Neural Networks

Fundamentals

Framework for distributed processing

Network topologies

Training of ANN's

Notation

Perceptron

Back Propagation

Neural Networks – Historical Perspective

A first wave of interest in neural networks also known as connectionist models or parallel distributed processing emerged after the introduction of simplfied neurons by McCulloch and Pitts in 1943. These neurons were presented as models of biological neurons and as conceptual components for circuits that could perform computational tasks

When Minsky and Papert published their book Perceptrons in 1969 in which they showed the deficiencies of perceptron models most neural network funding was redirected and researchers left the field Only a few researchers continued their efforts most notably Teuvo Kohonen, Stephen Grossberg, James Anderson, and Kunihiko Fukushima.

Neural Networks – Historical Perspective

The interest in neural networks reemerged only after some important theoretical results were attained in the early eighties most notably the discovery of error backpropagation and new hardware developments. This renewed interest is reflected in the number of scientists, the amounts of funding, and the number of large conferences and journals associated with neural networks

Aftificial neural networks can be most adequately characterised as computational models with particular properties such as the ability to adapt or learn to generalise or to cluster or organise data and which operation is based on parallel processing. However many of these properties can be attributed to existing non-neural models.

Neural Networks – Historical Perspective

Often parallels with biological systems are described. However there is still so little known even at the lowest cell level about biological systems that the models we are using for our artificial neural systems seem to introduce an oversimplification of the biological models.

The point of view we take in thisintroduction is that of a computer scientist. We are not concerned with the psychological implication of the networks. We consider neural networks as an alternative computational scheme rather than anything else.

Neural Networks – A Framework for Distributed Representation

An artificial network consists of a pool of simple processing units which communicate by sending signals to each other over a large number of weighted connections

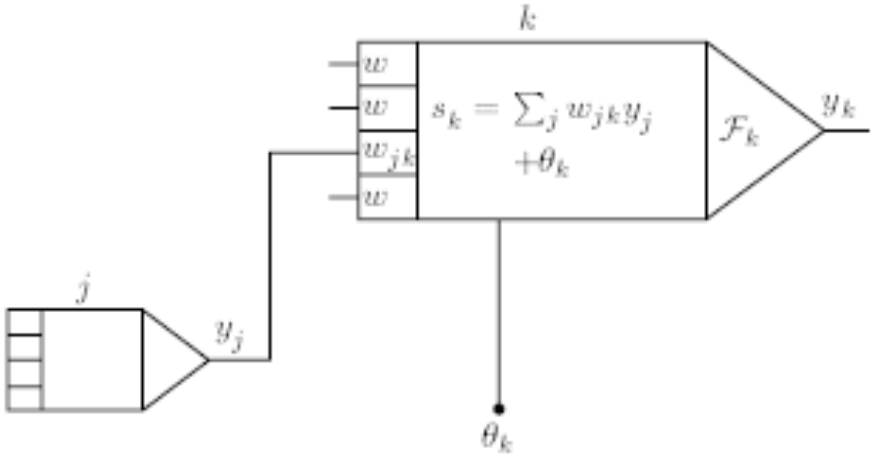
A set of major aspects of a parallel distributed model can be distinguished:

- a set of processing units {'neurons,' 'cells'};
- a state of activation y_k for every unit which equivalent to the output of the unit;
- connections between the units. Generally each connection is defined by a weight w_{jk} which determines the effect which the signal of unit j has on unit k;
- a propagation rule which determines the effective input s_k of a unit from its external inputs;

Neural Networks – A Framework for Distributed Representation

- an activation function F_k which determines the new level of activation based on the effective input $s_k(t)$ and the current activation $y_k(t)$, i.e., the update;
- an external input (aka bias, offset) Φ_k for each unit;
- a method for information gathering (the learning rule);
- an environment within which the system must operate, providing input signals and - if necessary - error signals

The figure illustrates these basics concepts:



Neural Networks – A Framework for Distributed Representation

Processing units

- Each unit performs a relatively simple job: receive input from neighbours or external sources and use this to compute an output signal which is propagated to other units. Apart from this processing a second task is the adjustment of the weights. The system is inherently parallel in the sense that many units can carry out their computations at the same time.
- Within neural systems it is useful to distinguish three types of units: input units (indicated by an index i) which receive data from outside the neural network, output units (indicated by an index o) which send data out of the neural network, and hidden units (indicated by an index h) whose input and output signals remain within the neural network

Neural Networks – A Framework for Distributed Representation

Processing units (cont)

- During operation, units can be updated either synchronously or asynchronously. With synchronous updating, all units update their activation simultaneously; with asynchronous updating each unit has a (usually fixed) probability of updating its activation at a time t and usually only one unit will be able to do this at a time. In some cases the latter model has some advantages

Connections between units

- In most cases we assume that each unit provides an additive contribution to the input of the unit with which it is connected. The total input to unit k is simply the weighted sum of the separate outputs from each of the connected units plus a bias or offset term Φ_k

Neural Networks – A Framework for Distributed Representation

Connections between units (cont)

$$s_k(t) = \sum_j w_{jk}(t) y_j(t) + \Phi_k(t) \quad \{1\}$$

The contribution for positive w_{jk} is considered an excitation and for negative w_{jk} an inhibition. At times more complex rules for combining inputs are used in which a distinction is made between excitatory and inhibitory inputs. We call units with a propagation rule {1} sigma units.

A different propagation rule introduced by Feldman and Ballard is known as the propagation rule for the sigma-pi unit:

$$s_k(t) = \sum_j w_{jk}(t) \prod_m y_{jm}(t) + \Phi_k(t) \quad \{2\}$$

Often the y_{jm} are weighted before multiplication. Although these units are not frequently used, they have their value for gating of input, as well as implementation of lookup tables.

Neural Networks – Activation and Output Rules

We also need a rule which gives the effect of the total input on the activation of the unit. We need a function F_k which takes the total input $s_k(t)$ and the current activation $y_k(t)$ and produces a new value f the activation of the unit k:

$$y_k(t+1) = F_k\,(y_k(t), s_k(t)) \qquad \{3\}$$

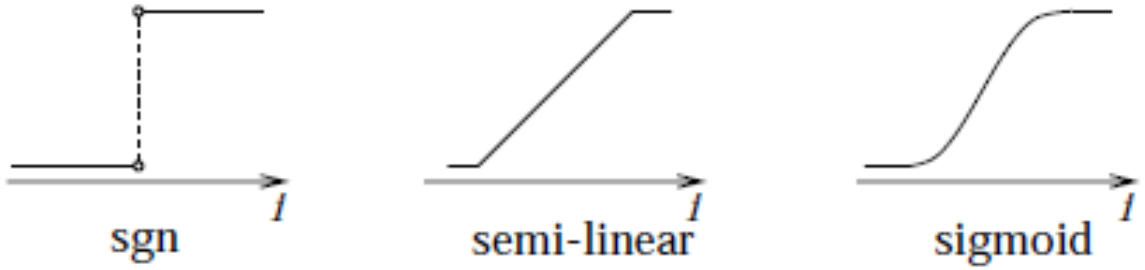
Often he activation function is a nondecreasing function of the total input of the unit:

$$y_k(t+1) = F_k\,(s_k(t)) = F_k\ [\sum_j w_{jk}(t)y_j(t) + \Phi_k(t)\,] \qquad \{4\}$$

although activation functions are not restricted to nondecreasing functions. Generally, some sort of threshold function is used: a hard limiting threshold function or a linear function or a smoothly limiting

Neural Networks - Activation and Output Rules (cont)

threshold (see figure below)



Various activation Functions for a unit

For this smoothly limiting function often a sigmoid (s-shaped) function like:

$$y_k = F(s_k) = 1 / (1 + e^{-sk}) \qquad \{5\}$$

is used. In some applications a hyperbolic tangentis used, yielding output values in the range {-1, +1}.

Neural Networks - Activation and Output Rules (cont)

In some cases the output of a unit can be a stochastic function of the total input of the unit. In that case the activation is not deterministically determined by the neuron input but the neuron input determines the probability p that a neuron get a high activation value:

$$p(y_k \leftarrow 1) = F(s_k) = 1 / (1 + e^{-Sk/T}) \qquad \{6\}$$

in which T (cf temperature) is a parameter which determines the slope of the probability function.

In all networks we describe we consider the output of a neuron to be identical to its activation level.

Neural Networks – Network Topologies

Connections between units and heir data propagation distinguishes:

- Feed-forward networks where the data flow from input to output units is strictly feed forward Data processing can extend over multiple layers of units but no feedback connections are present, that is, connections extending from outputs of units to inputs of units in the same layer or previous layers.
- Recurrent networks that do contain feedback connections. Contrary to feed-forward networks the dynamic network properties are important. In some cases the activation values of the units undergo a relaxation process such that the network will evolve to a stable state in which these activations do not change anymore. In other applications the change of the activation values of the output neurons are signicant such that the dynamic behaviour constitutes the output of the network.

Neural Networks – Training of ANN's

Classic examples of feed-forward networks are the Perceptron and Adaline which will be discussed below.

A neural network has to be configured such that the application of a set of inputs produces (either direct or via a relaxation process) the desired set of outputs. Various methods to set the strengths of the connections exist. One way is to set the weights explicitly using a priori knowledge. Another way is to train the neural network by feeding it teaching patterns and letting it change its weights according to some learning rule.

Neural Networks – Paradigms of Learning

We can categorise the learning situations in two distinct sorts:

Supervised learning or Associative learning in which the network is trained by providing it with input and matching output patterns. These input-output pairs can be provided by an external teacher or by the system which contains the network (self-supervised).

Unsupervised learning or Self-organisation in which an output unit is trained to respond to clusters of pattern within the input. In this paradigm the system is supposed to discover statistically salient features of the input population. Unlike the supervised learning paradigm there is no a priori set of categories into which the patterns are to be classified; rather the system must develop its own representation of the input stimuli.

Neural Networks – Modifying Patterns of Connectivity

Both learning paradigms described above result in an adjustment of the weights of the connections between units according to some modification rule. Virtually all learning rules for models of this type can be considered as a variant of the Hebbian learning rule suggested by Hebb in his classic book Organization of Behaviour in 1949. The basic idea is that if two units j and k are active simultaneously, their interconnection must be strengthened. If j receives input from k, the simplest version of Hebbian learning prescribes to modify the weight w_{jk} with

$$\Delta w_{jk} = \gamma y_j y_k \quad \{7\}$$

where γ is a positive constant of proportionality representing the learning rate. Another common rule uses not the actual activation of

Neural Networks – Modifying Patterns of Connectivity (cont)

$$\Delta w_{jk} = \gamma y_j (d_k - y_k) \qquad \{8\}$$

in which d_k is the desired activation provided by a teacher. This is often called the Widrow-Hoff rule or the delta rule.

Many variants (often very exotic ones) have been published over the last 20 years.

Neural Networks – Terminology

Output vs activation of a unit. We consider the output and the activation value of a unit to be one and the same thing, i.e., the output of each neuron equals its activation value.

Bias offset threshold. These terms all refer to a constant term (i.e., independent of the network input but adapted by the learning rule) which is input to a unit. They may be used interchangeably, although the latter two terms are often seen as a property of the activation function. This external input is usually implemented (and can be written) as a weight from a unit with activation value 1.

Number of layers. In a feed-forward network the inputs perform no computation and their layer is therefore not counted. Thus a network with one input layer, one hidden layer, and one output layer is referred to as a network with two layers This convention is widely though not universally used

Neural Networks – Terminology (cont)

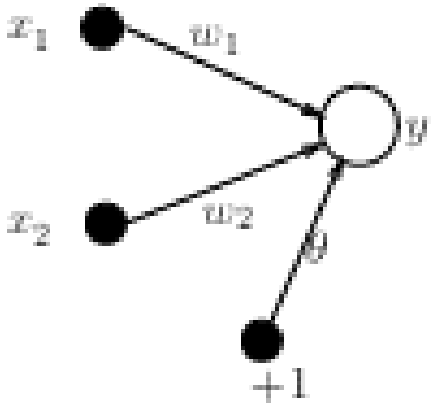
Representation vs learning. When using a neural network one has to distinguish two issues which influence the performance of the system The first issue is the representational power of the network, the second issue is the learning algorithm.

The representational power of a neural network refers to the ability of a neural network to represent a desired function. Because a neural network is built from a set of standard functions, in most cases the network will only approximate the desired function, and even for an optimal set of weights the approximation error is not zero

The second issue is the learning algorithm. Given that there exist a set of optimal weights in the network, is there a procedure to iteratively find this set of weights?

The Perceptron – networks with threshold activation functions

A single layer feed-forward network consists of one or more output neurons o, each of which is connected with a weighting factor w_{io} to all of the inputs I. In the simplest case the network has only two inputs and a single output as sketched in the figure below (we leave the output index o out. The input of the neuron is the weighted sum of the inputs plus the bias term. The output of the network is formed by the activation of the output neuron which is some function of the input:



single layer network with one input & two outputs

The image cannot be displayed. Your computer may not have enough memory to open the image, or the image may have been corrupted. Restart your computer, and then open the file again. If the red x still appears, you may have to delete the image and then insert it again.

The Perceptron – networks with threshold activation functions (cont)

The activation function F can be linear so that we have a linear network, or nonlinear. We consider the threshold function

$$F(s) = \{1 \text{ if } s>0 \mid -1 \text{ otherwise}\}$$

The output of the network thus is either +1 or -1, depending on the input. The network can now be used for a classification task: it can decide whether an input pattern belongs to one of two classes. If the total input is positive, the pattern will be assigned to class +1, if the total input is negative the sample will be assigned to class -1. The separation between the two classes in this case is a straight line, given by the equation

$$w_1x_1 + w_2x_2 + \theta = 0 \qquad \{9\}$$

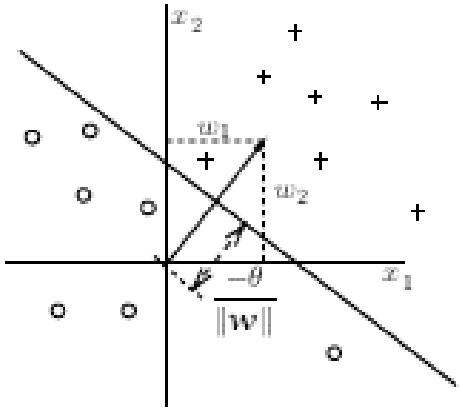
The single layer network represents a linear discriminant function.

The image cannot be displayed. Your computer may not have enough memory to open the image, or the image may have been corrupted. Restart your computer, and then open the file again. If the red x still appears, you may have to delete the image and then insert it again.

The Perceptron – networks with threshold activation functions (cont)

A geometrical representation of the linear threshold neural network is given in the figure below. Equation {9} can be written as

$$x_2 = -(w_1/w_2)x_1 - (\theta/w_2) \qquad \{10\}$$



Geometric representation
Of the discriminant function
And the weights

and we see that the weights determine the slope of the line and the bias determines the offset, i.e., how far the line is from the origin. Note also that the weights can be plotted in the input space: the weight vector is always perpendicular to the discriminant function.

The image cannot be displayed. Your computer may not have enough memory to open the image, or the image may have been corrupted. Restart your computer, and then open the file again. If the red x still appears, you may have to delete the image and then insert it again.

The Perceptron – networks with threshold activation functions (cont)

Now that we have shown the representational power of the single layer network with linear threshold units we come to the second issue: how do we learn the weights and biases in the network? We describe two learning methods for these types of networks: the perceptron learning rule and the delta or LMS rule. Both methods are iterative procedures that adjust the weights. A learning sample is presented to the network. For each weight the new value is computed by adding a correction to the old value. The threshold is updated in a same way:

$$w_i(t+1) = w_i(t) + \Delta w_i(t),$$

$$\theta(t+1) = \theta(t) + \Delta\theta(t)$$

The learning problem can now be formulated as how do we compute $\Delta w_i(t)$ and $\Delta\theta(t)$ in order to classify the learning patterns correctly?

The image cannot be displayed. Your computer may not have enough memory to open the image, or the image may have been corrupted. Restart your computer, and then open the file again. If the red x still appears, you may have to delete the image and then insert it again.

The Perceptron – perceptron learning rule and convergence theorem

Suppose we have a set of learning samples consisting of an input vector x and a desired output d(x). For a classication task the d(x) is usually +1 or -1.The perceptron learning rule is very simple and can be stated as follows:

1. start with random weights for the connections;
2. select an input vector x from the set of training samples;
3. If $y \neq d(x)$ (the perceptron gives an incorrect response), modify all connections w_i according to $\Delta w_i = d(x)x_i$;
4. Go back to 2.

Note that the procedure is very similar to the Hebb rule; the only difference is that, when the network responds correctly, no connection weights are modified. Besides modifying the weights we must also modify the threshold θ.

The image cannot be displayed. Your computer may not have enough memory to open the image, or the image may have been corrupted. Restart your computer, and then open the file again. If the red x still appears, you may have to delete the image and then insert it again.

The Perceptron – perceptron learning rule and convergence theorem

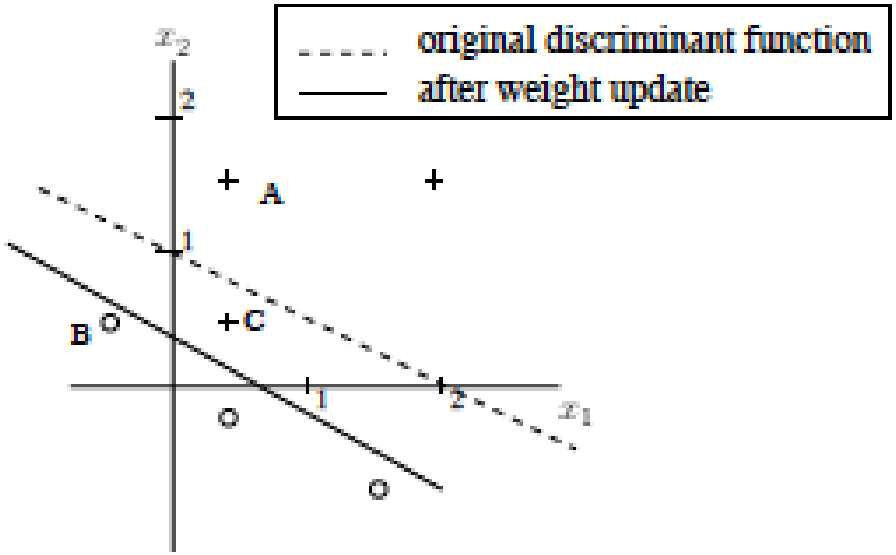
This θ is considered as a connection w_0 between the output neuron and a 'dummy' predicate unit which is always on: $x_0 = 1$. Given the perceptron learning rule as stated above this threshold is modied according to:

$\Delta\theta$ = {0 if the perceptron responds correctly; | d(x) otherwise }

The image cannot be displayed. Your computer may not have enough memory to open the image, or the image may have been corrupted. Restart your computer, and then open the file again. If the red x still appears, you may have to delete the image and then insert it again.

The Perceptron – Example of the Perceptron learning rule

A perceptron is initialized with the following weights $w_1 = 1$, $w_2 = 2$, $\theta = -2$. The perceptron learning rule is used to learn a correct discriminant function for a number of samples sketched in the figure below. The first sample A, with values x = {0.5, 1.5} and target value d(x)=+1 is presented to the network.



The image cannot be displayed. Your computer may not have enough memory to open the image, or the image may have been corrupted. Restart your computer, and then open the file again. If the red x still appears, you may have to delete the image and then insert it again.

The Perceptron – Example of the Perceptron learning rule

From this equation $$y = \mathcal{F}\left(\sum_{i=1}^{2} w_i x_i + \theta\right)$$

it can be calculated that the network output is +1, so no weights are adjusted. The same is the case for point B, with values x = {-0.5, 0.5} and target value d(x) = -1; the network output is negative so no change. When presenting point C with values x = {0.5, 0.5} the network output will be -1, while the target value d(x) = +1. According to the perceptron learning rule the weight changes are: $\Delta w_1 = 0.5$, $\Delta w_2 = 0.5$, $\Delta\theta = 1$. The new weights are now: $w_1 = 1.5$, $w_2 = 2.5$, $\Delta\theta = -1$, and sample C is classified correctly.

In the previous figure the discriminant function before and after this weight update is shown

The image cannot be displayed. Your computer may not have enough memory to open the image, or the image may have been corrupted. Restart your computer, and then open the file again. If the red x still appears, you may have to delete the image and then insert it again.

The Perceptron – Convergence theorem

For the perceptron learning rule there exists a convergence theorem which states the following

Theorem 1: If there exists a set of connection weights w* which is able to perform the transformation y = d(x), the perceptron learning rule will converge to some solution (which may or may not be the same as w*) in a finite number of steps for any initial choice of the weights.

Proof: Given the fact that the length of the vector w* does not play a role (because of the sgn operation), we take ||w*||=1. Because w* is a correct solution, the value |w* - x|, where • denotes dot or inner product, will be greater than 0 or: there exists a $\delta>0$ such that $|w^*-x|>\delta$ for all inputs x. Now define $\cos \alpha \equiv w \bullet w^* / ||w||$. When according to the perceptron learning rule connection weights are modified at a given input x, we know that $\Delta w = d(x) / x$ and the weight after modication is $w' = w + \Delta w$. From this it follows that:

•

The Perceptron – Convergence theorem (cont)

$$w' \bullet w^* = w \bullet w^* + d(x) \bullet w^* \bullet x$$
$$= w \bullet w^* + \mathrm{sgn}(w^* \bullet x)\, w^* \bullet x$$
$$> w \bullet w^* + \delta$$

$$\|w'\|^2 = \|w + d(x)\, x\|^2$$
$$= w^2 + 2d(x) w \bullet x \bullet x^2$$
$$= w^2 + x^2 \quad \text{(because } d(x) = -\mathrm{sgn}[w \bullet x] \text{ !!)}$$
$$= w^2 + M$$

After t modications we have

$$\|w'\|^2 = w^2 + tM$$

The image cannot be displayed. Your computer may not have enough memory to open the image, or the image may have been corrupted. Restart your computer, and then open the file again. If the red x still appears, you may have to delete the image and then insert it again.

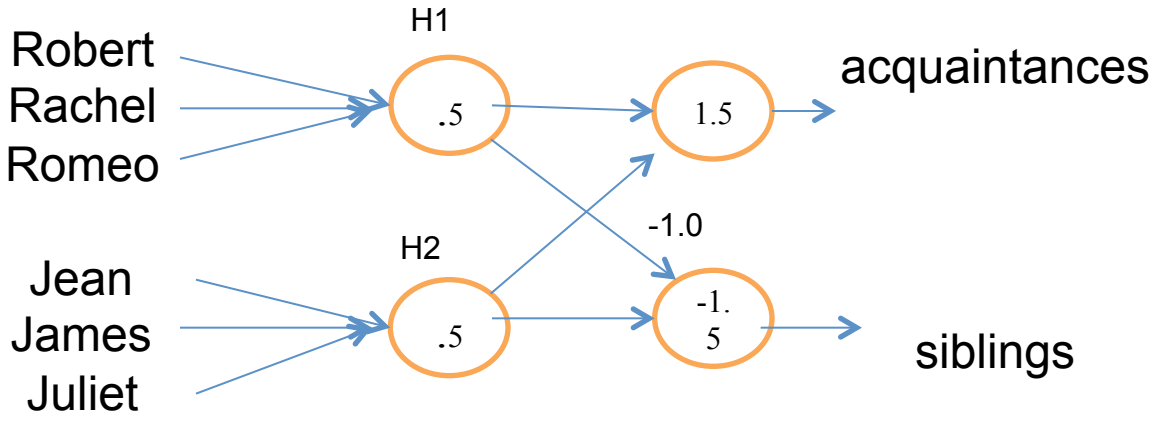
The Perceptron – Convergence theorem (cont)

The conclusion is that there must be an upper limit tmax for t The system modies its connections only a limited number of times In other words after maximally tmax modications of the weights the perceptron is correctly performing the mapping tmax will be reached when cos α=1. If we start with connections w =0,

$$t_{max} = \frac{M}{\delta^2}.$$

Feed-Forward Nets can Recognise Regularity in Data

The net below is equipped with werights thast enable it to recognise properties of pairs of people. Some of the pairs involve siblings nd others involve acquaintances. The input connjections receive a value of 1 to identify the pairs of people under consideration. All other input connections receive values of 0 because the corresponding people



are not part of the pair under consideration. Assume that the people in the top group of three are siblings, as are the people in the bottom group.

Feed-Forward Nets can Recognise Regularity in Data (cont)

Further assume that any pair of people who are not siblings are acquaintances.

The net is not fully connected to simplify discussion. Nodes to the right of the input links are hidden nodes because their outputs are not fully observable. Output nodes convey conclusions.

Any of the first three inputs produces enough stimulation to fire H1 because all connecting weights are 1.0 and because H1's threshold id 0.5. Similarly any of the second three produces enough to fire H2. Thus H1 and H2 act as logical OR gates. At least one of H1 and H2 has to fire because two inputs are always presumed to be on.

If both H1 and H2 fire then the weighted sum presented to the aquaintances node is 2 because both of the weights involved are 1.0

Feed-Forward Nets can Recognise Regularity in Data (cont)

and because the acquaintance nodes threshold is 1.5, If only one of H1 or h2 fire, then the acquaintance node does not fire. Thus the acquaintance node acts as a logical AND gate: it fires only when the input pair are acquaintances.

What happens if both H1 and H2 fire?

Note that in this example each link and node has a clear role. Generally this is not the case since recognition capability id distributed diffusely over many more nodes and weights and the role of particular links and hidden nodes becoes obscure.

Concluding Remarks

Out of Time

(A holiday thought)

My old clock used to tell the time

and subdivide diurnity;

but now it's lost both hands and chime

and only tells eternity.