Self-Supervised Adaptive Networks

S. P. Luttrell
Image Processing Research Section, Defence Research Agency,
St Andrews Rd, Malvern, Wors, WR14 3PS, United Kingdom

A scheme for training multilayer unsupervised networks is presented, in which control signals propagate downwards from the higher layers to influence the optimisation of the lower layers. Because there is no external teacher involved, this is called self-supervised training. The author demonstrates both theoretically and numerically how self-supervision emerges when a simple network built out of vector quantisers is optimised.

I. INTRODUCTION

The supervised training of adaptive networks can solve problems which require an optimal nonlinear transformation from an input to an output space. Conversely, unsupervised training of adaptive networks can solve problems which require an optimal encoding of an input space. Supervised training requires an external teacher, whereas unsupervised training does not.

There is a useful middle ground between the extremes of supervised and unsupervised networks. Assume that a multilayer unsupervised network is a stack of single-layer unsupervised networks. There are two distinct types of cost function that we could introduce: global and local. A local cost function might ensure that each layer strives to code the output of the previous layer, so that it can be reconstructed with low distortion, as in a single-layer unsupervised network. A global cost function might ensure that when the input is passed through all the layers, and then passed all the way back again, it still forms a low-distortion reconstruction. To achieve global optimisation, we must train the layers cooperatively so that each layer of the network internally supervises other layers. We call this mode of operation ‘self-supervision’, and in this paper, which is an extended version of [1], we aim to demonstrate this type of network training.

Although training a one-layer unsupervised network (input \(\rightarrow\) code) is equivalent to training an appropriate two-layer supervised network (input \(\rightarrow\) hidden \(\rightarrow\) output) in which the output is required to reproduce the input (i.e. an autoassociative supervised network), this equivalence does not generalise to multilayer unsupervised networks. The local cost functions that connect adjacent pairs of layers in multilayer unsupervised networks cannot be accounted for in standard autoassociative network models, which have a cost function defined in terms of the input and output only.

Self-supervision is important in low-level vision applications, such as the extraction of features from images. To ensure that useful information arrives at a given layer of a vision network, we must ensure that lower (or earlier) network layers process their inputs appropriately. This type of global cost function leads naturally to self-supervised training schemes. A similar application is data fusion of the information collected by separate sensors, where a multilayer system progressively transforms and collates the sensor information in a macroscopic version of low-level vision. Again, self-supervision emerges naturally from the global cost function.

In [2] we discuss a hierarchical network that computes the maximum relative entropy estimate of the joint probability density function of its own inputs. We show that this reduces to maximising the sum of mutual informations between various internal components of the network; this is a global cost function. This training scheme makes different regions of the hierarchical network mutually supervise each other, and is thus an example of a self-supervised network.

Because the goal of this paper is to explain the principles that underlie self-supervision, we limit our attention to the case of a simple two-layer unsupervised network, built out of vector quantisers [3-5].

II. VECTOR QUANTISATION MODEL

In this section we present a résumé of vector quantisation theory (e.g. the so-called LBG algorithm [6]), its extension to vector quantisation for communication over a noisy communication channel [4, 5, 7], and its further extension to mutually interfering communication channels [1]. The noisy communication channel model is formally equivalent to the model that is implicit in the theory of topographic mappings [3].

Throughout this paper we make use of functions such as \(y(x)\) and \(x'(y)\), whose ‘input’ and ‘output’ spaces we treat as continuous, rather than discrete. We do this for two reasons. First, it makes our derivations easier to perform using the rich language of continuum notation. Secondly, it demonstrates that our results are general statements about transformations between input and output spaces, and are thus not limited to vector quantisers (continuous \(x\), discrete \(y\)).
A. Vector quantisation for noiseless and noisy channels

\[ D_1 = \int dx P(x) ||x'(y(x)) - x||^2 \]  

To minimise \( D \), we need to optimise the pair of functions \((y(x), x'(y))\); the LBG algorithm \([9]\) is an effective solution to this.

Consider a noisy communication channel, which we may model as a generalisation of Equation 2.1.

\[ D_2 = \int dx P(x) \int dy' \pi(y' - y(x)) ||x'(y') - x||^2 \]  

In Equation 2.2 we assume that \( y' = y(x) + n \), where \( n \) is an additive random noise variable with PDF \( \pi(n) \). Furthermore, we assume that \( x \) and \( n \) are statistically independent, so that \( P(x, n) = P(x)\pi(n) \). A generalisation of the LBG algorithm \([9, 11]\) is an effective solution to this optimisation problem, as we explain below.

B. Nearest neighbour and minimum distortion encoding

In order to minimise \( D_2 \) we functionally differentiate it and then locate the zeros of \( \frac{\delta D_2}{\delta y(x)} \) and \( \frac{\delta D_2}{\delta x'(y)} \). After some calculation we obtain

\[ x'(y) = \frac{\int dx P(x) \pi(y - y(x)) x}{\int dx P(x) \pi(y - y(x))} \]  

\[ \Delta x'(y) = \epsilon \pi(y - y(x)) (x - x'(y)) \]  

\[ y(x) = \arg \min_y \int dy' \pi(y' - y) ||x'(y') - x||^2 \]  

where \( \arg \min_y \cdot \) means 'select the value of \( y \) that minimises \( \cdot \)'. These results are a compact specification of a pair of vector quantiser training algorithms:

1. Batch update (Equation 2.3). This is the generalisation to noisy communication channels of one cycle of the LBG algorithm \([9]\).

2. Continuous update (Equation 2.3). This is identical to the topographical mapping training algorithm \([3]\), apart from a subtle difference in the encoding prescription that we discuss below. \( \pi(n) \) can therefore be interpreted as a 'neighbourhood function'.

In Equation 2.4 there are two distinct cases to consider:

1. Nearest neighbour encoding (NN): when \( \pi(n) = \delta(n) \) (i.e. the noiseless channel case) Equation 2.4 specifies a nearest neighbour encoding prescription that we denote as \( y^0(x) \). This prescription selects the encoded version of \( x \) to be the \( y \) that minimises the distortion \( ||x'(y) - x||^2 \).

2. Minimum distortion encoding (MD): when \( \pi(n) \neq \delta(n) \) Equation 2.4 specifies a minimum distortion encoding prescription \( y(x) \), that anticipates the subsequent effect of possible channel distortions. This prescription selects the encoded version of \( x \) to be the \( y \) that minimises the expected distortion \( \int dy' \pi(y' - y) ||x'(y') - x||^2 \).

NN encoding is sometimes a suitable approximation to MD encoding when \( \pi(n) \neq \delta(n) \), and it is widely used in the literature as the standard encoding prescription irrespective of whether or not it minimises a meaningful distortion.

C. Relationship to unsupervised neural networks

Figure 2: Equivalence of a folded quantiser and a single-layer 'winner-take-all' neural network.
The vector quantiser in Figure 1 can be interpreted as a ‘winner-take-all’ neural network, as we show in Figure 2, where we fold the vector quantiser diagram to make the relationship clearer.

The noisy communication channel version of the vector quantiser is more complicated. The noise process \( \pi(n) \) causes confusion between different values of \( x \) and \( y \). The vector quantiser responds by ensuring that \( x' \) is not too sensitive to the types of change in the value of \( x \) that this noise process brings about. If \( \pi(n) \) is a localised function of \( n \), such as a zero-mean Gaussian, then \( x' \) must become a slowly varying function of \( y \).

In the neural network picture this would translate into a smooth variation of weight vector as we pass from node to node in the output layer. This is exactly what we observe in a topographical mapping neural network using a topographical mapping neural network using neighbourhood function \( \pi(n) \) [3]. The exact details of the standard topographical mapping training algorithm are not precisely the same as Equation 2.3b; we use minimum distortion encoding, whereas topographic mapping networks use nearest neighbour encoding. In [8] we demonstrate that using minimum distortion encoding in preference to nearest neighbour encoding, actually simplifies some of the properties of topographical mappings. In particular, the density of weight vectors acquires a functional form that is independent of the details of the neighbourhood function.

D. Vector quantisation for coupled channels

Now consider a pair of mutually coupled communication channels, which we may model as a generalisation of Equation 2.2. The Euclidean distortion in such a system is given by

\[
D_3 = \int dx_1 dx_2 P(x_1, x_2) \int dy_1' P(y_1'|y_1(x_1), y_2(x_2)) ||x_1'(y_1') - x_1||^2 + (1 \leftrightarrow 2) \tag{2.5}
\]

We may interpret the various contributions to this expression for \( D_3 \) as follows:

1. The \( \int dx_1 dx_2 P(x_1, x_2) \) integration averages over all pairs of inputs \( (x_1, x_2) \) to channels 1 and 2, and \( P(x_1, x_2) \) specifies the probability density with which each pair occurs.

2. The \( \int dy_1' P(y_1'|y_1, y_2) \) integration averages over all possible distortions of channel 1, due to the mutual coupling of channels 1 and 2.

3. \( ||x_1'(y_1') - x_1||^2 \) is the Euclidean distance between the input vector \( x_1 \) and its reconstruction \( x_1'(y_1') \) from the distorted version of channel 1.

4. \((1 \leftrightarrow 2)\) denotes an analogous term for channel 2.

By comparing Equation 2.5 with Equation 2.2 we see that the marginals \( P(y_1|y_1, y_2) \) and \( P(y_2|y_1, y_2) \) now play the part of the noise PDF \( \pi(y) \).

We functionally differentiate \( D_3 \) to obtain \((k = 1 \text{and} 2)\)

\[
\Delta x_k'(y_k') = \frac{\int dx_1 dx_2 P(x_1, x_2) P(y_1'|y_1(x_1), y_2(x_2))}{\int dx_1 dx_2 P(x_1, x_2) P(y_1'|y_1(x_1), y_2(x_2))} \tag{2.6}
\]

\[
(y_1(x_1), y_2(x_2)) = \arg \min_{y_1, y_2} \left( \int dy_1' P(y_1'|y_1, y_2)||x_1'(y_1') - x_1||^2 + (1 \leftrightarrow 2) \right) \tag{2.7}
\]

Strictly speaking, when we functionally differentiate \( D_3 \) to determine its dependence on \( y_1(x_1) \) and \( y_2(x_2) \), we should include any derivatives that arise from the implicit dependence of \( P(y_1'|y_1, y_2) \) and \( P(y_2'|y_1, y_2) \) on \( y_1(x_1) \) and \( y_2(x_2) \). However, we shall assume that \( P(y_1|y_1, y_2) \) and \( P(y_2|y_1, y_2) \) are time-averaged quantities, and so their instantaneous dependence on the functions \( y_1(x_1) \) and \( y_2(x_2) \) is weak enough to be ignored. This is a type of ‘mean field’ approximation.

Equation 2.7 specifies a minimum distortion prescription in which we simultaneously optimise \( y_1(x_1) \) and \( y_2(x_2) \); this is ‘cooperative encoding’. Note that, for simplicity, in Figure 3 we omit any reference to the fact that the encoding functions \( y_1(x_1) \) and \( y_2(x_2) \) are implicitly coupled via Equation 2.7. If we ignore the coupling between the channels, we replace \( P(y_1'|y_1, y_2) \) with...
vector quantisation, then the bias of im um distortion prescription. In Table I we summarise nearest neigh b our prescription, instead of the full min-
P(scrib es the distortion that arises due to coupling of the chan-
processes can b e mo delled using $P(y'_1, y'_2 | y_1, y_2)$; this is ‘inde-
Pendent encoding’. Furthermore, in both the cooperative and the independent encoding cases, we could use the nearest neighbour prescription, instead of the full minimum distortion prescription. In Table II we summarise the various modes in which the two channels operate.

E. A specific model of the channel coupling

![Diagram of channel coupling](image)

Figure 3: A pair of coupled vector quantisers of the type shown in Figure [1]. The conditional $P(y'_1, y'_2 | y_1, y_2)$ describes the distortion that arises due to coupling of the channels. If the channels were conditionally independent then $P(y'_1 | y_1)$ and $P(y'_2 | y_2)$ would suffice. Many distortion processes can be modelled using $P(y'_1, y'_2 | y_1, y_2)$, but in the Appendix we restrict our attention to the distorting effects of a second stage of vector quantisation $P(y'_2 | y_1)$ and $P(y'_2 | y_1, y_2)$ with $P(y'_2 | y_2)$; this is ‘independent encoding’. Furthermore, in both the cooperative and the independent encoding cases, we could use the nearest neighbour prescription, instead of the full minimum distortion prescription. In Table II we summarise the various modes in which the two channels operate.

In Figure 4 we represent diagrammatically in $(y_1, y_2)$-space (and $(y'_1, y'_2)$-space) the role of the various PDFs in Equation 2.5. We show in the Appendix that, if we pass the output pair $(y_1, y_2)$ through another stage of vector quantisation, then the bias of $P(y'_1, y'_2 | y_1, y_2)$ is

towards higher values of $P(y_1, y_2)$. The overall effect is to bias the two marginals $P(y'_1 | y_1, y_2)$ and $P(y'_2 | y_1, y_2)$ (as shown) in a data-dependent way. This type of effect cannot be accounted for in the original additive noise model using $\pi(y - y(x))$.

In the Appendix we derive the first-order approximation to the effect of a second-stage of vector quantisation in the form

$$P(y' | y) \propto \rho(y') \exp\left(-\frac{\alpha_N \rho(y)\|y' - y\|^N}{N}\right) \tag{2.8}$$

where we use the notation $y = (y_1, y_2)$, and we use $\rho(y)$ to denote the density of code vectors of the second-stage vector quantiser. For scalar $y_1$ and $y_2$ (i.e. $y_1 = y_1$ and $y_2 = y_2$) we can readily marginalise this result to obtain

$$P(y'_1 | y_1, y_2) \propto \rho(y'_1, y_2) \exp\left(-\pi \rho(y_1, y_2)(y'_1 - y_1)^2\right) \tag{2.9}$$

A similar result holds for $P(y'_2 | y_1, y_2)$. The mean and standard deviations of $y'_1 - y_1$ are

$$\text{mean} \approx \frac{1}{2\pi \rho(y_1, y_2)} \frac{\partial \rho(y_1, y_2)}{\partial y_1} \tag{2.10}$$

$$\text{SD} \approx \frac{1}{\sqrt{2\pi \rho(y_1, y_2)}}$$

A similar pair of results holds for $y'_2 - y_2$. Note that in Figure 4 the bias of the neighbourhood functions is consistent with Equation 2.10. Because $\rho(y_1, y_2) \propto \sqrt{P(y_1, y_2)}$ (for an optimised vector quantiser) we may also use Equation 2.9 to relate the neighbourhood functions $P(y'_1 | y_1, y_2)$ and $P(y'_2 | y_1, y_2)$ of the two channels to their joint PDF $P(y_1, y_2)$.

F. Self-supervision

A subtle side effect of channel coupling is ‘self-supervision’. This phenomenon is easiest to understand in terms of the neural network viewpoint. When we optimise the network in Figure 3 the interaction between the layers causes the optimisation of the bottom left (channel 1) and bottom right (channel 2) parts of the network to become indirectly coupled through the top (the coupled distortion) part of the network; this is ‘self-supervision’. If the top part of the network were absent, then it would reduce to two separate unsupervised networks (corresponding to channels 1 and 2).

The self-supervision principle is quite general: whenever we build a multilayer unsupervised network, the various layers must cooperate to minimise the global (and local) distortion function. This cooperation manifests itself as control signals that the higher layers of the network pass down to the lower layers of the network. The similarity to the back-propagating signals that we use to train a supervised network is not accidental. Indeed, we can build a hybrid network in which an external teacher gives rise to standard supervised backpropagating signals, and the internal layers of the network give rise to self-supervised back-propagating signals.
Table I: Various encoding and channel modes. NN — nearest neighbour encoding. MD — minimum distortion encoding. I — independent channels. C — correlated channels.

<table>
<thead>
<tr>
<th>Independent channels</th>
<th>Nearest neighbour encoding</th>
<th>Minimum distortion encoding</th>
</tr>
</thead>
<tbody>
<tr>
<td>Correlated channels</td>
<td>NN/I</td>
<td>MD/I</td>
</tr>
<tr>
<td></td>
<td>NN/C</td>
<td>MD/I</td>
</tr>
</tbody>
</table>

Figure 5: Two-layer self-supervised neural network. The orientation of this diagram is 90° clockwise relative to Figure 2. The bottom left and right parts of the diagram correspond to a pair of folded vector quantisers in channels 1 and 2, respectively (i.e. two copies of Figure 2). The top part of the diagram implements the joint distortion \( P(y_1', y_2' | y_1, y_2) \) that converts \((y_1, y_2)\) into \((y_1', y_2')\); this distortion acts after encoding (i.e. forward pass) but before decoding (i.e. backward pass) in the bottom part of the diagram. The top part of the diagram is only a schematic representation of top-down control of lower layers by higher layers of a multilayer network; the top-down control could come from any source.

III. NUMERICAL EXPERIMENTS

In this Section we present some numerical simulations that demonstrate self-supervision in a simple network of the type shown in Figure 3 (whose neural network interpretation is shown in Figure 5). We run all of our numerical simulations using a four-dimensional input vector \( \mathbf{x} = (x_1, x_2) = (x_{11}, x_{12}, x_{21}, x_{22}) \), and with scalar outputs from the encoders \( y_1(x_1) \) and \( y_2(x_2) \). This is the minimal network that we can use to demonstrate self-supervision. In realistic applications more complicated networks would arise, but they would all operate according to the same general principles.

Table II: Values of \( \pi_\pm \) and \( \pi_0 \) that we use in four separate numerical experiments. In experiment 1 we use an unbiased distortion, whereas in experiments 2-4 we use a biased distortion, as indicated in Equation 2.9.

<table>
<thead>
<tr>
<th>Experiment</th>
<th>( \pi_- )</th>
<th>( \pi_0 )</th>
<th>( \pi_+ )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.20</td>
<td>0.60</td>
<td>0.20</td>
</tr>
<tr>
<td>2</td>
<td>0.15</td>
<td>0.60</td>
<td>0.25</td>
</tr>
<tr>
<td>3</td>
<td>0.10</td>
<td>0.60</td>
<td>0.30</td>
</tr>
<tr>
<td>4</td>
<td>0.05</td>
<td>0.60</td>
<td>0.35</td>
</tr>
</tbody>
</table>

\section{A. Basic network operation}

\[
P_{\text{approx}}(y_1'|y_1, y_2) = \begin{cases} 
\pi(y_1' - y_1) & \partial \rho(y_1, y_2) / \partial y_1 \geq 0 \\
\pi(y_1 - y_1') & \partial \rho(y_1, y_2) / \partial y_1 < 0 
\end{cases} (3.1)
\]

\[
\pi(\Delta y) = \begin{cases} 
\pi_- & \Delta y = -1 \\
\pi_0 & \Delta y = 0 \\
\pi_+ & \Delta y = +1 \\
0 & \text{otherwise}
\end{cases} (3.2)
\]

\[
D(\mathbf{x}) = \int dy_1' P(y_1'|y_1, y_2) \left( (x_{11}'(y_1) - x_{11})^2 + (x_{12}'(y_1) - x_{12})^2 \right) + (1 \leftrightarrow 2) (3.3)
\]

1. Clamp the inputs. We generate \( \mathbf{x} = (x_1, x_2) = (x_{11}, x_{12}, x_{21}, x_{22}) \) as follows: we choose \((x_{11}, x_{12})\) to be a uniformly distributed random vector in a disc-shaped region, and then generate \((x_{21}, x_{22})\) by rotating \((x_{11}, x_{12})\) about the disc’s centre by a random angle uniformly sampled from the interval \([-\theta, +\theta]\). We use this prescription to ensure that the marginal PDFs \( P(x_{21}, x_{22}) \) and \( P(x_{21}, x_{22}) \) are the same, and to control the degree of correlation between \((x_{11}, x_{12})\) and \((x_{21}, x_{22})\). We use \( \theta = 0.5 \) and \( \theta = 1.0 \) in our simulations.

2. Compute the nearest neighbours. This yields \( y^0 = (y^0_1, y^0_2) \).

3. Compute the distortion PDFs. Although we couch the following discussion in terms of \( y_1 \) alone, the
same remarks also apply to \( y_2 \). We generate a convenient numerical approximation to \( P(y_1'|y_1, y_2) \) as follows. First, we use the asymptotic result \( \rho \propto P^{N/2} \propto P^{1/2} \) (for the case of \( N = 2 \) dimensions) in Equation 2.9. Secondly, we approximate \( P(y_1'|y_1, y_2) \) as in Equation 3.1, where, for discrete-valued \( y_1 \), we define \( \pi(y_1' - y_1) \) as in Equation 3.2 that \( P_{\text{approx}}(y_1'|y_1, y_2) \) is a fairly crude approximation, but nevertheless it can correctly preserve the sign of the bias of \( P(y_1'|y_1, y_2) \), which is crucial to the success of our self-supervision demonstration.

More sophisticated \( P_{\text{approx}}(y_1'|y_1, y_2) \) yield the same ranking when we compare the distortions that occur for each of the alternatives in Table I. Table II lists the values of \( \pi_2 \) and \( \pi_1 \) that we use in four separate numerical simulations.

If we are not using minimum distortion encoding, there is no need to refine the nearest neighbour encoding \( y^0 \) and so we jump to step 6.

4. Compare the expected reconstruction error. We may write the expected reconstruction error \( D(x) \) for the current input vector \( x \) as in Equation 3.3. We evaluate the integral over \( y_1' \) using the representation of \( P(y_1'|y_1, y_2) \) that we determined in step 3. An analogous result also holds for the integral over \( y_2' \).

5. Adjust the encoding to reduce the error. We must now investigate how \( D(x) \) varies in the vicinity of our initial guess \( (y_1^0, y_2^0) \) to locate the local minimum \( (y_1, y_2) \) of \( D(x) \), which in general is not equal to \( (y_1^0, y_2^0) \) (i.e. minimum distortion encoding is not the same as nearest neighbour encoding).

Note that, for each alternative value of \( (y_1, y_2) \) that we investigate, we must repeat step 4 and step 1 to determine the corresponding value of \( D(x) \), because the pair of neighbourhood functions \( P(y_1'|y_1, y_2) \) and \( P(y_2'|y_1, y_2) \) depend on \( (y_1, y_2) \). In our simulations we explore only the immediate neighbourhood of \( (y_1^0, y_2^0) \), as specified by \( y_1 \in \{ y_1^0, y_1^0 \pm 1 \} \), \( y_2 \in \{ y_2^0, y_2^0 \pm 1 \} \). Ideally we should perform an exhaustive search of all possible pairs \( (y_1, y_2) \) to find the one that yields the minimum distortion, but this is too costly.

6. Update the code vectors. The code vector update prescription is the scalar version of Equation 2.6. We do not gradually reduce \( \epsilon \) to zero as is usually the case, but instead use \( \epsilon = 0.1 \) throughout the optimisation.

We implement a refinement of Equation 2.6, in which we start with only two code vectors, which we then use to initialise a more refined optimisation using four code vectors, and so on. We optimise each generation of code vectors using 50 training vectors per code vector, before initialising the next generation. In our simulations we stop at eight code vectors. This ‘coarse to fine’ strategy is very effective, and rapidly produces an optimum set of code vectors. We discuss this whole training procedure in detail in [5].

7. Update the histogram. We update a leaky histogram representation of \( P(y_1, y_2) \) by both topping up and leaking away the contents of each bin of an \( 8 \times 8 \) bin histogram. In order to limit the computational cost, we top up the appropriate bin at every update step, but leak away all the bins (by multiplying by 0.5) on every 100th update step.

**B. Experimental results**

Figure 6 presents the results of several numerical simulations. Each result is the average of the value of the Euclidean distortion that we obtain from 16 independent simulations (in each simulation we accumulate statistics for 256 test set samples).

1. When we use a symmetrical neighbourhood function (i.e. entry number 1 in Table II) we obtain approximately the same distortion in all four cases.
We would expect NN/I = NN/C = MD/I = MD/C in the limit of a zero-width neighbourhood function, and NN/I = NN/C > MD/I = MD/C for a finite-width symmetrical neighbourhood function. The slight improvement that MD encoding gives compared to NN encoding shows that MD encoding yields measurable effects even for symmetrical neighbourhood functions.  

2. When we use a biased neighbourhood function, we find that MD encoding systematically produces a smaller distortion than the corresponding NN encoding. We would expect this result because MD anticipates the effect (on average) of the channel distortion, whereas NN does not.  

3. When we compare the different channel modes, we find that the C channels systematically produce a smaller distortion than the corresponding I channels. We would expect this, because correlated noise causes less damage than uncorrelated noise.  

4. When we compare the two different degrees of correlation, we find that the C channels systematically produce a smaller distortion for the higher degree of correlation (i.e. the smaller value of θ). The I channels do not distinguish between the different degrees of correlation.  

This behaviour demonstrates that using neighbourhood functions \( P(y'_1|y_1,y_2) \) and \( P(y'_2|y_1,y_2) \) that depend on the pair of outputs \((y_1,y_2)\), is better than using \( P(y'_1|y_1) \) and \( P(y'_2|y_2) \) that depend on only one output; this result is true whether we use NN or MD encoding. The cooperative behaviour that \( P(y'_1|y_1,y_2) \) and \( P(y'_2|y_1,y_2) \) induce can be described in a number of ways, but we prefer to use the multilayer unsupervised network viewpoint, and thus describe it as ‘self-supervision’.  

IV. CONCLUSIONS  

The main purpose of this paper is to explain how self-supervision arises in multilayer networks. Thus we consider the problem of minimising the distortion (between input and output) that occurs when a pair of communication channels mutually interfere with each other. This system is equivalent to an unsupervised winner-take-all neural network. We demonstrate how a set of neighbourhood functions (for ordering the vector quantiser codebooks) emerges from the analysis. Furthermore, we observe that each channel influences the other’s neighbourhood function - an effect that we call self-supervision.  

Self-supervision of multilayer unsupervised networks is important, because different parts of such a network cooperate to process the input data in a coordinated fashion, without the need to supply an external teacher. In general, an unsupervised multilayer network supervises its own internal operation by passing control signals back from the higher layers to lower layers, which in turn cause the lower layers to process their inputs more effectively.  

It is also possible to construct hybrid multilayer networks in which both external-supervision and self-supervision terms contribute. The external-supervision terms arise from the choice of the supervised contribution to the network cost function (e.g. minimum average squared output error), whereas the self-supervision terms arise from the choice of the unsupervised contribution to the network cost function (e.g. minimum average squared input reconstruction error).

Appendix A  

In this appendix we present an analytically solvable model of the neighbourhood functions that arise in the vector quantiser system that we discuss in Figure 4 where we use \( P(y'_1,y'_2|y_1,y_2) \) to describe the effect of a second stage of vector quantisation. We use \( y \) to denote \((y_1,y_2)\) and \( y' \) to denote \((y'_1,y'_2)\); we refer to \( P(y'_1,y'_2|y_1,y_2) \) as a ‘transition probability’, and we use \( \rho(y_1,y_2) \) to denote the density of code vectors of the second stage of vector quantisation. Note that \( \rho(y_1,y_2) \) contains no information about correlations between the positions of the code vectors; it models the average properties of an ensemble of optimised second-stage vector quantisers, assuming that each has a large codebook.  

We may write down an integral equation that relates \( P(y'|y) \) to \( \rho(y) \) as

\[
P(y'|y)\delta y' = \left(1 - \int_{||w-y'|| \leq ||y'-y||} dw P(w|y) \right) \rho(y')\delta y' \tag{A1}
\]

The first term on the right-hand side of Equation A1 is the probability that there is no nearest neighbour code vector within the sphere of radius \( ||y'-y|| \) centred on \( y \), and the second term is the probability of finding a code vector in the volume \( \delta y' \) located at \( y' \). The product of these two terms gives the probability of finding the nearest neighbour code vector in the volume \( \delta y' \) at \( y' \). We now solve Equation A1 for the case \( \rho(y) = \rho_0 \) is constant. The nearest neighbour code vector is then equally likely to lie in any direction from \( y \), so \( P(y'|y) \) must be a radial distribution function, which we shall denote as \( P(||y'-y||) \), which depends only on radial distance \( ||y'-y|| \), and which therefore satisfies the integral equation

\[
P(||y'-y||) = \left(1 - \int_{||w-y'|| \leq ||y'-y||} dw P(||w-y||) \right) \rho_0 \tag{A2}
\]

The integrand is spherically symmetrical, and so we may use the transformation
\[
\int_{\|w-y\| \leq \|y'-y\|} dw P(\|w-y\|) = \alpha_N \int_{\|w\| \leq \|y'-y\|} d\|w\| \|w\|^{N-1} P(\|w\|)
\]  

(A3)

where \(\alpha_N\) is a constant deriving from the angular integration in \(N\) dimensions (\(\alpha_2 = 2\pi\) for \(N = 2\)). Now differentiate Equation A2 with respect to the upper limit \(\|y'-y\|\) of the \(\|w\|\) integration to yield

\[
\frac{dP(\|y'-y\|)}{d\|y'-y\|} = -\alpha_N \rho_0 \|y'-y\|^{N-1} P(\|y'-y\|) 
\]

(A4)

and finally integrate to yield

\[
P(\|y'-y\|) = P_0 \exp(-\frac{\alpha_N \rho_0 \|y'-y\|^{N}}{N}) 
\]

(A5)

where we must adjust the integration constant \(P_0\) in order to normalise \(P(\|y'-y\|)\). The \(N = 2\) case reduces to a Gaussian distribution with \(P_0 = \rho_0\).

We now extend the previous results to the case

\[
\rho(y') \approx \rho(y) + (y'-y)^T \nabla \rho(y) 
\]

(A6)

which is a first-order Taylor expansion of \(\rho(y')\) about the point \(y' = y\). Throughout this section we use approximation signs to indicate where we omit second- (and higher)-order terms in Taylor expansions. We anticipate that the first-order expansion of \(P(y'|y)\) has the form of Equation A3 with an extra factor to account for the angular dependence in Equation A6. Thus we approximate \(P(y'|y)\) as

\[
P(y'|y) \approx P_0(y) \left(1 + (y'-y)^T . a(y)\right) \exp(-\frac{\alpha_N \rho(y) \|y'-y\|^N}{N}) 
\]

(A7)

where we have yet to determine \(a(y)\).

To solve for \(P(y'|y)\) we must first differentiate Equation A1 with respect to \(y'\) to obtain

\[
\frac{\partial P(y'|y)}{\partial y'} = -\left(\frac{\partial}{\partial y'} \int_{\|w-y\| \leq \|y'-y\|} dw P(w|y)\right) \rho(y') + \frac{P(y'|y) \partial \rho(y')}{\rho(y')} \frac{\partial}{\partial y'} 
\]

(A8)

The first term depends on the derivative with respect to the upper limit of the \(w\) integral, and the second term depends directly on the gradient of the code vector density.

To simplify the three terms that appear in Equation A8 we need the following three results:

\[
\frac{\partial P(y'|y)}{\partial y'} \approx \frac{\partial}{\partial y'} \left(P_0(y) \left(1 + (y'-y)^T . a(y)\right) \exp\left(-\frac{\alpha_N \rho(y) \|y'-y\|^N}{N}\right)\right) 
\]

(A9)

\[
\frac{\partial}{\partial y'} \int_{\|w-y\| \leq \|y'-y\|} dw P(w|y) = \frac{\|y'-y\|^N}{\|w-y\|^N} \int_{\|w-y\| \leq \|y'-y\|} dSP(w|y) 
\]

(A10)

\[
\frac{\partial \rho(y')}{\partial y'} = \nabla \rho(y') 
\]

(A11)

We use these three results to simplify Equation A8 into the form

\[
a(y) - \alpha_N \rho(y) \|y'-y\|^{N-2} (y'-y) \left(1 + (y'-y)^T . a(y)\right) \approx -\alpha_N \|y'-y\|^{N-2} (y'-y) + \left(1 + (y'-y)^T . a(y)\right) \nabla \rho(y') 
\]

(A12)
We then use $\rho'(y) \approx \rho(y)(1 + (y' - y)^T \nabla \rho(y) / \rho(y))$ and $\nabla \rho'(y) \approx \nabla \rho(y) / \rho(y)$ to simplify this result into the form

$$
\left( a(y) - \frac{\nabla \rho(y)}{\rho(y)} \right) \approx \alpha N \rho(y) \| y' - y \|^N (y' - y)^T \left( a(y) - \frac{\nabla \rho(y)}{\rho(y)} \right).
$$

(A13)

where we omit second (and higher)-order terms.

Finally, we solve this equation by choosing $a(y) = \frac{\nabla \rho(y)}{\rho(y)}$, and so we obtain the generalisation of Equation A5 as

$$
P(y'|y) \approx P_0(y)[1 + \frac{(y' - y)^T \nabla \rho(y)}{\rho(y)}] \exp\left(-\frac{\alpha N \rho(y)}{N} \| y' - y \|^N\right) 
\propto \rho(y') \exp\left(-\frac{\alpha N \rho(y)}{N} \| y' - y \|^N\right).
$$

(A14)