A Beginner's Guide to the Mathematics of Neural Networks

A.C.C. Coolen

Department of Mathematics, King's College London

Abstract

In this paper I try to describe both the role of mathematics in shaping our understanding of how neural networks operate, and the curious new mathematical concepts generated by our attempts to capture neural networks in equations. My target reader being the non-expert, I will present a biased selection of relatively simple examples of neural network tasks, models and calculations, rather than try to give a full encyclopedic review-like account of the many mathematical developments in this field.

Contents

1 Introduction: Neural Information Processing

Our brains perform sophisticated information processing tasks, using hardware and operation rules which are quite different from the ones on which conventional computers are based. The processors in the brain, the neurons (see figure 1), are rather noisy elements[1] which operate in parallel. They are organised in dense networks, the structure of which can vary from very regular to almost amorphous (see figure 2), and they communicate signals through a huge number of inter-neuron connections (the so-called synapses). These connections represent the 'program' of a network. By continuously updating the strengths of the connections, a network as a whole can modify and optimise its 'program', 'learn' from experience and adapt to changing circumstances.
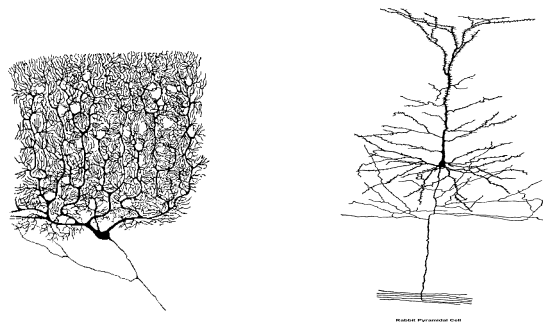

Figure 1: Left: a Purkinje neuron in the human cerebellum. Right: a pyramidal neuron of the rabbit cortex. The black blobs are the neurons, the trees of wires fanning out constitute the input channels (or dendrites) through which signals are received which are sent off by other firing neurons. The lines at the bottom, bifurcating only modestly, are the output channels (or axons).

From an engineering point of view neurons are in fact rather poor processors, they are slow and unreliable (see the table below). In the brain this is overcome by ensuring that always a very large number of neurons are involved in any task, and by having them operate in parallel, with many connections. This is in sharp contrast to conventional computers, where operations are as a rule performed sequentially, so that failure of any part of the chain of operations is usually fatal. Furthermore, conventional computers execute a detailed specification of orders, requiring the programmer to know exactly which data can be expected and how to respond. Subsequent changes in the actual situation, not foreseen by the programmer, lead to trouble. Neural networks, on the other hand, can adapt to changing circumstances. Finally, in our brain large numbers of neurons end their careers each day unnoticed. Compare this to what happens if we randomly cut a few wires in our workstation.

---

[1] By this we mean that their output signals are to some degree subject to random variation; they exhibit so-called spontaneous activity which appears not to be related to the information processing task they are involved in.
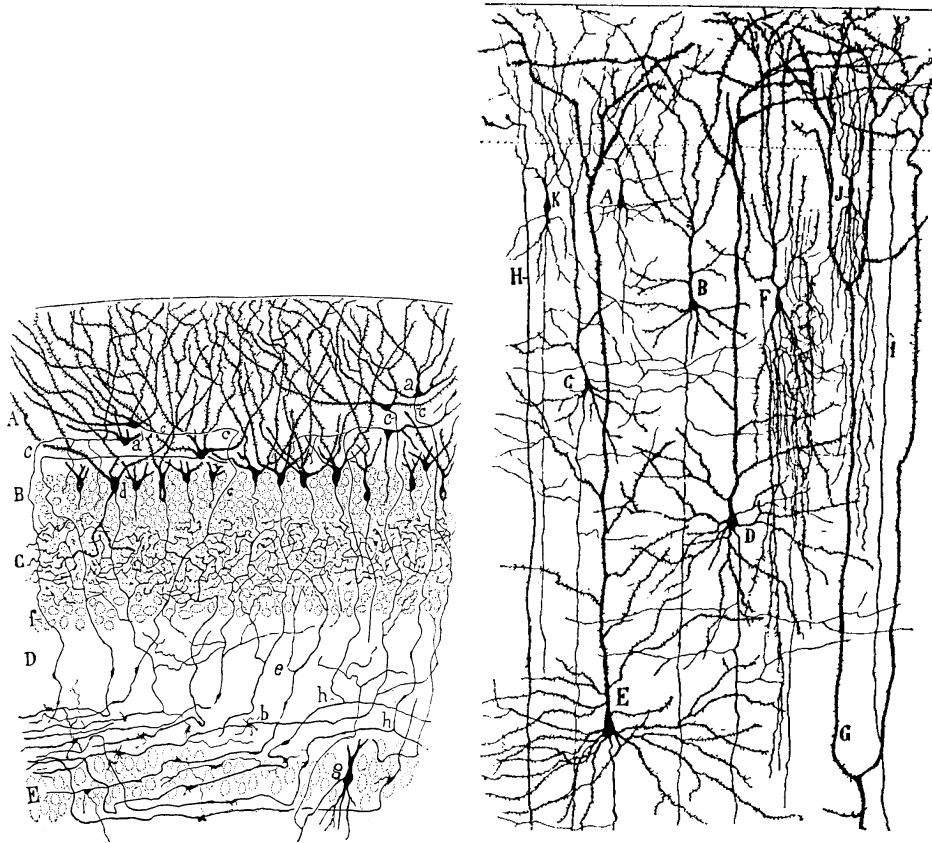
Figure 2: Left: a section of the human cerebellum. Right: a section of the human cortex. Note that the staining method used to produce such pictures colours only a reasonably modest fraction of the neurons present, so in reality these networks are far more dense.

Roughly speaking, conventional computers can be seen as the appropriate tools for performing well-defined and rule-based information processing tasks, in stable and safe environments, where all possible situations, as well as how to respond in every situation, are known beforehand. Typical tasks fitting these criteria are e.g brute-force chess playing, word processing, keeping accounts and rule-based (civil servant) decision making. Neural information processing systems, on the other hand, are superior to conventional computers in dealing with real-world tasks, such as e.g. communication (vision, speech recognition), movement coordination (robotics) and experience-based decision making (classification, prediction, system control), where data are often messy, uncertain or even inconsistent, where the number of possible situations is infinite and where perfect solutions are for all practical purposes non-existent.

One can distinguish three types of motivation for studying neural networks. Biologists, physiologists, psychologists and to some degree also philosophers aim at understanding information processing in real biological nervous tissue. They study models, mathematically and through computer simulations, which are preferably close to what is being observed experimentally, and try to understand the global properties and functioning of brain regions.

conventional computers	biological neural networks
processors $\quad operation\ speed \sim 10^8 Hz$ $\quad signal/noise \sim \infty$ $\quad signal\ velocity \sim 10^8 m/sec$ $\quad connections \sim 10$	neurons $\quad operation\ speed \sim 10^2 Hz$ $\quad signal/noise \sim 1$ $\quad signal\ velocity \sim 1m/sec$ $\quad connections \sim 10^4$
sequential operation program & data external programming	parallel operation connections, neuron thresholds self-programming & adaptation
hardware failure: fatal no unforseen data	robust against hardware failure messy, unforseen data

Engineers and computer scientists would like to understand the principles behind neural information processing in order to use these for designing adaptive software and artificial information processing systems which can also 'learn'. They use highly simplified neuron models, which are again arranged in networks. As their biological counterparts, these artificial systems are not programmed, their inter-neuron connections are not prescribed, but they are 'trained'. They gradually 'learn' to perform tasks by being presented with examples of what they are supposed to do. The key question then is to understand the relationships between the network performance for a given type of task, the choice of 'learning rule' (the recipe for the modification of the connections) and the network architecture. Secondly, engineers and computer scientists exploit the emerging insight into the way real (biological) neural networks manage to process information efficiently in parallel, by building artificial neural networks in hardware, which also operate in parallel. These systems, in principle, have the potential of being incredibly fast information processing machines.

Finally, it will be clear that, due to their complex structure, the large numbers of elements involved, and their dynamic nature, neural network models exhibit a highly non-trivial and rich behaviour. This is why also theoretical physicists and mathematicians have become involved, challenged as they are by the many fundamental new mathematical problems posed by neural network models. Studying neural networks as a mathematician is rewarding in two ways. The first reward is to find nice applications for one's tools in biology and engineering. It is fairly easy to come up with ideas about how certain information processing tasks could be performed by (either natural or synthetic) neural networks; by working out the mathematics, however, one can actually

quantify the potential and restrictions of such ideas. Mathematical analysis further allows for a systematic design of new networks, and the discovery of new mechanisms. The second reward is to discover that one's tools, when applied to neural network models, create quite novel and funny mathematical puzzles. The reason for this is the 'messy' nature of these systems. Neurons are not at all well-behaved: they are microscopic elements which do not live on a regular lattice, they are noisy, they change their mutual interactions all the time, etc.

Since this paper aims at no more than sketching a biased impression of a research field, I will not give references to research papers along the way, but mention textbooks and review papers in the final section, for those interested.

2   From Biology to Mathematical Models

We cannot expect to solve mathematical models of neural networks in which all electro-chemical details are taken into account (even if we knew all such details perfectly). Instead we start by playing with simple networks of model neurons, and try to understand their basic properties first (i.e. we study elementary electronic circuitry before we volunteer to repair the video recorder).

2.1   From Biological Neurons to Model Neurons

Neurons operate more or less in the following way. The cell membrane of a neuron maintains concentration differences between inside and outside the cell, of various ions (the main ones are Na^+, K^+ and Cl^-), by a combination of the action of active ion pumps and controllable ion channels. When the neuron is at rest, the channels are closed, and due to the activity of the pumps and the resultant concentration differences, the inside of the neuron has a net negative electric potential of around −70 mV, compared to the fluid outside. A sufficiently strong local electric excitation, however, making the cell potential temporarily less negative, leads to the opening of specific ion channels, which in turn causes a chain reaction of other channels opening and/or closing, with as a net result the generation of an electrical peak of height around +40 mV, with a duration of about 1 msec, which will propagate along the membrane at a speed of about 5 m/sec: the so-called action potential. After this electro-chemical avalanche it takes a few milliseconds to restore peace and order. During this period, the so-called refractory period, the membrane can only be forced to generate an action potential by extremely strong excitation. The action potential serves as an electric communication signal, propagating and bifurcating along the output channel of the neuron, the axon, to other neurons. Since the propagation of an action potential along an axon is the result of an active electro/chemical process, the signal will retain shape and strength, even after bifurcation, much like a chain of tumbling domino stones.

typical time-scales	
action potential:	$\sim 1 msec$
reset time:	$\sim 3 msec$
synapses:	$\sim 1 msec$
pulse transport:	$\sim 5 m/sec$

typical sizes	
cell body:	$\sim 50 \mu m$
axon diameter:	$\sim 1 \mu m$
synapse size:	$\sim 1 \mu m$
synaptic cleft:	$\sim 0.05 \mu m$

The junction between an output channel (axon) of one neuron and an input channel (dendrite) of another neuron, is called synapse (see figure 3). The arrival at a synapse of an action potential can trigger the release of a chemical, the neurotransmitter, into the so-called synaptic cleft which separates the cell membranes of the two neurons. The neurotransmitter in turn acts to selectively open ion channels in the membrane of the dendrite of the receiving neuron. If these happen to be Na^+ channels, the result is a local increase of the potential at the receiving end of the synapse, if these are Cl^- channels the result is a
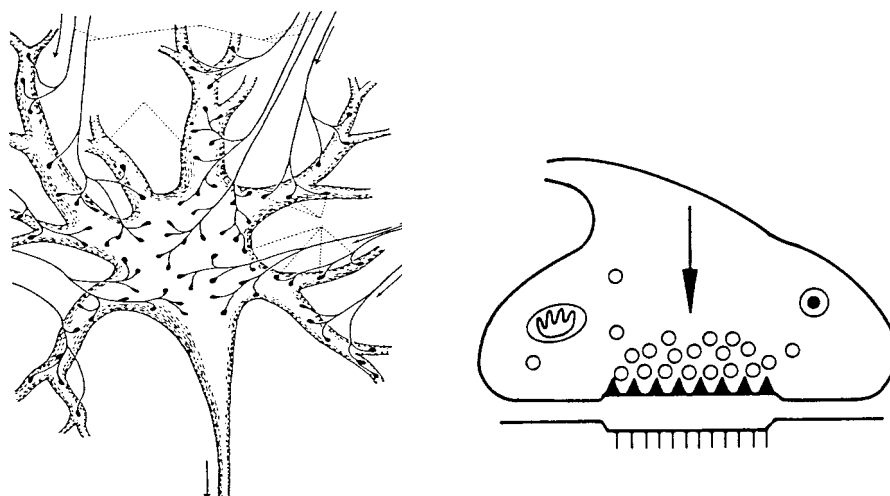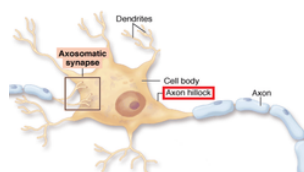
Figure 3: Left: drawing of a neuron. The black blobs attached to the cell body and the dendrites (input channels) represent the synapses (adjustable terminals which determine the effect communicating neurons will have on one another's membrane potential and firing state). Right: close-up of a typical synapse.

decrease. In the first case the arriving signal will increase the probability of the receiving neuron to start firing itself, therefore such a synapse is called excitatory. In the second case the arriving signal will decrease the probability of the receiving neuron being triggered, and the synapse is called inhibitory. However, there is also the possibility that the arriving action potential will not succeed in releasing neurotransmitter; neurons are not perfect. This introduces an element of uncertainty, or noise, into the operation of the machinery.

Whether or not the receiving neuron will actually be triggered into firing itself, will depend on the cumulative effect of all excitatory and inhibitory signals arriving, a detailed analysis of which requires also taking into account the electrical details of the dendrites. The region of the neuron membrane most sensitive to be triggered into sending an action potential is the so-called hillock zone, near the root of the axon. If the potential in this region, the post-synaptic potential, exceeds some neuron-specific threshold (of the order of -30 mV), the neuron will fire an action potential. However, the firing threshold is not a strict constant, but can vary randomly around some average value (so that there will always be some non-zero probability of a neuron not doing what we would expect it to do with a given post-synaptic potential), which constitutes the second main source of uncertainty into the operation.

The key to the adaptive and self-programming properties of neural tissue and to being able to store information, is that the synapses and firing thresholds are not fixed, but are being updated all the time. It is not entirely clear, however, how this is realised at a chemical/electrical level. Most likely the amount of neurotransmitter in a synapse, available for release, and the effective
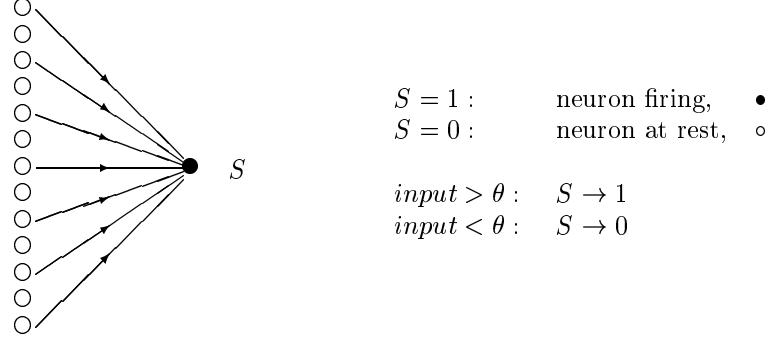
$$S = 1 : \qquad \text{neuron firing}, \quad \bullet$$
$$S = 0 : \qquad \text{neuron at rest}, \quad \circ$$

$$input > \theta : \quad S \to 1$$
$$input < \theta : \quad S \to 0$$

Figure 4: The simplest model neuron: a neuron's firing state is represented by a single instantaneous binary state variable S, whose value is solely determined by whether or not its input exceeds a firing threshold.

contact surface of a synapse are modified.

The simplest caricature of a neuron is one where its possible firing states are reduced to just a single binary variable S, indicating whether it fires ($S = 1$) or is at rest ($S = 0$). See figure 4. Which of the two states the neuron will be in, is dictated by whether or not the total input it receives (i.e. the post-synaptic potential) does ($S \to 1$) or does not ($S \to 0$) exceed the neuron's firing threshold, denoted by θ (if we forget about the noise). As a bonus this allows us to illustrate the collective firing state of networks by colouring the constituent neurons: firing $= \bullet$, rest $= \circ$. We further assume the individual input signals to add up linearly, weighted by the strengths of the associated synapses. The latter are represented by real variables w_ℓ, whose sign denotes the type of interaction ($w_\ell > 0$: excitation, $w_\ell < 0$: inhibition) and whose absolute value $|w_\ell|$ denotes the magnitude of the interaction:

$$input = w_1 S_1 + \ldots + w_N S_N$$

Here the various neurons present are labelled by subscripts $\ell = 1, \ldots, N$. This rule indeed appears to capture the characteristics of neural communication. Imagine, for instance, the effect on the input of a quiescent neuron ℓ suddenly starting to fire:

$$S_\ell \to 1 : \qquad input \to input + w_\ell \qquad \begin{cases} w_\ell > 0 : & input \uparrow, \quad excitation \\ w_\ell < 0 : & input \downarrow, \quad inhibition \end{cases}$$

We now adapt these rules for each of our neurons. We indicate explicitly at which time t (for simplicity to be measured in units of one) the various neuron states are observed, we denote the synaptic strength at a junction $j \to i$ (where

8

j denotes the 'sender' and i the 'receiver') by w_{ij}, and the threshold of a neuron i by θ_i. This brings us to the following set of microscopic operation rules:

$$\begin{aligned} w_{i1}S_1(t) + \ldots + w_{iN}S_N(t) > \theta_i : \quad S_i(t+1) = 1 \\ w_{i1}S_1(t) + \ldots + w_{iN}S_N(t) < \theta_i : \quad S_i(t+1) = 0 \end{aligned} \tag{1}$$

These rules could either be applied to all neurons *at the same time*, giving so-called parallel dynamics, or to one neuron at a time (drawn randomly or according to a fixed order), giving so-called sequential dynamics.[2] Upon specifying the values of the synapses $\{w_{ij}\}$ and the thresholds $\{\theta_i\}$, as well as the initial network state $\{S_i(0)\}$, the system will evolve in time in a deterministic manner, and the operation of our network can be characterised by giving the states $\{S_i(t)\}$ of the N neurons at subsequent times, e.g.

	S_1	S_2	S_3	S_4	S_5	S_6	S_7	S_8	S_9
$t = 0:$	1	1	0	1	0	0	1	0	0
$t = 1:$	1	0	0	1	0	1	1	1	1
$t = 2:$	0	0	1	1	1	0	0	0	1
$t = 3:$	1	0	0	1	1	1	1	0	1
$t = 4:$	0	1	1	1	0	0	1	0	1

or, equivalently, by drawing the neuron states at different times as a collection of coloured circles, according to the convention 'firing' = •, 'rest' = ∘, e.g.

We have thus achieved a reduction of the operation of neural networks to a well-defined manipulation of a set of (binary) numbers, whose rules (1) can be seen as an extremely simplified version of biological reality. The binary numbers represent the states of the information processors (the neurons), and therefore describe the system *operation*. The details of the operation to be be performed depend on a set of control parameters (synapses and thresholds), which must accordingly be interpreted as representing the *program*. Moreover, manipulating numbers brings us into the realm of mathematics; the formulation (1) describes a non-linear discrete-time dynamical system.

2.2 Universality of Model Neurons

Although it is not a priori clear that our equations (1) are not an oversimplification of biological reality, there are at least two reasons for not making things more complicated yet. First of all, solving (1) for arbitrary control parameters and nontrivial system sizes is already impossible, in spite of its apparent simplicity. Secondly, networks of the type (1) are found to be *universal* information

---

[2]Strictly speaking, we also need to specify a rule for determining $S_i(t+1)$ for the marginal case, where $w_{i1}S_1(t) + \ldots + w_{iN}S_N(t) = \theta_i$. Two common ways of dealing with this situation are to either draw $S_i(t+1)$ at random from $\{0,1\}$, or to simply leave $S_i(t+1) = S_i(t)$.

processing systems, in that (roughly speaking) they can perform any computation that can be performed by conventional digital computers, provided one chooses the synapses and thresholds appropriately.
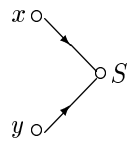
The simplest way to show this is by demonstrating that the basic logical units of digital computers, the operations AND: $(x, y) \to x \wedge y$, OR: $(x, y) \to x \vee y$ and NOT: $x \to \neg x$ (with $x, y \in \{0, 1\}$), can be built with our model neurons. Each logical unit (or 'gate') is defined by a so-called truth table, specifying its output for each possible input. All we need to do is to define for each of the above gates a model neuron of the type

$$w_1 x + w_2 y - \theta > 0 : \quad S = 1$$
$$w_1 x + w_2 y - \theta < 0 : \quad S = 0$$

by choosing appropriate values of the control parameters $\{w_1, w_2, \theta\}$, which has the same truth table. This turns out to be fairly easy:
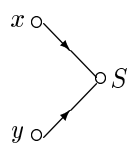
AND:

x	y	$x \wedge y$	$x+y-\frac{3}{2}$	S
0	0	0	$-3/2$	0
0	1	0	$-1/2$	0
1	0	0	$-1/2$	0
1	1	1	$1/2$	1

$w_1 = w_2 = 1$
$\theta = \frac{3}{2}$

OR:
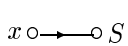
x	y	$x \vee y$	$x+y-\frac{1}{2}$	S
0	0	0	$-1/2$	0
0	1	1	$1/2$	1
1	0	1	$1/2$	1
1	1	1	$3/2$	1

$w_1 = w_2 = 1$
$\theta = \frac{1}{2}$

NOT:

x	$\neg x$	$-x+\frac{1}{2}$	S
0	1	$1/2$	1
1	0	$-1/2$	0

$w_1 = -1$
$\theta = -\frac{1}{2}$

This shows that we need not worry about a-priori restrictions on the types of tasks our simplified model networks (1) can handle.

Furthermore, one can also make statements on the architecture required. Provided we employ model neurons with potentially large numbers of input channels, it turns out that every operation involving binary numbers can in fact be performed with a feed-forward network of at most two layers. Again this is proven by construction. Every binary operation $\{0, 1\}^N \to \{0, 1\}^K$ can be reduced (split-up) into specific sub-operations M, each performing a separation of the input signals $\boldsymbol{x}$ (given by N binary numbers) into two classes:

$$M : \{0, 1\}^N \to \{0, 1\}$$

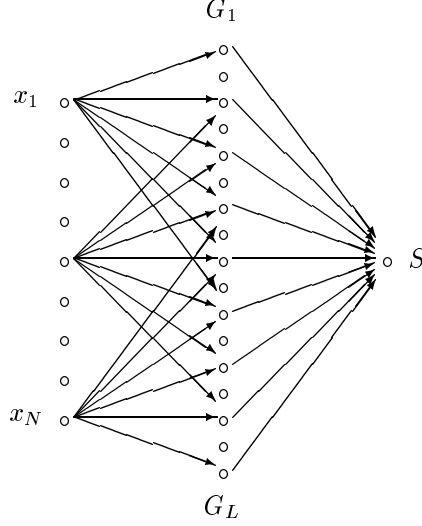Figure 5: Universal architecture, capable of performing any classification $M :$ $\{0,1\}^N \rightarrow \{0,1\}$, provided synapses and thresholds are choosen adequately.

(described by a truth table with 2^N rows). Each such M can be built as a neural realisation of a look-up exercise, where the aim is simply to check whether an $\boldsymbol{x} \in \{0,1\}^N$ is in the set for which $M(\boldsymbol{x}) = 1$. This set is denoted by Ω, with $L \leq 2^N$ elements which we label as follows: $\Omega = \{\boldsymbol{y}_1, \ldots, \boldsymbol{y}_L\}$. The basic tools of our construction are the so-called 'grandmother-neurons'[3] G_ℓ, whose sole task is to be on the look-out for one of the input signals $\boldsymbol{y}_\ell \in \Omega$:

$$w_1 x_1 + \ldots + w_N x_N > \theta : \quad G_\ell = 1$$
$$w_1 x_1 + \ldots + w_N x_N < \theta : \quad G_\ell = 0$$

with $w_\ell = 2(2y_\ell - 1)$ and $\theta = 2(y_1 + \ldots + y_N) - 1$. Inspection shows that with these definitions the output G_ℓ, upon presentation of input $\boldsymbol{x}$, is indeed (as required) given by

$$\boldsymbol{x} = \boldsymbol{y}_\ell : \quad G_\ell = 1$$
$$\boldsymbol{x} \neq \boldsymbol{y}_\ell : \quad G_\ell = 0$$

Finally the outputs of the grandmother neurons are fed into a model neuron S, which is to determine whether or not one of the grandmother neurons is active:

$$y_1 + \ldots + y_L > 1/2 : \quad S = 1$$
$$y_1 + \ldots + y_L < 1/2 : \quad S = 0$$

---

[3]This name was coined to denote neurons which only become active upon presentation of some unique and specific sensory pattern (visual or otherwise), e.g. an image of one's grandmother. Such neurons were at some stage claimed to have been observed experimentally.

The resulting feed-forward network is shown in figure 5. For any input $\boldsymbol{x}$, the number of active neurons G_ℓ in the first layer is either 0 (leading to the final output $S = 0$) or 1 (leading to the final output $S = 1$). In the first case the input vector $\boldsymbol{x}$ is apparently not in the set Ω, in the second case it apparently is. This shows that the network thus constructed performs the separation M.

2.3 Directions and Strategies

Here the field effectively splits in two. One route leading away from equation (1) aims at solving it with respect to the evolution of the neuron states, for increasingly complicated but prescribed choices of synapses and thresholds. Here the key phenomenon is *operation*, the central dynamical variables are the neurons, whereas synapses and thresholds play the role of parameters. The alternative route is to concentrate on the complementary problem: which are the possible modes of operation equation (1) would allow for, if we were to vary synapses and thresholds in a given architecture, and how can one find learning rules (rules for the modification of synapses and thresholds) that will generate values such that the resulting network will meet some specified performance criterion. Here the key phenomenon is *learning*, the central dynamical variables are the synapses and thresholds, whereas neuron states (or, more often, their statistics) induce constraints and operation targets.

Operation		Learning	
variables:	neurons	variables:	synapses, thresholds
parameters:	synapses, thresholds	parameters:	required neuron states

Although quite prominent, in reality this separation is, of course, not perfect; in the field of learning theory one often specifies neuron states only in part of the system, and solves for the remaining neuron states, and there even exist non-trivial but solvable models in which both neurons and synapses/thresholds evolve in time. In the following sections I will describe examples from both main problem classes.

A general rule in dealing with mathematical models, whether they describe phenomena in biology, physics, economics or any other discipline, is that one usually finds that the equations involved are most easily solved in extreme limits for the control parameters. This is also true for neural network models, in particular with respect to the system size N and the spatial distance over which the neurons are allowed to interact. Analysing models with just two or three neurons (on one end of the scale of sizes) is not much of a problem, but realistic systems happen to scale differently, both in biology (where even small brain regions are at least of size $N \sim 10^6$) and in engineering (where at least $N \sim 10^3$). Therefore one usually considers the opposite limit $N \to \infty$. In turn, one can only solve the equations desciribing infinitely large systems when either interactions are restricted to occur only between neighbouring neurons

(which is quite unrealistic), or when a large number (if not all) of the neurons are allowed to interact (which is a better approximation of reality).

The strategy of the model solver is then to identify global observables which characterise the system state at a macroscopic level (this is often the most difficult bit), and to calculate their values. For instance, in statistical mechanics one is not interested in knowing the positions and velocities of individual molecules in a gas, but rather in knowing the values of global obervables like pressure; in modelling (and predicting) exchange rates we do not care about which individuals buy certain amounts of a currency, but rather in the sum over all such buyers. Which macroscopic observables constitute the natural language for describing the operation of neural networks turns out to depend strongly on their function or task (as might have been expected). If the exercise is carried out properly, and if the model at hand is sufficiently friendly, one will observe that in the $N \to \infty$ limit clean and transparent analytical relations emerge. This happens for various reasons. If there is an element of randomness involved (noise) it is clear that in finite systems we can only speak about the probability of certain averages occurring, whereas in the $N \to \infty$ limit one would find averages being replaced by exact expressions. Secondly, as soon as spatial structure of a network is involved, the limit $N \to \infty$ allows us to take continuum limits and to replace discrete systems by continuous ones.

The operation a neural network performs depends on its program: the choice made for architecture, synaptic interactions and thresholds (equivalently, on the learning rule used to generate these parameters). I will now give several examples involving different types of information processing tasks and, consequently, different types of analysis (although all share the reductionist strategy of calculating global properties from underlying microscopic laws).

3 Neural Networks as Associative Memories

Our definition of model neurons has led to a relatively simple scenario, where a global network state is described by specifying for each neuron the value of its associated binary variable. It can be conveniently drawn in a picture with black circles denoting active neurons and white circles denoting neurons at rest. If we choose the neural thresholds such that a disconnected neuron would be precisely critical (with a potential *at* threshold), we can simplify our equations further by choosing $\{-1, 1\}$ as the two neuron states (rest/firing), instead of $\{0, 1\}$ (see below), giving the rules

$$
\begin{aligned}
w_{i1}S_1(t) + \ldots + w_{iN}S_N(t) > 0 : \quad & S_i(t+1) = 1 \\
w_{i1}S_1(t) + \ldots + w_{iN}S_N(t) < 0 : \quad & S_i(t+1) = -1
\end{aligned}
\tag{2}
$$

to be depicted as
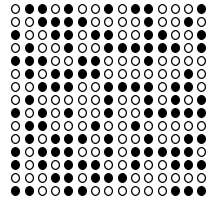
$$
\begin{aligned}
\bullet : \quad & S_i = 1 \quad \text{(neuron } i \text{ firing)} \\
\circ : \quad & S_i = -1 \quad \text{(neuron } i \text{ at rest)} \\
& input_i > 0 : \quad S_i \to 1 \\
& input_i < 0 : \quad S_i \to -1 \\
& input_i = w_{i1}S_1 + \ldots + w_{iN}S_N
\end{aligned}
$$

If this network is to operate as a memory, for storing and retrieving patterns (pictures, words, sounds, etc.), we must assume that the information is (physically) stored in the synapses, and that pattern retrieval must correspond to a dynamical process of neuron states. We are thus led to representing patterns to be stored as global network states, i.e. each pattern corresponds to a specific set of binary numbers $\{S_1, \ldots, S_N\}$, or, equivalently, to a specific way of colouring circles in the picture above. This is similar to what happens in conventional computers. However, in computers one retrieves such information by specifying the label of the pattern in question, which codes for the address of its physical memory location. This will be quite different here. Let us introduce the principles behind the neural way of storaging and retrieving information, by working out the details for a very simple model example.

Biologically realistic learning rules for synapses are required to meet the constraint that the way a given synapse w_{ij} is modified can depend only on information locally available: the electro-chemical state properties of the neurons i and j.[4] One of the simplest such rules is the following: increase w_{ij} if the neurons i and j are in the same state, decrease w_{ij} otherwise. With our definition of the allowed neuron states being $\{-1, 1\}$, this can be written as

$$
\begin{aligned}
S_i = S_j : \quad & w_{ij} \uparrow \\
S_i \neq S_j : \quad & w_{ij} \downarrow
\end{aligned}
\qquad\qquad
w_{ij} \to w_{ij} + S_i S_j
\tag{3}
$$

---

[4] This constraint is to be modified if in addition we wish to take into account the more global effects of modulatory chemicals like hormones and drugs.

If we apply this rule to just one specific pattern, denoted by $\{\xi_1, \ldots, \xi_N\}$ (each component $\xi_i \in \{-1, 1\}$ represents a specific state of a single neuron), we obtain the following recipe for the synapses: $w_{ij} = \xi_i \xi_j$. How would a network with such synapses behave ? Note, firstly, that

$$input_i = w_{i1} S_1 + \ldots + w_{iN} S_N = \xi_i \left[\xi_1 S_1 + \ldots + \xi_N S_N \right]$$

so the dynamical rules (2) for the neurons become

$$
\begin{aligned}
\xi_i \left[\xi_1 S_1(t) + \ldots + \xi_N S_N(t) \right] > 0 : \quad & S_i(t+1) = 1 \\
\xi_i \left[\xi_1 S_1(t) + \ldots + \xi_N S_N(t) \right] < 0 : \quad & S_i(t+1) = -1
\end{aligned}
\tag{4}
$$

Note also that $\xi_i S_i(t) = 1$ if $\xi_i = S_i(t)$, and that $\xi_i S_i(t) = -1$ if $\xi_i \neq S_i(t)$. Therefore, if at time t more than half of the neurons are in the state $S_i(t) = \xi_i$ then $\xi_1 S_1(t) + \ldots + \xi_N S_N(t) > 0$. It subsequently follows from (4) that

$$\text{for all } i : \quad \text{sign}(input_i) = \xi_i \quad \text{so} \quad S_i(t+1) = \xi_i$$

If the dynamics (4) is of the parallel type (all neurons change their state at the same time), this convergence $(S_1, \ldots, S_N) \to (\xi_1, \ldots, \xi_N)$ is completed in a single iteration step. For sequential dynamics (neurons change states one after the other), the convergence is a gradual process. In both cases, the choice $w_{ij} = \xi_i \xi_j$ achieves the following: the state $(S_1, \ldots, S_N) = (\xi_1, \ldots, \xi_N)$ has become a *stable state* of the network dynamics. The network dynamically reconstructs the full pattern $(\xi_1, \ldots, \xi_N)$ if it is prepared in an initial state which bears sufficient resemblance to the state corresponding to this pattern.

3.1 Recipes for Storing Patterns and Pattern Sequences

If the operation described above for the case of a single stored pattern, turns out to carry over to the more general case of an arbitrary number p of patterns, we arrive at the following recipe for information storage and retrieval:

- Represent each patterns as a specific network state $(\xi_1^\mu, \ldots, \xi_N^\mu)$.

- Construct synapses $\{w_{ij}\}$ such that these patterns become *stable states* (fixed-point attractors) for the network dynamics.

- An input to be recognised will serve as the initial state $\{S_i(t = 0)\}$.

- The final state reached $\{S_i(t = \infty)\}$ can be interpreted as the pattern recognised by the network from the input $\{S_i(t = 0)\}$.

From any given initial state, the system will, by construction, evolve towards the 'nearest'[5] stable state, i.e. towards the stored pattern which most closely resembles the initial state (see figure 6). The system performs so-called *associative pattern recall*: patterns are not retrieved from memory by giving an address

---

[5] The distance between two system states $(S_1, \ldots, S_N)$ and $(S'_1, \ldots, S'_N)$ is defined in terms of the number of neurons for which $S_i \neq S'_i$.
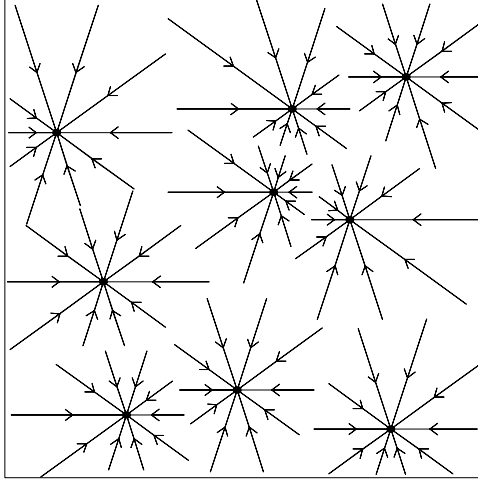
Figure 6: Information storage through the creation of attractors in the space of states. The state vector $(S_1, \ldots, S_N)$ evolves towards the nearest stable state. If the stable states are the patterns stored, and the initial state is an input pattern to be recognised, this system performs *associative* pattern recall.

label (as in computers), but by an association process. By construction, this system will also recognise corrupted or incomplete patterns.

If we apply the learning rule (3) to a collection of p patterns, $(\xi_1^\mu, \ldots, \xi_N^\mu)$, where the superscript $\mu \in \{1, \ldots, p\}$ labels the patterns, we obtain

$$w_{ij} = \xi_i^1 \xi_j^1 + \ldots + \xi_i^p \xi_j^p \tag{5}$$

Due to their simplicity it is easy and entertaining to write a computer program which simulates equations (2) for the choice (5), in order to verify that the recipe described above works. An example is shown in figures 7 and 8. We choose a set of ten patterns, each represented by $N = 841$ binary variables (pixels), see figure 7, and calculate synapses according to (5). For the initial state of the network equipped with these synapses we choose a corrupted version of one of the patterns. Following this initialisation the system is left to itself, and the dynamical rules (2) then generate processes such as those shown in figure 8. If the corruption of the state to be recognised is modest, the system indeed evolves towards (i.e. 'recognises') the desired pattern. Note, however, that the particular recipe (5) en passant creates additional attractors, in the form of mixtures of the p stored patterns, to which the system is found to evolve if started from a completely random initial state (see figure 8). Such 'mixture' states can be removed easily, either by adding noise to the dynamical rules, by introducing a non-zero threshold in the equations (2) or by using more sophisticated learning rules.
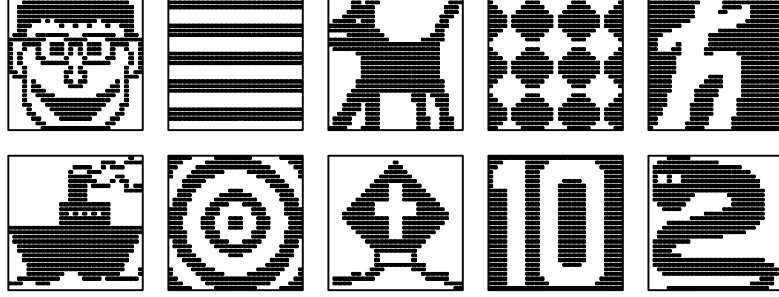
16

Figure 7: Ten patterns represented as specific microscopic states of an $N = 841$ attractor network. Individual pixels represent neuron states: $\{\bullet, \circ\} = \{1, -1\}$.

t=0	t=1	t=2	t=3	t=4
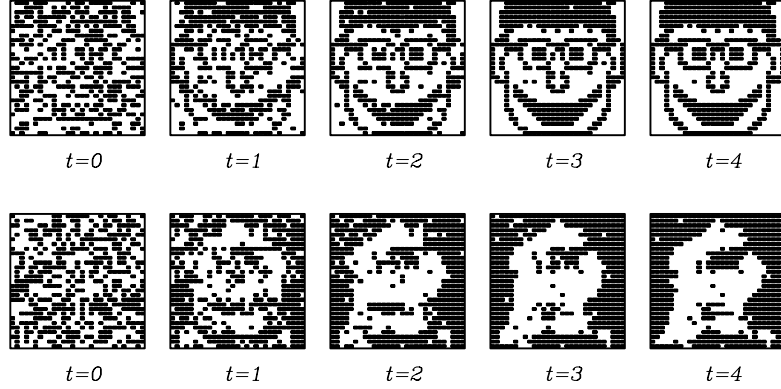
Figure 8: Two simulation examples: snapshots of the microscopic system state $\{S_1, \ldots, S_N\}$ at times $t = 0, 1, 2, 3, 4$ iteration steps per neuron. Dynamics: sequential. Top row: associative recall of a stored pattern from an initial state which is a corrupted version thereof. Bottom row: evolution towards a spurious (mixture) state from a randomly drawn initial state.

It turns out that the game described so far can be generalised to the situation where one wants to store not just individual (static) patterns, but sequences of patterns (films rather than individual pictures, sentences rather than words, or even an arbitrary set of required state transitions). To see this, let us make a small modification in the simple learning rule (3):

$$w_{ij} \to w_{ij} + S_i' S_j \qquad (6)$$

Now *two* microscopic configurations $(S_1, \ldots, S_N)$ and $(S_1', \ldots, S_N')$ play a role. Rules like (6) emerge naturally if one takes transmission delays into account. If we apply (6) to a single pair of specific patterns, $(\xi_1, \ldots, \xi_N)$ and $(\xi_1', \ldots, \xi_N')$,

Figure 9: Information storage through the creation of attractors in the space of states. The state vector $(S_1, \ldots, S_N)$ evolves towards the nearest attractor. If the attractors are the pattern sequences stored, and the initial state is a constituent pattern, this system performs *associative* sequence recall.
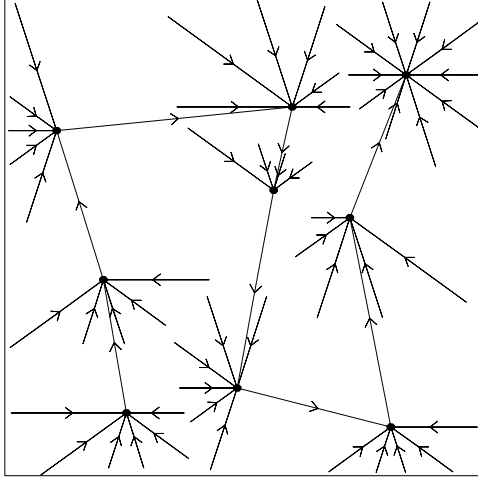
we obtain $w_{ij} = \xi_i' \xi_j$, giving

$$input_i = w_{i1}S_1 + \ldots + w_{iN}S_N = \xi_i' \left[\xi_1 S_1 + \ldots + \xi_N S_N \right]$$

so that the dynamical rules (2) for the neuron states become

$$\begin{aligned} \xi_1 S_1(t) + \ldots + \xi_N S_N(t) > 0: \quad & S_i(t+1) = \xi_i' \\ \xi_1 S_1(t) + \ldots + \xi_N S_N(t) < 0: \quad & S_i(t+1) = -\xi_i' \end{aligned} \tag{7}$$

If at time t more than half of the neurons are in the state $S_i(t) = \xi_i$ then for all i: $S_i(t+1) = \xi_i'$. In other words: if the system is in a state sufficiently 'close' to state $(\xi_1, \ldots, \xi_N)$ it will tend to evolve towards state $(\xi_1', \ldots, \xi_N')$.[6]

This simple example shows several interesting things. Firstly, we can apparently store pattern sequences with the following rule:

- Represent each pattern sequence as a sequence of network state

- Construct synapses $\{w_{ij}\}$ such that these sequences become attractors for the network dynamics.

- An input to trigger a squence will be the initial network state $\{S_i(t=0)\}$.

- The final attractor reached $\{S_i(t)\}$ (large t) can be interpreted as the sequence recalled by presenting input $\{S_i(t=0)\}$.

---

[6] The original recipe (3) just corresponds to the special case $(\xi_1, \ldots, \xi_N) = (\xi_1', \ldots, \xi_N')$.

For example, we can achieve the storage of a given sequence of p states by applying the rule (7) to each of the individual constituent state transitions $(\xi_1^\mu, \ldots, \xi_N^\mu) \to (\xi_1^{\mu+1}, \ldots, \xi_N^{\mu+1})$ that we want to build in, giving the recipe

$$w_{ij} = \xi_i^2 \xi_j^1 + \ldots + \xi_i^p \xi_j^{p-1} \tag{8}$$

It turns out that (8) indeed leads to the required operation described above, provided that the dynamics is of the parallel type. If the neurons change their states sequentially, an additional mechanism is found to be needed to stabilise the sequences (such as delayed interactions between the neurons).

Secondly, at least for parallel dynamics we now see how one might 'teach' these networks any arbitrary set of instructions, since it appears that the following interpretation holds:

$$
\begin{aligned}
&\textbf{synaptic change}: \quad w_{ij} \to w_{ij} + \xi_i' \xi_j \\
&\textbf{rule learned}: \qquad \text{if in state } (\xi_1, \ldots, \xi_N) \text{ go to state } (\xi_1', \ldots, \xi_N')
\end{aligned}
\tag{9}
$$

The resulting synapses are the sum over all individual stored instructions (9). Note the invariance of the synaptic change under $(\xi_i, \xi_i') \to (-\xi_i, -\xi_i')$ for all i. Although thinking in terms of instruction sets is reminiscent of conventional computers, the way these instructions are executed and combined is different. Let me just note a few points, some of which are immediately obvious, some of which require some more analysis which I will not discuss here:

- The procedure works best for orthogonal or random patterns.

- The microscopic realisation of the patterns is irrelevant. They define the language in terms of which instructions are written; if the 'words' of the language are sufficiently different from one another, any language will do.

- The system still operates by association: if it finds itself in a state not identical to any of the patterns in the instruction set, it will do the operation(s) corresponding to the pattern(s) it resembles most.

- Contradictory instructions just annihilate one another: $\xi_i' \xi_j + (-\xi_i') \xi_j = 0$.

3.2 Symmetric Networks: the Energy Picture

The simple learning rule (3) for storing (static) patterns will give rise to symmetric synapses, i.e. $w_{ij} = w_{ji}$ for all (ij), whereas the more general presciption (6) for storing attractors which are not fixed-points will generate predominantly non-symmetric synapses. It turns out that there is a deeper reason for this difference. For symmetric networks without self-interactions, i.e. $w_{ij} = w_{ji}$ for

$$S_i \; \circ\!\!\underset{\longrightarrow}{\overset{\longleftarrow}{=\!=\!=}}\!\!\circ \; S_j$$

19

all (ij) and $w_{ii} = 0$ for all i, one can easily show that the evolution of the neuron states is such that a certain quantity (termed the 'energy') is always decreasing. For sequential dynamics this energy is found to be

$$E = -\frac{1}{2}\left[S_1.input_1 + \ldots + S_N.input_N\right] \tag{10}$$

For parallel dynamics one finds a different but related quantity.[7] Since the energy (10) is bounded from below, and since with each state change the energy decreases by at least some minimum amount (which depends on the values of the synapses), this process will have to stop at some point. We conclude: whatever the synapses (provided they are symmetric), the state dynamics will always end up in a fixed-point. The converse statement is not true: although in most cases this will not happen, the states of non-symmetric networks could also evolve towards a fixed-point (depending on the details of the synapses).

During the march for the lowest energy state, each individual transition $(S_1, \ldots, S_N) \rightarrow (S'_1, \ldots, S'_N)$ must decrease the energy, i.e. $E' < E$, which implies that one need not end up in the state with the lowest energy. Just imagine a downhill walk in a hilly landscape; in order to arrive at the lowest point one will occasionally have to cross a ridge to go from one valley to another. The network cannot do this, and can consequently end up in a local minimum of E, different from the global one.

Although quite natural in the context of neural networks, having asymmetry in the interactions of pairs of elements is in fact for the modeller an unusual situation. In physics the equivalent would be, for instance, a pair of two molecules A and B, with A exerting an attractive force on B and at the same time B repelling A. Or, likewise, a pair of magnets A and B such that magnet A prefers to have its poles (north and south) opposite to the poles of B, whereas B is keen on configurations where similar poles point in the same direction. If we add noise to symmetric systems we find that interaction symmetry implies *detailed balance*, which guarantees an evolution towards equilibrium. Since this is what all physical systems do, most analytical techniques developed to study interaction particle systems are based on this property. This makes neural networks the more interesting: since they are mostly non-symmetric, they will in general not evolve to equilibrium (compared to the possible modes of operation of non-symmetric networks, the symmetric ones are in fact quite boring), and they require novel intuition and techniques for analysis.

3.3 Solving Models of Noisy Attractor Networks

So far we have been concerned with qualitative properties of attractor networks. Let us turn to analysis now, and show how one proceeds to solve such models. I will skip details and only discuss the solution for sequential dynamics (for parallel dynamics one proceeds in a similar way). I will also introduce noise into the dynamics; this simplifies many calculations and often turns out to be beneficial to the operation of the system.

---

[7] For parallel dynamics the condition that self-interactions must be absent can be dropped.

Stage 1: define the dynamical rules

The simplest way to add noise to the dynamics is to add to the each of the neural inputs at each time-step t an independent zero-average random number $z_i(t)$. This changes the noise-free dynamical laws (2) to

$$
\begin{aligned}
w_{i1}S_1(t) + \ldots + w_{iN}S_N(t) + Tz_i(t) > 0 : & \quad S_i(t+1) = 1 \\
w_{i1}S_1(t) + \ldots + w_{iN}S_N(t) + Tz_i(t) < 0 : & \quad S_i(t+1) = -1
\end{aligned}
\tag{11}
$$

Here T is an overall parameter to control the amount of noise ($T = 0$: no noise, $T = \infty$: noise only). We store various operations of the type (9), defined using a set of p patterns:

$$
w_{ij} = \frac{1}{N} \sum_{\mu,\nu=1}^{p} A_{\mu\nu} \xi_i^\mu \xi_j^\nu
\qquad
\overbrace{(\xi_1^1, \ldots, \xi_N^1)}^{\text{pattern 1}} \ldots \overbrace{(\xi_1^p, \ldots, \xi_N^p)}^{\text{pattern p}}
\tag{12}
$$

The prefactor $\frac{1}{N}$ is inserted to ensure that the inputs will not diverge in the limit $N \to \infty$ which we will eventually take. The associative memory rule (5) corresponds to $A_{\mu\nu} = \delta_{\mu\nu}$ (i.e. $A_{\mu\mu} = 1$ for all μ and $A_{\mu\nu} = 0$ for $\mu \neq \nu$).

Stage 2: rewrite dynamical rules in terms of probabilities

To suppress notation I will abbreviate $\boldsymbol{S} = (S_1, \ldots, S_N)$. Due to the noise we can only speak about the *probability* $p_t(\boldsymbol{S})$ to find a given state $\boldsymbol{S}$ at a given time t. In order to arrive at a description where time is a continuous variable, we choose the individual durations of the update steps at random from a Poisson distribution[8], with an average step duration of $\frac{1}{N}$, i.e. the probability $\pi_\ell(t)$ that at time t exactly ℓ neurons states have been updated is defined as

$$
\pi_\ell(t) = \frac{1}{\ell!}(Nt)^\ell e^{-Nt}
$$

In one unit of time each neuron will on average have had one state update (as in the simulations of figure 8). What remains is to do the bookkeeping of the possible sequential transitions properly, which results in an equation for the rate of change of the microscopic probability distribution:

$$
\frac{d}{dt}p_t(\boldsymbol{S}) = \sum_{i=1}^{N} \left\{ w_i(F_i\boldsymbol{S})p_t(F_i\boldsymbol{S}) - w_i(\boldsymbol{S})p_t(\boldsymbol{S}) \right\}
\tag{13}
$$

Here $w_i(\boldsymbol{S})$ denotes the rate at which the transition $\boldsymbol{S} \to F_i\boldsymbol{S}$ occurs if the system is in state $\boldsymbol{S}$, and F_i is the operation 'change the state of neuron i: $F_i\boldsymbol{S} = (S_1, \ldots, S_{i-1}, -S_i, S_{i+1}, \ldots, S_N)$'. Equation (13) is quite transparent: the probability to find the state $\boldsymbol{S}$ increases due to transitions of the type $F_i\boldsymbol{S} \to \boldsymbol{S}$, and decreases due to transitions of the type $\boldsymbol{S} \to F_i\boldsymbol{S}$. The values

---

[8]This particular choice turns out to generate the simplest equations later.

of the transition rates $w_i(\boldsymbol{S})$ depend on the choice made for the distribution $P(z)$ of the noise variables $z_i(t)$. One convenient choice is

$$P(z) = \frac{1}{2}[1 - \tanh^2(z)]: \qquad w_i(\boldsymbol{S}) = \frac{1}{2}\left[1 - \tanh[S_i \sum_{j=1}^{N} w_{ij}S_j/T]\right] \quad (14)$$

We have now translated our problem into solving a well-defined linear differential equation (13). Unfortunately this is still too difficult (except for networks obtained by making trivial choices for the synapses $\{w_{ij}\}$ or the noise level T).

Stage 3: find the relevant macroscopic features

We now try to find out which are the key macroscopic quantities (if any) that characterise the dynamical process. Unfortunately there is no general method to do this; one must rely on intuition, experience and common sense. Combining equations (11,12) shows that the neural inputs depend on the instantaneous state $\boldsymbol{S}$ only through the values of p specific macroscopic quantities $m_\mu(\boldsymbol{S})$:

$$input_i = \sum_{\mu\nu=1}^{p} A_{\mu\nu}\xi_i^\mu m_\nu(\boldsymbol{S}) + T z_i$$

$$m_\nu(\boldsymbol{S}) = \frac{1}{N}\left[S_1\xi_1^\nu + \ldots + S_N\xi_N^\nu\right] \qquad (15)$$

Note that these so-called 'overlaps' $m_\nu(\boldsymbol{S})$ measure the similarity between the state $\boldsymbol{S}$ and the stored patterns,[9] e.g.:

$$m_\mu(\boldsymbol{S}) = 1: \qquad (S_1, \ldots, S_N) = (\xi_1^\mu, \ldots, \xi_N^\mu)$$
$$m_\mu(\boldsymbol{S}) = -1: \quad (S_1, \ldots, S_N) = (-\xi_1^\mu, \ldots, -\xi_N^\mu)$$

Further evidence for their status as our macroscopic level of description is provided by measuring their values during simulations, see e.g. figure 10. Since a description in terms of the observables $\{m_1(\boldsymbol{S}), \ldots, m_p(\boldsymbol{S})\}$ will only be simpler than the microscopic one in terms of $(S_1, \ldots, S_N)$ for modest numbers of patterns, i.e. for $p \ll N$, we will assume p to be finite (if $p \sim N$ we will simply have to think of something else). Having identified our macroscopic description, we can now define the macroscopic equivalent $P_t(m_1, \ldots, m_p)$ of the microscopic probability distribution $p_t(\boldsymbol{S})$, and calculate the macroscopic equivalent of the differential equation (13), which for $N \to \infty$ reduces to

$$\frac{d}{dt}P_t(m_1, \ldots, m_p) = -\sum_{\mu=1}^{p} \frac{\partial}{\partial m_\mu}\{P_t(m_1, \ldots, m_p)F_\mu(m_1, \ldots, m_p)\} \qquad (16)$$

$$F_\mu(m_1, \ldots, m_p) = \sum_{\boldsymbol{\xi} \in \{-1,1\}^p} p(\boldsymbol{\xi})\ \xi_\mu \tanh[\sum_{\lambda,\nu=1}^{p} A_{\lambda\nu}\xi_\lambda m_\nu/T] - m_\mu \qquad (17)$$

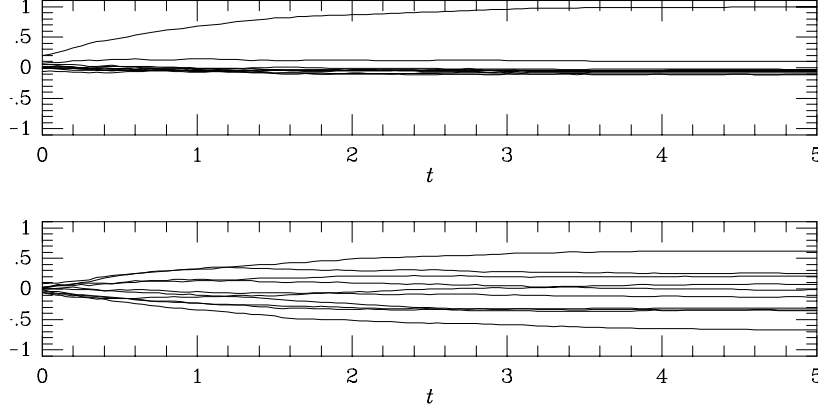[9] They are linearly related to the distance between the system state and the stored patterns.

Figure 10: The two simulation examples of figure 8: here we show the values of the p pattern overlaps $m_\mu(\boldsymbol{S})$, as measured at times $t = 0, 1, 2, 3, 4$ iteration steps per neuron. Top row: associative recall of a stored pattern from an initial state which is a corrupted version thereof. Bottom row: evolution towards a spurious (mixture) state from a randomly drawn initial state.

$$p(\boldsymbol{\xi}) = \lim_{N \to \infty} \frac{1}{N} \sum_{i=1}^{N} \delta_{\xi_i^1, \xi_1} \cdots \delta_{\xi_i^p, \xi_p} \tag{18}$$

Here $\boldsymbol{\xi} = (\xi_1, \ldots, \xi_p)$. For $N \to \infty$ the microsopic details of the pattern components are irrelevant; only the probability distribution (18) plays a role. For randomly drawn patterns one finds $p(\boldsymbol{\xi}) = 2^{-p}$ for all $\boldsymbol{\xi}$. Note that equation (16) is closed, i.e. the evolution of $P_t(m_1, \ldots, m_p)$ is given by a law in which knowledge of the microscopic realisations $\boldsymbol{S}$ or their distribution $p_t(\boldsymbol{S})$ is not required. The level of description of the overlaps is found to be autonomous.

Stage 4: solve the equation for $P_t(m_1, \ldots, m_p)$

The partial differential equation (16) has deterministic solutions: infinitely sharp probability distributions, which depend on time only through the location of the peak. In other words: in the limit $N \to \infty$ the fluctuations in the values of $(m_1, \ldots, m_p)$ become negligable, so that we can forget about probabilities and speak about the actual value of the macroscopic state $(m_1, \ldots, m_p)$. This value evolves in time according to the p coupled non-linear differential equations

$$\frac{d}{dt} \begin{pmatrix} m_1 \\ \vdots \\ m_p \end{pmatrix} = \begin{pmatrix} F_1(m_1, \ldots, m_p) \\ \vdots \\ F_p(m_1, \ldots, m_p) \end{pmatrix} \tag{19}$$

with the functions F_μ given by (17). This is our final solution. One can now analyse these equations, calculate stationary states and their stability properties (if any), sizes of attraction domains, relaxation times, etc.

23
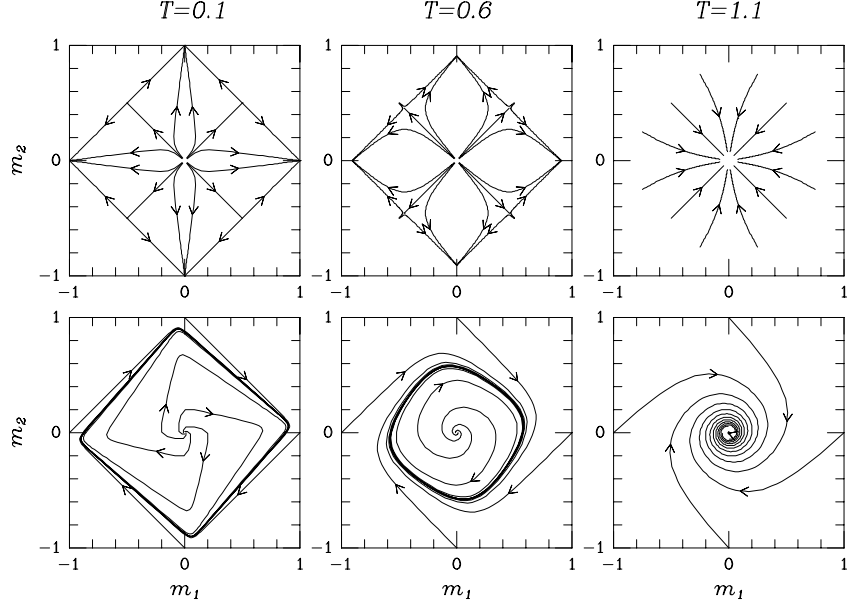
Figure 11: Solutions of the coupled equations (19) for the overlaps, with $p = 2$, obtained numerically and drawn as trajectories in the (m_1, m_2) plane. Row one: $A_{\mu\nu} = \delta_{\mu\nu}$, associative memory. Each of the four stable macroscopic states found for sufficiently low noise levels ($T < 1$) corresponds to the reconstruction of either a stored pattern $(\xi_1^\mu, \ldots, \xi_N^\mu)$ or its negative $(-\xi_1^\mu, \ldots, -\xi_N^\mu)$. Row two: $\boldsymbol{A} = \left(\begin{smallmatrix} 1 & 1 \\ -1 & 1 \end{smallmatrix} \right)$. For sufficiently low noise levels T this choice gives rise to the creation of a limit-cycle of the type $\boldsymbol{\xi}^1 \to (-\boldsymbol{\xi}^2) \to (-\boldsymbol{\xi}^1) \to \boldsymbol{\xi}^2 \to \boldsymbol{\xi}^1 \to \ldots$.
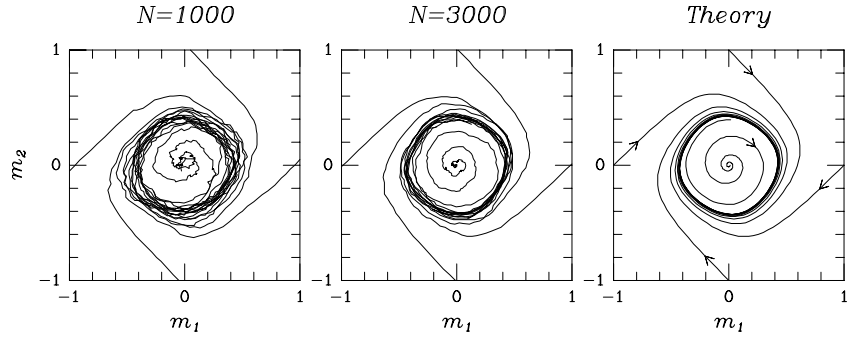
Figure 12: Comparison of the macroscopic dynamics in the (m_1, m_2) plane, as observed in finite-size numerical simulations, and the predictions of the $N = \infty$ theory, for the limit-cycle model with $\boldsymbol{A} = \left(\begin{smallmatrix} 1 & 1 \\ -1 & 1 \end{smallmatrix} \right)$ at noise level $T = 0.8$.

Equivalently we can solve the equations numerically, resulting in figures like 11 and 12. The first row of figure 11 corresponds to $A_{\mu\nu} = \delta_{\mu\nu}$ and $p = 2$, representing a simple associative memory network of the type (5) with two stored patterns. The second row corresponds to a non-symmetric synaptic matrix, with $p = 2$, generating limit-cycle attractors. Finally, figure 12 illustrates how the behaviour of finite networks (as observed in numerical simulations) for increasing values of the network size N approaches that described by the $N = \infty$ theory (described by the numerical solutions of (19)).

4　Creating Maps of the Outside World

Any flexible and robust autonomous system (whether living or robotic) will
have to be able to create, or at least update, an internal 'map' or represen-
tation of its environment. Information on its environment, however, is usu-
ally obtained in an indirect manner, through a redundant set of sensors which
each provide only partial and indirect information. The system responsible for
forming this map needs to be adaptive, as both environment and sensors can
change their characteristics during the system's life-time. Our brain performs
recalibration of sensors all the time; e.g. simply because we grow will the neu-
ronal information about limb positions (generated by sensors which measure
the stretch of muscles) have to be reinterpreted continually. Anatomic changes,
and even learning new skills (like playing an instrument), are found to induce
modifications of internal maps. At a more abstract level, one is confronted with
a complicated non-linear mapping from a relatively low-dimensional and flat
space (the 'physical world') into a high-dimensional one (the space of sensory
signals), and the aim is to find the inverse of this operation. The key to achiev-
ing this is to exploit continuity and correlations in sensory signals, assuming
similar sensory signals to represent similar positions in the environment, which
therefore must correspond to similar positions in the internal map.

4.1　Map Formation Through Competitive Learning

Let us give a simple example. Image a system operating in a simple two-
dimensional world, where positions are represented by two Cartesian coordi-
nates (x, y), observed by sensors and fed into a neural network as input signals.
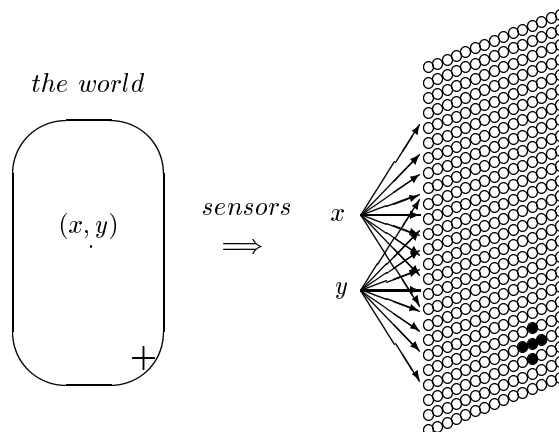

Each neuron i receives information on the input signals (x, y) in the usual way,
through modifiable synaptic interaction strengths: $input_i = w_{ix}x + w_{iy}y$. If
this network is to become an internal coordinate system, faithfully reflecting the
events (x, y) observed in the outside world (in the present example its topology

must accordingly be that of a two-dimensional array), the following objectives are to be met

1. each neuron S_ℓ is more or less 'tuned' to a specific type of signal (x_ℓ, y_ℓ)

2. neighbouring neurons are tuned to similar signals

3. external 'distance' is monotonically related to internal 'distance'

Here the internal 'distance' between two signals (x_A, y_A) and (x_B, y_B) is defined as the physical distance between the two (groups of) neurons that would respond to these two signals.
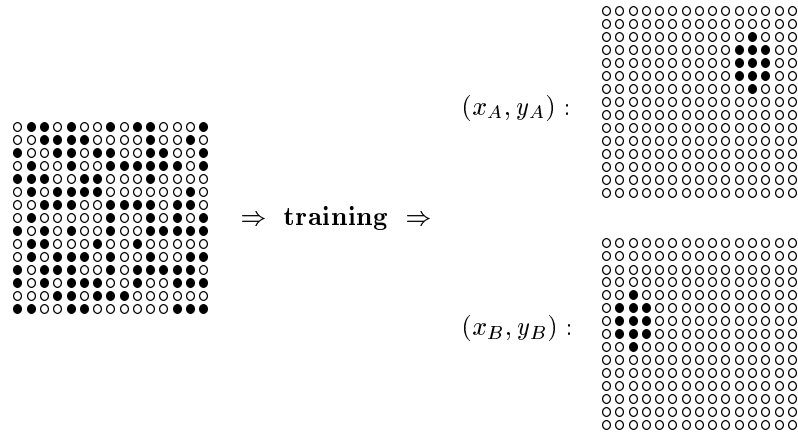
It turns out that in order to achieve these objectives one needs learning rules where neurons effectively enter a competition for having signals 'allocated' to them, whereby neighbouring neurons stimulate one another to develop similar synaptic interactions and distant neurons are prevented from developing similar interactions. Let us try to construct the simplest such learning rule. Since our equations take their simplest form in the case where the input signals are normalised, we define $(x, y) \in [-1, 1]^2$ and add a dummy variable $z = \pm\sqrt{1 - x^2 - y^2}$ (together with an associated synaptic interaction w_z), so

$\bullet: \quad S_i = 1 \quad$ (neuron i firing)
$\circ: \quad S_i = -1 \quad$ (neuron i at rest)

$input_i > 0: \quad S_i \to 1$
$input_i < 0: \quad S_i \to -1$

$input_i = w_{ix}x + w_{iy}y + w_{iz}z$

A learning rule with the desired effect is, starting from random synaptic interaction strengths, to iterate the following recipe until a (more or less) stable

situation is reached:

$$\begin{aligned}
&\text{choose an input signal}: && (x, y, z) \\[4pt]
&\text{find most excited neuron}: && i, \;\; input_i \geq input_k \;\; \text{(for all } k) \\[4pt]
&\text{for } i \text{ and its neighbours}: && \left\{ \begin{array}{lcl} w_{ix} & \rightarrow & (1-\epsilon)w_{ix} + \epsilon x \\ w_{iy} & \rightarrow & (1-\epsilon)w_{iy} + \epsilon y \\ w_{iz} & \rightarrow & (1-\epsilon)w_{iz} + \epsilon z \end{array} \right. \\[4pt]
&\text{for all others}: && \left\{ \begin{array}{lcl} w_{ix} & \rightarrow & (1-\epsilon)w_{ix} - \epsilon x \\ w_{iy} & \rightarrow & (1-\epsilon)w_{iy} - \epsilon y \\ w_{iz} & \rightarrow & (1-\epsilon)w_{iz} - \epsilon z \end{array} \right.
\end{aligned} \tag{20}$$

In words: the neuron that was already the one most responsive to the signal (x, y, z) will be made even more so (together with its neighbours). The other neurons are made less responsive to (x, y, z). This is more obvious if we inspect the effect of the above learning rule on the actual neural inputs, using the built-in property $x^2 + y^2 + z^2 = 1$:

$$\begin{aligned}
&\text{for } i \text{ and its neighbours}: && input_i \rightarrow (1-\epsilon)input_i + \epsilon \\[4pt]
&\text{for all others}: && input_i \rightarrow (1-\epsilon)input_i - \epsilon
\end{aligned}$$

In practice one often adds extra ingredients to this basic recipe, like explicit normalisation of synaptic interaction strengths to deal with non-uniform distributions of input signals (x, y, z), or a monotonically decreasing modification step size $\epsilon(t)$ to enforce and speed up convergence.

A nice way to illustrate what happens during the learning stage is based on exploiting the property that, apart from normalisation, one can interpret the synaptic strengths (w_{ix}, w_{iy}, w_{iz}) of a neuron as the signal (x, y, z) to which it is tuned. We can now draw each set of synaptic strengths (w_{ix}, w_{iy}, w_{iz}) as a point in space, and connect the points corresponding to neurons which are neighbours in the network. We end up with a graphical representation of the synaptic structure of a network in the form of a 'fishing net', with the positions of the knots representing the signals in the world to which the neurons are tuned and with the cords indicating neighbourship, see figure 13. The three objectives of map formation set out at the beginning of this section thereby translate into

1. all knots in the net are separated

2. all cords are similarly stretched

3. there are no regions with overlapping pieces of net

In figure 13 all knots are more or less on the surface of the unit sphere, i.e. $w_{ix}^2 + w_{iy}^2 + w_{iz}^2 \approx 1$ for all i. This reflects the property that the length of the input vector (x, y, z) contains no information, due to $x^2 + y^2 + z^2 = 1$.
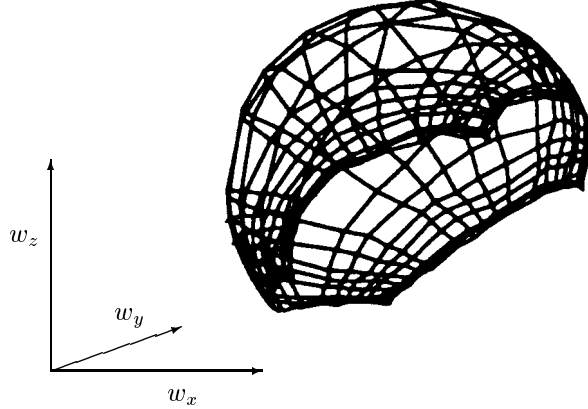
Figure 13: Graphical representation of the synaptic structure of a map forming network in the form of a 'fishing net'. The positions of the knots represent the signals in the world to which the neurons are 'tuned' and the cords connect the knots of neighbouring neurons.

4.2 Solving Models of Map Formation

Let us now try to describe such learning processes analytically. The specific learning rules I will discuss here serve to illustrate only; they are by no means the most sophisticated or efficient ones, but they are sufficiently simple and transparent to allow for understanding and analysis. In addition they provide a nice example of how similarities between mathematical problems in remote scientific areas can be exploited, as will become clear shortly.

One computationally nasty (and biologically unrealistic) feature of the learning rule described above is the need to find the neuron that is triggered most by a particular input signal (x, y, z) (to be given a special status, together with its neighbours). A more realistic but similar procedure is to base the decision about how synapses are to be modified only on the actual firing state of the neurons, and to realise the *neighbours-must-team-up* effect by a spatial smoothening of all neural inputs[10]. To be specific: before synaptic strengths are modified we replace

$$input_i \rightarrow Input_i = \langle input_j \rangle_{\text{near } i} - J \langle input \rangle_{\text{all}} \tag{21}$$

in which brackets denote taking the average over a group of neurons and J is a positive constant. This procedure has the combined effects that (i) neighbouring neurons will tend to have similar neural inputs (due to the first term in (21)), and (ii) the presence of a significant response somewhere in the network will evoke a global suppression of activity everywhere else, so that neurons are

---

[10] In certain brain regions spatial smoothening is indeed known to take place, via diffusing chemicals and gases such as NO

effectively encouraged to 'tune' to different signals (due to the second term in equation (21)).

Stage 1: define the dynamical rules

Thus we arrive at the following recipe for the modification of synaptic strengths, to replace (20):

$$
\begin{aligned}
&\text{choose an input signal}: \quad (x, y, z) \\
&\text{smooth out all inputs}: \quad input_i \to Input_i \\
&\qquad\qquad\qquad\qquad\qquad Input_i = \langle input_j \rangle_{\text{near } i} - J \langle input \rangle_{\text{all}}
\end{aligned}
$$

$$
\text{for all } i \text{ with } S_i = 1: \quad
\left\{
\begin{array}{rcl}
w_{ix} & \to & (1-\epsilon)w_{ix} + \epsilon x \\
w_{iy} & \to & (1-\epsilon)w_{iy} + \epsilon y \\
w_{iz} & \to & (1-\epsilon)w_{iz} + \epsilon z
\end{array}
\right.
\tag{22}
$$

$$
\text{for all } i \text{ with } S_i = -1: \quad
\left\{
\begin{array}{rcl}
w_{ix} & \to & (1-\epsilon)w_{ix} - \epsilon x \\
w_{iy} & \to & (1-\epsilon)w_{iy} - \epsilon y \\
w_{iz} & \to & (1-\epsilon)w_{iz} - \epsilon z
\end{array}
\right.
$$

As before, the 'world' from which the input signals (x, y, z) are drawn is the surface of a sphere: $x^2 + y^2 + z^2 = C^2$.

Stage 2: consider small modifications $\epsilon \to 0$

The dynamical rules (22) define a stochastic process, in that at each time-step the actual synaptic modification depends on the (random) choice made for the input (x, y, z) at that particular instance. However, in the limit of infinitesimally small modification size ϵ one finds the procedure (22) being transformed into a deterministic differential equation (if we also choose ϵ as the duration of each modification step), which involves only *averages* over the distribution $p(x, y, z)$ of inputs signals:

$$
\begin{aligned}
\tfrac{d}{dt} w_{ix} &= \int dx\,dy\,dz \; p(x, y, z) \; x \; \text{sgn}[Input_i(x, y, z)] - w_{ix} \\
\tfrac{d}{dt} w_{iy} &= \int dx\,dy\,dz \; p(x, y, z) \; y \; \text{sgn}[Input_i(x, y, z)] - w_{iy} \\
\tfrac{d}{dt} w_{iz} &= \int dx\,dy\,dz \; p(x, y, z) \; z \; \text{sgn}[Input_i(x, y, z)] - w_{iz}
\end{aligned}
\tag{23}
$$

in which the spatially smoothed out neural inputs $Input_i(x, y, z)$ are given by (21), and the function sgn[..] gives the sign of its argument (i.e. $\text{sgn}[u > 0] = 1$, $\text{sgn}[u < 0] = -1$). The spherical symmetry of the distribution $p(x, y, z)$ allows us to do the integrations in (23). The result of the integrations involves only the smoothed out synaptic weights $\{W_{ix}, W_{iy}, W_{iz}\}$, defined as

$$
\begin{aligned}
W_{ix} &= \langle w_{jx} \rangle_{\text{near } i} - J \langle w_x \rangle_{\text{all}} \\
W_{iy} &= \langle w_{jy} \rangle_{\text{near } i} - J \langle w_y \rangle_{\text{all}} \\
W_{iz} &= \langle w_{jz} \rangle_{\text{near } i} - J \langle w_z \rangle_{\text{all}}
\end{aligned}
\tag{24}
$$

and takes the form:

$$\frac{d}{dt}w_{ix} = \frac{1}{2}C\frac{W_{ix}}{\sqrt{W_{ix}^2 + W_{iy}^2 + W_{iz}^2}} - w_{ix}$$

$$\frac{d}{dt}w_{iy} = \frac{1}{2}C\frac{W_{iy}}{\sqrt{W_{ix}^2 + W_{iy}^2 + W_{iz}^2}} - w_{iy} \qquad (25)$$

$$\frac{d}{dt}w_{iz} = \frac{1}{2}C\frac{W_{iz}}{\sqrt{W_{ix}^2 + W_{iy}^2 + W_{iz}^2}} - w_{iz}$$

Stage 3: exploit equivalence with dynamics of magnetic systems

If the constant J in (24) (controlling the global competition between the neurons) is below some critical value J_c, one can show that the equations (25), with the smoothed out weights (24), evolve towards a stationary state. In stationary states, where $\frac{d}{dt}w_{ix} = \frac{d}{dt}w_{iy} = \frac{d}{dt}w_{iz} = 0$, all synaptic strengths will be normalised according to $w_{ix}^2 + w_{iy}^2 + w_{iz}^2 = \frac{1}{4}C^2$, according to (25). In terms of the graphical representation of figure 13 this corresponds to the statement that in stationary states all knots must lie on the surface of a sphere. From now on we take $C = 2$, leading to stationary synaptic strengths on the surface of the unit sphere.

If one works out the details of the dynamical rules (25) for synaptic strengths which are normalised according to $w_{ix}^2 + w_{iy}^2 + w_{iz}^2 = 1$, one observes that they are suspiciously similar to the ones that describe a system of microscopic magnets, which interact in such a way that neighbouring magnets prefer to point in the same direction (NN and SS), whereas distant magnets prefer to point in opposite directions (NS and SN).

synapses to neuron i :	(w_{ix}, w_{iy}, w_{iz})
neighbouring neurons :	prefer similar synapses
distant neurons :	prefer different synapses
orientation of magnet i :	(w_{ix}, w_{iy}, w_{iz})
neighbouring magnets :	prefer $\uparrow\uparrow$, $\downarrow\downarrow$
distant magnets :	prefer $\uparrow\downarrow$, $\downarrow\uparrow$

This relation suggests that one can use physical concepts again. More specifically, such magnetic systems would evolve towards the minimum of their energy E, in the present language given by

$$E = -\frac{1}{2}\left[w_{1x}W_{1x} + w_{1y}W_{1y} + w_{1z}W_{1z}\right] - \frac{1}{2}\left[w_{2x}W_{2x} + w_{2y}W_{2y} + w_{2z}W_{2z}\right]$$

$$\ldots \qquad -\frac{1}{2}\left[w_{Nx}W_{Nx} + w_{Ny}W_{Ny}\right] \qquad (26)$$

If we check this property for our equations (25), we indeed find that, provided $J < J_c$, from some stage onwards during the evolution towards the stationary state the energy (26) will be decreasing monotonically. The situation thus becomes quite similar to the one with the dynamics of the attractor neural networks in a previous section, in that the dynamical process can ultimately be seen as a quest for a state with minimal energy. We now know that the *equilibrium state* of our map forming system is defined as the configuration of weights that satisfies:

$$I: \quad w_{ix}^2 + w_{iy}^2 + w_{iz}^2 = 1 \quad \text{for all } i$$
$$II: \quad E \text{ is minimal} \tag{27}$$

with E given by (26). We now forget about the more complicated dynamic equations (25) and concentrate on solving (27).

Stage 4: switch to new coordinates, and take the limit $N \to \infty$

Our next step is to implement the conditions $w_{ix}^2 + w_{iy}^2 + w_{iz}^2 = 1$ (for all i) by writing for each neuron i the three synaptic strengths (w_{ix}, w_{iy}, w_{iz}) in terms of the two polar coordinates (θ_i, ϕ_i) (a natural step in the light of the representation of figure 13):

$$w_{ix} = \cos\phi_i \sin\theta_i \qquad w_{iy} = \sin\phi_i \sin\theta_i \qquad w_{iz} = \cos\theta_i \tag{28}$$

Furthermore, for large systems we can replace the discrete neuron labels i by their position coordinates (x_1, x_2) in the network, i.e. $i \to (x_1, x_2)$, so that

$$\theta_i \to \theta(x_1, x_2) \qquad \phi_i \to \phi(x_1, x_2)$$

In doing so we have to specify how in the limit $N \to \infty$ the average over neighbours in (21) is to be carried out. If one just expands any well-behaved local and normalised averaging distribution up to first non-trivial order in its width ϵ, and chooses the parameter $J > J_c$ (the critical value depends on ϵ), one finds, after some non-trivial bookkeeping, that the solution of (27) obeys the following coupled non-linear partial differential equations:

$$\sin\theta \left[\frac{\partial^2 \phi}{\partial x_1^2} + \frac{\partial^2 \phi}{\partial x_2^2} \right] + 2\cos\theta \left[\frac{\partial \phi}{\partial x_1} \frac{\partial \theta}{\partial x_1} + \frac{\partial \phi}{\partial x_2} \frac{\partial \theta}{\partial x_2} \right] = 0 \tag{29}$$

$$\frac{\partial^2 \theta}{\partial x_1^2} + \frac{\partial^2 \theta}{\partial x_2^2} - \sin\theta \cos\theta \left[\left(\frac{\partial \phi}{\partial x_1} \right)^2 + \left(\frac{\partial \phi}{\partial x_2} \right)^2 \right] = 0 \tag{30}$$

with the constraints

$$\int dx_1 dx_2 \ \cos\phi \sin\theta = \int dx_1 dx_2 \ \sin\phi \sin\theta = \int dx_1 dx_2 \ \cos\theta = 0 \tag{31}$$

The corresponding value for the energy E is then given by the expression

$$E = -\frac{1}{2} + \frac{\epsilon}{2} \int dx_1 dx_2 \left\{ \sin^2\theta \left[\left(\frac{\partial \phi}{\partial x_1} \right)^2 + \left(\frac{\partial \phi}{\partial x_1} \right)^2 \right] + \left(\frac{\partial \theta}{\partial x_1} \right)^2 + \left(\frac{\partial \theta}{\partial x_1} \right)^2 \right\}$$

Stage 5: use symmetries and pendulums ...

Finding the general solution of fierce equations like (29,30,31) is out of the question. However, we can find special solutions, namely those which are of the form suggested by the simulation results shown in figure 13. These configurations appear to have many symmetries, which we can exploit. In particular we make an 'ansatz' for the angles $\phi(x_1, x_2)$, which states that if the array of neurons would have been a circular disk, the configuration of synaptic strengths would have had rotational symmetry:

$$\cos\phi(x_1, x_2) = \frac{x_1}{\sqrt{x_1^2 + x_2^2}} \qquad \sin\phi(x_1, x_2) = \frac{x_2}{\sqrt{x_1^2 + x_2^2}} \qquad (32)$$

Insertion of this ansatz into our equations (29,30,31) shows that solutions with this property indeed exist, and that they imply a simple law for the remaining angle: $\theta(x_1, x_2) = \theta(r)$, with $r = \sqrt{x_1^2 + x_2^2}$) and

$$r^2\frac{d^2\theta}{dr^2} + r\frac{d\theta}{dr} - \sin\theta\cos\theta = 0 \qquad (33)$$

This is already an enormous simplification, but we need not stop here. A simple transformation of variables turns this differential equation (33) into the one describing the motion of a pendulum !

$$u = \log r, \qquad \theta(r) = \frac{1}{2}\pi + \frac{1}{2}G(u), \qquad \frac{d^2}{du^2}G(u) + \sin G(u) = 0 \qquad (34)$$

This means that we have basically cracked the problem, since the motion of a pendulum can be described analytically. There are two types of motion, the first one describes the familiar swinging pendulum and the second one describes a rotating one (the effect resulting from giving the pendulum significantly more than a gentle swing ...). If we calculate the synaptic energies E associated with the two types of motion (after translation back into the original synaptic strength variables) we find that the lowest energy is obtained upon choosing the solution corresponding to the pendulum motion which precisely separates rotation from swinging. Here the pendulum 'swings' just once, from one vertical position at $u = -\infty$ to another vertical position at $u = \infty$:

$$G(u) = \arcsin[\tanh(u + q)] \qquad (q \in \Re) \qquad (35)$$

If we now translate all results back into the original synaptic variables, combining equations (32,34,32,28), we end up with a beautifully simple solution:

$$w_x(x_1, x_2) = \frac{2kx_1}{k^2 + x_1^2 + x_2^2}$$

$$w_y(x_1, x_2) = \frac{2kx_2}{k^2 + x_1^2 + x_2^2} \qquad (36)$$

$$w_z(x_1, x_2) = \frac{k^2 - x_1^2 - x_2^2}{k^2 + x_1^2 + x_2^2}$$
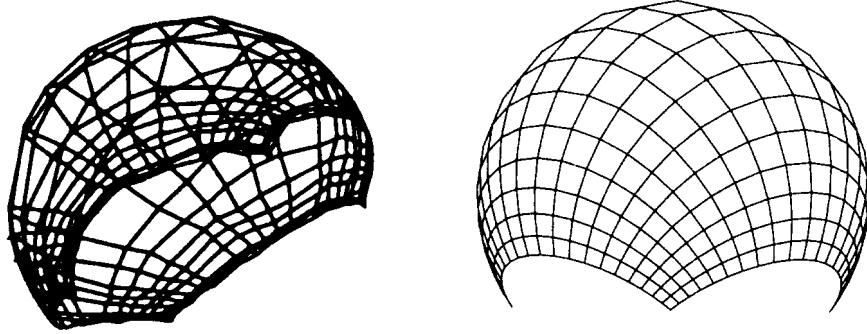
Figure 14: Graphical representation of the synaptic structure of a map forming network in the form of a 'fishing net'. Left: stationary state resulting from numerical simulations. Right: the analytical result (36).

in which the remaining constant k is uniquely defined as the solution of

$$\int dx_1 dx_2 \; \frac{k^2 - x_1^2 - x_2^2}{k^2 + x_1^2 + x_2^2} = 0$$

(which is the translation of the constraints (31)). The final test, of course, is to draw a picture of this solution in the representation of figure 13, which results in figure 14. The agreement is quite satisfactory.

5 Learning a Rule From an Expert

Finally let us turn to the class of neural systems most popular among engineers: layered neural networks. Here the information flows in only one direction, so that calulculating all neuron states at all times can be done iteratively (layer by layer), and has therefore become trivial. As a result one can concentrate on developing and studying non-trivial learning rules. The popular types of learning rules used in layered networks are the so-called 'supervised' ones, where the networks are trained by using examples of input signals ('questions') and the required output signals ('answers'). The latter are provided by a 'teacher'. The learning rules are 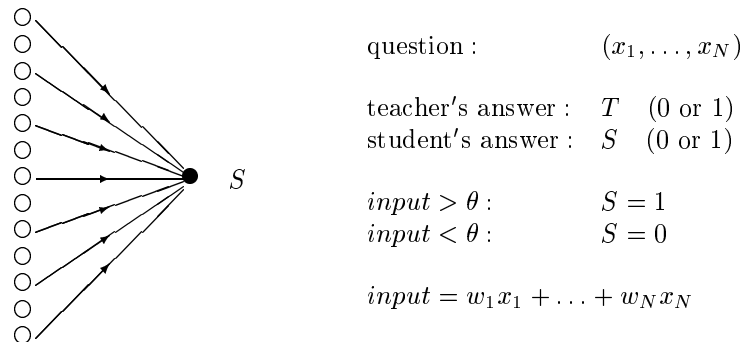based on comparing the network answers to the correct ones, and subsequently making adjustments to synaptic weights and thresholds to reduce the differences between the two answers to zero.

The most important property of neural networks in this context is their ability to generalise. In contrast to the situation where we would have just stored the question-answer pairs in memory, neural networks can, after having been confronted with a sufficient number of examples, generalise their 'knowledge' and provide reasonable, if not correct, answers even for new questions.

5.1 Perceptrons

The simplest feed-forward 'student' network one can think of is just a single binary neuron, with the standard operation rules,

question : $(x_1, \ldots, x_N)$

teacher's answer : T (0 or 1)
student's answer : S (0 or 1)

$input > \theta$: $S = 1$
$input < \theta$: $S = 0$

$input = w_1 x_1 + \ldots + w_N x_N$

trying to adjust its synaptic strengths $\{w_\ell\}$ such that for each question $(x_1, \ldots, x_N)$ its answer S coincides with the answer T given by the teacher. We now define a very simple learning rule (the so-called perceptron learning rule) to achieve this, where changes are made only if the neuron makes a mistake, i.e. if $T \neq S$:

1. select a question $(x_1, \ldots, x_N)$

2. compare $S(x_1, \ldots, x_N)$ and $T(x_1, \ldots, x_N)$:

   $S = T$: do nothing
   if $S = 1$, $T = 0$: change $w_i \to w_i - x_i$ and $\theta \to \theta + 1$
   if $S = 0$, $T = 1$: change $w_i \to w_i + x_i$ and $\theta \to \theta - 1$

Binary neurons, equipped with the above learning rule, are called 'Perceptrons' (reflecting their original use in the fifties to model perception). If a perceptron mistakenly produces an answer $S = 1$ for question $(x_1, \ldots, x_N)$ (i.e. *input* $> \theta$, whereas we would have preferred *input* $< \theta$), the modifications made ensure that the input produced upon presentation of this particular question will be reduced. If the perceptron mistakenly produces an answer $S = 0$ (i.e. *input* $< \theta$, whereas we would have preferred *input* $> \theta$), the changes made increase the input produced upon presentation of this particular question.

The perceptron learning rule appears to make sense, but it is as yet just one choice from an infinite number of possible rules. What makes the above rule special is that it comes with a convergence proof, in other words, it is guaranteed to work ! More precisely, provided the input vectors are bounded:

If values for the parameters $\{w_\ell\}$ and θ exists, such that $S = T$ for each question $(x_1, \ldots, x_N)$, then the perceptron learning rule will find these, or equivalent ones, in a finite number of modification steps

This is a remarkable statement. It could, for instance, easily have happened that correcting the system's answer for a given question $(x_1, \ldots, x_N)$ would affect the performance of the system on those questions it had so far been answering correctly, thus preventing the system from ever arriving at a state without errors. Apparently this does not happen. What is even more remarkable, is that the proof of this powerful statement is quite simple.

The standard version of the convergence proof assumes the set of questions Ω to be finite and discrete (in the continuous case one can construct a similar proof)[11]. To simplify notation we use a trick: we introduce a 'dummy' input variable $x_0 = -1$ (a constant). Upon giving the threshold θ a new name, $\theta = w_0$, our equations can be written in a very compact way. We denote the vector of weights as $\boldsymbol{w} = (w_0, \ldots, w_N)$, the questions as $\boldsymbol{x} = (x_0, \ldots, x_N)$, inner products as $\boldsymbol{w} \cdot \boldsymbol{x} = w_0 x_0 + \ldots w_N x_N$, and the length of a vector as $|\boldsymbol{x}| = \sqrt{\boldsymbol{x} \cdot \boldsymbol{x}}$. For the operation of the perceptron we get

$$\boldsymbol{w} \cdot \boldsymbol{x} > 0 : \ S = 1, \qquad \boldsymbol{w} \cdot \boldsymbol{x} < 0 : \ S = 0$$

whereas the learning rule becomes

$$\boldsymbol{w} \ \rightarrow \ \boldsymbol{w} + [T(\boldsymbol{x}) - S(\boldsymbol{x})]\boldsymbol{x} \tag{37}$$

There exists an error-free system (by assumption), with as yet unknown parameters $\boldsymbol{w}^\star$ (these include the threshold w_0). This system by definition obeys

$$T(\boldsymbol{x}) = 1 : \ \boldsymbol{w}^\star \cdot \boldsymbol{x} > 0, \qquad T(\boldsymbol{x}) = 0 : \ \boldsymbol{w}^\star \cdot \boldsymbol{x} < 0$$

Define $X = \max_\Omega |\boldsymbol{x}| > 0$ and $\delta = \min_\Omega |\boldsymbol{w}^\star \cdot \boldsymbol{x}| > 0$. The convergence proof relies on the following inequalities

$$\text{for all } \boldsymbol{x} \in \Omega : \qquad |\boldsymbol{x}| \le X \quad \text{and} \quad |\boldsymbol{w}^\star \cdot \boldsymbol{x}| \ge \delta \tag{38}$$

---

[11] One also assumes for simplicity that the borderline situation *input* $= \theta$ never occurs. This is not a big issue; such cases can be dealt with quite easily, should the need arise.

At each modification step $\boldsymbol{w} \to \boldsymbol{w}'$ (37), where $S = 1-T$ (otherwise: no modification !), we can inspect what happens to the quantities $\boldsymbol{w} \cdot \boldsymbol{w}^\star$ and $|\boldsymbol{w}|^2$:

$$\boldsymbol{w}' \cdot \boldsymbol{w}^\star = \boldsymbol{w} \cdot \boldsymbol{w}^\star + [2T(\boldsymbol{x})-1]\boldsymbol{x} \cdot \boldsymbol{w}^\star$$
$$|\boldsymbol{w}'|^2 = |\boldsymbol{w}|^2 + 2[2T(\boldsymbol{x})-1]\boldsymbol{x} \cdot \boldsymbol{w} + [2T(\boldsymbol{x})-1]^2|\boldsymbol{x}|^2$$

Note that $2T(\boldsymbol{x})-1 = -1$ if $\boldsymbol{w}^\star \cdot \boldsymbol{x} < 0$, and $2T(\boldsymbol{x})-1 = 1$ if $\boldsymbol{w}^\star \cdot \boldsymbol{x} > 0$, and that therefore $[2T(\boldsymbol{x})-1]\boldsymbol{x} \cdot \boldsymbol{w} < 0$ (otherwise one would have had $S = T$), so

$$\boldsymbol{w}' \cdot \boldsymbol{w}^\star = \boldsymbol{w} \cdot \boldsymbol{w}^\star + |\boldsymbol{x} \cdot \boldsymbol{w}^\star| \geq \boldsymbol{w} \cdot \boldsymbol{w}^\star + \delta$$
$$|\boldsymbol{w}'|^2 < |\boldsymbol{w}|^2 + |\boldsymbol{x}|^2 \leq |\boldsymbol{w}|^2 + X^2$$

After n such modification steps we therefore find

$$\boldsymbol{w}(n) \cdot \boldsymbol{w}^\star \geq \boldsymbol{w}(0) \cdot \boldsymbol{w}^\star + n\delta$$
$$|\boldsymbol{w}(n)|^2 \leq |\boldsymbol{w}(0)|^2 + nX^2$$

In combination this implies the following inequality:

$$\frac{\boldsymbol{w}(n) \cdot \boldsymbol{w}^\star}{|\boldsymbol{w}^\star||\boldsymbol{w}(n)|} \geq \frac{\boldsymbol{w}(0) \cdot \boldsymbol{w}^\star + n\delta}{|\boldsymbol{w}^\star|\sqrt{|\boldsymbol{w}(0)|^2 + nX^2}}$$

giving

$$\lim_{n \to \infty} \frac{1}{\sqrt{n}} \frac{\boldsymbol{w}(n) \cdot \boldsymbol{w}^\star}{|\boldsymbol{w}^\star||\boldsymbol{w}(n)|} \geq \frac{\delta}{|\boldsymbol{w}^\star|X} > 0 \tag{39}$$

We see that the number of modifications made *must* be bounded, since otherwise (39) leads to a contradiction with the Schwarz inequality $|\boldsymbol{w} \cdot \boldsymbol{w}^\star| \leq |\boldsymbol{w}||\boldsymbol{w}^\star|$. As soon as no more modifications are made, we must be in a situation where $S(\boldsymbol{x}) = T(\boldsymbol{x})$ for each question $\boldsymbol{x} \in \Omega$. This completes the proof.

Figure 15 gives an impression of the learning process described by the perceptron learning rule. In these simulation experiments the task T is defined by a teacher perceptron with a (randomly drawn) synaptic weight vector $\boldsymbol{w}^\star$:

$$\boldsymbol{w}^\star \cdot \boldsymbol{x} > 0: \quad T(\boldsymbol{x}) = 1 \qquad \boldsymbol{w}^\star \cdot \boldsymbol{x} < 0: \quad T(\boldsymbol{x}) = 0$$

The evolution of the student perceptron's synaptic vector $\boldsymbol{w}$ is monitored by calculating at each iteration step the quantity ω which already played a dominant role in the convergence proof for the perceptron learning rule, and which measures the resemblance between $\boldsymbol{w}$ and $\boldsymbol{w}^\star$:

$$\omega = \frac{\boldsymbol{w} \cdot \boldsymbol{w}^\star}{|\boldsymbol{w}||\boldsymbol{w}^\star|} \tag{40}$$

At each iteration step each component x_i of the questions $\boldsymbol{x}$ posed during the simulation experiments was drawn randomly from $\{-1, 1\}$. Figure 15 suggest that for large perceptrons, $N \to \infty$, our general strategy of trying to derive exact analytical and deterministic dynamical laws might again be succesful. This turns out to be true, as we will show in a subsequent section.
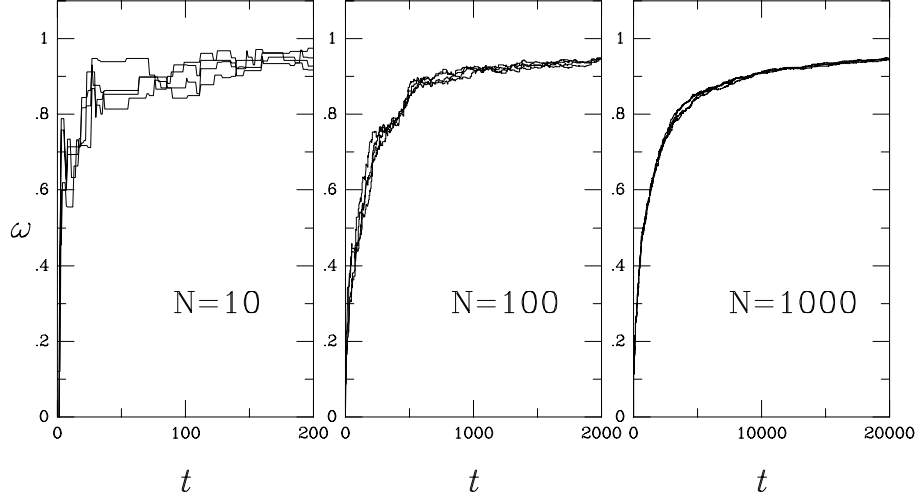
Figure 15: Evolution in time of the observable $\omega = \boldsymbol{w} \cdot \boldsymbol{w}^\star / |\boldsymbol{w}||\boldsymbol{w}^\star|$, obtained by numerical simulation of the perceptron learning rule, for a randomly drawn teacher vector $\boldsymbol{w}^\star$ and binary questions $(x_1, \ldots, x_N) \in \{-1, 1\}^N$. Each picture shows the results following four different random initialisations of the student vector $\boldsymbol{w}$.

Since we know that the perceptron learning rule will converge for each realisable task, we need only worry about which tasks are learnable by perceptrons and which are not. Tasks that can be performed by single binary neurons are called 'linearly separable'. Unfortunately, not all rules $(x_1, \ldots, x_N) \to T(x_1, \ldots, x_N) \in \{0, 1\}$ can be performed with simple binary neurons. The simplest counter-example is the so-called XOR operation, XOR: $\{0, 1\}^2 \to \{0, 1\}$, defined below. One can prove quite easily that there cannot exist a choice of parameters $\{w_1, w_2, \theta\}$ such that

$$\text{XOR}(x_1, x_2) = 1 : \quad w_1 x_1 + w_2 x_2 > \theta$$
$$\text{XOR}(x_1, x_2) = 0 : \quad w_1 x_1 + w_2 x_2 < \theta$$

by just checking explicitly the four possibilities for (x_1, x_2):

x_1	x_2	$\text{XOR}(x_1, x_2)$	requirement
0	0	0	$\theta > 0$
0	1	1	$w_2 > \theta$
1	0	1	$w_1 > \theta$
1	1	0	$w_1 + w_2 < \theta$

The four parameter requirements are clearly contradictory.

38

5.2 Multi-layer Networks

If we want a neural network to perform an operation T that cannot be performed by a single binary neuron, like the XOR operation in the previous section, we need a more complicated network architecture. For the case where the question variables x_i are binary we know that the two-layer architecture shown in figure 5 is in principle sufficiently complicated (see the proof in section 2.2), but as yet we have no learning rule available that will allow us to train such systems. For real-valued question variables one can prove that two-layer architectures can perform any sufficiently regular[12] task T with arbitrary accuracy, if the number of neurons in the 'hidden' layer is sufficiently large and provided we turn our binary neurons into so-called 'graded-response' ones:
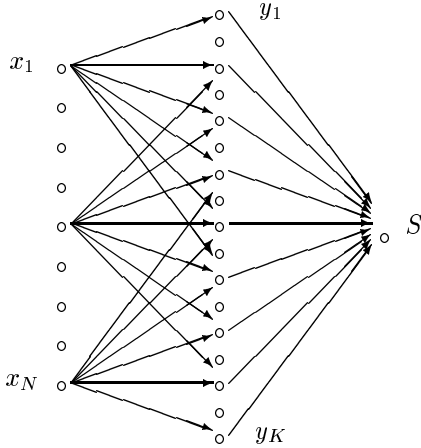
$$S = f(w_1 x_1 + \ldots w_N x_N - \theta) \tag{41}$$

in which the non-linear function $f(z)$ has the properties

$$f'(z) \geq 0, \qquad \lim_{z \to \pm\infty} |f(z)| < \infty$$

Commonly made choices for f are $f(z) = \tanh(z)$ and $f(z) = \mathrm{erf}(z)$. Definition (41) can be seen as a generalisation of the binary neurons considered so far, which correspond to choosing a step-function: $f(z > 0) = 1$, $f(z < 0) = 0$. From now on we will assume the task T and the nonlinear function f to be normalised according to $|T(x_1, \ldots, x_N)| \leq 1$ for all $(x_1, \ldots, x_N)$, and $|f(z)| \leq 1$ for all $z \in \Re$. Given a 'student' S in the form of a (universal) two-layer feed-forward architecture with neurons of the type (41), we can write the student answer $S(x_1, \ldots, x_N)$ to question $(x_1, \ldots, x_N)$ as

$$S(x_1, \ldots, x_N) = f(w_1 y_1 + \ldots + w_K y_K - \theta)$$
$$y_\ell(x_1, \ldots, x_N) = f(w_{\ell 1} x_1 + \ldots + w_{\ell N} x_N - \theta_\ell) \tag{42}$$

question : $\qquad (x_1, \ldots, x_N)$

teacher$'$s answer : $\quad T \in [-1, 1]$
student$'$s answer : $\quad S \in [-1, 1]$

$$S = f(w_1 y_1 + \ldots + w_K y_K - \theta)$$

$$y_\ell = f(w_{\ell 1} x_1 + \ldots + w_{\ell N} x_N - \theta_\ell)$$

---

[12] $T(x_1, \ldots, x_N)$ should, for instance, be bounded for all $(x_1, \ldots, x_N)$

As before, we can simplify (42) and eliminate the thresholds θ and $\{\theta_\ell\}$ by introducing dummy neurons $x_0 = -1$ and $y_0 = -1$, with corresponding synaptic strengths w_0 and $\{w_{\ell 0}\}$.

Our goal is to find a learning rule. In this case it would be a recipe for the modification of the synaptic strengths w_i (connecting 'hidden' neurons to the output neuron) and $\{w_{ij}\}$ (connecting the input signals, or questions components, x_i to the hidden neurons), based on the observed differences between the teacher's answers $T(x_0, \ldots, x_N)$ and the student's answers $S(x_0, \ldots, x_N)$. If we denote the set of all possible questions by Ω, and the probability that a given question $\boldsymbol{x} = (x_0, \ldots, x_N)$ will be asked by $p(\boldsymbol{x})$, we can quantify the student's performance at any stage during training by its average quadratic error E[13]:

$$E = \frac{1}{2} \sum_{\boldsymbol{x} \in \Omega} p(\boldsymbol{x}) \left[T(\boldsymbol{x}) - S(\boldsymbol{x})\right]^2 \tag{43}$$

Training is supposed to minimise E, preferably to zero. This suggest a very simple way to modify the system's parameters, the so-called 'gradient descent' procedure. Here one inspects the change in E following infinitesimally small parameter changes, and then chooses those modifications for which E would decrease most (like a blind mountaineer in search of the valley). In terms of partial derivatives this implies the following 'motion' for the parameters:

$$\begin{aligned} \text{for all } i = 1, \ldots, K: \quad & \frac{d}{dt} w_i = -\frac{\partial}{\partial w_i} E \\ \text{for all } i = 1, \ldots, K, \ j = 1, \ldots, N: \quad & \frac{d}{dt} w_{ij} = -\frac{\partial}{\partial w_{ij}} E \end{aligned} \tag{44}$$

Although the equations one finds upon working out the derivatives in (44) are rather messy compared to the simple perceptron rule (37), the procedure (44) is guaranteed to decrease the value of the error E until a stationary state is reached, since from (44) we can deduce via the chain rule:

$$\frac{dE}{dt} = \sum_{i=1}^{K} \left[\frac{\partial E}{\partial w_i}\right] \left[\frac{dw_i}{dt}\right] + \sum_{i=1}^{K} \sum_{j=1}^{N} \left[\frac{\partial E}{\partial w_{ij}}\right] \left[\frac{dw_{ij}}{dt}\right]$$

$$= -\sum_{i=1}^{K} \left[\frac{dw_i}{dt}\right]^2 - \sum_{i=1}^{K} \sum_{j=1}^{N} \left[\frac{dw_{ij}}{dt}\right]^2 \leq 0$$

A stable stationary state is reached only when every small modification of the synaptic weights would lead to an increase in E, i.e. when we are at a *local minimum* of E. However, this local minimum need not be the desired *global minimum*; the blind mountaineer might find himself in some small valley high up in the mountains, rather than the one he is aiming for.

It will be clear that the principle behind the learning rule (44) can be applied to any feed-forward architecture, with arbitrary numbers of 'hidden' layers of

---

[13] Note that this is an arbitrary choice; any monotonic function of $|T(\boldsymbol{x}) - S(\boldsymbol{x})|$ other than $|T(\boldsymbol{x}) - S(\boldsymbol{x})|^2$ would do.

arbitrary size, since it relies only on our being able to write down an explicit expression for the student's answers $S(\boldsymbol{x})$ in expression (43) for the error E. One just writes down equations of the form (44) for every parameter to be modified in the network. One can even generalise the construction to the case of multiple output variables (multiple answers):
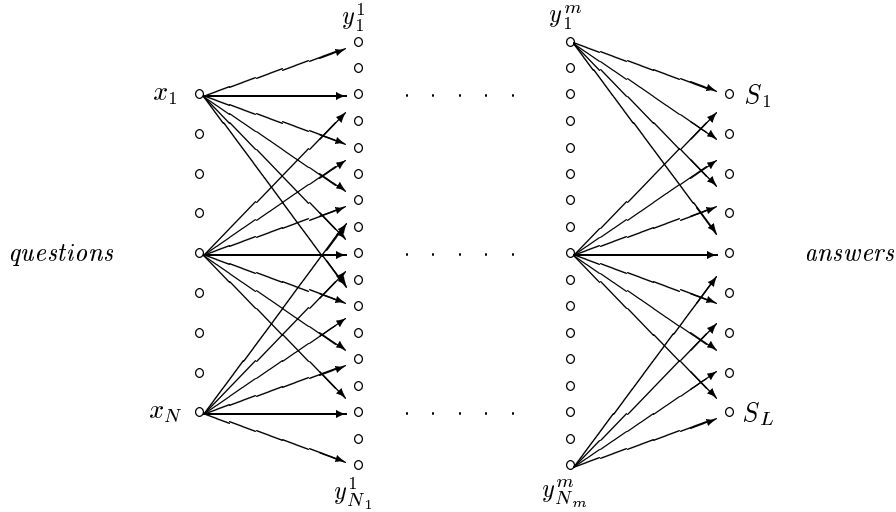
In this case there are L student answers $S_\ell(x_0, \ldots, x_N)$ and L teacher answers $T_\ell(x_0, \ldots, x_N)$ for each question $(x_0, \ldots, x_N)$, so that the error (43) to be minimised by gradient descent is to be generalised to

$$E = \frac{1}{2} \sum_{\boldsymbol{x} \in \Omega} p(\boldsymbol{x}) \sum_{\ell=1}^{L} \left[T_\ell(\boldsymbol{x}) - S_\ell(\boldsymbol{x}) \right]^2 \tag{45}$$

The strategy of tackling tasks which are not linearly separable (i.e. not executable by single perceptrons) using multi-layer networks of graded-response neurons which are being trained by gradient descent has several advantages, but also several drawbacks. Just to list a few:

  + the networks are in principle universal
  + we have a learning rule that minimises the student's error
  + the rule works for arbitrary numbers of layers and layer sizes

  − we don't know the required network dimensions beforehand
  − the rule cannot be implemented exactly (infinitely small modifications !)
  − convergence is not guaranteed (we can end up in a local minimum of E)
  − at each step we need to evaluate *all* student and teacher answers

The lack of a priori guidelines in choosing numbers of layers and layer sizes, and the need for discretisation and approximation of the differential equation (44) unfortunately generate quite a number of parameters to be tuned by hand. This
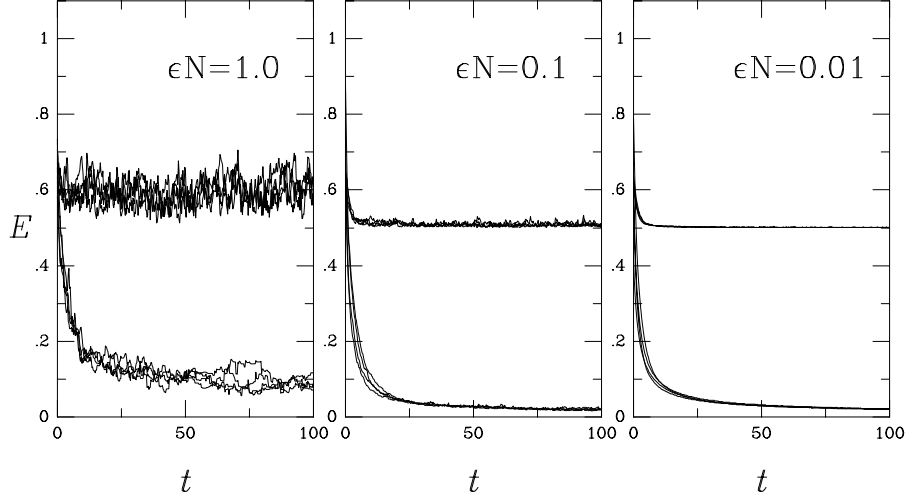
Figure 16: Evolution of the student's error E (43) in a two-layer feed-forward network with $N = 15$ input neurons and $K = 10$ 'hidden' neurons. The parameter ϵ gives the elementary time-steps used in the discretisation of the gradient descent learning rule. The upper graphs refer to task I (not linearly separably), the lower graphs to task II (which is linearly separable).

problem can be solved only by having exact theories to describe the learning process; I will give a taste of recent analytical developments in a subsequent section.

Figures 16 and 17 give an impression of the learning process described by the following discretisation/approximation of the original equation (44)

$$w_i(t+\epsilon) = w_i(t) - \epsilon\frac{\partial}{\partial w_i}E[\boldsymbol{x}(t)] \qquad w_{ij}(t+\epsilon) = w_{ij}(t) - \epsilon\frac{\partial}{\partial w_{ij}}E[\boldsymbol{x}(t)]$$

$$E[\boldsymbol{x}] = \frac{1}{2}[T(\boldsymbol{x}) - S(\boldsymbol{x})]^2$$

in a two-layer feed-forward network, with graded response neurons of the type (41), in which the non-linear function was choosen to be $f(z) = \tanh(z)$. Apart from having a discrete time, as opposed to the continuous one in (44), the second approximation made consists of replacing the *overall* error E (43) in (44) by the error $E[\boldsymbol{x}(t)]$ made in answering the current question $\boldsymbol{x}(t)$. The rationale is that for times $t \ll \epsilon^{-1}$ the above learning rule tends towards the original one (44) in the limit $\epsilon \to 0$. In the simulation examples the following two tasks T were considered:

$$\boldsymbol{x} \in \{-1, 1\}^N : \quad \begin{cases} \text{task I}: & T(\boldsymbol{x}) = \prod_{i=1}^N x_i \\ \text{task II}: & T(\boldsymbol{x}) = \text{sgn}[\boldsymbol{w}^\star \cdot \boldsymbol{x}] \end{cases} \tag{46}$$
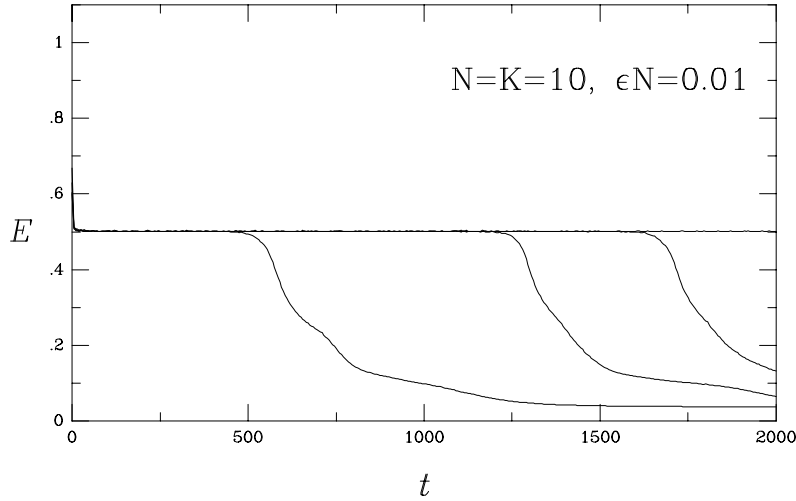
42

Figure 17: Evolution of the student's error E (43) in a two-layer feed-forward network with $N = 10$ input neurons and $K = 10$ 'hidden' neurons, being trained on task I (which is not linearly separably). The so-called 'plateau phase' is the intermediate (slow) stage in the learning process, where the error appears to have stabilised.

with, in the case of task II, a randomly drawn teacher vector $\boldsymbol{w}^\star$. Task I is not linearly separable, task II is. At each iteration step each component x_i of the question $\boldsymbol{x}$ in the simulation experiments was drawn randomly from $\{-1, 1\}$. In spite of the fact that task I can be performed with a two-layer feed-forward network if $K \geq N$ (which can be demonstrated by construction), figure 16 suggests that the learning procedure used does not converge to the desired configuration. This, however, is just a demonstration of one of the characteristic features of learning in multi-layer networks: the occurrence of so-called 'plateau phases'. These are phases in the learning process where the error appears to have stabilised (suggesting arrival at a local minimum), but where it is in in fact only going through an extremely slow intermediate stage. This is illustrated in figure 17, where much larger observation times are choosen.

5.3   Calculating what is Achievable

There are several reasons why one would like to know beforehand for any given task T which is the minimal architecture necessary for a student network S to be able to 'learn' this task. Firstly, if we can get away with using a perceptron, as opposed to a multi-layer network, then we can use the perceptron learning rule, which is simpler and is guaranteed to converge. Secondly, the larger the number of layers and layer sizes, the larger the amount of computer time needed to carry out the training process. Thirdly, if the number of adjusted

43

parameters is too large compared to the number actually required, the network might end up doing brute-force data-fitting which resembles the creation of a look-up table for the answers, rather than learning the underlying rule, leading to poor generalisation.

We imagine having a task (teacher), in the form of a rule T assigning an answer $T(\boldsymbol{x})$ to each question $\boldsymbol{x} = (x_0, \ldots, x_N)$, drawn from a set Ω with probability $p(\boldsymbol{x})$. We also have a student network with a given architecture, and a set of adjustable parameters $\boldsymbol{w} = (w_0, \ldots, w_N)$, assigning an answer $S(\boldsymbol{x}; \boldsymbol{w})$ to each question (the rule operated by the student depends on the adjustable paremeters $\boldsymbol{w}$, which we now emphasise in our notation). The answers could take discrete or continuous values. The degree to which a student with parameter values $\boldsymbol{w}$ has succesfully learned the rule operated by the teacher is measured by an error $E[\boldsymbol{w}]$, usually choosen to be of the form

$$E[\boldsymbol{w}] = \frac{1}{2} \sum_{\boldsymbol{x} \in \Omega} p(\boldsymbol{x}) \left[T(\boldsymbol{x}) - S(\boldsymbol{x}; \boldsymbol{w}) \right]^2 \tag{47}$$

What we would like to calculate is $E_0 = \min_{\boldsymbol{w} \in W} E[\boldsymbol{w}]$, since

$$E_0 = \min_{\boldsymbol{w} \in W} E[\boldsymbol{w}] : \qquad \begin{cases} E_0 > 0 : & \text{task not feasable} \\ E_0 = 0 : & \text{task feasable} \end{cases} \tag{48}$$

W denotes the set of allowed values for $\boldsymbol{w}$. The freedom in choosing W allows us to address a wider range of feasibility questions; for instance, we might have constraints on the allowed values of $\boldsymbol{w}$ due to restrictions imposed by the (biological or electrical) hardware used. If $E_0 = 0$ we know that it is at least in principle possible for the present student architecture to arrive at a stage with zero error; if $E_0 > 0$, on the other hand, no learning process will ever lead to error-free performance. In the latter case, the actual magnitude of E_0 still contains valuable information, since allowing for a certain fraction of errors could be a price worth paying if it means a drastic reduction in the complexity of the architecture (and therefore in the amount of computing time).

Performing the minimisation in (48) explicitly is usually impossible, however, as long as we do not insist on knowing for which parameter setting(s) $\boldsymbol{w}^\star$ the minimum $E_0 = E[\boldsymbol{w}^\star]$ is actually obtained, we can use the following identity[14]:

$$E_0 = \lim_{\beta \to \infty} \frac{\int_W d\boldsymbol{w} \ E[\boldsymbol{w}] e^{-\beta E[\boldsymbol{w}]}}{\int_W d\boldsymbol{w} \ e^{-\beta E[\boldsymbol{w}]}} \tag{49}$$

The expression (49) can also be written in the following way, requiring us just to calculate a single integral

$$E_0 = - \lim_{\beta \to \infty} \frac{\partial}{\partial \beta} \log \mathcal{Z}[\beta] \qquad \mathcal{Z}[\beta] = \int_W d\boldsymbol{w} \ e^{-\beta E[\boldsymbol{w}]} \tag{50}$$

---

[14] Strictly speaking this is no longer true if the minimum is obtained *only* for values of $\boldsymbol{w}$ in sets of measure zero. In practice this is not a serious restriction; in the latter case the system would be extremely sensitive to noise (numerical or otherwise) and thus of no practical use.

The integral in (50) need not (and usually will not) be trivial, but it can often be done. The results thus obtained can save us from running extensive (and expensive) computer simulations, only to find out the hard way that the architecture of the network at hand is too poor. Alternatively they can tell us what the minimal architecture is, given a task, and thus allow us to obtain networks with optimal generalisation properties.

Often we might wish to reduce our ambition further, in order to obtain exact statements. For instance, we could be interested in a certain *family* of tasks T, i.e. $T \in \mathcal{B}$, with $\mathcal{P}(T)$ denoting the probability of task T to be encountered, and try to find out about the feasibility of a generic task from this family, rather than the feasibility of each individual family member:

$$E[T; \boldsymbol{w}] = \frac{1}{2} \sum_{\boldsymbol{x} \in \Omega} p(\boldsymbol{x}) \left[T(\boldsymbol{x}) - S(\boldsymbol{x}; \boldsymbol{w}) \right]^2 \tag{51}$$

$$\langle E_0[T] \rangle_{\mathcal{B}} = \int_{\mathcal{B}} dT \; \mathcal{P}(T) \; E_0[T]$$

$$= -\lim_{\beta \to \infty} \frac{\partial}{\partial \beta} \int_{\mathcal{B}} dT \; \mathcal{P}(T) \; \log \left\{ \int_W d\boldsymbol{w} \; e^{-\beta E[T; \boldsymbol{w}]} \right\} \tag{52}$$

In those cases where the original integral in (50) cannot be done analytically, one often finds that the quantity (52) can be calculated by doing the integration over the tasks before the integration over the students. Since (47) ensures $E_0 \geq 0$, we know that $\langle E_0[T] \rangle_{\mathcal{B}} = 0$ implies that if we randomly choose a task from the family $\mathcal{B}$, then the probability that a randomly drawn task from the family $\mathcal{B}$ is not feasible is zero.

Application of the above ideas to a single binary neuron leads to statements on which types of operations $T(x_1, \ldots, x_N)$ are linearly separable. For example, let us define a task by choosing at random p binary questions $(\xi_1^\mu, \ldots, \xi_N^\mu)$, with $\xi_i^\mu \in \{-1, 1\}$ for all (i, μ) and $\mu = 1, \ldots, p$, and assign randomly an output value $T_\mu = T(\xi_1^\mu, \ldots, \xi_N^\mu) \in \{0, 1\}$ to each. The synaptic strengths $\boldsymbol{w}$ of an error-free perceptron would then be the solution of the following problem:

$$\text{for all } \mu : \quad \begin{cases} T_\mu = 1 : & w_1 \xi_1^\mu + \ldots + w_N \xi_N^\mu > 0 \\ T_\mu = 0 : & w_1 \xi_1^\mu + \ldots + w_N \xi_N^\mu < 0 \end{cases} \tag{53}$$

The larger p, the more complicated the task, and the smaller the set of solutions $\boldsymbol{w}$. For large N, the maximum number p of random questions that a perceptron can handle turns out to scale with N, i.e. $p = \alpha N$ for some $\alpha > 0$. Finding out for a specific task T, which is specified by the choice made for all question/answer pairs $\boldsymbol{\xi}^\mu = (\xi_1^\mu, \ldots, \xi_N^\mu; T_\mu)$, whether a solution of (53) exists, either directly or by calculating (50), is impossible (except for trivial and pathological cases). Our family $\mathcal{B}$ is the set of all such tasks T obtained by choosing different realisations of the question/answer pairs $(\xi_1^\mu, \ldots, \xi_N^\mu; T_\mu)$;

there are 2^{p+1} possible choices of question/answer pairs, each equally likely. The error (51) and the family-averaged mimimum error in (52) thus become

$$E[T; \boldsymbol{w}] = \frac{1}{2p} \sum_{\mu=1}^{p} [T_\mu - S(\boldsymbol{\xi}^\mu; \boldsymbol{w})]^2 \tag{54}$$

$$\langle E_0[T] \rangle_{\mathcal{B}} = - \lim_{\beta \to \infty} \frac{\partial}{\partial \beta} \frac{1}{2^{p+1}} \sum_{\boldsymbol{\xi}^\mu} \log \left\{ \int_W d\boldsymbol{w} \; e^{-\frac{\beta}{2} \sum_\mu [T_\mu - S(\boldsymbol{\xi}^\mu; \boldsymbol{w})]^2} \right\} \tag{55}$$

If one specifies the set W of allowed student vectors $\boldsymbol{w}$ by simply requiring $w_1^2 + \ldots + w_N^2 = 1$, one finds that the average (55) can indeed be calculated[15], which results in the following statement on $\alpha = p/N$: for large N there exists a critical value α_c such that for $\alpha < \alpha_c$ the tasks in the family $\mathcal{B}$ are linearly separable, whereas for $\alpha > \alpha_c$ the tasks in $\mathcal{B}$ are not. The critical value turns out to be $\alpha_c = 2$.

We can play many interesting games with this procedure. Note that, since the associative memory networks of a previous section consist of binary neurons, this result also has immediate applications in terms of network storage capacities: for large networks the maximum number p_c of random patterns that can be stored in an associative memory network of binary neurons obeys $p_c/N \to 2$ for $N \to \infty$. We can also investigate the effect on the storage capacity of of the degree of symmetry of the synaptic strengths w_{ij}, which is measured by

$$\eta(\boldsymbol{w}) = \frac{\sum_{ij} w_{ij} w_{ji}}{\sum_{ij} w_{ij}^2} \in [-1, 1]$$

For $\eta(\boldsymbol{w}) = -1$ the synaptic matrix is anti-symmetric, i.e. $w_{ij} = -w_{ji}$ for all neuron pairs (i, j); for $\eta(\boldsymbol{w}) = 1$ it is symmetric, i.e. $w_{ij} = w_{ji}$ for all neuron pairs. If one now specifies in expression (55) the set W of allowed synaptic strengths by requiring both $w_1^2 + \ldots + w_N^2 = 1$ and $\eta(\boldsymbol{w}) = \eta$ (for some fixed η), our calculation will give us the storage capacity as a function of the degree of symmetry: $\alpha_c(\eta)$. Somewhat unexpectedly, the optimal network turns out not to be symmetric:

$$\eta = -1 : \quad \text{fully anti-symmetric synapses,} \quad \alpha_c = \tfrac{1}{2}$$

$$\eta = 0 : \quad \text{no symmetry preference,} \quad \alpha_c \sim 1.94$$

$$\eta = \tfrac{1}{\pi} : \quad \text{optimal synapses,} \quad \alpha_c = 2$$

$$\eta = 1 : \quad \text{fully symmetric synapses,} \quad \alpha_c \sim 1.28$$

---

[15] This involves a few technical subtleties which go beyond the scope of the present paper.

5.4 Solving the Dynamics of Learning for Perceptrons

As with our previous network classes, the associative memory networks and the networks responsible for creating topology conserving maps, we can for the present class of layered networks obtain analytical results on the dynamics of supervised learning, provided we restrict ourselves to (infinitely) large systems. In particular, we can find the system error as a function of time. Here I will only illustrate the route towards this result for perceptrons, and restrict myself to specific parameter limits. A similar approach can be followed for multi-layer systems; this is in fact one of the most active present research areas.

Stage 1: define the rules

For simplicity we will not deal with thresholds, i.e. $\theta = 0$, and we will draw at each time-step t each bit $x_i(t)$ of the question $\boldsymbol{x}(t)$ asked at random from $\{-1, 1\}$. Furthermore, we will only consider the case where a perceptron S is being trained on a linearly separable (i.e. feasable) task, which means that the operation of the teacher T can itself be seen as that of a binary neuron, with synaptic strengths $\boldsymbol{w}^\star = (w_1^\star, \ldots, w_N^\star)$. Since the (constant) value of the length of the teacher vector $\boldsymbol{w}^\star$ has no effect on the process (37), we are free to choose the simplest normalisation $|\boldsymbol{w}^\star|^2 = w_1^{\star 2} + \ldots + w_N^{\star 2} = 1$.

In the original perceptron learning rule we can introduce a so-called learning rate $\epsilon > 0$, which defines the magnitude of the elementary modifications of the student's synaptic strengths $\boldsymbol{w} = (w_1, \ldots, w_N)$, by rescaling the modification term in equation (37). This does not affect the convergence proof. It just gauges the time-scale of the learning process, so we choose this ϵ to define the duration of individual iteration steps. For realisable tasks the teacher's answer to a question $\boldsymbol{x} = (x_1, \ldots, x_N)$ depends only on the sign of $\boldsymbol{w}^\star \cdot \boldsymbol{x} = w_1^\star x_1 + \ldots + w_N^\star x_N$. Upon combining our ingredients we can replace the learning rule (37) by the following expression

$$\boldsymbol{w}(t+\epsilon) = \boldsymbol{w}(t) + \frac{1}{2}\epsilon \; \boldsymbol{x}(t) \left\{ \mathrm{sgn}[\boldsymbol{w}^\star \cdot \boldsymbol{x}(t)] - \mathrm{sgn}[\boldsymbol{w}(t) \cdot \boldsymbol{x}(t)] \right\} \qquad (56)$$

with $\mathrm{sgn}[z > 0] = 1$, $\mathrm{sgn}[z < 0] = -1$ and $\mathrm{sgn}[0] = 0$. If we now consider very small learning rates $\epsilon \to 0$, the following two pleasant simplifications occur[16]: (*i*) the discrete-time iterative map (56) will be replaced by a continuous-time differential equation, and (*ii*) the right-hand side of (56) will be converted into an expression involving only *averages* over the distribution of questions, to be denoted by $\langle \ldots \rangle_{\boldsymbol{x}}$:

$$\frac{d}{dt}\boldsymbol{w} = \frac{1}{2}\langle \boldsymbol{x} \left\{ \mathrm{sgn}[\boldsymbol{w}^\star \cdot \boldsymbol{x}] - \mathrm{sgn}[\boldsymbol{w} \cdot \boldsymbol{x}] \right\} \rangle_{\boldsymbol{x}} \qquad (57)$$

---

[16]Note that this is the same procedure we followed to analyse the creation of topology conserving maps, in a previous section.

Stage 2: find the relevant macroscopic features

The next stage, as always, is to decide which are the quantities we set out to calculate. For the perceptron there is a clear guide in finding the relevant macroscopic features, namely the perceptron convergence proof. In this proof the following two observables played a key role:

$$J = \sqrt{w_1^2 + \ldots w_N^2} \qquad \omega = \frac{w_1 w_1^\star + \ldots w_N w_N^\star}{\sqrt{w_1^2 + \ldots w_N^2}} \tag{58}$$

Last but not least one would like to know at any time the accuracy with which the student has learned the task, as measured by the error E:

$$E = \frac{1}{2} \left\langle \left\{ 1 - \text{sgn}[(\boldsymbol{w}^\star \cdot \boldsymbol{x})(\boldsymbol{w} \cdot \boldsymbol{x})] \right\} \right\rangle_{\boldsymbol{x}} \tag{59}$$

which simply gives the fraction of questions which is wrongly answered by the student. We can use equation (57) to derive a differential equation describing the evolution in time of the two observables (58), which after some rearranging can be written in the form

$$\frac{d}{dt} J = -\int_0^\infty \int_0^\infty dy \, dz \; z \left[P(y, -z) + P(-y, z) \right] \tag{60}$$

$$\frac{d}{dt} \omega = \frac{1}{J} \int_0^\infty \int_0^\infty dy \, dz \; [y + \omega z] \left[P(y, -z) + P(-y, z) \right] \tag{61}$$

in which the details of the student and teacher vectors $\boldsymbol{w}$ and $\boldsymbol{w}^\star$ enter only through the probability distribution $P(y, z)$ for the two local field sums

$$y = w_1^\star x_1 + \ldots + w_N^\star x_N \qquad z = \frac{1}{J}(w_1 x_1 + \ldots + w_N x_N)$$

Similarly we can write the student's error E in (59) as

$$E = \int_0^\infty \int_0^\infty dy \, dz \; \left[P(y, -z) + P(-y, z) \right] \tag{62}$$

Note that the way everything has been defined so far guarantees a more or less smooth limit $N \to \infty$ when we eventually go to large systems, since with our present choice of question statistics we find for any N:

$$\int dy \, dz \; P(y, z) y = \int dy \, dz \; P(y, z) z = 0 \tag{63}$$

$$\int dy \, dz \; P(y, z) y^2 = \langle [\boldsymbol{w}^\star \cdot \boldsymbol{x}]^2 \rangle_{\boldsymbol{x}} = \boldsymbol{w}^{\star 2} = 1 \tag{64}$$

$$\int dy \, dz \; P(y, z) z^2 = J^{-2} \langle [\boldsymbol{w} \cdot \boldsymbol{x}]^2 \rangle_{\boldsymbol{x}} = J^{-2} \boldsymbol{w}^2 = 1 \tag{65}$$

$$\int dy \, dz \; P(y, z) yz = J^{-1} \langle [\boldsymbol{w}^\star \cdot \boldsymbol{x}][\boldsymbol{w} \cdot \boldsymbol{x}] \rangle_{\boldsymbol{x}} = J^{-1} \boldsymbol{w}^\star \cdot \boldsymbol{w} = \omega \tag{66}$$
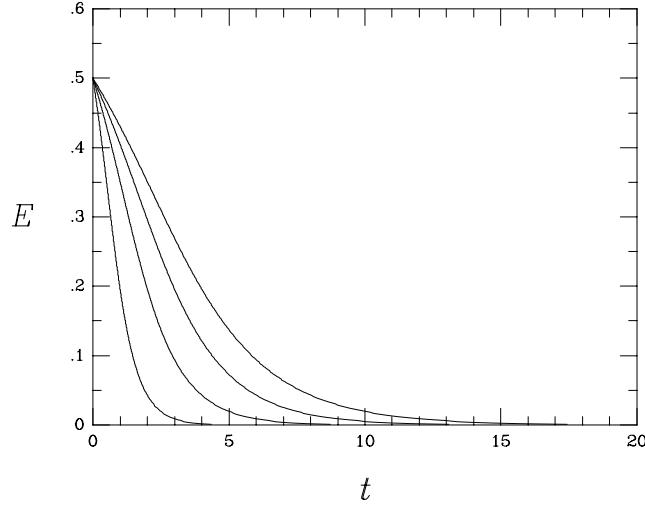
Figure 18: Evolution in time of the student's error E in an infinitely large perceptron in the limit of an infinitesimally small learning rate, according to (70). The four curves correspond to four random initialisations for the student vector $\boldsymbol{w}$, with lengths $J(0) = 2, \frac{3}{2}, 1, \frac{1}{2}$ (from top to bottom).

Stage 3: calculate $P(y, z)$ in the limit $N \to \infty$

For finite systems the shape of the distribution $P(y, z)$ depends in some complicated way on the details of the student and teacher vectors $\boldsymbol{w}$ and $\boldsymbol{w}^\star$, although the moments (63-66) will for any size N depend on the observable ω only. For large perceptrons ($N \to \infty$), however, a drastic simplification occurs: due to the statistical independence of our question components $x_i \in \{-1, 1\}$ the central limit theorem applies, which guarantees that the distribution $P(y, z)$ will become Gaussian[17]. This implies that it is characterised only by the moments (63-66), and therefore depends on the vectors $\boldsymbol{w}$ and $\boldsymbol{w}^\star$ *only* through the observable ω:

$$P(y, z) = \frac{1}{2\pi\sqrt{1-\omega^2}} \, e^{-\frac{1}{2}(y^2 + z^2 - 2\omega yz)/(1-\omega^2)}$$

As a result the right-hand sides of both dynamic equations (60,61) as well as the error (62) can be expressed solely in terms of the two key observables J and ω. We can apparently forget about the details of $\boldsymbol{w}$ and $\boldsymbol{w}^\star$; the whole process can be described at a macroscopic level. Furthermore, due to the Gaussian shape of $P(y, z)$ one can even perform all remaining integrals analytically ! Our main

---

[17]Strictly speaking, for this to be true certain conditions on the vectors $\boldsymbol{w}$ and $\boldsymbol{w}^\star$ will have to be fulfilled, in order to guarantee that the random variables y and z are not effectively dominated by just a small number of their components.
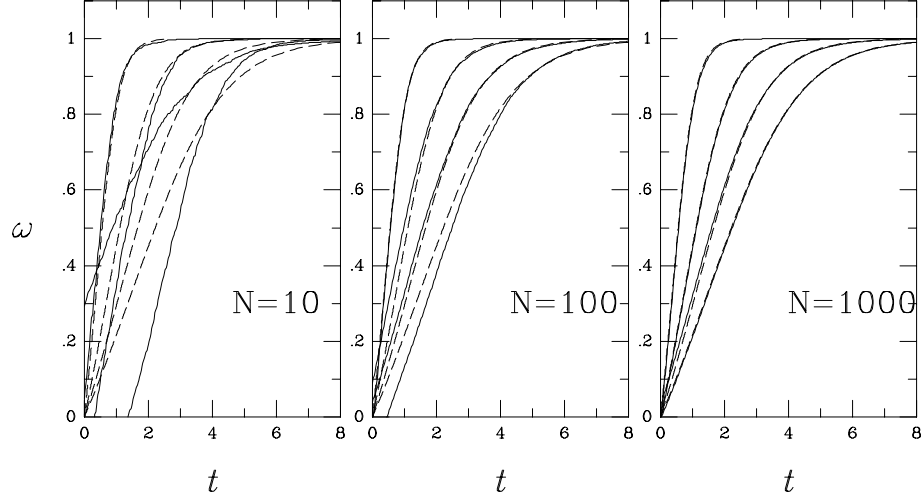
Figure 19: Evolution of the observable ω in a perceptron with learning rate $\epsilon = 0.01/N$ for various sizes N and a randomly drawn teacher vector $\boldsymbol{w}^\star$. In each picture the solid lines correspond to numerical simulations of the perceptron learning rule (for different values of the length $J(0)$ of the initial student vector $\boldsymbol{w}(0)$), whereas the dashed lines correspond to the theoretical predictions (69) for infinitely large perceptrons ($N \to \infty$).

target, the student's error (59) turns out to become

$$E = \frac{1}{\pi} \arccos(\omega) \tag{67}$$

whereas the dynamic equations (60,61) reduce to

$$\frac{dJ}{dt} = -\frac{1-\omega}{\sqrt{2\pi}} \qquad \frac{d\omega}{dt} = \frac{1-\omega^2}{J\sqrt{2\pi}} \tag{68}$$

Stage 4: solve the remaining differential equations

In general one has to resort to numerical analysis of the macroscopic dynamic equations at this stage. For the present example, however, the dynamic equations (68) can actually be solved analytically, due to the fact that (68) describes evolution with a conserved quantity, namely the product $J(1+\omega)$ (as can be verified by substitution). This property allows us to eliminate the observable J altogether, and reduce (68) to just a single differential equation for ω only. This equation, in turn, can be solved. For initial conditions corresponding to a randomly choosen student vector $\boldsymbol{w}(0)$ with a given length $J(0)$, the solution

50

takes its easiest form by writing t as a function of ω:

$$t = J(0)\sqrt{\frac{\pi}{2}} \left\{ \log\left[\frac{1+\omega}{1-\omega}\right]^{\frac{1}{2}} + \frac{\omega}{1+\omega} \right\} \tag{69}$$

In terms of the error E, related to ω through (67), this result becomes

$$t = J(0)\sqrt{\frac{\pi}{8}} \left\{ 1 - \tan^2(\frac{1}{2}\pi E) - 2\log\tan(\frac{1}{2}\pi E) \right\} \tag{70}$$

Examples of curves described by this equation are shown in figure 18. What more can one ask for ? For any required student performance, relation (70) tells us exactly how long the student needs to be trained. Figure 19 illustrates how the learning process in finite perceptrons gradually approaches that described by our $N \to \infty$ theory, as the perceptron's size N increases.

6 Puzzling Mathematics

The models and model solutions described so far were reasonably simple. The mathematical tools involved where mostly quite standard and clear. Let us now open Pandora's box and see what happens if we move away from the nice and solvable region of the space of neural network models.

Most traditional models of systems of interacting elements (whether physical or otherwise) tend to be quite regular and 'clean'; the elements usually interact with one another in a more or less similar way and there is often a nice lattice-type translation invariance[18]. In the last few decennia, however, attention in science is moving away from these nice and clean systems to the 'messy ones', where there is no apparent spatial regularity and where at a microscopic level all interacting elements appear to operate different rules. The latter types, also called 'complex systems', play an increasingly important role in physics (glasses, plastics, spin-glasses), computer science (cellular automata) economics and trading (exchange rate and derivative markets) and biology (neural networks, ecological systems, genetic systems). One finds that such systems have much in common: firstly, many of the more familiar mathematical tools to describe interacting particle systems (usually based on microscopic regularity) no longer apply, and secondly, in analysing these systems one is very often led to so-called 'replica theories'.

6.1 Complexity due to Frustration, Disorder and Plasticity

I will now briefly discuss the basic mechanisms causing neural network models (and other related models of complex systems) to be structurally different from the more traditional models in the physical sciences. Let us return to the relatively simple recurrent networks of binary neurons as studied in section 3, with the neural inputs

$$input_i = w_{i1}S_1 + \ldots + w_{iN}S_N$$

Here excitatory interactions $w_{ij} > 0$ tend to promote configurations with $S_i = S_j$, whereas inhibitory interactions $w_{ij} < 0$ tend to promote configurations with $S_i \neq S_j$. A so-called 'unfrustrated' system is one where there exist configurations $\{S_1, \ldots, S_N\}$ with the property that each pair of neurons can realise its most favourable configuration, i.e. where

$$\text{for all } (i,j): \qquad \begin{cases} S_i = S_j & \text{if } w_{ij} > 0 \\ S_i \neq S_j & \text{if } w_{ij} < 0 \end{cases} \qquad (71)$$

In a frustrated system, on the other hand, no configuration $\{S_1, \ldots, S_N\}$ exists for which (71) is true.

---

[18]i.e. the systems looks exactly the same, even microscopically, if we shift our microscope in any direction over any distance.

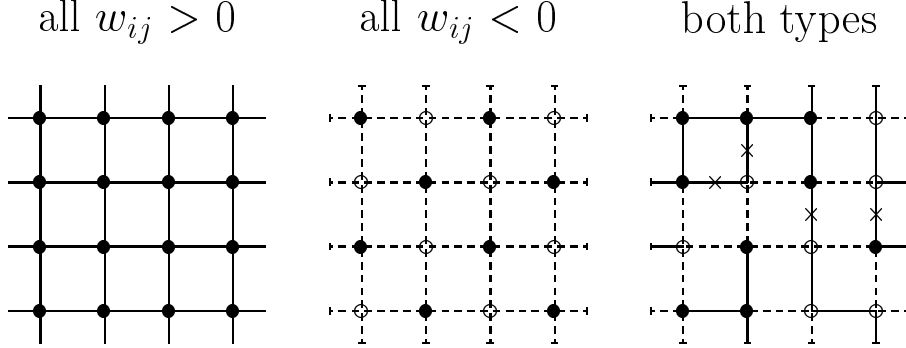all $w_{ij} > 0$ all $w_{ij} < 0$ both types

Figure 20: Frustration in symmetric networks. Each neuron is for simplicity assumed to interact with its four nearest neighbours only. Neuron states are drawn either as • (denoting $S_i = 1$) or as ∘ (denoting $S_i = 0$). Excitatory synapses $w_{ij} > 0$ are drawn as solid lines, inhibitory synapses $w_{ij} < 0$ as dashed lines. An unfrustrated configuration $\{S_1, \ldots, S_N\}$ now corresponds to a way of colouring the vertices such that solid lines connect identically coloured vertices (•• or ∘∘) whereas dashed lines connect differently coloured vertices (•∘ or ∘•). In the left diagram (all synapses excitatory) and the middle diagram (all synapses inhibitory) the network is unfrustrated. The right diagram shows an example of a frustrated network: here there is no way to colour the vertices such that an unfrustrated configuration is achieved (in the present state the four 'frustrated' pairs are indicated with ×).

Let us consider at first only recurrent networks with symmetric interactions, i.e. $w_{ij} = w_{ji}$ for all (i, j) and without self-interactions (i.e. $w_{ii} = 0$ for all i). Depending on the actual choice made for the synaptic strengths such networks can be either frustrated or unfrustrated, see figure 20. In a frustrated network compromises will have to be made; some of the neuron pairs will have to accept that for them the goal in (71) cannot be achieved. However, since there are often many different compromises possible, with the same degree of frustration, frustrated systems usually have a large number of stable or meta-stable states, which generates non-trivial dynamics. Due to the symmetry $w_{ij} = w_{ji}$ of the synaptic strengths it takes at least three neurons to have frustration. In the examples of figure 20 the neurons interact only with their nearest neighbours; in neural systems with a high connectivity and where there is a degree of randomness (or disorder) in the microscopic arrangement of all synaptic strengths, frustration will play an even more important role. The above situation is also encountered in certain complex physical systems like glasses and spin-glasses, and gives rise to relaxation times[19] which are

---

[19] The relaxation time is the time needed for a system to relax towards its equilibrium state, where the values of all relevant macroscopic observables have become stationary.
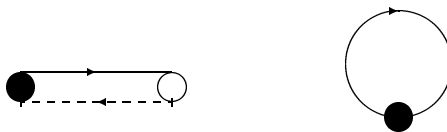
Figure 21: Frustration in non-symmetric networks and networks with self-interactions. Neuron states are drawn as • (denoting $S_i = 1$) or as ∘ (denoting $S_i = 0$). Solid lines denote excitatory synapses $w_{ij} > 0$, dashed lines denote inhibitory synapses $w_{ij} < 0$. An unfrustrated configuration corresponds to a way of colouring the vertices such that solid lines connect identically coloured vertices (•• or ∘∘) whereas dashed lines connect differently coloured vertices (•∘ or ∘•). The left diagram shows two interacting neurons S_1 and S_2; as soon as $w_{12} > 0$ and $w_{21} < 0$ this simple system is already frustrated. The right diagram shows a single self-interacting neuron. If the self-interaction is excitatory there is no problem, but if it is inhibitory we always have a frustrated state.

measured in years rather than minutes (for example: ordinary window glass is in fact a liquid, which takes literally ages to relax towards its crystalline and non-transparent stationary state.)

In non-symmetric networks, where $w_{ij} \neq w_{ji}$ is allowed, or in networks with self-interactions, where $w_{ii} \neq 0$ is allowed, the situation is even worse. In the case of non-symmetric interactions it takes just two neurons to have frustration; in the case of self-interactions even single neurons can be frustrated, see figure 21. In the terminology associated with the theory of stochastic processes we would find that non-symmetric networks and networks with self-interacting neurons fail to have the property of 'detailed balance'. This implies that they will never evolve towards a microscopic equilibrium state (although the values of certain macroscopic observables might become stationary), and that consequently they will often show a remarkably rich phenomenology of dynamic behaviour. This situation never occurs in physical systems (although it does happen in cellular automata and ecological, genetic and economical systems), which, in contrast, always evolve towards an equilibrium state. As a result, many of the mathematical tools and much of the scientific intuition developed for studying interacting particle systems are based on the 'detailed balance' property, and therefore no longer apply.

Finally, and this is perhaps the most serious complication of all, the parameters in a real neural system, the synaptic strengths w_{ij} and the neural thresholds θ_i are not constants, but they evolve in time (albeit slowly) according to dynamical laws which, in turn, involve the states of the neurons and the values of the post-synaptic potentials (or inputs). The problem we face in trying to model and analyse this situation is equivalent to that of predicting what a computer will do when running a program that is continually being

rewritten by the computer itself. A physical analogy would be that of a system of interacting molecules, where the formulae giving the strengths of the inter-molecular forces would change all the time, in a way that depends on the actual instantaneous positions of the molecules. It will be clear why in all model examples discussed so far either the neuron states were the relevant dynamic quantities, or the synaptic strengths and thresholds; but never both at the same time.

In the models discussed so far we have taken care to steer away from the specific technical problems associated with frustration, disorder and simultaneously dynamic neurons and synapses. In the case of the associative memory models of section 3 this was achieved by restricting ourselves to sutuations where the number of patterns stored p was vanishingly small compared to the number of neurons N; the situation changes drastically if we try to analyse associative memories operating at $p = \alpha N$, with $\alpha > 0$. In the case of the topology conserving maps we will face the complexity problems as soon as the set of 'training examples' of input signals is no longer infinitely large, but finite (which is more realistic). In the case of the layered networks learning a rule we find that the naive analytical approach described so far breaks down (i) when we try to analyse multi-layer networks in which the number of neurons K in the 'hidden' layer is proportional to the number of input neurons N, and (ii) when we consider the case of having a restricted set of training examples. Although all these problems at first sight appear to have little in common, at a technical level they are quite similar. It turns out that all our analytical attempts in dealing with frustrated and disordered systems and systems with simultaneously dynamic neurons and synapses lead us to so-called replica theories.

6.2   The World of Replica Theory

It is beyond the scope of this paper to explain replica theory in detail. In fact most researchers would agree that it is as yet only partly understood. Mathematicians are often quite hesitant in using replica theory because of this lack of understanding (and tend to call it 'replica trick'), whereas theoretical physicists are less scrupulous in applying such methods and are used to doing calculations with non-integer dimensions, imaginary times etc. (they call it 'replica method' or 'replica theory'). However, although it is still controversial, all agree that replica theory works.

The simplest route into the world of replica theory starts with the following representation of the logarithm:

$$\log z = \lim_{n \to 0} \frac{1}{n} \left\{ z^n - 1 \right\}$$

For systems with some sort of disorder (representing randomness in the choice of patterns in associative memories, or in the choice of input examples in layered networks, etc.) we usually find in calculating the observables of interest that we end up having to perform an average over a logarithm of an integral (or

sum), which we can tackle using this representation:

$$\int dx \; p(x) \log \int dy \; z(x,y) = \lim_{n \to 0} \frac{1}{n} \left\{ \int dx \; p(x) \left[\int dy \; z(x,y) \right]^n - 1 \right\}$$

$$= \lim_{n \to 0} \frac{1}{n} \left\{ \int dy_1 \cdots dy_n \int dx \; p(x) z(x,y_1) \cdots z(x,y_n) - 1 \right]$$

The variable x represents the disorder, with $\int dx \; p(x) = 1$. We have managed to replace an average of a logarithm of a quantity (which is usually nasty) by the average of integer powers of this quantity. The last step, however, involved making the crucial replacement $[\int dy \; z(x,y)]^n \to \int dy_1 z(x,y_1) \cdots \int dy_n z(x,y_n)$. This is in fact only allowed for *integer* values of n, whereas we must take the limit $n \to 0$! Another route (which turns out to be equivalent to the previous one) starts with calculating averages:

$$\int dx \; p(x) \left\{ \frac{\int dy \; f(x,y) z(x,y)}{\int dy \; z(x,y)} \right\} =$$

$$\int dx \; p(x) \left\{ \frac{\int dy \; f(x,y) z(x,y) [\int dy \; z(x,y)]^{n-1}}{[\int dy \; z(x,y)]^n} \right\}$$

by taking the limit $n \to 0$ in both sides we get

$$\int dx \; p(x) \left\{ \frac{\int dy \; f(x,y) z(x,y)}{\int dy \; z(x,y)} \right\} =$$

$$\lim_{n \to 0} \int dy_1 \cdots dy_n \int dx \; p(x) f(x,y_1) z(x,y_1) \cdots z(x,y_n)$$

Now we appear to have succeeded in replacing the average of the ratio of two quantities (which is usually nasty) by the average of powers of these quantities, by performing a manipulation which is allowed only for integer values of n, followed by taking the thereby forbidden limit $n \to 0$[20].

If for now we just follow the route further, without as yet worrying too much about the steps we have taken so far, and apply the above identities to our calculations, we find in the limit of infinitely large systems and after a modest amount of algebra a problem of the following form. We have to calculate

$$f = \lim_{n \to 0} \text{extr} \; \mathcal{F}[\boldsymbol{q}] \tag{72}$$

---

[20] The name 'replica theory' refers to the fact that the resulting expressions, with their n-fold integrations over the variables y_α (with $\alpha = 1, \dots, n$), are quite similar to what one would get if one were to study n identical copies (or replica's) of the original system.

where $\boldsymbol{q}$ represents an $n \times n$ matrix with zero diagonal elements,

$$
\boldsymbol{q} = \begin{pmatrix}
0 & q_{1,2} & \cdots & q_{1,n-1} & q_{1,n} \\
q_{2,1} & 0 & & & q_{2,n} \\
\vdots & & \ddots & & \vdots \\
q_{n-1,1} & & & 0 & q_{n-1,n} \\
q_{n,1} & q_{n,2} & \cdots & q_{n,n-1} & 0
\end{pmatrix}
$$

$\mathcal{F}[\boldsymbol{q}]$ is some scalar function of $\boldsymbol{q}$, and the extremum is defined as the value of $\mathcal{F}[\boldsymbol{q}]$ in the saddle point which for $n \geq 1$ minimises $\mathcal{F}[\boldsymbol{q}]$[21]. This implies that for any given value of n we have to find the critical values of $n(n-1)$ quantities $q_{\alpha\beta}$ (the non-diagonal elements of the matrix $\boldsymbol{q}$). There would be no problem with this procedure if n were to be an integer, but here we have to take the limit $n \to 0$. This means, firstly, that the number $n(n-1)$ of variables $q_{\alpha\beta}$, i.e. the dimension of the space in which our extremisation problem is defined, will no longer be integer (which is somewhat strange, but not entirely uncommon). But secondly, the number of variables becomes *negative* as soon as $n < 1$! In other words, we will be exploring a space with a negative dimension. In such spaces life is quite different from what we are used to. For instance, let us calculate the sum of squares, as in $\sum_{\alpha \neq \beta = 1}^{n} q_{\alpha\beta}^2$, which for integer $n \geq 1$ is always non-negative. For $n < 1$ this is no longer true. Just consider the example where $q_{\alpha\beta} = q \neq 0$ for all $\alpha \neq \beta$, which gives

$$
\sum_{\alpha \neq \beta = 1}^{n} q_{\alpha\beta}^2 = q^2 n(n-1) < 0 \ !
$$

To quote one of the founding fathers of replica theory: 'The whole program seems to be completely crazy'. Crazy or not, if we simply persist and perform all calculations required, accepting the rather strange objects we find along the way as they are, we end up with results which are, as far as the available evidence allows us to conclude, essentially correct.

The key to success is not to try to calculate individual matrix elements $q_{\alpha\beta}$, but to concentrate wherever possible in the calculation on quantities that (at least formally) have a well-defined $n \to 0$ limit, such as $P(q)$, which is defined as the relative frequency with which the value $q_{\alpha\beta} = q$ occurs among the non-diagonal entries of the matrix $\boldsymbol{q}$. Since $q \in \Re$ this function becomes a probability density:

$$
P(q)dq = \frac{\text{number of entries with } q - \frac{1}{2}dq < q_{\alpha\beta} < q + \frac{1}{2}dq}{\text{total number of entries}}
$$

with $0 < dq \ll 1$. This quantity remains well-behaved. It will obey the relations $P(q) \geq 0$ and $\int dq \, P(q) = 1$, whatever the value of the dimension n; for $n < 1$ the minus sign generated by the denominator will be cancelled by a similar minus sign in the numerator. The problem in (72) of finding a saddle-point $\boldsymbol{q}$

---

[21] This version of the saddle-point problem is just the simplest one; in most calculations one finds several additional $n \times n$ matrices and n-vectors to be varied.

can be translated into one which is formulated in terms of the $n \to 0$ limit of the function $P(q)$ only:

$$f = \text{extr } \mathcal{G}[\{P(q)\}] \qquad (73)$$

Although at a technical level non-trivial, compared to (72) the problem (73) is conceptually quite sane; now one has to explore the (infinite-dimensional) space of all probability distributions, as opposed to a space with a negative dimension. Later it was discovered that the function $P(q)$ for which the extremum in (73) occurs can be interpreted in terms of the average probability of two identical copies A and B of the original system to be in a state with a given mutual overlap dfined by q. For instance, for the associative memory networks of section 3, storing $p = \alpha N$ patterns (with $\alpha > 0$) we would find:

$$P(q)dq = \text{average probability of } q - \frac{1}{2}dq < \frac{1}{N}\sum_{i=1}^{N} S_i^A S_i^B < q + \frac{1}{2}dq \qquad (74)$$

whereas for the perceptron feasability calculations of section 5.3 we would find:

$$P(q)dq = \text{average probability of } q - \frac{1}{2}dq < \sum_{i=1}^{N} \frac{w_i^A w_i^B}{|\boldsymbol{w}^A||\boldsymbol{w}^B|} < q + \frac{1}{2}dq \qquad (75)$$

with $0 < dq \ll 1$. This leads to a convenient characterisation of complex systems. For non-complex systems, with just a few stable/metastable states, the quantity $P(q)$ would be just the sum of a small number of isolated peaks. On the other hand, as soon as our calculation generates a solution $P(q)$ with continuous pieces, it follows from (74,75) that the underlying system must have a huge number of stable/metastable states, which is the fingerprint of complexity.

Finally, even if we analyse certain classes of neural network models in which both neuron states and synaptic strengths evolve in time, described by coupled equations, but with synapses changing on a much larger time-scale than the neuron states, we find a replica theory. This in spite of the fact that there is no disorder, no patterns have been stored, the network is just left to 'program' its synapses autonomously. However, in these calculations the parameter n in (72) (the replica dimension) does not necessarily go to zero, but turns out to be given by the ratio of the degrees of randomness (noise levels) in the two dynamic processes (neuronal dynamics and synaptic dynamics).

It appears that replica theory in a way constitutes the natural description of complex systems. Furthermore, replica theory clearly works, although we do not yet know why. This, I believe, is just a matter of time.

7 Further Reading

Since the present paper is just the result of a modest attempt to give a taste of the mathematical modelling and analysis of problems in neural network theory, by means of a biased selection of some characteristic solvable problems, and without distracting references, much has been left out. In this final section I want to try to remedy this situation, by briefly discussing research directions that have not been mentioned so far, and by giving references. Since I imagine the typical reader to be the novice, rather than the expert, I will only give references to textbooks and review papers (these will then hopefully serve as the entrance to more specialised research literature). Note, however, that due to the interdisciplinary nature of this subject, and the inherent fracturisation into sub-fields, each with their own preferred library of textbooks and papers, it is practically impossible to find textbooks with sketch a truely broad and impartial overview.

There are by now many books to serve as introductions to the field of neural computing, such as [1, 3, 2, 4, 5]. Most give a nice overview of the standard wisdom around the time of their appearance[22], with differences in emphasis depending on the background disciplines of the authors (mostly physics and engineering). A taste of the history of this field can be provided by one of the volumes with reprints of original articles, such as [6] (with a stronger emphasis on biology/psychology), and the influential book [7]. More specialised and/or advanced textbooks on associated memories and topology conserving maps are [8, 9, 10, 11]. Examples of books with review papers on more advanced topics in the analysis of neural network models are the trio [12, 13, 14], as well as [15]. A book containing review chapters and reprints of original articles, specifically on replica theory, is [16].

One of the subjects that I did not go into very much concerns the more accurate modelling of neurobiology. Many properties of neuronal and synaptic operation have been eliminated in order to arrive at simple models, such as Dale's law (the property that a neuron can have only one type of synapse attached to the branches of its axon; either exitatory ones or inhibitory ones, but never both), neuromodulators, transmission delays, genetic pre-structuring of brain regions, diffusive chemical messengers, etc. A lot of effort is presently being put into trying to take more of these biological ingredients into account in mathematical models, see e.g. [13]. More general references to such studies can be found by using the voluminous [17] as a starting point.

A second sub-field entirely missing in this paper concerns the application of theoretical tools from the fields of computer science, information theory and applied statistics, in order to quantify the information processing properties of neural networks. Being able to quantify the information processed by neural systems allows for the efficient design of new learning rules, and for making comparisons with the more traditional information processing procedures. Here

---

[22] This could be somewhat critical: for instance, most of the models and solutions described in this paper go back no further than around 1990, the analysis of the dynamics of on-line learning in perceptrons is even younger.

a couple of suitable introductions could be the textbooks [18] and [19], and the review paper [20], respectively.

Finally, there are an increasing number of applications of neural networks, or systems inspired by the operation of neural networks, in engineering. The aim here is to exploit the fact that neural information processing strategies are often complementary to the more traditional rule-based problem-solving algorithms. Examples of such applications can be found in books like [21, 22, 23].

Acknowledgement

It is my pleasure to thank Charles Mace for helpful comments and suggestions.

References

[1] D. Amit, **Modeling Brain Function**, Cambridge U.P., 1989

[2] B. Müller and J. Reinhardt, **Neural Networks, an Introduction**, Springer (Berlin), 1990

[3] J. Hertz, A. Krogh and R.G. Palmer, **Introduction to the Theory of Neural Computation**, Addison-Wesley (Redwood City), 1991

[4] P. Peretto, **An Introduction to the Modeling of Neural Networks**, Cambridge U.P. 1992

[5] S. Haykin, **Neural Networks, A Comprehensive Foundation**, Macmillan (New York), 1994

[6] J.A. Anderson and E. Rosenfeld (eds.), **Neurocomputing, Foundations of Research**, MIT Press (Cambridge Mass.), 1988

[7] M.L. Minsky and S.A. Papert, **Perceptrons**, MIT Press (Cambridge Mass.), 1969

[8] T. Kohonen, **Self-organization and Associative Memory**, Springer (Berlin), 1984

[9] Y. Kamp and M. Hasler, **Recursive Neural Networks for Associative Memory**, Wiley (Chichester), 1990

[10] H. Ritter, T. Martinetz and K. Schulten, **Neural Computation and Self-organizing Maps**, Addison-Wesley (Reading Mass.), 1992

[11] T. Kohonen, **Self-organizing Maps**, Springer (Berlin), 1995

[12] E. Domany, J.L. van Hemmen and K.Schulten (eds.), **Models of Neural Networks I**, Springer (Berlin), 1991

[13] E. Domany, J.L. van Hemmen and K.Schulten (eds.), **Models of Neural Networks II**, Springer (Berlin), 1994

[14] E. Domany, J.L. van Hemmen and K.Schulten (eds.), **Models of Neural Networks III**, Springer (Berlin), 1995

[15] J.G. Taylor, **Mathematical Approaches to Neural Networks**, North-Holland (Amsterdam), 1993

[16] M. Mezard, G. Parisi and M.A. Virasoro, **Spin-Glass Theory and Beyond**, World-Scientific (Singapore), 1987

[17] M.A. Arbib (ed.), **Handbook of Brain Theory and Neural Networks**, MIT Press (Cambridge Mass.), 1995

[18] M. Anthony and N. Biggs, **Computational Learning Theory**, Cambridge U.P., 1992

[19] G. Deco and D. Obradovic, **An Information-theoretic Approach to Neural Computing**, Springer (New-York), 1996

[20] D.J.C. MacKay, *Probably Networks and Plausible Predictions - a Review of Practical Bayesian Methods for Supervised Neural Networks*, **Network** 6, 1995, p. 469

[21] A.F. Murray (ed.), **Applications of Neural Networks**, Kluwer (Dordrecht), 1995

[22] C.M. Bishop, **Neural Networks for Pattern Recognition**, Oxford U.P., 1995

[23] G.W. Irwin, K. Warwick and K.J. Hunt (eds.), **Neural Network Applications in Control**, IEE London, 1995