding the source vector we get the input distribution $\tilde{\mathbf{X}}$. Given the input distribution, the neural system yields an output distribution $\tilde{\mathbf{Y}}$. The input–output function defines the relation between $\tilde{\mathbf{X}}$ and $\tilde{\mathbf{Y}}$. Decoding produces a vector $\mathbf{t}$ of target variables based on $\tilde{\mathbf{Y}}$. We will call this the target vector.

In section 7.2 we walk through a concrete example using these concepts. Before doing that, let us introduce one more concept, namely the distribution function.

7.1 DISTRIBUTION FUNCTION

The input–output function deals with individual vectors and not with distributions. It would be nice to have a function – let us call it the *distribution function* – describing the behaviour of the neural system in terms of distributions. We will shortly find that such a distribution

function exists for any deterministic memoryless neural system. This would change the preceding expression only slightly:

58) $$\mathbf{s} \xrightarrow{\text{encoding}} \tilde{\mathbf{X}} \xrightarrow{\text{distribution function}} \tilde{\mathbf{Y}} \xrightarrow{\text{decoding}} \mathbf{t}$$

Let us introduce two vectors of random variables, $\mathbf{X} = [X_1 X_2 \ldots X_n]$ and $\mathbf{Y} = [Y_1 Y_2 \ldots Y_m]$, whose distributions are the above-mentioned $\tilde{\mathbf{X}}$ and $\tilde{\mathbf{Y}}$, respectively. This way, we can write the two distributions as

59) $$\tilde{\mathbf{X}} \equiv \left\{ \left\langle \mathbf{x}, \Pr\{\mathbf{X} = \mathbf{x}\} \right\rangle \middle| \, \mathbf{x} \in \{0,1\}^n \right\}$$

60) $$\tilde{\mathbf{Y}} \equiv \left\{ \left\langle \mathbf{y}, \Pr\{\mathbf{Y} = \mathbf{y}\} \right\rangle \middle| \, \mathbf{y} \in \{0,1\}^m \right\}$$

where we have ordered pairs of binary vectors and their probabilities. Note that n it is not required to equal m.

We could define the distribution function as a function $d_{\tilde{\mathbf{Y}}}$ from distributions of $\tilde{\mathbf{X}}$ to distributions of $\tilde{\mathbf{Y}}$ that satisfies

61) $$\tilde{\mathbf{Y}} = d_{\tilde{\mathbf{Y}}}\left(\tilde{\mathbf{X}}\right)$$

for all $\tilde{\mathbf{X}}$. It would be conceptually appealing. In practice, however, it is rather inconvenient to use. To solve $\tilde{\mathbf{Y}}$, it is sufficient to solve the joint probability distribution function of $\mathbf{Y}$ because all else is given. Thus, we will say that the distribution function is any function yielding $p_{\mathbf{Y}}(\mathbf{y})$

62) $$p_{\mathbf{Y}}(\mathbf{y}) \equiv \Pr\{\mathbf{Y} = \mathbf{y}\}$$

for all $\mathbf{y} \in \{0,1\}^m$ for any given $\tilde{\mathbf{X}}$.

In general, $p_{\mathbf{Y}}(\mathbf{y})$ is not a function of the value of $\mathbf{X}$ nor is it a function of the probability of any single point. Instead, it is a function of the overall probability distribution of $\mathbf{X}$, that is, of $\tilde{\mathbf{X}}$.

7.2 DISTRIBUTION FUNCTION OF DETERMINISTIC MEMORYLESS NEURAL SYSTEM

In the case of deterministic memoryless neural system, a given input vector always yields the same output vector. Therefore, if we sum together all the probabilities of all the input vectors that produce a given output vector, we get the probability of that output vector. Hence, the distribution function for a deterministic neural system is

63)
$$p_{\mathbf{Y}}(\mathbf{y}) = \sum_{\mathbf{y}=f(\mathbf{x})} \Pr\{\mathbf{X} = \mathbf{x}\}$$

for any given distribution $\tilde{\mathbf{X}}$. From this we can conclude that determinism in mapping individual vectors implies determinism in mapping distributions.

We will now walk through an example. Let us have three source variables that together form the vector $\mathbf{s}$. We choose a suitable range for the variables just to keep the example simple.

64)
$$s_i \in (0, 1], \quad \text{where } i \in \{1, 2, 3\}$$
$$\mathbf{s} = \begin{bmatrix} s_1 \\ s_2 \\ s_3 \end{bmatrix}$$

The notation that we use to describe the input distribution $\tilde{\mathbf{X}}$ is a set of ordered pairs. The first element in the pair is binary vector and the second pair is the probability that such a vector is an input to the neural system. In this example the neural system has two inputs, which implies that 4 different input vectors need to be considered.

We define the encoding by giving the probabilities of those vectors in terms of $\mathbf{s}$. Even the simple encoding that was chosen for this example succeeds to encode three source variables with just two input bits without losing information. Note, that we must do normalisation to guarantee that the sum of the probabilities of the vectors equals 1.

65)
$$\begin{aligned}\tilde{\tilde{\mathbf{X}}} &= \left\{\left\langle\begin{bmatrix}0\\0\end{bmatrix}, \Pr\left\{\mathbf{X}=\begin{bmatrix}0\\0\end{bmatrix}\right\}\right\rangle, \left\langle\begin{bmatrix}0\\1\end{bmatrix}, \Pr\left\{\mathbf{X}=\begin{bmatrix}0\\1\end{bmatrix}\right\}\right\rangle,\right.\\ &\quad \left.\left\langle\begin{bmatrix}1\\0\end{bmatrix}, \Pr\left\{\mathbf{X}=\begin{bmatrix}1\\0\end{bmatrix}\right\}\right\rangle, \left\langle\begin{bmatrix}1\\1\end{bmatrix}, \Pr\left\{\mathbf{X}=\begin{bmatrix}1\\1\end{bmatrix}\right\}\right\rangle\right\}\\ &= \left\{\left\langle\begin{bmatrix}0\\0\end{bmatrix}, \frac{s_1}{2\sum_{i=1}^{3} s_i}\right\rangle, \left\langle\begin{bmatrix}0\\1\end{bmatrix}, \frac{s_1}{2\sum_{i=1}^{3} s_i}\right\rangle, \left\langle\begin{bmatrix}1\\0\end{bmatrix}, \frac{s_1}{2\sum_{i=1}^{3} s_i}\right\rangle, \left\langle\begin{bmatrix}1\\1\end{bmatrix}, \frac{\sum_{i=1}^{3} s_i}{2}\right\rangle\right\}\end{aligned}$$

The input–output function gives the output as a function of the input vectors. Let the neural system have a single output and the input–output function be the XOR function.

In this simple case it is easy to see that the output distribution is such that the probability of the output being 1 is the sum of probabilities in the two cases where one of the input bits is 0 and the other is 1:

66)
$$\begin{aligned}\tilde{\tilde{\mathbf{Y}}} &= \left\{[0], \langle \Pr\{Y=[0]\}\rangle, \langle[1], \Pr\{Y=[1]\}\rangle\right\}\\ &= \left\{\left\langle[0], \Pr\left\{\mathbf{X}=\begin{bmatrix}0\\0\end{bmatrix}\right\} + \Pr\left\{\mathbf{X}=\begin{bmatrix}1\\1\end{bmatrix}\right\}\right\rangle, \left\langle[1], \Pr\left\{\mathbf{X}=\begin{bmatrix}1\\0\end{bmatrix}\right\} + \Pr\left\{\mathbf{X}=\begin{bmatrix}0\\1\end{bmatrix}\right\}\right\rangle\right\}\end{aligned}$$

7.3 DIFFERENT DISTRIBUTIONS

In section 7.1, we assumed that $\tilde{\mathbf{X}}$ and $\tilde{\mathbf{Y}}$ were probability distributions. Yet, in the beginning of this chapter we described them as

normalised sample distributions. In other words, we have been using the two concepts interchangeably. Such practice is more or less a corner stone in mathematical statistics but still, a few words of caution are in place.

Let us assume that we have been given the normalised input sample distribution $\tilde{\mathbf{X}}$ and the neural system is deterministic. Applying Equation 63 will correctly give us the output distribution or $\tilde{\mathbf{Y}}$, which is also a normalised sample distribution. On the other hand, if $\tilde{\mathbf{X}}$ is a probability distribution, so will be $\tilde{\mathbf{Y}}$. As for applying Equation 63 and getting the result, it makes no difference which kind of distribution we are dealing with.

The key difference comes in interpreting the result, which in this case means decoding. Let us use an example where we have a one-dimensional case (i.e., $n = m = 1$): There are two different normalised input sample distributions, $\tilde{\mathbf{X}}_1$ and $\tilde{\mathbf{X}}_2$, which yield two different normalised output sample distributions $\tilde{\mathbf{Y}}_1$ and $\tilde{\mathbf{Y}}_2$, respectively. The sample of $\tilde{\mathbf{Y}}_1$ contains 1s and 0s in equal amounts while $\tilde{\mathbf{Y}}_2$ is all 0s. We are asked to predict the outcome of an experiment where there is a 0.25 probability that the input distribution is $\tilde{\mathbf{X}}_1$ and 0.75 probability that it is $\tilde{\mathbf{X}}_2$.

One approach is to combine the two normalised input sample distributions weighting them according to their respective probabilities:

67) $$\tilde{\mathbf{X}} = 0.25\tilde{\mathbf{X}}_1 + 0.75\tilde{\mathbf{X}}_2$$

In this case, $\tilde{\mathbf{X}}$ would obviously be a probability distribution. Applying Equation 63 we would get a probability distribution $\tilde{\mathbf{Y}}$ where there is a 0.125 probability for a 1 and 0.875 probability for a 0. Thus, we would conclude that the expected value of the target variable is 0.125. This is correct and in an infinitely long sequence of experiments the mean target variable value would converge towards 0.125.

However, we could take another approach and apply Equation 63 twice; once to $\tilde{\mathbf{X}}_1$ and once to $\tilde{\mathbf{X}}_2$. This would give us two output distributions. Since they would be normalised sample distributions, decoding them would give the actual values of the target variable – 0.5 and 0, respectively – as opposed to its expected values.

Thus, it is possible to use either kind of distribution, but one should be careful in how to interpret the results.

One might argue that sample distributions should be always used because it gives the most information about the situation. For instance, we could have computed the expected values from the actual values of the target variable in the example, above, while the opposite would have been impossible

There are, however, situations where there is randomness involved and where probability distributions are much more appropriate. Consider a source vector with a thousand random variables or consider a single source variable that needs to be coded using a probabilistic distribution code with very long codewords. In both cases, there are too many combinations of normalised sample distributions to deal with. If the nature of the problem is probabilistic so that it can be solved using a probability distribution, this is often the easier way.

The problem of learning a probability distribution when a sample distribution is given is discussed, e.g., in (Bialek et al. 1996).

7.4 RECURSIVE FORM

The above definition of the distribution function of a deterministic neural system is probably handy for some numerical problems. It does not, however, give very much leverage for further analysis. Therefore, we would like to find an alternative way of expressing $p_{\mathbf{Y}}(\mathbf{y})$. In particular, we are looking for a recursive expression for $p_{\mathbf{Y}}(\mathbf{y})$.

Let us start by introducing some notation for both input and output distributions. We have already denoted the probability distribution function of $\mathbf{Y}$ by $p_{\mathbf{Y}}(\mathbf{y})$. Analogously, we will denote the probability distribution function of $\mathbf{X}$ by $p_{\mathbf{X}}(\mathbf{x})$. We will use the following shorthand notation for conditional probability distribution functions:

$$p_{\mathbf{X}}(\mathbf{x})_{z_k, z_{k+1}, \ldots, z_t} = \Pr\left\{X_1 = x_1, X_2 = x_2, \ldots, X_{k-1} = x_{k-1} \,\middle|\, X_k = z_k, X_{k+1} = z_{k+1}, \ldots, X_t = z_t\right\} \quad (68)$$

In other words, this is a conditional probability distribution function for the first $k-1$ bits of $\mathbf{X}$, while the values of the remaining bits are given in the subscript.

A similar notation is introduced for $\mathbf{Y}$. However, we still want the subscript to stand for the bits of $\mathbf{X}$**;** not for the bits of $\mathbf{Y}$ (i.e., all m bits of $\mathbf{Y}$ are free to vary):

$$
\begin{aligned}
& p_{\mathbf{Y}}(\mathbf{y})_{z_k, z_{k+1}, \ldots, z_n} \\
&= \Pr\{Y_1 = y_1, Y_2 = y_2, \ldots, Y_m = y_m \mid X_k = z_k, X_{k+1} = z_{k+1}, \ldots, X_n = z_n\} \\
&= \Pr\{\mathbf{Y} = \mathbf{y} \mid X_k = z_k, X_{k+1} = z_{k+1}, \ldots, X_n = z_n\}
\end{aligned}
\tag{69}
$$

The subscript always gives the values of the last $n-k+1$ bits of $\mathbf{X}$. Because we have a full freedom in choosing the permutation of bits when we assign them their indexes, the notation can be used to assign any bits of $\mathbf{X}$ their values by choosing the right permutation.

It is possible to write a recursive expression for $p_{\mathbf{Y}}(\mathbf{y})$. First we state that $p_{\mathbf{Y}}(\mathbf{y})$ conforms to the notation introduced in Equation 69 since $p_{\mathbf{Y}}(\mathbf{y})$ is the limiting case where the subscript is nonexistent and, therefore, none of the bits of $\mathbf{X}$ are given.

Let us now expand the right-hand side of Equation 69 by splitting the sum into two according to the value of X_n. This gives us

$$
\begin{aligned}
& \sum_{f(\mathbf{x})=\mathbf{y}} \Pr\{\mathbf{X} = \mathbf{x}\} \\
&= \sum_{f([x_1 \ldots x_{n-1}\ 0])=\mathbf{y}} \Pr\{X_1 = x_1, \ldots, X_{n-1} = x_{n-1}, X_n = 0\} \Pr\{X_n = 0\} \\
&\quad + \sum_{f([x_1 \ldots x_{n-1}\ 1])=\mathbf{y}} \Pr\{X_1 = x_1, \ldots, X_{n-1} = x_{n-1}, X_n = 1\} \Pr\{X_n = 1\}
\end{aligned}
\tag{70}
$$

This can also be written as

$$
p_{\mathbf{Y}}(\mathbf{y}) = p_{\mathbf{Y}}(\mathbf{y})_0 \Pr\{X_n = 0\} + p_{\mathbf{Y}}(\mathbf{y})_1 \Pr\{X_n = 1\} \tag{71}
$$

which shows the first step of recursion. The same line of reasoning gives us the general recursive rule for deterministic neural systems:

72) $$p_{\mathbf{Y}}(\mathbf{y})_{z_{k+1},\ldots,z_n} = p_{\mathbf{Y}}(y)_{0,z_{k+1},\ldots,z_n} \Pr\{X_k = 0\} + p_{\mathbf{Y}}(y)_{1,z_{k+1},\ldots,z_n} \Pr\{X_k = 1\}$$

These equations show that we can express the output distribution – and, therefore, the distribution function – as a linear combination of two lower level distributions. This can be done recursively until all the bits of $\mathbf{X}$ have been assigned values.

Just to give a feeling of how this could work in practice, let us look at how the first two levels of recursion would behave:

73)
$$\begin{aligned}
& p_{\mathbf{Y}}(\mathbf{y}) \\
&= p_{\mathbf{Y}}(\mathbf{y})_0 \Pr\{X_n = 0\} + p_{\mathbf{Y}}(\mathbf{y})_1 \Pr\{X_n = 1\} \\
&= \left(p_{\mathbf{Y}}(\mathbf{y})_{00} \Pr\{X_{n-1} = 0 \mid X_n = 0\} + p_{\mathbf{Y}}(\mathbf{y})_{10} \Pr\{X_{n-1} = 1 \mid X_n = 0\}\right) \\
&\quad \Pr\{X_n = 0\} \\
&\quad + \left(p_{\mathbf{Y}}(\mathbf{y})_{01} \Pr\{X_{n-1} = 0 \mid X_n = 1\} + p_{\mathbf{Y}}(\mathbf{y})_{11} \Pr\{X_{n-1} = 1 \mid X_n = 1\}\right) \\
&\quad \Pr\{X_n = 1\} \\
&= p_{\mathbf{Y}}(\mathbf{y})_{00} \Pr\{X_{n-1} = 0, X_n = 0\} \\
&\quad + p_{\mathbf{Y}}(\mathbf{y})_{10} \Pr\{X_{n-1} = 1, X_n = 0\} \\
&\quad + p_{\mathbf{Y}}(\mathbf{y})_{01} \Pr\{X_{n-1} = 0, X_n = 1\} \\
&\quad + p_{\mathbf{Y}}(\mathbf{y})_{11} \Pr\{X_{n-1} = 1, X_n = 1\}
\end{aligned}$$

In this case, we would get a linear combination of four distributions and, obviously, if we would decide to extend the recursion all the way, we would finally get a linear combination of 2^n distributions.

Recursion of this kind could be used for describing the distribution function but that would be, arguably, quite inefficient. In computer programs, rounding issues may also arise if the dimensionality is high thus inevitable leading to small probability values. Instead of high practical value, this exercise with recursion has shown us some of the structure of the distribution function. We will later use these results for further analysis.

7.5 MONOTONIC FUNCTION

A deterministic memoryless neural system – say, a typical feed-forward neural network – is a rather common place in the literature and in applications.

One might be tempted to apply some distribution code – say, a PDM code – without any further thought about its feasibility. The lessons learned from Minsky and Papert (1969), who proved that a single-layer perceptrons are inherently incapable of learning the XOR function, should keep us from making such assumption. In fact, it will indeed turn out that certain distribution codes combined with deterministic neural systems suffer from similar inability: The neural systems cannot learn other than monotonic functions.

In this section we give the impossibility proof and in later chapters we will study various remedies – an undertaking that gives interesting insights to various aspects of neural systems.

Let us assume that we have a number of source variables. We also have a dedicated physical input in the neural system for each source variable. In other words, we encode each source variable using a one-dimensional distribution code and feed each codeword to a physical input corresponding to that particular source variable. Furthermore, we assume that the codewords are mutually independent. The probabilistic one-dimensional distribution code, which we discussed in Chapter 6, would fulfill these requirements assuming that the source variables s_i are in $[0,1]$

74) $$\Pr\{X_i = 1\} = s_i$$

for $i \in \{1,\ldots,n\}$.

We will focus on a single output of such a neural system. Therefore, the entire distribution of the output is described if we can find the function describing the probability of the output being 1. Based on Equation 71 we can write the probability as

75) $$\begin{aligned} p_Y(1) &= p_Y(1)_0 \Pr\{X_n = 0\} + p_Y(1)_1 \Pr\{X_n = 1\} \\ &= p_Y(1)_0 (1 - \Pr\{X_n = 1\}) + p_Y(1)_1 \Pr\{X_n = 1\} \end{aligned}$$

Let us now show that this function is monotonic with respect to any of the source variables s_i. Since the indexes assigned to the physical inputs can be permuted freely (see page 54), it is sufficient to show that $p_Y(1)$ is monotonic with respect to s_n, that is, with respect to $\Pr\{X_n = 1\}$.

For it to be meaningful to say that a multivariable function is monotonic with respect to one of its variables implies that the other source variables – and hence also $\Pr\{X_1 = x_1\}$, $\Pr\{X_2 = x_2\}$, …, $\Pr\{X_{n-1} = x_{n-1}\}$ – have fixed values while the value of the one variable changes. We will return to the implications of this later, in Chapter 8 and in Chapter 10.

To show the monotonic nature of Equation 75 we will start by looking at the first term, or $p_Y(1)_0$. First we write it as a sum. Since the codewords are mutually independent, we can then express the conditional probability with as a product.

76)
$$\begin{aligned} p_y(1)_0 &= \sum_{f([x_1,\ldots x_{t-1},0])=1} \Pr\{X_1 = x_1,\ldots, X_{n-1} = x_{n-1} | X_n = 0\} \\ &= \sum_{f([x_1,\ldots x_{n-1},0])=1} \prod_{i=1}^{n-1} \Pr\{X_i = x_i\} \end{aligned}$$

This is constant with respect to $\Pr\{X_n = 1\}$. Let us denote that constant by C_0. In analogous way, it is possible to show that also $p_Y(1)_1$ is constant with respect to $\Pr\{X_n = 1\}$. We will denote that constant by C_1. Hence, we can give $p_Y(1)$ a form that is clearly a monotonic function of s_n:

77)
$$\begin{aligned} p_Y(1) &= C_0(1 - \Pr\{X_n = 1\}) + C_1 \Pr\{X_n = 1\} \\ &= C_0 + (C_1 - C_0)\Pr\{X_n = 1\} \\ &= C_0 + (C_1 - C_0)s_n \end{aligned}$$

Thus, the distribution function is a monotonic function of each source variable.

CHAPTER 8
DYNAMIC DISTRIBUTION-FUNCTION ALTERATION

In this chapter we will discuss what can be achieved by assigning a special role to some of the inputs of a neural system. We will be calling the inputs with the special role control inputs.

The control inputs can be used to alleviate the problem of having a monotonic distribution function. Perhaps more importantly, they introduce a whole new concept of dynamic distribution-function alteration – changing the shape of the input–output function, or the shape of the distribution function, without changing the weights – at the level of individual neurons.

We will start by looking at the behaviour of a neuron when distribution code is not being used.

Consider the input–output function of a binary neuron or a real-valued neuron. Typically, in artificial neural network models the output y of a neuron is

78) $$y = f\left(\sum_i w_i x_i\right)$$

where x_i is an input of the neuron and w_i is the corresponding weight. The exact nature of function f depends on the type of the network. A threshold function is often used for binary neurons. For real-valued neurons a number of different functions can be used. Commonly, those functions are continuous and monotonic and they map any given value onto a limited interval, say $[-1,1]$. Because of the last property, they are often called squashing functions; a term introduced by D.E. Rumelhart, J.L. McClelland and the PDP research group (Anderson 1995).

Assume now that the weights of the network have been fixed, i.e., the network is fully trained. In this situation, the input–output function cannot be altered without readjusting the weights.

We can, however, use some of the inputs as *proper* inputs and assign the remaining inputs a special role of a *control* input. One might expect that the input–output function of the proper inputs could be modified dynamically by altering the values of the control inputs. This can be done, but only in a very limited sense.

Mathematically, the output y would be

79) $$y = f\left(\sum_i w_i x_i + \sum_j w_j c_j\right)$$

where c_j is the value of a control input of the neuron and w_j is the corresponding weight. If we consider this equation as a function of the proper inputs, we realise that the second sum can always be replaced with a single number. Let us call it c:

80) $$y = f\left(\sum_i w_i x_i + c\right)$$

Thus, the only effect on the input–output function that any number of control inputs can have is a translation, i.e., moving the function with respect to the origin while preserving the shape and scale.

This same idea can be in some network models in the form of a constant input, whose weight is being altered during network training, or in the form of an additional training parameter such as a threshold parameter. In either case, readjusting that weight or parameter dynamically is equivalent of adjusting a control input.

On the level of a neural network the control inputs can obviously be used to change the shape of the input–output function, but unless the number of control inputs is large, the control that can be achieved over the shape is quite limited. Note also that there is no advantage in having more than one control input per neuron. Consequently, unless the number of neurons in the network is large, the control that can be achieved over the shape of the input–output function is quite limited.

Next, let us look how neurons using distribution code.

Looking at the following form of distribution function of a deterministic neural system we realise that a single control input X_n creates a weighted sum of two distribution functions of the proper inputs. The shape of the two functions can be quite different and X_n can give them any relative weights.

81) $$p_Y(1) = p_Y(1)_0\left(1 - \Pr\{X_n = 1\}\right) + p_Y(1)_1 \Pr\{X_n = 1\}$$

If we add more control inputs, we can gain more control over the shape of the distribution function. For instance, if there are two control inputs X_n and X_{n-1}, we have the following combination of distribution functions:

82) $$\begin{aligned} p_Y(y) = {} & p_Y(1)_{00} \Pr\{X_{n-1} = 0, X_n = 0\} + p_Y(1)_{10} \Pr\{X_{n-1} = 1, X_n = 0\} \\ & + p_Y(1)_{01} \Pr\{X_{n-1} = 0, X_n = 1\} + p_Y(1)_{11} \Pr\{X_{n-1} = 1, X_n = 1\} \end{aligned}$$

To proceed, let us now assume a situation where all the codewords of the inputs – control inputs as well as the proper ones – are mutually independent. Then, by changing the joint distribution of the control input we can control the shape of the distribution function within the limits of the four distribution functions with predefined shapes.

We can either introduce a relation between the distributions or a dependency between the codewords of the proper inputs and control inputs.

An example of introducing a relation between the distributions would be by encoding the same source once for a proper input and once for a control input so that if the probability distribution of the proper input (X_1) changes so will the probability distribution of the control input (X_n):

83) $$s = \Pr\{X_1 = 1\} = \Pr\{X_n = 1\}$$

Here we assume that the codewords are mutually independent, that is

84) $$\Pr\{X_1 = x_1, X_n = x_n\} = \Pr\{X_1 = x_1\}\Pr\{X_n = x_n\}$$

This can be achieved for instance by applying the probabilistic distribution code of Chapter 6 separately to each of the two inputs.

We will see in Chapter 9 an example of how this kind of introduction of relations between the distributions can make the distribution function nonmonotonic.

Talking about a relation between the distributions is, of course, meaningful only when dynamics of the source variables are involved. For instance, when discussing the monotonic nature, we implicitly allow at least one of the source variables to have more than one value.

In contrast, we can introduce dependencies between the codewords even when considering static situations. Another way of saying that a dependency is introduced between the codewords is to say that the source variables are encoded using a code that introduces dependencies among the input variables or, in other words:

85) $$\exists \mathbf{x} \in \{0,1\}^n : \quad \Pr\{\mathbf{X} = \mathbf{x}\} \neq \prod_{i=1}^{n} \Pr\{X_i = x_i\}$$

If one needs to encode more source variables than there are physical inputs, it is not possible to avoid using such dependencies. This does not mean that such dependencies could not be introduced whenever there is more than one source variable.

Dynamic distribution-function alteration through mutually dependent codewords illustrates a fundamental difference between real-valued neural systems and neural systems using distribution codes. Let us think that we have a biological system that uses a distribution code and a real-valued artificial system mimicking the biological system. Consequently, both systems have the same number of inputs and outputs. The common analogy, as we discussed on page 9, is to map the firing rate of each input and output of the biological system onto the value of the corresponding input or output of the real-valued system.

We could introduce a relation between two inputs of the biological system by, for instance, requiring that the two firing rates are always equal even if they change over time. This is easily mimicked by the real-valued system by requiring that the two corresponding input values are always equal.

In dynamic distribution-function alteration, however, we introduce a dependency between the codewords of one or more inputs or, in other words, in the timing of the spikes going into the biological system. The firing rates of the outputs of the biological systems can be controlled without any change in the firing rates of the inputs.

This is something that the real-valued system cannot mimic. The analogy between the two systems breaks because there is nothing in the domain of the real-valued system corresponding to a dependency between codewords.

The situation does not change much even if we abandon the idea that the input and output values of the real-valued system should correspond to the firing rates of the biological system. No matter how many additional control inputs we add to the real-valued system to mimic the effect of adding dependencies between codewords we still come short. Unlike the biological system – or some other system using distributions codes – the real-valued system must resort to changing the weights in the network if the input-output function should undergo changes beyond simple translation.

CHAPTER 9
BINARY NEURON

9.1 INTRODUCTION

A binary neuron can be used as an excellent case study on the use of distribution codes. A single neuron is the simplest possible neural system. Yet, it is complex enough to allow us to demonstrate the previously discussed ideas. A binary neuron has also interesting properties of its own, which we now have the opportunity to study.

The input–output function of a binary neuron is

86) $$y = f\left(\sum_{i=1}^{n} w_i x_i - t\right)$$

where x_i is an input of the neuron and w_i is the corresponding weight, t is called the threshold. The output y of the neuron is given by the function commonly known as the threshold function, where

87) $$f(x) = \begin{cases} 0 & , \quad x < 0 \\ 1 & , \quad otherwise \end{cases}$$

The input vectors of a binary neuron correspond to the cornerpoints of an n-dimensional hypercube, while the weights of the neuron together with the threshold define a linear dichotomy of the hypercube. There are continuously many different values that the threshold and the weights can be assigned. The number of cornerpoints is, however, limited to 2^n. This

makes the number of different dichotomies also limited. Consequently, the number of effectively different input–output functions is finite. Binary neurons differ from real-valued neurons in this respect as the latter can have continuously many different input–output functions. However, the number of linear dichotomies increases rapidly as n increases.

Also, one could choose a different input–output function so that the dichotomies could be nonlinearly separable. Cover (1964; 1965) discusses extensively different polynomially separable dichotomies and gives an upper bound to their number.

9.2 INDEPENDENT INPUTS

Let us assume that each source variable is encoded using a one-dimensional distribution code and the resulting codeword is fed to the corresponding physical input of the binary neuron. Furthermore, let us assume that the codewords are mutually independent.

In many models in the literature, the input–output functions are:

- monotonic
- continuous
- limited to an interval

A binary neuron is a special case of a memoryless deterministic neural system and we just made the assumptions that we used in proving that the distribution function is monotonic. Thus, we know that the distribution function of a binary neuron is also monotonic. The value of the function is limited to an interval – $[0,1]$ to be specific – because it is a probability. Equation 63 clearly implies that the function is continuous.

Therefore, we now know that the main characteristics of the distribution function of a binary neuron are similar to those commonly associated with the input–output functions.

In Figure 13 we see some sample cross-sections of the distribution function. The most note-worthy points in the figure are that:

- The function is in this case S-shaped, which is similar to the input–output function of a real-valued neuron.
- The shape is slightly different for each threshold value, unlike in the case of a real-valued neuron.

- The S-shape becomes steeper the more inputs the neuron has.

As a curiosity, the output y of the binary neuron can in this case be given in terms of a cumulative binomial distribution as follows:

88) $$y = 1 - \sum_{k=0}^{t-1} \binom{n}{k} p^k (1-p)^{n-k}$$

Here n is the number of inputs, t is the threshold, and p is the probability of the inputs being 1.

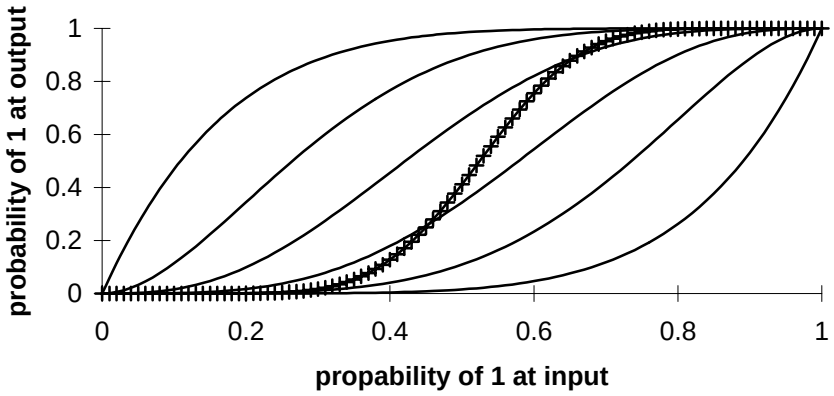

Figure 13. Seven samples of the shape of the distribution function. The codewords are assumed to be independent of one another and all the weights have been set to 1. The figure gives a cross section of the function when all the inputs have the same marginal probability of being 1. The horizontal axis gives this probability. The smooth curves represent six neurons of six inputs. Each neuron has a different threshold. The seventh curve, different in appearance, represents a neuron of 20 inputs and a threshold of 10.

9.3 UNRELATED AND INDEPENDENT CONTROL INPUTS

Let us look at what happens if we add a control input to a binary neuron. In this example, all the codewords will be independent and we do not introduce any relation between the control inputs and proper inputs; that is, the control inputs are assumed to have constant values.

Let us have a binary neuron that has sixty inputs and all the weights have been set to 1. The threshold is 25. We add a control input whose weight is –20. The control input would allow us to dynamically choose between different linear combinations of two distribution functions, which would correspond to thresholds 5 and 25 if there were no control input. This is illustrated in .

9.4 RELATED, INDEPENDENT INPUTS

Let us now consider the dynamics of a case where the distributions of two inputs are related while their codewords are mutually independent.

We will use a binary neuron with two inputs and a single source variable. We will set the weights and the threshold so that the output is 1

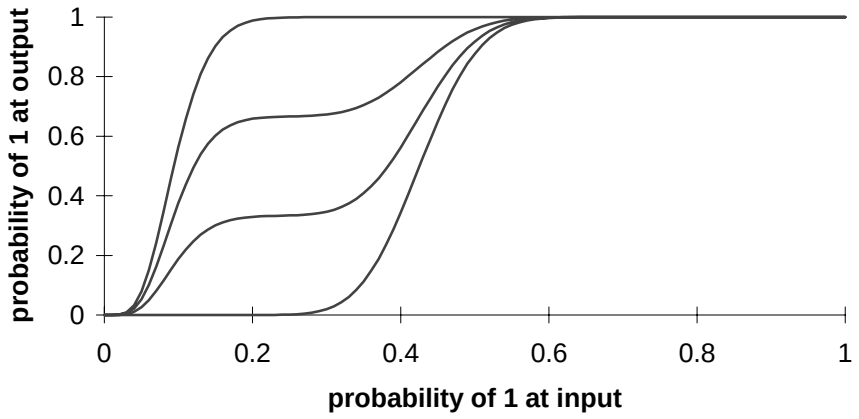

Figure 14. The effect of a control input as explained in text Let us have a binary neuron that has sixty inputs and all the weights have been set to 1. The

if and only if the first input is 1 and the second input is 0. For instance, $w_1 = 1.4$, $w_2 = -0.5$ and $t = 1.0$ would be a valid combination.

The relation between the distributions is

89) $$\Pr\{X_1 = 1\} = \Pr\{X_2 = 1\} = s$$

The resulting distribution function, given in terms of the source variable, is

90) $$\Pr\{Y = 1\} = s(1 - s)$$

which is a nonlinear and nonmonotonic function of s. Figure 15 illustrates the function.

9.5 DEPENDENT INPUTS

One of the most interesting cases is where the codewords are not mutually independent. To demonstrate such a case, we will consider a binary neuron that has two inputs. The weights and the threshold are, again, $w_1 = 1.4$, $w_2 = -0.5$ and $t = 1.0$. This is a static case; that is, we will consider only one, fixed value of the source variable: $s = 0.5$.

Instead, we will vary the degree of dependency between the codewords of the two inputs. For that purpose we formulate the distributions in terms of a parameter, r, so that

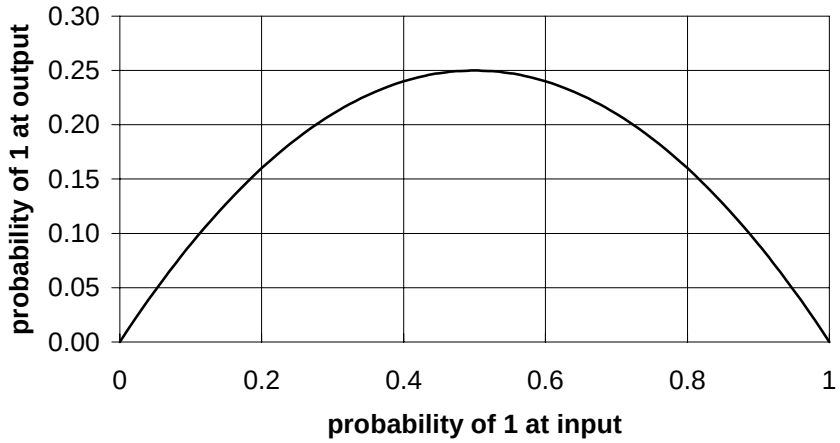

Figure 15. The related, independent inputs. The value of the source variable is given on the horizontal axis while the vertical axis gives the value of the distribution function, which is not monotonic in this case. More details are given in the text above.

91) $$r \in [0,1]$$

92) $$\Pr\{X_1 = 0, X_2 = 0\} = \Pr\{X_1 = 1, X_2 = 1\} = s - rs$$

93) $$\Pr\{X_1 = 0, X_2 = 1\} = \Pr\{X_1 = 1, X_2 = 0\} = rs$$

Thus, when $r = 0$ the codewords are fully correlated or, in other words, the two inputs have always the same value, when $r = 0.5$ the codewords are mutually independent and when $r = 1$ the two inputs have always the opposite values. The resulting distribution function is

94) $$\Pr\{Y=1\} = \Pr\{X_1 = 1, X_2 = 0\} = rs$$

If we consider the distribution function as a function of s we see that r is a parameter by which we can change the distribution function. As the value of r is determined by the nature of encoding, this demonstrates how dependencies among the codewords can be used to control the distribution function that is given as a function of the source variables.

We could have introduced some other kind of dependency but this particular way is interesting because it preserves the marginal distributions constant over r. Therefore it also demonstrates how dependencies among the codewords can be used to control the distribution function that is given as a function of the marginal distributions of the codewords of individual inputs. Here is the distribution function written in that manner:

95) $$\Pr\{Y=1\} = r\Pr\{X_1 = 1\} = r\Pr\{X_2 = 1\}$$

CHAPTER 10
STRUCTURAL AND FUNCTIONAL COMPARISON

We will now look at some common structures of neural networks not found in deterministic neural systems and interpret them in term of how they solve or alleviate the problem of monotonicity discussed in section 7.5. We will also look at some functionality issues keeping the same focus in mind.

To be able to apply distribution codes to the design of artificial neural systems and, in particular, in modelling biological neural systems, one must understand how different structures and functionalities behave when distribution code is being used. This chapter is the first step in that direction.

Let us first discuss a bit about the effect of using inputs whose codewords are independent but whose distributions are related. In particular, we will assume that the distributions are the same, and that the probability of the input to be 1 equals the source variable s.

We have already seen in Chapter 9 how this can be used to create distribution functions that are nonmonotonic functions of the source variable. Assuming that there are two such inputs, we can rewrite Equation 73 – based on the fact that the codewords are independent – as

$$\begin{aligned} p_Y(1) = {} & p_Y(1)_{00}\left(1 - \Pr\{X_n = 1\}\right)^2 \\ & + \left(p_Y(1)_{10} + p_Y(1)_{01}\right)\left(1 - \Pr\{X_n = 1\}\right)\Pr\{X_n = 1\} \\ & + p_Y(1)_{11}\left(\Pr\{X_n = 1\}\right)^2 \\ = {} & p_Y(1)_{00}(1 - s)^2 \\ & + \left(p_Y(1)_{10} + p_Y(1)_{01}\right)s(1 - s) \\ & + p_Y(1)_{11}s^2 \end{aligned} \quad \text{96)}$$

which is obviously a second-degree function of s. Thus, such a function could have at most one minimum or maximum other than at the end points. Obviously, each additional input of this kind could not increase the degree of the polynomial by more than one. This would also be true even if some other valid linear function than $\Pr\{X_i = 1\} = s$ was used to define the (marginal) distributions of some or all the inputs. We will use this in our analysis.

In the following, we will always use a deterministic memoryless neural system as the baseline for comparison. Furthermore, we will assume that the baseline uses distribution coding where each source variable is encoded using a one-dimensional distribution code, that the resulting codewords are each fed to their corresponding dedicated inputs and that the codewords are mutually independent. All the following cases differ from this baseline only in the respects specifically mentioned.

10.1 ENCODING

A trivial approach for creating nonmonotonic distribution functions is to encode the source variables so that the marginal distribution $\Pr\{X_i = 1\}$ of a given input is a suitably chosen function of the given source variable s_i. Obviously, this would be most unsatisfactory because a different function would be needed for almost every case and this would just be moving the information processing away from the network. One could even go one step further and require that the entire distribution function be built into the encoding. That way, the neural network would not be needed at all. Therefore, we rather assume that $\Pr\{X_i = 1\}$ is a linear function s_i such as $\Pr\{X_i = 1\} = s_i$.

One solution, which we have already discussed, is to introduce a relation among the distributions of two or more inputs without introducing codeword dependency. This approach, however, is rather limited in its usefulness as the number of maxima and minima of the distribution functions is limited by the number of inputs.

Another solution would be to encode the source variables using the joint distribution of the inputs beyond just their marginal distributions. The proof that a distribution function is a monotonic function of any

given source variable was based on the assumption that only marginal distributions are used. Thus, the proof does not apply in this case.

The latter solution includes the former as a limiting case: In the former case we could say, for instance, that two marginal distributions $\Pr\{X_{n-1}=1\}$ and $\Pr\{X_n=1\}$ are always equal to the same source variable s and that the two random variables are mutually independent. Obviously, this is a special case of saying how the source variable should be encoded into the joint distribution.

10.2 RECURSION

Let us assume that an output of a neural system is a nonconstant function of a given input and that the output is connected to another input. Recurrent artificial neural networks, for instance, have this kind of structures. The marginal distributions of these two inputs are therefore functions of the same source variable. The output acts also as the external input that is decoded.

In this situation, also the input where the output has been connected can affect the output value and the marginal distribution of the input itself and, at the same time, so can the other input as a new binary value is fed to it. Thus, old values of the external input can effect the value of the output long afterwards.

While the recursion adds memory to the deterministic network it also enables one to avoid or at least lessen the problem of monotonic distribution function.

To model a recursive network one often uses a state vector. If the delay in the recursion loop is longer than one time step, there is more data stored in the state vector of the network, and more combinations of the state vector and the internal inputs can occur. As far as the recursion is concerned, there is no need for the state vector to contain data on such parts of the network that do not contribute to a recursion loop. Thus, if larger portions of the network are involved in the recursion by adding more recursively connected (internal or external) inputs, more data will be in the state vector and, again, more combinations can occur. Similarly, adding more external inputs allows for more combinations and, thus, potential for more complex functions.

Recursion is a particularly interesting way to create nonmonotonic functions as the recursion loops can be very long and as the amount of different combinations can be made rather large with just a few inputs, outputs, recursive connections and steps of delay.

10.3 ROUTES OF UNEQUAL DELAY

When we have discussed deterministic neural systems, we have implicitly assumed that there are no delays introduced by the network or, at least, that we can safely ignore such delays. For instance, if each input vector completely defines the output vector but after a delay, all the mathematics that we have used is valid. It is just a matter of interpreting the meaning of the mathematics correctly when applying it to the physical situation.

A multilayered feed-forward network where each neuron introduces an equal delay meets our criteria for a deterministic neural system. However, if each neuron has a different delay, we have a situation where the effect of the value of a given input reaches a neuron in the hidden layers or in the output layer through several different routes and at different times. Thus, the effect of consequent bits of a codeword of the given input may reach a neuron at the same time. Assuming that the consequent bits of the codeword are mutually independent, this is comparable to having inputs that have related distributions but mutually independent codewords. However, the other inputs of the neural system may also affect the inputs of the neuron at the same time and through the same inputs of the neuron.

Without recursion, the number of routes that have different delays and lead from a given input of the neural system to a given neuron is limited. This may limit how complicated functions this kind of network can reproduce. On the other hand, the marginal distribution of an input of a neuron can be a nonlinear function of a source variable (see page 71), which affects in the opposite direction.

A whole new layer of complexity can be added by assuming that the delays are not constants. See, for instance, (Maass and Schmitt 1997) on the complexity of learning in the presence of programmable delays. Depending on the details, the computational abilities of the network may be increased if the delays are affected by the data passing through the

neurons. Apparently, this idea has not been discussed in the literature and should be studied further in the future.

10.4 NONDETERMINISTIC NEURONS

Nondeterministic neurons can introduce nonmonotonic behaviour to a neural system where distribution coding is used.

From a given sequence of input vectors a deterministic neuron always produces the same sequence of output vectors (assuming that the weights and other parameters of the neuron are kept fixed). This is not true about a nondeterministic neuron.

While the sequence of output vectors produced by a nondeterministic neuron may vary, this does not mean that the probability distribution of those vectors could not remain constant and, at least in theoretical cases, even the sample distribution of the output sequences could do that.

Let us consider a deterministic neural network whose input is such that the distribution function must be a monotonic function of the source variables. The same input vectors that are fed into the network are also fed into a deterministic neuron outside the network. If we add an extra input anywhere in the network (without introducing routes of unequal delay, see section 10.3) for the output of the neuron, the distribution function of the extended network is still bound to be monotonic. This is easily proven since the combination of the neuron and the network is but a deterministic neural system.

Quite different results can achieved by using a nondeterministic neuron. Adding such a neuron to the network creates a situation where the distribution function no longer needs to be monotonic.

This is because the output of the nondeterministic neuron acts as a control input, which we discussed in Chapter 8, as far as the rest of the network is conserned.

After studying the coding fidelity and efficiency of various simulated neural populations Gruner and Johnson (1999) have stated that "… diverse population may be more useful for coding than a homogeneous population in situations of massive convergence of inputs" (p. 167). While their work was not based on mixed populations of nondeterministics and deterministic neurons, it does suggest that such populations could have their uses. The use of nondeterministic neurons

as control inputs ought to be included in any future study that may focus on such mixed populations.

10.5 EXTENDED SAMPLING

One of the limitations of distribution coding is that in its basic form it always assumes that all the elements of an input vector (or an output vector) correspond to the one and same moment in time. This is illustrated in Figure 16.

There are, however, plenty of codes that disperse the encoded information in time. If we want to, say, model biological neural phenomena of that kind, we would need to expand our model. Luckily, there is nothing particular about the mathematics of distribution codes that would prevent us from doing that.

Figure 17 shows how this could be done. The vectors in the sample do not need to correspond to the column of the matrix. Instead, the vectors can include a varying number of consecutive or nonconsecutive values of a given physical input (or output) of the neural system. Also, the vectors may or may not overlap with each other and there is not necessarily a vector for each time step but, say, every second time step and so on.

Some of such extensions also lead to nonmonotonic functions. For instance, if the vectors include several values of a given physical output, it means that several columns of the input matrix have contributed to the output vector. This is equivalent to having routes of unequal delay.

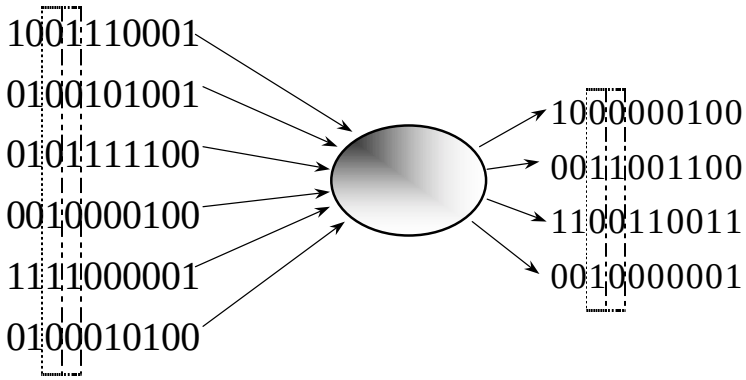

Figure 16. Two elements, which are consecutive in time, of the sample. On the left-hand side is the input matrix, in the middle is the neural system and on the right is output matrix. Two different dotted lines are used to indicate two consecutive pairs of input vectors and output vectors.

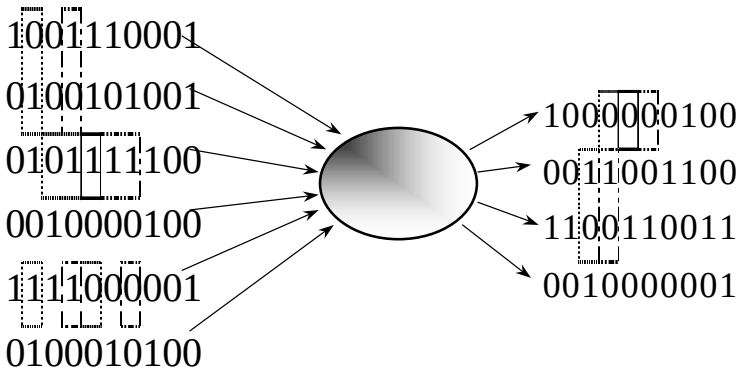

Figure 17. An example on extended sampling. Again, two different dotted lines are used to indicate two consecutive pairs of input vectors and output vectors. In extended sampling, the vectors do not correspond to the columns of the matrix: The columns of the matrix represent points in time and the input and output vectors may comprise data from several different points in time. The consecutive vectors may also overlap one another. Continuous line indicates the overlapping parts.

Part 3
Biological Neural Systems and Information Theory

CHAPTER 11
LITERATURE SURVEY

This chapter gives a summary of the previous work done on and by applying information theory to the analysis of biological neural systems and the codes used by them. This digest is comprised of articles that are considered either key articles in the field or representative samples of some key area within the field.

The articles are categorised into two groups according to their main topic. The first group focuses the methods and methodology of the field. The second group discusses coding in biological neural systems and systems themselves, including synapses, neurons and neural networks. Some of the articles do not fall nicely into one of these categories but rather belong to both of them. In those cases, the choice has been rather arbitrary.

Within each category, the articles have been ordered chronologically according to the year of publication.

11.1 METHODS

ECKHORN AND PÖPEL: RIGOROUS AND EXTENDED APPLICATION OF INFORMATION THEORY TO THE AFFERENT VISUAL SYSTEM OF THE CAT. I. BASIC CONCEPTS, 1974

This could be considered the founding paper of the field. For the first time, it was demonstrated how information theory could be used in the study of biological neural systems.

In motivating the use of information theory, the paper points out that (p. 191) “For information transmission one has to use code. A message is a sequence of symbols <…>” , “Symbols, however, are transmitted by means of a carrier, the signals. The concept of the code denote the (not necessarily one-to-one) relation between signals and symbols”. The final

point in favour of information theory over any and all code assumptions used in previous work is that “In neuronal systems the code is principally unknown” and, thus, calculations based on code assumptions would normally yield estimates of information quantities that are less than the correct values.

Following the motivation, the paper introduces the relevant theoretical and mathematical framework and succeeded by a discussion on the related information-theoretic and practical issues.

Among other things, this 1974 paper points out that the information-theoretic method requires much computing capacity. In the present day, we still find the generally available computing capacity too limited to carry out information-theoretic calculations without simplifying. In this light, it is quite refreshing to read the comment given in the paper on the size of the joint probability matrix: “Usually this matrix is so large that it cannot be covered by the central memory even of a large computer (CDC-Cyber 72, 256 K memory).”

The paper is the first of two. The second paper (Eckhorn and Pöpel, 1975) focuses on experimental results obtained by applying the methods of the first paper. Both papers deal only with cases of single input and single output; a limitation remedied in later papers.

WINDHORST AND SCHULTENS: MEASURES OF TRANSINFORMATION FOR MULTIPLE INPUT/SINGLE OUTPUT NEURONAL SYSTEMS, 1982

This paper gives a theoretical treatment of how the method of Eckhorn and Pöpel (1974), which we discussed above, could be extended to systems with multiple inputs and a single output.

The mathematical treatment used in the paper is interesting. The authors have made an effort to discuss the extension “in analogy to the mathematical description of linear multiple input/single output systems <…>” (p. 57). In particular, they introduce the concept of information-theoretic "cross-talk" between two inputs. In terms of information theory, "cross-talk" between two inputs is actually their mutual information. The name "cross-talk" is borrow from the domain of linear systems, where a analogous concept carries the same name. The authors use the concept of “cross-talk” to derive the information transfer rate between a given input and the output.

Despite the otherwise mathematical nature of the treatment, the paper also describes how the autocorrelation and crosscorrelation properties could be used to alleviate the practical problem of limited computational

resources; the ever-present problem in information-theoretic computations of this kind.

FULLER AND WILLIAMS: A CONTINUOUS INFORMATION THEORETIC APPROACH TO THE ANALYSIS OF CUTANEOUS RECEPTOR NEURONS, 1983

This paper suggests that continuous information-theoretic quantities should be used to estimate the information transfer rate.

The interval distribution of a tonic neuron is known to be very close to Gaussian distribution whose standard deviation is a linear function of the mean rate or, in other words, the interval. The offset and the incline parameters of the function need to be determined experimentally for each neuron (pp. 13–14).

Assuming that a given stimulus always yields the same known mean rate and that the distribution of the stimuli is known, the distribution of the intervals is known. Thus the information transfer rate can be solved analytically for the given stimulus distribution.

The paper then proceeds to carry out those calculations on data of Schreiner et al. (1978), who used a statistical method to estimate the number of stimulus levels discernible by the neuron. The exact numerical results depend on the offset and incline parameters of the given neuron. Also, high and low levels of stimulus are dealt with as separate cases.

In general, it turns out that the statistical method tends to grossly overestimate the information transfer rate due to the nature of the method. The corrected values in terms of distinguishable levels of firing intensity are between 1.27 and 3.34, while the original ones are between 1 and 7 (p. 14, Table 1). The average corrected value is 1.84 and the average correction is +1.91.

OPTICAN, GAWNE, RICHMOND AND JOSEPH: UNBIASED MEASURES OF TRANSMITTED INFORMATION AND CHANNEL CAPACITY FROM MULTIVARIATE NEURONAL DATA, 1991

This paper is mostly about methodology but contains also some experimental results achieved through the suggested methods.

The paper focuses on finding unbiased measures of information transfer rate and channel capacity based on observed data. Three sources of bias are recognised:

- Quantisation
- Noise

- Small sample set size

The information transfer rate is the decrease in the entropy of the input when the output becomes known, or, in other words, the difference between the prior and posterior entropies. If the experiment uses a discrete stimulus, i.e., the input variable has a discrete set of values, only the output needs to be quantised. Consequently, only the posterior entropy is affected by the quantisation. Quantisation tends to decrease the variability of the data and, thus, the entropy. The decrease in the posterior entropy increases the information transfer rate, thus causing bias in the estimate.

To avoid the bias caused by quantisation, the paper suggests using the kernel approach where each data point is replaced by a continuous probability density function, such as a Gaussian pulse (pp. 306–307). This approach is based mostly on previous work of others (see, e.g., Fukunaga 1972).

The method requires that the covariance matrix of the kernel data is computed and this can be done by multiplying a set of one-dimensional kernels together if and only if the kernel is separable. This is not always the case and to avoid this problem, the paper suggests the following: “Basically, the procedure obtains the desired probability function by 1) transforming to a domain where the distribution is separable <…>, 2) generating an appropriately distributed set of points, 3) transforming the set of points back to the original data domain, and 4) building a histogram of the points from the transformed set” (p. 307).

Noise — referring to the randomness of the selection of the output — and small sample set size together tend to increase the information transfer rate and the channel capacity. A tacit assumption is that the empirical distribution of the input vectors is always identical to their probability distribution. If this is the case and the output is a deterministic function of the input, the joint distribution of the inputs and the outputs is identical to their joint probability distribution thus yielding correct estimates. However, if there is noise, i.e. if the output is chosen randomly to some degree, the empirical joint distribution tends to differ from the true joint probability distribution and this tends to bias the estimates upwards by reducing the posterior entropy. This tendency is stronger the smaller the sample size and the greater the noise.

To avoid the bias caused by a small sample set size and noise, the paper suggests that the size of the bias be estimated by creating several sample sets where the input and output pairs of the original set are

broken and new pairs are created at random. This breaks any input-output dependencies. An estimate of the information transfer rate and the channel capacity is then calculated using both the original and the newly created sample sets. The corrected value of the information transfer rate and that of the channel capacity is the result obtained by subtracting a corrective term from the value obtained from the original sample set. The corrective factor is the ratio between the square of the mean value of the results obtained from the new sample sets and the result obtained from the original sample set.

The paper identifies three major properties of the corrected estimate of the information transfer rate:

- It is asymptotically unbiased as the sample set size increases.
- It is approximately zero for all sample set sizes if the true value is zero
- It is biased downwards for small sample set sizes.

In the experimental part of the paper, the methodology is among other things applied to recording from individual neurons in primate visual cortex. The main conclusion were that there seems to be redundancy in the way information is encoded in the neurons, and that the proportion of the available channel capacity that the neuron utilises is increased as the dimensionality of the code is increased. In this context the dimensionality is the number of mutually orthogonal components of the neuron response that are taken into account (pp. 309–310).

The experimental part does not compute the true values of the information transfer rate and the channel capacity because additional assumptions are made about the coding. Without the assumptions, the calculations would have been too memory consuming and long-running to be carried out.

CHEE-ORTS AND OPTICAN: CLUSTER METHOD FOR ANALYSIS OF TRANSMITTED INFORMATION IN MULTIVARIATE NEURONAL DATA, 1992

This paper suggests a method to alleviate the problems caused by the excessive memory and processing-time requirements that plague information-theoretic analyses of biological data.

The method is based on clustering the output data into clusters of equal and nonintersecting volumes. The centroid of the cluster is the arithmetic mean of the vectors that are members within the cluster. On the other hand, a vector is a member of the cluster if it falls within a

"hyperrectangle" (i.e., hypercube) centred on the centroid. A recursive, partially random algorithm is used to create the clusters (p. 31).

It is by no means obvious from the description given by the paper that the algorithm would always stop. Neither it is obvious that the clusters occupy nonintersecting volumes, which is essential in obtaining correct results (p. 31), if and when the algorithm stops. It is rather troubling that the paper omits to discuss these issues[2].

The probability density in each cluster is then simply the number of sample vectors in the cluster divided by the total number of sample vectors. This is combined with what is basically an enhanced version of the methodology given in (Optican et al. 1991, discussed above). Since the result depends on the random choices made in clustering, the procedure is repeated several times to obtain an average.

Finally examples are given using simulated and measured data.

11.2 EXPERIMENTAL WORK

DE RUYTER VAN STEVENINCK AND BIALEK: REAL-TIME PERFORMANCE OF A MOVEMENT-SENSITIVE NEURON IN THE BLOWFLY VISUAL SYSTEM: CODING AND INFORMATION TRANSFER IN SHORT SPIKE SEQUENCES, 1988

This paper applies information theory to the study of neuronal coding. The subject is a movement-sensitive neuron (H1) in the visual system of the blowfly.

Firing sequences of one or two spikes are considered. The stimulus is generated randomly by moving a random figure on a screen. The output of the neuron is measured and interpreted as different combinations of spikes and empty intervals between them, which are then used to estimate the probability distribution.

The paper shows (p. 400) that, under the particular conditions used in the experiment:

- A single spike carries about 0.36 bits of information about the stimulus

[2] Perhaps the reason for the omission is that obviousness, like beauty, lies in the eye of the beholder.

- The length of an empty interval carries 0.17 bits of information about the stimulus
- Two consecutive spikes and the interval between them carries more than 1 bit. This is clearly more than 2 times 0.36 plus 0.17, thus demonstrating "that an important part of the information is carried by *interactions* between spikes."

Considering only individual spikes, the information transfer rate of the neuron is 14 bits per second. The information transfer rate of the intervals between the spikes is much larger, 87 bits per second. (p. 401).

Based on comparing the estimates of the information transfer rate to the known theoretical limit, the paper concludes with a not-so-common view described as "a tentative picture of blowfly visual system as performing optimal, nearly noiseless computations, with much of the low noise level ascribed not to collective interactions among neurons but rather to the intrinsic precision of the individual cells" (p. 412).

TOVEE, ROLLS AND TREVES: SHORT SAMPLE PERIODS ARE SUFFICIENT TO EXTRACT A SIGNIFICANT PROPORTION OF THE TOTAL INFORMATION ENCODED IN A NEURAL SPIKE TRAIN, 1992

In this paper, spike trains were recorded from single neurons in the inferior temporal cortex and wall of the superior temporal sulcus of macaques monkeys. The monkey was given a visual fixation task during which the recordings were made.

Figure 18 shows the amount of information that the firing rate carried about the stimulus during a period of 20 and 50ms relative to amount of information during a period of 400 ms. Timing was started at the beginning of the elicited spike train. This is not necessarily the total amount of information.

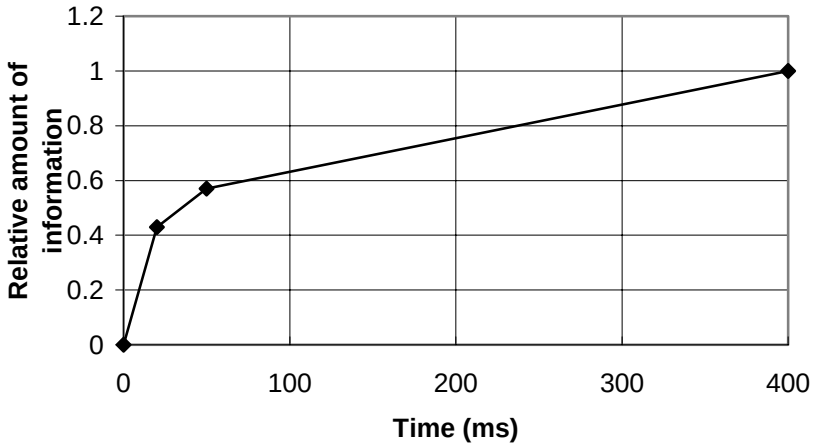

Figure 18. The relative amount of transferred information as a function of time since the beginning the elicited spike train as compared to the total amount of information transferred during the first 400 ms after the beginning of the spike train.

The paper concludes that these results are consistent with the hypothesis that a significant proportion of the total available information is available within short periods of the spike trains of neurons in the visual system and with the hypothesis that "information from each cortical area is extracted from a short estimate of neuronal firing, rather than a long estimate which allows temporal encoding to be analyzed."

The paper presents also results of a principal component analysis on the same spike train: The first principal component was found related to the firing rate and the second one usually to the onset latency of firing.

RIEKE, WARLAND AND BIALEK: CODING EFFICIENCY AND INFORMATION RATES IN SENSORY NEURONS, 1993

This article focuses not only on the information transfer rate of sensory spike trains but also on how efficient their coding is, i.e. how close to the theoretical limit imposed by timing precision the information transfer rate of sensory spike trains come.

The analysed data was obtained by experiments where the input signals approximate white Gaussian noise. Two mechanosensor systems were studied. In the first case, the input was a stimulus to the vibratory receptors of the bullfrog inner ear and, in the second case, to the filiform

hairs of the cricket. The variations in the input signal are on a considerably shorter time scale than the mean interspike interval.

As the interspike intervals were determined to approximately 3 bits of accuracy, the information transfer rate was 155 ± 3 bits per second in the bullfrog and 294 ± 6 bits per second in the cricket, which correspond to 2.6 ± 0.05 bits per spike and 3.2 ± 0.07 bits per spike, respectively (p. 153).

The paper defines coding efficiency as a function of the timing precision, namely, as the information transfer rate divided by the entropy — i.e., autoinformation — of the spike trains, both measured with the given timing precision. Both systems have coding efficiencies between 50% and 60% (p. 155), yet the coding efficiency depends on the timing precision in a noticeable way (Fig. 3 on p. 154).

It should be noted that the paper makes some simplifying assumptions about the coding in calculating the information transfer rate and another set of assumptions in calculating the entropy of the spike trains. Both sets of assumptions tend to cause the coding efficiency to be underestimated.

In the conclusions the authors state: “The information rates we measure are quite high: if single cells are carrying information at rates of (100 ÷ 300) bits per second, an array of cells may be able to code a significant fraction of the information present at the receptor level, transmitting a detailed image of the sensory environment to the central nervous system. The fact that so much information is transmitted suggests that it may not be necessary to limit the flexibility of neural computation by detecting only a limited array of «features» in a natural signal while discarding large amounts of «irrelevant» information [13] <Lettvin et al. 1959>. In fact, the coding process may not limit the information throughput of these systems.” The last sentence is a particularly striking statement.

KJAER, HERTZ AND RICHMOND: DECODING CORTICAL NEURONAL SIGNALS: NETWORK MODELS, INFORMATION ESTIMATION AND SPATIAL TUNING, 1994

In this paper the information transfer rate in the primary visual cortex of awake monkeys is studied. The data used was recorded in the primary visual cortex of two rhesus monkeys while different patterns were presented to them in pseudorandom order on a video monitor (pp. 111–112).

Instead of computing the information transfer rate directly from the measured data, different models — such as artificial neural networks and

Parzen-windows and Gaussian mixture models — are fitted to the data. The cost function, which the fitting attempted to maximise, was the sum of the logarithm of probability of the model producing the given stimulus–response pair taken over all the stimulus–response pairs in the training set (p. 113).

As inputs, the models had the mean firing rate or a varying number of principal components. Once the fitting was done, the models were compared using test sets and the best model was identified. It turned out to be a feed-forward neural network with one hidden layer regardless of the type of inputs that were used.

It was then considered whether the model was overfitted, which would bias the information-theoretic estimates upwards. The method used was similar to that of Optican et al. 1991, discussed above. "The result of this calculation for our network was a vanishingly small amount of spurious information <…>. Thus, there was essentially no overfitting in our method" (p. 122).

The model was then used to estimate the probability distribution needed in the calculations. In particular, calculations were made to study the role of temporal coding and the effect of spatial tuning.

The paper contains a rather lengthy discussion where several conclusions are drawn. Here are some of the most important ones:

- As so often in information-theoretic computations, the adequacy of computational resources is of interest: "The computations we have done to find the optimal model are well within the capacity of current workstations" (p. 134). The authors estimate that similar future studies require even less work by relying on some of their results (pp. 134–135).
- Several potential sources of error are identified in the method. They are described in terms of underfitting and overfitting. Not only are there the sources discussed in Optican et al. (1991, discussed above) but also those created by the choice of model and preprocessing. The authors chose to use feed-forward networks as their model because there were no clear grounds for the choice and "all we can say is that it is the best one we have found so far" (p. 135). In preprocessing, "the truncation to a small number of principal components" (p. 135) is an example of assumptions about the code that may be sources of error. Such assumptions are typical of most studies applying information theory to biological neural systems and

often attributable to limitations in present-day computing resources.

- The paper discusses the amount of information carried in temporal coding and estimates that "about a third of the total transmitted information is carried in the temporal variation of the response" (p. 135). The higher estimate given earlier by Richmond and Optican (1990) is attributed to shortcomings in the methodologies that had been used.

DE RUYTER VAN STEVENINCK AND LAUGHLIN: THE RATE OF INFORMATION TRANSFER AT GRADED-POTENTIAL SYNAPSES, 1996

This paper extends the use of information theory to graded-potential synapses. It is also the only paper among the ones we have discussed that uses other than the most basic theorems in information theory.

The paper studies the information transfer rate of the photoreceptors of blowfly through chemical synapses to large monopolar cells (LMC). The stimuli are in the form of pseudorandom contrast sequences. It is assumed, and shown, that the data can be approximated by an easily analysable continuous function. In this case, the signal and the noise are assumed to be Gaussian.

Under these assumptions, the information transfer rate R is given by Shannon's (1949) formula as

97)
$$R = \int_0^\infty \log\left(1 + \frac{S(f)}{N(f)}\right) df$$

where $N(f)$ is the spectral density of the noise and $S(f)$ is that of the signal. Since the neurons used in the measurements were unidentical, the measured noise spectral densities were normalised to produce a common measure (p. 643). Also, the channel capacity was determined.

As a result, the photoreceptors seem to have channel capacity of 1000 bits per second whereas LMCs, which receive signals from six photoreceptors each, have channel capacity of 1650 bits per seconds thus demonstrating considerable redundancy. Within each photoreceptor there are approximately 200 separate active zones. Based on mostly structural evidence, the paper computes that each active zone has information transfer rate of some 55 bits per second or higher (p. 644).

The paper concludes that “sensory neurons and chemical synapses are remarkably effective. The information capacities of photoreceptors and LMCs are respectively 3 and 5 times higher than the maximum rates reported by spiking neurons in cricket cercal afferents <…>” (p. 645). This is seen as a sign of the suitability of graded synapses to short-distance, fast and accurate communication and, thus, to sensory systems. It is also pointed out that in conversion to spike trains the information in graded synaptic inputs is easily lost.

Since the information transfer rate of, say, the active zones must be explainable by the physical description of the system, the paper points out: “Thus measures of information capacity provide benchmarks for understanding the performance and design of neural systems at the network, cellular and molecular levels” (p. 645).

CHAPTER 12
SIMULATION METHOD.

This chapter proposes a new, simulation-based methodology for information-theoretic study of biological neurons and neural networks. The chapter also reports a case study evaluation of the methodology based on a particular software implementation and a particular study where the methodology is applied. The implementation is described in this chapter while the study is described in Chapter 13.

The methodology is based on a software simulator to obtain data for information-theoretic computations. It is particularly suited for situations where true biological data is hard or impossible to obtain in required quantities. Such situations may occur, for instance, in studies focusing on how the information-theoretic properties of the neural system vary with the changes in the physical properties of the system. Chapter 13, which studies the connection between the information-theoretic properties and the topology of a neuron, is an example of such a study.

It should be noted that a key to any successful study using simulations is obtaining a valid model of the simulated neural system. A further discussion on this topic can be found in Section 12.3.

12.1 ANALYSIS OF PROBLEM DOMAIN

The problem domain for which the methodology was developed was analysed using OMT++ approach (see, e.g., Jaaksi 1997). Figure 19 summarises the results of the analysis using the OMT notation of Rumbaugh et al. (1991). OMT is used here because it is a well established notation in software industry and the aim was to develop a software solution. When used in analysis only (and not in software design) OMT gives an formal description of the analysed domain without references to the implementation details.

The problem domain has the following relevant components:

The goal is to produce a *study* (see Figure 19). A study always defines at least one set of problem parameters. If there are more than one set of problem parameters, the sets define variations within the neural system and the stimulus probability distribution. For instance, if the study were concerned with the topology of the neural system, the sets of problem parameters would be used to vary the topology of the otherwise predetermined neural system. If the stimulus probability distribution — the probability distribution of the inputs, that is — varies along with the topology, the set of problem parameters describes this variation as well. The set of problem parameters could also be an empty set if there is no variation to be considered. In this case, the neural system and the stimulus probability distribution are fixed in the study.

A study, once completed, includes all the produced data points. A data point is an ordered pair comprising a set of problem parameters and at least one set of information-theoretic properties, such as entropies or information transfer rates, obtained via the experimental set-up that uses the particular set of problem parameters. Some studies may require that, say, deviation of the information-theoretic properties are studied, in which case, several sets may result from the study.

A set of information-theoretic properties is always the result of some computation, which is based on a stimulus–response distribution. A stimulus–response distribution is a collection of stimulus–response pairs. It is not a probability distribution, but the distribution of the observed data. One set of problem parameters may yield one or more such stimulus response distributions depending on how the study is carried out. For instance, in Chapter 13 we discuss a study, where several stimulus–response distributions are produced for each set of problem parameters in order to estimate the accuracy of the resulting set of information-theoretic properties.

Each stimulus–response pair is an ordered pair comprising a stimulus, which is randomly drawn according to the stimulus probability distribution, and a response. Since a neural system may in itself contain random elements, such as sources of noise, each stimulus is capable of invoking more than one possible response in the neural system. Each time the response of the neural system to a stimulus is measured (or, rather, simulated), values must be drawn for the random elements. This produced a neural system invocation, which is like the neural system, but without any randomness. This is analogous of drawing a stimulus according to the stimulus probability distribution, in which case the stimulus could be seen as an “invocation” of the stimulus probability

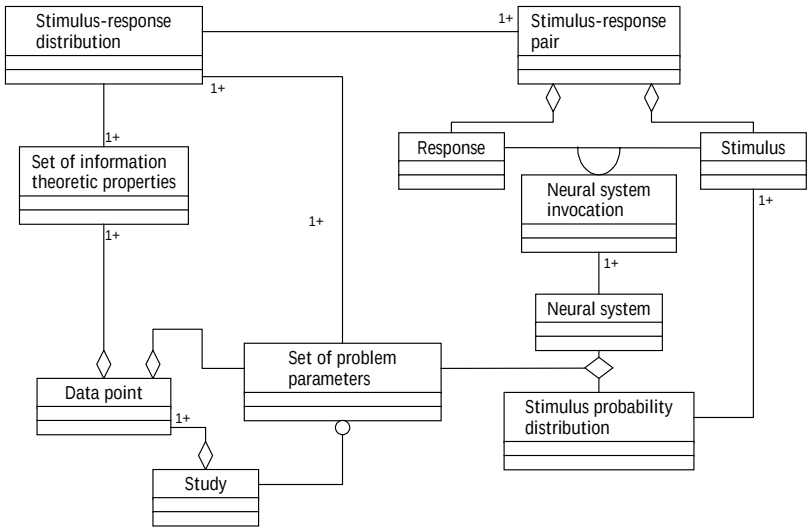

Figure 19. The analysis object model of the problem domain. The object model is explained in more detail in the text. The figure follows the OMT++

distribution. The response in a response-stimulus pair is whatever response the neural system invocation produces given the stimulus in the pair.

The responses and the stimuli may be in the form of spike trains but quite as well they may be given in terms of some other quantity such as mean firing rate, frequency, temperature or muscle activity, if considered appropriate.

12.2 IMPLEMENTATION DESIGN

The following approach was used to implement what was described in the analysis of the problem domain:

A program, which we shall call the *simulator*, was used to implement the neural system invocations. In other words, the simulator was used to compute response to each given stimulus.

The sets of problem parameters were enumerated by assigning each one of them a name, an integer. A program called the *input generator* was used to generate a description of the neural system and the stimulus probability distribution. The input generator was simply a piece of program that accepted an integer as its input and produced a file, which was subsequently read by the simulator. One might either say that the integers are the (sets of) problem parameters or that the integers were handles to or names of the actual sets of problem parameters which themselves were a part of the code within the input generator. It does not matter which point of view is taken.

The stimulus–response pairs were collected in files, which formed stimulus–response distributions. The files were read by the *analyser* program, which then computed the sets of information-theoretic properties.

The SWIM simulator, which has been developed at the department of Numerical Analysis and Computing Science at the Royal Institute of Technology in Stockholm, was chosen for the study. The input generator and the analyser were coded using C++ along with a set of utilities and scripts used to connect the pieces of software together. “SWIM is a simulation program for numerical simulation of networks of biologically realistic model neurons” (Ekeberg et al. 1990, page without a number in the chapter titled "Abstract")

12.3 EVALUATION

SIMULATOR VERSUS REALITY

Whenever simulation is used to gain experimental results, one should question the reliability of the results: Does the simulation capture the features of reality essential to the experiment?

The behaviour of a biological neural system is very complex, and modern simulators, such a SWIM, are still far from describing the behaviour in all its details. However, the kinds of experiments that we are interested in focus on the electric aspects of the behaviour and that is where our knowledge of biological neurons is strong and is based on a variety of experiments.

In this light, it seemed reasonable to assume that SWIM is capable of capturing the low-order statistical nature of the spike trains of a single biological neuron and, consequently, the related information-theoretic properties. Of course, the final judgement on this could only come from a set of experiments comparing these aspects of simulated and biological neurons.

In contrast, it is fair to say that the higher-order statistics are not captured by SWIM, since there has been no or very few studies on how to include them in a simulation. Therefore, the information-theoretic analysis was designed to ignore the higher-order aspects, as explained shortly.

INPUT FORMULATION AND OUTPUT ANALYSIS

Since it was decided that the higher-order statistics are represented unreliably in SWIM, the analysis of the output was limited to mean spiking interval. In other words, even though the output of the SWIM simulator describes the entire spiking sequence, the response in each stimulus–response pair was not the spiking sequence per se, but a single number: the mean spiking interval. Similarly, the stimulus was recorded in terms of the mean spiking interval even though each neural-system invocation used a complete spiking sequence as the input to the neuron.

One second of activity was simulated. Stationary spiking trains this long rarely occur in the nature as the nervous system tends to operate on timescales of 10-100 milliseconds. One second simulation was seen a justifiable method to increase accuracy of the analysis, however, because the use of mean spiking interval already rules out the possibility of taking the non-stationary phenomena, such as the commonly occurring adaptation to the stimulus, into consideration in the analysis.

In principle, entire spiking sequences could be used both as the stimuli and as the responses within the framework of this methodology if one trusts that the simulation is accurate enough to produce correct results. However, there are some practical problems:

- Computational complexity, memory requirements and the number of required samples increase rapidly as the order of statistics of the spiking sequences is increased reaching their peak when the entire spiking sequence is taken into account. This problem has been recognised repeatedly in the literature and no good solutions exist. Taking into account more than

just a few orders would constitute a considerable strain to the computing resources available in a typical laboratory.

- There is not much data available on the higher-order statistics of most types of biological neurons. Besides making it difficult to verify the simulations this introduces the problem of how do we know that we are using a biologically correct stimulus probability distribution. This problem can be solved, of course, by collecting such data.
- If the simulator is SWIM, there are some additional problems in describing the spiking sequence in full. We shall take a closer look at this problem shortly.

CHOOSING THE PROBLEM PARAMETERS

A problem in using a simulator is verifying that each neural system has biologically realistic properties. For instance, if we want to study the effect of the number of branching points in the dendritic tree of a neuron, the set of problem parameters would be the number of the branching points (and, perhaps, some other parameters defining, say, the location of those points). If the number of branching points is allowed to vary between 1 and 10, we must define ten different and biologically realistic types of neural systems. If there is another parameter with another 10 possible values, we have 100 different types of neural systems.

The biological plausibility of the behaviour of the simulated neural system needs to be verified every time a new type of system is introduced. This requires a great deal of work and expertise.

In special cases, this process can be automated, however. In studies on the topology of the dendritic tree of a neuron it is enough to find a single valid neural system having one of the required topologies and the other neural systems can be obtained through systematic variation.

The mathematical model used by SWIM (and similar simulators) tends to be insensitive to variations of the topology of the dendritic tree if the tree has an equivalent cylinder. The exact definition of an equivalent cylinder was given by Rall (1977, p. 48). In simplified terms an equivalent cylinder exists for a dendritic tree if there could be another dendritic tree comprising a single cylinder shaped branch with equivalent key electrical properties. It is also required that the equivalent cylinder must remain

constant under the variations of the topology.[3] This is due to the simulated mathematical model (see, e.g., Ekeberg 1990). Thus, in Chapter 13 we use the neuron used by Fransén and Lansner (1998) as our starting point and modify the topology of its basal dendritic tree.

To add a branch to a leaf segment of the dendritic tree is the same as replacing the segment with two segments and their common branching point. To maintain the equivalent cylinder, the sum of the $3/2$ powers of the diameters of the new segments need to equal to the $3/2$ power of the original segment (Rall 1977, p. 48) or

98)
$$d_{original}^{3/2} = d_1^{3/2} + d_2^{3/2}$$

The same principle can be used to combine two leaf branches into one. The distance of the branches may vary as long as all leaf branches have the same boundary condition and their electrotonic distances are equal. The electrotonic distance L is the ratio between the actual length ℓ of the branch and the length constant λ of the core conductor of the branch. (For specifics on these concepts, see Rall, pp. 46–50).

Using the same notation as Rall (1977), the electrotonic distance is defined as

99)
$$L = \frac{\ell}{\lambda} = \frac{\ell\sqrt{r_i + r_e}}{\sqrt{r_m}} = \frac{\ell\sqrt{dR_i}}{\sqrt{4R_m}}$$

where r_i is core resistance in per unit length (Ωcm^{-1}), r_e is resistance per unit length of a thin external cylindrical layer of a given thickness (Ωcm^{-1}) and r_m is resistance across a unit length of passive membrane cylinder (Ωcm). For more details on the definitions, see Rall's paper (1977).

The last form, which is commonly used, assumes a cylindrical core conductor and large external volume. The latter assumption makes r_e zero. Here R_i is volume resistance of the intracellular medium, R_m is the

[3] Erik Fransén, private communication 1996.

resistance across a unit area of passive membrane and d is the diameter and ℓ the length of the segment.

For the SWIM simulator, one needs to compute for each given segment:

- The core conductance $\frac{1}{\ell r_i}$
- The membrane conductance $\frac{\ell}{r_m}$
- The membrane capacitance ℓc_m

These quantities can be defined using R_i, R_m and C_m, which is the capacitance per unit area of the membrane, as:

100) $$\frac{1}{\ell r_i} = \frac{\pi d^2}{4\ell R_i}$$

101) $$\frac{\ell}{r_m} = \frac{\ell \pi d}{R_m}$$

102) $$\ell c_m = \ell C_m \pi d$$

Thus, starting with a neuron for which we have defined these three values in each segment – or compartment, to use the terminology of SWIM – we can combine segments with common branching point or split a segment into two such segments. Instead of branching, we can also divide a segment into two adjacent parts or combine two such parts together. In each case, the original and the new segments differ only in their diameter d or their length ℓ or both. We get the new values, of the core conductance, the membrane conductance and the membrane capacitance as

103) $$\frac{1}{\ell' r_i'} = \frac{\pi d'^2}{4\ell' R_i} = \frac{d'^2 \ell}{d^2 \ell'} \frac{\pi d^2}{4\ell R_i} = \frac{d'^2 \ell}{d^2 \ell'} \frac{1}{\ell r_i}$$

104) $$\frac{\ell'}{r_m'} = \frac{\ell' \pi d'}{R_m} = \frac{\ell' d'}{\ell d} \frac{\ell \pi d}{R_m} = \frac{\ell' d'}{\ell d} \frac{\ell}{r_m}$$

105) $$\ell' c_m' = \ell' C_m \pi d' = \frac{\ell' d'}{\ell d} \ell C_m \pi d = \frac{\ell' d'}{\ell d} \ell c_m$$

where ´ is used to indicate a changed value. Obviously, the new value of each quantity can be obtained by simple multiplication under any change of topology that keeps the equivalent cylinder unchanged.

SIMULATION AND ANALYSIS PARAMETERS

In this kind of setup, one has to decide on one simulation parameter and at least one analysis parameter.

While carrying out the simulations one must decide when to stop or, in other words, how many stimulus–response pairs need to be collected. The decision should be based on reliability considerations. Firstly, there may be some nonbiased statistical fluctuation due to the limited number of samples. To estimate the amount of this error, five stimulus–response distributions were created for each set of problem parameters yielding five sets of information-theoretic properties.

Secondly, the limited number of samples may cause a systematic error in estimating the entropies — typically a downwards bias — whose difference, the information transfer rate, may therefore contain either an upwards or a downwards bias. Assuming no prior knowledge about the stimulus–response distribution it is impossible to know with certainty that the number of samples is adequate. One might always suspect that by increasing the number of samples and using higher resolution in analysis one might find new structure in the stimulus–response distribution affecting the entropies, the information transfer rate and other potential members of the set of information-theoretic properties.

There are, however, heuristic solutions. We shall discuss one of them shortly.

While running the analyser to produce the set of information-theoretic properties one has to decide on how the quantisation is done. Which variables need to be quantised depends on what kind of properties we want to have in the set of information-theoretic properties and how exactly we carry out the calculations. In the simulations, the only member of the set was the information transfer rate, or mutual information, and it was computed by subtracting the observed mutual entropy of the stimuli and the responses from the sum of their individual observed entropies.

A rather straightforward method was used where the range of stimuli and the range of responses were divided into the same predetermined number of bins. Thus the only analysis parameter to be determined was this number. This approach was chosen mostly to keep the determination as simple as possible.

The heuristic method that was used to determine the number of samples and the number of bins is as follows. Before the actual simulations several test runs were made using different sets of problem parameters. In each run, one million samples were generated. The analysis was done repeatedly using a varying number of samples from the set of a million and a varying number of bins. Figure 20 shows an example of the results gained from one such test run.

If the number of samples is too small, the estimate of the information rate grows somewhat linearly as the function of the number of bins after a certain point (see Figure 20) . As the number of samples is increased, the estimate tends to grow asymptotically towards a constant value. This suggests that the linear growth is just an artefact caused by the small number of samples per bin. Therefore it seems justifiable to choose a number of samples that yields the asymptotic behaviour and a number of bins that yields an estimate almost equal to the asymptote.

It was not seen necessary to obtain the best possible estimate of the absolute value of the information transfer rate for each set of problem parameters since the absolute values obtainable from a simulation would at best be only suggestive of the true values found in nature. Instead it was seen important to be able to estimate reliably and consistently the relative changes in the information transfer rates caused by changes in the set of problem parameters. Thus, it was seen sufficient to make sure that the error in estimating the asymptote was reasonably and consistently small. No special effort was put into producing the best possible estimate.

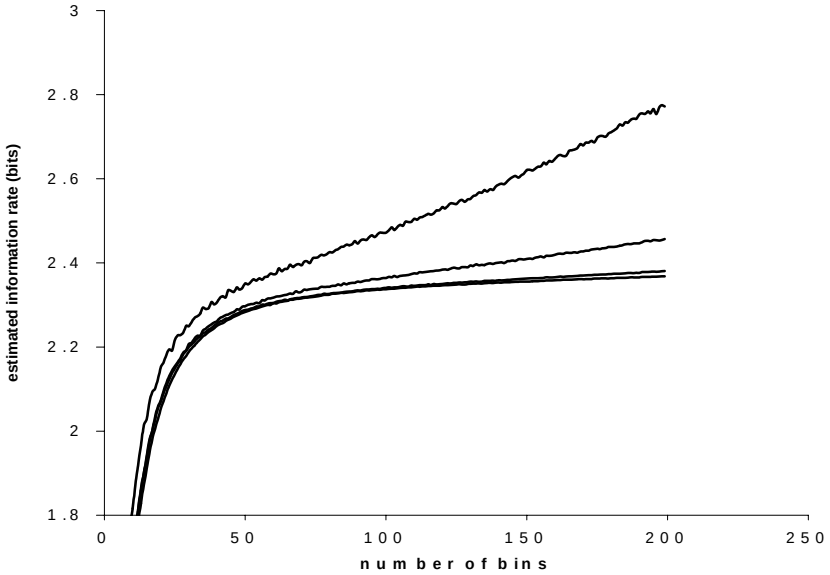

Figure 20. The estimated information rate as a function of the number of bins, i.e., categories used in the analysis. Several graphs are shown, each corresponding to a different sample set size. The lines correspond to 12 500, 50 000, 200 000 and 400 000 samples respectively from the top to the bottom.

To make the set of information-theoretic properties comparable over the various sets of problem parameters, a single value for the number of samples and the number of bins had to be found. Thus, the values had to be chosen large enough to give reliable results even in the worst case. On the other hand, the number of samples could not be chosen arbitrarily large since the simulation time was the bottleneck in the process: During the tests runs, it became obvious that the simulations would take months to run even though up to 26 computers were available for the simulations. On the other hand, the time needed for the analysis was negligible.

After the actual simulations, the previously described method was used to verify that the number of samples and the number of bins were sufficient: First, the nonbiased statistical fluctuation in the estimates of the information transfer rate was estimated by comparing the results obtained from the five *stimulation*-response *distributions*. Then the samples in those five distributions were combined to a single distribution and the previously described heuristic method was applied. In each case, it turned out that original simulation and analysis parameters gave acceptable results taking into account that the focus of the study was on the relative changes and not in the absolute values.

SWIM RELATED ISSUES

One of the problems in using the SWIM simulator in this kind of work was describing the stimulus probability distribution in a way that SWIM could use.

In SWIM version 1.5 there is no general way to define a stimulus probability distribution. One can only inject current into a compartment, which then causes spiking to occur, or one can introduce random noise into the compartment.

The random noise could be seen as a special synapse opening and closing. One can control the conductance and the voltage of the synapse as well as the expected (reciprocal of the) spike interval, which is Poisson distributed, and the speed at which each spike decays. In comparison to injecting a current, this approach allows a more direct — yet very limited — control over the stimulus probability distribution. There is no full control over the timing of individual spikes. This was reflected in the choice of the stimulus probability distribution.

In these studies, all the stimulus probability distributions were chosen to be identical. The Poisson-distributed random noise was selected as the starting point for creating a reasonably constant distribution for the firing rate or the stimulus probability distribution. SWIM does not comprise a separate mechanism for accepting a description of the stimulus probability distribution per se as input, but the necessary data is included in the description of the neural system.

The final stimulus probability distribution was the result of two steps. First, the *input generator* created the description of a neural system, in which Poisson-distributed noise was injected. This emulated the behaviour of the neuron for a given value of the source variable. Included

in the description was the expected reciprocal of the spike interval drawn at random from a uniform distribution between 0 and 300 Hz. This emulated the distribution of a random source variable. The neural-system invocation — that is, running the SWIM — then chose the timing of the individual spikes according to the resulting Poisson distribution. Due to this approach, only one neural-system invocation was allowed per neural system.

Since such a two-step approach was used, it is important to state that the aim was to study the information transfer rate between the actual presynaptic spiking frequency created in the neural-system invocation and the output of the neuron; not between the expected reciprocal of the spike interval, which was given in the description of the neural system, and the output of the neuron.

The SWIM simulator provides no direct mechanism for observing presynaptic spiking frequency related to the special synapses which are used to introduce random noise. Therefore, the noise was used to stimulate a single compartment input neuron. The parameters of that input neuron were chosen so that it would reproduce the spiking of the special synapse as faithfully as possible. The input neuron was then connected to the neuron under study via a standard neuron. By observing the spiking frequency of the input neuron and the spiking frequency of the neuron under study, stimulus–response pairs could be collected.

Both the input neuron and the synapse connecting it to the neuron under study are described in the Appendix.

To realise the stimulus probability distribution, the SWIM simulator was recoded so that the random seed was stored in a file between the invocations of the process. This way it was possible to run simulations in a piecewise fashion, which would otherwise cause the random seed to be reinitialised to the same value every time SWIM was started. Also, it was possible to regenerate any sequence of simulations simply by storing the original random seed of the sequence. The input generator used a similar mechanism for the same purpose.

In both cases, the requirements for the quality of the random number generator are not very demanding. It is sufficient that the generated random number sequence has the desired distribution when the sequence is long, since the order in which the neural-system invocations take place does not affect the stimulus–response distribution. Therefore, the drand48 random number generator, which is one of the default generators used by SWIM, was considered good enough for the input generator as well.

More information on the drand48 generator can be found for instance in (Knuth 1981 pp. 9-25; Roberts 1982).

PERFORMANCE ISSUES

In this new methodology, it would first seem that the performance bottleneck is the *analyser*. This may actually be the case if the number of bins used in the analysis is large, e.g., if multidimensional analysis is done. However, in the simple cases that were studied, the performance bottleneck turned out to be the SWIM simulator. The analysis of 200 000 one second samples using 200 bins in both dimensions took typically a minute or two of real time. The creation of those 200 000 samples would take a couple of days.

SWIM allows one to generate the samples either sequentially, which would yield one sample per simulation run, or in parallel, which would generate all the samples in one run. The performance of SWIM was tested by measuring the CPU time needed to simulate varying numbers of samples both in the sequential and the parallel case. The sequential approach turned out to be the more effective way of using SWIM: The time needed for the simulation in the parallel approach grew about 1.6 times faster than the number simulated neurons when parallel simulation was used. In contrast, if the samples were created each in its own simulation, the simulation time grew at the same rate as the number of simulated neurons. The latter result seems obvious, since each sample was created in an independent simulation. Figure 21 illustrates these results.

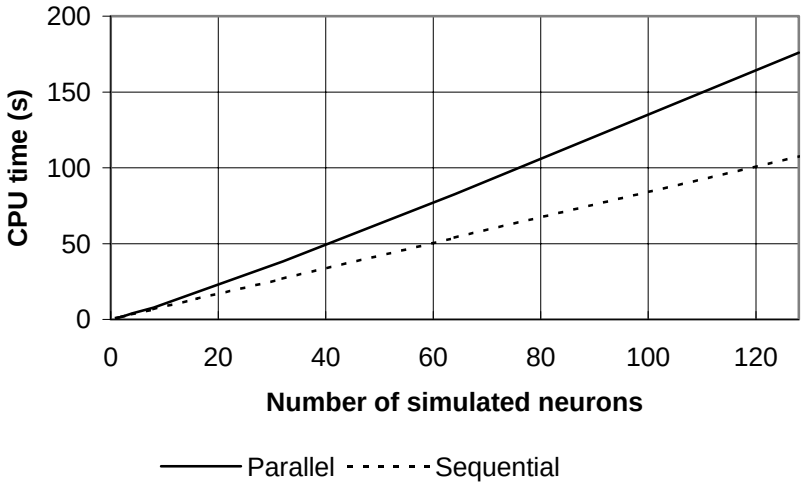

Figure 21. The CPU time of parallel and sequential simulations as a function of the number of simulated neurons

ADDING OTHER METHODS

Our discussion has focused on the basic ideas of the new methodology. Many of the ideas that we discussed in Chapter 11 could, however, be incorporated in the same framework.

The case study in Chapter 13 focuses also on the basic ideas. This choice was made so that the evaluation of the basic methodology would not be clouded by the more advanced variations.

CHAPTER 13
DENDRITIC TOPOLOGY

This chapter discusses two simulations made with the proposed simulation methodology. The simulations were done with two goals in mind. First, to evaluate the new methodology and its implementation in the form of a case study. The results of the evaluation were discussed in Chapter 12.3. Second, to gain some insight into the relationship between the topology of a dendritic tree and the information-theoretic properties of a neuron. Those results are discussed here.

Topology was chosen as the topic of this study, because the variations in dendritic topology are hard to study using live neurons and because topology is the key to understanding connectivity in biological neural systems (see e.g. Maass 1998 for further discussion).

13.1 THE NEURON

Fransén and Lansner (1998) simulated a pyramidal neuron and the same model is used in this study. Figure 22 shows the compartments of the original version of the neuron.

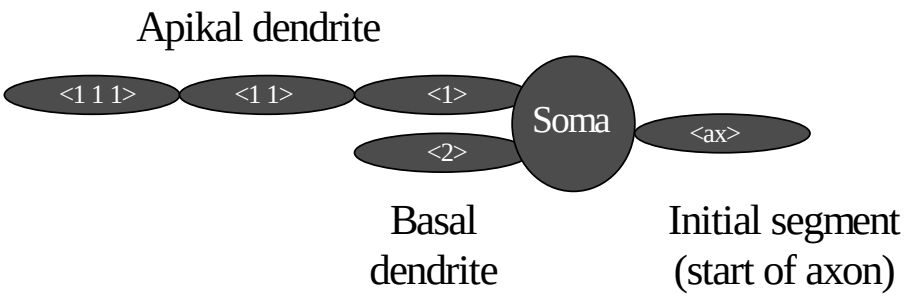

Figure 22. The compartments of the original version of the neuron.

The two simulations focused on the topology of the basal dendrite. The first simulation studied the effect of the location of the synapse along the dendrite with no branches. The second simulation studied a situation where the dendrite has one branching point and a single synapse on one of the branches, and where the relative sizes or, more precisely, the diameters of the branches are varied. In both simulations, the information transfer rate of the cell was calculated as the mutual information between the firing rate of presynaptic cell and the firing rate of the pyramidal cell.

In each simulation, one second of virtual time[4] passed. The firing rate of each cell was computed based on the average distances of consecutive spikes it generated during that time.

In the first simulation, the basal dendrite was divided into 16 compartments as shown in Figure 23. The compartment structure of the neuron used in a simulation where the location of the input synapse was varied along the basal dendrite. The division was done in the manner that was described in Chapter 12.

The compartments were assigned numbers 1 through 16 with 1 next to the soma. The soma was assigned number 0. The set of problem parameters comprised a single variable — a number between 0 and 16, inclusive — indicating to which compartment the synapse between the pyramidal neuron and the input neuron was attached.

[4] To create single stimulus–response distribution — i.e., to run 200,000 simulations — took 8–20 days depending on the computer being used and on what else the computer was being used at the time. That is about 700,000 to 1,700,000 seconds.

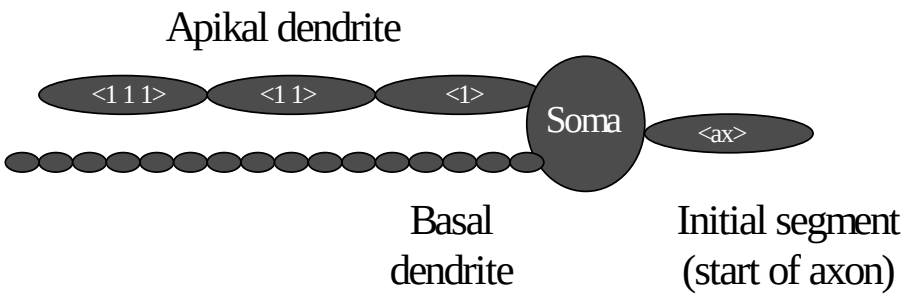

Figure 23. The compartment structure of the neuron used in a simulation where the location of the input synapse was varied along the basal dendrite.

Each resulting data point comprised the number of the compartment and the information transfer rate. For each compartment, 5 data points were generated and each data point was based on stimulus–response distribution of 200 000 stimulus–response pairs.

Figure 24 shows the information transfer rate as a function of the compartment. For each compartment, the minimum, the mean and the maximum of the five data points is shown. Also, a trend line based on linear regression is shown for all three.

In Figure 24 we can see that the information transfer rate is almost constant with a small tendency to decrease as the distance from the soma increases. This result is well in line with our general understanding of the nature of spiking and of the spike propagation in dendrites with no branching and therefore quite expected and, therefore, speaks for the validity of this new simulation method rather than against it.

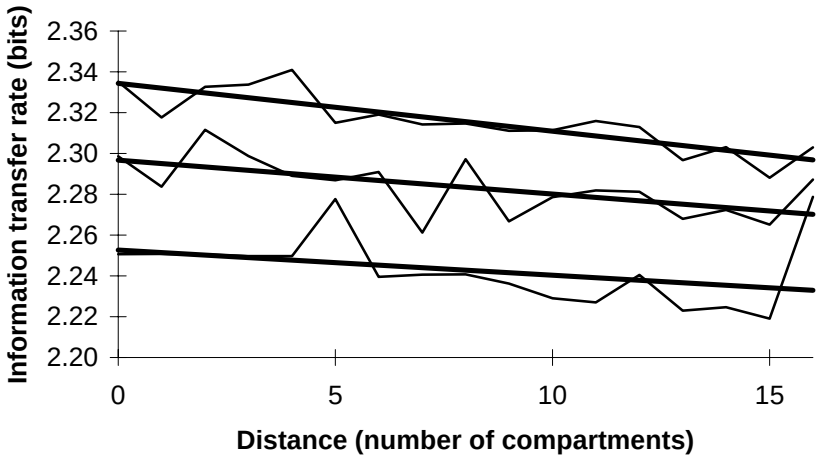

Figure 24. The information transfer rate as a function of the distance between the input synapse and the soma. The three graphs give the minimum, the average and the maximum of five data points *that were generated for each corresponding distance.*

13.2 INFORMATION TRANSFER RATE AND RELATIVE BRANCH DIAMETERS

In this section we shall take a look of a more complicated situation: The basal dendrite is split into two branches and the synapse is always located at the tip of one of the branches. The topology of the basal dendrite is derived from the topology used by Fransén and Lansner (1998), by dividing the dendrite first into 16 compartments and then, counting outwards from the soma, creating a branching point between the ninth and the tenth compartments. All this is done while maintaining the equivalent cylinder as we discussed in section 12.3. The compartment structure is shown in Figure 25.

The above said still leaves one parameter undefined. In creating the branching point, the equivalent cylinder can be maintained regardless of the relative diameters of the two branches as long as Equation 98 is

satisfied. The equation requires that if we compare the 3/2 powers of the branch diameters, the power of the original dendrite equals the sum of the powers of the branches replacing that dendrite.

Thus, in this study, the set of problem parameters comprises the values of a single variable: the 3/2 power of the diameter of the branch where the synapse is located divided by the sum of the 3/2 powers of the diameters of the two branches.

The values of the 3/2 power that were chosen to be studied were 1/4, 1/8, 1/16, 1/32 and 1/64 as well as the corresponding values for the other branch: 3/4, 7/8, 15/16, 31/32 and 63/64. The point of adding these corresponding values to the set of problem parameters is that, effectively, one first chooses the diameters — say, the 3/2 powers being 1/32 and 31/32 — and then studies two cases: the synapse being connected to the thinner (1/32) branch and the synapse being connected to the thicker branch (31/32). Also, the value 1/2 was added to the set of problem parameters. This is, of course, a symmetric case where it does not matter to which branch the synapse connects.

Figure 26 shows how the actual diameter behaves as the function of its 3/2 power. It also shows the behaviour of the sum of the actual diameters of the two branches. The sum of the 3/2 powers of the two branches is shown for ease of comparison; it always equals 1 as required by Equation 98.

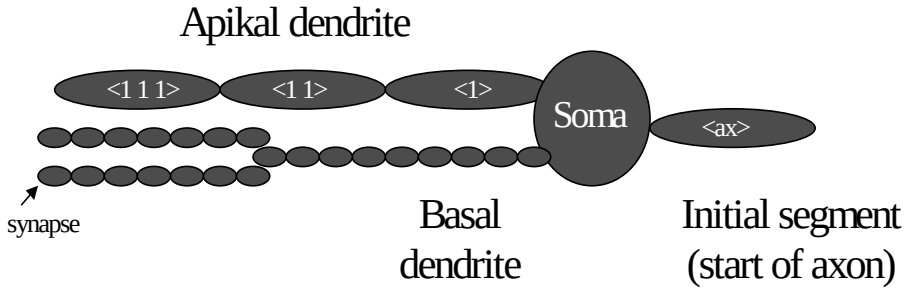

Figure 25. The compartment structure used in the simulations where the information transfer rate was studied as a function of the 3/2 powers of the branch diameters.

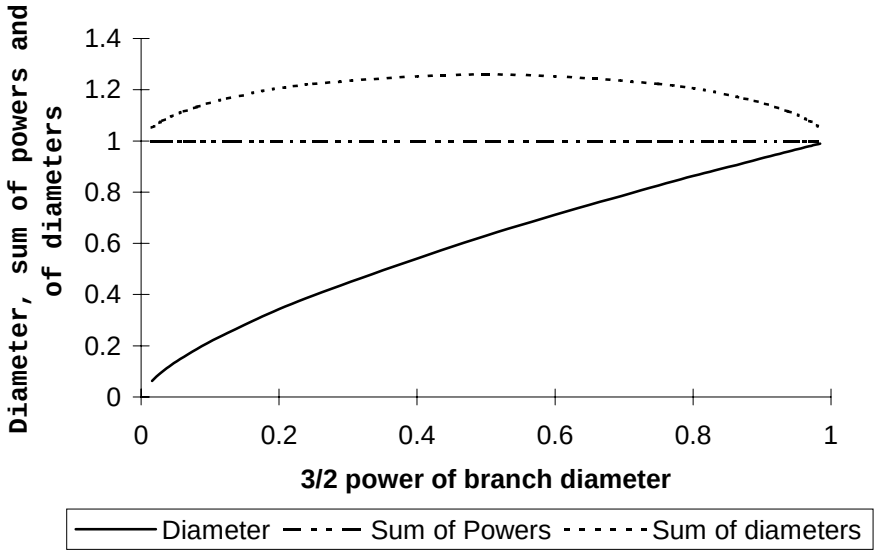

Figure 26. These three curves are indicate the diameter of one of the two branches, the 3/2 power of the diameters of both branches and the sum of those diameters as a function of the 3/2 power of the one branch.

Figure 27 shows the results of the simulations. For each set of problem parameters five information transfer rates were produced. The figure shows the minimum, the mean and the maximum of those values. They are given as a function of the 3/2 power of the diameter of the branch where the synapse was connected.

In the previous study where no branching took place, the information transfer rate was about 2.3 when the synapse was connected to the most compartment at the tip of the basal dendrite. Here, the synapse is also connected to a tip, but the information transfer rate never goes above 0.25.

It does, therefore, seem that branching of this kind is very damaging to the information transfer rate. However, the study applies only to the firing rate and ignores all other forms of information. Also, it only considers a single case in terms of the nature of branching, the distribution and behaviour of the synapses and the type of the neuron. Thus, at best these results raise some questions about how information is

transferred within in the dendritic tree. It should be studied more closely in future work.

Mainen and Sejnowski (1996) studied the influence of dendritic structure to the firing patterns in model neocortical neurons. They modified the dendritic geometry while maintaining the distribution of ion channels. They results suggested that changes in dendritic structure causes changes in the firing patterns. This is in line with our observation that the information transfer rate depends on the structure of the dendritic tree. It should, however, be kept in mind that the neural model used by Mainen and Sejnowski was different. Also, while changes in information transfer rate require changes in firing patterns, the opposite is not true.

On the other hand, we interpret the results in Figure 27 with considerable confidence in that the figure reflects some true property of the simulation. This can be argued by observing how consistently and closely the three curves follow one another. In particular, the differences are quite small compared to the changes in the values themselves over the studied portion of the x-axis.

Thus, it seems that in the simulation one cannot maximise the information transfer rate by maximising the diameter of the branch where the synapse connects. Instead the $3/2$ power of the optimal diameter would be in the neighbourhood of 0.96 of the diameter and the information transfer rate falls rapidly to about 0.05 bits per second in the neighbourhood of 0.8.

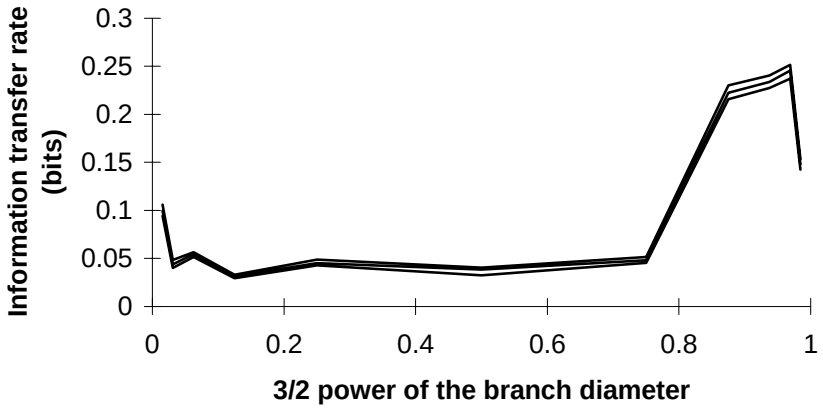

Figure 27. The information transfer rate as a function of the diameter of the branch where the synapse is connected. The three curves show the minimum, average and maximum of five runs.

Some data shown in Figure 27 is also shown in Figure 28 but the latter figure emphasises different aspects of the data. First, all the curves are based on average curve of Figure 27. Second, the curve if folded back at the 0.5 point of the x-axis forming the bottom and the middle curves in Figure 28. This way, if the 3/2 powers of the branch diameters are 1/8 and 7/8, one can read the information transfer rate at 0.125: the bottom curve for the thinner branch and the middle curve for the thicker branch. The top curve shows the sum of the other two curves.

An interesting property is the considerable degree of symmetry in the shapes of the two bottom curves. Where one is concave the other is convex. There is no apparent explanation why this is the case. A matter of future study is whether this is just an artefact of the simulation or something observable in the nature as well.

Another issue requiring a further study is comparing the branched and the non-branched cases. It seems that the information transfer rate is radically less in the branched case, even when the diameter of the branch without the input synapse is near zero. According to the results in this chapter, the information transfer rate is about 0.25 bits in the branched case whereas Section 13.1 shows information transfer rate of 2.28 for the non-branched case.

It is obvious that all the biological relevance of all these results should be questioned. The main reason is that they were obtained using a new

unproven methodology. Thus, all the results should rather be considered to be means of seeking validation to the methodology than be taken as biological truths. A secondary reason to question the biological relevance of the results is that some simplifications were needed for carrying out the simulations. These simplifications were needed mostly to overcome the limitations in the available computational power. The neural model was borrowed from previous work by Fransén and Lansner (1998) and is therefore not in question. To produce the stimuli, an addition to the neural model was needed, however. A complete description of that addition can be found in the Appendix.

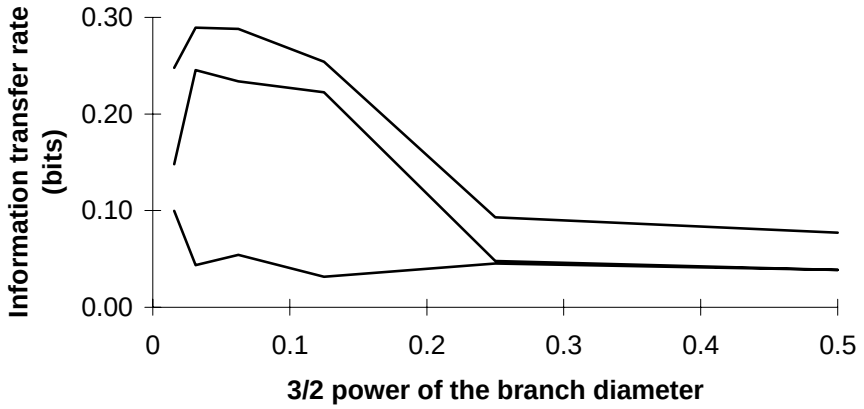

Figure 28. The bottom curve gives the information transfer rate as a function of the 3/2 power of the diameter of the branch where the synapse is connected (average of five runs). The middle curve gives it as a function of the 3/2 power of the diameter of the other branch (average of five runs). The top curve is the sum of the other two.

CHAPTER 14
SUMMARY AND CONCLUSIONS

14.1 PART ONE

This thesis starts by introducing the concept of a distribution code as a generalisation of the firing rate. A mathematical model is presented that unifies the representations of the codes found both in biological neural systems and in artificial, binary-valued, neural systems (Chapter 3).

When a distribution code is used, the values of source variables define the distribution of the input vectors. In decoding, the output values are determined by the distribution of the output vectors.

One-dimensional distribution codes were studied first. An upper limit for their channel capacity was found. The asymptotic behaviour, in terms of number of bits in the sample, of the upper limit turned out to be logarithmic. (Chapter 4)

A deterministic distribution code is one where the codewords are chosen deterministically as opposed to stochastically. A one-dimensional deterministic distribution code was studied. The code always minimises the error between the source variable and the target variable regardless of the number of bits used.

The channel capacity for this code was found and it equals the upper limit thus giving proof that the upper limit is achievable. If the source variable is uniformly distributed, it uses the channel capacity in a near-optimal way approaching the channel capacity asymptotically as the length of codewords increases. Some information is lost due to decoding based on the distribution of the bits (i.e., the relative frequency of 1s) as opposed to taking the ordering of bits into account. Apparently, no more than 50 per cent of information is ever lost (Chapter 5).

Also, a probabilistic one-dimensional distribution code was studied. A source variable that is uniformly distributed between 0 and 1 was used to determine the information transfer rate of the code. It turned out that the channel capacity is not reached with uniform distribution. It remains unknown whether there exists a distribution of the source variable that would make the information transfer rate reach the channel capacity. Unlike in the case of the deterministic code, no information is lost in decoding based solely on the distribution of the bits (Chapter 6).

14.2 PART TWO

The distribution function of a neural system is defined as the mapping from the distribution of the input vectors to the distribution of the output vectors. The distribution function is an analogue of the input-output function.

The distribution function of a deterministic memoryless neural system was identified. The function was also given in a recursive form, where each step of recursion corresponds to one unique, arbitrarily chosen bit position in the input vectors (Chapter 7).

It was then shown that the distribution function of such a network is monotonic assuming that one-dimensional distribution code is used – that is, there is one source variable per physical input and mutually independent codewords for each physical input – and that the values of the source variables can be chosen independently of each other. This is a severe limitation (Section 7.5). Finding remedies to it, however, gave interesting insights to the structures of neural systems – especially biological ones – and to the way they may function (Chapter 8, Chapter 9 and, specifically, Chapter 10).

By dividing the inputs of a neural system into arbitrarily chosen groups of proper inputs and control inputs it was learned that it is possible to affect the distribution function in an unexpected way. If we consider the input-output function of a real-valued neuron to be parameterised by the control inputs, having the proper inputs as its arguments and having the output of the neuron as its value, only translation of the function can be achieved by altering the control input values. If we consider the distribution function of a binary neuron, the function can be given as a linear combination of several parts. The exact

nature of the combination is determined by the values of the control inputs. This ability to change the shape of the distribution function without changing the weights of the neuron was named dynamic distribution-function alteration (Chapter 8).

A deterministic memoryless neuron was used for a case study. Several alternatives were considered.

First, it was assumed that each physical input has its own source variable and each source variable is coded using a one-dimensional distribution code. This results in mutually independent codewords for the physical inputs. In this case the distribution function has an S-shape similar to that of a typical input-output function of a real-valued neuron. However, the shape varies in discrete steps depending on the threshold value of the neuron.

Second, a control input was added to demonstrate the effect of dynamic distribution-function alteration. By using a single control input, a linear combination of two distribution functions can be created. The weights of the two distribution functions in the linear combination are not limited to discrete values and they can be changed dynamically.

Third, a single source variable was mapped onto two physical inputs. One-dimensional probabilistic distribution code was used resulting in a situation where the codewords of the physical inputs were independent of each other. This resulted in a non-monotonic distribution function.

Fourth, a multidimensional distribution code was used. In other words, the codewords of the physical inputs were no longer independent of one another. It was shown that changing the joint input distribution, even while maintaining the marginal distributions of each physical input constant, can change the output distribution.

Thus, the case study demonstrated key differences that a binary neuron combined with a distribution code can have compared to a real-valued neuron. The differences include the ability to change the shape of the distribution function – input-output function for a real-valued neuron – without changing the weights, and the ability to change the output value without changing any individual input value. The case study also showed that a neural system can have a non-monotonic distribution function (Chapter 9).

Neural systems of different structure and internal functioning were studied. As a starting point, it was shown that if one-dimensional distribution codes are used, a deterministic memoryless neural system must have at least n inputs in order to have an n-dimensional distribution function of its external inputs.

Using the deterministic memoryless neural system with one-dimensional distribution code as a baseline for comparison it was then shown how various functions and structures can be used to create non-monotonic distribution functions as well. This includes multidimensional distribution codes, recursive network structures, routes of unequal delay between two points in the network, non-deterministic neurons and an extended way of sampling the inputs.

The main limitation of the basic model used in this thesis is that the model does not take into account information that is dispersed over time. This problem can, however, be easily be avoided by extended sampling models, which were discussed in Chapter 9.

14.3 PART THREE

The last part of the thesis focuses on experimental work and biological neural systems.

First, the literature was surveyed to find the key methodologies that have been used in the information-theoretic study of biological neural systems and the key results obtained using those methodologies. It became apparent that a lot of work still needs to be done in solving two problems. First, how to gather enough data on biological systems for reliable statistical and information-theoretical analyses and, second, how to do comprehensive information-theoretical analyses with the computational tools of the present day or near future (Chapter 11).

Then, a new methodology was proposed where a software simulator is used to generate enough data for the analysis. Compared to gathering data by measuring live neurons, simulation is much simpler and faster. Simulation also gives strict control over the attributes of the studied neural system. This is especially useful if one is studying the effect of varying those attributes, e.g., the point where a synapse is attached in the dendritic tree of a neuron. Should one rely on biological samples, it might turn out to be a major task to find a set of neurons that differ from one another by this property alone. The proposed methodology helps with the problem of gathering data for analysis but gives nothing new in how the analysis could be carried out (Chapter 12).

A software implementation (11.3) of the methodology was described together with an easy-to-automate way of varying the topology of the

dendritic tree of a neuron without affecting the neurons other attributes. This approach is based on Rall's (1977) equations. The strengths and weaknesses of the implementation were discussed (Section 12.3).

All this was brought together in two simulations, which established a proof of concept as well as provide some interesting results. Both simulations studied a pyramidal neuron.

The first simulation modelled the basal dendrite of a neuron using a single branch. The axon of another neuron was connected to the dendrite with a synapse whose position was varied. The information transfer rate of the pyramidal neuron was computed as a function of the distance between the soma and the synapse. The result was, as expected, that the information transfer rate slowly decreases as the distance increases.

In the second simulation, the basal dendrite had two branches whose relative diameters were varied while maintaining other properties of the neuron unchanged – in terms of maintaining an equivalent cylinder (see Rall, 1977). Again, another neuron was connected to the basal dendrite with a synapse. The information transfer rate was computed. The result was that, if the relative diameter of the branch with the synapse is very small, the information transfer rate is higher than with equally sized branches. The information transfer rate becomes even higher when the relative diameter is very large.

There are no prior studies that would have helped in evaluating the validity of the results of the second simulation. The underlying causes for the results remain unknown thus suggesting further study.

Based on the two simulations, the proposed methodology is feasible and presents opportunities for doing new kind of research, e.g., in studying the effect of the topology of a neuron on the computational properties (Chapter 13).

REFERENCES

Anderson, J.A.: *An Introduction to Neural Networks*. Cambridge, Mass.: MIT Press, 1995.

Arleo, A., Gerstner, W.: A Vision-driven Model of Hippocampal Place Cells an Temporally Asymmetric LTP-induction for Action Learning. In *ICANN99. Ninth International Conference on Artificial Neural Networks. Vol. 1*, London: IEE, 1999.

Beale, R., Jackson, T.: *Neural Computing: An Introduction*. Bristol: Adam Hilger, 1991.

Bialek, W., DeWeese, M., Rieke, F., Warland, D.: Bits and brains: Information flow in the nervous system. *Physica A*, Vol. 200, No 1-4, pp. 581-593, 1993.

Bialek, W., Zee, A.: Coding and Computation with Neural Spike Trains. *Journal of Statistical Physics*, Vol. 59, No 2, pp. 103-115, 1990.

Bialek, W., Callan, C.G., Strong, S.P.: Field Theories for Learning Probability Distributions. *Physical Review Letters*, Vol. 77, No 23, pp. 4693-4697, 1996.

Cheeorts, M.N., Optican, L.M.: Cluster Method for Analysis of Transmitted Information in Multivariate Neuronal Data. *Biological Cybernetics*, Vol. 69, No 1, pp. 29–35, 1993.

Cover, T.M.: Geometrical and Statistical Properties of Linear Threshold Devices, Report SEL-64-052, TR 6107-1. Stanford, Calif.: Stanford University, 1964.

Cover, T.M.: Geometrical and Statistical Properties of Systems of Linear Inequalities with Applications in Pattern Recognition. *IEEE Transactions on Electronic Computers*, Vol. 14, pp. 326–334, 1965.

Deco, G., Shürmann, B.: The coding of information by spiking neurons: an analytic study. *Network: Computation in Neural Systems*, Vol. 9, No 3, pp. 303-17, 1998.

DeWeese, M., Bialek, W.: Information Flow in Sensory Neurons. *Nuovo Cimento D*, Vol. 17D, Ser. 1, No 7-8, pp. 733-741, 1995.

de Ruyter van Steveninck, R., Bialek, W.: Real-Time Performance of a Movement-Sensitive Neuron in the Blowfly Visual-System: Coding and Information Transfer in Short Spike Sequences. *Proceedings of the Royal Society of London, Series B – Biological Sciences*, Vol. 234, No 1277, pp. 379–414, 1988.

de Ruyter van Steveninck, R.R.., Laughlin, S.B.: The Rate of Information-Transfer at Graded-Potential Synapses. *Nature*, Vol. 379, pp. 642–645, 1996.

Eckhorn, R., Pöpel, B.: Rigorous and extended application of information theory to the afferent visual system of the cat. I. Basic concepts. *Kybernetik*, Vol. 16, No 4, pp. 191–200, 1974.

Eckhorn, R., Pöpel, B.: Rigorous and extended application of information theory to the afferent visual system of the cat. II. Experimental Results. *Biological Cybernetics*, Vol. 17, No 1, pp. 7–17, 1975.

Ekeberg, Ö., Stensmo M., Lansner A.: SWIM — A Simulator for Real Neural Networks. Technical report TRITA-NA-P9014. Stockholm: Royal Institute of Technology, Department of Numerical Analysis and Computing Science, 1990.

Eggert, J., van Hemmen, J.L.: Unifying framework for neuronal assembly dynamics. *Physical Review E*, Vol. 61, No 2, pp. 1855-1874, 2000.

Fransén, E., Lansner A.: A Model of Cortical Associative Memory Based on a Horizontal Network of Connected Columns, *Network: Computation in Neural systems*, Vol. 9, No 2, pp. 235–264, 1998.

Fukunaga, K.: *Introduction to Statistical Pattern Recognition.* New York, N.Y.: Academic Press, 1972.

Fuller, M.S., Williams, W.J.: A Continuous Information Theoretic Approach to the Analysis of Cutaneous Receptor Neurons. *Biological Cybernetics*, Vol. 47, No 1, pp. 13–16, 1983.

Gerstner, W.: Population dynamics of spiking neurons: fast transients, asynchronous states and locking. *Neural Computation*, Vol. 12, No 1, pp. 43-89, 2000.

Gerstner, W., van Hemmen, J.L.: Universality in neural networks: the importance of the `mean firing rate´. *Biological Cybernetics*, Vol 67, No 3, pp. 195-205, 1992a.

Gerstner, W., van Hemmen, J.L.: Associative memory in a network of `spiking´ neurons. *Network: Computation in Neural Systems*, Vol. 3, No 2, pp. 139-164, 1992b.

Gerstner, W., van Hemmen, J.L., Cowan, J.D.: What Matters in Neuronal Locking. *Neural Computation*, Vol. 8, No 8, pp.1653-1676, 1996.

Gerstner, W., Ritz, R., van Hemmen, J.L.: Why spikes? Hebbian learning and retrieval of time-resolved excitation patterns. *Biological Cybernetics*, Vol. 69, No 5-6, pp. 503-515, 1993.

Guttman, I., Wilks, S.S., Hunter, J.S.: *Introductory Engineering Statistics*. New York, N.Y.: Wiley, 1971.

Gruner, C.M., Johnson, D.H.: Correlation and neural information coding fidelity and efficiency. *Neurocomputing*, Vol. 26-27, pp. 163-168, 1999.

Hopfield J.J.: Neural Networks and Physical Systems with Emergent Collective Computational Abilities. *Proceedings of the National Academy of Sciences (Biophysics)*, Vol. 79, No 8, pp. 2554–2558, 1982.

Jaaksi, A.: Object-Oriented Development of Interactive Systems. Tampere University of Technology Publications 201. Tampere: Tampere University of Technology, 1997.

Johnson, D.H.: Point Process Models of Single-Neuron Discharges. *Journal of Computation Neuroscience*, Vol. 3, No 4, pp. 275-279, 1996.

Johnson, D.H., Gruner, C.M.: Information-Theoretic Analysis of Signal Processing Systems: Application to Neural Coding. In *Proceedings. 1998 IEEE International Symposium on Information Theory*, New York, N.Y.: IEEE, 1998a.

Johnson, D.H., Gruner, C.M.: Information-theoretic analysis of neural coding. In *Proceedings of the 1998 IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP '98, Vol. 4*, New York, N.Y.: IEEE, 1998b.

Johnson, D.H., Wang, W.: Symbolic Signal Processing. In *1999 IEEE International Conference on Acoustics, Speech and Signal Processing. Proceedings. ICASSP99. Vol. 3*, Piscataway, NJ: IEEE, 1999.

Johnson, D.H, Gruner, C.M., Glantz, R.M.: Quantifying Information Transfer in Spike Generation. *Neurocomputing*, Vol. 32-33, pp. 1047-1054, 2000.

Kanerva, P.: *Sparse Distributed Memory*. Cambridge, Mass.: MIT Press, 1988.

Kempter, R., Gerstner, W., van Hemmen, J.L., Wagner, H.: Extracting Oscillations: Neuronal Coincidence Detection with Noisy Periodic Spike Input. *Neural Computation*, Vol. 10, No 8, pp. 1987-2017, 1998.

Kistler, W.M., van Hemmen, J.L.: Delayed reverberation through time windows as a key to cerebellar function. *Biological Cybernetics*, Vol. 81, No 5-6, pp. 373-380, 1999.

Kjaer, T.W., Hertz, J.A., Richmond, B.J.: Decoding Cortical Neuronal Signals: Network Models, Information Estimation and Spatial Tuning. *Journal of Computational Neuroscience*, Vol. 1, pp. 109–139, 1994.

Knuth, D.E.: *The Art of Computer Programming, Vol. 2: Seminumerical Algorithms*, second edition. Reading, MA: Addison-Wesley, 1981.

Kohonen, T.: *Self-Organization and Associative Memory*, second edition. Berlin: Springer–Verlag, 1984.

Lettvin, J.Y., Maturana, H.R., McCulloch, W.S., Pitts, W.H.: What the Frog's Eye Tells the Frog's Brain. *Proceedings of the Institute of Radio Engineers*, Vol. 47, No 11, pp. 1940–1951, 1959.

Maass, W.: On the computational power of noisy spiking neurons. In *Advances in Neural Information Processing Systems 8. Proceedings of the 1995 Conference*, Touretzky, D.S., Mozer, M.C., Hasselmo M.E., eds., Cambridge, MA: MIT Press, 1996

Maass, W.: Noisy Spiking Neurons with Temporal Coding have more Computational Power than Sigmoidal Neurons. In *Advances in Neural Information Processing Systems 9. Proceedings of the 1996 Conference*, Mozer, M.C., Jordan, M.I., Petsche, T., eds., London: MIT Press, 1997a.

Maass, W.: Networks of spiking neurons: the third generation of neural network models. *Neural Networks*, Vol. 10, No 9, pp.1659-1671, 1997b.

Maass, W.: Fast sigmoidal networks via spiking neurons. *Neural Computation*, Vol. 9, No 2, pp. 279-304, 1997c.

Maass, W.: On the relevance of time in neural computation and learning. In *Algorithmic Learning Theory. 8th International Workshop, ALT '97. Proceedings.*, Li, M., Maruoka, A., eds., Berlin: Springer-Verlag, 1997d.

Maass, W.: On the Role of Time and Space in Neural Computation. In *Mathematical Foundations of Computer Science 1998. 23rd International Symposium, MFCS'98. Proceedings*, Brim, L., Gruska, J., Zlatuska, J., eds., Berlin: Springer-Verlag, 1998.

Maass, W., Natschläger, T.: Networks of Spiking Neurons Can Emulate Arbitrary Hopfield Nets in Temporal Coding. *Network: Computation in Neural Systems*, Vol. 8, No 4, pp. 355-371, 1997.

Maass, W., Orponen, P.: On the effect of analog noise in discrete time analog computations. *Neural Computation*, Vol. 10, No 5, pp. 1071-1095, 1998.

Maass, W., Ruf, B.: On the Relevance of the Shape of Postsynaptic Potentials for the Computational Power of Spiking Neurons. In *ICANN `95. Proceedings of International Conference on Artificial Neural Networks, Vol. 2. ICANN `95*, Fogelman-Soulie, F., Gallinari, P., eds., Paris: EC2 & Cie, 1995.

Maass, W., Ruf, B.: On Computation with Pulses. *Information and Computation*, Vol. 148, No 2, pp. 202-218, 1999.

Maass, W., Schmitt, M.: On the Complexity of Learning for a Spiking Neuron. In *Proceedings of the Tenth Annual Conference on Computational Learning Theory*, New York, N.Y.: ACM, 1997

Mainen, Z.F., Sejnowski, T.J.: Influence of dendritic structure on firing pattern in model neocortical neurons. *Nature*, Vol. 382, No 6589, pp. 363-366, 1996.

McCulloch, W.S., Pitts, W.H.: A Logical Calculus of the Ideas Immanent in Nervous Activity. *Bulletin of Mathematical Biophysics*, Vol. 5, pp. 115–133, 1943.

Minsky, M., Seymour, P.: *Perceptrons*, Cambridge, Mass: MIT Press, 1969.

Murray, A.F., Smith, A.V.W.: Asynchronous VLSI Neural Networks Using Pulse-Stream Arithmetic. *IEEE Journal of Solid-State Circuits*, Vol. 23, No. 3, pp. 688–697, 1988.

Natschläger, T., Maass, W.: Fast Analog Computation in Networks of Spiking Neurons Using Unreliable Synapses. In *7th European Symposium on Artificial Neural Networks. ESANN'99. Proceedings*, Verleysen, M., ed., Brüssels: D Facto, 1999.

Oppenheim, A.V., Johnson, D.H.: Discrete Representation of Signals: *Proceedings of the IEEE*, Vol. 60, No 6, pp. 681-691, 1972.

Optican, L.M., Gawne, T.J., Richmond, B.J., Joseph, P.J.: Unbiased Measures of Transmitted Information and Channel Capacity from Multivariate Neuronal Data. *Biological Cybernetics*, Vol. 65, No 5, pp. 305–310, 1991.

Pohja, S.: Probabilistic Interpretation of the Behaviour of a Binary Neuron. In *Brain Processes, Theories and Models*, R. Moreno-Díaz, J. Mira-Mira, eds., Cambridge, Mass: MIT Press, 1996.

Rall, W.: Core Conductor Theory and Cable Properties of Neurons. In *Handbook of Physiology, A Critical, Comprehensive Presentation of Physiological Knowledge and Concepts, Section I: The Nervous System, Volume I. Cellular Biology of Neurons, Part 1*, J.M. Brookhart, V.B. Mountcastle, E.R. Kandel, S.R. Geiger, eds., Bethesda, Maryland: American Physiological Society, 1977.

Richmond, B.J., Optican, L.M.: Temporal Encoding of Two-Dimensional Patterns by Single Units in Primate Visual Cortex. II. Information Transmission. *Journal of Neurophysicology*, Vol. 64, No 2, pp. 370–380, 1990.

Rieke, F., Warland, D., Bialek, W.: Coding Efficiency and Information Rates in Sensory Neurons. *Europhysics Letters*, Vol. 22, No 2, pp. 151–156, 1993.

Roberts, C.S.: Implementing and Testing New Versions of a Good, 48-bit, Pseudo-Random Number Generator. *The Bell System Technical Journal*. Vol. 61, No 8, pp. 2053-2063, 1982.

Ruderman, D.L., Bialek, W.: Seeing beyond the Nyquist limit. *Neural Computation*, Vol. 4, No 5, pp. 682-90, 1992.

Rumbaugh., J., Blaha, M., Premerlani, W., Eddy, F., Lorensen, W.: *Object-Oriented Modelling and Design*. Englewood Cliffs, N.J.: Prentice-Hall International, 1991.

Rumelhart, D.E., Hinton, G.E., Williams, R.J.: Learning Internal Representations by Error Propagation. In *Parallel Distributed Processing, Vol. 1*, D. Rumelhart, J. McClelland, eds., Cambridge, Mass.: MIT Press, 1986.

Schreiner, R.C., Eggick, G.K., Whitsel, B.L.: Variability in Somatosensory Cortical Neuron Discharge: Effects on Capacity to Signal Different Stimulus Conditions Using a Mean Rate Code. *Journal of Neurophysiology*, Vol. 41, No 2, pp. 338–349, 1978.

Shannon, C.E.: Communication in the Presence of Noise. *Proceedings of the Institute of Radio Engineers*, Vol. 37, pp. 10–21, 1949.

Spiridon, M., Gerstner, W.: Noise Modulation by Stochastic Neurons of the Integrate-and Fire Type. In *Foundations and Tools for Neural Modeling. International Work-Conference on Artificial and Natural Neural Networks, IWANN'99. Proceedings, Vol. 1 (Lecture Notes in Computer Science Vol. 1606)*, Mira, J., Sanchez-Andres, J.V., eds., Berlin: Springer-Verlag, 1999a.

Spiridon, M., Gerstner, W.: Noise spectrum and signal transmission through a population of spiking neurons. *Network: Computation in Neural Systems*, Vol. 10, No 3, pp. 257-272, 1999b.

Strong, S.P., de Ruyter van Steveninck, R.R., Bialek, W., Koberle, R.: On the Application of Information Theory to Neural Spike Trains. In *Pacific Symposium on Biocomputing '98*, Altman, R.B., Dunker, A.K., Hunter, L., Klein, T.E., eds., Singapore: World Scientific, 1997.

Strong, S.P., Koberle, R., de Ruyter van Steveninck, R.R., Bialek, W.: Entropy and information in neural spike trains, *Physical Review Letters*, Vol. 89, No 1, pp. 197-200, 1998.

Sugihara, I., Lang, E.J., Llinás, R.: Uniform olivocerebellar conduction time underlies Purjinke cell complex spike synchronicity in the rat cerebellum. *The Journal of Physiology*, Vol. 470, pp. 243-271,1993.

Tomberg, J.E., Kaski, K.K.K.: Pulse-Density Modulation Technique in VLSI Implementations of Neural Network Algorithms. *IEEE Journal of Solid-State Circuits*, Vol. 25, No. 5, pp. 1277–1286, 1988.

Tovee, M.J., Rolls, E.T., Treves, A.: Short Sample Periods Are Sufficient to Extract a Significant Proportion of the Total Information Encoded in a Neural Spike Train. *European Journal of Neuroscience*, No S5, pp. 262–262, 1992.

van Hemmen, J.L., Ritz, R.: Neural coding: A Theoretical Vista of Mechanisms, Techniques and Applications. In *Analysis of Dynamical and Cognitive Systems. Advanced Course. Proceedings*, Andersson, S.I., ed., Berlin: Springer-Verlag, 1995.

Wasserman, P.D.: *Neural Computing: Theory and Practice*. New York, N.Y.: Van Nostrand Reinholdm, 1989.

Widrow, B.: Adaptive Sampled-Data Systems: A Statistical Theory of Adaptation. In *1959 IRE WESCON Convention Record*, Part 4. New York, N.Y.: Institute of Radio Engineers, 1959.

Widrow, B., Hoff, M.: Adaptive Switching Circuits. In *1960 IRE WESCON Convention Record*. New York: Institute of Radio Engineers, 1960.

Windhorst, U., Schultens, H.A.: Measures of Trans-Information for Multiple Input Single Output Neuronal Systems. *Biological Cybernetics*, Vol. 45, No 1, pp. 57–61, 1982.

APPENDIX A: PARTS OF THE SIMULATION MODEL

This is a description of the input neuron that was used in the simulations of Chapter 13. Below, it is referred to as “Input”. The description is given in the specification language used by SWIM (see Ekeberg et al. 1990). The value of a random variable with uniform distribution between 0 and 300 should be inserted to replace **intensity**.

Also a description of the synapse that connects the input neuron and the neuron being studied (called “N”) is given. The desired compartment should be inserted to replace **compartment**.

This appendix, (Fransén and Lansner 1998) and description of the changes in the dendritic topology, which was given in the body of this thesis, together give a complete description of the simulated neural model.

```
define Input_Neuron
    Cm = 1e-4
    E.initial = -0.065
    Eleak = -0.065
    GCore = 0
    Gm = 1
    E.plot = Yes
    E.plot.min=-0.2
    E.plot.max=0.2
    Noise = Yes
    Noise
        TimeConst = 0.001
        Intensity = 20
        E = 0.0
        G = 2
end

define Input_Default_Synapse
  open plot
      min = 0.0
      max = 1.0

  delay = 0.001
  duration = 0.001
  raisetime = 0.0005
  decaytime = 0.020
end
```

```
define Input_Synapse (Input_Default_Synapse)
  E = 0.000
  G = 40e-9
end

NEURON Input (Input_Neuron)
    <soma> Noise.Intensity = intensity
END

SYNAPSE Input - N (Input_Synapse)
    pre = <soma>
    compartment = compartment
END
```

ISBN 951-22-5197-3
ISSN 1455-0474