An Angular Quantising Self Organising Map for Scale Invariant Classification

Stephen McGlinchey and Colin Fyfe,
Department of Computing and Information Systems,
University of Paisley.
email: mcgl0ci@student.paisley.ac.uk, fyfe0ci@paisley.ac.uk,
Tel: +44 141 848 3328, +44 141 848 3305.

Abstract

We use a simple network that uses negative feedback of activation and simple Hebbian learning to self organise in such a way as to produce a feature map which has the property of identifying the relative proportions of the components of the input data. Thus, it evaluates the angular properties of the input data space and ignores the magnitude of the input data. We have previously shown[1] that we can easily find a one dimensional mapping of input data; we now present an improved algorithm which finds a two dimensional mapping of the data. When we train this network on uniformly distributed artificial data, we can observe convergence to the shape of a sphere which exhibits the scale invariant classifying properties previously seen in the network.

1 Introduction

We have introduced an artificial neural network that self-organises to find a mapping of the input data that preserves neighbourhood relations. The difference between this mapping and Kohonen's original self-organising map [2] is that the mapping is scale invariant. When the Kohonen map is trained it approximates a Voronoi tessellation of the input space. The scale invariant map, however, creates a mapping where each neuron captures a "pie slice" of the data according to the angular distribution of the input data (see Figure 1). The network has been successfully applied to the processing of raw voice data[3] to extract phase and frequency information while ignoring the amplitude of that data.

When the network is trained on uniformly distributed artificial data, we expect each neuron to capture an equal size of "pie slice". The method worked well when a network with a one dimensional neighbourhood function was trained on higher dimensional data, however, when we used a two-dimensional neighbourhood function on three-dimensional (or higher dimensional) data, we found:

- A map based on a square or rectangle does not form easily into a sphere and so the neighbourhood relations force certain areas of the map to differ from others in the size of the angle that they capture.
- During training, twists frequently appeared and were difficult to remove. This caused some weight vectors to lie on approximately the same straight line through the origin and other neuron(s), thus losing the scale invariant properties of the map.

We now present an improved algorithm which overcomes these problems

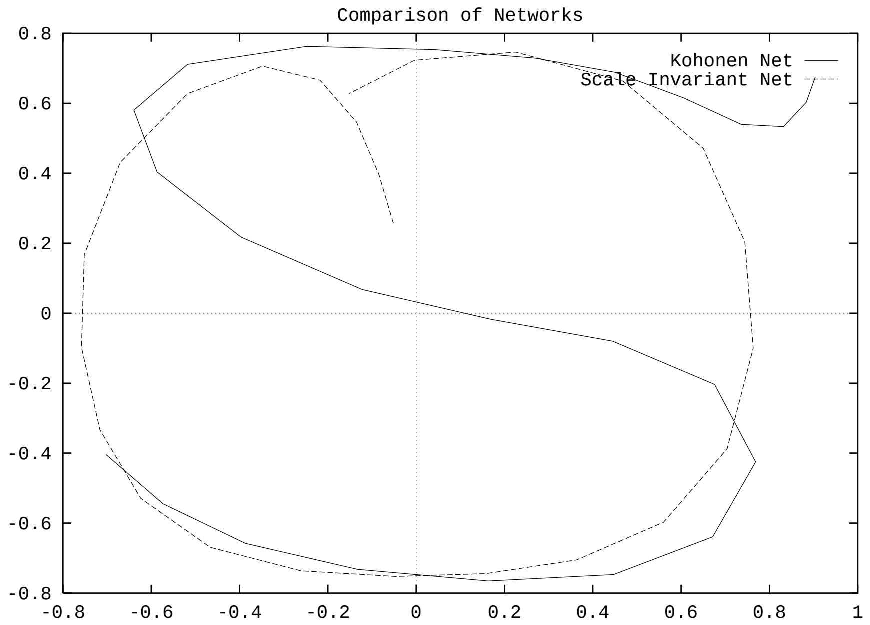

Figure 1: A one dimensional scale invariant feature map and a Kohonen map on the data drawn from a uniform distribution.

2 The Algorithm

Consider a network with n dimensional input data and having m output neurons and a $m \times n$ weight vector that is initialised to small random values. A competition takes place between all outputs. We have previously used two types of competition criterion:

Type A: The activation of each neuron is a weighted sum of its inputs as shown in equation (1) and the neuron with the highest activation wins.

$$\max_i(act_i) = \max_i(\sum_{j=1}^{N} w_{ij}x_j) \tag{1}$$

Type B: The neuron whose weight vector is closest to the input vector wins(see equation (2)). If $\mathbf{x}$ is the input vector and $\mathbf{w}_i$ is the weight vector for output neuron i, then the competition is given by:

$$\min_i(f_i) = \min_i(\parallel \mathbf{x} - \mathbf{w}_i \parallel) \tag{2}$$

We investigate the effects of different types of competition in section 4.

When a winner, e.g the p^{th} is selected, it is deemed to be maximally firing (=1) and all other output neurons are suppressed (=0). Its firing is then fed back through the same weights to the inputs as inhibition as shown in equation (3)

$$x_j(t+1) \leftarrow x_j(t) - w_{pj}, \forall j \tag{3}$$

Now the winning neuron excites those neurons close to it, i.e.we have a neighbourhood function $\Lambda_(p,j)$ which satisfies $\Lambda(p,j) \leq \Lambda(p,k)$ for all $j,k :\parallel p-j \parallel \geq \parallel p-k \parallel$ where $\parallel . \parallel$ is the Euclidean

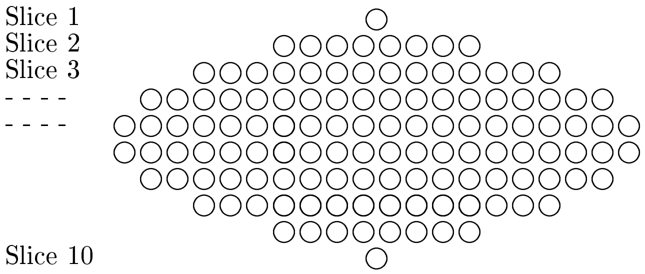

Figure 2: The shape of the map

norm. For this experiment the neighbourhood function is a gaussian whose variance is decreased during the course of training. Then simple Hebbian learning gives:

$$\begin{aligned} \delta w_{ij} &= \eta_t \Lambda(p,i).x_j(t+1) \\ &= \eta_t \Lambda(p,i).(x_j(t) - w_{pj}) \end{aligned}$$

where we have used $x_j(t)$ as the activation of the j^{th} input neuron at time t and w_{ij} is the weight between this and the i^{th} output neuron. Note that every weight vector is updated in the direction of the vector $\mathbf{x} - \mathbf{w}_p$ with a magnitude dependant on the neighbourhood function.

3 Spherical Mapping of Artificial Data

In order to achieve scale invariant classification, we need to produce a mapping that will converge such that all nerons have an equal probability of winning a competition when an input vector chosen at random from the training data. When uniformly distributed artificial data is used, this will have the form of a circle, sphere or hyper-sphere, depending on the dimensionality of the output space. We have already shown that a one-dimensional map can converge to an approximation of a circle when trained on uniformly distributed two-dimensional data. In this paper we train a two-dimensional map on three-dimensional data to converge to the shape of a sphere. The commonly used square or rectangular map does not give good convergence as was explained in section 1 and therefore, in this experiment, a non-rectangular map was used and its shape was calculated as follows.

To allow the map to naturally converge around a sphere we can consider each row of neurons in the map to represent a horizontal "slice" of a sphere with the angle of latitude between adjacent slices being equal. The number of neurons n_k in the slice k is proportional to the circumference of the slice. If we have a total of d slices, $n_k = 2 \times d \sin(\frac{\pi}{2} - \theta_n)$ where θ_n is the angle of latitude of slice n. The resulting map is the correct shape to converge to the topology of a sphere. A diagram of this map is shown in Figure 2.

The neighbourhood function must also wrap around the map in the direction of the slices such that the first and last neurons in the same row are considered to be adjacent. There is no wrap-around in the columns of the map since the top and bottom of the columns should approach opposite sides (poles) of the sphere during training.

4 Angular Vector Quantization

Using a map of the shape described in the previous section is suitable for convergence of weights around the surface of a sphere using type A or B competition. However, it was found that not all weight vectors had similar magnitudes i.e. some were closer to the origin than others. This observation can be seen in Figure 1 where the vectors at one end of the scale invariant map are

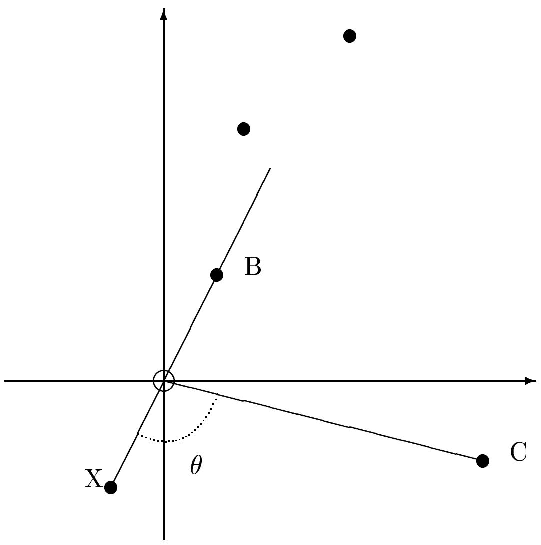

Figure 3: X is the input vector, B is the winner using type B competition, and C is the winner using type C competition.

closer to the origin than all of the others, reducing the scale invariant properties of the network. To solve this problem we use a third type of competition:

Type C: The neuron with the smallest angle from the input vector wins. This is the neuron with highest activation after the the inputs are fed forward as shown in equation (4).

$$\min_i(f_i) = \min_i(\theta) \text{ where } cos(\theta) = \frac{\mathbf{x}.\mathbf{w}_i}{\| \mathbf{x} \| . \| \mathbf{w}_i \|} \tag{4}$$

For each competition, $\| \mathbf{x} \|$ is equal for all i and therefore can be ignored. Type C competition has the effect of initially pulling weights away from the origin but after the weights $\mathbf{w_i}$ have similar magnitudes, type A and type C competition are equivalent.

Type C competition quantizes an input to the neuron whose weight vector has the closest direction to the input vector. When this method is applied to an untrained network, it is possible for weights to become stuck near the origin, depending on the initial (random) state of the weights. Consider, for example, a training vector represented by point X in Figure 3. Using the nearest neighbour rule (type B competition), the neuron whose vector is represented by point B would be the winner. This is desirable since we want to move the vectors to cover the distribution of the angles in the training data. However, with type C competition this point would not be chosen since it is on the opposite side of the origin from the input vector and these vectors form an angle of up to π radians. Instead, the neuron labelled C would be the winner, since the angle θ is smallest.

Figure 4 shows the situation where the training vector, X, is closest in Euclidean distance to the weight vector at point B, but this is not the most ideal winner. The neuron at point C has a more similar angle to the input vector, and it should be chosen as the winner even although it has a dissimilar position. This has an improved effect on forming a uniform distribution of angles around the origin and there is therefore an advantage in using type C competition.

After a network has been trained for a short time with nearest neighbour competition, the weight vectors are spread around the origin and the problem described in figure 3 has no effect. A good strategy is to train the network for a few hundred cycles using type A or B competition and then switch to type C. This technique gave better convergence than using any one type on its own.

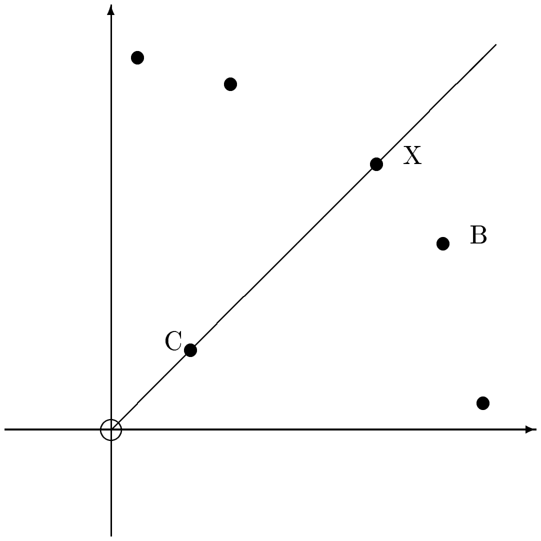

Figure 4: X is the input vector, B is the nearest neighbour and C is the winner using type C competition.

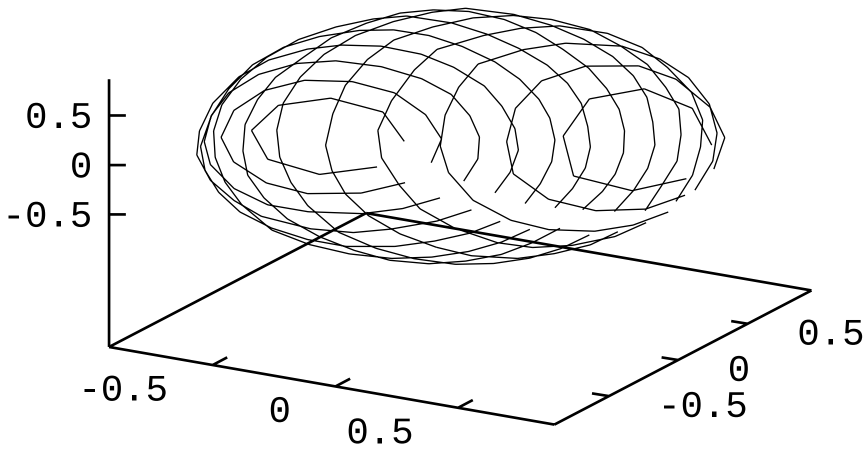

Figure 5: The weights of a two dimensional scale invariant map trained on uniformly distributed three dimensional data.

	Inputs							
	bat	bed	beep	boot	but	far	pit	put
bat	608	0	104	93	71	269	23	53
bed	7	1000	0	0	28	17	133	0
beep	0	0	896	0	0	1	0	3
boot	0	0	0	856	0	0	0	100
but	6	0	0	0	855	7	39	0
far	379	0	0	0	43	703	2	101
pit	0	0	0	0	3	3	800	0
put	0	0	0	51	0	0	3	743
% error	1.3	0	10.4	9.3	14.5	2.8	20	15.7

Table 1: Results of speaker dependant phoneme classification using a 2D scale invariant map. Each phoneme was presented 1000 times and the percentage error was calculated.

5 Experimental Results

Figure 5 shows a converged map with 15 slices of neurons. The map has been successfully trained with each neuron capturing a similar volume to all other neurons. The map has formed the shape of a sphere with all weight vectors approximately equidistant from the origin, indicating that scale invariant classification is possible. Twists did not occur in any of the experiments using this method and, therefore, the use of type C competition is deemed to be justified.

We have previously used a one-dimensional map in [3] to classify voiced phonemes from raw sound data. The two-dimensional map with type C competition was applied to the same problem. A single speaker was recorded speaking eight different words ("bat," "bed," "beep," "boot," "but," "far," "pit" and "put") and the unvoiced sounds were cut out, leaving only the vowel phonemes. Eight maps with 15 slices each were trained on a different vowel. With the particular speaker that was recorded, some of the words contained the same vowel phoneme. "Bat" and "far" were equivalent(Scottish speaker) and "boot" and "put" were also equivalent. For each vowel sound, a window of 70 samples (sampled at 8 kHz) was fed forward through all eight networks and an overall winner was selcted. This was repeated 1000 times for each word and the results of this test are shown in table 1.

References

[1] C. Fyfe. Radial feature mapping. In *International Conference on Artificial Neural Networks, ICANN95*, Oct. 1995.

[2] Tuevo Kohonen. *Self-Organising Maps.* Springer, 1995.

[3] S. McGlinchey and C. Fyfe. Automatic extraction of phase and frequency information from raw voice data. In *International Conference on Artificial Neural Networks and Genetic Algorithms, ICANNGA97.* Springer, April 1997.