CHAPTER 10

Cortical neurocomputation, language and cognition

Robert W. Kentridge

Introduction

Understanding or producing natural language requires considerable computational power. Although we cannot specify precisely what these requirements are, it is clear that they differ qualitatively from the simplest formal grammars. I argue that the computation taking place in most artificial neural networks as they process stimuli “on-line” cannot exceed the power of these simple regular grammars so alternative models of linguistic neurocomputation must be sought. One alternative framework for neurocomputation, pattern formation in networks with critical dynamics, is investigated. The results of reconstructions of the symbolic dynamics of one of these networks suggests that they may indeed be capable of implementing qualitatively more powerful forms of computation that natural language processing requires.

Any process can, in principle, be described in formal computational terms. Chomsky (1963) defined a hierarchical classification of different classes of formal computation which qualitatively distinguishes processes in terms of the types of grammars necessary to generate (or parse) different behaviours. This hierarchy can also be defined in terms of the automata equivalent to these grammars (Table 10.1).

Table 10.1 The Chomsky hierarchy of grammatical classes and their equivalent formal machine types.

<table>
<thead>
<tr>
<th>Grammar</th>
<th>Machine</th>
<th>Machine description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type 0</td>
<td>Unrestricted Turing machine</td>
<td>FSA with infinite tape</td>
</tr>
<tr>
<td>Type 1</td>
<td>Context-sensitive Linear-bounded automaton</td>
<td>FSA with finite tape</td>
</tr>
<tr>
<td>Type 2</td>
<td>Context-free Pushdown stack automaton</td>
<td>FSA with stack</td>
</tr>
<tr>
<td>Type 3</td>
<td>Regular Finite-state automaton (FSA)</td>
<td>Bare FSA</td>
</tr>
</tbody>
</table>

181
This is a strict hierarchy, no grammar in it can accept (or produce) behaviour only describable by a higher-level grammar. Because the hierarchy is equally applicable to the generation and the parsing of behaviours we can use it to infer the properties of unknown systems which generate particular behaviours from the class of computation required to parse those behaviours. In other words, if we can only meaningfully interpret a code produced by an unknown machine with a context-free grammar then we know that the machine cannot be described by a regular grammar (or finite-state automaton), it must be performing computation of at least context-free power in order to generate this behaviour (unless, of course we are interpreting “information” which was never encoded in the first place!).

Although it is not currently a popular area of research, one of the central questions addressed by cognitive scientists in the 1960s and early 1970s was the position of natural language in the Chomsky hierarchy of formal grammars. It is reasonably certain that natural language generation requires computational power above the lowest level of the hierarchy, regular grammar. It is less clear how the generative power of natural language is restricted relative to the highest class of the hierarchy (e.g. see Winograd 1983: 174–5). Nevertheless, we might assert that since humans generate and understand natural language, and presumably use their brains to do it, then one behaviour which we can assume a model of human brain function should exhibit is the ability to perform the computation necessary to parse or generate natural language. In other words, its intrinsic computational power should exceed that of regular grammars.

**Neural networks**

The dominant approach to modelling linguistic and cognitive processes used to be based around systems of symbolic propositions. Such “traditional” artificial intelligence systems could clearly implement computation above the power of regular grammars. They did, however, encounter many other problems when dealing with ambiguous or incomplete information. Artificial neural networks, on the other hand, are well suited to dealing with such imperfect stimuli and have the added appeal that they may provide us with some insights into the biological implementation of cognition.

Artificial neural networks consist of a large number of “neuron-like” elements connected together by links through which the activity of one unit influences the activities of those it is connected to. Entities are represented in neural networks as patterns of activation in these units. The strength of the connections between units determines how the pattern of activity in the network changes in response to an input stimulus. Depending on the arrangement of connections the effect of an external input can either propagate through the network in a single pass, or, if some of the connections feed back into the net, then its effects can develop over time (perhaps forever). In this way the pattern of activity produced in some of the units by an externally applied stimulus, together, perhaps, with
the initial state of the network, evolve to produce a new pattern of activity which represents the network’s “interpretation” of the stimulus. Given an appropriate method of adjusting the strengths of the connections between units a neural net can be made to classify stimuli and produce useful outputs even if the stimulus is incomplete or quite novel.

Many methods of adjusting connection strengths in neural networks have been devised, and have been used to solve real problems. This appears to be a very promising situation; however, some more detailed consideration of the way in which knowledge is represented in neural networks reveals potentially serious problems.

The problem of representing rules and relations in neural networks

There are two broad classes of neural nets, those in which a unit in the net “stands for” some quality (e.g. “redness”, “having feathers”), and those in which some “input” units stand for qualities of the stimulus (perhaps in terms of a sensory map, or perhaps in terms of more abstract qualities assumed to have been analyzed at an earlier stage) and some “output” units stand for qualities of the system’s interpretation of the stimulus (e.g. “is a fire engine”, “can fly”, “is a connected figure”, etc.), but most units are “hidden” and, although the relationship between input and output depends on them, they have no explicit individual meaning – their response to particular stimuli is determined by the algorithm used to construct appropriate connections in the net. In both cases, however, the representation of a stimulus in the network, unlike its representation in a symbolic system, is not arbitrary – it is dependent, albeit indirectly, on some qualities of the stimulus itself reflected in the coding of the stimulus in the input to the network. As representations are not arbitrary and are distributed over the whole network it is not possible to distinguish between a state of the network which represents the presence of two separate objects and one which represents a single complex object. The consequence of this is very serious – there is not only a problem representing multiple objects, it is also follows that it is equally hard to represent any relation between objects, and hence any rule. As Fodor & Pylyshyn (1988) argue, models of thinking require systematic representation of knowledge at a very minimum – that is, knowledge represented in a manner which allows generalizable relations to be represented. This does not appear to be the case for most neural networks.

Representing multiple objects in networks

One obvious method of allowing multiple representation to exist in a net, which is the first step towards encoding relations and rules, is to separate representations in time. This principle can be applied either sequentially, so that representations of items and relations follow one another, or in parallel, so that representations of multiple items and relations exist together in the network but
are distinguished on the basis of the temporal characteristics of their encoding (for example in terms of the relative phase or oscillation frequency of units involved in particular representations). In general, rule representation has been the explicit aim of networks using parallel temporal coding schemes (e.g. von der Malsburg & Bienenstock 1986, Shastri & Ajjanagadde 1989, Shastri & Ajjanagadde 1993), while it emerges and may not even be recognized in networks which identify structures over time. These networks are, however, capable, in principle, of encoding relations between items as well.

Computational power and network dynamics

There are then a number of methods which, either in principle or in practice, are capable of allowing neural networks to represent relations between objects. Before considering what types of computation these methods can, in principle, implement we must avoid a possible confusion. It has been shown that purely feedforward networks with at least one “hidden” layer of units are, in principle, capable of approximating any function in their mapping of input onto output states (Hornik et al. 1990). This corresponds to the highest level of the Chomsky hierarchy – unrestricted grammar. I would like, however, to draw a distinction between the computation performed by a neural network as it responds to stimuli (“on-line” computation) and the computation performed by the combined processes of training the network and preprocessing the stimuli combined with the network’s response to those stimuli. Although a feedforward network may behave as a universal function approximation, the network’s response to stimuli is essentially just a matter of table look-up (which can be conceived of as a very simple regular grammar in which a single transition is made from each starting state to a corresponding terminal state). The computation which makes this table look-up interpretable as an unrestricted grammar actually takes place during the training of the network. Given sufficient training and enough units, a feedforward network could be trained to decode messages produced by a stack automaton for example. The network is not, however, learning an algorithm, it is learning a set of input/output mappings. If we present the network with a coded message which it has not been exposed to during training, the extrapolation or interpolation of the function it has learned may not produce an appropriate output. Consider the extremes of representation which a feedforward net could apply to the problem. One possibility is that each starting state could be mapped onto an output consisting of the complete decoding of the message. At the other extreme, each symbol in the message could be mapped onto a single decoded symbol and a new “context” added to the next symbol of code to be processed. The first scheme is likely to fail to produce generalization as there is no way of simply partitioning input patterns (which is all a feedforward network can do) in order to uncover the regularities salient to a correct decoding. On the other hand, the single-symbol approach is likely to fail because, while the output required in response to a novel input state is related to generalizable features in
the pattern of the input, this relationship is ambiguous unless it is set in the context of more than one prior code symbol.

Having put "computation" in networks with no feedback, and hence no dynamics, to one side, let us now consider the behaviour of dynamic networks in a computational framework. The behaviour of networks with dynamics is not just dependent on the stimuli to which they are exposed at any instant, but also on their prior state. If we wish to make inferences about the computational power of networks it is important to establish how far this dependence of responses on the processing of prior stimuli extends back in time. There is a clear computational distinction between systems in which the transitions between states depend on current state and input — finite-state automata, and those in which state transitions depend on current input and a potentially infinite history of prior states — pushdown stack automata, linear-bounded automata, Turing machines, etc. (the distinctions between these latter types derive from constraints on the construction of the state histories they can use). What implication does this argument have for information processing with dynamic neural networks?

Information is often represented in dynamic neural networks as point attractors in the network's dynamics. From any given starting state, the state of the network evolves towards the attractor state whose basin of attraction the starting state happens to be in. The rules used to determine where attractors are formed and the addition of noise and other constraints allow this general scheme to be used in a wide variety of problems (e.g. pattern recognition or content-addressable memory (Hopfield 1982), optimization (Hopfield & Tank 1985) and constraint satisfaction (Ackley et al. 1985)). In computational terms the "normal" way of using these networks is of minimal computational power. The transition from an arbitrary starting state to an encoded attractor state is simply a mapping, just as the behaviour of the feedforward network was. If, however, we ask the question of how a subsequent stimulus might affect the network the situation becomes a little more interesting. If this new stimulus perturbs the states of some, but not all, of the neurons in the network then the basin of attraction which the network will consequentially find itself in depends on both the new stimulus and the prior state of the network. The transitions from basin of attraction to basin of attraction (which are the informationally significant subdivisions of the network's states) depend on the history of the network. This dependence is, however, only the "single-step" dependence of a finite-state machine, not the arbitrarily long time dependence of more powerful stack or tape machines. This becomes clear when we consider the nature of attractors. The reason that attractors are useful information-processing tools is that their use allows a system to selectively throw away "irrelevant" information. All of the different states of a system within one basin of attraction are reduced to a single attracting state. Once the network has reached one of these attracting states no information remains which can distinguish where in that state's basin of attraction the network had previously been. When a system makes a transition to a new attractor, all information of its prior states is therefore lost. The direct dependence of the
response of an attractor network to a stimulus cannot therefore extend beyond its current state. Exactly the same argument can be applied to networks which support multiple limit cycle or chaotic attractors as opposed to point attractors (as do various recurrent error-propagating networks (Pearlmutter 1988, Williams & Zipser 1989)) – the point is that once the system has converged onto one of a set of attractors it cannot directly recover information about which other members of the set it has visited or when it did so.

Any system using attractors to represent the informationally significant subdivisions of its state space does not appear capable of supporting computation above the level of finite-state automata. Nevertheless, physical dynamic systems which have greater computational power clearly exist, and some natural examples appear to have neural network processors. How can their dynamics be harnessed for computation if not in terms of a sequence of attractor transitions?

Much has been made in recent work on information processing in complex dynamical systems of the importance of “the edge of chaos” (Packard 1988, Crutchfield & Young 1989, Crutchfield & Young 1990, Mitchell et al. 1993). Phenomenologically, the interesting thing about physical systems undergoing “slow”, “second-order” or “critical” phase transitions (one of which involves being at “the edge of chaos”) is that their behaviour is dependent on interactions between their components which extend over arbitrary distances of space and time (e.g. see de Gennes 1975, Uzunov 1993). In terms of the computational consequences of dynamics we have been discussing this is very interesting since it implies that under these conditions dynamical systems may implement computation of greater power than finite-state automata. In some macroscopic physical systems these long-range interactions manifest themselves in the formation of coherent spatiotemporal patterns (e.g. see Nicolis & Prigogine 1989). The formation of these patterns in systems composed of billions of interacting components indicates that long-range interactions between components at phase transitions can greatly simplify the dynamics of complex systems – some relatively low-dimensional dynamics are governing the behaviour of the system, as opposed to those in which the system’s components behave more or less independently. There are two quite different ways in which fluctuations (which, for our purposes, may be the stimuli we occasionally inject into a network) can effect pattern-forming systems – multistability and defects (see Gaponov-Grekhov & Rabinovich (1992: ch. 4) for an excellent concise review). Effects which operate via multistability essentially move the system from a regime in which one type of pattern is stable into a regime where a different type of pattern is stable. Although it can be difficult to conceptualize complex pattern-forming systems in terms of simple attractors, these different pattern-forming states are clear analogues of attractors and, just as we argued in the case of multiple-attractor systems, information processing acting through fluctuations which move a system between stable pattern-forming modes cannot be more powerful than that of finite-state automata. In contrast to effects operating through multistability, fluctuations which produce defects do not result in the wholesale
replacement of one mode of behaviour by another. Defects are boundaries between areas whose dynamics differ in some way. These areas may be forming different types of patterns, or they may be forming different variants on the same pattern (just as defects in crystals separate regions of the crystal lattice which are out of register with one another). Once formed, defects can move and may either spread or decay. Most importantly, an existing defect can influence the effect that a fluctuation has on a system and, although it may be changed in some way if the fluctuation induces a new defect, the original defect is not inevitably destroyed. If we wish to think about information processing using defects in terms of attractors it is simplest to imagine a system governed by a single attractor with many different lobes (the Lorenz butterfly attractor has two lobes for example). The dynamics of the system lies on a transient converging towards the attractor. The system may move from orbiting around one lobe to another lobe spontaneously; however, externally applied fluctuations can force such changes to occur. In contrast to the information processing schemes discussed earlier which move between attractors, we can conceive of fluctuations in systems with critical phase transition dynamics as moving the state of the system within a single complex attractor (or at least on a transient very near to it).

Experiments on network dynamics and computational power

Introducing the model

The arguments presented above suggest that it would be interesting to examine pattern formation in neural networks with critical dynamics, and, if possible, to assess their computational power in this regime. As I mentioned earlier in the chapter, current artificial neural networks are not constructed with these dynamics in mind. On the other hand, both the argument above and some evidence from electrophysiological recordings (Freeman & van Dijk 1987, Kelso et al. 1992, Young et al. 1992) suggest that real brains may well support such dynamics. I have therefore studied these phenomena in network models which are more closely based on the anatomy and physiology of the cerebral cortex than most artificial neural network models. In the First Neural Computation and Psychology Workshop I discussed a biologically plausible neural network model which could be maintained in a critical dynamic regime over a wide range of conditions (Kentr1idge 1994). Very briefly, the model consists of simple spiking neurons which fire when their membrane potentials exceed a threshold. They are leaky integrators of their input stimulation, their membrane potential decaying exponentially with time: if $v(t) > \theta_i$ then

$$\forall \{t; t_r < t + r\}, v_i(t_r) = 0, x_j(t_r) = 0$$

$\{t; t_r = t + r\} \ v(t_r) = 0, x_j(t_r) = a p_j$ (10.1a)
if $v_j(t) < \theta_j$ then

$$\frac{dv_j(t)}{dt} = -kv_j(t) + \sum_{i \in \{i \neq j\}} w_{ij} x_i$$  \hspace{1cm} (10.1b)$$

where $v_j(t)$ is the membrane potential of neuron $j$ at time $t$, $\theta_j(t)$ is its threshold, $x_j(t)$ is its action potential, $a_{p_j}$ is the sign of action potential for that neuron (+1 for excitatory cells, -1 for inhibitory ones), $r$ is the refractory period of the neuron, $k_j$ is the membrane decay time-constant of the neuron and $w_{ij}$ is the strength of synapse from neuron $i$ to neuron $j$.

The network consists of a two-dimensional sheet of these neurons. Connectivity between neurons in this sheet is spatially localized – the probability of a synapse from one neuron to another being present being inversely proportional to a Gaussian of the distance between them (Fig. 10.1). The strengths of all such
Figure 10.2 A two-dimensional network in which a small amount of potential is uniformly and constantly added to each neuron. The waves of activity produced are partially desynchronized allowing more complex interactions than those occurring in very weakly driven networks with completely synchronized “target wave” activity patterns. Firing units are shown as solid circles. The membrane potential of other units is proportional to the diameter of the open circles corresponding to them.

synapses are set randomly. All neurons are excitatory in the simulations discussed here. When such a network is driven by uniform low-intensity stimulation of all neurons, spatiotemporally organized patterns of activity develop (Fig. 10.2). The network exhibits a range of behaviours typical of systems undergoing critical phase transitions, including pattern formation.

In the rest of this chapter I analyze the behaviour of this model in computational terms.

Measuring computation in dynamical systems

The general problem we face is to characterize the computational power of a system without prior knowledge of the way in which invariances in the system's
behaviour might correspond to “symbols” or “rules” which could be interpreted as forming that computation. This problem has been faced by a number of other authors. One approach is to attempt to adapt the system to solve computational problems of some known complexity. Attempts to use genetic algorithms to adapt cellular automata to solve particular computational problems in this manner (Packard 1988) have been beset by difficulties both in characterizing the dynamics of these discrete space and time systems (Li et al. 1990) and by determining whether it is those dynamics themselves, rather than their interaction with the genetic algorithms used to search the systems’ parameter space, which determines the capabilities of the systems found (Mitchell et al. 1993). The problem of searching the parameter space of the model neural network would be much more severe than those encountered in one-dimensional cellular automata space. This approach does not therefore seem appropriate.

An alternative to studying the solution of explicit computational problems is to characterize the implicit computational complexity of systems, that is, to determine the computational power of the simplest machine which can describe the behaviour of the system irrespective of any interpretation of that behaviour (Crutchfield & Young 1989). The aim is not, therefore, to find a way of achieving a particular computation, but rather of finding what class of computation the system would be capable of if a suitable interpretation could be found. Application of this method to very simple dynamical systems shows a qualitative difference between the implicit finite-state computation found in both ordered and chaotic regimes and the computational power of the type of stack machine required to describe the dynamics of the system at the transition between ordered and chaotic regimes (Crutchfield & Young 1990). Our hypothesis regarding the network is that it too should show this type of behaviour in the critical regime.

Before describing Crutchfield & Young’s algorithm for producing symbolic description of dynamical systems I will pause briefly to consider some general issues regarding the computational description of dynamical systems. In general we expect computational systems to be composed of discrete symbols and rules, whereas dynamical systems are much more general, covering systems with both continuous and discrete space and time evolution. The symbolic dynamics of a system with continuous space or time variables can be investigated by discretely sampling or averaging those variables to produce a version of the system with completely discrete dynamics. This is one of the features of Crutchfield & Young’s algorithm. Once we have produced a discrete picture of a system’s dynamics we then need to abstract a set of rules which describe those discrete dynamics. Rule abstraction can be based upon two separate views of the fundamental processes governing a system’s dynamics: deterministic or probabilistic. Rule abstraction from discrete sequences is, perhaps, most usually encountered in the concepts of algorithmic complexity (Chaitin 1975); here the aim is to find the most concise description of a discrete sequence of symbols – the rate of growth of the size of this description with the length of string to be described is an index of the complexity of the string. The most concise descriptions will often
EXPERIMENTS ON NETWORK DYNAMICS AND COMPUTATIONAL POWER

be in terms of sets of mappings between symbol sequences in the string and new sets of meta-symbols and of rules governing the allowable transitions between meta-symbols. These rules are deterministic, the outcome of this is that a symbol sequence generated from a chaotic data stream will always have maximal complexity (because no prior sequence of symbols is guaranteed to predict the next symbol deterministically, so the only effective description of the string is the string itself) even though we know the sequence was produced by a mathematically simple process and we know that only a limited set of states can ever be encountered. The induction of purely deterministic rules is therefore unlikely to capture a true picture of a dynamical system in symbolic computational terms. The alternative is the production of a probabilistic symbolic description of a system's dynamics. Such a description can capture important features of dynamical systems symbolically, for example the two distinct orbits of the Lorenz "butterfly" attractor and the relative probabilities of orbiting within and switching between them. Crutchfield & Young's algorithm produces probabilistic descriptions of this type, although, if presented with data which are described by a simple deterministic process, then an appropriate deterministic reconstruction will be produced.

The Crutchfield & Young algorithm

The aim of the algorithm is to produce the minimum complexity symbolic description of a dynamical system relative to a random-register Turing machine. The output of the system is the formal machine equivalent (referred to by Crutchfield & Young as an "ε-machine") of a stochastic grammar. Essentially the algorithm depends on finding the set of time invariances in a system's dynamics which have the simplest (minimum entropy) set of transitions between them. In outline the algorithm produces an ε-machine from a system's dynamics according to the following procedure.

Produce a quantized time series from the system's dynamics by dividing its state space into regions and sampling its position in state space in terms of these regions at fixed time intervals. From this time series construct a probabilistically labelled tree of all paths through the series up to a given length. From this tree produce the set of all unique types of subtrees (that is, subtree equivalence classes) of some smaller depth. These subtree equivalence classes correspond to time translation invariances in the original data stream. Subtree uniqueness is defined both by node labels, topology and by transition probabilities within the subtrees. The accuracy of the match in transition probabilities required for subtrees to be considered identical is a parameter δ. From the original tree find all of the transitions which transform one subtree into another (i.e. the transitions from the head of one subtree to the head of another subtree). Using these transitions produce a probabilistically labelled directed graph of the allowable transitions between subtrees (subtree equivalence classes correspond to nodes in this digraph). Repeat the above procedure with different tree and subtree depths and
values of the matching parameter $\delta$ until the graph indeterminacy $I_G$ is minimized. The graph indeterminacy $I_G$ is defined as

$$I_G = \sum_{v \in V} p(v) \sum_{s \in A} p(s | v) \sum_{v' \in V} p(v' | v; s) \log(p(v' | v; s))$$

(10.2)

where $V$ is the set of vertices in which $v \rightarrow v'$ is a particular transition whose edge is labelled $s$, $p(v' | v; s)$ is the probability of that transition and $p(s | v)$ is the probability that $s$ is emitted on leaving $v$. This is simply the weighted conditional entropy of the graph. It sums the entropies of all individual transitions between nodes $(p(v' | v; s)) \log(p(v' | v; s))$ weighted by the probability of encountering the output node $(p(v))$ times the probability that the symbol labelling an edge from that node will occur at that node $(p(s | v))$.

Symbolic reconstructions of cyclic and chaotic dynamics

As the machine reconstructed by this algorithm is stochastic, the algorithm can produce concise symbolic descriptions of both chaotic and limit cycle systems. Figure 10.3 shows some $\epsilon$-machine reconstructions from limit cycles produced by an iterated logistic map ($X_{t+1} = rX_t(1 - X_t)$). Figure 10.4 shows $\epsilon$-machine reconstructions from chaotic regions with single, multiple and merging chaotic bands. In these cases the $\epsilon$-machine reflects the different probability densities of value ranges of $X$ in the map and the transitions between those regions.

Note that all of these machines are finite – they correspond to regular languages – the lowest level of the Chomsky hierarchy. $\epsilon$-machines corresponding to higher-level grammars can, however, be produced by the Crutchfield & Young algorithm. If a system’s dynamics is optimally described by a higher-level grammar the algorithm produces truncations of an infinite machine corresponding to that grammar. Regularities in the structure of this machine may allow us to infer the data structures (stacks or tapes) which would allow the machine to be represented in finite terms and the grammar to which the machine corresponds. We can distinguish truncations of infinite machines from finite machines by examining the machines produced from a single data set using successively larger tree depths in the reconstruction algorithm. If the optimal $\epsilon$-machine is finite then this machine will eventually be produced using a particular tree depth, and when reconstructions are performed using deeper trees the same machine will be produced. On the other hand, if the optimal $\epsilon$-machine is

![Figure 10.3 $\epsilon$-machine reconstructions of periods 1, 2 and 4 limit cycles from the iterated logistic map $X_{t+1} = rX_t (1 - X_t)$ with $r = 2.5, 3.4$ and $3.5$ respectively.](image-url)
The optimal machine size does not reach a limit as reconstruction depth is increased. Applying the Crutchfield & Young algorithm to the neural network

It may appear impractical to apply the Crutchfield & Young algorithm to a network consisting of thousands of neurons (although it is in principle possible) when it can take considerable computational resources to reconstruct an $\varepsilon$-machine from one-dimensional maps in some parameter ranges. If, however, there is some systematic collective behaviour occurring in the network then this should be reflected in the behaviour of single neurons if sampled over a sufficiently long time. I now present results of machine reconstructions from time series of membrane potentials of a single unit from a network showing critical dynamics sampled over 100 000 time steps. As a control, the same neuron was sampled under identical driving conditions but disconnected from all other units. First of all I present some data which give a general impression of the behaviour of the network and of a single neuron in isolation.
Figure 10.6 Power spectrum (frequency on the abscissa, power on the ordinate) of the 5625-neuron network from one neuron of which the machine reconstructions were subsequently made.

Figure 10.7 Interspike interval histogram collected from an individual unit in a network over approximately 50,000 time steps.
The network’s power spectrum shown in Figure 10.6, measured in terms of the number of neurons firing in the network per time step, gives some indication of the global dynamics of the network. The spectrum shows the 1/f pattern typical of critically organized systems but superimposed with a single peak indicating the presence of some collective periodicity in the network. This spectrum is remarkably similar to those found by Young et al. (1992) in the visual and medial–temporal cortices of macaque monkeys and is quite similar to those found in the visual cortex of rhesus monkeys by Freeman & Van Dijk (1987) (who found 1/f power overlaid with multiple, rather than single, periodic peaks). Work is currently in progress on a more accurate simulation of EEG recording from the network for comparison with these results.

The power spectrum of individual units does not convey much useful information due to the response of the Fourier transform to the instantaneous drops of membrane potential as the unit fires. An interspike interval histogram, showing the distribution of times between the generation of successive action potentials in a single neuron (Fig. 10.7), however, gives a good conception of an individual unit’s behaviour when embedded in the network.

The same neuron when isolated produces a constant interspike interval of 18, as would be expected from its simple evolution equation and constant driving current. It is also obvious that the effect of being embedded in an excitatory network can only reduce a unit’s interspike interval.

Initially, I would like to explore the application of the e-machine reconstruction algorithm to the isolated neuron. The relative simplicity of its behaviour allows the meaning of the machine reconstructions to be understood easily. At the maximum possible sampling rate we can use for reconstruction (corresponding to single time steps in the simulation which generated the data) we can choose a potential value at which to code 0s and 1s in the initial reconstruction string which discriminates perfectly between the refractory and active phases of the neuron. This discretization value falls at a potential between 0 and 0.4 for the current simulation. The machine reconstructed at this sampling rate and using this discretization value is shown in Figure 10.8.

This machine was reconstructed to a subtree depth of 19 in order to capture the full 18-step periodicity of the unit’s behaviour. The indeterminacy $I_{e}$ is zero. The first two layers of reconstruction establish the initial point in the time series.

![Figure 10.8 Machine reconstructed from the membrane potential time series of an isolated model neuron. The sampling rate and potential discretization were chosen to highlight the refractory and active phases of the neuron’s behaviour. A subtree depth of 19 was used in the reconstruction.](image)
Figure 10.9 Machine reconstructed from the membrane potential time series of an isolated model neuron. The sampling rate and potential discretization were chosen to minimize the reconstruction depth required to describe the neuron’s behaviour. A subtree depth of 10 was used in the reconstruction.

at which a tree is encountered. Beyond that the paths by which four refractory steps are always followed by 14 active ones can clearly be traced around the machine reconstruction.

Although this reconstruction is easy to relate to the neuron’s membrane potential time series, we can produce simpler reconstructions. If, instead of discretizing in order to discriminate refractory and active phases of the neuron’s behaviour, we discretize so that half of the labels in our initial string are 0s and half are 1s by choosing a value near to half of the neuron’s firing threshold (in this case about 2.5) then much shallower reconstructions can capture the full dynamics of the neuron, as shown in Figure 10.9.

Instead of needing to reconstruct to a subtree depth of 19, an invariant zero indeterminacy machine is produced at depths of 10 and above.

In addition to changing the discretization value we can also change our sampling rate in order to further simplify the reconstructed machine. The machine shown in Figure 10.10 was produced using a sampling rate four times as fast as that used in the previous examples.

This appears to be the most concise symbolic description of the neuron’s behaviour possible. Higher sampling rates can fail to discriminate the refractory period of the neuron (which also represents the strongest nonlinearity in its otherwise almost linear behaviour) and hence tend to produce non-zero indeterminacy reconstructions.

196
We now turn to reconstruction of the behaviour of the same neuron when it is embedded in the network. A reconstruction using exactly the same parameters as used in Figure 10.10 is shown in Figure 10.11.

The difference between Figures 10.10 and 10.11 is striking. The machine reconstructed from the isolated neuron is clearly finite whereas the neuron embedded in the network produces a machine which appears to grow as reconstruction depth increases, which gives no sign of imminent closure and yet which still has zero indeterminacy. Note, however, that reconstruction was only possible to a subtree depth of 6, as the number of possible paths through the data becomes extremely large for the embedded neuron, and computing deeper reconstructions becomes very time- and space-intensive.

It is also possible to produce machine reconstruction by producing strings labelled 1 whenever an action potential has just occurred and 0 otherwise. This initial step does not follow Crutchfield & Young's algorithm; however, the rest of the reconstruction is standard. This method produces far fewer initial paths through the data, so a depth 12 reconstruction of the embedded neuron's behaviour was possible. This is particularly useful since, at a sampling rate of four time steps, machine closure due to the maximum periodicity of 18 time steps would be expected at a depth of no more than 8. The depth 10 machine produced is shown in Figure 10.12.

This reconstruction also appears to be a truncation of an infinite machine; moreover, it shows the lengthening sequences of deterministic transitions which are typical signs of data structures such as stacks or tapes. Its indeterminacy was zero. These findings strongly suggest that the behaviour of this single neuron, when embedded in a network, is optimally described by, and is capable of implementing, computation at the level of context-free grammars or above.
Before accepting this conclusion, however, an alternative explanation for this complex behaviour must be investigated. Would the simple presence of multiple interspike interval periodicities in the data stream produce such behaviour? This alternative explanation can easily be tested using surrogate data.

Figure 10.13 shows the invariant machine reconstruction from surrogate data in which three periodicities, 4, 5 and 6, were mixed with different probabilities. Each choice of period was independent of previous ones. It can clearly be seen that the reconstructed machine is finite. In fact, it does not differ in form from a period 6 reconstruction; the only influence of the shorter periodicities is to change the transition probabilities in the graph.

It might still be argued that the strong nonlinearity of refractoriness combined with multiple independent periodicities might produce complex machines such as that reconstructed from the neuron embedded in the network. Figure 10.14, which shows a machine reconstructed from surrogate data with three independent periodicities of 4, 5 and 6 each followed by a fixed two-step “refractory” period, indicates that this is not the case.
Figure 10.12 Machine reconstructed from an action potential time series of a model neuron embedded in a network. A subtree depth of 12 was used in the reconstruction.

Figure 10.13 Machine reconstructed from surrogate data in which periodicities of 4, 5 and 6 follow one another randomly with different occurrence probabilities (0.3, 0.4 and 0.3, respectively).
The conclusions we can reach from these e-machine reconstructions are that the collective effect of the network on an individual neuron’s dynamics is to raise its intrinsic computation to the level of context-free grammars or above. This effect is due to the production of multiple periodicities; moreover, the length of each interspike interval must depend on the previous behaviour of the neuron. This dependence continues to outweigh a simple random prediction of all of a neuron’s subsequent behaviour in minimizing graph indeterminacy even at large reconstruction depths. Such long-range influences are typical of systems with critical dynamics, lending credence to the hypothesis that neural networks in a critical regime might perform computation beyond regular grammars.

Discussion

Reconstruction issues

The work presented here represents a considerable advance on preliminary work I reported in Kentridge (1995a). In that paper the network used produced interspike intervals in the range of 30–50 time steps at the maximum sampling rate. Although the reconstructions presented could identify the refractory and active phases of an isolated neuron, and could differentiate between the relatively simple isolated neuron behaviour and more complex embedded behaviour, it was quite impractical to reconstruct to a depth sufficient to reveal the true nature of the neuron’s dynamics. Higher sampling rates could not significantly improve the situation, probably because the refractory period of the units was relatively small and its active behaviour almost linear. These results, together
with those presented in this chapter, suggest that adding an explicit nonlinearity in addition to refractoriness to the model neuron's behaviour may have a significant effect on the practicalities of machine reconstruction.

Explicit models of "neural" symbol processing

The recent work aimed at producing "neural" implementations of explicitly symbolic processes mentioned in the introduction (e.g. von der Malsburg & Bienenstock 1986, Shastri & Ajjanagadde 1989, Shastri & Ajjanagadde 1993) has taken a very different approach from the one I have described here. These explicit approaches have centred on the problem of variable binding. When the features of objects are represented as distributed patterns of activity in a network, the representation of more than one object at a time in a network becomes problematic - how can different aspects of the overall pattern of activity in the network be associated with distinct objects? The ability to temporarily bind sets of representations or characteristics together has far more general uses than the avoidance of perceptual chimeras. Any use of syntax in information processing depends upon binding - how, otherwise, can "the man" sometimes be the subject and sometimes be the object of a sentence? The most common solution to the binding problem in recent years has been to link the aspects of a network's activity to be associated with a distinct entity on a finer timescale than that at which the co-occurrence of those entities is represented. In practice this separation of timescales is achieved by using "neurons" whose activity oscillates quickly relative to the rate of appearance of the entities being represented. Bindings are made through the relative phases of these oscillations. Such a system allows the patterns of activity (defined by the amplitudes of oscillations) corresponding to long-term representations (e.g. "happy", "sad", "man", "dog", "subject", "object") to be separated from specific instances in which sets of these patterns are temporarily bound together (defined by the phase of oscillations). We can, for example, begin to represent a sentence like "The sad man hit the happy dog" by arranging for neurons in the patterns corresponding to "sad", "man" and "subject" to oscillate in phase with one another while those representing "happy", "dog" and "subject" also oscillate together but in a different phase from the first set. This solution to the binding problem is especially attractive in the light of recent evidence of stimulus-dependent oscillations of neural activity in the visual cortex. In the framework I have been discussing, variable binding is only part of the more general problem of implementing symbolic processes neurally. Taken in isolation, the oscillation through binding story implies that bound symbols are represented as limit cycles; however, the necessarily temporary nature of binding implies that, at another level, these limit cycles are only transiently stable. Although I will argue that the neural computation underlying cognitive processes is best viewed in terms of transient, rather than attractor, dynamics, the difference between the approach I have described in this chapter and work on oscillation and binding is really one of emphasis. Whether we
choose to concentrate on stability or the organization of systematic transitions in
neural dynamics depends on our interests - on one hand the nature of neural
implementation of representations, on the other hand the nature of the processes
which transform those representations. On short timescales, when concentrating
on mechanisms of representation, it may be appropriate to treat oscillations, both
in model networks and in real brains, as attractors. In this chapter, however, I
have attempted to begin addressing a problem which is set in a longer timescale
where the transient qualities of a network’s dynamics determine its functional
properties.

**Implications for cognitive neurocomputation**

The work presented in this chapter can be seen as a hypothesis of brain function,
and one can indeed derive propositions from it which may be tested electrophysiologically. Our purpose here, however, is to examine the implications of
this work for our understanding of the nature of neural computation and its
relationship to cognitive processes in language and memory.

The rest of this section rests on the premise that computation at the level of
context-free grammars or above in dynamical systems requires nonlinear feedback (see also Kentridge 1993, Kentridge 1995b). This should be quite uncontroversial. If this computation is to proceed effectively in the presence of noise then
the system must be in a critical regime, as the network model described earlier
was. Given these assumptions, how are the equivalents of symbols and rules
instantiated in dynamical systems?

**Neural computation is transient, memories and experience are not identical**

When we think about dynamical systems we tend to envisage them in terms of
attractors. Computation at the level of context-free grammars or above in noisy
dynamical systems cannot, in principle, occur if an attractor is reached. Compu-
tation at this level requires the system to follow a potentially infinite transient,
perturbed occasionally by external stimuli. The path of this transient is, of
course, influenced by topography of the system’s state space, which in turn is
determined by the form of attractors in the system. There is a reason for this
beyond the fact that critical dynamics imply infinite transients per se (the phe-
nomenon of “critical slowing” in physical systems near criticality is evidence of
the “reluctance” of such systems to reach attractors). In a system performing
computation at or above the level of context-free grammars the prior history of
the system at some arbitrary point in the past must be able to influence the sys-
tem’s evolution. Once a system has reached an attractor it is impossible to deter-
mine the path taken into the attractor. When an external influence moves the
system out of the attractor, all information about the system’s history prior to
reaching that basin of attraction is lost. In order to implement computation more
powerful than regular grammars, then, the dynamics of the system must be
forever transient.
What does this mean psychologically? In this model, information is stored by altering the form of the attractor which governs the system’s dynamics. The attractor is, however, never reached. Stimuli therefore do not recreate memories. The effect a stimulus has on the state of the system is influenced by memories (the attractor) but the state of the system is not the same as those memories. This, of course, makes perfect psychological sense!

Symbols are non-atomic
Symbols in a dynamical system are invariances in the system’s dynamics. However, in chaotic or nearly chaotic regimes the system will never exactly retrace its path, so the invariances we can make use of as symbols are only approximate. Although one particular level of approximation may produce some desired characteristics, such as minimal indeterminacy in the system’s symbolic description, there is no reason to assume that invariances at finer or coarser levels of approximation might not also be regarded as symbols. As these invariances are approximate, predictions of the transitions between them will also often have to be approximate. More accurate predictions can be made for the transitions between finer-grained approximate invariances than between coarse ones. The upshot of this is that a whole range of different ways of coding a segment of dynamics in symbolic terms is possible. Transitions which appear random at one level of coding are less so at other levels. When viewed in these terms, “symbols” in neurocomputation are not truly symbolic – rather than being atomic and content-free they consist of hierarchies of more detailed contents. These hierarchies may mix semantic and syntactic information in a similar way to the structures revealed by analyses of invariances in text corpora (e.g. see Chater & Conkey 1994, Finch & Chater 1994). This conclusion says slightly more than “representations are distributed”. Their distribution is also meaningfully structured. It is particularly interesting to note that Wuensche (1993) arrives at a very similar notion from a quite different direction in his cellular automaton model of memory.

Acknowledgements
This research was supported by DARA Fort Halstead, UK. Contract number 2051/047/RARDE.

References
CORTICAL NEUROCOMPUTATION, LANGUAGE AND COGNITION


REFERENCES


