# Noise to Extract Independent Causes

Darryl Charles and Colin Fyfe University of Paisley, Scotland

Abstract. Noisy threshold activation functions are used to force sparse responses on the output neurons of an unsupervised neural network enabling the network to identify the underlying independent factors of visual data. The addition of noise into the network enables us to control the response of the network to the data so we can force only as many outputs to respond to the data as there are significant factors in the data. Noise is also used to modularise the response of the network so that factors with temporal correlation may be coded in the same module of the output space.

## 1. Introduction

The coding of visual data has been a subject of interest recently [3, 1]. One line of thought in this field of research is that natural images are composed of a few structural primitives and the distribution on the output neurons of a neural network trained on such data should be sparse (i.e.the distribution is highly peaked around zero with heavy tails). It is argued [3] that not only are there links to biological neurons with sparse coding networks which only "fire" infrequently but that this form of coding is an efficient compromise between an entirely local coding and a compact (or dense) coding.

The advantage of multiple cause coding [2] in the context of image data is clear, as most visual objects may be best represented as a combination of independent features, for example someones head may be thought of as being made up of eyes, ears, nose etc. A difficulty with problems of this type can be that the underlying causes in the data are hidden, that is we do not know how many fundamental causes that there are in the data. We present a solution to this by controlling the number of factors that we extract in the data with additive noise on the outputs.

A common benchmark which is used to test networks of this kind is the multiple cause bars data (Figure 1). In this, the artificial image data set is made up of a random mix of horizontal and vertical bars where bar pixels have a value '1' (black) and the white background pixels have a value '0'. Bars appear at the inputs of the network with an equal, fixed probability (normally 1/8 where an input grid has an area of 8x8). The network should respond so that separate outputs to the network each identify a single whole bar.



Figure 1: Sample mixture of bars input data

# 2. The Network

The basic operation of our algorithm is similar to non-linear PCA [4]. The input data is fed forward via weights from the input neurons to the output neurons where a linear summation  $\mathbf{a} = \mathbf{W}^T \mathbf{x}$  is performed, before a non-linearity is applied to give  $\mathbf{y} = f(\mathbf{a})$ . This activation is then fed back through the same weights and subtracted from the inputs to calculate a residual error  $\mathbf{r} = \mathbf{x} - \mathbf{W}\mathbf{y}$ . The residual error is then used to perform Hebbian learning between inputs and outputs so that  $\Delta \mathbf{W} = \eta \mathbf{r} \mathbf{y}^T$  where  $\eta$  is a learning rate parameter. Growth of the weights in the network is automatically controlled due to the negative feedback.

A non-linearity that we have previously used within this methodology was a straightforward rectification of the outputs by setting any negative values on the outputs to zero, i.e.  $\mathbf{y} = f(\mathbf{a}) = [\mathbf{a}]^+$ . We have linked these rectification constraints within this methodology to Factor Analysis [1] and have shown that this network is capable of identifying all of the bars in the data described above. A more robust function is  $\mathbf{y} = f(\mathbf{a}) = \sigma \log(1 + \frac{\exp(\mathbf{a}-\theta)}{\sigma})$ . This function not only rectifies negative values but also act as a *soft threshold function* on the output values. Each output then only responds strongly to projections from the inputs through the weights on to  $y_i$  that are above a certain threshold value. ( $\theta$  controls the extent of the thresholding and  $\sigma$  the gradient).

## 3. Results

In this section the variety of aspects of our network are illustrated with the benchmark "bars data". The networks are trained over 50000 presentations of the data with a learning rate of 0.05 which is annealed linearly to zero over the training period. Twenty four outputs are used in each experiment and the data contains 16 causes. The squares in the following figures are individual weight vectors into an output neuron arranged 2-dimensionally for convenience of viewing the results. The diameter of circles within the squares represent the individual weight values where black is a positive weight and white is a negative weight.

### 3.1. Random Mixes of Horizontal and Vertical Bars

We use the more difficult form of this problem in which horizontal and vertical bars may appear together. Each bar may appear at the input to the network with a probability of 1/8 as described in Section 1.

The converged weights of our network when using using the straightforward rectification of the outputs is shown on the left in Figure 2. We have found



Figure 2: Converged weights in the straightforword rectification network (left) and the soft threshold network (right)

that the soft threshold non-linearity (right of Figure 2) is more effective than the plain rectification on the outputs (left of Figure 2). With exactly as many outputs as bars then both networks identify all of the bars, whereas when there are more outputs in the network than bars that make up the data set then all of the bars are identified but in the case of the rectified network some bars are identified more than once, and junctions of bars or combinations of bars are also found.

#### 3.2. Additive Noise

Additive noise (zero mean, gaussian noise of standard deviation 0.01 is added to every output) is beneficial in these networks when added to the outputs after the application of the non-linearity. Figure 3 show that additive noise enables the network to identify all of the individual bars. That is, only as many outputs are used as are required in the coding; the weights connected to the other outputs all tend to zero. We can also add noise in a graduated way (minimum variance on first output) across the outputs so that the bars are learned so that they are grouped at one side of the outputs.

Figure 3: Trained soft threshold network with additive noise - 24 outputs.

#### 3.3. Illusory Causes

In the situation where bar patterns are non-sparse (the bars appear with a random probability of 7/8 each) then networks with built in sparse priors cannot expect to identify the individual bars. Using our threshold network with additive noise the results (Figure 4) indicate that the network converges to a very sparse representation - i.e. the illusory bars between the actual bars patterns. As the network operates so as to find a sparse response and because the individual bars are appearing in dense patterns together, then the network cannot learn the individual bars. Instead the network learns the spaces between bars patterns which is the appropriate sparse response (illusory bars). Note that in this experiment that the horizontal and vertical bars are not mixed so as to allow the weights to learn a more visually interesting response.

 ****		

Figure 4: The noisy soft threshold network discovers illusory bars - 24 outputs

#### 3.4. Topographical Ordering and Modular Noise.

By introducing temporal context via lateral connections between the outputs, we can achieve an ordering in the coding of the bars at the outputs. This is achieved by giving values to the lateral weights proportional to the distance between any particular pair of outputs and increasing an output's activation in proportion to the previous values of the other outputs weighted by the lateral connections. The lateral connections come into play after the feedforward stage of the algorithm but before the application of the non-linearity. So  $a_i \leftarrow a_i(t) +$  $\sum_{j=-k, j\neq 0}^{k} (a_{i+j}(t-1)\tau_j)$ , where  $\tau_j$  is a lateral weight. The lateral weights can be set in a number of ways, for example using a gaussian or difference of gaussians, or gaussian weighted cosines or sines, however the method that we use here is simply to connect an output only to its nearest four neighbours. Lateral weight values to the left of an output are -0.2 and -0.7 (nearest) and to the right 0.2 and 0.7 (nearest). If the bars appear as part of horizontally or vertically moving sequences then this network now can order the outputs so that temporally close features are coded spatially close at the output neurons. Now using these lateral connections along with modular noise we can force vertical and horizontal bars to be learned in different modules of the output space. This is achieved by creating two "wells of attraction" at the outputs by adding zero mean gaussian noise after the application of the non-linearity proportional to  $|\cos(2^*\pi^*)$  (number of output/total number of outputs) where each of the outputs are numbered 1,2,3 ... . This has the effect of encouraging one set of features to be coded around the output that is one third from the left and one set of features to be coded around the output that is one third from the right (Figure 5). This happens because the noise is of lower magnitude at these points and so the error minimisation term of (3) dominates.


Figure 5: Trained weights of soft threshold network with modular noise.

# 4. Discussion on the use of Noise

Non-linear PCA can be shown to be an approximation to the minimization [4] of  $\mathbf{J} = \mathbf{E}(\mathbf{x} - \mathbf{W}\mathbf{y})^2 = \mathbf{E}(\mathbf{x} - \mathbf{W}\mathbf{f}(\mathbf{a}))^2$ , where E() is the expectation operator. Now we add noise to the process so that  $\mathbf{y} = \mathbf{f}(\mathbf{a}) + \mu$  where  $\mu$  is a vector of independently drawn noise from a zero mean distribution. So defining  $\mathbf{f} = \mathbf{f}(\mathbf{a})$ ,

$$\mathbf{J}' = E(\mathbf{x} - \mathbf{W}\mathbf{y})^2 \qquad (1)$$

$$= E(\mathbf{x} - \mathbf{W}(\mathbf{f} + \mu))^2$$

$$= E\{\mathbf{x}\mathbf{x}^{\mathrm{T}} - \mathbf{x}(\mathbf{f} + \mu)^{\mathrm{T}}\mathbf{W}^{\mathrm{T}} - \mathbf{W}(\mathbf{f} + \mu)\mathbf{x}^{\mathrm{T}}$$

$$+ \mathbf{W}(\mathbf{f} + \mu)(\mathbf{f} + \mu)^{\mathrm{T}}\mathbf{W}^{\mathrm{T}}\}$$

$$= E\{\mathbf{x}\mathbf{x}^{\mathrm{T}} - \mathbf{x}\mathbf{f}^{\mathrm{T}}\mathbf{W}^{\mathrm{T}} - \mathbf{W}\mathbf{f}\mathbf{x}^{\mathrm{T}} + \mathbf{W}\mathbf{f}\mathbf{f}^{\mathrm{T}}\mathbf{W}^{\mathrm{T}} + \mathbf{W}\mu\mu^{\mathrm{T}}\mathbf{W}^{\mathrm{T}}\}$$

$$= \mathbf{J} + \mathbf{W}\mathbf{A}\mathbf{W}^{\mathrm{T}} \qquad (2)$$

where  $\Lambda$  is a diagonal matrix with positive elements on the diagonal equal to the variance of the noise. Terms containing single expectations of  $\mu$  can be removed from the equation as they are drawn from zero mean noise. Intuitively, we see that, with a low magnitude of added noise to the network the first term of (2) dominates and so non-linear PCA is performed in the normal manner. Extracting a weight update rule from the energy equation (2), we have

$$\Delta \mathbf{W} \propto \mathbf{f}(\mathbf{a})(\mathbf{x} - \mathbf{W}\mathbf{f}(\mathbf{a})) - \mathbf{\Lambda}\mathbf{W}^{\mathrm{T}}$$
(3)

If the noise level is increased then the learning is moderated by this additional weighted noise term which has the effect of forcing some weight vectors to zero (the degenerate solution). As the amplitude of the noise becomes larger then the right term has more effect and pushes the weight values downwards.

An alternative view point is to minimise  $F(\mathbf{W}, \mathbf{x}) = (\mathbf{x} - \mathbf{W}\mathbf{f}(\mathbf{a}))^2$  under the constraint that  $g(\mathbf{W}) = \mathbf{W}^{T}\mathbf{W}$ . In terms of a Langrange multiplier then we have  $L(\mathbf{W}, \lambda) = \mathbf{F}(\mathbf{W}, \mathbf{x}) + \lambda \mathbf{g}(\mathbf{W})$  and using this we can gain further insight

into the use of noise when added on to the outputs. We wish to miminise  $F(\mathbf{W}, \mathbf{x})$  subject to  $g(\mathbf{W}) \geq \mathbf{0}$ , therefore at the optimal point of this equation the Kuhn-Tucker conditions must hold so that:

$$\frac{\partial \mathbf{F}}{\partial w_{ij}} + \lambda_{ij} \frac{\partial g(\mathbf{W})}{\partial w_{ij}} = 0$$
(4)

$$g(\mathbf{W}) \geq 0 \tag{5}$$

$$\lambda_{ij} \leq 0 \tag{6}$$

$$\lambda_{ij} g(\mathbf{W}) = 0 \tag{7}$$

Now if noise is added to the network then  $\lambda_{ij}$  is non-zero and so  $g(\mathbf{W})$  must be zero, that is all of the weights must be zero or the columns of the weight matrix must be orthogonal. So there are two possible solutions to this equation that any particular weight vector can adopt: where there is a strong, persistent feature in the input space (and the noise on the network is not too high) then the first term of (4) dominates and therefore the fourth condition (7) is held so that learning is based mostly on the minimization of the error term. If on the other hand a feature is of small magnitude or appears with a low probability and as a result is overwelmed by noise then the weights are pushed towards zero because the second regularization term in (4) now dominates.

### 5. Conclusion

The introduction of noise into the network is very powerful in that a very sparse response to the data is enforced and may be used to control the extraction of underlying factors from the data. By adding graduated packets of noise on the outputs so that there are areas of low amplitude noise in some sections of the outputs and high in others then the network can be encouraged to learn a modular response to features in the data.

## References

- D. Charles and C. Fyfe. Modelling multiple cause structure using rectification constraints. Network: Computation in Neural Systems, (9):167-182, May 1998.
- [2] P. Dayan and R. S. Zemel. Competition and multiple cause models. Neural Computation, 7:565-579, 1995.
- [3] D. Field. What is the goal of sensory coding. Neural Computation, 6:555-601, 1994.
- [4] Juha Karhunen and Jyrki Joutsensalo. Representation and separation of signals using nonlinear pca type learning. Neural Networks, 7(1):113-127, 1994.