UNIVERSITY OF LONDON
External System

Goldsmiths
UNIVERSITY OF LONDON

BSc and Diploma in
Computing and Related Subjects

Neural networks

A. Vella and C. Vella

2009

2910**311**

The material in this subject guide was prepared for the University of London External System by:

Dr Alfred D. Vella, BSc, PhD, MSc, BA, FIMA, MBCS, CEng, CMath

and

Carol A. Vella, BSc, MSc, BA, PGCert. T&L in HE, PGDip. T&L in HE.

The External System
Publications Office
University of London
32 Russell Square
London
WC1B 5DN
United Kingdom

Website: www.londonexternal.ac.uk

Published by: University of London Press

© University of London 2009

Printed by: Central Printing Service, University of London, England

Contents

Notes

Chapter 1

Introduction

1.1 About this half unit

Aims

The aim of this half unit is to introduce you to the exciting topic of neural networks, which are often called artificial neural networks (ANNs) to distinguish them from natural neural networks (such as the brain). Although this is quite an old topic as far as computing subjects go, many believe that there remain important discoveries to be made – these are just waiting for the right person to come along. Perhaps that person is you!

While this half unit will not make you an expert in neural networks, it will introduce you to many of the important concepts and techniques in this area of computing, the major types of ANN and also some of the literature and other resources that are freely available on the internet.

We hope it will also encourage you to explore this area further, and perhaps to contribute to future developments or new applications.

Objectives

The objectives of this subject guide are to:

- enable students to understand important concepts and theories of artificial neural networks (ANNs)
- enable students to understand how ANNs can be designed and trained
- enable students to calculate simple examples of ANNs
- give students an appreciation of some of the limitations and possibilities of ANNs.

Learning outcomes

By the end of this half unit, and having completed the relevant readings and activities, you should be able to:

- describe various types of ANNs
- carry out simple simulations of ANNs
- discuss the theory on which ANNs are based
- explain how simple ANNs can be designed
- explain how ANNs can be trained
- discuss the limitations and possible applications of ANNs.

Coursework

Coursework will be designed to:

- encourage students to consolidate their learning by practising simulating simple neural networks by hand, by programming and/or by using readily available software, and to present their results appropriately
- encourage students to read more widely than the subject guide and set textbook and to evaluate and summarise their findings.

Essential reading

The set text for this half unit is Rojas, R. (1996) *Neural networks: a systematic introduction.* (Berlin: Springer-Verlag, 1996) [ISBN 3540605053; 978354060508]. It is available online at www.inf.fu-berlin.de/inst/ag-ki/rojas_home/documents/1996/NeuralNetworks/neuron.pdf (Last accessed June 2009.)

We agree with a review of the book on the Association for Computing Machinery's website which says: 'If you want a systematic and thorough overview of neural networks, need a good reference book on this subject, or are giving or taking a course on neural networks, this book is for you.'

References to Rojas will take the form r3.2.1 for Section 2.1 of Chapter 3 or rp33 for page 33 of Rojas (for example) – you should have no difficulty interpreting this. Do try to read this valuable resource – even if you are not told to do so directly. Also, if the mathematics is too much for you on first reading, it might be worth having a skim through it and returning to it again later.

Further reading

Complementary resources include:

Kröse, Ben J.A. and P. Patrick van der Smagt *An introduction to neural networks*. (Amsterdam: University of Amsterdam, 1996). Available online at: http://math.uni.lodz.pl/~fulmanp/zajecia/nn/neuro-intro.pdf (Last accessed June 2009.)

MacKay, D.J.C. *Information theory, inference and learning algorithms*. (Cambridge: Cambridge University Press, 2005) [ISBN-13: 9780521642989; ISBN-10: 0521642981]. Available online at: www.inference.phy.cam.ac.uk/mackay/itila/book.html) (Last accessed June 2009.)

1.2 About this guide

This guide sets out some of the important concepts in neural networks, and guides you through important further reading from the set text Rojas (1996).

Many people working with ANNs like to take shortcuts by using mathematical arguments and proofs. Do not be put off by these. There are some formulae that you need to remember but the derivations are not part of our study – read them 'lightly'.

This guide is written in a fairly informal style. Some use of formal notation is necessary, but this is introduced as appropriate. Some mathematical ideas are also introduced, but we have tried to avoid too much maths for maths' sake (for the benefit of students who may not have much mathematical background). An understanding of mathematics equivalent to the Level 1 mathematics courses for the Computing and Other Related Subjects and Mathematics, Statistics and Computing degree programmes is assumed. In particular, simple algebraic manipulation of vectors and matrices is important, and you may like to refresh your memory of these topics.

For those students interested in exploring the mathematics in more detail, we indicate the sections in the set text where a fuller account can be found.

This guide is not intended to be a comprehensive account or a textbook, but to introduce you to some important concepts and to help you explore these in more depth by guided reading of the set text and selected internet resources.

You may also need to research particular topics in more depth as part of the coursework. When doing so, you should carefully evaluate online resources in terms of their relevance, timeliness, objectivity and authoritativeness to make sure the information they contain is reliable and useful. Remember too that online resources must be referenced just as carefully as print resources.

How to use this guide

Each chapter of this guide starts with:

- a list of learning outcomes
- a list of essential reading
- a list of further reading where appropriate.

The learning outcomes should be used as a guide for your skills development. Use them to focus your learning.

The essential reading aims to act as a warning of what reading you will be asked to do as you study the material of the chapter. You are advised to make sure that the material is easily accessed but do not read the material until told to do so in the text. Of course we encourage you to read the 'rest' of the material when you have time.

It is good practice to review the material again once you have worked through the chapter and at least once again when you are revising for the examination.

Further reading, if any is given, is best studied after working through the chapter unless otherwise directed in the text. After studying a chapter it is also worth using Google or another search engine to see what other information on the topic is available. Lots of material finds its way onto the web each week.

1.3 Materials on the CD-ROM

Accompanying this guide is a CD-ROM that contains Excel spreadsheets that implement some of the models that you will study. The use of the spreadsheets is explained in Section D of the document on the CD-ROM entitled *Guide to the CD-ROM*. This document also includes:

- an introduction to the CD-ROM (Section A)
- sample answers to the exercises in the guide (where appropriate) (Section B)
- a note on simulating neural networks (Section C)
- a section on writing up coursework (Section E)
- a section on examinations, including advice on answering Sample examination questions (Section F)
- a section on evaluating ANNs (Section G)
- a list of some freely available data sets that you might like to use in your work (Section H)
- a copy of the half unit's syllabus (Section I)
- a list of recommended readings (Section J)
- a list of useful resources (Section K).

The best way of using the CD-ROM is to become familiar with its contents and then to consult this material when advised to by this text, or when you feel confident with its contents, or when you think that it might be useful.

1.4 Recommendation on study time

As p.35 of the CIS and CC *Handbook* [available online at http://www.londonexternal.ac.uk/current_students/general_resources/handbooks/cis_cc/ciscc_comphb.pdf] says: 'To be able to gain the most benefit from the programme, and hence do well in examinations, it is likely that you will have to spend at least 250 hours studying for each full unit, although you are likely to benefit from spending up to twice this time.' So we would expect you to spend between 125 and 250 hours in total on study in order to do justice to this half unit subject.

1.5 Examinations

Advice on revising

Experience of teaching many hundreds of students suggests that two very important strategies for success when studying neural networks are:

1. Read widely.
2. Practise by doing lots of examples.

This guide will lead you through the important topics in artificial neural networks and will give you the most important results. However, we learn best when we become familiar with a topic and this familiarity is aided by being presented with many views of the same topic. That is why we have tried to find many sources on the web. Some of the books that we have recommended to you are also freely available on the web. You should use them!

Neutral networks is a practical topic in which the computer is made to perform many, many calculations. However, it is only by doing calculations by hand on some easy examples that we can become confident that we understand what we are asking the computer to do.

The examination format has not changed with the revision of this guide. You are expected to answer four questions out of six in 2 hours 15 minutes. The CD-ROM contains a document, entitled *Guide to the CD-ROM*, which contains advice on answering the examination questions (see Section F).

Important: The information and advice given in the above section are based on the examination structure used at the time this guide was written. We strongly advise you to always check the current *Regulations* for relevant information about the examination. You should also carefully check the rubric/instructions on the paper you actually sit and follow those instructions.

A reminder of your learning outcomes

By the end of this half unit, and having completed the relevant readings and activities, you should be able to:

- describe various types of ANNs
- carry out simple simulations of ANNs
- discuss the theory on which ANNs are based
- explain how simple ANNs can be designed
- explain how ANNs can be trained
- discuss the limitations and possible applications of ANNs.

Chapter 2

Motivation for artificial neural networks

Learning outcomes

By the end of this chapter, and having completed the relevant readings and activities, you should be able to:

- explain the motivations for studying ANNs
- explain the basic structure, working and organisation of biological neurons
- interpret diagrams of units such as Figure 2.2b and diagrams containing many connected units
- define the terms: **unit**, **weight**, **activation function**, ***activation***, ***net***, **threshold**, ***bias***, **fires** and **architecture** as they relate to ANNs.
- understand and explain the notational equivalence of: *?a*, *?b…*; $input_1$, $input_2$… and x_1, x_2…
- understand and explain the use of the following symbols: w_{iu}, w_{uv}, Σ, $A()$, a_u, $T(<0,0,1)$.

Essential reading

Rojas 1.1, 1.2 and 1.3. However, do not read these sections until prompted to do so by the text.

Farabee, M. (2009) *The nervous system*. Available online at: www.estrellamountain.edu/faculty/farabee/biobk/BioBookNERV.html (Last accessed June 2009.)

Clabaugh, C., D. Myszewski and J. Pang (2009) *Neural networks*. Available online at: http://cse.stanford.edu/class/sophomore-college/projects-00/neural-networks/Files/presentation.html (Last accessed June 2009.)

Carnegie Mellon (2009) *Center for the Neural Basis of Cognition*. Available online at: www.cnbc.cmu.edu (Last accessed June 2009.)

Further reading

Kleinberg, J. and C. Papadimitriou *Computability and Complexity in Computer Science: Reflections on the Field, Reflections from the Field*. (Washington, D.C.: National Academies Press, 2004) Available online at: www.cs.cornell.edu/home/kleinber/cstb-turing.pdf (Last accessed August 2009.)

2.1 Introduction

Well before the era of computing machines, people wondered about the nature of intelligence and whether it would be possible to make machines that displayed 'intelligence'. It is from these questions that our subject emerges in the 1940s.

As its title indicates, this subject guide is about 'neural networks' or NNs: that is, networks of 'neurons'. Although much of the time we will be considering networks, the choice of the name 'neuron' which was made many years ago, is a little unfortunate – although well meaning. Like many in the field we prefer to call the items in our networks 'units' and to use the acronym ANN to emphasise that these are networks of 'artificial neurons'.

Neural networks are not a new idea and in fact their history can be traced back to 1943, long before that of Artificial Intelligence (or AI, a term which was coined around 1952). Alan Turing had spent much time thinking about computation and what could – and what could not – be computed by 'automatic machines'; which (at least in his early work) he probably thought of as humans obeying written step-by-step instructions. Later he spent much time during the Second World War attempting to make machines do tasks with which the brightest minds in the land were having difficulty! As humans were the 'systems' where intelligence, at least as it was then understood, was most evident, it seems natural that Turing should wonder if mankind's thinking capabilities could be simulated on the new breed of machines that were being built after the war. His thoughts therefore turned to making machines appear more intelligent – that is, he thought about what has now come to be called Artificial Intelligence. Artificial neural networks can be thought of as one of many 'biologically inspired' attempts at AI.

To understand fully the motivation for ANNs, we need to look at the motivation for Artificial Intelligence itself.

There are three equally valid reasons for studying Artificial Intelligence and thus ANNs:

1. To make better machines.

   Turing was interested in making machines more useful – this is what one might call the 'engineer's rationale'. After all, he had spent time during the Second World War trying to decipher intercepted messages with the aid of the immediate precursors of computers.

2. To model human/animal intelligence.

   Another, equally valid reason is an attempt to understand better the ways that the intelligent behaviour of humans might arise from a processing point of view – this might be termed the 'psychologist's rationale'. Turing showed his interest in this study – possibly as a means of informing attempts at 1. above. Understanding this process might allow us to help humans with difficulties and also might feed back to goal 1. above.

3. To explore the idea of intelligence.

   Finally, we might broaden our view of intelligence to include intelligent behaviour of other primates, or more broadly other mammals – perhaps whales and dolphins, among others. Going beyond this to the extreme, we might ask 'what forms could intelligence have?' – this might be the 'philosopher's rationale'. There is absolutely no reason why intelligence cannot exist in forms almost unrecognisable to us. In fact, there is a whole industry of people asking if machines could have mind, consciousness, etc. You might meet this if you study AI.

Once you have studied this subject we hope that you will have at least some idea of how artificial neural networks might go some way towards each of these three goals, but much of the guide concentrates just on Goal 1. above – making better machines.

2.2 Historical perspective

Turing had shown that it was possible to design 'universal calculating machines'; that is, machines which could, at least in principle, calculate anything that was theoretically computable. For more on what is or is not computable see Kleinberg and Papadimitriou (2004) and the references that they cite. It is also worth reading Rojas sections 1.1.2 and 1.1.3 at this stage.

However, Turing knew that there was a big difference between being able to do something 'in principle' and actually doing it. He had made many hand simulations of his 'universal machines' and from this experience understood that 'programming' them could be (at the very least) extremely tedious.

So Turing thought about the possibility of machines that could be taught to do things by **experience** rather than by **instruction**. It is this ability that sets neural networks apart from many other branches of computing.

We can summarise the goals of Artificial Neural Network research as:

The study of networks of simple processing elements and of what can and cannot be done with them especially in the context of the three goals 1., 2. and 3. mentioned above.

However, we do not have the space (and you do not have the time) to consider in detail all three goals!

2.3 ANN Goals, tools and techniques

The last section included a one-sentence summary of the goals of our study (artificial neural networks) but we need to focus a little more.

Despite its age, the science of artificial neural networks continues to progress, with new techniques and applications being developed all the time. This is both a strength and a weakness of the subject. It is a strength because there continues to be a healthy interest among 'amateurs' and a weakness because it is difficult to settle upon those fundamental principles of the subject that are likely to be around in (say) 50 years' time. Instead we are forced to give a snapshot of a few of the many types of systems that go under the name 'artificial neural network' and try to give some principles that underlie at least some of them.

A key issue in the use of computers to solve problems is the presentation of potential or candidate solutions to those problems in a form that is amenable to efficient representation on a computer. By manipulating such **candidate solutions** it is hoped that we can find one that will serve our purpose.

Artificial neural networks (ANNs) are representations of computing problems in the form of one or more networks of processing elements, historically called 'neurons' – hence the name 'neural network'. However, nowadays the term **unit** is preferred to distance these from the biological neurons in our brains. As our understanding of the latter improves, we see just how different the biological neuron is from the artificial units in our networks.

Different problems have different representations, but some representations are powerful enough to represent all problems – known as **universal representations**. Whether they do this efficiently or not is another issue.

So, for example, although we know that almost any general purpose programming language will do for any computing problem, we also know that some problems lend themselves well to particular languages. If you have studied a range of languages (declarative, functional, procedural and object oriented) such as Prolog, SML, Pascal and Java or similar languages, you will know what we mean.

In general, the types of problem that we try to solve using ANNs are those that are very difficult to solve otherwise. This is what makes it worthwhile putting the effort into finding new ways of solving the problems – there are no real alternatives known.

Those used to programming will know that finding a ‘good’ way of representing our problem is a key step in its solution. Once we have found such a representation we use this with a set of data structures and algorithms – possibly in the form of objects, their members and their methods.

However, some problems, although reasonably easy to state (and represent), are very difficult to solve because we cannot find a suitable algorithm. When finding algorithms proves too difficult, we may take another line of attack: perhaps we can represent our problems as search problems where we might use some well known search techniques. We will see later how the use of ANNs can be thought of as a type of search.

The ways in which the Artificial Intelligence community has tackled these problems can be classified into ‘Knowledge Rich Strategies’ and ‘Knowledge Poor Strategies’. In the Knowledge Rich strategies, some of which you may learn about in the half unit on AI, we build into our computer models as much knowledge about the problem domain as we can. The language used in Expert Systems, for example, is that of experts. In contrast with this, the knowledge poor strategies concentrate on general techniques that do not depend upon the particular problem at hand. This enables us to refine the techniques in the hope and expectation that they are applicable widely rather than to a narrow set of applications.

There is much competition between the camps of researchers who prefer one strategy type over the other and sometimes this rivalry has had some negative effects on progress. However, it is not our intention to say any more about that here.

2.4 Natural versus artificial neural networks

The biology of physical neurons within the nervous system provided the original inspiration for artificial neural networks. In fact, ANNs were originally an attempt to explain how our neurons worked. Of course, once the power of computing with ANNs was appreciated, a large section of the neural network community moved on to investigate modifications of the original neural models in order to optimise their computational possibilities, with little or no attempt at preserving any plausible connection with natural neurons. In fact, there was very little plausibility even in the first neural networks.

Natural neurons are activated, generating an electrical signal, if the received stimulus is above a certain threshold. This is sometimes referred to as the neuron ‘firing’. The electrical signal is transmitted along the neuron, and may be passed on to other neurons which are connected, and these in turn may either ‘fire’ or not, depending on whether the signal received is above the threshold of the receiving neuron.

A neuron is either activated, or it is not: there are no partial activations.

It is easy to see that a neuron activating or not activating can be modelled within a binary system, with *1* representing activation and *0* representing non-activation. You may have noticed that we have used 'activation' and 'non-activation' above. We did this to avoid later confusion, because unfortunately, within artificial neural networks, the term 'firing' is applied whenever the calculation takes place whether the outcome is *1* or *0*. So don't be caught out!

The threshold (or –bias) idea is also readily modelled within artificial neural networks.

Artificial neural networks do not approach the complexity of natural nervous systems, but can still be usefully applied to solve certain classes of problem. On this topic it would be worth reading Rojas section 1.1.1.

Exercise 1

At this stage it is worth you looking at a couple of useful resources that are available on the web.

A website that has a good summary of the biological aspects of nerves is Farabee (2009). Navigate to it and read up to '*Divisions of the Nervous System*'.

Next, visit Stanford University's website Clabaugh (2009) which has a short tutorial on ANNs. Have a quick look at this now – and maybe again after you have studied all of this half unit.

While reading these sites, make brief notes about the main properties of natural neurons.

We will leave it up to you to decide just how alike natural and artificial neural networks are – but we might ask you for your opinion, with reasoning, as part of an examination or assignment question.

The University of Pittsburgh and Carnegie Mellon have set up the 'Center for the Neural Basis of Cognition: Integrating the sciences of mind and brain'. This site Carnegie Mellon (2009) is definitely worth looking at.

Reading

Now read Rojas Section 1.2, which covers much of this topic in more detail.

2.5 Representations of neural networks

As the name 'neural network' implies, we are studying the potential of networks of **neurons** – so a good place to start might be with an idea of what 'a neuron' might be. We do, however choose to use the term **unit** rather than neuron. At its simplest we have an 'object' with **inputs** and **outputs** and which processes the inputs to produce these outputs. This model is much like that of the computer itself.

A neural network is a network of 'simple' computing units – in a sense that is all that they have in common. Assuming that you do the reading and web work recommended, you will have met many different but by no means all neural network types after studying this half unit.

Some authors use a rectangular box, of no particular size to represent a unit. Figure 2.1 shows such a box.

We will use the letter *u* to denote a particular unit. Three lines meet the box in the figure and these lines, or edges, represent a means of this unit's communication with other units and with the outside world. Communication often takes the form of an integer or real number being passed, although sometimes we think of other forms of message. Some units have **directed** edges: that is, the messages (sometimes integers or sometimes real numbers) go just one way, while other types of unit can have two-way edges. The unit in the figure has one edge going in, one going out and one bidirectional edge.

Each incoming edge has a numeric **weight** (*w* say), associated with it. This may be an integer or it may be a real value. We imagine an input being sent along the edge and this input is 'magnified' or amplified by the weight so that an input of x arrives as $w*x$ at the node. Most of the time we draw the diagrams so that messages go from left to right but there is no requirement for this convention and sometimes we may break it. Edges are named by their start and end unit (if they have them) and we often use the same notation for edges and their associated weights. Thus an edge going from unit u to unit v will have weight w_{uv}. There is no significance placed upon the length of the edges and although we might later say that the diagram **shows** a unit, we must always remember that this is just a representation for humans!

In an ANN, besides the messages that are passed between units, we often have messages coming from outside along edges with weights w_{iu} signifying from input i to unit u. In this guide we use a special notation for inputs. Instead of following most authors and labelling them $input_0$, $input_1$, … etc. or something similar, we believe that it is clearer if we use *?a*, *?b*, etc. for the first, second etc… inputs. However, following other authors we might sometimes use x_1, x_2… or even something else if this makes things clearer.

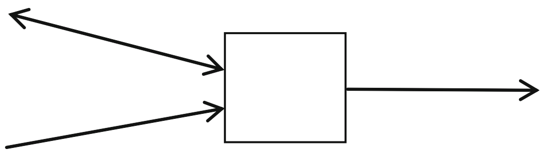


Figure 2.1: A common representation of a unit.

Once the magnified inputs arrive at the unit, it performs a calculation on these. This calculation often has two parts. For historic reasons we call the result of the first part the **net** of the unit. The second part of the calculation then works on the unit's value of **net** to produce the unit's **activation a**.

For the first part of the calculation, the most common units add the weighted inputs to form a **sum** which is the value of **net**. We will use the symbol Σ for **net** in such cases. So it is this sum that is passed through the **activation function**, *A()*, to produce the output which we represent using the letter $a = A(\Sigma)$. There are many functions that have been used as activation functions, and we shall meet a few.

In general, there may be any number of input edges and any number of output edges. However, while the values input can be different, there can only be one value of output, although it can exit on many arrows. The output or *activation* of unit u is labelled a_u.

2.6 Notation specific to this guide

While you will see lots of variations on the diagram above during your reading, some with round 'boxes' for example, we do not find them so informative. One of the frustrating aspects of working in neural networks is the plethora of notations used. It sometimes seems as if authors do not take the trouble to find out what others have done and just invent their own notation. It may be that with computer power becoming readily available just as neural networks became more widely explored, the notation has not yet had time to settle properly.

Unfortunately, we feel the need to add to this plethora of notation – for a good reason – that of the ease with which we can communicate with our readers here in this guide and more importantly in coursework and examinations.

For learning how single units work, it is good to implement them as spreadsheet cells. To help you to see how this might work, we introduce in this section a notation that is useful for networks with just a few units. The notation is not standard but is much easier to wordprocess than any other that we have seen.

This notation for a unit is given in Figure 2.2a and Figure 2.2b. It is not as useful though when there are lots (more than 10 say) of units.

Figure 2.2a shows that we have three inputs. The first is fixed and is just a notational convenience allowing for the bias. We use bias in our calculations: it is a property of a unit that may be given or learnt. The other inputs *?a* and *?b* will receive values either from another unit or from outside the neural net. We have question marks for the weights as they are not given values in this diagram. Different types of unit will have different parameters so this is just a typical one. We will discuss the use of the learning rate later when we cover training. *Net* represents the first of two calculations that the unit does on its inputs. The Σ indicates that it forms the sum of its weighted inputs. In this case we write:

$$net_u = bias_u + ?a\ w_{au} + ?b\ w_{bu}$$

The second calculation takes net_u (which is Σ_u in this case) and produces a value for the activation a_u of the unit. One of the simplest activation functions is the **step** or threshold function:

$$A(\Sigma_u) = [if\ \Sigma_u \geq 0\ then\ 1\ else\ 0]$$

When the activation function is of this form we write $A(\Sigma) = T(<0,0,1)$ which stands for: a_u *= If net < 0 then 0 else 1*. That is if the value of *net* is less than zero the activation is *0* otherwise the activation is *1*. This is called a **threshold** function and the unit a 'threshold unit' because of this. This notation for threshold relates to the way that it would be implemented in Excel, for example. Note that *T(<0,0,1)* is the same as *T(≥0, 1, 0)* but the former is often easier to write in a wordprocessor).

The name **bias** comes from the fact that it biases the value of **net**.

Inputs	Weights	Parameters	Form	Value
1	*bias*	Learning rate	η	**??**
?a	?	Net	Σ	??
?b	?	Activation	T(<0,0,1)	??

Figure 2.2a: A skeleton of a unit with ? showing items that need filling in.

Inputs	Weights	Parameters	Form	Value
1	–0.1	Learning rate	η	**0.15**
–1	0.3	Net	Σ	–0.65
–1	0.25	Activation	T(<0,0,1)	0

Figure 2.2b: Some typical values inserted into Figure 2.2a.

In Figure 2.2b we have put in some values for the inputs and the weights.

As net_u $= bias_u + ?a\ w_{au} + ?b\ w_{bu}$
$= –0.1 + –1*0.3 \quad + –1 * 0.25$
$= –0.1 – 0.3 – 0.25$
$= –0.65$ as shown to the right of Σ in the figure,

and since $–0.65 < 0$, the value of the activation (and thus the output of the unit) is zero. This again is shown to the right of the expression for the activation. We call the process of a unit doing these calculations 'firing' and we say that the unit **fires** when it produces its output – even if that output is zero. This diverges from the terminology used in biological neurons, where a neuron is said to 'fire' only if it produces an electrical impulse.

When calculating with only a few units we will present units in this form.

You may be surprised to realise that we have now introduced you to three different representations of units. One easy to draw, as in Figure 2.1; the next, rather more formal one, is useful when working in Excel or Word because it shows how the calculations should be performed – this is shown in Figure 2.2; and finally the most formal – the mathematics. You will need to be able to work with all of these.

We will come across a number of different types of unit and a number of different calculations for *net* and *activation*.

Many, if not most, authors use units to represent inputs. Thus when they draw a 'two-layer network' they actually draw three layers – an input layer and layers 1 and 2. We do not do this. Any units that we draw are 'real' units.

Another notational convenience that many authors use is the inclusion of a 'unit 0' and an 'input 0'. This is just a shorthand way of taking into account any biases that the other units have and although we do not draw these, we do use the index zero in mathematical formulae to include biases where they are present. This is especially important when we train units as the biases need to be learnt too. So do not forget to update biases along with the other weights.

One final issue that we need to mention is that when specifying an ANN we need to give its **architecture**. That is, we need to specify the number and types of units, how they are arranged and connected, as well as the algorithms used for calculation and for learning.

Reading

See Rojas Section 1.3, which gives another view of this topic.

A reminder of your learning outcomes

By the end of this chapter, and having completed the relevant readings and activities, you should be able to:

- explain the motivations for studying ANNs
- explain the basic structure, working and organisation of biological neurons
- interpret diagrams of units such as Figure 2.2b and diagrams containing many connected units
- define the terms: **unit**, **weight**, **activation function**, ***activation***, ***net***, **threshold**, ***bias***, **fires** and **architecture** as they relate to ANNs
- understand and explain the notational equivalence of: *?a*, *?b…*; $input_1$, $input_2$… and x_1, x_2…
- understand and explain the use of the following symbols: w_{iu}, w_{uv}, Σ, *A()*, a_u, *T(<0,0,1)*.

Notes

Chapter 3

The artificial neuron: Perceptrons

Learning outcomes

By the end of this chapter, and having completed the relevant readings and activities, you should be able to:

- describe the architecture of the ANN that we call a Perceptron
- carry out simple hand simulations of Perceptrons
- manipulate the equations defining the behaviour of Perceptrons with step activations
- explain how Perceptrons can be thought of as modelling lines in the plan
- explain how simple Perceptrons (such as those implementing a **NOT**, **AND**, **NAND** and **OR** gates) can be designed
- discuss the limitations and possible applications of Perceptrons
- understand the behaviour of single threshold units with feedback
- understand the use of a single layer of Perceptrons to divide a plane into parts
- define the terms: **threshold units**, **step units, step activation**, **extended truth table**, **clamped**, **threshold**, **bipolar activation** and **recurrent**.

Essential reading

Rojas, Chapter 3.

3.1 Introduction: Single units

In this chapter we will walk you slowly through some very simple ANNs consisting of just one simple unit. However, as we will see, even a single unit can be very powerful (although later we will see that one unit may often not be powerful enough). Following other authors we use the term ‘Perceptron’ for a single unit, giving its details when necessary. Historically, a Perceptron was a particular type of unit but we prefer to follow the common usage.

Units with just this simple step activation function are surprisingly powerful when combined as we shall soon see, but for now let us see what a lone unit can do!

One of the main problems that we will repeatedly come across during our studies is that of knowing what a given ANN actually does. Another problem is the ‘inverse’ of this – being able to build a network to do what we want it to do. We shall see that in a few cases this is no problem but in the majority of circumstances both of these are very difficult.

3.2 Units with binary inputs and step activation

3.2.1 One or two inputs

To make our calculations easier we will for the time being restrict ourselves to looking at a single unit whose inputs are all binary and whose activation function is the threshold (*T(<0,0,1)*) given above, so that the outputs are also either *0* or *1*. We call units with threshold activations **threshold units**. You may also see them called **step units**, as a step is another way of describing a threshold.

A simple example is shown in Figure 3.1 below.

Inputs	Weights	Parameters	Form	Value
0	*bias*	Learning rate	η	**??**
?a	?	Net	Σ	??
		Activation	T(<0,0,1)	??

Figure 3.1: A simple unit with no bias.

Notice that we have set the 'input' at the bias to zero. This means that the bias has no effect on the calculations as whatever its value the result of multiplying by zero will be zero. We say 'no bias' when the bias has no effect. Also notice that there is just one input, *?a.*

Although this is a very simple network, it allows us to introduce the concept of an 'extended' truth table. An **extended truth table** is a truth table that has entries that evaluate to *0* or *1* but these entries could be variables or even expressions. Because all the inputs are binary, we can use an extended truth table to show how inputs map to outputs.

Here is the extended truth table for the single input unit:

?a	*Net Σ*	*Activation* $a = A(\Sigma) = T(0,0,1)(\Sigma)$
0	*0*	*1*
1	*w*	*if w < 0 then 0 else 1*

Figure 3.2: An extended truth table.

Notice that we have used the 'value' of *net* (ie *1* times the weight when the input is *1*). Also notice that we have spelt out the form of activation function.

Although not very exciting, this truth table shows us that, by making *w* positive, we can make the output always *1* and by making *w* negative we can make the unit output the opposite to its input. This possibility provides us with a way of building a **NOT gate**, to use the Boolean name for a device whose binary output is opposite to its binary input.

That was too easy, so let us now turn to two inputs as in the following diagram, Figure 3.3.

Inputs	Weights	Parameters	Form	Value
0	*bias*	Learning rate	h	**??**
?a	?	Net		??
?b	?	Activation	T(<0, 0, 1)	??

Figure 3.3: A two-input threshold unit.

Again drawing an extended truth table:

?a	*?b*	*net S*	*Activation a = A(net)*
0	*0*	*0*	*1*
0	*1*	w_b	*if* $w_b < 0$ *then 0 else 1*
1	*0*	w_a	*if* $w_a < 0$ *then 0 else 1*
1	*1*	$w_a + w_b$	*if* $w_a + w_b < 0$ *then 0 else 1*

Figure 3.4: An extended truth table for the unit of Figure 3.3.

As you can see things are beginning to get complicated!

Exercise 2

a. Rewrite the table in Figure 3.4 using values *(1, 1), (1, –1), (–1, 1)* and *(–1, –1)* for (w_a, w_b).

b. Rewrite the table in Figure 3.4 using values *(0, 1)* and *(1, 0)* for (w_a, w_b).

c. Do you recognise any of the resulting truth tables?

Comment: One thing that is worth noting here is that if a weight is zero, then the node's behaviour is independent of the corresponding input.

3.2.2 Three or more inputs

It is important that you have some facility with working out the outputs, given inputs and weights (examination questions often ask you to do this), so let us look at some more examples. The diagram below represents a unit with three or more inputs. The 'dotted' input '…' represents 0 or more edges, allowing for an unspecified number of extra inputs.

Inputs	**Weights**	**Parameters**	**Form**	**Value**
1	*bias*	*Learning rate*	η	*0.15*
?a	*?*	*Net*	Σ	*??*
?b	*?*	*Activation*	*T(<0,0,1)*	*??*
…	…	…	…	…
?N	*?*			

Figure 3.5: An N input threshold unit.

We use capital *N* for the number of inputs. When we introduced the notation of the Figures, we said that it is often useful to add to the external inputs a special 'input' which is fixed at the value 1. We called the weight associated with this input the ***bias*** of the unit and now we are including it in the discussion. Previously we set the input to *0* but now it is **clamped**, that is fixed at *1*. Notice that the bias does not count in the *N* inputs – one often finds it thought of as input *0* but, unfortunately, it is also called input *N+1* by some authors.

Although dealing with such units will in general require a computer to keep track of *net* and *activation*, we will look to see if we say something about the output if the values of all but one of the weights are the same. Let us call the value of the common weight *w.*

In this case the value of *net* is the value of *bias* plus *w* times the sum of the other inputs. We can write this as:

$$net = bias + w\Sigma_1^N ?_i$$

Here we have used $?_i$ to denote the ith input – ? reminding us that it is an input.

The equation shows that *bias* indeed acts as a bias by giving the rest of the sum a head start.

To make life a little more complicated the bias, or at least an equivalent of it, has again for historical reasons, been given another name, threshold, or rather bias is 'minus threshold'.

Suppose that we put this sum into the step activation function. We get:

$$a = [\text{if } (bias + w\Sigma_1^N ?_i) < 0 \text{ then } 0 \text{ else } 1]$$

This is, of course the same as saying:

$$a = [\text{if } w\Sigma_1^N ?_i < -bias \text{ then } 0 \text{ else } 1]$$

Now we can see that *–bias* is acting as a (variable) threshold that the rest of the sum must equal or exceed before the output can become 1. When you read around the subject you will see **threshold** being mentioned, and you now know that this is just minus the bias, which in turn is the name of a weight connected to an input that is always 1.

Before we move on, let us look at what we can make with units of the type where

$$net = bias + w\Sigma_1^N ?_i$$

To see what we can do let us simplify the equation a little by using the symbol *M* to represent the number of inputs with value 1. Also if we let *bias* be any real number, rather than just a whole number, then we can write our equation as:

$$net = bias + wM$$

We have been able to do this because the inputs are either *0* or *1* and all the weights are the same.

Remember that the activation is *1* if *net* is greater than or equal to zero, so we can write the condition for our unit to have activation *1* as:

$$net = bias + wM \geq 0$$

or as

$$M \geq -bias/w \text{ and } bias \geq -wM$$

These formulae give us a means of designing units which, in order to output a 1 on firing:

a. require all inputs to be a 1 (that is an **AND gate**) by setting $w = 1$ and $bias = -N$ (the number of inputs)
b. require at least one input to be 1, by setting $w = 1$ and $bias = -1$ (this is an **inclusive OR gate**)
c. require at least a certain number of inputs to be 1, again setting $w = 1$ and *bias = – the number of inputs that we want to be on*. This represents a sort of 'voting' circuit
d. require at most *M* inputs to be 1 by setting $w = -1$ and $bias = M$
e. require at least one of the inputs not to be 1 by setting $w = -1$ and $bias = N - 1$ (this is the **NAND gate**).

You can see that just by adjusting the weights (including the *bias*) we can design a number of useful units. In fact we know from Boolean algebra that by combining a number of NAND gates (e. above) we can make any Boolean function that we want.

This last result might suggest that there is nothing else to do – but that would be an incorrect conclusion. All that it implies is that we can build an ANN to compute any 'computable function' so that they correspond to universal computing devices. Finding the required weights is another, much harder, question. We will see how weight can be found by the use of learning algorithms that train the networks.

3.2.3 A unit as a line in the plane

Let us now turn to the limitations of single units of this type, where we no longer insist that the weights are the same. Weights, inputs and bias are now arbitrary real numbers. We are going to do this by giving another way of looking at the calculation implied by the equation:

$$a = [if\ (bias + \Sigma_1^N\, w_i ?_i) < 0\ then\ 0\ else\ 1]$$

Note that $w_i ?_i$ above stands for weight w_i times input $?_i$. The argument, though not the diagrams of lines in the plane below, works with any number of inputs – that is values for N, but to make our diagrams easy to imagine and draw we will take it to be just 2. Instead of *?a* and *?b* we shall use the letters x and y – for reasons that you will soon see. We will use v and w for the weights corresponding to x and y respectively.

With this notation our equation becomes:

$$a = if\ (bias + vx + wy) < 0\ then\ 0\ else\ 1$$

From co-ordinate geometry we may know that the inequality $(bias + vx + wy) < 0$ represents a 'half plane' determined by the line $(bias + vx + wy) = 0$ and the sign of *bias*. For an easy 'trick' to find out which side of the line corresponds to an activation of 1, just substitute $x = 0$ and $y = 0$ into the equation. This gives the activation at the origin which turns out to be the same as the bias. So if the bias is < 0, the origin has activation of zero and if the bias is positive the activation at the origin is 1. (This link with co-ordinate geometry is the reason we are using x and y notation here, as you have probably realised.)

Exercise 3

On a single piece of graph paper plot the line: $(bias + vx + wy) = 0$ for the values of *bias*, v and w given in the following table:

bias	*v*	*w*
0	0	0
0	0	1
0	1	0
0	1	1
1	0	0
1	0	1
1	1	0
1	1	1

Figure 3.6: Simple parameters for showing parameters of two-input threshold units.

For each line, mark which half plane represents the *activation* being 1 – you might like to use different colours for the lines to help keep tabs on which half plane belongs to which line.

Of course you could use Excel or another package to draw the lines for you – but you cannot do this in an examination so it is important to practise doing it by hand too.

We have included a simple line drawing Excel spreadsheet on the CD-ROM, which also has the solution to this exercise. Take a look at this spreadsheet now.

Exercise 4

In much of this section we have used the activation *A(net) = if net < 0 then 0 else 1*.

How would the results differ if we used: *A(net) = if net ≤ 0 then 0 else 1* instead?

(You should find this a seemingly small but very significant difference.)

3.2.4 Drawing a line in a plane

How about the inverse of this: given a straight line graph, can we build a unit that separates the plane along the line?

Suppose that we have a line given by $y = mx + c$. We can see that this can be written as $mx - y + c = 0$ so that setting $bias = c$, $v = m$ and $w = -1$ will provide the required weights. Not all straight lines, however, can be written as $y = mx + c$: for example, a vertical line cannot be so written. However, it can be written in the form represented by units. The vertical line which goes through $x = c$ and can be written as $c - x + 0y = 0$, that is $bias = c, v = -1$ and $w = 0$.

We have now seen that any straight line in the plane can be represented by a unit and that any two-input (plus bias) unit represents a straight line. If we have more inputs then we must work in 'higher dimensions' with such units representing and being represented by 'hyperplanes'. You will read about these in the reading associated with this topic which is given at the end of this section.

There are a few 'loose ends' we need to tie up to complete our discussions on single units.

Firstly, we note that if you multiply the equation of a straight line by any non-zero number it still represents the same line – so strictly speaking a line is represented by a family of units rather than a unique one. For example, the line $y = mx + c$ is exactly the same line as $7y = 7mx + 7c$. The unit with $bias = c$, $v = m$ and $w = -1$ represents the same line as that with $bias = 7c$, $v = 7m$ and $w = -7$.

Next, we may want the activation to be one 'above the line' or to be one 'below the line'. The same line is involved, so the same family of units have to be used. However if you change the sign of *bias* (by, for example, multiplying the equation of the line by –1) you change the side of the line with *activation = 1*.

Finally, consider the truth table in Figure 3.7.

This truth table is that of the 'exclusive or' function **XOR**; that is, the Boolean function of two variables that is true when one or other, but not both, of its inputs are 1.

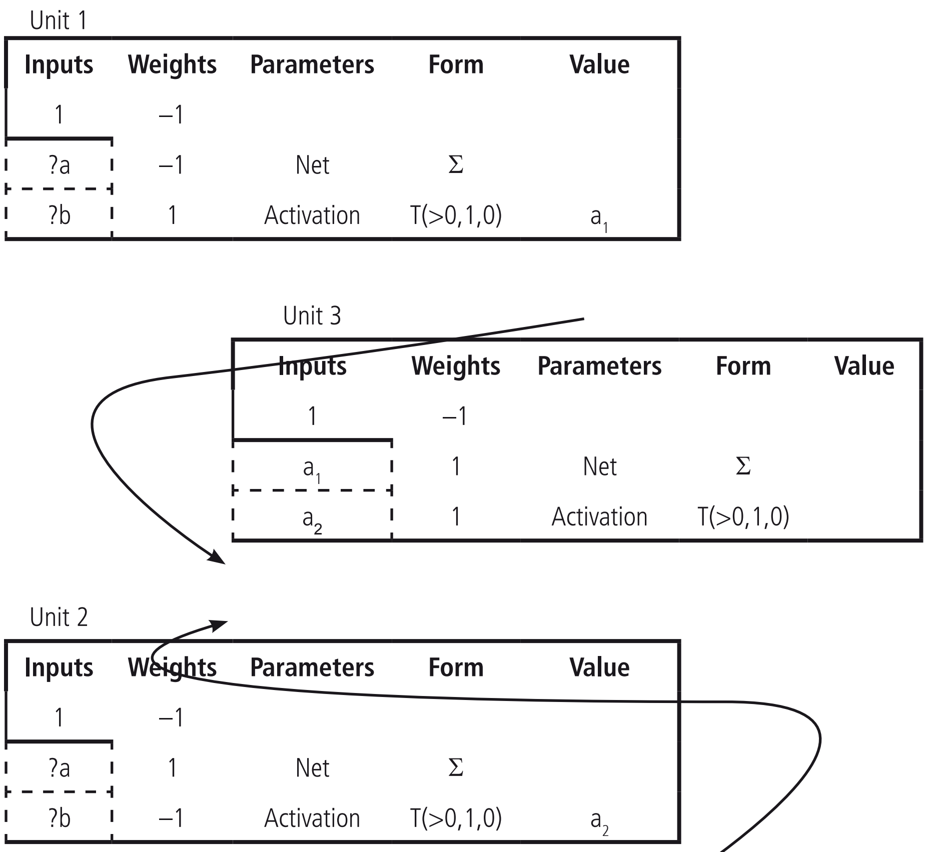
x	*y*	*Activation*
0	0	0

0	1	1
1	0	1
1	1	0

Figure 3.7: Truth table of an exclusive OR unit (had one existed).

If you mark these four points on a graph and try to find a straight line which separates the zeros from the ones, you will fail – there is no such line. This means that there is no single unit that can do this separation and so no single unit can implement this truth table.

This is not a contradiction with our earlier statement that a network can compute anything, for here is an XOR made with units:



Unit 1

Inputs	Weights	Parameters	Form	Value
1	–1			
?a	–1	Net	Σ	
?b	1	Activation	T(>0,1,0)	a_1

Unit 3

Inputs	Weights	Parameters	Form	Value
1	–1			
a_1	1	Net	Σ	
a_2	1	Activation	T(>0,1,0)	

Unit 2

Inputs	Weights	Parameters	Form	Value
1	–1			
?a	1	Net	Σ	
?b	–1	Activation	T(>0,1,0)	a_2

Figure 3.8: XOR made from units.

It was the fact that 'simple' Boolean functions such as XOR cannot be modelled by a single unit, mentioned in Minsky and Papert's book on Perceptrons (Minsky and Papert, 1969) that seems to have persuaded many in the 1970s not to develop neural networks further. They appear to have been mistaken.

Reading

Take a look at the first two chapters of Rojas. Skim any sections with deep mathematical derivations (there are just a few) but try to get a sense of what he is trying to say.

Now read pages 55–76 of Rojas, noting the following points as you do so:

- r3.1.1 makes an analogy between the way that the eye works and classical Perceptrons.
- r3.1.2 reports the limitations of this model.

- r3.2 covers much of what we have said in this chapter – using different notation and extending the materials – but the maths may be more difficult to follow.
- r3.3 extends this work to higher dimensions than two. Also note the idea of weight space is just a matter of treating different variables as dependent or independent.
- You can omit r3.3.3 and r3.3.4 at this stage but you may wish to read them later.
- r3.4 (omitting the maths) is an interesting account of how Perceptrons might be used to model vision.
- r3.4.1 shows how the process of detecting if a pixel belongs to an edge in a picture can be modelled using a Perceptron unit. A set of such units could be used to find all edge pixels in parallel.
- Read the account of the Laplacian operator in r3.4.2, although you do not need to know this.
- r3.4.3 and 4 continues the explanation of how a retina might be modelled.
- r3.5 summarises the history of these topics and is worth a quick read.

Exercise 5

Should we be surprised at proposition 6 of Rojas?

Learning activity

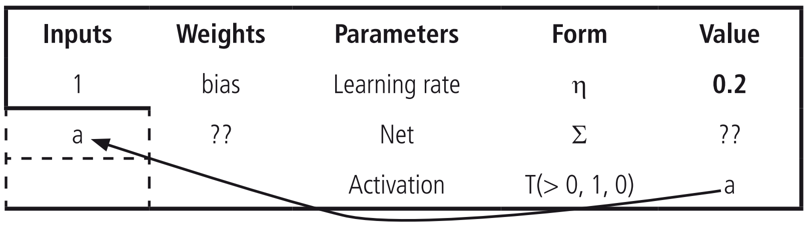
The CD-ROM that accompanies this guide contains a spreadsheet called **lines** that should help you understand and get a feel for the equivalence between Perceptrons and lines in the plane. Use the spreadsheet until you feel confident about designing Perceptrons that implement given lines.

3.3 A single unit with feedback

Up to now our units have had no feedback – that is there were no connections (direct or indirect) from a unit's output back to any of its inputs. Units and networks which have some sort of feedback are called **recurrent**. They are very useful in the right circumstances, but here we will just take a brief look at some complications they introduce.

First, let us look at units with step activation.

Consider a unit with just one input, *?a* say, whose output is connected to its input.



Inputs	Weights	Parameters	Form	Value
1	bias	Learning rate	η	**0.2**
a	??	Net	Σ	??
		Activation	T(> 0, 1, 0)	a

Figure 3.9: A single unit with feedback.

Notice that we are using *T(> 0,1,0)* as threshold. That is *a = if(net > 0 then 1 else 0).*

Our first complication is that the only way to influence this unit is by choice of weight, as its input is determined already by whatever its output happens to be.

To understand the behaviour of this unit we can draw a table of some possible bias, weight and input values.

bias	*?a*	weight	net	activation
–1	0	–1		
–1	0	0		
–1	0	1		
–1	1	–1		
0	0	–1		
0	0	0		
0	0	1		
0	1	–1		
1	0	–1		
1	0	0		
1	0	1		
1	1	–1		

Figure 3.10: How net and activation depend on bias, input and weight.

Exercise 6

Fill in the table above.

Note that sometimes the second and the last columns are not the same. But they should be the same, if the activation is supposed to be fed back to the input. Those rows where the entries are the same are stable, while those that have different entries under *?a* and *activation* are unstable. To make sense of these rows, we need to introduce the concept of time and of a delay between an input being set and an output being determined.

We have to introduce a concept of time and time steps so that the input may follow the output. If we do this, we might expect three possible types of behaviour: a) nothing changes – we have already seen this; b) the input changes in a predictable way; or c) the input changes in an unpredictable way.

In fact only a) and b) are possible with this setup.

Exercise 7

Using experimentation or algebra (or both), try to write down conditions for each of the possible behaviours.

Exercise 8

Repeat Exercises 6 and 7 above with bipolar and sigmoid activations. Are the results any different?

This is as far as we will go with recurrent networks for now. Later we will see just how recurrent networks can achieve useful performance.

3.4 Single layers of units

The section title is a little deceptive because a single layer of units is no more than a number of independent units – possibly sharing some inputs. A single layer of units, instead of producing just one activation as output, produces as many activations as there are units.

Nothing new is needed to understand what these achieve **but** something does emerge from the fact that they are working on the same inputs. Suppose that we have two two-input threshold units and thus two lines in a plane as shown in Figure 3.11.

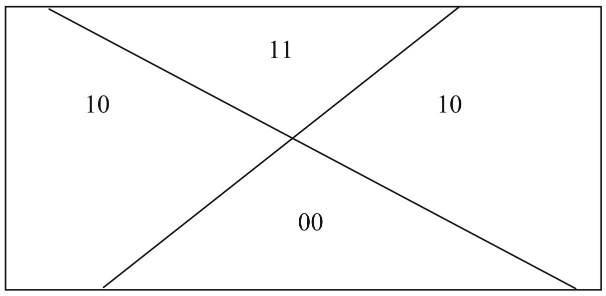


Figure 3.11: Two units dividing the plane into four classes.

Taken together the activations can be thought of as a binary number. If we take the line with a positive slope in the figure as belonging to the unit representing the most significant bit and the line with negative slope representing the least significant bit, then together the two units separate the plane into four regions labelled 00, 01, 10 and 11.

We now have the potential to separate all two digit binary numbers.

A reminder of your learning outcomes

By the end of this chapter, and having completed the relevant readings and activities, you should be able to:

- describe the architecture of the ANN that we call a Perceptron
- carry out simple hand simulations of Perceptrons
- manipulate the equations defining the behaviour of Perceptrons with step activations
- explain how Perceptrons can be thought of as modelling lines in the plan
- explain how simple Perceptrons (such as those implementing a **NOT, AND, NAND and OR** gates) can be designed
- discuss the limitations and possible applications of Perceptrons
- understand the behaviour of single threshold units with feedback
- understand the use of a single layer of Perceptrons to divide a plane into parts
- define the terms: **threshold units**, **step units**, **step activation**, **extended truth table**, **clamped**, **threshold**, **bipolar activation** and **recurrent**.

Chapter 4

Multi-layered feed-forward networks

Learning outcomes

By the end of this chapter, and having completed the relevant readings and activities, you should be able to:

- describe the architecture of the ANN called a multi-layered feed-forward network (MLFFN)
- explain the difference between MLFFNS and MLFFNT
- carry out simple hand simulations of MLFFNTs
- build an MLFFNT corresponding to a given arbitrary truth table
- discuss the limitations and possible applications of MLFFNTs.

Essential reading

Chapter 6 of Rojas is worth reading after you have read this chapter. Rojas's treatment is rather deeper than needed for this half unit but it is worth persevering with it if you can. Parts of this will be needed in later chapters of this guide.

4.1 Introduction

If we combine units in a way that ensures that there is no recurrence, that is there are no 'loops' (no output of a unit can affect its input), even indirectly, then we obtain what we call a multi-layered feed-forward network or **MLFFN** for short. Of course, during the training process, all weights may affect each other. When we wish to distinguish between MLFFN whose units are threshold units from those whose units have a sigmoid activation function we use **MLFFNT** for the former and **MLFFNS** for the latter.

Such networks are multi-layered as we can give each unit a layer number. Starting at the input layer, which we call layer zero (but remember that in this course, we never draw these as units), we give each unit a layer number which is one greater than the maximum of the layer numbers of all those units that feed it.

It is 'feed forward' because we can think of each layer as feeding forward to subsequent layers – there is no feedback.

4.2 Building a neural network for an arbitrary truth table

Suppose that we are given a truth table and wish to find a neural network which, given the table inputs, produces its output. Although we know that we may not be able to do this with a single unit, we will now see how to build one such network.

Section 3.4 above, Single layers of units, gave us a hint as to how this might be done.

As the output column of a truth table contains only *0*s or *1*s, we use units with thresholds and so our ANN will be a MLFFNT. Next we observe that it is easy to find a unit that produces *1* for a given binary input. Suppose, for example, that we want a unit which gives an activation of *1* if and only if its four inputs are *0101*. That is, *0101* is a particular row of the truth table that we are trying to represent.

If we connect all the inputs that need to be *1* with connections with weight *1* and we have a threshold (– *bias* remember!) that is one less than the number of *1*s input, then we get the required response. However, we need to ensure that the inputs that we want to be zero are in fact zero. This is quite easy, as we simply connect them to the unit with a negative weight. If any input that should be *0* is not, or any input that should be *1* is not, then the unit will output *0*. The result is shown in Figure 4.1.

Inputs	**Weights**	**Parameters**	**Form**	**Value**
1	–2	Learning rate	η	**0.15**
	–1	Net	Σ	–2
	1	Activation	T(<0,0,1)	0
	–1			
	1			

Figure 4.1: A four-input recogniser for 0101.

In Figure 4.2, we present the results of inputting values from 0000 to 1111 to this unit. We have omitted the middle column of each unit to make the table more legible. You should be able to see that the unit has the correct behaviour.

We can see that the unit will have activation of *1* if and only if the input corresponds with this row of the truth table *(0101)* that we are trying to represent. If any of those that should be *1* are not, then their contribution is missing from the sum, which will therefore be at least *1* too small. Similarly, if any of those that should not be *1* are *1*, then they contribute a negative amount to the sum, which in turn cannot get big enough. So that only if the input corresponds to this row of the table will it give a *1* – as we require it to. Any other input and the unit will produce a zero. Our unit thus can be used to represent this row of the truth table.

Suppose that we take a unit for each row that has a *1* as output and connect it to another unit in such a way that this new unit acts as an OR. Then this final unit gives a *1* if any of the rows with a *1* as output is presented as an input. Otherwise it will give a *0*.

So we have built the required MLFFNT.

Inputs	Weights	Form	Value
1	–2	η	**0.15**
0	–1	Σ	–2
0	1	T(<0,0,1)	0
0	–1		
0	1		

Inputs	Weights	Form	Value
1	–2	η	**0.15**
0	–1	Σ	–1
0	1	T(<0,0,1)	0
0	–1		
1	1		

Inputs	Weights	Form	Value
1	–2	η	**0.15**
0	–1	Σ	–1
1	1	T(<0,0,1)	0
0	–1		
0	1		

Inputs	Weights	Form	Value
1	–2	η	**0.15**
0	–1	Σ	0
1	1	T(<0,0,1)	1
0	–1		
1	1		

Inputs	Weights	Form	Value
1	–2	η	**0.15**
1	–1	Σ	–3
0	1	T(<0,0,1)	0
0	–1		
0	1		

Inputs	Weights	Form	Value
1	–2	η	**0.15**
1	–1	Σ	–2
0	1	T(<0,0,1)	0
0	–1		
1	1		

Inputs	Weights	Form	Value
1	–2	η	**0.15**
1	–1	Σ	–2
1	1	T(<0,0,1)	0
0	–1		
0	1		

Inputs	Weights	Form	Value
1	–2	η	**0.15**
1	–1	Σ	–1
1	1	T(<0,0,1)	0
0	–1		
1	1		

Figure 4.2: Results of unit in Figure 4.1 being given all possible inputs showing that it recognises only 0101 (continued overleaf).

Inputs	Weights	Form	Value
1	–2	η	**0.15**
0	–1	Σ	–3
0	1	T(<0,0,1)	0
1	–1		
0	1		

Inputs	Weights	Form	Value
1	–2	η	**0.15**
0	–1	Σ	–2
0	1	T(<0,0,1)	0
1	–1		
1	1		

Inputs	Weights	Form	Value
1	–2	η	**0.15**
0	–1	Σ	–2
1	1	T(<0,0,1)	0
1	–1		
0	1		

Inputs	Weights	Form	Value
1	–2	η	**0.15**
0	–1	Σ	–1
1	1	T(<0,0,1)	0
1	–1		
1	1		

Inputs	Weights	Form	Value
1	–2	η	**0.15**
1	–1	Σ	–4
0	1	T(<0,0,1)	0
1	–1		
0	1		

Inputs	Weights	Form	Value
1	–2	η	**0.15**
1	–1	Σ	–3
0	1	T(<0,0,1)	0
1	–1		
1	1		

Inputs	Weights	Form	Value
1	–2	η	**0.15**
1	–1	Σ	–3
1	1	T(<0,0,1)	0
1	–1		
0	1		

Inputs	Weights	Form	Value
1	–2	η	**0.15**
1	–1	Σ	–2
1	1	T(<0,0,1)	0
1	–1		
1	1		

Figure 4.2: Results of unit in Figure 4.1 being given all possible inputs showing that it recognises only 0101 (Continued from previous page).

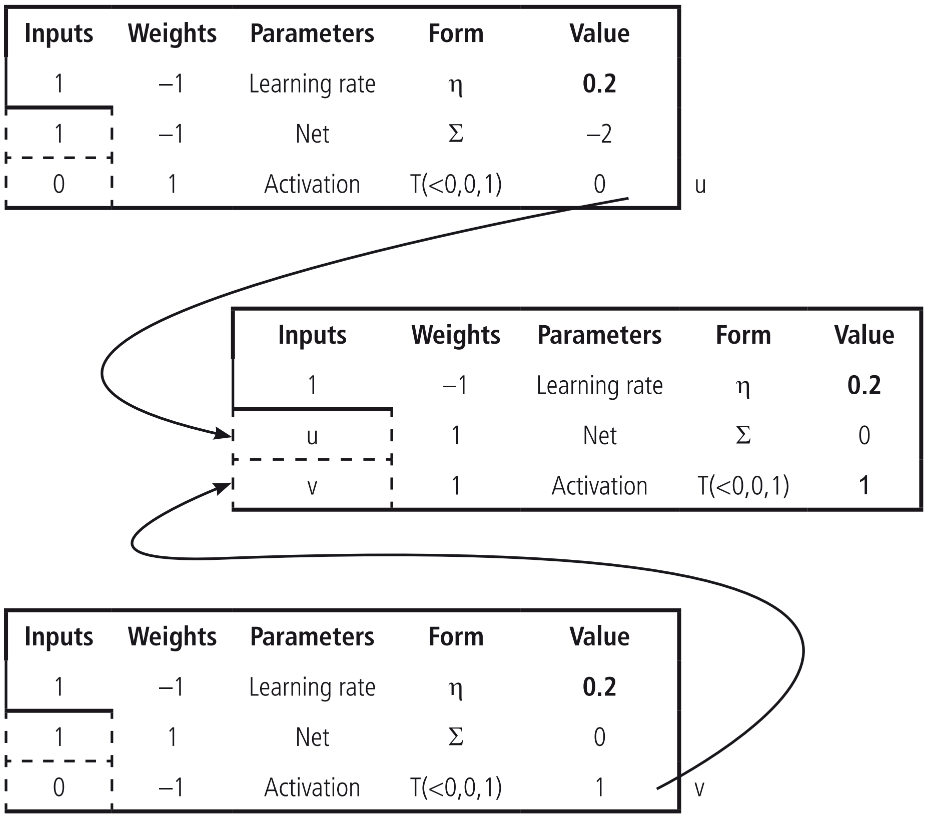
Example

The exclusive OR function has the following truth table:

?a	?b	activation
0	0	0
0	1	1
1	0	1
1	1	0

Figure 4.3: Truth table for Exclusive OR

The table is modelled by:



Inputs	Weights	Parameters	Form	Value
1	–1	Learning rate	η	**0.2**
1	–1	Net	Σ	–2
0	1	Activation	T(<0,0,1)	0

Inputs	Weights	Parameters	Form	Value
1	–1	Learning rate	η	**0.2**
u	1	Net	Σ	0
v	1	Activation	T(<0,0,1)	**1**

Inputs	Weights	Parameters	Form	Value
1	–1	Learning rate	η	**0.2**
1	1	Net	Σ	0
0	–1	Activation	T(<0,0,1)	1

Figure 4.4: A three-unit network that models an Exclusive OR.

Exercise 9

Check that the network in Figure 4.4 has the required behaviour, by drawing its truth table.

So we have seen that although the XOR table could not be implemented by a single (threshold) unit, it is relatively easy to build one from three units.

A quick way of testing this sort of unit

We have built our units in such a way that the correct input (that is, input patterns that require the unit to produce a *1*) have a *net* value of *0*. We can do this because our activation function is the 'threshold' represented as *T(< 0, 0, 1)*, that is *a(net) = 0* if *net<0 else a(net)= 1*.

In cases like this we can check that changing any input from its required value to another **decreases** the value of *net,* thus making it negative and so preventing the unit from producing a *1*.

4.3 More complex problems

In the last section we saw how any given Boolean function can be made up from two layers of units.

However, there are many problems where we do not know the complete function that we are trying to model, but have some examples of its output value for certain values of its inputs. Sometimes we have no idea what output we want – but we might recognise a good outcome when we see it.

Another area where the results of the last section are not so useful is where the inputs and/or outputs are not Boolean. For these problems we need other sorts of units.

There are many varieties of 'multi-layered feed-forward networks' and so the next section must of necessity be very selective. We concentrate on what is probably the most popular and most studied – the 'backpropagation network'.

We need, however, a short detour first into search.

A reminder of your learning outcomes

By the end of this chapter, and having completed the relevant readings and activities, you should be able to:

- describe the architecture of the ANN called a multi-layered feed-forward network (MLFFN)
- explain the difference between MLFFNS and MLFFNT
- carry out simple hand simulations of MLFFNTs
- build an MLFFNT corresponding to a given arbitrary truth table
- discuss the limitations and possible applications of MLFFNTs.

Chapter 5

ANN as search

Learning outcomes

By the end of this chapter, and having completed the relevant readings and activities, you should be able to:

- explain how all problems can be written as a search
- distinguish between **supervised** and **unsupervised** learning
- distinguish **batched** learning from unbatched or **online** learning
- define the term **network architecture** and its parameters I, N, O and E
- given a diagram of an ANN, describe it in terms of its architecture and define all of the symbols used in the diagram.

Essential reading

Parts of Chapters 4 and 6 of Rojas are important sources for this Chapter. You should have already read Chapter 6 of Rojas; his Section 6.1 will need rereading later.

5.1 Introduction: problem solving as search

It can be considered that all problems are essentially search problems or at least that they can be written that way.

Suppose that we have a problem – one that has proven too hard to solve by ‘conventional means’. One strategy that we often try in AI is to represent the problem in such a way that we can represent potential solutions explicitly. The effect of this representation is to turn our problem into a search. In this way **all** computing problems can be thought of as search problems.

Exercise 10

Take a computing problem that you have ‘solved’ recently (it could, for example, be an exercise from another half-unit). Can you translate this problem into one of search?

We think of the problem of designing a neural network which models a function that we have incomplete information about, or even a function which is too complex for the manual techniques that we have used up to now, as searching for an architecture and a set of parameters that results in a final activation that models our required function.

So far we have built our networks ‘by hand’. In general, this is no different from programming and, for complex problems, is harder than writing the equivalent program in a modern programming language.

However, one of the main attractions of the ANN model is that the ‘search’ for the weights that are required to make our ANN produce the correct output for a given input, can be automated (as we shall see in the next chapter).

Before considering this automation we need to look a little into the subject of learning and its vocabulary.

5.2 Supervised and unsupervised learning

Humans learn in many ways, several of which are still not fully understood by those studying human learning.

For a moment, let us suppose that learning involves improving our performance at a task. To make this more concrete let us further suppose that the task involves the learner being presented with a stimulus and responding in some way.

For some types of learning there is no obvious 'correct' response – it is possibly in this way that mankind first learnt to use tools and clothes. If it is up to the learner to determine the correct response, we call this 'unsupervised learning', although of course before we can say that improvement has taken place we need some way of measuring improvement.

However, for many learning tasks the learner is presented with both a stimulus and a corresponding 'required response'. This is the way that children are often taught – at least for some tasks such as arithmetic (especially multiplication tables). If this is all that is involved, then a computer should easily perform well at this type of learning, which we call 'supervised learning' – all that seems to be required is for it to remember the response to each stimulus.

However, we expect much more of our learners. In languages, for example, we expect them not only to understand the sentences that they have met before but also to understand new ones – sometimes 'guessing' their meaning when a simple 'word for word' translation will not do. In mathematics, we expect our learners to be able to solve new problems 'by analogy' with the ones that we have shown them. We could go on but there should be no need – you can probably think of several other examples.

Other types of learning are corrective and reinforcement learning. You can see that some learning requires supervision – it is called 'supervised' learning.

Unsupervised learning is also common among both children and adults. We are not given examples of what we are to do, nor is someone needed to correct or guide us. Sometimes we just learn our own way of doing things. In a sense it is the data that we are presented with that acts as a supervisor. For example, left alone in a sweet shop (with permission perhaps to sample) a child will have no difficulty in classifying sweets according to their taste.

When we teach students we are sometimes able to give examples to illustrate our points. In fact it is often the case that we explain our method (algorithm).

Now read Rojas pages 77–79 for his classes of learning algorithms.

5.3 Batched and unbatched learning

When learning from examples we might 'update' each time we have considered one example. This is called 'online' learning. Alternatively, it might be better to update only after we have considered a number (a batch) of examples. This is 'offline' or batched learning.

In future while reading the set book and other sources try to classify the techniques that you see according to Rojas's Figure 4.3.

5.4 ANN architecture

Read Rojas 6.1, to the end of 6.1.1. Although Rojas defines an architecture abstractly for us, all we need to understand is that for a neural network to be defined we need to know:

I The set of inputs that we are going to have in our network.

N The set of computing units; this includes their types and any parameters that are needed; and those that are not to be learnt. Included here is also the learning algorithm that is to be used.

O The set of output sites.

E The connections between the units, with their weights – although some or all of these may be learnt.

A reminder of your learning outcomes

By the end of this chapter, and having completed the relevant readings and activities, you should be able to:

- explain how all problems can be written as a search
- distinguish between **supervised** and **unsupervised** learning
- distinguish **batched** learning from unbatched or **online** learning
- define the term **network architecture** and its parameters I, N, O and E
- given a diagram of an ANN, describe it in terms of its architecture and define all of the symbols used in the diagram.

Notes

Chapter 6

Training networks

Learning outcomes

By the end of this chapter, and having completed the relevant readings and activities, you should be able to:

- state and apply the Perceptron learning algorithm giving explanations of every symbol and term used
- state and apply the Pocket algorithm giving explanations of every symbol and term used
- explain the implications of the statement that 'two-layer feed-forward neural networks made of units with sigmoidal activation are universal'
- state and use the Backpropagation training algorithm, giving the meaning of every symbol and term used
- define the terms: Backpropagation, sigmoid activation function, epoch, the Widrow-Hoff rule and forward and backward passes.

Essential reading

Rojas, Chapter 4, especially Section 4.1 and Figure 4.3.

Rojas, Chapter 7.

Further reading

Webster (2009) *Webster's online dictionary*. Available online at www.websters-online-dictionary.org/Wi/Widrow-Hoff_rule.html (Last accessed June 2009.)

6.1 Introduction

In this chapter we study two learning algorithms, one for Perceptrons and one, backpropagation, for feed-forward layers of units with the sigmoid activation.

6.2 Perceptron learning

Training a single threshold unit turns out to be almost trivial but its discussion provides us with an excuse to revisit some of the mathematics that you have learnt in the first year of this degree.

Reading

Now reread Rojas section 4.1.

Of this, sections 4.1.3 and 4.1.4 are a little technical but there is little, if any, mathematics that you have not seen before. You should recognise the matrix multiplication that calculates the values of *net* for the different inputs. Note the unfortunate convention that Rojas (and some others) use of having the bias as the last weight, rather than what seems to us the more natural zeroth weight.

Continue reading Chapter 4, section 4.2. Algorithms; 4.2.1 'Perceptron learning' and 4.2.2 'The Pocket algorithm' are the key and almost trivial results of this section. Rojas gives good insights into how they work, although do not be put off by the mathematics.

We can paraphrase the Perceptron learning algorithm as follows:

> If the unit correctly classifies all training examples then stop, otherwise add a correction of ***η(target - activation)*input*** to each of the weights before testing the next example.

Weights may get different corrections depending upon their input. In the Perceptron case it is easy to assign 'credit' (or rather blame) when things go wrong because we know what the error is and we know what caused the error – weights that had non-zero inputs. Remember, of course, to update weight 0, the bias – it is easy to forget. In fact, this is the most updated weight as it is always updated when an error occurs.

Exercise 11

Can you explain the last statement of the paragraph above?

The Pocket algorithm just adds a step of remembering good sets of weights and how good they were. There are a number of variations on this idea.

Now read the rest of Chapter 4 of Rojas. Although you will not be examined on 4.3, it might add to your understanding of what happens on training.

6.3 Backpropagation networks

We have used a threshold or step function as the activation function of choice. However, for our current purposes we need something different. Figure 6.1 shows the sigmoid activation function $\sigma(net) = 1/(1 + exp(-net))$. This is the activation function used for backpropagation networks.

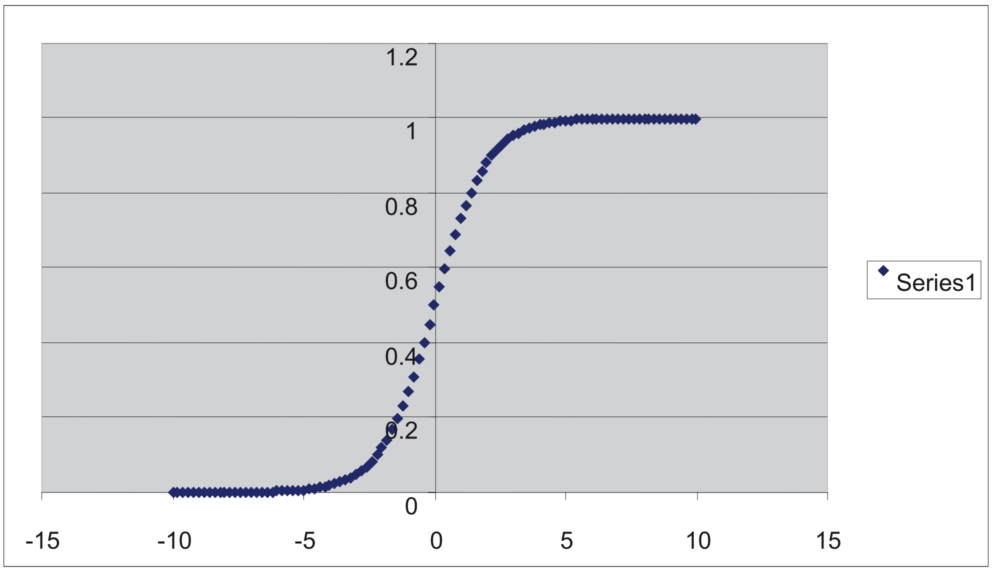


Figure 6.1: The sigmoid activation function σ(net) = 1/(1 + exp(-net)).

A backpropagation neural network is a feed forward, two or more layered network of units which have **sigmoidal** activation preceded by a 'summing block' that calculates the *net* of the unit. Although we can have as many inputs and outputs and as many hidden layers as we like, it is simpler (for the time being at least) to think of **inputs**, just one hidden layer and an output unit. What makes it a backpropagation network is the algorithm used for learning.

Backpropagation is probably the most used feed-forward artificial neural network and so we are going to see how this works.

We use backpropagation for **supervised learning** when we have a set of **training** data that we are trying to model – to approximate the function or classifier that the training set represents. The **training set** consists of a number of (input, output) pairs which we label ($?_i$, t_i) to emphasise that the x-components are inputs or (x_i, t_i) when wanting to be more explicit. Take care not to confuse our (and others') use of *x* here with a coordinate – although inputs might represent coordinates, they can represent any characteristic that we are interested in. For us they are just input values. It is worth noting here that we use the term **epoch** to mean one cycle of learning through the whole training set.

Two-layer feed-forward neural networks made of units with sigmoidal activation are universal in the sense that any computable function can be represented by one of these networks.

Of course we only use a training method if we do not know an easier way of calculating the weights required for success. For example, it would be foolish to train a neural network to learn the Boolean function for majority voting.

The power of backpropagation stems from its training algorithm as we shall see below.

To make our notation simple we consider only one output unit and use *net* and *a* for its net and activation respectively). The *H* hidden units are labelled *1 ... H* with net_h and a_h being their respective nets and activations. In mathematical formulae reference to unit 0 is just a shorthand for taking into account any biases that the units have. The *N* inputs are likewise labelled *0 ... N* to include the bias. The weights are w_h from the *h*th hidden unit to the output and w_{ih} from input *i* to hidden unit *h*. Do not forget to include terms with index zero for the bias in all calculations of *net*.

For a backpropagation neural network, units usually have *nets* that add (Σ) and sigmoid activations so that:

$$net_h = \Sigma_{i=0}^{N} w_{ih} x_i \qquad \text{and} \qquad a_h = \sigma(net_h) = 1/(1 + exp(-net_h))$$

for the hidden layer of a two-layer network and

$$net = \Sigma_{h=0}^{H} w_h a_h \qquad \text{and} \qquad a = \sigma(net) = 1/(1 + exp(-net))$$

for the output unit.

Why use a sigmoidal activation function? Firstly, people wanted to use a function that they could use mathematics to understand. Thresholds are neither continuous nor differentiable, whereas the sigmoid function is both continuous and differentiable. It looks very similar to a threshold with rounded edges.

The sigmoid has another useful property:

If we differentiate $\sigma(net)$ with respect to *net* we get:

$$\frac{d\sigma(net)}{dnet} = -\,net\; exp(-net)\; (1 + exp(-net))^{-2} = net\; (1-net) \text{ after some algebra.}$$

Thus working out the gradient of the activation requires little extra computation beyond working out the activation itself.

Calculating the output of the network is a simple process as we feed forward the inputs through the hidden layer and then through the output layer. Note that if there is more than one output, it is easier to treat each output separately as they have no means of communication.

We can easily derive an expression for the activation of a backpropagation network. When an equal sign has a superscript to its left this indicates the equation (i to iv below) that is being used in that step):

Using the four relations that define *net* and *activation*:

i) $a = \sigma(net) = 1/(1 + exp(-net))$

ii) $net = \Sigma_{h=0}^{H} w_h a_h$

iii) $a_h = \sigma(net_h) = 1/(1 + exp(-net_h))$

iv) $net_h = \Sigma_{i=0}^{N} w_{ih} x_i$

we get

$\boldsymbol{a}$

${}^{i}= \boldsymbol{\sigma}(net)$

${}^{i}= 1/(1 + exp(-\mathbf{net}))$

${}^{ii}= 1/(1 + exp(-\Sigma_{h=0}^{H} w_h \boldsymbol{a}_h))$

${}^{iii}= 1/(1 + exp(-\Sigma_{h=0}^{H} w_h (1/(1 + exp(-\mathbf{net}_h)))))$

${}^{iv}= 1/(1 + exp(-\Sigma_{h=0}^{H} w_h (1/(1 + exp(-\Sigma_{i=0}^{N} w_{ih} x_i)))))$

Where we have used bold for the term that is about to be replaced.

Training the network is slightly more complicated and we won't derive the expressions used; we just state them. Derivations are usually mathematical and are to be found in Chapter 7 of Rojas. However, it is worth noting that the learning algorithms are variations of 'steepest descent'.

The backpropagation learning algorithm uses the Widrow-Hoff rule; that is, it minimises a measure of error. According to *Webster's Online Dictionary* (Webster, 2009) this is an 'Error correction learning rule in which the amount of learning (i.e. the modification of the connection strengths), is proportional to the difference (or delta) between the activation achieved and the target activation'. You will see how this works from the following sketch of the algorithm:

Forward pass: calculate the output of the network by forward propagating the outputs of one layer to the inputs of the next.

Backward pass: Given the actual output of the network, calculate the error:

error = e = desired (target) – actual

modify the weights leading to the output unit using: $\Delta w = \eta e a(1 - a)x$

For each weight the change is:

Learning rate * error * activation * (1 – activation) * input.

Error assigned to unit = sum of weighted errors of units fed.

If there were no error in the output then there would be no need for training. Therefore there must be an element (x, t) of the training set for which the calculated output is incorrect. Correcting the weight to the output unit is necessary, and this is done by adding a correction Δw_h to w_h for each hidden unit and adding a correction Δw_{ih} to w_{ih} to each weight from the inputs to the hidden units.

Rojas 7.1.1 gives us an expression for this correction which, for a single output unit in our notation is:

ε	=	$(t - a)a(1 - a)$	the error on output
Δw_h	=	$\eta\varepsilon\, a_h$	the change of weight from hidden unit *h*
ε_h	=	$\varepsilon\, a_h\, (1 - a_h)w_h$	the error for a hidden unit *h*
Δw_{ih}	=	$\eta\varepsilon_h\, x_i$	the change of weight from input *i* to hidden unit *h*

Note that you will see lots of different looking formulae in the literature – you need to get used to this as no standard is yet agreed.

If a unit feeds more than one successor unit then there will be a contribution from each successor, so that εw_h becomes a sum over all successors. We will not need this formula but you need to know that it exists.

Rojas Chapter 7, gives an interesting account of the derivation of the equations used for backpropagation. Although you are expected to learn and to be able to use the equations given, there is no need to learn the proofs.

Rojas also suggests that all the errors are calculated before any weight updates are done. There seems to be some disagreement in the literature on this point, but we follow Rojas's lead.

Read Chapter 7 of Rojas now. (You can skim through 7.4 if you wish.)
It presents an interesting way of looking at backpropagation.

A reminder of your learning outcomes

By the end of this chapter, and having completed the relevant readings and activities, you should be able to:

- state and apply the Perceptron learning algorithm giving explanations of every symbol and term used
- state and apply the Pocket algorithm giving explanations of every symbol and term used
- explain the implications of the statement that 'two-layer feed-forward neural networks made of units with sigmoidal activation are universal'
- state and use the Backpropagation training algorithm, giving the meaning of every symbol and term used
- define the terms: Backpropagation, sigmoid activation function, epoch, the Widrow-Hoff rule and forward and backward passes.

Notes

Chapter 7

Kohonen-Grossberg counterpropagation networks

Learning outcomes

By the end of this chapter, and having completed the relevant readings and activities, you should be able to:

- describe the architecture of Kohonen-Grossberg Counterpropagation networks
- state the algorithm used to train the Kohonen layer of such a network
- explain how Rojas's algorithm 5.1.1 differs from that given in this chapter and comment on its significance, if any
- describe how initial values and number of units are determined.

Essential reading

Rojas, Section 5.1 and Chapter 16.

Kohonen, T. (2009a) *Kohonen network*. Available online at www.scholarpedia.org/article/Kohonen_network (Last accessed June 2009.)

Further reading

Kohonen, T. (2009b) www.cis.hut.fi/teuvo/ (Last accessed June 2009.)

7.1 Introduction

When Darwin went on his voyage of discovery on the Beagle, part of his task was to collect and classify the variety of living things that he came across. In those days there was no 'genetic fingerprinting' to help and biologists had to rely on the examination of the visible characteristics of animals and plants in order to classify them.

People have tried to get ANNs to undertake similar tasks of classification. Suppose that we have a set of 'things' each with a number of properties. For example, a catalogue of dates and times and the epicentres of earthquakes or eruptions of volcanoes. How might one go about searching for or putting some order into this set? In other words, is it possible to classify the elements of the set in some way according to their properties? Clearly the answer will depend on the set and on the properties that we have to use.

We want to make an analogy with the evolution of companies. Suppose an organisation wants some work done. It asks for estimates and chooses the lowest estimate. The company that gets chosen, assuming that they are competent at the task, make money and are in a better position to do similar work next time. They may, for example, have bought some specialist equipment or gained some skills in doing the task. In a similar way, we 'ask' a number of units how well they can match a particular object. The 'winning' unit – all being well – should do better next time.

The classification problem has two phases. First we choose a set of objects that are to be used as a training set. This training set is used to determine a set of classes that the objects can be divided into. Then once we have some classes we need a process that assigns any new objects to the classes found during training.

It is possible to attempt this task using 'neural networks'. We have put the terms in quotes because it is not always clear that the techniques that we describe here are in fact ANNs – although of course almost any algorithm can be rewritten as an ANN one.

Because we do not know at the start the classes to which the objects belong, this is an example of unsupervised learning. We are not going to tell the ANN what the classes are. Instead we expect the ANN to find some suitable ones.

Suppose the objects to be classified have properties which we code into numeric values so that each object can be represented by a vector with *N* values – that is a point in *N* dimensional space.

As you may already have guessed, there are a number of techniques that have been developed to undertake this task. We are going to describe one, called counterpropagation networks and being even more specific than this, a type of counterpropagation network called a Kohonen-Grossberg network.

A Kohonen-Grossberg network has inputs, a 'Kohonen layer' and a 'Grossberg layer'. The inputs are just the same as we have had many times before – they simply pass on the inputs to whatever they are connected to. The Grossberg layer is often omitted, but sometimes contains linear units which transform the output of the Kohonen layer into something more useful. It is the Kohonen layer that is of most interest to us here.

7.2 The Kohonen layer

In this layer each unit represents a vector. Its first operation (what we have called calculation of ***net*** in previous work) involves the calculation of the inner product between its inputs and the vector that it represents. The activation is then worked out by:

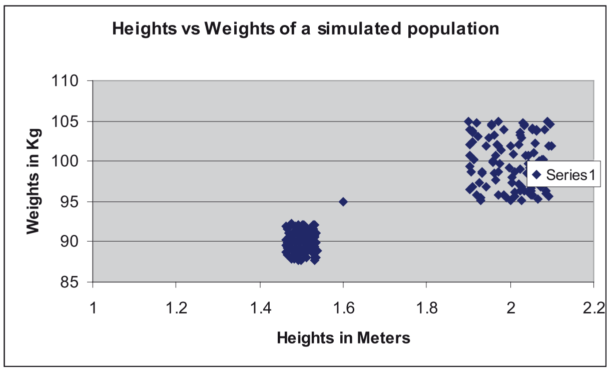
$$a_u = \textit{if } net_u > net_v \textit{ for all } v \textit{ then } 1 \textit{ else } 0$$

That is, all of the units have output *0* except the one with the maximum *net* which then has an activation of *1*.

This is a 'winner takes all' strategy. If there is a tie in *net*s then just one of the winning units outputs a *1* – that is, an arbitrary choice is made.

Eventually we want to say that object '*X* **belongs to class** *u*' if unit *u* outputs a *1* when the network is presented with *X*. This is the same as saying that *X* is 'nearer to' unit *u* than *X* is to any other unit. There is a slight technical issue that we want to point out here and that is just what 'nearest' means.

The two diagrams below show plots of similar simulated populations of objects. There is one object that we wish to classify as belonging to either 'bottom-left' or 'top-right'. The diagrams differ only in units of height but the confidence with which the object can be added to 'bottom-left' seems different in each case.



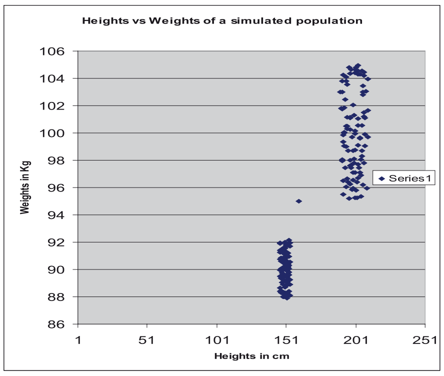


Figure 7.1: How changing the scale can influence nearness.

This problem, often described as 'comparing chalk and cheese', does have some statistical answers but clearly one has to be careful.

Now to the details of how the Kohonen layer works.

We take our set of vectors from which we wish to find a number of classes in order to classify others of the same type. To make calculations more efficient we 'normalise' each of these vectors by dividing each component by the length of the vector. If, for example, our object is (3, 4) we normalise it to (0.6, 0.8) because (3, 4) has length 5. Note that one needs to check that normalisation does not affect the potential results. For example, if we have reason to believe that (3, 4) and (0.6, 0.8) are very different and are likely to belong to different classes, then normalisation is not a good idea as it would confuse the two. To see why normalisation may make classification easier one needs to consider what it actually does.

If we have classes that are sets of points in a plane, then normalising these would place them onto the unit circle – thus changing the problem from a two-dimensional into a one-dimensional problem.

Remember that units are the same as classes, which in turn are equivalent to vectors. As the network is trained, the units are changed by changing the vectors. This in turn changes the classes.

Here is a sketch algorithm for training the network.

1) Normalise the training set.
2) Choose a set of initial classes – that is, points in the space that our objects are from. We think of these initial classes as vectors of weights – making the analogy with neural networks. We identify these classes with their vectors and call them 'units' of the network.
3) Normalise the initial classes.
4) Until we have 'done enough' repeat the following:
    i) choose a training example x, (at random or systematically)
    ii) find the class w_c closest to the chosen example
    iii) update w_c by adding $\eta(x - w_c)$ to it and then normalising the result.

A few 'loose ends' need to be considered:

a) How do we choose initial classes? There are a number of ways of doing this. Choosing weights (vectors) randomly is one way, and this is often followed by a slightly different learning strategy for a short time. One 'tweak' is that rather than adjusting just one unit, we adjust the losers in the other directions. An alternative is to start with a large learning rate (η) and reduce this as learning progresses.

b) How many units (classes) do we use? This is not an easy question and again a number of heuristics (rules of thumb) have been tried. One might start with just a few units and after training see what happens if others are added 'in between'. One can see if the resulting network is better or worse. Alternatively, one might start with ample units and, again after training, see what happens if one or more is removed.

c) Finally what does 'done enough' mean? This might be how long learning has been going for, or how many times around the loop we have gone – or better it might mean that the performance of the network is 'more than adequate'. Often, however, it means that the classes are not changing much and so the return for extra effort seems not to be worthwhile.

We end this section with a simple worked example so that you can see Kohonen units in action before reading about them elsewhere.

Suppose we have the following data to classify:

Original	
x	**y**
2.1010	3.3819
– 0.8316	1.9591
– 0.4898	– 0.8128
20.5960	35.0536
0.5950	– 0.9575
– 2.2123	4.2541

Our first task is to normalise this. Adding a few extra columns will help.

Original			**Normalised**		**Check**
x	**y**	**length**	**x′**	**y′**	
2.1010	3.3819	3.9814	0.5277	0.8494	
– 0.8316	1.9594	2.1286	– 0.3907	0.9205	
– 0.4898	– 0.8128	0.9490	– 0.5161	– 0.8565	
20.5960	35.0536	40.6565	0.5066	0.8622	
0.5950	– 0.9575	1.1273	– 0.5278	– 0.8494	
– 2.2123	4.2541	4.7950	– 0.4614	0.8872	

We have added an extra column so that you can check that all data points are of length 1.

We next choose three random vectors to be our units and normalise them:

Original			**Normalised**	
x	**y**	**length**	**x′**	**y′**
0.1239	0.3335	0.3558	0.3483	0.9374
0.0166	0.1279	0.1290	0.1286	0.9917
0.5525	0.5550	0.7831	0.7055	0.7087

Starting with the first normalised data vector we multiply by each unit in turn:

	x	y	x′	y′	net
Unit $_1$	0.5277	0.8494	0.3483	0.9374	0.9800
Unit $_2$	0.5277	0.8494	0.1286	0.9917	0.9102
Unit $_3$	0.5277	0.8494	0.7055	0.7087	0.9742

The first value of *net* is the highest, so we update unit$_1$, using a learning rate of 0.1, for example.

Unit$_1$ = (0.3483, 0.9374) + 0.1 [(0.5277, 0.8494) – (0.3483, 0.9373)] = (0.3663, 0.9286).

This has length 0.9982 and after normalising the new unit becomes (0.3669, 0.9303).

The process repeats with the next data point, and so on.

Exercise 12

Plot the points and the three random vectors on a graph. Continue the calculation until you have run through each data point a number of times, showing the movement of the three points.

You may be expected to do this sort of calculation in an assignment and/or an examination question. Practise it well!

We have gone through this example very quickly as there is little new in it. However, do make sure that you know how to train such a network.

Reading

Read Section 5.1 of Rojas and compare his training method with that given above. Read Chapter 16 ‘Modular neural networks’ in Rojas which explains the way that different ANNs can be combined into more powerful systems. Although r16.2.3 is our focus, the rest of the chapter is worth reading too. R16.2.3 treats counterpropagation in a slightly more general way than we have here but our method is more detailed.

Remember, though, that you may need to perform the calculations that we have shown you above in an examination.

A reminder of your learning outcomes

By the end of this chapter, and having completed the relevant readings and activities, you should be able to:

- describe the architecture of Kohonen-Grossberg Counterpropagation networks
- state the algorithm used to train the Kohonen layer of such a network
- explain how Rojas’s algorithm 5.1.1 differs from that given in this chapter and comment on its significance, if any
- describe how initial values and number of units are determined.

Notes

Chapter 8

Boltzmann machines

Learning outcomes

By the end of this chapter, and having completed the relevant readings and activities, you should be able to:

- describe the process of physical annealing as applied to metals
- explain how the essential features of annealing are modelled in simulated annealing
- explain the terms energy, temperature, simulated annealing
- describe the algorithm known as Boltzmann learning, giving explanations of every symbol and term used.

Essential reading

Rojas, Chapter 14.

Hinton, G.E. (2007) *Boltzmann machine.* Available online at www.scholarpedia.org/article/Boltzmann_machine (Last accessed June 2009.)

Further reading

MacKay, D.J.C. *Information theory, inference, and learning algorithms.* (Cambridge: Cambridge University Press, 2005) [ISBN-13: 9780521642989; ISBN-10: 0521642981]. Available online at www.inference.phy.cam.ac.uk/mackay/itila/book.html (Last accessed June 2009.)

8.1 Introduction

We often look to the physical world to provide us with inspiration, and the idea of Boltzmann training is an excellent example of this. Let us take a detour to explain why.

Probably over 5,000 years ago metal workers of the copper age (Chalcolithic) discovered that letting a hot metal cool slowly improved its properties. This **annealing** of metal is explained by physicists as allowing the microscopic structure of the object to order itself in a way that minimises stresses and consequently energy.

It is easy to see this effect when, for example, trying to fit a number of small irregularly shaped objects into a jar. We put as much as we can in and then shake the jar so that the material settles, leaving more room at the top for some more objects. A systematic way of doing this without a lid would be to place the empty jar on a vibrating table, add material slowly and watch it pack. While the jar is not too full, large vibrations would allow the material to settle quickly, relying on the fact that it is harder to jump out the further down the jar the material starts. As the jar fills we will need to reduce the amplitude of vibration to avoid material leaving the jar. It is this slow reduction of the amplitude of vibration as metal cools that enables the metal to organise itself when annealing. When we use a Boltzmann machine to solve a problem we set it up so that 'good solutions' to our problem correspond with well packed material and the 'temperature' or 'vibration' is reduced as the process comes to completion.

8.2 Simulated annealing in ANNs

Suppose that we have an ANN which is to model a set $\{ (i_v, t_v) \}$. That is, given inputs i_v, the required outputs are t_v.

Among the measures of performance error that we met in Chapter 7 of Rojas is:

$$E = ½ \Sigma_v \, // \, t_v - a_v // \, ^2$$

Where a_v is the activation in response to i_v. Note that constant factors, such as the ½ in the formula, can be omitted without changing the learning behaviour of the network.

Consider the following strategies for improving the network.

First strategy: choose new weights and recalculate *E*. If the proposal decreases the value of *E* we accept the proposal, otherwise we reject it.

Now this strategy, assuming that all weight changes are possible, will eventually lead to a network with a minimum value of *E*. So if a set of weights for the network with zero *E* exists, it will eventually be found.

The strategy described so far is just a random search with replacement – because in principle a set of weights that has already been considered and rejected could occur again (although of course it would be rejected again).

The strategy suffers from not making use of knowledge of the current weights and how good a model the current ANN is. It does not use any information gained while searching.

A number of alterations can be made to the strategy which might lead to a more orderly search:

a. We might restrict the number of weights that are changed at any one time.
b. We might restrict the size of change made to the weights.
c. We might decide to accept a proposed change even if it makes the value of *E* higher.

The effect of a) and b) is to make the search that we are doing in weight space more local. That is, depending upon the details of how many weights and how big the weight changes can be, any new set of weights will be quite close to the previous one. But herein lies the problem. It just might happen that there is no improvement possible with this number of weight changes and this size of change. We have a 'local' minimum in *E* and so cannot improve our ANN further, unless we allow rule c) to apply. This is necessary as a consequence of the locality of the search. Using the shaking analogy, sometimes you have to shake hard to get a beneficial effect – even if it spoils things a little.

In around 1983 Hinton and Sejnowski showed how to use this idea in searching and in learning by a process of simulated annealing. The account given here may differ from the original, but we think that it is an equally valid strategy.

A Boltzmann machine is a network of units that are connected through symmetric weights, $w_{uv} = w_{vu}$. Some units have activations imposed or **clamped** upon them while the activations of others are treated as outputs – any other units can thus be considered 'hidden' although there is no real difference between these three types – input, output and hidden.

When a Boltzmann machine is used to search for a solution to a problem, the problem has to be written as a minimisation of a cost or evaluation function. The evaluation function, also called the energy function E (to make the analogy with physics) has to take the form:

$$E = -\Sigma_{uv}\, w_{uv}\, a_u\, a_v$$

where candidate solutions are assignments of *0* or *1* to the *a*'s. That is, we are looking for a set of $a_u s$, each either zero or one, which minimises E. The w_{uv} are determined by the problem to be solved.

With the problem in this form, a set of units with fixed weights given by w_{uv} are set off with randomised initial values of a_u and the Boltzmann machine goes through the following algorithm:

a. Choose a unit, *u* randomly and calculate its net: $net_u = a_u\, \Sigma_v\, w_{uv}\, a_v$.
b. As *net* represents a contribution of the unit to *E,* choosing $a_u = 1$ if *net* is negative and $a_u = 0$ otherwise, is a strategy that reduces E.

The question arises about what it is best to do if *net* is positive. In simulated annealing, even if a unit has a positive *net* we allow it to give an *activation* of *1* with a probability, *p*, which reduces as *net* increases. Often the probability used is $p = (1 + e^{-net/T})^{-1}$. As with much of neural networks, there are a variety of details that differ from one author to another.

This idea of simulated annealing or statistical aided learning can be extended to all types of neural networks that undergo supervised learning.

Suppose that we have a network with training set *t* and current weights *w.* We calculate the evaluation function E using:

$$E = \Sigma_v\, || t_v - a_v ||^2$$

Note that the network is trained if $E = 0$ because this implies that all of the activations have their target values.

Next we consider the effect on E of a change in weights. Some schemes have only one weight changing at a time but others allow more to change. Again, calculating E for this proposed state let us call it E'. Clearly if $E' < E$, then this new state is an improvement on the old. We accept it and repeat the process.

We could continue looking for other changes until we can no longer find any – although this might of course take some time.

Although we might end with a final E of *0*, it is more likely that we reach a state where E is stuck on a local minimum (recall that we have mentioned this idea before) and no adjusting of weights seems to work. We need a strategy to reduce this possibility. If, instead of always accepting a lowering of the evaluation function and rejecting increases, we accept a weight change with a probability given by:

$$p = (1 + e^{-(E' - E)/T})^{-1}$$

where **temperature** T is a parameter to be chosen, we avoid getting stuck – although we do allow the risk of E increasing as compensation! As in annealing of a metal, the temperature T is slowly reduced towards zero as the training progresses. How slowly to lower the temperature is more an art than a science but we essentially leave the system to settle down and reach 'thermal equilibrium' at the current temperature. In this state the system is no longer improving so we reduce the temperature a little.

Reading

Now read Chapter 14 of Rojas. Sections 14.2.1, 14.2.2 and 14.3 are especially important.

Read what Hinton has to say about Boltzmann machines at Hinton (2007).

A reminder of your learning outcomes

By the end of this chapter, and having completed the relevant readings and activities, you should be able to:

- describe the process of physical annealing as applied to metals
- explain how the essential features of annealing are modelled in simulated annealing
- explain the terms energy, temperature, simulated annealing
- describe the algorithm known as Boltzmann learning, giving explanations of every symbol and term used.

Chapter 9

Hopfield nets

Learning outcomes

By the end of this chapter, and having completed the relevant readings and activities, you should be able to:

- describe the architecture of Hopfield nets
- describe the firing scheme used for Hopfield nets
- explain the term 'energy' as used in Hopfield nets
- determine and draw the state transition network for a given Hopfield net
- explain how simple problems and classic ones (such as the Travelling Salesman and Eight Queens Problems) are solved using a Hopfield net
- define the terms: predecessors, successors, synchronous, asynchronous, Hebb's rule, Widrow-Hoff rule and sink as used in working with Hopfield nets.

Essential reading

Rojas, Chapters 12 and 13.

Chapter 42 of MacKay, D.J.C. *Information theory, inference, and learning algorithms*. (Cambridge: Cambridge University Press, 2005) [ISBN-13: 9780521642989; ISBN-10: 0521642981]. Available online at www.inference.phy.cam.ac.uk/mackay/itila/book.html (Last accessed June 2009.)

Hopfield, J.J. (2009a) *Hopfield network*. Available online at www.scholarpedia.org/article/Hopfield_network (Last accessed June 2009.)

Further reading

Hopfield, J.J. (2009b) *Website of John J. Hopfield*. Available online at www.genomics.princeton.edu/hopfield/ (Last accessed June 2009.)

9.1 Introduction

A Hopfield net is a set of N units ($U_1 \dots U_N$) each connected directly to all the others. Authors often add a bias unit, but we include the possibility of a bias in our units.

The units in the Hopfield net are usually binary threshold units. However, there is some confusion in the literature about the actual threshold used. Common thresholds are:

i. $a = \text{if } net < 0 \text{ then } 0 \text{ else } 1$

ii. $a = \text{if } net > 0 \text{ then } 1 \text{ else } 0$

iii. $a = \text{if } net < 0 \text{ then } -1 \text{ else } 1$

iv. $a = \text{if } net > 0 \text{ then } 1 \text{ else } -1$

Although Hopfield (2009a) uses ii) and Rojas uses iii) (excluding a state change if net = 0), we will use i) above for our calculations. You will need to take care whenever you read about such ANNs that you understand the types of unit being discussed.

During your reading you should also notice that there are continuous, or 'graded' versions of Hopfield nets.

When we say that the units are connected directly to **all others** we imply that a unit is not connected directly to itself. Another restriction on Hopfield nets is that the two connections between each pair of units have the same weight: we can write this mathematically simply as $w_{uv} = w_{vu}$. It is sometimes convenient in a mathematical treatment (and especially when programming) to allow self connections with a fixed weight of zero, so that we have $w_{uu} = 0$.

Because of the recurrency, all units have **predecessors** and we cannot easily think of any as the input units. Instead we think of the initial values (which after all will be needed to calculate the inputs to other units) as our inputs – we have control over these.

Similarly all units have **successors** and so no unit is solely an output. Instead we choose which units to monitor and these will be called outputs. Units whose outputs we ignore (except for propagation) can be thought of as hidden units. So the same unit could be both an input and an output.

In a feed-forward system it seemed natural to work from the inputs to the outputs in a systematic way, only calculating the output of units for which the inputs were already calculated. In a recurrent system the output of a unit may well depend upon itself. That is, its output determines the outputs of its successors and each unit is indirectly one of its own successors. A different strategy must therefore be used in our current circumstances.

Suppose that we choose a unit and, using the current states of its inputs, calculate its *net* and its new *activation*. Before moving on to the next unit we have a choice: do we use this new value in the calculations or do we wait until all units have been calculated before updating any values? If we wait and change all of the values together we call the strategy '**synchronous**' as the effects appear synchronously. If, on the other hand, we use the new value in the calculation of the next and in our subsequent calculations we call the strategy '**asynchronous**'. Although the synchronous strategy may seem more 'ordered', it turns out that the asynchronous is more easily dealt with in theoretical and practical investigations.

For the asynchronous strategy, we must also choose the unit to update and again we have a choice. We could choose in a systematic way so that each unit is updated the same number of times, or we could choose a unit to update randomly even if that unit has recently been updated. It is the latter 'random choice with replacement' that is commonly used.

When doing our calculations we use the 'current' values of the activations of the units – we can describe the network's behaviour only when we know all of these values. The list of units and their corresponding activations is known as the 'state' of the network. The idea of a state is important for systems with feedback. It is not so important for feed-forward systems.

We also have a concept of 'time' in that we can think of the sequence of states through which our network 'travels' as units fire. Of course, we have to be careful because sometimes a unit may 'fire' without a change in state – we simply think of the network as going from a state to itself.

To recap, in a synchronous system the activations of all the units are calculated at each step. In an asynchronous system the calculation of just one activation completes a step.

If there is no feedback in a network and where the behaviour of a unit depends only upon its inputs, we can easily trace its behaviour by considering how it responds to every possible input combination.

One complication of using asynchronous firing of units is that it is slightly more difficult to trace. For a Hopfield net, for example, we have to take account of the order of 'firing' when tracing its behaviour.

9.2 Calculating transitions between states

To fully understand the behaviour of a given network we need to look at all possible firing orders from all possible states that the net can be in. With *N* binary units we have 2^N possible states and so we can describe its behaviour using a truth table with 2^N rows and $N(N+1)$ columns – *N* to represent the state before firing and *N* sets of *N* to represent the state after firing – one for each unit. That's a lot of columns!

A useful trick to use is to do this calculation as if we were going to have the units firing synchronously – this makes for a more compact representation of what is going on – using only *2N* columns. But when using this notation we have to remember to keep all but one of the units constant. We do this as follows:

The first *N* columns are labelled *1… N* to represent the current state of the corresponding unit. The next *N* columns are also labelled *1… N* to represent the state of the corresponding unit **if** it should fire. Note that we do not need to record any changes to non-firing units as they cannot change. Such a table is called a state table. For convenience we often write the columns from *N* down to *1* so that the rows appear as binary numbers from *0* to 2^N. In our treatment below we have added extra columns, such as 'state number' that help us to navigate the table.

The main table in Figure 9.1 shows the results of calculations on a three-unit Hopfield network whose weights are given in the table. Note that we put the bias weights as the first row and column of our weights table, the boxed table at the bottom left.

The first column of the results is labelled 'state' as it represents the number of the current state if the unit's states are taken as binary numbers but with the units taken in reverse order. That is, *unit 1* is the least significant bit of the state number.

The three columns labelled **Before** give the states of each unit, and the columns labelled **After** show what happens if corresponding units fire. As explained above we do not give the state of units that do not fire.

Next, we show the state of the whole network after each firing. For example, if the network starts in state 3 (011) and *unit 2* fires, then as *units 1* and *3* do not change the new state is 001 (*state 1*). This is because the net of *unit 2* is negative and so *unit 2* goes to zero. This result is shown in the final three columns where the middle column has a '*0*' in the row corresponding to *state 3*.

Hopfield discrete

weights	***bias***	***1***	***2***	***3***
bias	0.00	0.10	–0.05	–0.05
1	0.10	0.00	–0.40	0.20
2	–0.05	–0.40	0.00	0.60
3	–0.05	0.20	0.60	0.00

State	***Before***			***After***			***State after***		
	3	***2***	***1***	***3***	***2***	***1***	***3***	***2***	***1***
0	0	0	0	0	0	1	0	0	1
1	0	0	1	1	0	1	5	1	1
2	0	1	0	1	0	0	6	0	2
3	0	1	1	1	0	0	7	1	2
4	1	0	0	0	1	1	0	6	5
5	1	0	1	1	1	1	5	7	5
6	1	1	0	1	1	0	6	6	6
7	1	1	1	1	1	0	7	7	6

Figure 9.1: Calculation of state transitions.

9.3 State transition diagrams

Another very useful way of representing the behaviour of a Hopfield net is by drawing a 'state transition diagram'. This is a more pictorial representation as a diagram where state changes occur from left to right.

In this form of diagram we draw each state, and from that state a directed edge is drawn to every other state that can be obtained from that state – the successors to that starting state. An important point to note is that loops from a state to itself are possible. States which have loops do not change when some of the units fire. Except for these loops, there is no need to label edges to see which unit is firing, as this will be clear by comparing the state at the end of the edge. Such a comparison will show that exactly one unit has changed (and so must have fired) on going from its start to its end. A state from which there is no escape is called a **sink** or stable state and there is also no need to put a loop on such a state.

Here is how we draw such a diagram:

We scan down the three columns and notice that state 6 is a sink because whatever unit fires, the network stays in this state once it gets there. We show this by writing:→6. Now we look for states that either stay the same or go only to state 6. State 7 is the only one of these. We thus have: 7 → 6.

Repeat this process, this time looking for any states that end in either themselves or 6 or 7. We find state 5 has this property so we have 5 → 7 → 6. Continuing we find that 1 → 5 → 7 → 6 and 0 → 1 → 5 → 7 → 6. Next we have a new situation arising: both 2 and 4 depend only on what we have already met. We can choose any of these – so we have chosen 2 to get 2 → 0 → 1 → 5 → 7 → 6. We could insert 4 next but it is neater (see later) to add 3 to get 3 → 2 → 0 → 1 → 5 → 7 → 6. Finally we are left with the 4. This needs to be before 0 but is unconnected to the 2 and the 3. For the time being we just put it in front of the 3 but with no arrow connecting 4 with 3.

4 3 → 2 → 0 → 1 → 5 → 7 → 6. Note this does not go to 3 but to 0. We are about to connect the 4 to the 0.

Of course, this is not quite the whole story as there are lots of other transitions still to mark. We put these in with curves. The final picture is shown in Figure 9.2.

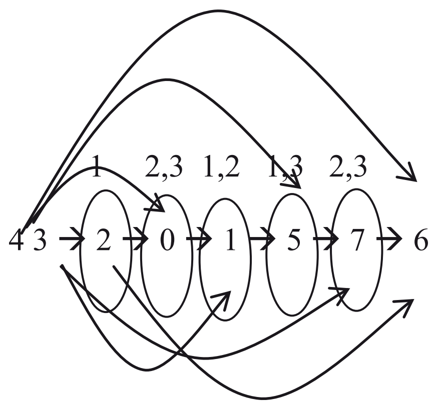


Figure 9.2: State transition diagram.

9.4 Understanding Hopfield networks

The rule for a Hopfield unit activation is *a = if net < 0 then 0 else 1.*

Normally we are only interested in the activation of a unit; the summation is just part of the calculation of *net* to determine this activation. However, for Hopfield nets there is an interesting property of the summation blocks.

Suppose that we calculate the sum of the nets of all units:

$$S = \sum_{u=0}^{u=N} net_u$$

If a unit has an activation of *0*, it changes to *1* on firing if its *net* is non negative.

Similarly if a unit has an activation of *1* it changes to *0* on firing if its *net* is negative.

Looking at the contribution of such a unit to *S* we can see that a unit changing from *0* to *1* causes *S* to increase by net_k and one that changes from *1* to *0* also increases *S* by its *net*. In both cases *S* increases because of the change.

Hopfield noticed this property and consequently the analogy between *S* and energy in physics. If we define *Energy*, *E = –0.5 S* we have the proposition that a change of state is never accompanied by an increase in energy. The factor of *0.5* in the formula is there to simplify its derivation.

For calculations we can make the sum simpler by using the fact that $w_{ij} = w_{ji}$ and so we can rewrite E as $E = -\sum_{u=0}^{u=N} a_u \sum_{v=0}^{v<u} w_{uv}\, a_v$.

We can summarise one property of a Hopfield net as being that its Energy never increases and in fact decreases whenever a unit changes from a *1* to a *0*.

Clearly this implies a number of things:

a. A Hopfield net has states which, if entered, cannot be left – as well as being called **sink** or **stable** states, such states are sometimes called **energy wells** to conform to the usage of physics. We have seen this in the example above.

b. For an energy well, the summation block of each unit that has value *1* is non negative and the summation blocks of each unit that has value *0* is negative.

Sink states can be very useful. Suppose that we can make a Hopfield network with specified sink states. If one of these networks is started in a state which is not a sink state, it will eventually move into such a state. This is the basis of some attempts at pattern recognition – handwriting recognition, for example. Of course, we would have to ensure that all stable states were meaningful and, more importantly, we would hope that starting from a state which is 'near' (in handwriting terms) to a stable state would eventually end up in that state – unfortunately this is not always the case.

There are many ways that we can attempt to build a network with specified sinks. Besides the 'trial and error' (with practice and some luck) two more formal methods are worth noting here. They are the Hebb rule and the Widrow-Hoff rule.

9.5 Hebb's rule

For the threshold units that we have considered, a learning rule might be:

i. Take a random pattern from the set of required sink states that we are trying to impose on our network.
ii. For each pair of units u, v in the network, if their activations (for this chosen pattern) are the same, that is if $a_u = a_v$, then add a small positive amount η to w_{uv}. Otherwise subtract η from w_{uv}. In a computer program we would not store w_{vu} separately from w_{uv} , so these two values automatically stay the same. See Rojas p.316.

This version of Hebb's rule differs from some that you may find in the literature. This is either because a) the literature uses different types of unit for Hopfield nets or b) there is a lot of scope for alternatives in this subject.

In fact, as long as the changes are based on the idea of making correlations between units reflect correlations in the training set, then one can say that Hebbian learning is taking place.

9.6 The Widrow-Hoff rule

This rule also uses reinforcement learning. If a pattern is not a stable state then this means that for some units the state that the pattern represents differs from that which would be generated by looking at the activations generated by the unit's *net*. In such cases, the weights are modified to correct that particular unit's *net*.

You will see more details of these learning rules in your reading.

Reading

A good place to start your reading would be by looking at what Hopfield has to say on the subject. His website (Hopfield 2009b) and his entry on *Scholarpedia* (Hopfield 2009a) should be consulted. Do not worry about technical details beyond those in this guide – but do take the time to look so that you can fill in some of the details and have a different perspective on the topics.

You should read the following parts of Rojas:

Chapter 12 of Rojas on associative networks is important, although it contains too much mathematics for our needs. It is well worth reading, while skimming the maths.

Chapter 13 of Rojas (again mathematical) has lots of useful material that will reinforce what is said above. Sections r13.3, r13.5 and r13.7 should be read as they contain important material. It is worth knowing how to set up Hopfield networks to solve the Eight Queens Problem and the Travelling Salesman Problem.

Chapter 42 of MacKay (2005) is also worth reading – again skim the mathematics but look for applications and insight into the possibilities that Hopfield networks give.

A reminder of your learning outcomes

By the end of this chapter, and having completed the relevant readings and activities, you should be able to:

- describe the architecture of Hopfield nets
- describe the firing scheme used for Hopfield nets
- explain the term 'energy' as used in Hopfield nets
- determine and draw the state transition network for a given Hopfield net
- explain how simple problems and classic ones (such as the Travelling Salesman and Eight Queens Problems) are solved using a Hopfield net
- define the terms: predecessors, successors, synchronous, asynchronous, Hebb's rule, Widrow-Hoff rule and sink as used in working with Hopfield nets.

Notes

Chapter 10

Problems in ANN learning

Learning outcomes

By the end of this chapter, and having completed the relevant readings and activities, you should be able to:

- list some of the problems that arise when an ANN learns
- define the terms: **overfitting**, **network paralysis, momentum**
- explain the strategies that have been devised to overcome the problems that may arise.

Essential reading

Rojas, Chapter 8.

10.1 Introduction

Having looked at a number of unit types and their learning algorithms, it is now time to be a little more realistic. In our discussions we have omitted to mention some of the many things that can go wrong when we try to use a neural network to model some unknown function that is given by a set of examples.

It is important that we know what sort of problems we might face when training an ANN. It almost goes without saying that any particular implementation of ANNs on a computer will have limitations of memory and time, but it is worth emphasising this point. You may be surprised how often it is forgotten.

Our problem may be so complex that the number of units required is too large for our implementation. It may be that our search takes too long, either because of the speed with which convergence takes place or because the weights are oscillating – never actually settling down.

Even if these problems do not occur, real implementations may have limitations in the range of numbers that can be represented. It may be that the ANN that we are searching for has weights that are too big to represent.

10.2 Overfitting, network paralysis and momentum

Overfitting sometimes occurs because our learner 'sees' only a sample of the possible things that it is likely to come across during actual use; there is a danger of it being biased towards the training set and as a result not reacting well to real examples.

One might imagine a child learning the concept of cat from many examples that she sees at home. However, when presented, for the first time, with a dog, it might be taken to be a cat – the child has taken cat to represent four legged animals within a particular range of sizes.

There is no really good way of avoiding this happening, but we sometimes split our known examples into three parts: one part for training, one part, the **selection** set for 'interim' testing and one for actual or final testing.

We train using the training set, until performance on the selection set seems to be getting worse and worse. At this stage we choose the network parameters as they were when the best results on the interim test were obtained as our final parameters. Then we test on the final testing set.

The number of units needed for any particular application is not well understood. We might use a strategy of growing a neural network. Growing means start small and if the results are not good enough, try again with a larger number of units. We are often in a situation where the number of input and output units is fixed by the given problem (although, of course, some of the choices that we have already made might well impact on this). For example, we might choose to have one class per output unit or else have an output unit contribute to some coding of the class.

When growing in this fashion one may experiment with either using the smaller network as a starting point or else starting from scratch – both strategies are likely to have problems where they are better than the other.

An alternative strategy is pruning. Pruning means making sure that you have a trained network and then seeing if you can 'tighten it up' by removing units.

A good strategy here is to see what happens if each hidden unit is removed in turn and the resulting network tested, trained and tested again. It may be that performance of the retrained smaller network is:

a. much better
b. not much different
c. much worse.

Clearly a. is good but might be expecting too much – although even here there is a danger of overfitting.

If b. is the case then the gain/loss of functionality may be worth accepting for the simpler network.

If c. is the case then one can tentatively class the unit as 'essential', although in systems such as ANNs where performance is 'distributed', 'essential' is a difficult concept.

It is often worth looking carefully at such units in case they provide some insight into how the ANN is doing its job.

Reiterating somewhat, here are some of the things that may cause us difficulty when applying ANNs:

1. There may be no set of weights that are 'correct'.
2. The correct weights are too big for the software being used.
3. The final activations are too sensitive for the accuracy with which calculations are done.
4. The learning algorithms get stuck in local minima.
5. The weights oscillate without converging.
6. Training is too slow.
7. Overtraining – the network learns and performs well on training set but cannot perform well on new examples.
8. **Network paralysis** occurs when the parameters to a sigmoid are large so that even large changes in input cause very small output changes.

Very many strategies have been tried in an attempt to avoid some of the problems. Chapter 8 of Rojas covers some of these and should be read at this stage.

Read Chapter 8 of Rojas focusing especially on 8.1.1 and 8.2.

Important points to note are use of:

- careful choice of initial weights
- alternative network architectures
- growing and pruning the networks as they learn or as they are used
- alternative learning algorithms
- alteration of the weight update formula to include a '**momentum**' term (that is, take into account the previous change when determining that of the current step)
- **momentum**: add μ times Δw from last step – where parameter μ is the momentum coefficient
- different learning rates for different weights
- adaptive learning rates – rates that depend upon stage of learning, etc.
- use of lookup tables instead of function evaluation to speed up the process
- use of special hardware designed for running ANNs.

A reminder of your learning outcomes

By the end of this chapter, and having completed the relevant readings and activities, you should be able to:

- list some of the problems that arise when an ANN learns
- define the terms: **overfitting**, **network paralysis**, **momentum**
- explain the strategies that have been devised to overcome the problems that may arise.

Notes

Chapter 11

Applications of neural networks

Learning outcomes

By the end of this chapter, and having completed the relevant readings and activities, you should be able to:

- list some of the applications of ANNs
- describe applications of each of the following types of ANN: single unit, multilayered feed-forward networks, Kohonen-Grossberg counterpropagation (or related networks), Boltzmann machines and Hopfield nets.

Essential reading

Sejnowski, T.J. and C.R. Rosenberg (1987) *Parallel networks that learn to pronounce English text.* Complex Systems 1, pp.145–168. Available online at www.cnl.salk.edu/ParallelNetsPronounce/index.php (Last accessed June 2009.)

Further reading

Clabaugh, C., D. Myszewski and J. Pang (2009) *Neural networks.* Available online at http://cse.stanford.edu/class/sophomore-college/projects-00/neural-networks/Files/presentation.html (Last accessed June 2009.)

Lisboa, P.J.G., B. Edisbury and A. Vellido *Business applications of neural networks: the state-of-the-art of real-world applications.* (Singapore: World Scientific, 2000) [ISBN-10 9810240899; ISBN-13 9789810240899].

Fogelman-Soulié, F. and P. Gallinari *Industrial applications of neural networks.* (Singapore: World Scientific, 1998) [ISBN-10 981023175X; ISBN-13 9789810231750].

Rabunal, J.R. and R. Dorrado *Artificial neural networks in real-life applications.* (Hershey, PA.: Idea Group Inc (IGI), 2006) [ISBN-10 1591409020; ISBN-13 9781591409021].

Tang, H. and Y. Zhang *Neural networks: computational models and applications.* (Berlin: Springer, 2007) [ISBN-10 3540692258; ISBN-13 9783540692256].

11.1 Introduction: The diversity of ANN applications

The application areas of ANNs can easily be summarised as ‘everywhere’; they have universal application.

Artificial neural networks have been applied to a wide range of areas but it is true to say that obtaining detailed accounts that give sufficient details for the reader to replicate these results is quite hard.

There are a number of reasons for this. Often the ‘details’ are thought of as not necessary for the reader, especially since a great deal of information such as initial weights might be needed.

At other times commercial interests exclude the possibility of giving full disclosure.

However, here is a list of resources that the student can look at which give a flavour of the depth and the breadth of applications.

One site gives character recognition, image compression, stock market prediction, medicine, security loan applications, etc. See: Clabaugh (2009).

Here is a list of applications from just one book (Murray, 1995):

- face finding in images
- gender recognition from faces
- classification of arrhythmias
- classification of cells in cervical smears
- multiphase flow monitoring in oil pipelines
- electrical load forecasting
- process control
- robot control
- a self-teaching Backgammon program
- EEG analysis.

The Manufacturing Engineering Centre (MEC) at Cardiff University has used neural networks for many manufacturing applications. An excellent book, *Neural networks for identification, prediction and control.* (Pham, 1995) is worth looking at if you can obtain a copy. It is also worth looking on Google Books (books.google.com) for up-to-date accounts of applications.

Rojas talks about many applications of neural networks, although his treatment tends to be distributed throughout the text.

For example, see the following Sections in Rojas:

- 3.4 application in the biology of vision
- 5.3 on principle component analysis
- 5.4 image compression
- 9.3 a number of applications including Nettalk
- 15.4 applications of a slightly more complex network than the Kohonen networks that we considered.

An interesting diagram is included as Figure 18.1 of Rojas, where performance requirement of speech and image processing applications is shown.

You might be asked about applications in coursework and/or examinations so you should make sure that you know some details of five or six.

Find out more about Nettalk and even listen to some of its output by visiting Sejnowski (1987).

You should have no difficulty finding many more applications by using Google or another web search engine.

Other sources that you might like to try are:

- Business Applications (Lisboa, Edisbury and Vellido, 2000)
- Industrial Applications (Fogelman-Soulié and Gallinari, 1998)
- Other Applications (Rabunal and Dorrado 2006; Tang and Zhang, 2007).

Learning activity

This activity involves you in finding technical information about applications of ANNs. For each of the five types of ANN (see below) that we have studied, you are to make notes of:

i. the aims and objectives of the application
ii. how its success is measured
iii. the technical details of the network used, training and use of the trained network.

Using a search engine of your choice find details of applications of each of the following types of ANNs:

a. single unit
b. multilayered feed-forward networks
c. Kohonen-Grossberg counterpropagation or related networks
d. Boltzmann machines
e. Hopfield nets.

Write up your notes so that you can describe each of these applications to colleagues who have studied this half unit.

A reminder of your learning outcomes

By the end of this chapter, and having completed the relevant readings and activities, you should be able to:

- list some of the applications of ANNs
- describe applications of each of the following types of ANN: single unit, multilayered feed-forward networks, Kohonen-Grossberg counterpropagation (or related networks), Boltzmann machines and Hopfield nets.

Notes

Appendix A

Excel and Word-friendly notation for units

Suppose that we wish to represent an *N* input unit.

1. Draw a table with *N+2* rows and five columns. The first row and column will be used for headings and labelling.
2. Label the columns: **Inputs**, **Weights**, **Parameter**, **Form** and **Value** respectively.
3. Label the rows: Inputs, *1*, *?a*, *?b*, … etc., unless you know the names of the inputs, in which case you can use these names for row two onwards. The *?* signifies an input. The *1* in row two represents the fact that the *bias* will be next to it in row two – you can make this *0* if you want no bias.
4. The column labelled **Weights** has the weights w_0 ... w_N where weight w_0 is the *bias*.
5. **Parameters** give a list of parameter names, **form** gives more details, if necessary, and **value** should contain the formula to be used. Check that this is as you expect. Different types of unit will have different items under **parameter** – so you may need to add extra rows.

The first parameter is 'learning rate'; it can be omitted or left blank if there is no learning involved.

Most units have a 'summation block': that is, they form the sum of products of the inputs and associated weights (Σ is used to signify summation). Another possibility is product (Π), but in principle any rule can be tried. In any case we call the resulting value the **net** of the unit. The formula will need to be checked!

Much of the time the final parameter will be the unit's activation. The symbol for it should be in the fourth column and the formula in the fifth. Again, check this before using. Frequently used activations are **threshold**, **linear** and **sigmoidal** and each of these has subtypes depending upon where the activation is zero and any other parameters necessary. Bipolar outputs (*–1* and *1*) are often used instead of (*0* and *1*).

The **activation** is the output of the unit that is passed on to the next units.

This tabular representation is the representation to use when you are demonstrating a simple neural network with few units. It becomes too time-consuming and unwieldy for complex networks when a flat spreadsheet (plus a hand sketch) is probably best for both calculation and for presentation. Obviously if you use a package that draws a representation of the network on the screen, then a screenshot is best.

Here is an example of a general unit:

Inputs	Weights	Parameter	Form	Value
1	??			
?a	??	Net	Σ	
?b	??	Activation		

Here it has been completed to represent a two-input inclusive OR gate before the inputs have been applied.

Inputs	Weights	Parameter	Form	Value
1	–1			
?a	1	Net	Σ	
?b	1	Activation	T(<0,0,1)	

And finally again, this time with some inputs set and values calculated:

Inputs	Weights	Parameter	Form	Value
1	–1			
1	1	Net	Σ	0
0	1	Activation	T(<0,0,1)	1

It is important that if you are going to use these in Excel, you make sure that the formulae are correct. A common mistake is to copy formulae, only to find that they do not refer to the correct cells. Automation can help you work faster but it needs much more setting up than pencil and paper, and lack of care can result in a great deal of wasted time. Note that the notations T(< 0, 0, 1) and T(≥ 0, 1, 0) for the threshold function are equivalent.

Appendix B

Bibliography

Before buying books you should see if they are available for preview at Google Books (books.google.com). Type in the ISBN and with luck you will find the book. Sometimes no preview or a limited preview is available and sometimes you might be lucky and discover that the whole book is available.

It is also worthwhile using Google, or another search engine, to see if the book is available for download. Authors in academia are increasingly making their writing freely available on the Web.

The 'set book' or essential reading for this guide is Rojas (1996). Although out of print it is available free from a number of websites (see below).

To get an alternative but similar account of our subject you should also look at Kröse and Van der Smagt (1996).

Another useful book that seems to be freely available is MacKay (2005). Although this book is too technical overall, Chapters 38–46 on neural networks might be very useful revision. Note his notation though! One good point is that the book uses Octave for examples.

The website www.scholarpedia.org/, although much more authoritative than Wikipedia, is far inferior in terms of quantity. However, you can read what the originators of some of the neural network models have to say about them. We hope that this website will be updated soon.

Here is a list of useful resources that you might like to look at:

Carnegie Mellon (2009) *Center for the Neural Basis of Cognition*. Available online at www.cnbc.cmu.edu (Last accessed June 2009.)

Clabaugh, C., D. Myszewski and J. Pang (2009) *Neural networks.* Available online at http://cse.stanford.edu/class/sophomore-college/projects-00/neural-networks/Files/presentation.html (Last accessed June 2009.)

Farabee, M. (2009) *The nervous system.* Available online at www.estrellamountain.edu/faculty/farabee/biobk/BioBookNERV.html (Last accessed June 2009.)

Fogelman-Soulié, F. and P. Gallinari *Industrial applications of neural networks.* (Singapore: World Scientific, 1998) [ISBN-10: 981023175X; ISBN-13: 9789810231750].

Hinton, G.E. (2007) *Boltzmann machine.* Available online at www.scholarpedia.org/article/Boltzmann_machine (Last accessed June 2009.)

Hopfield, J.J. (2009a) *Hopfield network.* Available online at www.scholarpedia.org/article/Hopfield_network (Last accessed June 2009.)

Hopfield, J.J. (2009b) *Website of John J. Hopfield.* Available online at www.genomics.princeton.edu/hopfield/ (Last accessed June 2009.)

Kleinberg, J. and C. Papadimitriou *Computability and Complexity in Computer Science: Reflections on the field, reflections from the field.* (Washington, D.C.: National Academies Press, 2004). Available online at www.cs.cornell.edu/home/kleinber/cstb-turing.pdf (Last accessed June 2009.)

Kohonen, T. (2009a) *Kohonen network*. Available online at www.scholarpedia.org/article/Kohonen_network (Last accessed June 2009.)

Kohonen, T. (2009b) www.cis.hut.fi/teuvo/ (Last accessed June 2009.)

Kröse, B. and P. van der Smagt *An introduction to neural networks.* (Amsterdam: University of Amsterdam, 1996) Available online at http://math.uni.lodz.pl/~fulmanp/zajecia/nn/neuro-intro.pdf and at http://en.scientificcommons.org/42887807 (Last accessed June 2009.)

Lisboa, P.J.G., B. Edisbury and A. Vellido *Business applications of neural networks: The state-of-the-art of real-world applications.* (Singapore: World Scientific, 2000) [ISBN-10: 9810240899; ISBN-13: 9789810240899].

MacKay, D.J.C. *Information theory, inference, and learning algorithms.* (Cambridge: Cambridge University Press, 2003) [ISBN-13: 9780521642989; ISBN-10: 0521642981]. Available online at www.inference.phy.cam.ac.uk/mackay/itila/book.html (Last accessed June 2009.)

McClelland, J.L., D.E. Rumelhart and the PDP Research Group *Parallel Distributed Processing: Explorations in the microstructure of cognition. Vols 1 and 2.* (San Diego: MIT Press, 1986) [ISBN-10: 026268053Xl; ISBN-13: 9780262680530].

Minsky, M. and S. Papert *Perceptrons*. (Cambridge MA: MIT Press, 1969).

Murray, A.F. *Applications of neural networks*. (New York: Springer-Verlag, 1995) [ISBN-10: 0792394429; ISBN-13: 9780792394426]. 322 pages. Available at http://books.google.com

Pham, D.T. and X. Liu *Neural networks for identification, prediction and control.* (Berlin: Springer-Verlag, 1995) [ISBN 3540199594].

Rabunal, J.R. and R. Dorrado *Artificial Neural networks in real-life applications.* (Hershey, P.A.: Idea Group Inc (IGI, 2006)) [ISBN-10: 1591409020; ISBN-13: 9781591409021].

Rojas, R (1996) *Neural networks: a systematic introduction.* (Berlin: Springer-Verlag, 1996) [ISBN 3540605053; 978354060508]. Available online at www.inf.fu-berlin.de/inst/ag-ki/rojas_home/documents/1996/NeuralNetworks/neuron.pdf (Last accessed June 2009.)

Sejnowski, T.J. and C.R. Rosenberg (1987) *Parallel networks that learn to pronounce English text.* Complex Systems 1, pp.145–168. Available online at www.cnl.salk.edu/ParallelNetsPronounce/index.php (Last accessed June 2009.)

Tang, H. and Y. Zhang *Neural Networks: Computational Models and Applications.* (New York: Springer-Verlag, 2007) [ISBN-10: 3540692258; ISBN-13: 9783540692256].

Taylor, J.G. *Neural Networks and Their Applications.* (Chichester; New York: Wiley, 1996) [ISBN-10: 0585274371; ISBN-13: 9780585274379].

UCI (2009) *Knowledge Discovery in Databases Archive.* Available online at http://kdd.ics.uci.edu/ (Last accessed June 2009.)

UK MetOffice (2009) *Historic station data: Oxford.* Available online at www.metoffice.gov.uk/climate/uk/stationdata/oxforddata.txt (Last accessed June 2009.)

Webster (2009) *Webster's online dictionary.* Available online at www.websters-online-dictionary.org/Wi/Widrow-Hoff_rule.html (Last accessed June 2009.)

Appendix C

Web resources

Googling ‘neural networks’, ‘free neural network software’, etc. will throw up lots of useful results. To get you started we have gathered some details of websites in the *Guide to the CD-ROM* document, which you can find on the CD.

The web is a wonderful thing and we will not expound these well-known traits here. However, it **does** have its limitations.

Among its problems are:

a. There is **little or no quality control** on much of what is posted. Therefore the user must carefully evaluate any resource that is relied upon. In general, University and government websites are reasonably good at quality control **but** sometimes the content of their websites is not 100 per cent accurate to say the least.

   Take care using materials from commercial sites – their owners often have a vested interest in what they have to say.

   Be very sceptical about personal sites, as these are often very inaccurate.

b. It is **dynamic**. This is of course a great thing which allows news websites to be kept up to date. However, it does mean that some or even lots of the pointers that we give you may become unavailable as the websites change – a feature that will, of course, get worse over time.

   There is some debate about how detailed a web reference should be. Some give only the top level domain because this is least likely to date. However we take an opposite view and have chosen to give full URI of the resources that we wish to recommend to you.

   Use these as a starting point and go deeper if you can. If they are out of date try removing endings incrementally.

   So for example if we give a URI www.x.y.z/a/b/c/d and this fails, try without the /d then without the /c/d, etc. It is easier for us to include the directories and for you to leave them out as necessary than it is for you to find the relevant information when we have left them out!

   Do not be frustrated if a link turns out to be ‘bad’; this is a small price to pay for having the power of the internet at your fingertips.

c. It is **not always clear what is free and what is not** – and you can waste lots of time looking for something that does not exist!

Notes

Appendix D

Coursework assignments

CIS 311 has two assignments.

Typically assignments may involve:

a. Hand simulation of a neural network.
b. The implementation of a neural network either in a programming language or using a freely available package.
c. A literature search on applications of neural networks.
d. The implementation of an application using data either supplied by us or available for download online.

D.1 Sample assignment questions and advice on answering assignments

CIS 311 Sample Coursework 1

This coursework concerns a single unit that is shown in the diagram below:

Note: Sections written in italics are not part of the coursework – they are advice to you when answering this sort of question.

Inputs	Weights	Parameter	Form	Value
1	–1	Learning rate	η	**0.2**
1	–1	Net	Σ	–2
0	1	Activation	T(<0,0,1)	0

Figure D.1

Note that if a parameter (η, say) were missing you would give it a sensible value and explain how you came about this value in your write-up. Also note that any sizes, shading and line styles in the diagram are for visual impact only – they carry no significance.

Question 1

Answer each part of this question in sequence. You should use a (non-programmable) calculator to perform the calculations, as this is meant to be practice for examination calculations. Answers should be rounded to four decimal places.

There will be a temptation to use a program or package for this – resist the temptation. By all means you can check your results against a program but the exercise is aimed at building up your ability to calculate accurately. This ability may well be tested in an examination, so don't skip the practice!

a. Describe the unit fully giving an explanation of each number and symbol that it uses.

   Use your own words – do not copy definitions out from books – but of course you should refer to these. You will not be able to use books in the examination, so it is important that you have gained the necessary understanding to express ideas in your own words. Check that you have covered all the numbers and symbols, possibly by crossing each one out in pencil from the question as you deal with it.

[10]

b. Complete the following truth table for the unit. *A* stands for its activation, of course.

?a	?b	A
–1	–1	
–1	1	
1	–1	
1	1	

This is an easy calculation ***but*** *you must also give your calculations* ***including*** *any formulae that you use. Give reference to the guide when using formulae from it and to books, etc. when using materials from elsewhere. You must not omit references – because that constitutes plagiarism.*

[8]

c. Describe the output as a function of the inputs.

It may not always be the case that the output function has a simple description but try to make your description as simple as possible. Only two marks are available, after all.

[2]

d. Train the unit using the following training set for two epochs, giving the formulae that you used and the intermediate results of each step.

?a	?b	T
0	0	1
0	1	1
1	0	1
1	1	0

You need to explain your answers. A copy of the results of a spreadsheet will gain few marks.

[8]

e. Complete the following truth table for the resulting unit.

?a	?b	A
0	0	
0	1	
1	0	
1	1	

Again working is expected even if not explicitly asked for.

[2]

f. Comment on your results.

This should be answered carefully. Sometimes there may be a straightforward answer but sometimes you may have to dig a little to find out what is going on.

[5]

Question 2

a. Implement the unit shown in Figure D.1 using a programming language or a package of your choice and describe in detail how you did this.

Notice that there are 40 marks for this so make sure that your description is worth it! Give explanations in words and in screenshots whenever possible.

[40]

b. Run the implementation for at least 100 epochs, drawing a graph of error versus epoch. Give screenshots of the results for the epochs 1, 10, 100 and the final epoch (if more than 100).

Your results will depend upon the details of the particular question asked. It might be worth using a log scale for epochs!

[10]

c. Complete the following truth table for the resulting unit.

?a	?b	A
0	0	
0	1	
1	0	
1	1	

Answers should include screenshots and not only printouts of text files.

[4]

d. Repeat a), b) and c) of this question with the training set being the exclusive OR truth table rather than the given one.

Again include screenshots.

[5]

e. Comment on the results of b) and c) and d) above.

Always think carefully about what you are going to say for questions like these. Sometimes everything is straightforward, so a simple comment will do, but at other times you may have to think quite hard to explain what is going on.

[6]

CIS 311 Sample coursework 2

Although such a venture is likely to fail, a possible topic for a neural network project is weather forecasting. This coursework is centred around a small example of this.

If you feel that statements such as the above are unjustified – or you know of successful examples – then you may give an explanation, with references, as part of your answer.

Some of the parameters of the network are to be determined from your University of London student registration number. We are asking you to do this to provide some variability in your results as a group. Use the following letters for the actual digits that are in **your** number as follows: *a* for the first, *b* the second, etc. so that student David whose student number is *071234567* would *have a = 0, b = 7, c = 1, d = 2, … h = 6* and *i = 7*. We shall use this as an example throughout this assignment.

It is important that you follow the instructions carefully.

Here is what you should do:

a. Use the internet and any other information source you have available to find out what work has been done by others in the use of neural networks for weather forecasting. Choose the five sources you find the most informative and summarise what was done, how it was done and the results, in a report no more than two pages long.

[20]

b. Implement a back propagation neural network with the following architecture:

Layer	Number of units
Input	Either h or i, whichever is the larger. Use 1 if both are zero.
Hidden	c + d + e
Output	1
	So David (student 071234567) would implement a 7 input back propagation network with a single hidden layer containing 6 (= 1 + 2 + 3) hidden units.

Test and document your implementation carefully as described in the guide.

[30]

c. i. Download the 'Historic station data' for rain for Oxford UK from the internet (see URI below). Use the rain data for the period up to and including December 2007 as a source for **training** and **selection** sets.

There is a slight chance that data from websites will become unavailable. The University will make alternative data sets available should the need arise.

ii. Use the data for the period from January 2008 as a **test** set.

In your write up you will be expected to show that you know why you are doing what you do. Ensure that you make this clear.

iii. Set up your network so that it trains using the most recent values as input. So David's network would use the previous seven rain values to predict the next one. That is, his seven-input network uses the most recent seven values available. So to predict the August 1994 value for rain, his ANN would use the seven values for January 1994 to July 1994.

iv. Train the network with the training set for as long as you reasonably can, taking measurements against the test set every 1,000 epochs.

[30]

d. Use Excel or any other tools of your choice and use the 'naïve' predictors given in the guide as alternatives against which to compare your trained network. Write up your results fully in line with the guidance given on the CD-ROM.

[20]

Data can be obtained from:

www.metoffice.gov.uk/climate/uk/stationdata/oxforddata.txt (Last accessed March 2009.)

It is very important that you give a full explanation of your work, your results, how you got them and what they mean. Try to leave nothing important unsaid.

Appendix E

Sample examination paper

Time allowed: TWO hours and FIFTEEN minutes

Answer FOUR questions.

Question 1

a. What are the main constituents of an Artificial Neural Network?

[6]

b. ANNs are often classified according to their connectivity patterns. Give the names of these classes and briefly describe how they differ.

[6]

c. Authors often include an input which acts as a bias. Describe how this is used and why it can be thought of as an input.

[4]

d. Describe the design of a 'majority voting unit' taking N inputs and outputting 1 if and only if at least half of its inputs are 1. Explain how this unit works.

[9]

Question 2

a. A sigmoidal unit is pictured in Figure E.1. Describe the information that the diagram shows.

[5]

b. Describe the operation of a unit with sigmoidal activation.

[5]

c. What is the update rule for such a unit?

[5]

Inputs	Weights	Parameter	Form	Value
1	1	Learning rate	η	**0.1**
?a	–1	Net	Σ	??
?b	–1	Activation	σ	??

Figure E.1

d. Before training this neuron, test it with the training set given in e. below.

[2]

e. Calculate the effect on the unit of training with input the following training set (show all your calculations)

?a	?b	target
1	0	0
0	2	1

[6]

f. Now test the neuron with the training set. Did it learn?

[2]

Question 3

Write notes on each of the following:

a. the difference between natural and artificial neurons

b. the concept of 'energy' in neural networks

c. Boltzmann machines

d. momentum as a means of speeding up training

e. activation functions.

[5×5]

Question 4

a. What is a Kohonen-Grossberg network?

[5]

b. Sketch the training algorithm used to train the Kohonen layer in such a network.

[5]

c. The two units of a Kohonen layer are given by (–1.3, –1.2, 1.4) and (1.2, –1.8, 1.5). The network is to be trained using a set of examples; the first one of which is: (1.5, –1.4, 2.2). Showing all of your working, calculate the units resulting from the presentation of just this one example if the learning rate is 0.2.

[15]

Question 5

Unit$_3$

Inputs	Weights	Form	Value
0	−1	η	**0.2**
a_1	0.2		
a_2	−0.1	Σ	??
?b	−0.15	σ	**??**

Unit$_1$

Inputs	Weights	Form	Value
0	−1	η	**0.2**
?a	0.1	Σ	??
		$a_1 = \sigma$	??

Unit$_2$

Inputs	Weights	Form	Value
0	−1	η	**0.2**
?a	0.3	Σ	??
?b	−0.25	$a_2 = \sigma$	??

The diagram above shows a two-input three-unit Backpropagation network. Notice that it is not layered in that Unit$_3$ receives input both from the other two units and directly from input *?b*. If the inputs are ?a = 1 and ?b =1 and the target is also 1:

i. What are the initial values for *bias$_u$*, *w$_{ih}$*, and *w$_h$*?

[6]

ii. Giving all of your workings, calculate:

a. *net$_1$, net$_2$, activation$_1$, activation$_2$, net$_3$ and activation$_3$*.

[6]

b. The errors ε_u for u = 1, 2 and 3.

[3]

c. The changes to the weights Δw_h for h = 1 and 2.

[4]

d. The changes to the weights Δw_{ih} for i=1 and 2 and h = 1 and 2.

[4]

e. The initial and final weights from *input$_2$* to *unit$_3$*.

[2]

Question 6

The diagram shows a three-unit Hopfield net:

Unit$_3$

	Inputs	Weights	Parameters	Form	Value
	1	0.2	Learning rate	η	
a_2		0.3	Net	Σ	?
a_3		–0.3	Activation	T(>0,1,0)	?

Unit$_2$

	Inputs	Weights	Parameters	Form	Value
	1	–0.4			
a_1		0.3	Net	Σ	??
a_3		0.2	Activation	T(>0,1,0)	??

Unit$_3$

	Inputs	Weights	Parameters	Form	Value
	1	–0.1			
a_1		–0.3	Net	Σ	?
a_2		0.2	Activation	T(>0,1,0)	?

a. Write down the formula that determines the activation of Unit$_3$ if it has been chosen to fire.

[5]

b. Giving your workings in your examination answer book, calculate the entries of the state transition table below and copy the completed table into your examination book.

[15]

Hopfield					***State***	***Before***			***After***			***State after***		
discrete						***3***	***2***	***1***	***3***	***2***	***1***	***3***	***2***	***1***
					0	0	0	0						
					1	0	0	1						
					2	0	1	0						
Weights	***bias***	***1***	***2***	***3***	3	0	1	1						
bias					4	1	0	0						
1					5	1	0	1						
2					6	1	1	0						
3					7	1	1	1						

c. Draw the state transition diagram of the network.

[5]

END OF PAPER

Appendix F

Artificial neural network projects

In the past, many topics have been chosen to demonstrate the power of neural networks. Some of these topics are more appropriate than others.

Firstly, one needs to decide what gain might be made if an ANN solution were available. Possible advantages might be:

a. speed of calculation
b. transparency of calculation
c. difficulty of alternatives.

A number of data sets are available on the web at http://kdd.ics.uci.edu/ (Last accessed July 2009). You may like to try to do better than others have achieved on them!

What makes an appropriate data set? This is not an easy question to answer but it is likely that something that can easily be done using conventional programming is less impressive than something that is difficult to do conventionally.

F.1 Advice on project credibility

When assessing computing projects of any type, we look for a number of indicators, telltale signs of a good project.

Firstly, there is the context of the project. Has the author set the scene by telling the reader of the importance of the topic under discussion? Predicting the weather or the stock market may seem obvious and we guess they may be important to most. However, even in these cases it is nice to have some indication of the benefits both to individuals and to mankind of such an undertaking. The current size of the market for competing solutions is also a good thing to include if you can.

Projects are rarely impressive if the author has ignored the work of others. Signs of your awareness of the work that has gone before include your citing of that work and, most importantly, a review of the important literature on your subject.

If you find that there is 'no literature on my subject' then ask yourself whether this might be because:

a. 'I missed it' – have you learnt how to search for documents effectively?
b. 'I am the only one interested in the topic' – then why do it? Of course, we are not saying that things are only worth doing if others are interested **but** you should think carefully before embarking on such an undertaking.
c. The topic is important but the results are secret – because of its importance.
d. The topic is important but no one has managed to do anything worth reporting.

You should think about the measures of success. How can you, and the reader, evaluate the worth of your results? It is not enough to report that you have trained a network and it achieves x per cent on the test set. You should report also the expected success rate of other simple techniques. For example, I can guess the gender of an individual with 50 per cent accuracy – but no one would be impressed by this!

Similarly, if I predict that the weather tomorrow will be the same as that of today, I am likely to be right more often than I am wrong. Again, no one would be impressed by this.

You should always think of naïve performance measures like this and report your results against them.

For example, if our network is to predict a time series then it is easy to invent some naïve predictors against which to measure any proposed solution.

Here are a few:

a. ‘the next value is the same as the current one’
b. ‘the next value is an extrapolation of the last few’
c. exponential smoothing (you can look this one up on the web).

These are just some of the considerations that you need to keep in mind when embarking upon a project.

And finally, do read and reread (lots of times) the specification for Projects that, for example, is to be found in the CIS320 guide. Even if you do a wonderful, ground breaking, Nobel prize winning piece of work, you may not get the marks that you expect if you have not met the specifications of the unit that you are on.

Comment form

We welcome any comments you may have on the materials which are sent to you as part of your study pack. Such feedback from students helps us in our effort to improve the materials produced for the External System.

If you have any comments about this guide, either general or specific (including corrections, non-availability of essential texts, etc.), please take the time to complete and return this form.

Title of this subject guide: ....

Name ....

Address ....

....

Email ....

Student number ....

For which qualification are you studying? ....

Comments

....

....

....

....

....

....

....

....

....

....

....

....

....

....

....

....

....

Please continue on additional sheets if necessary.

Date: ....

Please send your comments on this form (or a photocopy of it) to:
Publishing Manager, External System, University of London, Stewart House, 32 Russell Square, London WC1B 5DN, UK.