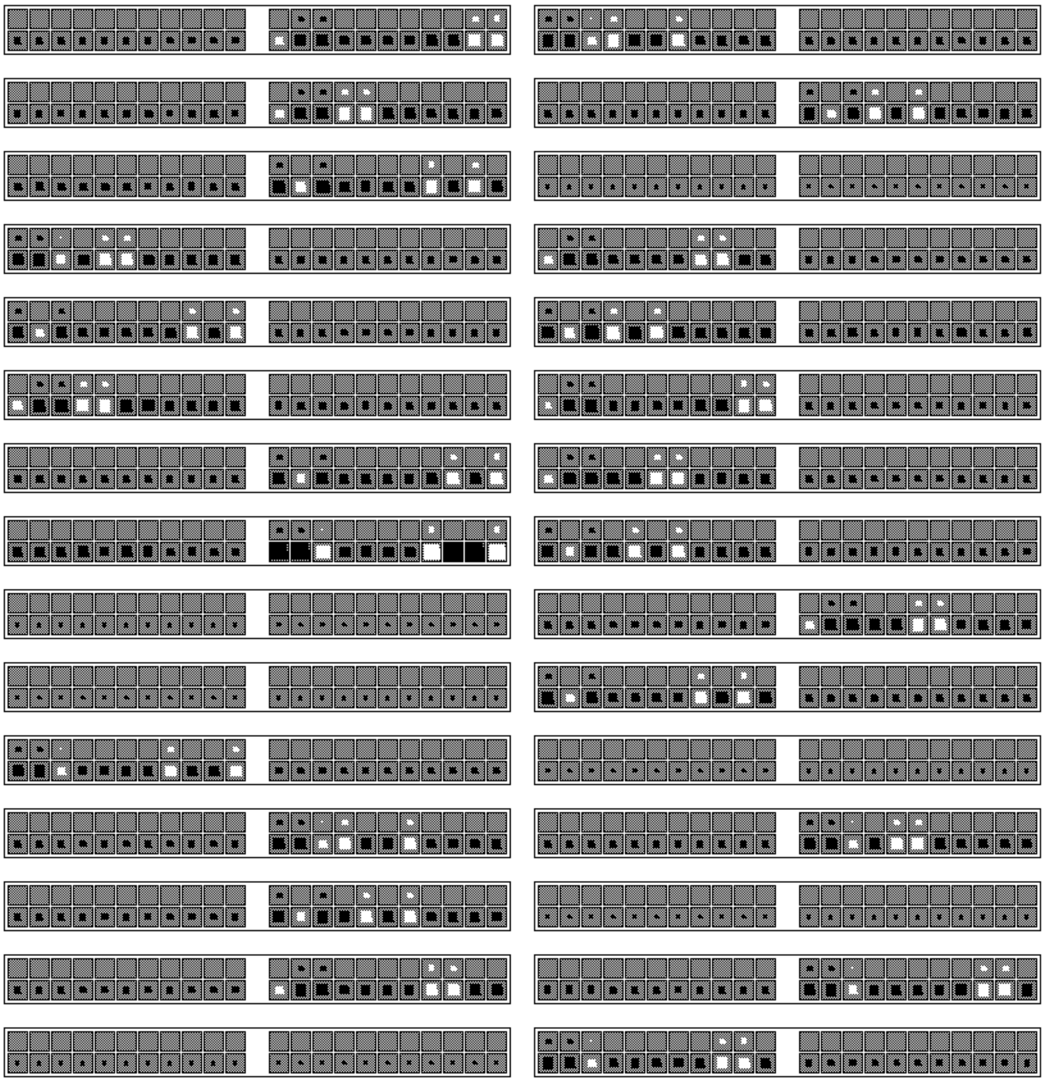

Figure 4.6: This figure shows the incoming and outgoing weights for the 30 representation units in a network trained on the dual-component dataset. For each unit, the input weights are shown below the output weights, and both sets are split in half, corresponding to the two 11-bit halves of the input vector. These units are initially fully connected, however, and know nothing about this structure. Note that four units are inactive for all inputs, and have null outgoing weights. Each of the other 26 units correspond to one of the 13 11-bit vectors that occur on one half of the input or the other.

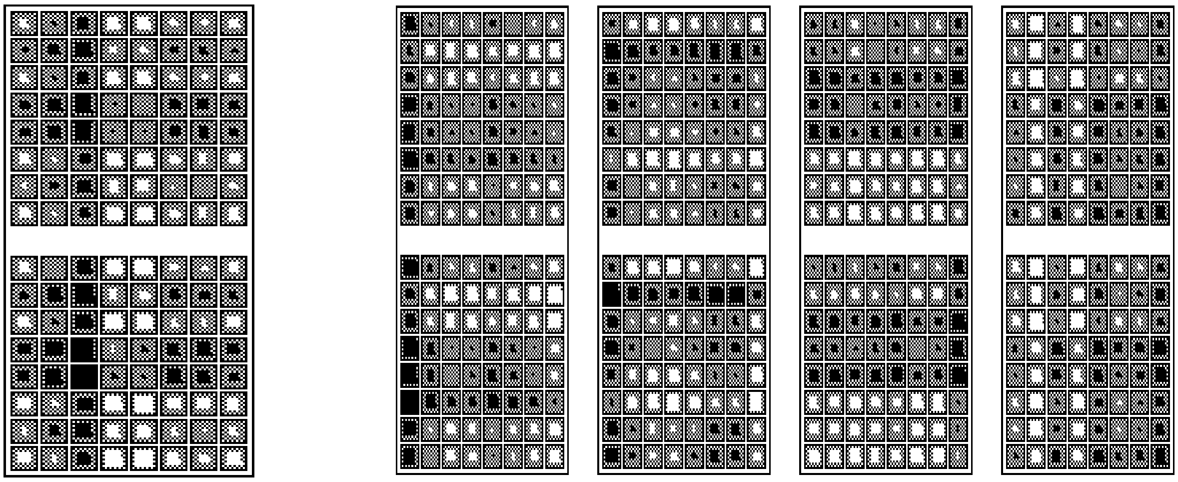

Figure 4.7: These figures show the incoming and outgoing weights for two linear autoencoders trained on a horizontal/vertical lines dataset. The left side shows the weights for a network containing one representation unit. The weights on the bottom half represent the principal component of the dataset. The second network contained four representation units; the four weight sets on the lower right-hand side span the subspace of the first four principal components.

where the M representation units span a subspace of the first M principal components, we trained such a network on the datasets described above. Figure 4.7 shows the principal component, and some rotated subspace of the first four components, for the larger enumerated lines training set. In addition, for the sake of comparison to our algorithm, we also trained a PCA network with 24 representation units on this dataset. The mean reconstruction cost on the test set this network was 0.17 bits. We also trained an autoencoder with a non-linear representation layer on the same dataset. This network reduced the redundancy in the input representation from $R_{IN} = 5.00$ to $R_{REP} = 1.19$. Even though it captured some features of the lines, it failed to find the structure, as Figure 4.8 illustrates.

For the dual-component dataset, we trained a PCA-autoencoder with the same architecture as the one shown in Figure 4.6. Here again the network failed to find the true causes of the data. Figure 4.9 shows the weights for this network.

A kernel-method algorithm is more likely to find the underlying structure in these datasets, since the assumption underlying this class of algorithms is that each datum belongs to, or was generated by, a cluster-center. The key difference is that in these datasets there are multiple components of each input, so trying to find a single cluster for each example will miss this componential structure. This difference highlights one of the advantages of a sparse representation over a purely local one, in that the number of representation units can be combined combinatorially to represent a much larger dataset. Figure 4.10 shows that a competitive learning (see Section 3.2.3) algorithm fails to find the underlying components in the same horizontal/vertical lines dataset. While this method does well with respect to representational entropy, since each unit is infrequently active, it does not capture the information in the input. We could convert this network to an autoencoder by copying the incoming weights for a unit into its outgoing weights; such an autoencoder would clearly have a

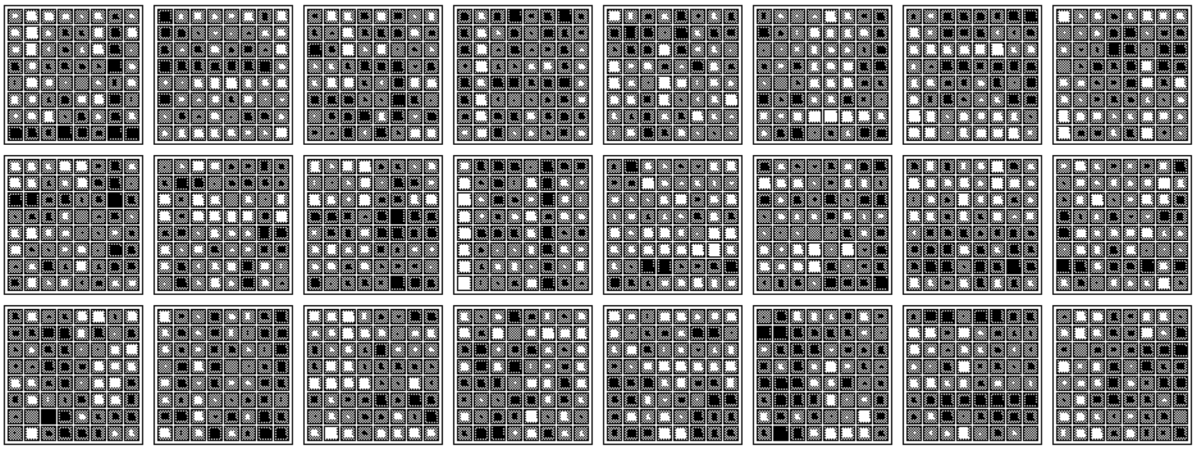

Figure 4.8: This figure shows the incoming weights for a linear autoencoder with 24 representation units trained on a horizontal/vertical lines dataset. The outgoing weights for each unit approximately equal the incoming weights (as in the previous figure).

high reconstruction error. In addition, the representation formed by this network does not separate the independent elements (i.e., the underlying features) of the input distribution, which will make later learning more difficult.

4.4 Discussion

The algorithm introduced in this chapter is capable of discovering the underlying features in a dataset, producing a non-redundant, highly informative representation of each input. The experimental results demonstrate that the algorithm can find the true causes of the input cases, finding as many causes as necessary to explain each input. These causes are captured in separate units, and then the few active units per input cooperate to reconstruct that case.

We have presented two variations of the basic algorithm. Other, more elaborate variations and extensions are also possible within this same framework. For instance, we could adopt a representational scheme where sets of units would have similar response frequencies. The corresponding encoding scheme for the representation activities could involve an adaptive mixture of base probabilities, where each representation unit would be encoded relative to an adaptive base probability for a group of units. This scheme resembles the adaptive fitting of a mixture of Gaussians to the weights in a network (Nowlan and Hinton, 1992) (see Section 2.4.3). This would allow the algorithm to excise truly extra representation units from the network, by grouping these units and allowing their base probability to go to zero.

Our algorithm bears several similarities to that of Nowlan (1988). His scheme also used an autoencoder with a layer of stochastic binary representation units. The objective was to minimize the entropy of the distribution represented in these representation units, while simultaneously minimizing the reconstruction error, estimated using the mean-field approximation (see below). An additional cost term encouraged the representation units to have equal entropy. The combined effect of these costs was to encourage the representation units to correspond to non-redundant features of the input,

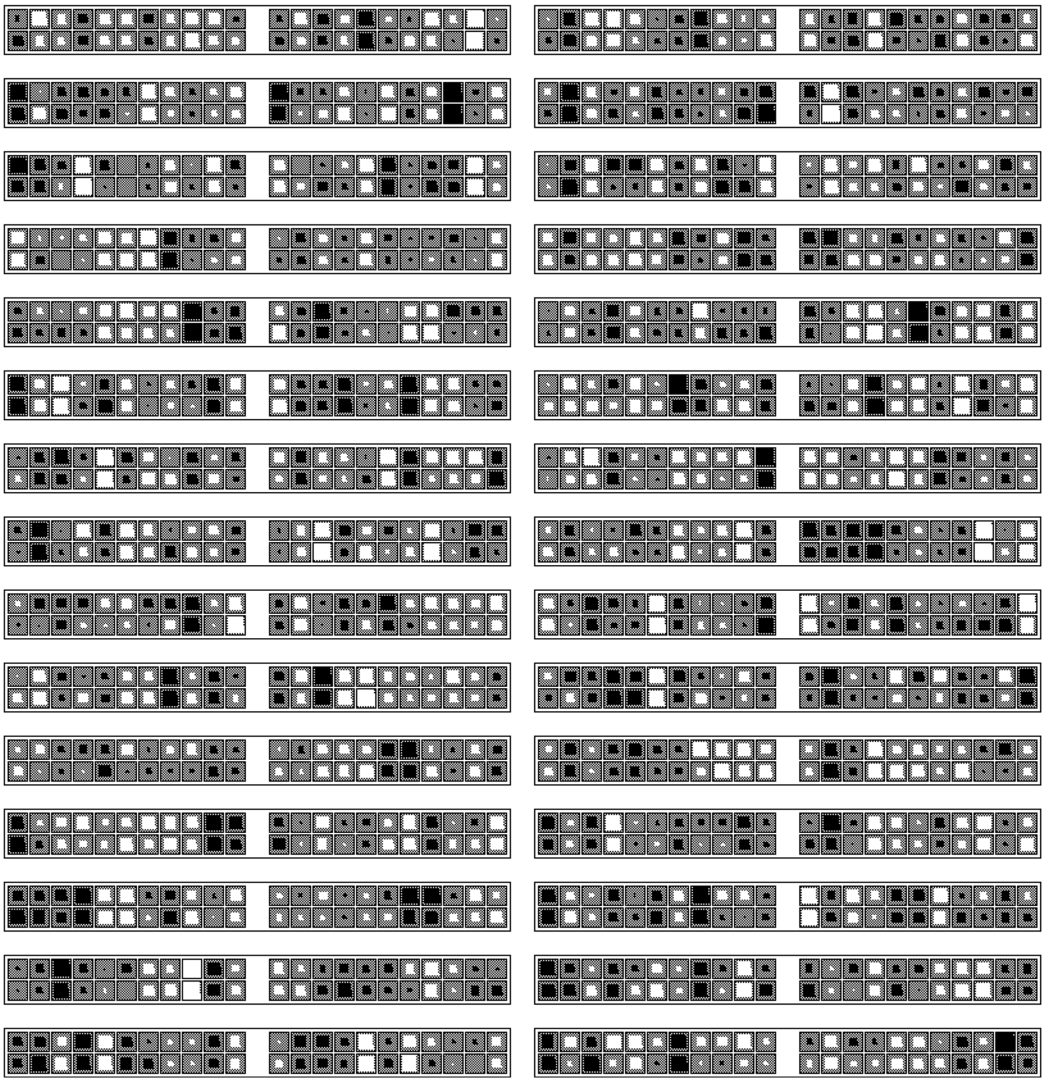

Figure 4.9: This figure shows the incoming and outgoing weights for the 30 representation units in a PCA network trained on the dual-component dataset.

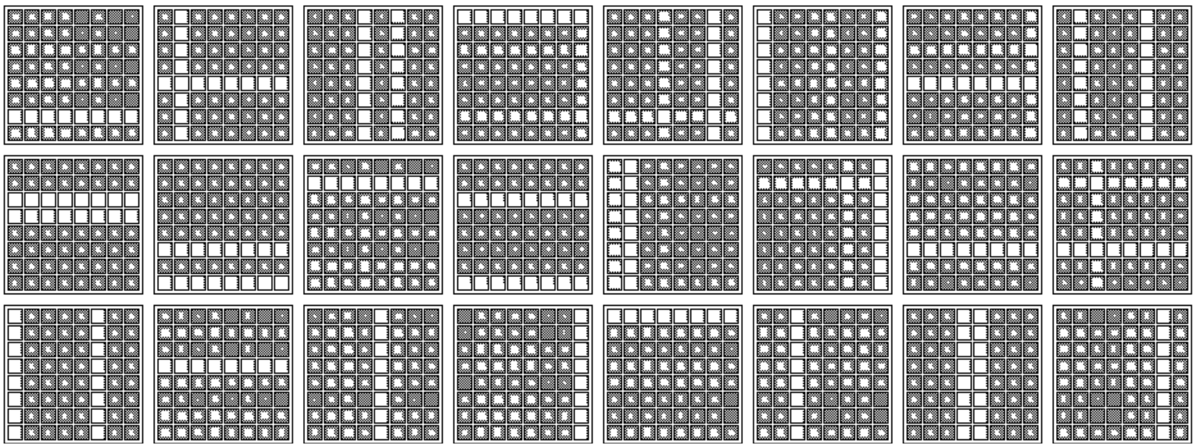

Figure 4.10: This figure shows the incoming weights for a competitive learning network with 24 output units trained on a horizontal/vertical lines dataset.

producing distributed representations that were easy to decode.

Despite their similarities, several significant differences exist between our algorithm and this earlier one. First, we introduce the idea of coding the representation activities relative to a prior distribution. The cost then involves asymmetric-divergence between a distribution based on a unit's activity on a particular case and a cross-case distribution; this encourages units to maintain some baseline level of activity, as determined by a single parameter, while discouraging the degenerate solution in which all the units are always off. The Nowlan scheme did not discover such sparse distributed representations as readily; without any "cross-case" pressure on a representation unit's activity, the network tended to fall into a solution in which nearly all units were eliminated, and the remainder were on half the time. Also, in our algorithm, units can be neatly excised from the network simply by making their activities always conform to the prior. This allows the network to determine the appropriate number of representation units required for the dataset, as the experiments described above demonstrate. In addition, by constructing an exact model of the stochastic system, the minimum-entropy term drops out in our scheme as a by-product of minimizing the reconstruction-error: entropy indicates variance in the representation which produces variance in the output and higher reconstruction error. Our scheme thus encourages the network to find a single binary representation for each input.

We are currently exploring several extensions to this algorithm. One important extension would generalize the algorithm to run on architectures other than a 3-layer network with linear output units. For this network, the output is simply the sum of binomial variables, so the calculation of the mean and variance of the output is straightforward and exact. In more complex networks, taking into account the stochastic nature of unit activities is more difficult. For example, if the output units of a 3-layer network are sigmoidal, then the output is a nonlinear function of a random variable, and the reconstruction error cannot be directly computed based on a simple sum of binomial variables. Instead, either Monte Carlo sampling or an approximation must be used to estimate the values of these stochastic output variables, and the expected error.

We have explored applying an approximation known as the *mean-field approximation*, which states that once the system has reached equilibrium, the stochastic variables can be approximated by their mean values. In this approximation, the variables are treated as independent, and the system probability distribution is simply the product of the probability distributions for the individual units.[4] This approximation is only strictly valid in the limit of an infinite number of degrees of freedom. Yet it has been shown (Galland, 1992) to be empirically valid in shallow networks (i.e., those containing few layers).

A standard mean-field network (Peterson and Anderson, 1987; Hinton, 1989b) approximates each unit's probability distribution by its mean. Our scheme represents an elaboration to this approximation, where we not only take into account the means of the unit's distributions but also their variances. For instance, in the example given above, we can approximate the net input to each sigmoidal output unit as a normal random variable with a known mean and variance (computed as in Section 4.2.2). The distribution for an output unit can then be described as a normal distribution

[4] The trade-off for the increase in computational speed of the mean-field approximation is a a loss of information concerning the correlations between units' states, and a decrease in the number of states that can be represented, since only N real values are used to represent distributions over 2^N binary states.

put through a nonlinear transfer function. We can construct a table for the reconstruction error based on various values of the mean and variance of the input to a unit. Preliminary experiments with such a scheme show promising results. A significant contribution of this scheme is that we are therefore maintaining and utilizing more information about the probability distribution of values in the network, thereby improving the quality of the mean-field approximation. It would be interesting to examine whether incorporating this additional information could improve the performance of other mean-field networks.

Chapter 5

Cooperative Vector Quantization

A key assumption underlying the algorithm described in the previous chapter is that each input example is the product of the combination of basically independent processes. For example, a visual scene can be described as the combination of several separate objects in some spatial configuration, under some source of illumination. A representation for such scenes can be formulated from a vocabulary of independent elements, where a few are relevant to a particular scene and cooperate to reconstruct it. A significant aspect here is that the input is assumed to have been caused not by a single hidden cause, as is typically assumed in clustering algorithms, but rather by some combination of many hidden causes. The goal of the system is to develop a representation that captures these multiple hidden causes.

In this chapter, we present a new learning algorithm that extends some of these primary ideas from the previous chapter. The next section introduces the *Cooperative Vector Quantization* (CVQ) algorithm, and describes the underlying representational scheme and its motivations. The following section describes the network architecture and activation functions. We derive the objective function for this network from the MDL framework, and analyze the resulting terms in this function. We then discuss the results of applying this algorithm to two datasets, including one that was introduced in Chapter 4. Finally, we conclude by relating this algorithm to other approaches.

5.1 Introduction

In the algorithm described in the previous chapter, each representation unit comes to represent a feature, or cause, and the few active units correspond to causes of the particular input case. In many situations, however, there may be more structure in the data. The set of hidden features may not consist of one large pool, from which a few are relevant to each case. Instead, each cause may correspond to one possibility selected from a set, or category of features. For example, a car can be represented as the composition of elements drawn from several categories, such as color, engine size, upholstery style. Each of these categories contains a finite number of mutually-exclusive possible values; some may have only two elements (e.g., air-conditioning or not), but others have more (e.g., number of doors). The entire input is then formed as the combination of these independent features.

The representational scheme for this algorithm therefore involves several cooperating features, where these features are alternatives within categories. Each category can be modeled as a separate cluster, or vector quantizer (VQ). This points out a clear relationship between this algorithm and standard competitive learning/vector quantization, in which there is one winner per cluster. Here instead we have multiple VQs, with one winner per VQ, and these winners then cooperate to reconstruct the input.

This representational scheme shares many of the advantages of the scheme described in the previous chapter. This system also produces a sparse distributed representation for each datum. The elements within the representation are non-redundant, as the categories are intended to correspond to independent aspects of the dataset. An additional advantage of this representation is its combinatorial nature. While a standard VQ containing N elements can represent at most N items, if we divide the N into k-element VQs, we can represent $k^{N/k}$ items. Finally, in some cases, i.e., when the input set truly has componential structure, this representational scheme can lead to more compact messages. Rather than having to code the values of a large set of binary hypotheses for each input,

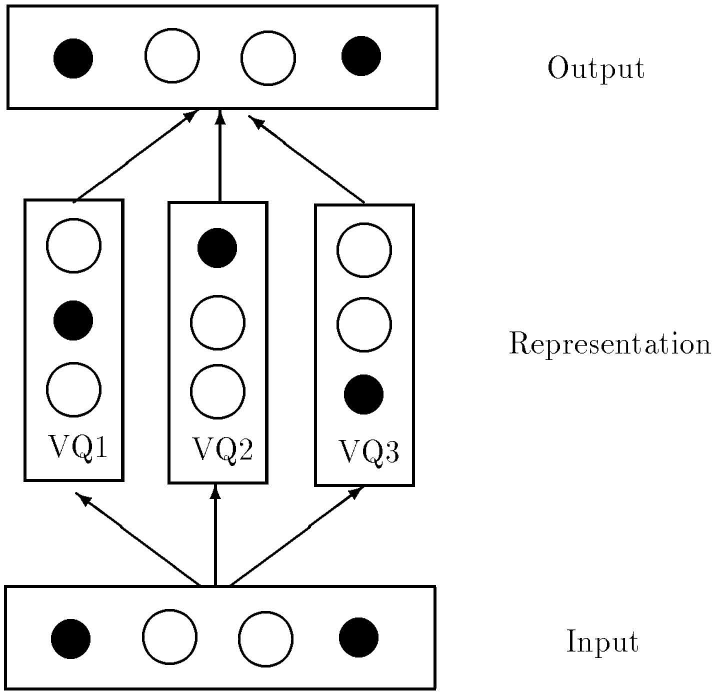

Figure 5.1: The network architecture for the Cooperative Vector Quantization algorithm. The input is a 4-element binary vector, with 2 active units. Within each of the three vector quantizers that comprise the representation layer, one unit is active per input case. These units then cooperate to reconstruct the input.

the new scheme can be seen as grouping some of the hypotheses, so that we only have to specify which one of the set applies to the particular input.

5.2 CVQ network architecture and activation functions

In the CVQ algorithm, we form a representation of each input vector by making 1-of-N choices in several different groups of units. Each group behaves like a vector quantizer. The network architecture is shown in Figure 5.1.

The activation function for units in a standard VQ is a competitive winner-take-all function, where the unit with the largest input has an activity of one, and the others all have zero activity. In our algorithm, however, we make a modification similar to what was done in the previous chapter. We form a probability distribution over the elements of the VQ, where the activity of a representation unit describes the expected value of that unit over the ensemble of representations for a particular input.

This is accomplished by the "softmax" activation function (Bridle, 1990; Nowlan, 1990), where the activity of representation unit j in VQ v on case c depends on its own net input, $x_j^c = \sum_i w_{ji} y_i^c$, as well as the net input to every other unit k in VQ v:

$$< S_j^c > = y_j^c = e^{x_j^c} / \sum_{k \in v} e^{x_k^c} \tag{5.1}$$

where S_j^c is the random binary value of representation unit j on case c. The unit's activity y_j^c is then the expected activity of unit j over the distribution of these binary 1-of-N representations for case c.

Given the architecture and activation functions in CVQ, we can view the algorithm described in Chapter 4 as a special case of CVQ. We associate a separate 2-state VQ with each binary unit where the first state, corresponding to the unit being active, has an input of $x/2$, while the second state, corresponding to the unit being inactive, has an input of $-x/2$ and a fixed output of zero. The softmax activation function now acts like a sigmoid, so the probability that the unit is active (has state one) on a given input equals:

$$exp(x/2)/(exp(x/2) + exp(-x/2)) = 1/(1 + exp(-x)) = \sigma(x)$$

The active units then cooperate to reconstruct the input. Adopting this view reveals that CVQ allows for a wider variety of structure within a VQ, where several alternatives may compete to participate in the representation of a given input.

5.3 CVQ objective function

5.3.1 Sending noisy representations

Based on the activation functions described in the previous section, the representations formed within each VQ are "noisy", i.e., there is not a single representation for each input, but rather a probability distribution over representations. The activity value of each unit reflects the degree of belief that that unit participates in the cooperative representation of the input. These values are constrained to sum to one in each VQ, so that each input is represented by a set of active representation units.

As in Chapter 4, these noisy representations serve two purposes. First, the representation units take on continuous values, which eases the search over possible binary representations for each training case, as it makes the derivatives smooth. If the algorithm instead selected a single representation for each input, switching to a different representation could lead to a sudden change in error or error derivatives. The second purpose is that the stochasticity can decrease the coding cost for the representation activities. The explanation for this decreased cost was given in the previous chapter (see Section 4.2.1).

The representations of the activities are noisy, so the activity cost $C_{\mathcal{A}}$ again involves the asymmetric divergence G between two distributions. In the previous chapter, the activity cost depended on the asymmetric divergence between two binomial distributions: the prior distribution, where the activity probability equaled the base rate b, and a posterior distribution, where the activity probability equaled the unit's activity on a single example. All training cases and representation units were considered independently, so this divergence was summed over the cases and units.

Here, each VQ v is considered independently. We begin with the simplest case, where the prior distribution P_v is a uniform distribution over the units in v, so the probability that unit j in VQ v is active is $p_v = 1/|v|$. The posterior distribution Q_j^c is the probability distribution over these units j for a particular example c, as determined by the softmax activation function (Equation 5.1). The activity cost is then:

$$C_{\mathcal{A}} = \sum_{v,c} G(P_v, Q_v^c) = \sum_{v,c} \sum_{j \in v} y_j^c \log(y_j^c / p_v) \tag{5.2}$$

This cost is minimized if the probability distribution across the units on each case matches the prior distribution, which here is the maximum entropy distribution for each VQ. Given this prior, the cheapest code is the noisiest.

As in the previous chapter, we can adapt the prior distribution to match the statistics of the network on the training set. We now allow each representation unit to have its own *base rate* p_j, or prior probability that it will participate in representing an input example. If we find the p_j that minimizes $G(P_v, Q_j^c)$ where $P_v = \{p_j | j \in v\}$ and $Q_j^c = \{y_j^c\}$, it turns out that the optimal individual base rate is simply that unit's mean activity across the training set: $p_j = \overline{y_j}$. One obtains this simply by using Lagrange multipliers to minimize $C = -\sum_{c,j} y_j^c \log p_j$ with respect to p_j, subject to the constraint that $\sum_{j \in v} p_j = 1$. Given this new prior distribution P_v, the activity cost is now:

$$C_{\mathcal{A}} = \sum_{v,c} G(P_v, Q_v^c) = \sum_{v,c} \sum_{j \in v} y_j^c \log(y_j^c / \overline{y_j}) \tag{5.3}$$

5.3.2 The reconstruction cost

The output units in the networks we describe in this chapter are linear. Since the representation units are stochastic, we must again take into account this stochasticity when computing the reconstruction cost for the network. We use the same notation here, so the activity of unit k on case c is a random variable Y_k^c. One can think of this random variable taking on values by sampling from the stochastic representations in each VQ (i.e., choosing one unit per VQ v to be active based on the probability distribution Q_v^c), and then taking the weighted sum of these active units. The expected activity of this unit is:

$$y_k^c \;= < Y_k^c > = < \sum_j w_{kj} S_j^c > = \sum_j w_{kj} y_j^c \tag{5.4}$$

By the same derivation given in the previous chapter (Section 4.2.2), we find that the expected reconstruction cost is:

$$C_{\mathcal{R}} \;=\; \frac{1}{2V_{\mathcal{R}}} [\sum_{k,c} (y_k^c - d_k^c)^2 + V_k^c] \tag{5.5}$$

where $V_{\mathcal{R}}$ is the fixed variance of the Gaussian used to code the reconstruction error; V_k^c is the variance of unit k on case c; and d_k^c is the desired activity of that unit for that case.

In order to calculate the variance of an output unit, we consider the variance contribution from each independent VQ. The variance of unit k on case c, V_k^c, is the sum across the VQs of the variance of X_{kv}^c, the net input to unit k from VQ v:

$$V_k^c \;=\; \sum_v var(X_{kv}^c)$$

$$
\begin{aligned}
&= \sum_{v} < (X^c_{kv} - < X^c_{kv} >)^2 > \\
&= \sum_{v} < (X^c_{kv} - \sum_{m \in v} w_{km} y^c_m)^2 > \\
&= \sum_{v} \sum_{j \in v} y^c_j (w_{kj} - \sum_{m \in v} w_{km} y^c_m)^2 \qquad (5.6)
\end{aligned}
$$

Examining the variance terms inside the sum over the VQs v reveals that each can be minimized if either the outgoing weights from the units in VQ v are equal (or near zero), or the probability distribution across the units in the VQ has no variance, i.e., one unit has an activity of 1, and y^c_j for the other units is 0. If the former situation holds true, then that VQ cannot contribute much to reconstructing the inputs, since its output will be constant across the training set. The algorithm will be likely to effectively eliminate this VQ then, by making it have the same pattern of activity on each case, which will minimize the cost of coding its activity.

The algorithm cannot eliminate all VQs in this manner, as the network would then have constant outputs across the training set, and high reconstruction error. The algorithm can instead take the second option described above, where variance is minimized as the probability distribution for a VQ approaches a delta function. This reveals a familiar trade-off between the activity cost and the reconstruction cost. It is cheaper to communicate noisy representation activities, as a representation of a given input is free if its entropy matches the cross-case entropy (by Equation 5.3). Entropy in the representation activities, however, contributes to variance in the output units, thereby increasing the reconstruction cost. This variance increases as the outgoing weights from a VQ increase in magnitude.

The final cost function for the network is the sum of the reconstruction and activity costs:

$$
\begin{aligned}
C &= C_{\mathcal{A}} + C_{\mathcal{R}} \\
&= \sum_{v,c} \sum_{j \in v} y^c_j \log(y^c_j / \overline{y_j}) + \frac{1}{2V_{\mathcal{R}}} [\sum_{k,c} (y^c_k - d^c_k)^2 + V^c_k] \qquad (5.7)
\end{aligned}
$$

Empirically we find that in a typical run of the learning algorithm, the network starts off with lots of entropy in the representation, with a roughly uniform distribution across the units in each VQ on every case, and small outgoing weights to minimize the output variance. Then as the network gets better at reconstructing the input, these outgoing weights grow, and the variance is minimized by lowering the entropy of the representation activities. Ultimately, a few units responding to different aspects of the input are active and cooperate in the reconstruction.

5.4 Experimental results

We tested this algorithm on several sets of training examples. Here we present the results of experiments with two types of datasets. The first training set is the dual-component set introduced in Chapter 4. The second is a somewhat less artificial set, where each image contains a spline. The algorithm successfully captures the underlying structure in both cases.

In the experiments described below, the weights of the network were adjusted using a conjugate gradient training procedure. The objective function minimized by the procedure was the sum of the representation and activity cost (Equation 5.7). The only adjustable parameter in this objective function is $V_{\mathcal{R}}$, the variance of the Gaussian used to code the reconstruction error. We did not carefully optimize the values of this parameter, as we instead tried a few values and selected one that produced consistent performance; for all experiments below, $V_{\mathcal{R}} = 0.1$.

The other variables in these experiments concern the architecture of the network, and particularly, the number of VQs and units per VQ. As was the case for the algorithm discussed in the previous chapter, we will see that CVQ is able to find the appropriate architecture for the dataset, effectively eliminating excess units and entire VQs, by setting their outgoing weights to zero and adjusting their incoming weights so that they ignore the input and have a constant activity level across the training set.

We compare different architectures based on the description length of a trained network on a set of test examples. We calculate this measure by simply computing the objective (Equation 5.7), and dividing by the number of examples in the test set to find the expected number of bits to represent a novel input. This measure takes into account the generalization ability in terms of both the efficiency and informativeness of the representation.

5.4.1 Dual-component dataset

In the dual-component training set, each input is a binary vector consisting of two halves. Each half contains one of M possible binary vectors, yielding a total of M^2 inputs. In these experiments, the input is a 22-bit vector, and each 11-bit half has a vocabulary of 13 vectors. Each input vector has three bits set to 1.0 and the other eight are 0.0. The network is trained on all but one of the 169 pairings of these vectors.

The first network architecture we use for this problem contains two VQs, with 15 units in each one. This architecture is chosen to match the structure of the problem, as the two halves of the input each act as a VQ with 13 different possible values. The algorithm consistently learns the solution shown in Figure 5.2, where each VQ learns to pay attention to one half of the input, and 13 of the representation units in each comes to represent (i.e., respond to, and reconstruct on the output units) one of the 13 vectors. The other two units in each VQ are eliminated. For each input example, one unit in each VQ is nearly fully active ($y_j^c \approx 1.0$), while the others are generally inactive. The activities of the units in the network on the test example is shown in Figure 5.3. This network clearly generalizes from the training set, producing a reconstruction error of 0.00 on this test example. The representational redundancy of this network is also zero, as the entropy of the representation (the sum of the entropies of the two VQs) matches the true entropy of the input. The final description length of the training set, and this test example, is approximately 7.5 bits per input example: the reconstruction cost approaches zero, but the activities are not free, since the two active units on each case cost approximately $\log 13 = 3.7$ bits each.

The representational scheme of the CVQ algorithm matches the structure of the data in this case, so it learns a good, non-redundant representation. The algorithm presented in Chapter 4 was also able to find the underlying vocabulary in this training set, as it assigned a representation unit to each of the 13 11-bit vectors on either side, for a total of 26 useful representation units.

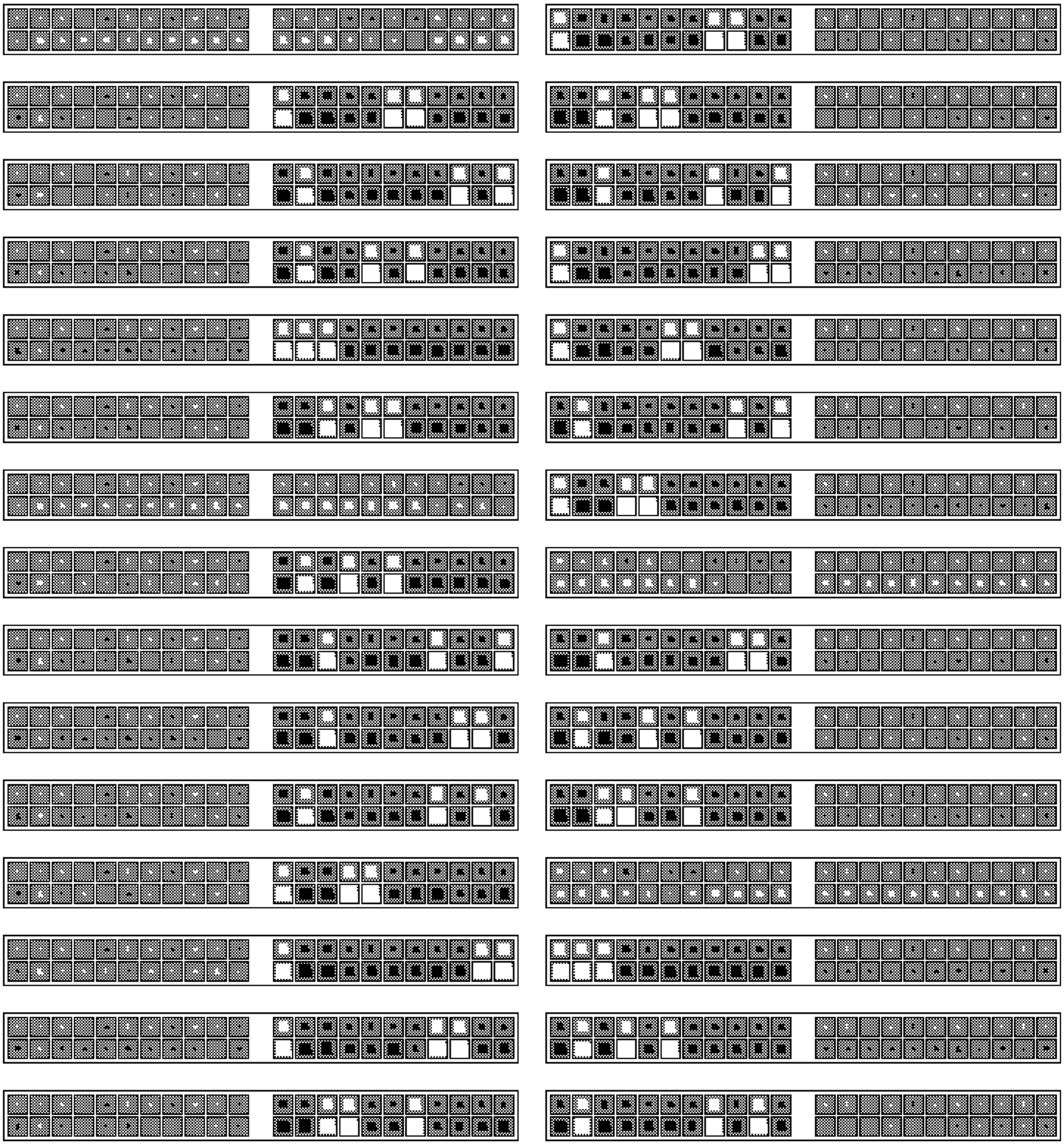

Figure 5.2: This figure shows the incoming and outgoing weights for a CVQ network trained on the dual-component task. The left column shows the weights (incoming below, outgoing above) for the 15 units in one VQ, and the right column shows the units' weights in the other VQ. The weights are split so that the two halves of the input are separated by a space; the network is not given this structure, as each representation unit begins with random, complete connectivity to the input and output, and learns to focus on one half or the other.

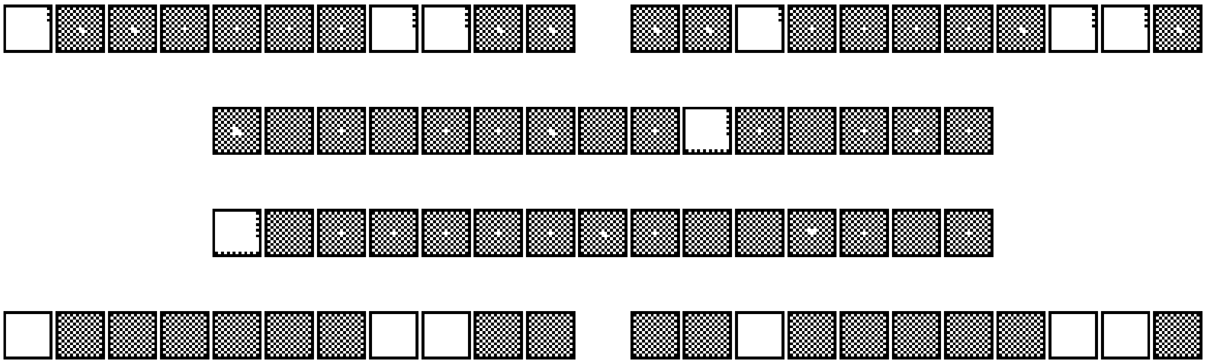

Figure 5.3: This figure shows the activities of units in a two 15-unit VQ network trained on the dual-component task. The bottom row shows the two halves of the input, the next two rows are the VQs, and the top row is the reconstructed output. The network generalizes well, as the output on this test case closely matches the input.

The extra representation units were effectively eliminated by the algorithm. But the redundancy of this representation was 0.51, as compared to 0.00 for CVQ. Since these representational units are considered independent in the former algorithm, the network fails to find the whole structure, i.e., that the 13 vectors are mutually exclusive on each half. In CVQ, bits need not be devoted to communicating the *inactive* units as well as the active ones, so the final description length is shorter.

We have also experimented on this dataset with other network architectures. The algorithm finds interesting solutions in each case. For example, if we add more units to each VQ, the network finds very similar solutions to that shown in Figure 5.2, as the extra units are eliminated. However, if the network contains only 10 units in each of the two VQs in the representation layer, the algorithm typically finds a solution where the VQs still specialize in one half of the input. Since there are not enough units in a VQ to devote one unit to each of the 13 vectors, some of the units end up responding to a pair of these vectors (see Figure 5.4).[1] The generalization length for this network is high (nearly 18 bits per input), primarily due to the higher reconstruction cost and remaining variance in the representation. This network architecture clearly does not contain enough capacity to represent this dataset.

With a single 30-unit VQ in the representation layer, the algorithm finds a solution similar to that found in the previous chapter, as 26 of the representation units respond to and reconstruct the components of the input vocabulary. The resulting representation has two active units per input, but since these units belong to the same VQ, the units have an activity of approximately 0.5, and this combined with the requisite higher outgoing weights leads to high variance. This network also has a high description length, of nearly 13 bits per input.

When the representation layer contains 3 VQs, the algorithm produces other interesting solutions. When the 3 VQs each consist of 10 units, the network divides the 26 vocabulary items up among the VQs in various ways. A typical solution finds 2 VQs specializing in alternate halves of the input; since the VQ cannot model all 13 possible values on that side, though, it devotes one unit to be

[1] This solution also occurs sometimes if the two VQs contain 13 or 14 units each. The network can get stuck in a local minimum, where some units become null units, while others respond to a pair of vectors. With 15 units per VQ, the network does not tend to get stuck in these local minima.

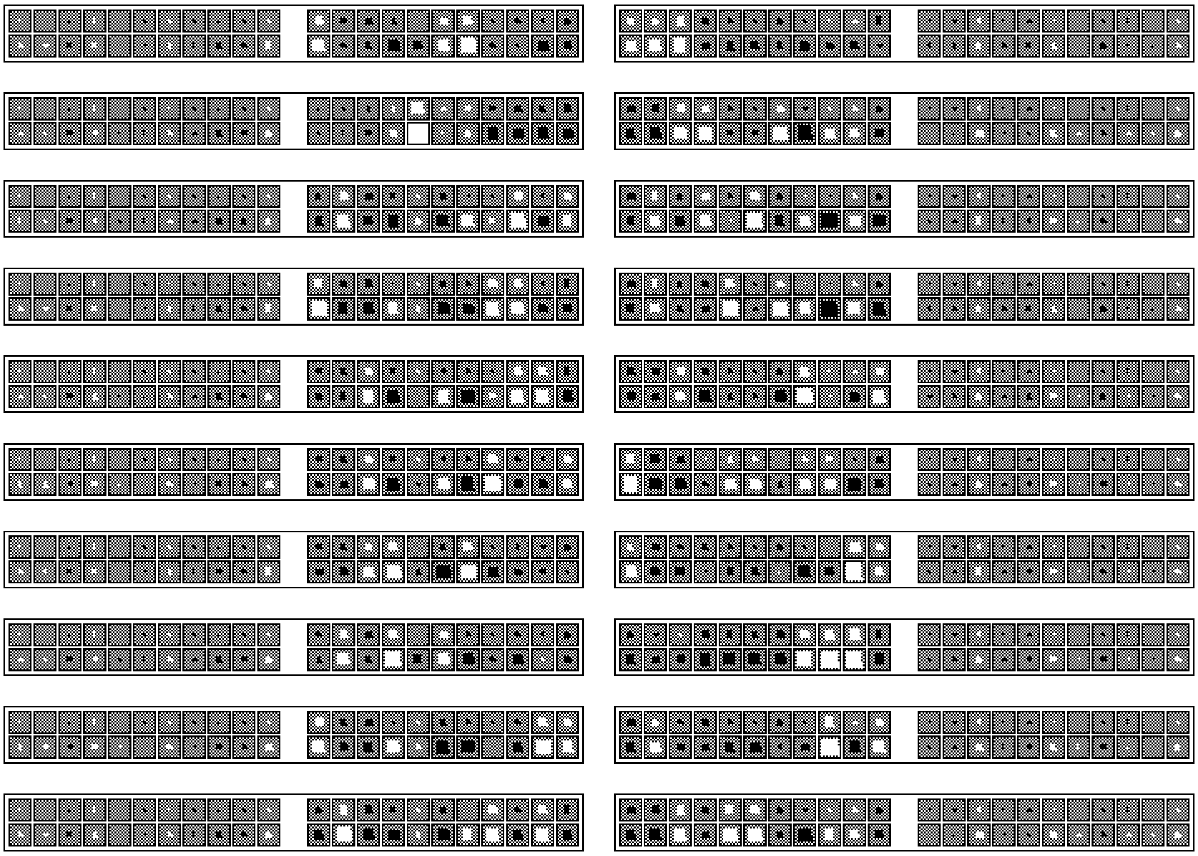

Figure 5.4: This figure shows the incoming and outgoing weights for a CVQ network with two 10-unit VQs in the representation layer trained on the dual-component task. In each of the VQs, some units clearly still respond to one of the 13 vectors, while others (those with more than 3 positive incoming weights) respond to a pair of these inputs. This representation gives this network a higher generalization length, and more redundancy than the network shown in Figure 5.2.

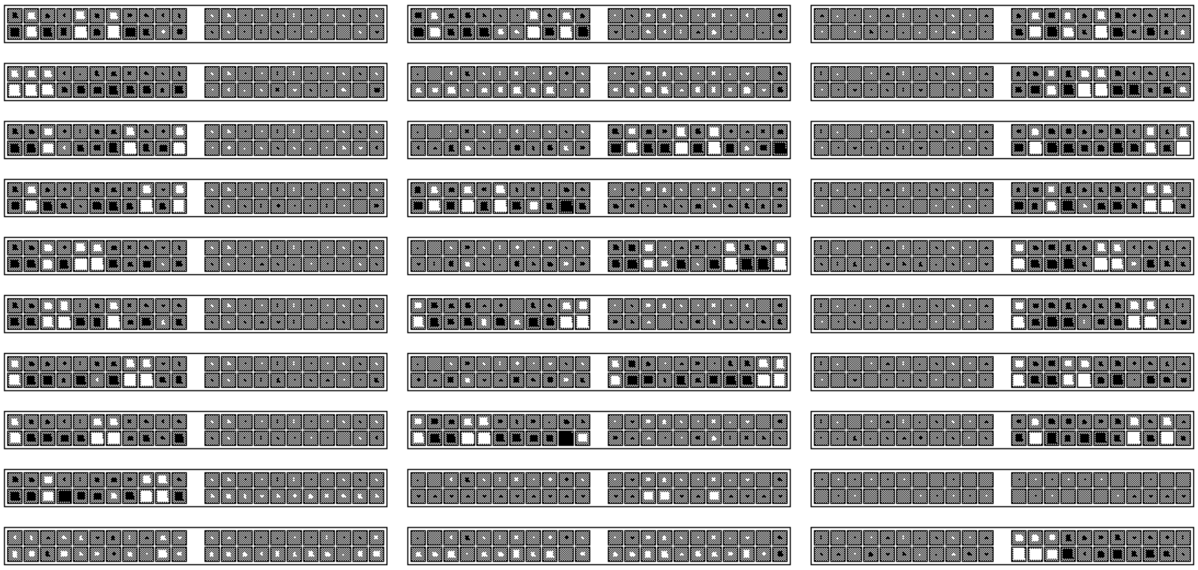

Figure 5.5: This figure shows the incoming and outgoing weights for a CVQ network with three 10-unit VQs in the representation layer trained on the dual-component task. The two VQs in the left and right column each specialize in one half of the input. Each of these contains one "none-of-the-above" unit (the last unit on the left, second-to-last on the right), which is active when the input does not match any of the vectors represented by the other nine units in that VQ. The middle VQ represents the remaining input vectors.

active when the others are not (see Figure 5.5). That is, if the input contains vectors not represented by the units in the VQ, then this unit is active, but its zero outgoing weights means that it does not contribute to the reconstruction. The network thus creates a *none-of-the-above* unit. This network has a description length of 12 bits per input.

Finally, if we expand the VQs in this network so that each contains 15 units, the algorithm finds the weights shown in Figure 5.6. Here one entire VQ is eliminated, as the other two VQs contain enough units to capture the structure of the input set. The network shown here has a description length of 8 bits, which is slightly higher than the first network (with 2 15-unit VQs). This difference can largely be explained by the residual entropy in the deleted VQ. When the algorithm finds an empirical global minimum of the cost function for this network, the resulting description length matches that of the first network.

5.4.2 Splines

The second dataset consists of 200 images of simple curves. Each image contains a single smooth curve, generated by fitting a spline to five randomly chosen control points. The control points have a fixed x-position, but a random y-position, so each image has only five degrees of freedom, even though it occupies many pixels (see Figure 5.7). The y-positions are chosen uniformly in the range of the image, within a buffer-zone so as to avoid edge effects. We fit a cubic B-spline to this sequence of 5 points, and then apply a Gaussian blurring function to create a grey-scale image of the curve. The test set contains 100 images generated in the same manner.

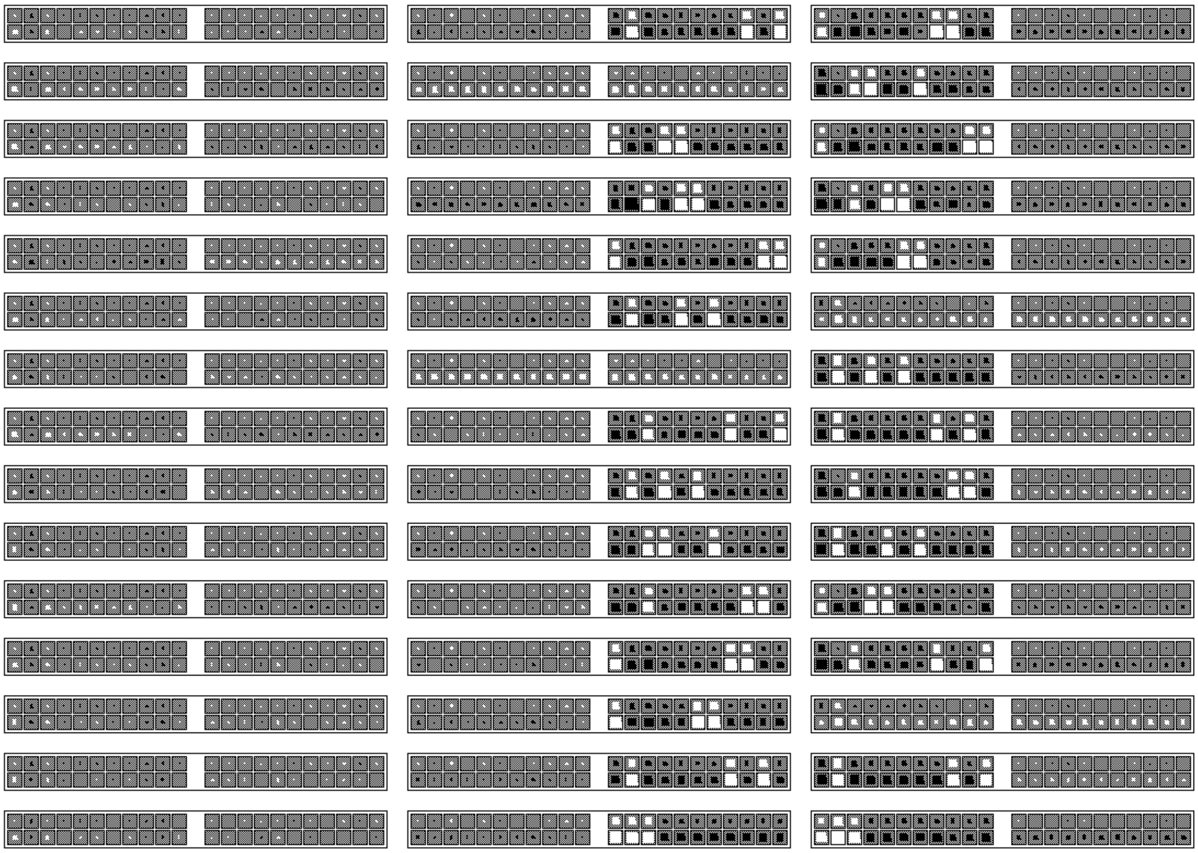

Figure 5.6: This figure shows the weights for a CVQ network with three 15-unit VQs. The left VQ is eliminated from the network: the near-zero outgoing weights means that this VQ does not contribute to output activities nor variances. The other two VQs learn to decompose the input into its two halves, with 13 of the units in each VQ representing one of the component features of the input. This solution is similar to that shown in Figure 5.2.

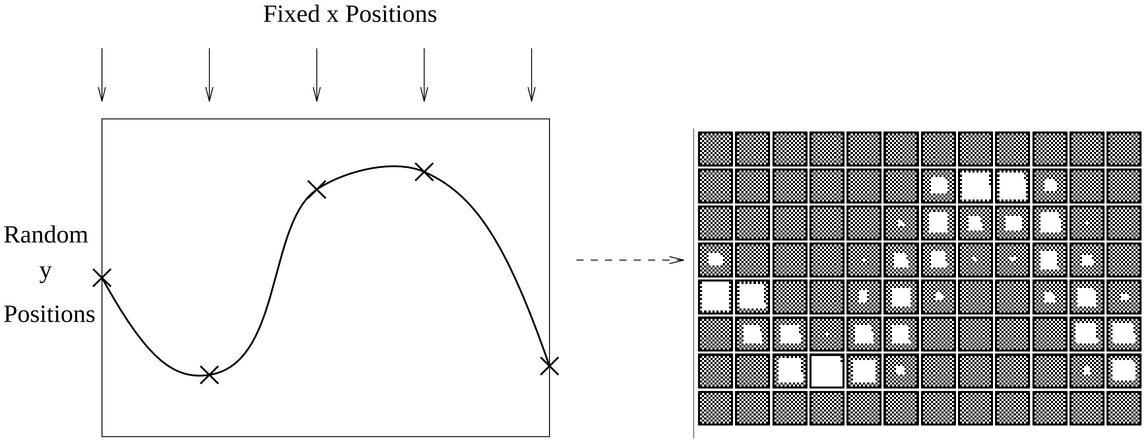

Figure 5.7: Each image in the spline dataset is generated by fitting a spline to 5 control points with randomly chosen y-positions. The resulting images are 8x12 pixels, but have only 5 underlying degrees of freedom.

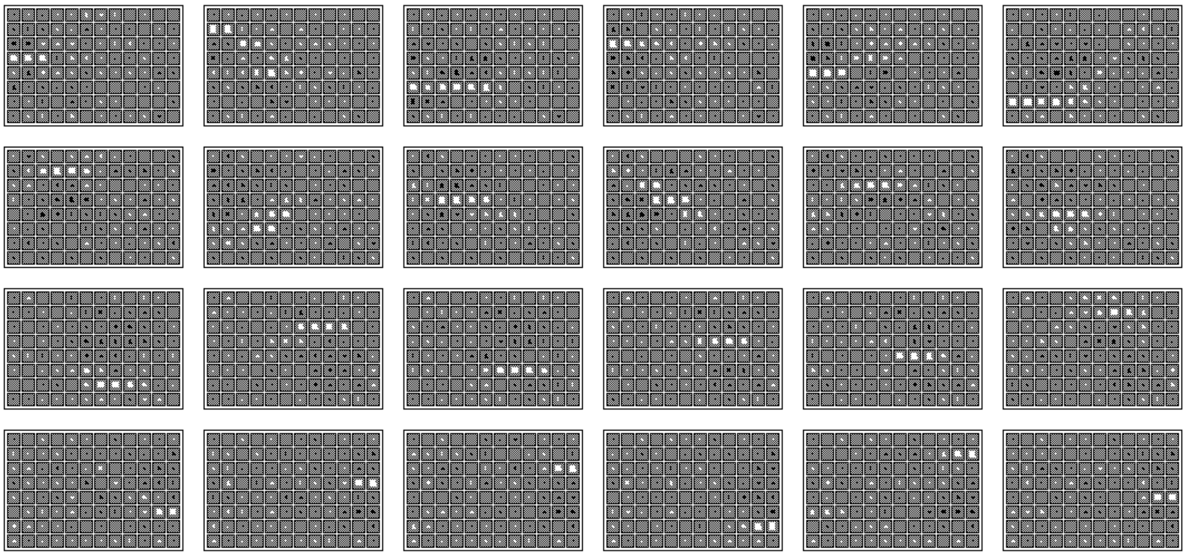

Figure 5.8: This figure shows the outgoing weights for a CVQ network containing 4 VQs with 6 units in each, trained on the spline dataset. Each 8x12 weight block corresponds to a single unit, and each row of these blocks corresponds to one VQ.

Each y-position is a random float between 0.0 and 1.0, represented to 8 bits of precision. We thus have a total of 40 bits in each input case. This is actually an upper bound, as the Gaussian blurring function together with the coarseness of the 8x12 image make it so that only a few bits of precision per point significantly affect each final image.

We have run the CVQ algorithm using several different architectures for this dataset. For the splines, it is not clear *a priori* what should be a good architecture. While we know that the dataset has 5 underlying degrees of freedom, the statistics of the data may lead to unexpected efficient representations.

One of the best architectures for this dataset in terms of the ultimate description length after learning turns out to be one with 4 VQs. Figure 5.8 shows the final outgoing weights for the representation units in a 4-VQ CVQ network with 6 units per VQ. Each VQ learns to represent the height of the spline segment connecting a pair of control points. Chaining together four of these segments then gives a piece-wise approximation to the entire curve. Figure 5.9 shows the unit activities in the network for two test examples. The final description length for this network is approximately 25 bits per image. The majority of this cost (18 bits) is due to the reconstruction term. The piece-wise approximation leaves some residual error, and often involves the combined activity of two units within a VQ, which leads to variance in the output. This is discussed further below.

Further examination of the weights diagram (Figure 5.8) reveals how the network has discovered some of the regularities of the dataset. If the first control point has a high y-value, then the next control point is likely to be lower, since the y-positions are chosen randomly and uniformly in the range of the image. The curve is therefore likely to descend from this first point, and vice versa for a low initial y-position. The second and last weight block in the first row of the weights diagram

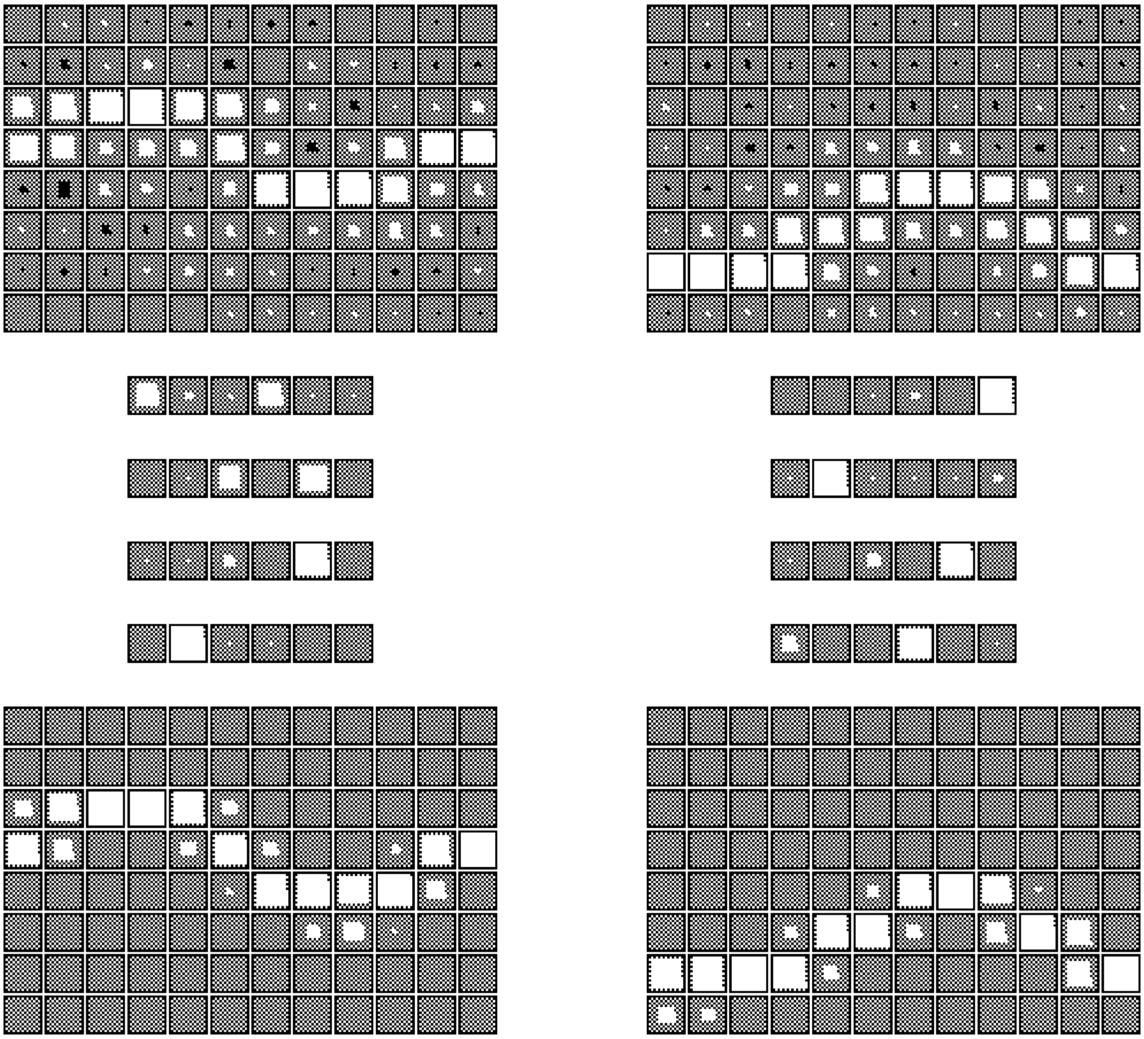

Figure 5.9: These figures show the activities of units in the CVQ network whose weights were shown in the previous figure. The two columns correspond to 2 different test examples, shown on the bottom. The next four rows show the activities in the VQs, and the final block is the network's reconstruction of the input.

capture these two regularities, respectively. Segments that start towards the center of the image are equally likely to rise as fall, so the weight pattern is more horizontal for the other units. A consequence of this is that diagonal segments are represented by the combined activity of the units that are sensitive to the relevant horizontal segments. An example of this can be seen in the second VQ in the left column of Figure 5.9.

If the VQs contain fewer than 6 units, the quality of the reconstruction suffers, and the description length rises. For example, reducing the number of units in each of the 4 VQs to 4 more than doubles the reconstruction cost. Extending the number of units beyond 6 per VQ can sometimes improve the representation. The additional units typically respond to, and reconstruct line segments between the control points of different orientations, in effect carving up the space of segments at a finer resolution. The cost of additional units, however, is reflected in the activity cost. With more units per VQ, if the network does not make them null units, the expected activity of each unit in the VQ