Artificial Neural Networks and Information Theory

Colin Fyfe,
Department of Computing and Information Systems,
The University of Paisley.
Room H242
Phone 848 3305.

Edition 1.2,
2000

Contents

Chapter 1

Introduction

1.1 Objectives

After this chapter, you should

1. understand the basic building blocks of artificial neural networks (ANNs)
2. understand the two modes of operation in ANNs
3. understand the importance of learning in ANNs
4. be able to use a simple rule to create learning in an ANN
5. begin to understand the importance of linearly inseparable problems
6. know some of the problems on which ANNs have been used.

1.2 Intelligence

This course comprises an advanced course to those new information processing simulations which are intended to emulate the information processors which we find in biology.

Traditional artificial intelligence is based on high-level symbol processing i.e. logic programming, expert systems, semantic nets etc all rely on there being in existence some high level representation of knowledge which can be manipulated by using some type of formal syntax manipulation scheme - the rules of a grammar. Such approaches have proved to be very successful in emulating human prowess in a number of fields e.g.

- we now have software which can play chess at Grand Master level
- we can match professional expertise in medecine or the law using expert systems
- we have software which can create mathematical proofs for solving complex mathematical problems.

Yet there are still areas of human expertise which we are unable to mimic using software e.g. our machines have difficulty reliably reading human handwriting, recognising human faces or exhibiting common sense. Notice how low-level the last list seems compared to the list of achievements: it has been said that the difficult things have proved easy to program whereas the easy things have proved difficult.

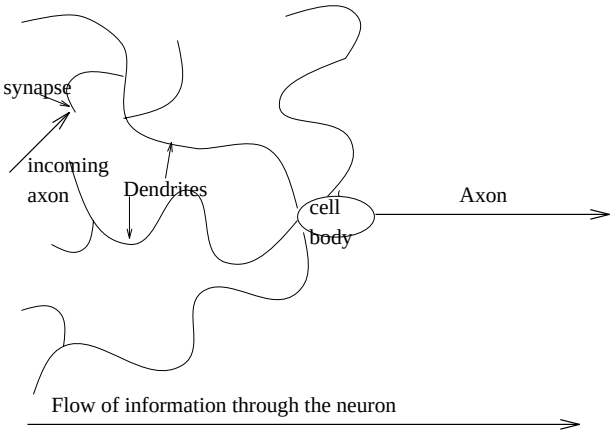

Figure 1.1: A simplified neuron

1.3 Artificial Neural Networks

Now tasks such as those discussed above seemingly require no great human expertise - young children are adept at many of these tasks. This explains the underlying presumption of creating artificial neural networks (ANNs): that the expertise which we show in this area is due to the nature of the hardware on which our brains run. Therefore if we are to emulate biological proficiencies in these areas we must base our machines on hardware (or simulations of such hardware) which seems to be a silicon equivalent to that found within our heads.

First we should be clear about what the attractive properties of human neural information processing are. They may be described as :

- Biological information processing is robust and fault-tolerant: early on in life[1], we have our greatest number of neurons yet though we daily lose many thousands of neurons we continue to function for many years without an associated deterioration in our capabilities
- Biological information processors are flexible: we do not require to be reprogrammed when we go into a new environment; we adapt to the new environment, i.e. we learn.
- We can handle fuzzy, probabilistic, noisy and inconsistent data in a way that is possible with computer programs but only with a great deal of sophisticated programming and then only when the context of such data has been analysed in detail. Contrast this with our innate ability to handle uncertainty.
- The machine which is performing these functions is highly parallel, small, compact and dissipates little power.

We shall therefore begin our investigation of these properties with a look at the biological machine we shall be emulating.

1.4 Biological and Silicon Neurons

A simplified neuron is shown in Figure 1.1. Information is received by the neuron at synapses on its dendrites. Each synapse represents the junction of an incoming axon from another neuron with a dendrite of the neuron represented in the figure; an electro-chemical transmission occurs at the synapse which allows information to be transmitted from one neuron to the next. The information is then transmitted along the dendrites till it reaches the cell body where a summation of the electrical impulses reaching the body takes place and some function of this sum is performed. If

[1] Actually several weeks before birth

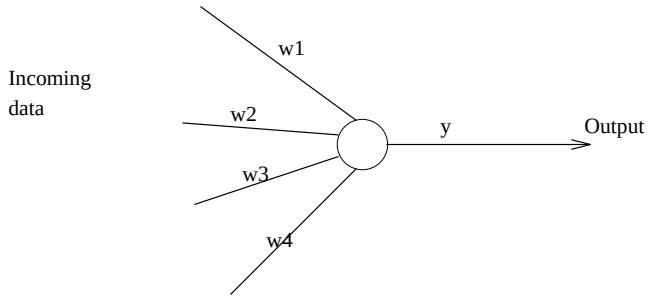

Figure 1.2: The artificial neuron. The weights model the synaptic efficiencies. Some form of processing not specified in the diagram will take place within the cell body.

this function is greater than a particular threshold the neuron will fire: this means that it will send a signal (in the form of a wave of ionisation) along its axon in order to communicate with other neurons. In this way, information is passed from one part of the network of neurons to another. It is crucial to recognise that synapses are thought to have different efficiencies and that these efficiencies change during the neuron's lifetime. We will return to this feature when we discuss learning.

We generally model the biological neuron as shown in Figure 1.2. The inputs are represented by the input vector $\mathbf{x}$ and the synapses' efficiencies are modelled by a weight vector $\mathbf{w}$. Therefore the single output value of this neuron is given by

$$y = f(\sum_i w_i x_i) = f(\mathbf{w}.\mathbf{x}) = f(\mathbf{w}^T\mathbf{x}) \tag{1.1}$$

You will meet all 3 ways of representing the operation of summing the weighted inputs. Sometimes f() will be the identity function i.e. f($\mathbf{x}$)=$\mathbf{x}$. Notice that if the weight between two neurons is positive, the input neuron's effect may be described as excitatory; if the weight between two neurons is negative, the input neuron's effect may be described as inhibitory.

Consider again Figure 1.2. Let $w_1 = 1, w_2 = 2, w_3 = -3$ and $w_4 = 3$ and let the activation function, f(), be the Heaviside (threshold) function such that

$$f(t) = \begin{cases} 1 & \text{if } t > 0 \\ 0 & \text{if } t \leq 0 \end{cases} \tag{1.2}$$

Now if the input vector $\mathbf{x} = (x_1, x_2, x_3, x_4)$ = (1,2,1,2) then the activation of the neuron is $\mathbf{w}.\mathbf{x}$ = $\sum_j w_j x_j$ = 1*1 + 2*2 + 1*(-3) + 2*3 = 8 and so y = f(8) =1. However if the input vector is (3,1,2,0), then the activation is 3*1 + 1*2 + 2*(-3) + 0*3 = -1 and so y = f(-1) =0.

Therefore we can see that the single neuron is an extremely simple processing unit. The power of neural networks is believed to come from the accumulated power of adding many of these simple processing units together - i.e. we throw lots of simple and robust power at a problem. Again we may be thought to be emulating nature, as the typical human has several hundred billion neurons. We will often imagine the neurons to be acting in concert in layers such as in Figure 1.3.

In this figure, we have a set of inputs (the input vector, $\mathbf{x}$) entering the network from the left-hand side and being propagated through the network via the weights till the activation reaches the output layer. The middle layer is known as the hidden layer as it is invisible from outwith the net: we may not affect its activation directly.

1.5 Learning in Artificial Neural Networks

There are two modes in artificial neural networks:

1. activation transfer mode when activation is transmitted throughout the network

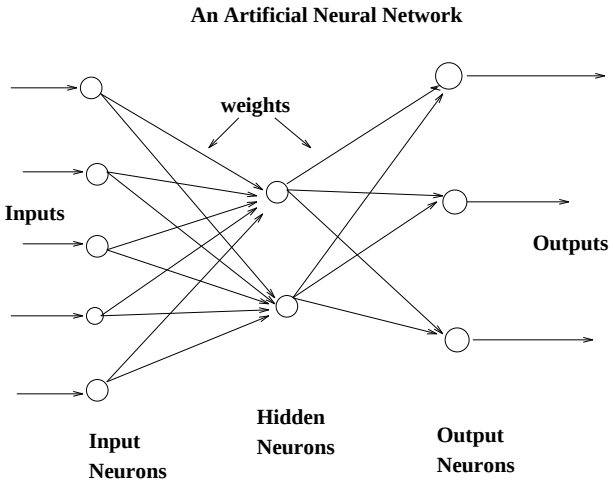

Figure 1.3: A typical artificial neural network consisting of 3 layers of neurons and 2 conecting layers of weights

2. learning mode when the network organises usually on the basis of the most recent activation transfer.

We will now consider the learning mode.

We stated that neural networks need not be programmed when they encounter novel environments. Yet their behaviour changes in order to adapt to the new environment. Such behavioural changes are due to changes in the weights in the network. We call the changes in weights in a neural network learning. The changes in weights in an artificial neural network are intended to model the changing synaptic efficiencies in real neural networks: it is believed that our learning is due to changes in the efficiency with which synapses pass information between neurons.

There are 3 main types of learning in a neural network:

Supervised learning: with this type of learning, we provide the network with input data and the correct answer i.e. what output we wish to receive given that input data. The input data is propagated forward through the network till activation reaches the output neurons. We can then compare the answer which the network has calculated with that which we wished to get. If the answers agree, we need make no change to the network; if, however, the answer which the network is giving is different from that which we wished then we adjust the weights to ensure that the network is more likely to give the correct answer in future if it is again presented with the same (or similar) input data. This weight adjustment scheme is known as supervised learning or learning with a teacher. The tutorial at the end of this chapter gives an example of supervised learning.

Unsupervised learning: with this type of learning, we only provide the network with the input data. The network is required to self-organise (i.e. to teach itself) depending on some structure in the input data. Typically this structure may be some form of redundancy in the input data or clusters in the data.

Reinforcement learning: is a half-way house between these two: we provide the network with the input data and propagate the activation forward but only tell the network if it has produced a right or a wrong answer. If it has produced a wrong answer some adjustment of the weights is done so that a right answer is more likely in future presentations of that particular piece of input data.

For many problems, the interesting facet of learning is not just that the input patterns may be learned/classified/identified precisely but that this learning has the capacity to generalise. That

is, while learning will take place on a set of *training patterns* an important property of the learning is that the network can generalise its results on a set of *test patterns* which it has not seen during learning. One of the important consequences here is that there is a danger of overlearning a set of training patterns so that new patterns which are not part of the training set are not properly classified.

For much of this course we will concentrate on unsupervised learning; the major exceptions occur in Chapters 5 and 8 in which we will use supervised learning methods.

1.6 Typical Problem Areas

The number of application areas in which artificial neural networks are used is growing daily. Here we simply produce a few representative types of problems on which neural networks have been used

Pattern completion: ANNs can be trained on sets of visual patterns represented by pixel values. If subsequently, a part of an individual pattern (or a noisy pattern) is presented to the network, we can allow the network's activation to propagate through the network till it converges to the original (memorised) visual pattern. The network is acting like a content-addressable memory. Typically such networks have a recurrent (feedback as opposed to a feedforward) aspect to their activation passing. You will sometimes see this described as a network's *topology.*

Classification: An early example of this type of network was trained to differentiate between male and female faces. It is actually very difficult to create an algorithm to do so yet an ANN has been shown to have near-human capacity to do so.

Optimisation: It is notoriously difficult to find algorithms for solving optimisation problems. A famous optimisation problem is the Travelling Salesman Problem in which a salesman must travel to each of a number of cities, visiting each one once and only once in an optimal (i.e. least distance or least cost) route. There are several types of neural networks which have been shown to converge to 'good-enough' solutions to this problem i.e. solutions which may not be globally optimal but can be shown to be close to the global optimum for any given set of parameters.

Feature detection: An early example of this is the phoneme producing feature map of Kohonen: the network is provided with a set of inputs and must learn to pronounce the words; in doing so, it must identify a set of features which are important in phoneme production.

Data compression: There are many ANNs which have been shown to be capable of representing input data in a compressed format losing as little of the information as possible; for example, in image compression we may show that a great deal of the information given by a pixel to pixel representation of the data is redundant and a more compact representation of the image can be found by ANNs.

Approximation: Given examples of an input to output mapping, a neural network can be trained to approximate the mapping so that a future input will give approximately the correct answer i.e. the answer which the mapping should give.

Association: We may associate a particular input with a particular output so that given the same (or similar) output again, the network will give the same (or a similar) output again.

Prediction: This task may be stated as: given a set of previous examples from a time series, such as a set of closing prices for the FTSE, to predict the next (future) sample.

Control: For example to control the movement of a robot arm (or truck, or any non-linear process) to learn what inputs (actions) will have the correct outputs (results).

1.7 A short history of ANNs

The history of ANNs started as long ago as 1943 when McCullogh and Pitts showed that simple neuron-like building blocks were capable of performing all possible logic operations. At that time too, Von Neumann and Turing discussed interesting aspects of the statistical and robust nature of brain-like information processing, but it was only in the 1950s that actual hardware implementations of such networks began to be produced. The most enthusiastic proponent of the new learning machines was Frank Rosenblatt who invented the *perceptron* machine, which was able to perform simple pattern classification.

However, it became apparant that the new learning machines were incapable of solving certain problems and in 1969 Minsky and Papert wrote a definitive treatise, 'Perceptrons', which clearly demonstrated that the networks of that time had limitations which could not be transcended. The core of the argument against networks of that time may be found in their inability to solve XOR problems (see Chapter 5). Enthusiasm for ANNs decreased till the mid '80s when first John Hopfield, a physicist, analysed a particular class of ANNs and proved that they had powerful pattern completion properties and then in 1986 the subject really took off when Rumelhart, McClelland and the PDP Group rediscovered powerful learning rules which transcended the limitations discussed by Minsky and Papert.

1.8 A First Tutorial

This tutorial will emphasise learning in neural nets. Recall that learning is believed to happen at the synapses (the meeting points between neurons) and that we model synaptic efficiency with weights.

You are going to hand simulate a simple neural net (see Figure 1.4) performing classification according to the AND (see table) OR and XOR rules. We will use a network with three input neurons - the two required for the input values and a third neuron known as the bias neuron. The bias always fires 1 (i.e. is constant).

You will work in groups of 3 - one person in charge of selecting the input, one in charge of calculating the feedforward value of the neuron, and one person in charge of calculating the change in weights.

To begin with, the group selects random (between 0 and 1) values for the three weights shown in the figure.

1. The INPUTER selects randomly from the set of patterns shown in the table and places the cards in the appropriate places on the table.

2. The FEEDFORWARDER multiplies the weights by the input patterns to calculate the output.

$$y = \sum_{i=0}^{2} w_i x_i \tag{1.3}$$

   Then y = 1 if $y > 0$, y = -1 if $y < 0$.

3. The WEIGHTCHANGER changes the weights *when the value of y is not the same as the target* y_T according to the formula

$$w_i = w_i + \eta * (y_T - y) * x_i \tag{1.4}$$

Steps (1)-(3) are repeated as often as necessary.

1.8.1 Worked Example

Let us have initial values $w_0 = 0.5, w_1 = 0.3, w_2 = 0.7$ and let η be 0.1.

Bias	first input	second input	target output
1	1	1	1
1	1	-1	-1
1	-1	1	-1
1	-1	-1	-1

Table 1.1: The values for the AND patterns

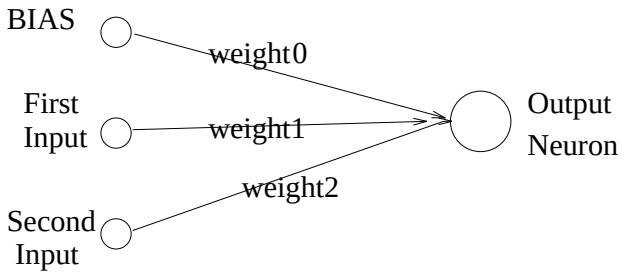

Figure 1.4: The Simple Neural Network

1. "Randomly" select pattern 1. The FEEDFORWARDER calculates y = 0.5 +0.3 +0.7 =1.5. So y=1 which is the same as the target and so the WEIGHTCHANGER does nothing.

2. Imagine pattern 2 is chosen. The FEEDFORWARDER calculates y = 0.5+0.3 -0.7 = 0.1. So y=1. Now y_T = -1 and so WEIGHTCHANGER calculates

$$\begin{aligned} w_0 &= w_0 + 0.1 * (-2) * 1 = 0.5 - 0.2 = 0.3 \\ w_1 &= w_1 + 0.1 * (-2) * 1 = 0.3 - 0.2 = 0.1 \\ w_2 &= w_2 + 0.1 * (-2) * (-1) = 0.7 + 0.2 = 0.9 \end{aligned}$$

3. Now pattern 3 is chosen. The FEEDFORWARDER calculates y = 0.3 - 0.1 + 0.9 = 1.1. So y =1 and y_T = -1 and so WEIGHTCHANGER calculates

$$\begin{aligned} w_0 &= w_0 + 0.1 * (-2) * 1 = 0.3 - 0.2 = 0.1 \\ w_1 &= w_1 + 0.1 * (-2) * (-1) = 0.1 + 0.2 = 0.3 \\ w_2 &= w_2 + 0.1 * (-2) * 1 = 0.9 - 0.2 = 0.7 \end{aligned}$$

4. Now pattern 4 is chosen. The FEEDFORWARDER calculates y = 0.1 - 0.3 - 0.7 = -0.9. So y = -1 and y_T = -1. Therefore the WEIGHTCHANGER does nothing.

5. Now select pattern 2. The FEEDFORWARDER calculates y = 0.1 - 0.3 + 0.7 =0.5. Then y =1 but y_T = -1. WEIGHTCHANGER calculates

$$\begin{aligned} w_0 &= w_0 + 0.1 * (-2) * 1 = 0.1 - 0.2 = -0.1 \\ w_1 &= w_1 + 0.1 * (-2) * (-1) = 0.3 + 0.2 = 0.5 \\ w_2 &= w_2 + 0.1 * (-2) * 1 = 0.7 - 0.2 = 0.5 \end{aligned}$$

6. Now all of the patterns give the correct response (try it).

We can draw the solution found by using the weights as the parameters of the line $ax_1+bx_2+c = 0$. Using $a = w_1, b = w_2, c = w_0$ we get

$$0.5x_1 + 0.5x_2 - 0.1 = 0 \tag{1.5}$$

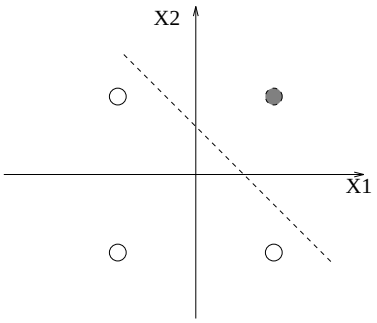

Figure 1.5: The line joining (0,0.2) and (0.2,0) cuts the 4 points into 2 sets correctly

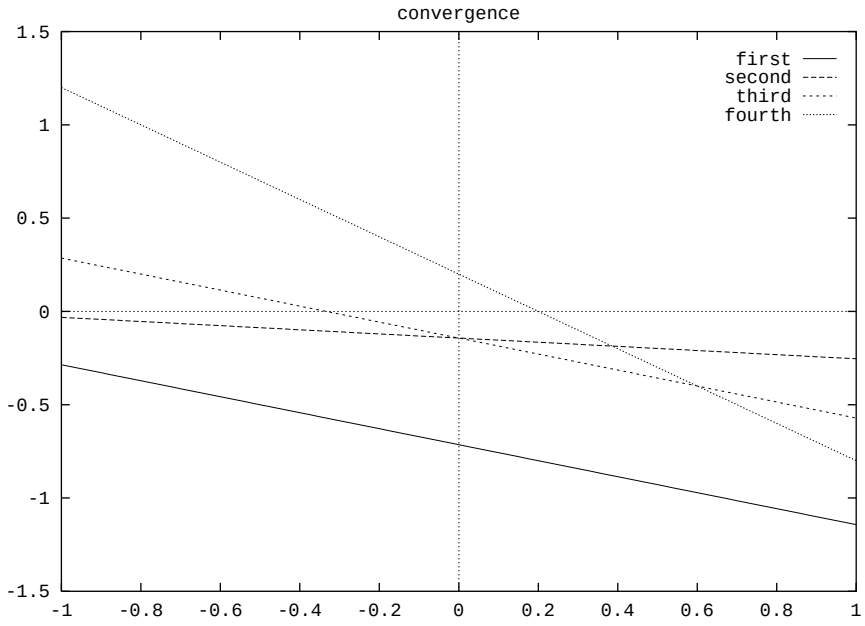

Figure 1.6: The iterative convergence of the network to a set of weights which can perform the correct mapping is shown diagrammatically here.

which we can draw by getting two points. If $x_1 = 0$, then $0.5x_2 = 0.1$ and so $x_2 = 0.2$ which gives us one point (0,0.2). Similarly we can find another point (0.2,0) which is drawn in Figure 1.5. Notice the importance of the BIAS weight: it allows a solution to be found which does not go through the origin; without a bias we would have to have a moving threshold.

We can see the convergence of $w_0 + w_1x_1 + w_2x_2 = 0$ in Figure 1.6. Notice that initially 2 patterns are correctly classified, very quickly a third is correctly classified and only on the fourth change are all 4 patterns correctly classified.

1.8.2 Exercises

1. Repeat the worked example with different initial values for the weights. (Objectives 3, 4).
2. Repeat for the OR patterns. (Objectives 3, 4).
3. Experiment with different initial values, learning rates. (Objectives 3, 4).
4. Now do the XOR problem. Don't spend too long on it - it's impossible. (Objective 5).

Nut	type A - 1	type A - 2	type A - 3	type B - 1	type B - 2	type B - 3
Length (cm)	2.2	1.5	0.6	2.3	1.3	0.3
Weight (g)	1.4	1.0	0.5	2.0	1.5	1.0

Table 1.2: The lengths and weights of six instances of two types of nuts

5. Draw the XOR coordinates and try to understand why it is impossible. (Objective 5).

6. Now we will attempt to train a network to classify the data shown in Table 1.2. Then we will train a network with the input vectors , **x**, equal to
   (1, 2.2, 1.4) with associated training output 1 (equal to class A)
   (1, 1.5, 1.0) with associated training output 1
   (1, 0.6, 0.5) with associated training output 1
   (1, 2.3, 2.0) with associated training output -1 (equal to class B)
   (1, 1.3,1.5) with associated training output -1
   (1, 0.3,1.0) with associated training output -1
   Note that the initial 1 in each case corresponds to the bias term

7. Describe in your own words an Artificial Neural Network and its operation (Objectives 1, 2).

8. Describe some typical problems which people have used ANNs to solve. Attempt to describe what features of the problems have made them attractive to ANN solutions. (Objective 6).