8: Alternative node types

Kevin Gurney

Dept. Human Sciences, Brunel University
Uxbridge, Middx. UK

In the first lecture the basic functionality of the most popular artificial neuron - the semilinear unit - was introduced. Thus, the activation a was defined as a linear weighted sum of inputs, $a = \mathbf{w} \cdot \mathbf{x}$, and the output y as the sigmoid of the activation $y = \sigma(a)$. Some variants on this have occurred but the linear-weighted-sum-of-inputs activation has been common to them all. In this lecture, two alternative families of node are introduced which, at first appear dissimilar, but which may be shown to be essentially equivalent. Their possible connection with biological neurons is also outlined.

As a step to introducing alternative types of artificial neuron, note the following features of the semilinear node. The input- output function of the unit is defined in two stages. First, there is an activation which is a function of the inputs x_i and some internal node parameters ζ_j (Greek 'zeta'); that is, $a = a(x_i, \zeta_j)$. For all node types discussed so far the internal parameters are just the weights and the threshold, $\{\zeta_j\} = \{w_k, \theta\}$. The second stage consists of modifying the activation by some non-linearity to produce the final output y; that is $y = g(a)$. So far $g()$ has been either a step function or a sigmoid. This part will be retained - it is the activation which will be modified in what follows.

1 Cubic Nodes

1.1 Using computer memories to generate the activation

Suppose we restrict ourselves to binary inputs which take the values 0 or 1. Consider now the computer memory component shown below.

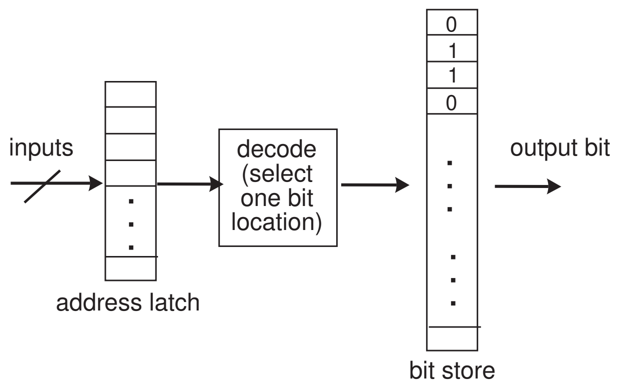
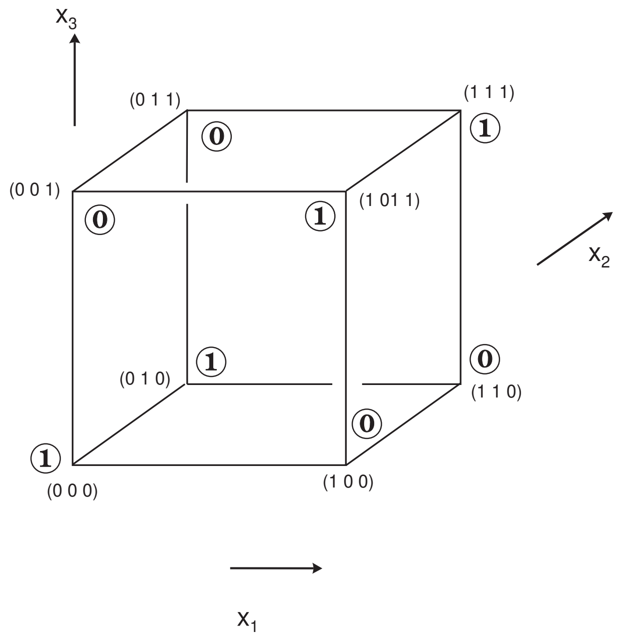


diagram of RAM

Such memories are usually referred to as Random Access Memories or RAMs. * The inputs form an *address* which is decoded to locate one of the memory cells whose contents, either a 0 or a 1, are then read out. If there are n inputs there will be 2^n possible addresses since each location in the address may be either a 0 or a 1. There will therefore be 2^n memory cells whose values, in correspondence with their addresses, may be drawn up in a table similar to the ones used previously to portray the functionality of 2-input TLUs. A 3-input example is shown below.

0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

We may think of the addresses as locating the vertices or *sites* at the corners of the n-dimensional hypercube, just as we did for the TLUs. This is the preferred way of viewing things as it allows a geometric interpretation of generalisation. (next lecture).



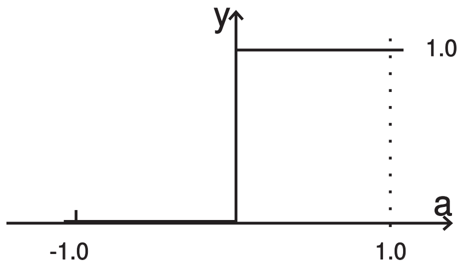
3-cube of table above

However, the key difference now is that we are *not* restricted to those functions which are linearly separable; each RAM location is able to be set to either value independently of the others. This extra functionality may prove useful in solving certain problems and, as discussed later, at least a subset of it may be involved in biological neural processing. The

*This terminology is now largely a misnomer and is rooted in the historical development of computer memories, some of which had to be accessed serially rather than allowing access to arbitrary locations at any time.

further advantage is, of course, the ability to implement our neurons easily in readily available digital hardware since RAMs are a commonplace component in all conventional computers.

In order to accommodate the RAM-lookup table into the general two stage, activation-output scheme described above, it is useful to think of the storage cells as containing the polarised binary values ± 1 (see notes on Hopfield storage prescription).



hard limiter for a RAM

The operation of a RAM as an artificial node now proceeds as follows: A binary input vector is applied to the RAM and is used as the memory address; the (polarised) binary value addressed is read out from the site located and this is then 'converted' back to its unpolarised form using a step function. Within the formalism given above, the internal parameters are now the hypercube site values (memory contents), and the activation is the polarised site value addressed. The site at address μ will be designated S_μ so that $a = a(S_\mu, x_i)$. The function $g(a)$ is then just step function.

RAMs figure strongly in the WISARD machine which was developed at Brunel (Aleksander and Stonham, 1979). The WISARD uses them in a very specific architecture and may be thought of as a hardware implementation of the n-tuple technique developed by Bledsoe & Browning (1959). However, our aim here is to show how the basic node structure described above may be extended and used in conventional network architectures and trained using the well known algorithms. We shall refer to nodes based on the RAM-lookup table model as *cubic nodes* in order to emphasise the geometric view which thinks of them as defined by a population of values at the vertices of the n-cube.

1.2 Writing the activation in closed form

The key that enabled us to develop training algorithms for semilinear nodes was the ability to make explicit the form of the activation

$$a(w_i, x_i) = \sum_{i=1}^{n} w_i x_i \tag{1}$$

That is, it is a 'well-behaved' function of the weights and the inputs which may be manipulated according to normal mathematical procedures. We could then evaluate the gradients, for example, of an error with respect to the weights †. At the moment the activation for cubic nodes is just a lookup table - there is no expression corresponding to the right hand side of (1). As a step towards such a closed expression, consider a 2-input cubic node. There

†Although we didn't actually do this in the lectures - only plausibility arguments were given - this is the route for a rigorous derivation of BP and the delta rule

are four site values $\{S_{00}, S_{01}, S_{10}, S_{11}\}$, and I now claim that the following is an expression for the activation

$$a = S_{00}(1-x_1)(1-x_2) + S_{01}(1-x_1)x_2 + S_{10}x_1(1-x_2) + S_{11}x_1x_2 \tag{2}$$

The reason for this is that for any input (address), only one of the product terms in the input variables is non-zero. Thus the expression 'picks out' the relevant site value or activation. Consider for example the input factors in the second term $(1-x_1)x_2$. This is zero unless $x_1 = 0, x_2 = 1$, that is we have the input (0,1), and then it is just equal to 1. In this case, by similar arguments, all the other input products in the other terms are zero. So we compute $a = S_{01}$ which is what is required.

The expression in (2) may be made more symmetrical in the inputs if we use their polarised form. To highlight this we place a 'bar' over the values so that the inputs are now denoted by $(\bar{x}_1, \bar{x}_2)$. The relation between the two forms is

$$x_i = \frac{1}{2}(\bar{x}_i + 1) \tag{3}$$

Then substituting this in (2)

$$\begin{aligned} a \;=\; & S_{00}\frac{(1-\bar{x}_1)}{2}\frac{(1-\bar{x}_2)}{2} + S_{01}\frac{(1-\bar{x}_1)}{2}\frac{(1+\bar{x}_2)}{2} + \\ & S_{10}\frac{(1+\bar{x}_1)}{2}\frac{(1-\bar{x}_2)}{2} + S_{11}\frac{(1+\bar{x}_1)}{2}\frac{(1+\bar{x}_2)}{2} \end{aligned} \tag{4}$$

The pattern of signs in the brackets is clearly determined by the pattern of 1s and 0s in the address μ of the site associated with these terms. Thus, if the address μ is written out in terms of its polarised components as $\bar{\mu}_1\bar{\mu}_2$, then the general term in (4) is given by

$$S_\mu \frac{(1+\bar{\mu}_1\bar{x}_1)}{2}\frac{(1+\bar{\mu}_2\bar{x}_2)}{2} \tag{5}$$

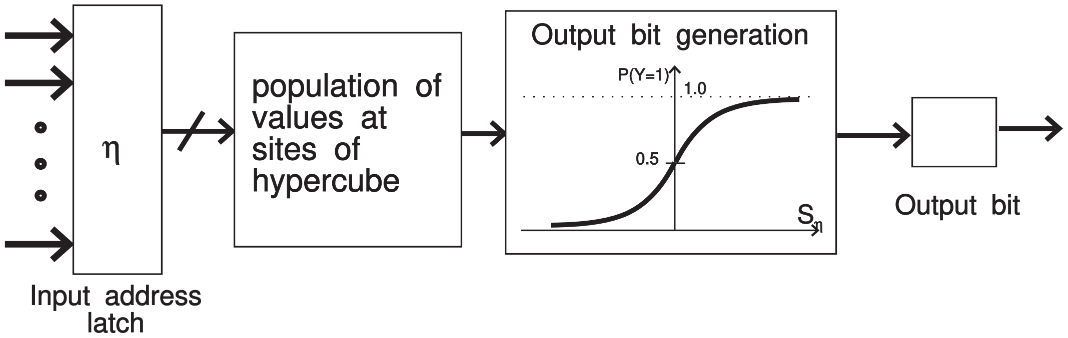
In the general n-input case we have

$$a = \frac{1}{2^n}\sum_\mu S_\mu \prod_{i=1}^{n}(1+\bar{\mu}_i\bar{x}_i) \tag{6}$$

The symbol Π (greek upper case 'pi') means 'product' and all terms under it are multiplied together.

1.3 Training cubic nodes: loss of information and its solution

Cubic nodes get trained by changing their site values. Let '+' denote an increment operation - change the site value to 1, and '-' a decrement operation - change the site value to -1. Suppose a sequence of training steps occurs for a site of the form $++-++++++-$, that is 8 increments and 2 decrements. The final value of the site is given by the last training operation, in this case a -1. However, it is clear that training is trying, on average, to increment the site; we are losing information. The solution - as noted as long ago as 1962 by Bledsoe & Blisson (Bledsoe and Blisson, 1962) - is to allow the site to take on more than two values. Thus, we allow sites to take values in the range $(-S_m, -S_m+1, \ldots, -1, 0, 1, \ldots S_m - 1, S_m)$. Now, applying the same training sequence to a site with $S_m = 10$, starting with $S_\mu = 0$ we get the following values for the site as it is trained: 1,2,1,2,3,4,5,6,7,6. The output function needs to reflect this flexibility and we now make use of a sigmoid rather than the hard-limiter.



Full cubic node with sigmoid output

Of course, to work with these nodes mathematically it is necessary to assume that the sites are continuous, just as the weights were for semilinear nodes. Only then does it make sense to take gradients, etc. with respect to the site values. In any implementation however, we would allow only discrete site values as described above, so that we may take advantage of RAM technology. Using site values that may only take on a set of discrete values introduces noise into learning since the exactly computed site changes cannot be made.

1.4 Working with analogue inputs

Clearly we can't simply apply non-binary numbers to the inputs of a cubic node; there has to be a binary address in order to locate one of the sites on the cube. The way out is to communicate analogue values by stochastic bit streams (recall stochastic semilinear nodes). The result is that the $\bar{x}_i$ get reinterpreted as 'polarised probabilities' (simply probabilities rescaled to the interval [-1, 1]). Full details may be found in (Gurney, 1992a; Gurney, 1992b).

2 Sigma-Pi nodes

Consider a 3 input semilinear unit. Its activation is

$$a = w_1 x_1 + w_2 x_2 + w_3 x_3 \tag{7}$$

This supposes that each input contributes to the activation independently of the others. That is, the contribution to the activation from input 1 say, is always a constant multiplier (w_1) times x_1. Suppose however, that the contribution from input 1 depends also on input 2 and that, the larger input 2, the larger is input 1's contribution. The simplest way of modelling this is to include a term in the activation like $w_{12}x_1x_2$ where $w_{12} > 0$ (for a diminishing influence of input 2 we would, of course, have $w_{12} < 0$). In general we might have terms containing all possible pairs of inputs and also a term in the three inputs together

$$\begin{aligned} a \;=\; & w_1 x_1 + w_2 x_2 + w_3 x_3 + \\ & w_{12} x_1 x_2 + w13 x_1 x_3 + w_{23} x_2 x_3 + \\ & w_{123} x_1 x_2 x_3 \end{aligned} \tag{8}$$

In general for n-inputs

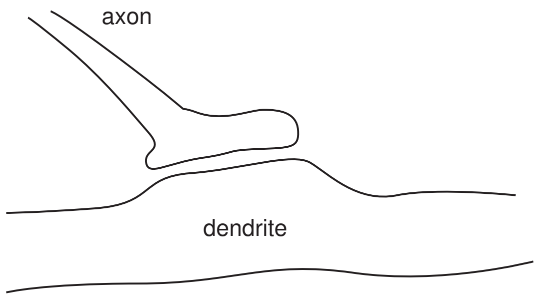
$$a = \sum_{k=1}^{N} w_k \prod_{i \in I_k} x_i \tag{9}$$

where I_k is the kth in a series of index sets, each of which contains one of the possible 2^n selections of the first n integers. There are therefore 2^n 'weights' in this type of node. The presence of the 'sigma' and 'pi' symbols together here gives rise to the term *sigma-pi* units. (PDP vol 1 page 72-74). Now, although the terms contain products of inputs, there are no powers of each input greater than one; this gives rise to the name *multilinear* for the terms in this kind of expression. Nodes with multilnear terms are also sometimes called *higher- order* nodes, since their activation depends on terms whose multiplicative order is greater than one.

Comparison of (9) and (6) shows that they are superficially similar. In fact, after a little manipulation, the latter expression may be cast in exactly the same form as the first. Thus, cubic nodes may be thought of as a kind of sigma-pi node (Gurney, 1992b).

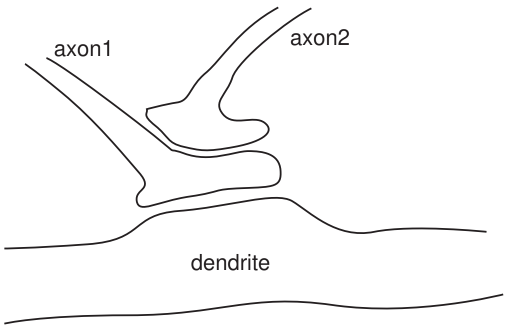
3 Biological neurons and sigma-pi units

The stereotypical synapse shown below is the inspiration for most neural net modelling and was introduced in lecture 1. It consists of an electro-chemical connection between an axon and a dendrite - hence it is an *axo-dendritic* synapse



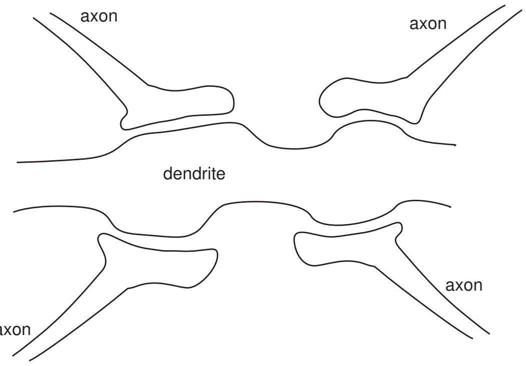
[axo-dendritic synapse]

However there is a large variety of synaptic types and connection grouping (See for example, Dayhoff ch. 8 for a review). Of special importance here are the pre-synaptic inhibitory synapses of the serial variety and clusters of densely packed axo- dendritic synapses



[presynaptic inhibition]

Here the efficacy of the axo-dendritic synapse between axon 1 and the dendrite is modulated (inhibited) by the activity in axon 2 via the axo-axonic synapse between the two axons. This might therefore be modelled by a quadratic term like $w_{12}x_1x_2$



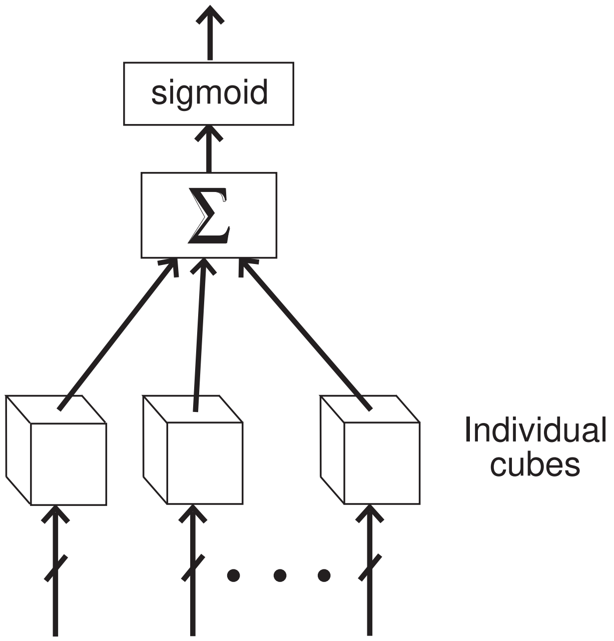
[synapse cluster]

Here the effect of the individual synapses will surely not be independent and we should look to model this with a multilinear term in all the inputs.

4 Pruning sigma-pi units: the multi-cube node

Consider a cubic node with n-inputs. There are 2^n sites. For $n = 8$ this gives 256 sites. For $n = 32$ this gives $2^{32} \approx 10^9$ sites. Clearly there is an explosion of the number of sites as n grows which is not only impractical from an implementation point of view, but also suspect from a theoretical viewpoint: do we really need to make use of all the functional possibilities available in such nodes? (remember problem sheet 2, and how the number of functions increases as the number of inputs).

One way of overcoming this is developed in the *Multi-cube unit* or MCU (Gurney, 1992a; Gurney, 1992b) where several small cubes sum their outputs to form the activation.



[MCU]

This also has a biological analogue in relation to the synaptic clusters described above. The small cubes are supposed to correspond to these clusters which then sum their contributions linearly. Mathematically, in terms of the sigma-pi form, we are limiting the order of terms that are used. So for example, if the cubes all have just 4 inputs, there can only be terms containing, at the most, products of 4 inputs.

5 The terminological Tower of Babel...

I have used the term 'cubic node' to denote one in which the activation is found by looking up the value at the corner of a hypercube. I do this to give an implementation-free viewpoint although, as we have seen, the nodes are equivalent to a RAM lookup with several bits stored at each memory location. Because of Aleksander's emphasis on the hardware, he and his group accent the RAM-based approach and talk of RAM-nets. Further, when talking about the multivalued site nodes with sigmoidal output, they talk of Probabilistic Logic Nodes or PLNs. Aleksander has also tried recently to emphasise the distinction to be made with normal weighted units by referring to them as 'weightless' nodes. However, the sigma-pi equivalence that I have noted would lead me to counter this by saying that they are far from weightless! John Taylor has his own set of acronyms which is centered on that for 'probabilistic RAM' or pRAM'.

There was almost a consensus about 3 years ago when everyone agreed to call them generically 'digital nodes' since they could be implemented using digital hardware. This was pretty sensible and therefore seems to have been abandoned!

A more complete exposition of the cube based (implementation-free) approach may be found in (Gurney, 1989) which is available in a (physically) more compact form as a Technical Memo.

References

Aleksander, I. and Stonham, T. (1979). Guide to pattern recognistion using random-access memories. *Computers and digital techniques*, 2:29 – 40.

Bledsoe, W. and Blisson, C. (1962). Improved memory matrices for the n-tupel pattern recognition method. *IRE Transactions on Electronic Computers*, EC11:414 – 415.

Bledsoe, W. and Browning, I. (1959). Pattern recognition and reading by machines. In *Proceedings of the Eastern Joint Computer Conference*, pages 225 – 232.

Gurney, K. (1989). *Learning in networks of structured hypercubes.* PhD thesis, Dept. Electrical Engineering, Brunel University, Uxbridge, Middx, UK. Available as Technical Memorandum CN/R/144.

Gurney, K. (1992a). Training nets of hardware realisable sigma-pi units. *Neural Networks*, 5:289 – 303.

Gurney, K. (1992b). Weighted nodes and ram-nets: A unified approach. *Journal of Intelligent Systems*, 2:155 – 186.