9. Applications to Computer Vision

So far we have looked at the theory of neural networks and introduced some well known examples of their use. The aim of this section is to examine the direct application of neural networks within the field of computer vision. We do this by examining three examples of the ways in which they are of interest. These are:

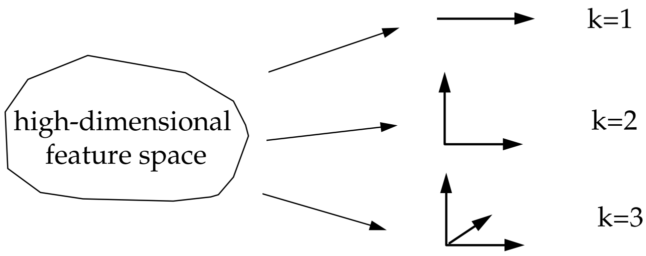
- as a source of useful algorithms;
- as a 'construction kit' for machine learning;
- as a methodology that leads to parallel algorithms

This list is not intended to be exhaustive and really ought to include the importance of neural networks in modelling biological vision systems.

Neural networks as a source of useful algorithms

There is some controversy about the proper contribution of neural networks beyond other more established disciplines such as statistics (see for example Ripley, 1993). However, there is little doubt that the field has spawned new algorithms which may be of general use in engineering computer vision systems. Perhaps the most common application has been as an alternative to classical pattern recognition solutions to cluster formation and classification within feature spaces. The standard feedforward network is well suited to this task.

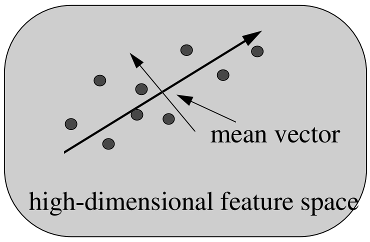
For these notes, we've chosen to look instead at the use of Kohonen's Self-Organising Feature Map (SOFM) in seeking useful projections of point clusters from high-dimensional spaces (R^p) into low-dimensional spaces $R^k, k = 1,2,3$.



High-dimensional feature spaces arise in many ways in vision. For example:

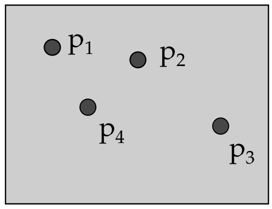
- the neighbourhood about an image pixel can be characterised by the vector of outputs from banks of spatial filters centred on the pixel (e.g. Gabor filters in a range of sizes and orientations). Clustering such vectors is the basis of many segmentation algorithms.
- a simple shape model composed of selected landmark points on an object can be represented by concatenating the spatial coordinates of landmarks into a single (*shape*) vector. This idea may be generalised to include any number of numerical parameters of a geometric model.

It is often necessary to derive mappings from collections of points in such spaces onto lower dimensional spaces so that, for example, the distribution of points may be visualised as a 2-D picture. Various mathematical and algorithmic tools exist for deriving such projections. Perhaps the most widely known is principal component analysis in which the first k components provide the required projection.



The classical solution in the theory of multi-dimensional scaling is a closely related method in which only the distance between all pairs of feature vectors (a *distance* matrix) is required to derive a configuration of points in k-dimensional Euclidean space that optimally preserves these distances (see for example, Mardia et al., 1979). An illustrative distance matrix and computed result are shown below:

$$\begin{array}{c c} & \begin{array}{cccc} p_1 & p_2 & p_3 & p_4 \end{array} \\ \begin{array}{c} p_1 \\ p_2 \\ p_3 \\ p_4 \end{array} & \begin{bmatrix} 0 & 7 & 12 & 5 \\ 7 & 0 & 6 & 4 \\ 12 & 6 & 0 & 8 \\ 5 & 4 & 8 & 0 \end{bmatrix} \end{array} \longrightarrow$$



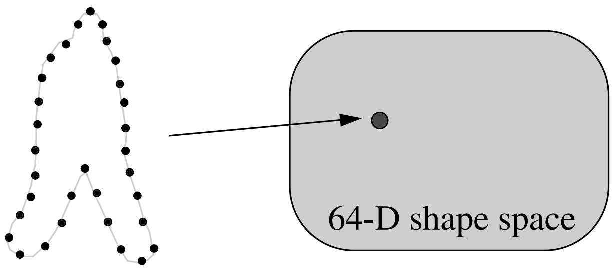
The cost function minimised by this solution method is:

$$\sum_{i,j=1}^{n} \left\| p_i - p_j \right\|^2 - \left\| \hat{p}_i - \hat{p}_j \right\|^2$$

where $\hat{p}_i, \hat{p}_j$ are the projected points.

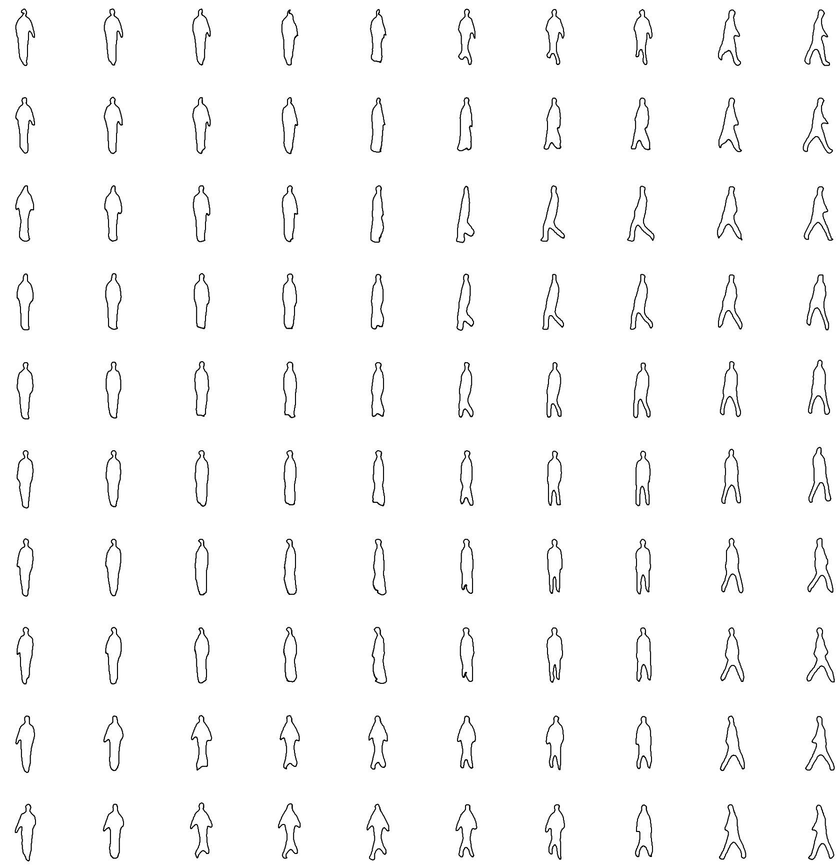
A projection is obtained using Kohonen's SOFM by placing a k-dimensional grid of units into the feature space and adapting to the distribution of data points. The required projection is defined by mapping a given point onto the nearest unit or onto a point interpolated between units.

To illustrate this idea, we use a training set of shape vectors representing the outlines of pedestrians, each composed of 32 boundary points. A principal component analysis applied to this data set captures 99% of the variance with only 9 components (see Baumberg and Hogg, 1984).

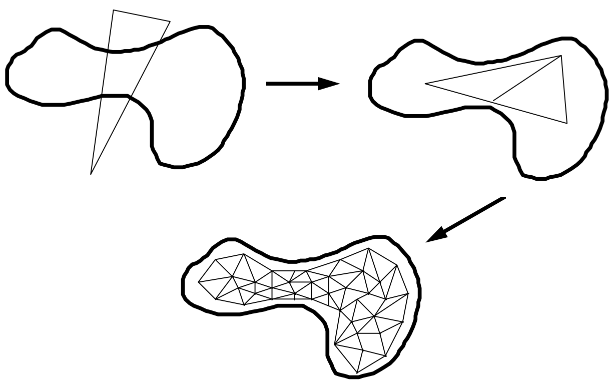


Applying a 10x10 SOFM[1] to the same data set positions each unit at the shape vectors depicted below.

[1] The research from which this originates has been carried out by Neil Johnson at Leeds (neilj@scs.leeds.ac.uk)



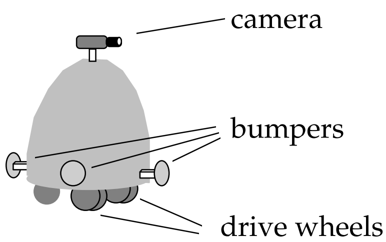
An interesting recent development by Fritzke (1994) attempts to grow SOFMs at the same time as adapting to the data set. For a given value of k, the network is initialised to a k-dimensional simplex (see below). New units added as the network extends across to the data set to ‘plug’ gaps appearing between units. The result is generally a non-regular network which uniformly covers the dataset.



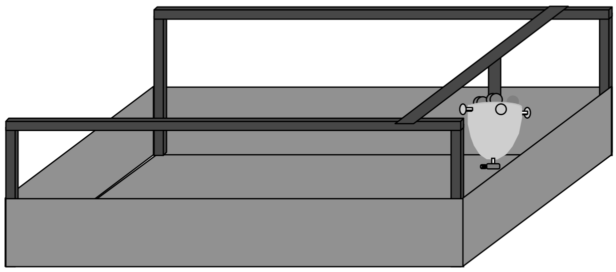
Neural networks as a construction kit for machine learning

Neural networks define computations in terms of primitive elements (e.g. units and connections) combined in simple ways. Since even simple networks can do useful things, they provide a powerful construction kit for studies of machine learning. In particular, there have been several attempts to evolve neural networks using evolutionary learning strategies, and in particular using genetic algorithms. We examine recent work in this direction by Harvey, Inman and Cliff (1994).

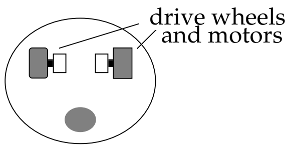
The aim of this work is to evolve a robot to move towards pre-defined targets using touch sensors and a simple vision system. Both the control system of the robot (a dynamic recurrent neural network) and the visual morphology are evolved in parallel.



The first attempt at doing this worked entirely in simulation. A major limitation of this simulation was in providing realistic visual input without excessive computation. To avoid this problem and to provide a more realistic test-bed, the current work uses a physical robot equipped with touch sensors and TV camera mounted on an overhead x-y traction system to simulate the drive wheels.



These simulated drive wheels are controlled by a real-valued input in the range [-1 1] as follows:

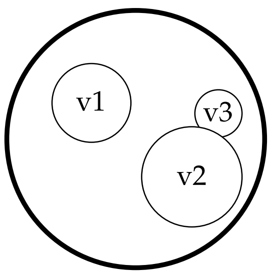


full reverse	half reverse	stop	half forward	full forward

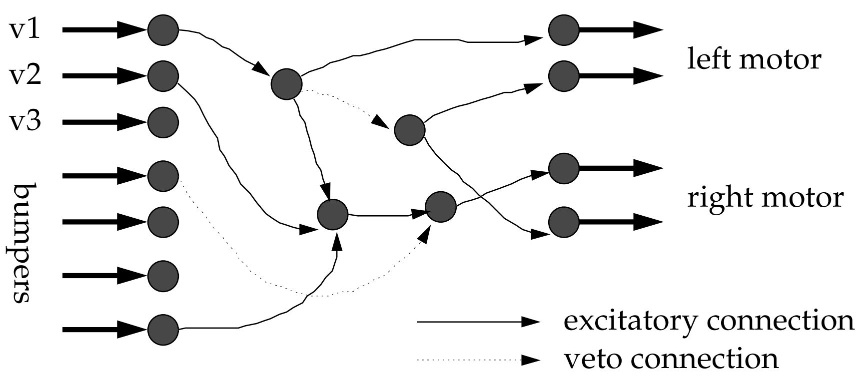
-1 1

The robot has four touch sensors mounted around its base. These output 1 if in contact with an obstacle and 0 otherwise.

A CCD TV camera mounted on the robot provides visual input as a standard rectangular array which is pre-processed to compute the mean light intensity over three receptive fields. The output from the visual sub-system is just these three values scaled to the range [0 1] to 4-bit accuracy. The position and radius of these fields (the *visual morphology*) is allowed to vary during evolution.

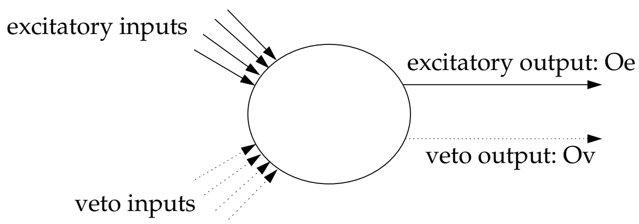


The robot controller is a simple kind of dynamic recurrent neural network receiving input from the four touch sensors and three visual fields and providing output to the (simulated) drive wheels via a number of hidden units. The figure below shows the fixed set of inputs and outputs together with a possible collection of hidden units and connections for illustrative purposes. The aim is to evolve the network topology (including the number of units) through genetic learning.



The signal travelling along all connections is a real value in the range [0 1]. A control output to the drive wheels in the range [-1 1] is computed by subtracting a pair of outputs associated with each wheel.

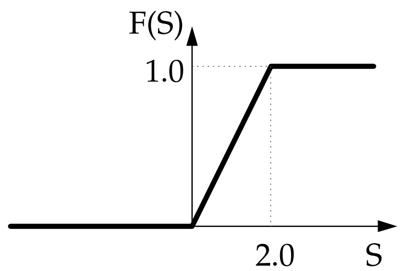
Each unit has zero or more excitatory inputs and veto inputs. It also has an excitatory output and a veto output which can connect to any number of other units. When all veto inputs are 0, the excitatory output is a linear threshold function of the summed excitatory inputs. If any of the veto inputs are above 0, the excitatory output from the unit is 0. The veto output is 1 if the summed excitatory input exceeds a threshold and 0 otherwise. Finally, each connection (excitatory and veto) introduces a time delay to the signal propagation.



$$\text{Let } S = (\textstyle\sum \text{excitatory inputs}) + \text{noise in range } [-0.1 \quad 0.1]$$

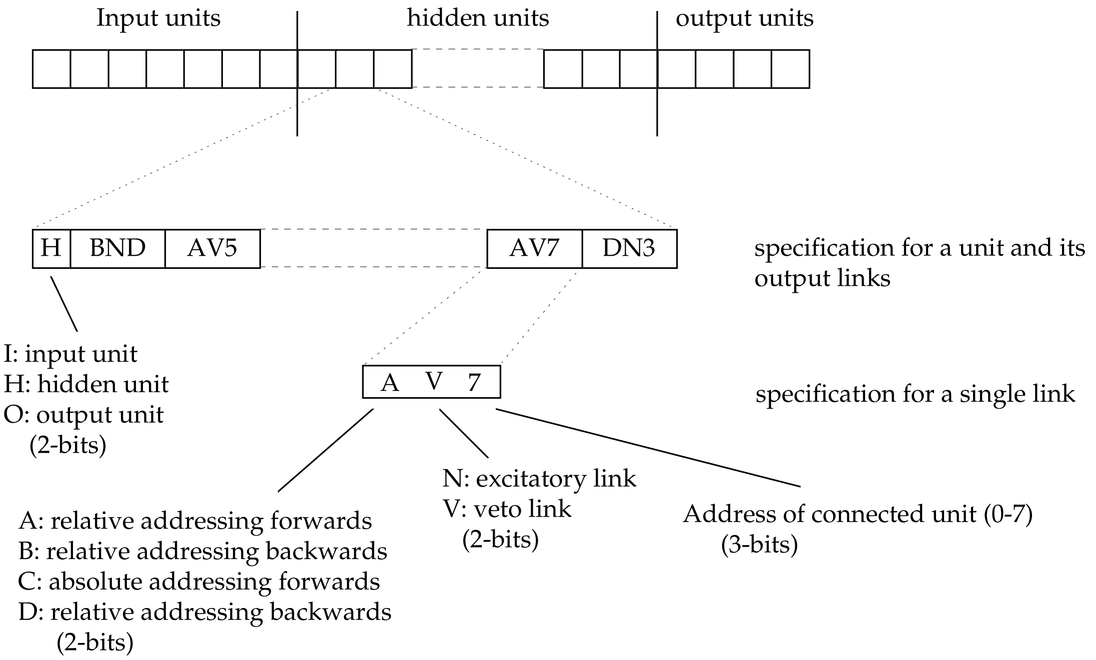
$$O_e = \begin{cases} 0 & \text{any veto input} > 0 \\ F(S) & \text{otherwise} \end{cases}$$

$$O_v = \begin{cases} 0 & S < 0.75 \\ 1 & S \geq 0.75 \end{cases}$$

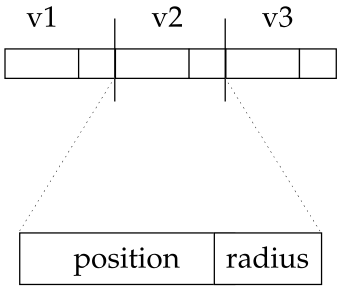


Genetic encoding

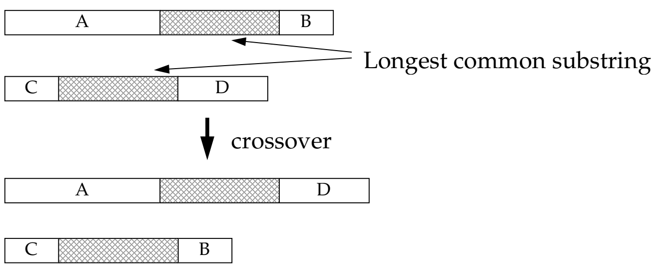
To operate a genetic algorithm, individual recurrent networks are encoded as single ‘chromosomes’ (bit-strings). Each unit, together with its connections, is encoded as a separate sub-string. The concatenation of these sub-strings is the complete specification for a network. Strings will typically be of different lengths due to variations in the number of units and the topology of their connections. The encoding for each unit identifies its type (input, hidden, or output) and specifies each of its output connections. Full details are given in the figure below.



Visual morphology is encoded in a separate chromosome. This string is divided into three parts, one for each receptive field, encoding its position and radius within the camera image.



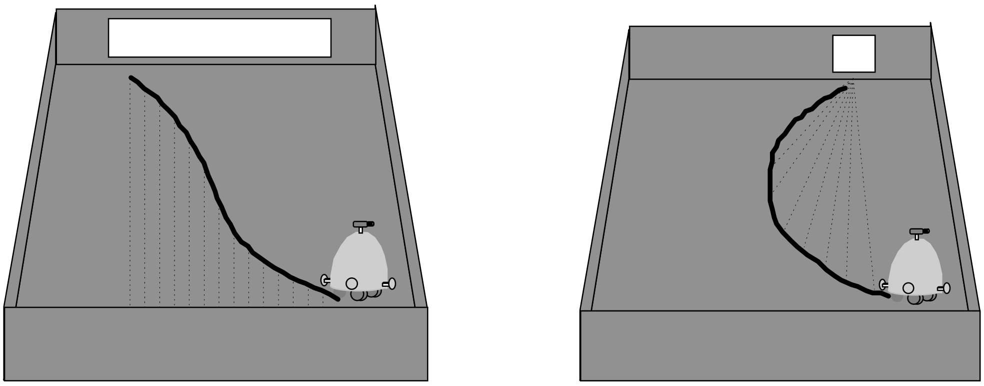
The standard genetic algorithm crossover operator requires strings of the same length. In the current situation, strings may be of different lengths corresponding to networks of different complexity. Unfortunately, simple adaptations of the standard operator give poor evolutionary performance. However, it turns out that a crossover operator that preserves the longest common substring (common genetic material) works well.



Experiments

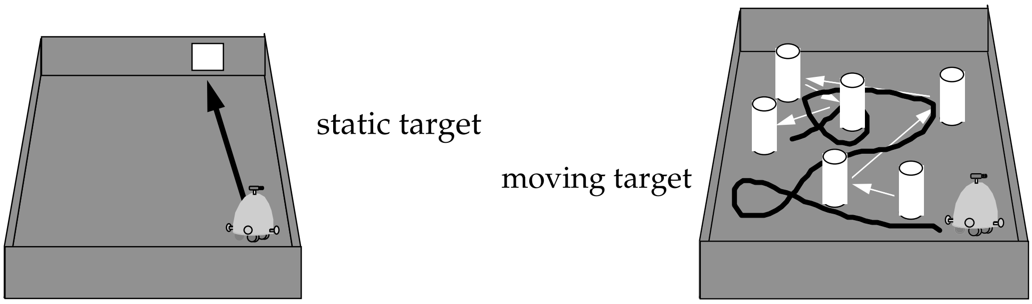
In the first experiment, the robots were placed in a rectangular arena and required to move towards a large white target covering most of one of the walls. Each robot was evaluated by running a trial in which it was placed in the arena for 25 seconds. There were 30 robots in successive generations.

The initial population was created by cloning an individual with interesting behaviour (i.e. it exhibits some 'exploratory' activity) chosen from a selection of randomly produced robots. The evaluation function was the accumulated distance of the robot from the wall opposite the white target, sampled at regular time intervals during the trial (see left figure below).

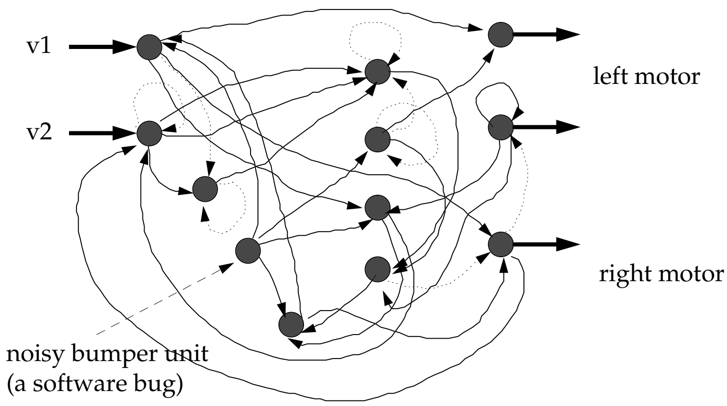


After a fixed number of generations the fittest individual successfully navigated rapidly to the target.

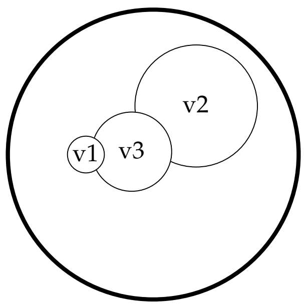
In a second experiment, the size of the target was reduced and the evaluation function modified to record the accumulated distance to the target (see right figure above). The population was initialised to the final generation of the first experiment. The best robot after 15 generations moved directly to the target. Moreover, when the target was moved around the arena, the same robot followed it around thereby exhibiting a generalisation of the original task.



The active part of the recurrent neural network in this robot is shown below. It is interesting to observe that only two receptive fields are used in this network.



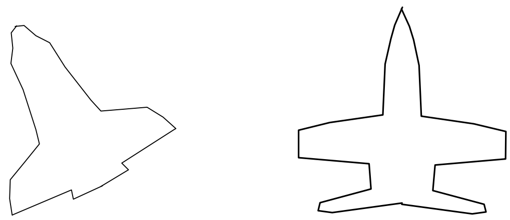
The visual morphology of this same robot is shown below.



Neural networks as a methodology that leads to parallel algorithms

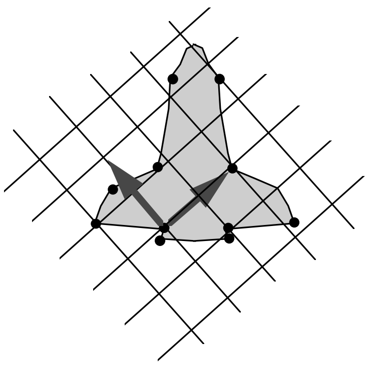
It is often stated that a major advantage of neural networks is their inherent parallelism. Although most networks are currently simulated on conventional serial machines, the potential exists for direct implementation on massively parallel hardware. In this section, we examine an ANN design for object recognition that exploits the same kind of geometric invariance that has become important in computer vision within the last few years. However, the ANN context for this work leads to an inherently parallel algorithm.

The aim is to recognise one or more objects in any position, size or orientation within the image plane. In other words, it is to recognise an object in any 2-D pose - only rotation about an axis orthogonal to the image plane is allowed. It is assumed grey-level images are suitably pre-processed to produce binary images highlighting the intensity edges. Target objects might therefore appear as in the diagram below:



Geometric hashing

Before explaining the details of the ANN we review the *geometric hashing* procedure (Wolfson, 1992) since this is closely related. In geometric hashing, object models are pre-compiled into a hash table in an off-line process. For the equivalent task to that stated above, all pairs of landmark points from the object are selected in turn. For each pair, a Cartesian coordinate frame is constructed such that one of the basis vectors extends between the two points and the other is orthogonal to this and of the same length.

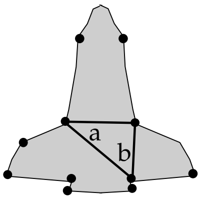


Coordinates	Model/Frame
3, 4	shuttle, {P1, P2}
-2, 7	shuttle, {P3, P4}
5, 9	F-18, {P5, P6}

Now the coordinates in this frame of all other landmark points are computed and recorded in a hash table as the key to the name of the model and the identity of the ‘basis’ points.

To recognise an object, preprocess the given image to detect landmark points. Now select a pair of landmarks that appear likely to belong to the same object (e.g. they lie on the same boundary) and construct a coordinate frame as before. Chose several other landmarks points and compute their coordinates with respect to this frame. Using these coordinates as keys into the hash table, choose the most frequently occurring object name.

The critical feature of this procedure is that the coordinate pairs generated from sets of three points (2 basis points + 1) are invariant to similarity transformations (translation, uniform scaling, rotation) and will therefore be the same irrespective of the object’s 2-D pose. Other geometric invariants would do equally well. For example, the interior angles of a triangle defined by three points, are invariant to similarity transformations.



Using a Higher Order Neural Network

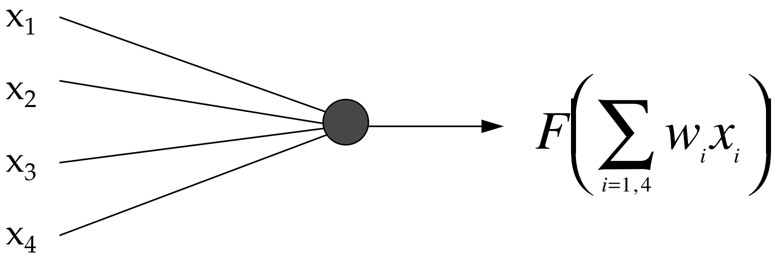
In a Higher Order Neural Network (HONN), individual units compute the following from their inputs (see for example, Giles and Maxwell, 1987):

$$y_i = F(T_0(i) + T_1(i) + T_2(i) + \ldots + T_k(i))$$

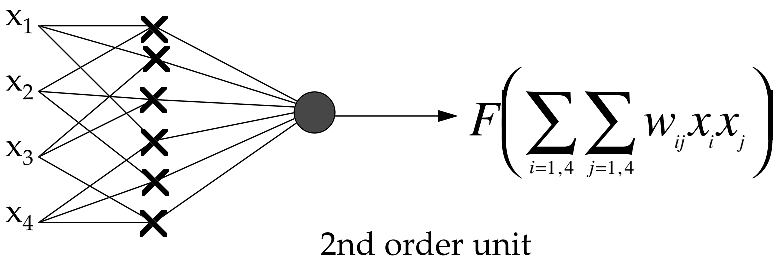
where $T_n(i)$ is the weighted sum of

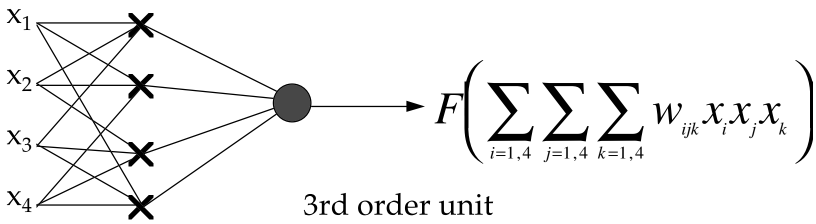
nth order products of the inputs

For example, a regular neural network consists of 1st order units containing only a 1st order term:

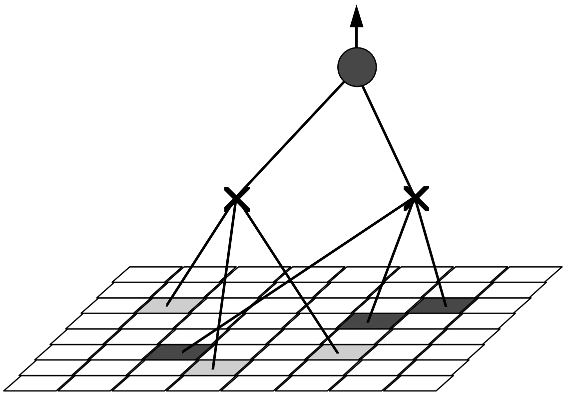


In general, higher order units contain terms of different orders. Examples of 2nd and 3rd order units containing only a 2nd and 3rd order term respectively can be represented as follows:



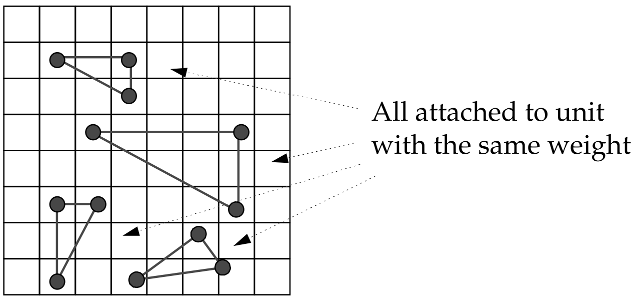


A HONN for recognising objects in binary contour images, is constructed as follows (Spirkovska and Reid, 1993*).* The HONN has a single layer of 3rd order units, one for each object to be recognised. Each unit receives input from every possible triple of pixels in the binary image to give a sum of products. Two such triple products are illustrated in the diagram below.

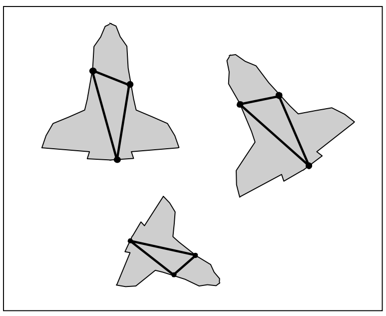


The network is trained by altering the input weights to approximate the required input-output behaviour across a training set of target objects. This amounts to non-linear optimisation.

To have any chance of recognising objects irrespective of pose, it may appear that the training set would have to include examples spanning the range of possible poses. For example, it would seem unreasonable to expect the net to generalise from training objects seen at one position in the image to recognise those objects at a different position. However, there is a way around this apparent limitation. Each weighted input product is derived from three points in the image. These three points form a triangle and must each be on an intensity edge for there to be a net contribution from the input. The key is to constrain those weights associated with similar triangles to be the same.



Since an object in any position and orientation contains approximately the same set of triangles, the summed output from each unit will be the same. As a consequence, a network trained to recognise objects in a single pose will automatically generalise to all other positions and orientations.



This property does not hold for scaled version of the object since enlarging an object will introduce additional boundary points with a consequential increase in the number of triangles. However, it turns out that the network will accommodate small variations in scale without marked deterioration in performance.

Unfortunately, this approach is infeasible for anything but very small images because of the rapid growth in the number of point triples as the image size increases. One way around this problem is to use coarse coding of inputs (Spirkovska and Reid, 1993).