A Minimum Description Length Framework for Unsupervised Learning

by

Richard S. Zemel

A thesis submitted in conformity with the requirements
for the degree of Doctor of Philosophy
Graduate Department of Computer Science
University of Toronto

Abstract

A fundamental problem in learning and reasoning about a set of information is finding the right representation. The primary goal of an unsupervised learning procedure is to optimize the quality of a system's internal representation. In this thesis, we present a general framework for describing unsupervised learning procedures based on the Minimum Description Length (MDL) principle. The MDL principle states that the best model is one that minimizes the summed description length of the model and the data with respect to the model. Applying this approach to the unsupervised learning problem makes explicit a key trade off between the accuracy of a representation (i.e., how concise a description of the input may be generated from it) and its succinctness (i.e., how compactly the representation itself can be described).

Viewing existing unsupervised learning procedures in terms of the framework exposes their implicit assumptions about the type of structure assumed to underlie the data. While these existing algorithms typically minimize the data description using a fixed-length representation, we use the framework to derive a class of objective functions for training self-supervised neural networks, where the goal is to minimize the description length of the representation simultaneously with that of the data. Formulating a description of the representation forces assumptions about the structure of the data to be made explicit, which in turn leads to a particular network configuration as well as an objective function that can be used to optimize the network parameters. We describe three new learning algorithms derived in this manner from the MDL framework. Each algorithm embodies a different scheme for describing the internal representation, and is therefore suited to a range of datasets based on the structure underlying the data. Simulations demonstrate the applicability of these algorithms on some simple computational vision tasks.

Acknowledgements

I would first off like to thank my supervisor, Geoff Hinton, for his guidance and inspiration. Geoff's insistence on adopting a principled approach, and always thinking it through, will hopefully having a lasting effect on me, and it was a treat to witness a fountain of ideas and energy at work.

Peter Dayan was an amazingly prompt and thorough reader of this thesis. His red scrawl prompted several important changes in the document, and it has also generally been a pleasure and inspiration to be around and talk to Peter. I thank my most dedicated and consistent reader, Toni Pitassi, for her helpful comments and changes; she has had the misfortune of reading nearly every document I've produced, and has cheerfully made them clearer and cleaner. Sue Becker provided constructive and insightful comments; she is an excellent colleague and friend. And I thank people who made useful comments on sections of the thesis: Klaus Obermayer, and Paul Viola. Without the help of all these people, I might still feel like a hippopotamus trying to get out of a tub of molasses, to paraphrase H. L. Mencken.

An important thanks goes to Terry Sejnowski, who generously provided me with a terrific working environment for the home-stretch of the thesis. I also thank my committee—Allan Jepson, Ken Sevcik, Demetri Terzopoulos, and John Tsotsos—for easing me through the checkpoints along the way, as well as my external members, Jay McClelland and Irwin Guttman.

I also owe a debt to Mike Mozer, who took me under his wing at the beginning of grad school, and taught me a lot about how to do research. I've since enjoyed our collaborations, as well as his taste in tabloids and happy home-making.

Chris Williams, Tony Plate, Evan Steeg, Conrad Galland, Radford Neal, Sid Fels, Dave Plaut, Steve Nowlan and the other members of the Connectionist Research Group at Toronto made for an interesting and diverse group in which good discussions extended beyond research. I have also been lucky to work with Marlene Behrmann. Throughout graduate school, Eyal Shavit has been a great friend, as has Tim Brecht, Tom Fairgrieve, Sheila McIlraith and Diane Horton.

Finally, I would like to thank my family. Madeleine has provided excellent support. King Queet has always been there, and I am indebted to Ger, Shar, and my father, because their career paths convinced me that 3 doctors in the family was enough so I better find something else to do. I especially thank my father for showing me how to ask questions, to think scientifically, and to persevere. My mom has guided me through the micro and macro aspects of life with tons of love and encouragement. Toni has inspired and supported me for many years now—she sets a great example in both research and her warm, enthusiastic approach to life. My last thanks goes to the wonderful little bundle, Ella, who has changed our lives, all for the better. I especially thank her for being in such a hurry, because she's the reason I finally sat down to write up this thesis.

Contents

List of Figures

Chapter 1

Introduction

1.1 General motivation

A fundamental problem in learning and reasoning about a set of observations is finding the right representation. A good representation can make a difficult problem easy to understand and solve. For instance, a jazz solo may sound like a string of random notes to an untrained listener, but a musician who knows the standard chord progressions and transposition rules can often follow the piece and recreate it. A patient viewing a magnetic resonance image (MRI) of his knee may be unable to identify or distinguish bone from tissue, or even properly orient the image. A trained radiologist, however, who should know both knee anatomy and MRI techniques, will likely be able to match image elements to anatomical structures and create a representation of the knee which will allow proper diagnosis. In both of these cases, the representation captures some underlying structure in the domain, allowing a meaningful, useful description of the observations.

In both of these examples, the term *representation* refers to a particular description that accurately serves to characterize an observation. We can define each such representation as an instance within a *representation system*. For example, chords and sequences of chords form a representation system for music, while the major organs and connecting tissues provide one possible representation system for the human body. A representation system thus provides a framework for interpreting observations.

1.1.1 Task-independent representation systems

If the representation is to be applied to a specific task, it is clearly best to develop the representation with the requisite characteristics of that task in mind. Then the quality of this *task-dependent* representation can be judged on its suitability for that task.

In many cases, however, a *task-independent* representation is required, and even preferred. The task may be too complex to formulate desired characteristics of the representation system, or a procedure for extracting such a representation from the data may not exist. For example, in a visual recognition task, a representation of an object in terms of its salient features would greatly facilitate recognition, but defining those features for a broad class of objects, and then extracting them from images, are both very difficult problems.

One may also want to apply the same representation system to multiple domains. The examples above demonstrate that the same system may be useful for a number of tasks. An anatomical description of the major organs and connecting tissues can be used in diagnosis and repair. The chord progressions can be useful in recognition, memorization, and improvisation.

The central issue concerns what makes a "good" representation. In this thesis, we adopt a definition of a good representation as one that captures the underlying structure, or *true causes* of the domain. This definition contains two implicit assumptions:

1. the instances are produced by a small set of underlying structured processes
2. discovering this structure and casting the examples in terms of it will produce a useful task-independent representation

The first assumption is based on the idea that examples drawn from a common domain, or generated by a single distribution, are not simply random, but rather share some underlying structure. If the domain is the set of pixel-based images of instances of certain objects, then even though they may appear random, each is caused by a specific combination of object examples, viewing angle, and lighting conditions.

The second assumption posits that finding these causes will facilitate further processing of the data. Representing the set of examples in terms of their underlying causes may be useful in two ways. First, it should provide a more compact, structured description which will aid in later reasoning and inferencing about the domain. Second, casting examples drawn from a different domain in terms of this structure should produce poor representations, which allows the representation system to act as a classifier.

1.1.2 Learning good representations

It is difficult to hand-design a representation system suitable for a range of domains. Ideally, we would like to design a computational procedure capable of adapting the representation system to a particular domain of observations; we would like the procedure to take a representative sample of input examples, and induce a representational system in which each example has a proper representation.

This learning procedure is intended to capture structure in a given domain; the key question remains: *What is the structure, or set of regularities that underlie a particular domain?* An important theme of the work is that we can define different types of structure that apply to a range of domains. Our learning procedure then incorporates prior knowledge as to the appropriate type of structure to constrain the possibilities and aid the search for the underlying causes of a dataset.

This thesis focuses on a particular computational framework for addressing this problem—artificial neural networks. A neural network can be viewed as a type of nonlinear approximation method, in which it is relatively easy to express certain kinds of constraints and biases about the problem to be solved.[1] In terms of the discussion above, we can interpret the neural network as a representation system, and the states of units in the network as forming particular representations in this system. In broad terms, the goal then is to define and refine the representation system to yield representations that capture the true causes of examples from a particular domain.

1.1.3 Efficient reconstruction

One simple method of determining if a procedure found the causes of a dataset is to use the new representation to regenerate the data. This ensures that the representation preserves the original information in the data.

One representation that clearly can reproduce the original data is a copy of the data. This representation, however, does not capture the underlying causes of this data. If there are only a few causes, by Assumption 2 above, then representing the data in terms of its causes should produce a succinct, efficient representation. We would thus like to express the data in a compact form that allows its recovery. Then, a crucial test to determine whether the system embodies the true causes is its ability to *generalize* to unseen examples from the same domain.

[1] See Section 1.2 for a discussion of why we focus on this framework in the thesis.

The jazz listener remembers the chord structure of a piece, and this provides a succinct representation relative to the number of notes. He can use it recreate the gist of the piece at home, and can also apply the same structure to other pieces, and use it in his own improvisations. The orthopedic surgeon can form a working representation of the damaged knee based on interpreting the 2D images with respect to his anatomical knowledge; this representation is capable of recapitulating the important aspects of the original images. The surgeon's representation should also be capable of predicting anomalies in other 2D views of the same knee. The key idea is that a good representation system that captures structure can efficiently preserve the significant information in each observation.

In this thesis, we consider a neural network learning paradigm that is based on this idea—the *efficient reconstruction* paradigm. The goal of reconstruction networks is to create a compact, or efficient representation of each observation from which the original observed data can be regenerated. An important trade-off exists between the efficiency of the representation system and its reconstruction ability. The most accurate representation is clearly the original data itself. But this is typically not compact. The most efficient representation system uses the same representation for each input, yet these representations will be unable to reconstruct the individual inputs. This idea of balancing the conflicting goals of efficiency and accurate reconstruction underlies this paradigm.

The thesis focuses on developing representations that capture structure in the input domain. We apply ideas from information theory to obtain a computational framework for developing efficient representations in the reconstruction paradigm. This framework is useful in understanding existing techniques, as it reveals some important implicit assumptions. The framework also allows for a range of new algorithms. The various algorithms can be distinguished from one another by the form of pre-existing structure assumed about the data that is built into the representation system. We describe three new learning algorithms developed from within this framework that produce different forms of representations from datasets.

The remainder of this chapter presents an overview of the entire thesis, after first discussing some related work and introducing some necessary concepts and terminology from the field of neural networks.

1.2 Related work

In this section we briefly relate the work described in this thesis to two broad areas—data compression and machine learning—for the purpose of placing the work in a more general context.

1.2.1 Data compression

A central aspect of the work described in this thesis concerns constructing compact data descriptions that maintain data fidelity. The primary contribution of this thesis is a general framework for attacking this goal. In Chapter 3, we show how versions of the standard compression techniques, such as Karhunen-Loeve transform coding and vector quantization, can be described within this framework. By adopting this viewpoint, we will also see that existing methods occupy only a limited subspace of the space of possible algorithms within the general framework.

An important difference between the new algorithms explored here and many of the standard data compression techniques (e.g., JPEG) is that these methods typically assume that some predetermined coding approach provides a good representation of the data, while we instead learn the representations, effectively adapting the coding to the data statistics. Existing methods are popular since they are simple both conceptually and computationally (e.g., waveform techniques, such as Pulse Code Modulation or Delta Modulation), or because they can greatly compress the data with only minimal computation (e.g., transform techniques, such as the Discrete Cosine Transform, or the Discrete Fourier Transform). Adaptive techniques, on the other hand, generally perform better for a given bit rate, but they add complexity to the coding.[2] The underlying assumption of these adaptive techniques is that the additional cost in terms of computational requirements of the coder is not significant in applications where the decoding process where remains simple and there are many more receivers than transmitters, such as in digital television.

The work described in this thesis therefore falls into the general area of adaptive data compression. As such, it is related to data compression algorithms that adapt to a training sequence, such as vector quantization (VQ) techniques (see (Gray, 1984) for a review). Indeed, it turns out that one of the new algorithms presented in this thesis (see Chapter 5) is closely related to a *product VQ*, a novel variation of vector quantization suggested by Gray (1984). Different types of VQs, as well as other adaptive compression methods, are discussed further in Chapter 3.

Finally, our approach contrasts with many of the traditional techniques such as JPEG, where the data is quantized while attempting to maximize data fidelity, and then in a separate step, a procedure such as Huffman encoding is used to assign bits to the codes. One important novel aspect of the thesis with respect to these compression techniques is that we are adapting the bit rate of the code at the same time as we attempt to maintain data fidelity. We see below that the framework provides a principled way of trading off these two often conflicting goals.

1.2.2 Machine learning

The broad aim of the work described in this thesis is to induce data descriptions that capture underlying regularities of a set of input examples. In the area of machine learning, one popular related paradigm is based on analyzing input examples with respect to some initial knowledge. For example, in explanation-based learning, e.g., (Mitchell, Keeler and Kedar-Cabelli, 1986; DeJong and Mooney, 1986), the system begins with a domain theory, which contains deep background knowledge of a domain. This approach is not suited to the problem of learning from scratch. Other machine learning paradigms do inductively construct representations without detailed prior knowledge. These symbolic approaches, e.g., (Michalski and Stepp, 1983; Fisher, 1987), typically attempt to form simple descriptions that define necessary and sufficient conditions for the input domain, usually in terms of conjunctions or disjunctions of a small number of attributes. These methods induce descriptions from the data that allow instances from the input domain to be rederived or generated, while disallowing instances outside the domain.

[2] Note that an adaptive technique, such as a neural network approach, can perform at least as well as any standard non-adaptive technique, since the network could be initialized with weights corresponding to the standard technique, and then adapting the weights to minimize the objective function will improve the compression.

These methods operate directly on the input data, assuming that the input features are predefined. While these methods generally consider these input variables independently, we are instead interested in finding higher-order, "hidden" structure. An important defining feature of the algorithms we describe in this thesis is that they search for a representation in terms of the hidden variables that capture the interactions among the observed input variables. Several methods have been developed for constructing this type of representation. For example, *belief networks* (Pearl, 1988) are designed as a probabilistic representation of the observations in terms of a set of underlying causes. While Pearl relied on experts to specify the parameters in these networks, Neal (1992) introduced a neural network learning algorithm that allows the parameters to be estimated from the data. We focus here on other methods of discovering these representations.

1.3 A Neural network approach

In this thesis we investigate approaches within the neural network paradigm. Neural networks offer a compromise between computational power and searching ability. Their power derives from the fact that they do not presume that the feature set is predefined, but instead can infer hidden variables; they can work with real-valued inputs; and the models can be highly non-linear. Yet they also provide the ability to search, as several algorithms for optimizing the network parameters have been investigated. Indeed, some of the novel aspects of the thesis involve methods of defining network algorithms in order to facilitate search.

Neural networks can also be investigated as models of biological systems. Research in neural networks related to biology can be generally divided into two approaches. The first builds networks faithful to current knowledge of biological systems, and investigates what these systems can compute. The second approach builds computational systems that handle similar inputs and perform tasks presumed relevant to biology, and then attempts to link these to current biological knowledge. We adopt this latter approach in this thesis, as the computations described in our novel learning algorithms are rather more complicated than we can expect to be characteristic of biology. But as part of this latter approach, we turn to biology for motivation. We attempt to keep the computations simple and local, and also favor certain representations that resemble current ideas of neural coding (see Section 1.4.3).

We postpone discussion of related work in the neural networks literature to the appropriate chapters below, and now introduce terminology and concepts from this area that we will use throughout the thesis.

1.3.1 Basic terminology

A neural network model consists of a set of *units* and a set of *links* between them. We will consider the links as being directed; bidirectional links can be equivalently modeled by two separate directed links. Each link connects a *presynaptic* unit to a *postsynaptic* unit.

A link from unit i to unit j has a single parameter associated with it—a *weight* w_{ji}—which determines the strength with which the signal in unit i is transmitted to unit j (see Figure 1.1). We denote the *weight vector* into unit j by $\mathbf{w}_j$. The total *input* to a unit depends on the links that point

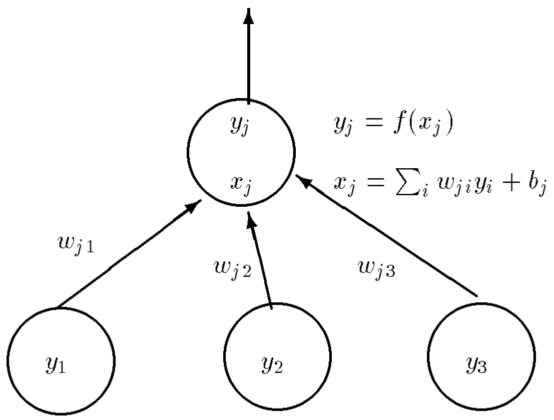

Figure 1.1: The output of a unit in a standard neural network is a function of its weighted summed input.

into that unit: it is some function of these weights and the state of the appropriate presynaptic unit. The standard definition of the input to a unit is the inner product of the states of its presynaptic units and its incoming weights:

$$x_j = \sum_i w_{ji} y_i + b_j$$

In this equation, b_j is the *bias* of unit j, and y_i represents the *output*, or *activity* of unit i, which is generally some fixed function f of its input, known as the *activation function*:

$$y_j = f(x_j)$$

Standard activation functions include a simple linear function ($f(x_j) = cx_j$), or some nonlinear function, such as a binary threshold function, or the logistic function ($f(x_j) = (1 + e^{-x_j})^{-1}$). The activation function for a unit can also be more complicated. For example, the activity level of a *Radial Basis Function* (RBF) unit is a Gaussian function of the distance between its incoming weights and the input vector:

$$y_j = (2\pi)^{-d/2} |\Sigma|^{-1/2} exp(-\frac{1}{2}(\mathbf{w}_j - \mathbf{a})^T \Sigma^{-1} (\mathbf{w}_j - \mathbf{a}))$$

where a is the input vector of dimensionality d, and Σ is the covariance matrix. The activation function may also be non-local, i.e., it can depend on states of other units. For instance, the activity of a normalized exponential, or *softmax* unit (Bridle, 1990; Nowlan, 1990) is a function of a set of other units k, where k typically indexes other units in the same *layer* as j:

$$y_j = \frac{exp(x_j)}{\sum_k exp(x_k)}$$

The configuration of the units into layers, together with the configuration of links between units comprise the *network architecture*. The architecture usually designates a set of units to be the *input units*, whose states are determined by some external process, and a set of *output units*, whose states are considered the output of the network. The rest of the units in the network are considered *hidden units*. We will refer to a network with at least one hidden layer as a *multilayer network*.

If there are no cycles formed by its connections, the network is *feedforward*, otherwise it is *recurrent*. In feedforward networks, the computation proceeds sequentially from one layer of units to the next, and the current state of any single unit only depends on the current input to the network and not on any previous activity. In this thesis, we concentrate primarily on feedforward networks.

In many neural network models, there is a probabilistic interpretation of the activities of the units in the network. For example, some models contain *binary stochastic units*, where the unit's activity is a binary random variable. The network associates a continuous value between 0 and 1 with each such unit; this value is then interpreted as the expected value of the unit, i.e., the probability that the unit is on, with respect to some ensemble of network states. Many of the models that we consider in the thesis utilize units of this type.

1.3.2 Learning in neural networks

A key theme in neural network models is that the weights are the only modifiable parameters of the model, and that all long-term knowledge is stored in the weights. Learning involves changing these weights; the weights are adjusted to reflect the experience of the system through a *learning procedure* (see (Hinton, 1989a) for a comprehensive review of neural network learning procedures).

A neural network learning procedure can be defined simply as a *learning rule*—an equation for updating the weights in the network. In many cases, this learning rule is derived from some function to be optimized (minimized or maximized) by the network. This function is referred to as an *objective*, *error*, or *cost function*.

Many of the standard methods to train the networks use a *gradient-based* technique to optimize the objective function. In this thesis, we use the *back-propagation* algorithm to obtain this gradient (Werbos, 1974; Parker, 1985; Rumelhart, Hinton and Williams, 1986). This algorithm efficiently computes the partial derivatives of the objective with respect to each weight (and any other adjustable parameter) in the network for each input pattern. The computation is a 2-step procedure: first, a forward-pass determines the activity of each unit from the input up through the output; then a backward-pass propagates the partial derivatives by applying the chain rule back from the output layer through the intermediary hidden layers to the input. This gradient information is then utilized by some optimization method to adjust the weights so as to decrease the error, or cost function.

Learning procedures can roughly be divided into two categories: supervised and unsupervised. In *supervised* learning, the network is presented with examples of input and desired output patterns. The goal is to adjust the weights so that the network can compute a mapping that best fits this training data, as measured by some sort of optimality criterion, such as mean-squared error (MSE). The most popular and successful supervised learning procedure uses back-propagation to minimize the MSE of a network.

In *unsupervised* learning, the network is only given a set of inputs; the desired outputs of the network are not specified. Instead, we define some task-independent measure of the quality of the representation to be learned, and the weights are then optimized with respect to that measure. An important point here is that the general procedure of learning by back-propagating derivatives of a

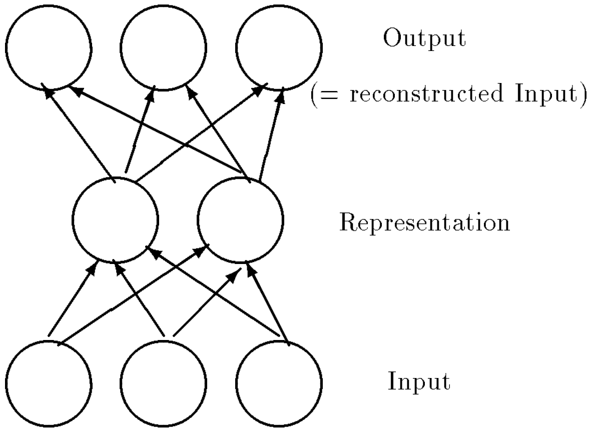

Figure 1.2: A sample architecture for a simple feedforward autoencoder, containing a single hidden, or representation layer. This network is *unsupervised*, or self-supervised, in that the input, rather than some external signal, provides the target output.

cost function through the network can also be applied to unsupervised networks.

These two learning procedures actually lie at either end of a spectrum of learning procedure classes, where different degrees and types of supervision may be provided to the network. For instance, in a third class of learning procedures known as *reinforcement* learning, the supervision is typically in the form of a scalar, often binary reward signal which indicates the quality of the output. We return to a discussion of the various learning procedures in the following chapter; for now, the basic definitions suffice.

1.4 Thesis summary

This thesis focuses on developing representations in a particular framework of unsupervised learning, where the goal of the system is to develop representations that are capable of reconstructing the data. A key aspect of this work concerns how *a priori* assumptions and knowledge about the dataset can be built into the definition of a network, i.e., its architecture and activation functions. These assumptions will then influence the types of representations that the network will tend to develop.

1.4.1 An information-theoretic framework for reconstruction

A basic contribution of this thesis is to propose a general framework for a particular form of unsupervised learning—the *reconstruction paradigm.* The second chapter of the thesis presents this framework, as well as describing some of the relevant fundamental ideas from information theory.

In the reconstruction paradigm, a network must learn to reproduce each input case. That is, when a given training pattern is presented on the input units of the network, it must learn to duplicate this pattern on the output units. A reconstruction network is also known as a *self-supervised* network, or an *autoencoder.*

A reconstruction network is typically defined in one of two forms. The first type is a multilayer feedforward network, the second is a recurrent network. We focus on the former here. The standard feedforward autoencoder (Figure 1.2) contains a single hidden layer between the input and output layers; there may be additional hidden layers. The network is typically trained to reconstruct the input on the output units, by minimizing an error function based on the difference between these two activity patterns, computed for the entire training set. Once the network learns the task, the activities in the hidden layer can be viewed as an information-preserving representation for the input; we refer to this layer as the *representation* layer. In this type of network, the hidden layer typically contains fewer units than the input layer, thereby forcing the network to find compressed representations of the inputs. Reconstruction networks are often used as a pre-processing technique, as the compact representations they produce can then be used as inputs to some further processing stages.

The following chapter describes an information-theoretic framework for reconstruction. We begin by restating the two important assumptions from Section 1.1. First, we assume that the observed data is the result of some underlying structured process. That is, the input set, typically visual images, is not simply a collection of random vectors, but rather has some coherence. Second, we assume that discovering this structure and representing the data in terms of it will facilitate further processing of the data.

We re-cast the problem as a communication problem. The goal is to *encode* the data so that it can be transmitted across a communication channel. We would like to encode and transmit the data as *accurately* and *succinctly* as possible.

These two terms capture the important trade-off that underlies the framework. The accuracy refers to the degree to which the input information is preserved in the encoding. This goal is inherent in the reconstruction paradigm. The motivation for succinct encodings derives from the idea that a representation system that captures structure or regularities underlying the data set allows for compact descriptions of the data. The strategy is therefore to search for structure by forcing the network to form encodings for a set of inputs which are accurate and succinct.

This strategy of searching for data descriptions which optimize the two conflicting goals of accuracy and succinctness is captured in computational, quantitative terms by an information-theoretic version of Ockham's razor. The *Minimum Description Length* (MDL) principle (Rissanen, 1985) informally states that the best model is the one which minimizes the sum of two terms:

1. the length of the model, and

2. the length of the data when encoded using the model as a predictor for the data

In the standard MDL framework, the model acts as a representation for the data, and the length of the data is the misfit between this representation and the data. We adopt a slightly different approach, where the model, here a neural network, is not itself a representation, but rather a representation system, and can be used to produce representations for different datasets. We thus add another level to the data description, so that the second item in the sum itself contains two terms:

2a. the representation of the data item using the model, and

2b. the mismatch between this representation and the actual data.

The MDL principle suggests a means of evaluating this representation system. We quantify the description length of the entire model (containing all three terms) for the set of training cases; the evaluation criterion is the minimum number of bits in the description that some computational procedure can use to reproduce the observations.

This idea can be applied to the reconstruction paradigm by viewing the problem in terms of the simple communication model shown in Figure 1.3. We imagine that a sender must transmit a description of a set of input vectors. For now, we assume that she uses a simple autoencoder to find representations for the input set. She then must *encode* these representations and transmit them to the receiver. In order for the receiver to *decode* these representations, we assume that the sender and receiver agree beforehand on a particular network architecture which the receiver will use for decoding. The sender then must transmit a message describing the input set, consisting of three distinct parts:

1. the values of the weights ($\mathcal{W}$) that connect the representation units to the output units
2. the activity levels of the representation units on each training case ($\mathcal{A}$)
3. the error in the prediction made by the representation units, or the mismatch between the actual output and the input on each case ($\mathcal{R}$)

Given these three terms, the receiver has all the information required to recover the original input dataset.

We can measure the succinctness of the representation in terms of the first two terms, while its *in*accuracy is described by the third term. The three terms together comprise the cost function for the reconstruction task; the sender attempts to minimize the total cost of the message, measured in bits.

Adopting this communication model viewpoint requires some nonstandard terminology for the autoencoder. The hidden units are typically known as the *code* layer, since the activity levels encode the input, and the goal is to find a compact code. We, however, are encoding these activity levels, and a compact code for these activities is only one of three components in a succinct representation of the input. We thus call the hidden activity levels a *representation* of the input.

Chapter 2 presents some examples of various ways in which these terms may be defined, as well as describing some of the more general background for this novel framework.

1.4.2 Unsupervised learning: Representations and algorithms

The third chapter describes some of the background for the unsupervised learning methods presented in the thesis. We discuss motivations underlying unsupervised learning, and present a key theme of the thesis—unsupervised learning algorithms are closely associated, either implicitly or explicitly, with the particular types of representations they tend to form from the data. We introduce two of the primary classes of unsupervised learning algorithms: kernel (i.e., clustering-based) and projection (i.e., principal-component-based) methods. These two classes form representations that occupy two ends of a spectrum, which ranges from purely local to fully distributed representations.

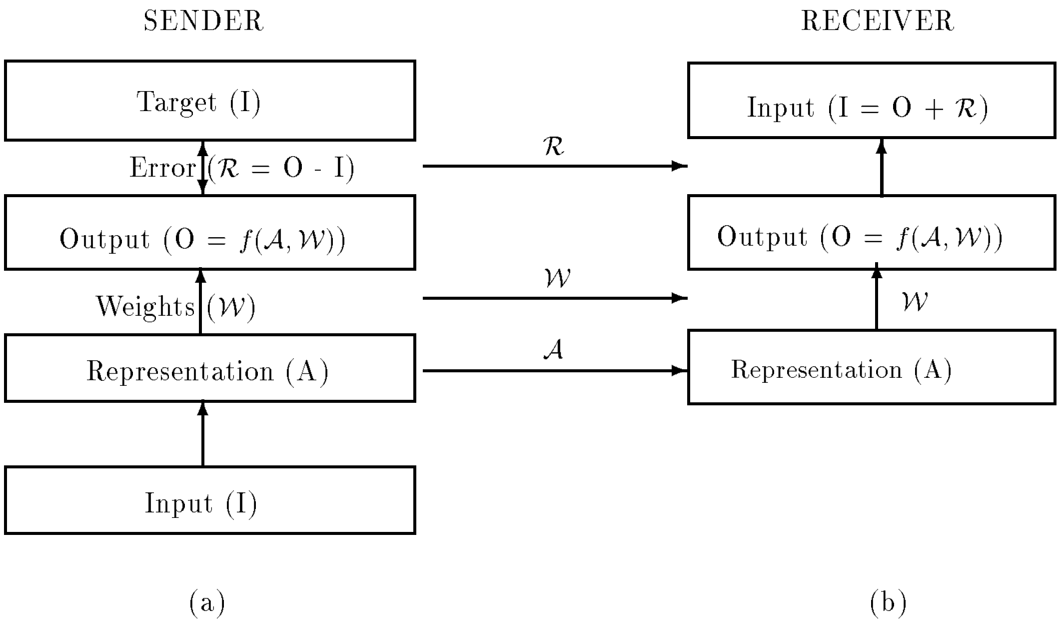

Figure 1.3: (a) The sender uses a reconstruction network to develop representations for the input set. She communicates a single message to the receiver, containing 3 parts: the weights $\mathcal{W}$, the set of representation activities $\{\mathcal{A}\}$, and reconstruction errors $\{\mathcal{R}\}$ for the ensemble of input patterns. (b) The receiver knows the network architecture and activation functions. He first applies f to the weights $\mathcal{W}$ and the representation activities $\mathcal{A}$ on each case to obtain the pattern of activity O on the output units, and then adds on the reconstruction error $\mathcal{R}$, to reconstruct the input I.

This chapter focuses on three of the most common representational frameworks, whose associated representations fall within these bounds; these three are particularly relevant to the new algorithms presented in the thesis. Each of these representations typically involve binary units, and a unit is considered to be representing an input if it is *active*, i.e., its activity is 1 rather than 0. The first type of representation is a *minimally redundant representation*, in which the units that represent each data item are as nearly as possible independent of each other. Barlow (1961) originally proposed the utility and biological importance of this form of representation, and it has remained a common aim of unsupervised learning procedures. The second representation type is a *vector quantizer* (VQ), where each data item is represented by a single unit from a set of units, which in effect compete to represent the data. Standard algorithms such as LBG quantization (Linde, Buzo and Gray, 1980) and competitive learning (Rumelhart and Zipser, 1985) form this type of representation. The third representation is a *topographic representation*, where each data item is represented by a few active units, and similar items are represented by nearby sets of units. Several unsupervised learning procedures, including the Kohonen algorithm (1982), have this type of representation as their goal.

We can view most of the algorithms described in this chapter as special cases within the MDL framework. A wide variety of previous learning schemes, including those based on principal-components or clustering, have focused on optimizing only one of the three MDL cost terms, assuming that the others are fixed in advance. These methods have typically concentrated solely on optimizing accuracy, while building succinctness directly into the network architecture or activation function. For example, in winner-take-all competitive learning, only one representation unit is allowed to be active for any input case, so the activity level cost $\mathcal{A}$ reduces to specifying which of the N representation units is on, which requires at most $\log N$ bits. The various methods differ primarily in their fixed, *a priori* approach to the succinctness issue.

The framework thus allows for a unifying view of existing algorithms. We can see how different choices of encoding methods, i.e., different notions of succinct representations, directly lead to various major classes of unsupervised learning techniques. The framework sheds light on underlying ideas and limitations of some of these methods. Chapter 3 provides some examples of how these methods may be viewed within the framework, and introduces some other ways in which these terms may be defined.

1.4.3 Variations on reconstruction in the MDL framework

This framework, however, is not just another set of criteria for viewing existing algorithms. Adopting this view makes it clear that existing methods occupy only a limited subspace of the space of possible algorithms within the general framework.

An important aspect of this thesis consists of explorations of some novel regions of this space. A number of new algorithms are proposed. A key distinctive characteristic of all of these algorithms is that they attempt to optimize simultaneously both the accuracy and succinctness of the encoding, rather than fixing the latter term in advance.

The new algorithms differ from one another primarily in the structural constraints placed on the representation. The framework only specifies that the cost is comprised of the total description length of the weights, the representation unit activities, and the reconstruction error. Each of these terms is highly dependent on the *encoding scheme* used to transmit these values. This encoding scheme, as well as the network architecture and activation functions, explicitly embody the prior assumptions. For example, if we expect the representation units' activities to be uniformly distributed between 0.0 and 1.0, our encoding scheme will allocate a fixed number of bits per value. On the other hand, if we know (either through *a priori* knowledge or training statistics) that the activity levels will generally be 0.5, then we can adopt a scheme which requires just a single bit to say whether the activity level is 0.5.[3] The encoding scheme thus reflects prior assumptions about the probability distributions of the values to be transmitted. Through the encoding scheme then, these assumptions have a direct effect on the objective function that the network is trained to optimize, and thus constrain the representations that the network will learn.

In addition to the encoding scheme, the network architecture and activation functions may reflect prior assumptions. For example, the algorithm described in Chapter 5 assumes that the data can be succinctly described as a function of a number of independent processes, each of which has several possible values. This assumption determines the structure of the network. The representation units are divided into m different pools, each pool acting independently of the others. Within a pool, the units compete, as the activation function ensures that only one unit can be active per pool. The encoding scheme is then optimized to transmit m independent values, each describing which unit is on in that pool.

In general, each algorithm imposes some degree of structure on the network, either through the architecture and objective function, or through the encoding scheme, and the learned description of the dataset reflects this structure. This structure embodies prior assumptions which limit the applicability of each algorithm, but since they are explicitly built into the algorithm, one may estimate the potential utility of the algorithm for a particular dataset in advance. Assumptions can arise from various sources, e.g., prior knowledge about the data distribution, such as the independent-process assumption mentioned above; knowledge as to which representation is likely to be most useful for later processing; or biological or psychophysical constraints.

Developing non-redundant binary representations

The three variations within the framework explored in the thesis demonstrate how different prior assumptions determine the type of representations discovered by a network. The first algorithm assumes that a good representation should capture the information in the input data while removing the redundancy. The encoding scheme reflects this assumption, as each representation unit's activity is encoded independently. The network thus wastes bits if these activities are redundant. The resulting network implements a version of a well-known objective for unsupervised learning—Barlow's scheme for minimum entropy coding (Barlow, 1983), where the sum of the entropies of the individual representation units is minimized. The motivation for these minimum entropy codes arises both from information processing issues as well as biological considerations (see Section 3.3.1).

We describe two versions of the algorithm. The first version assumes that a good representation is one where there are many representation units, but only a few units participate in representing

[3] This is a fundamental idea in coding theory; a more common example involves Morse code, where the most common letter (e) is sent with a single dot.

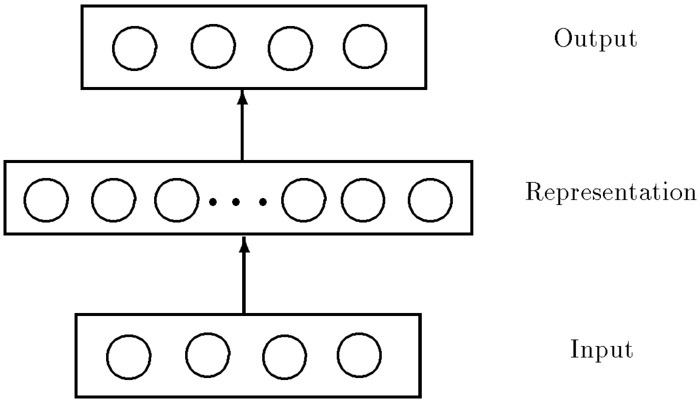

Figure 1.4: The network architecture for the reconstruction network used to find minimum entropy representations in Chapter 4. A single arrow between two layers of units indicates that the units are *fully connected*—there is a link from each unit in the first layer to each unit in the second.

any particular input. This assumption is captured both in the network architecture and encoding scheme. The architecture we use here is very different than the standard autoencoder with a few hidden units acting as a bottleneck. Instead, there are many hidden units (Figure 1.4). The encoding scheme, however, stipulates that a low cost is incurred when a representation unit has activity zero. The representation layer contains binary stochastic units; associated with each unit is a real value describing the probability that its binary activity equals one. The encoding scheme encourages units generally to have activity zero, and hence low entropy across the training set. The second version uses a more flexible, adaptive encoding scheme with the same basic goal of minimizing the entropies of the representation units.

Chapter 4 shows that both versions of the algorithm are capable of learning minimum-entropy representations for some datasets. We demonstrate this using experimental results from two sets of images. In the first task, the dataset consists of input images, where the vocabulary of parts in the images consists of horizontal and vertical lines in a fixed number of positions. The network learns a non-redundant representation system where each line instance has a corresponding unit, which enables a test image containing a few such lines to be accurately reconstructed based on the activity of a few units. In the second set, the network learns that input vectors actually can be described as a concatenated pair of vectors, each of which has a limited number of possible values. The algorithm again develops a method of decomposing each input and representing it using only a limited number of units.

Cooperative vector quantization

While the first algorithm is capable of learning representations that capture the regularities in the two datasets, we find that incorporating additional structure into the network allows the system to find these regularities more easily. The second algorithm in this framework incorporates this additional structure in the representation layer, grouping the units into pools so that they compete within a pool to be active, while cooperating to reconstruct the input (see Figure 1.5). This method

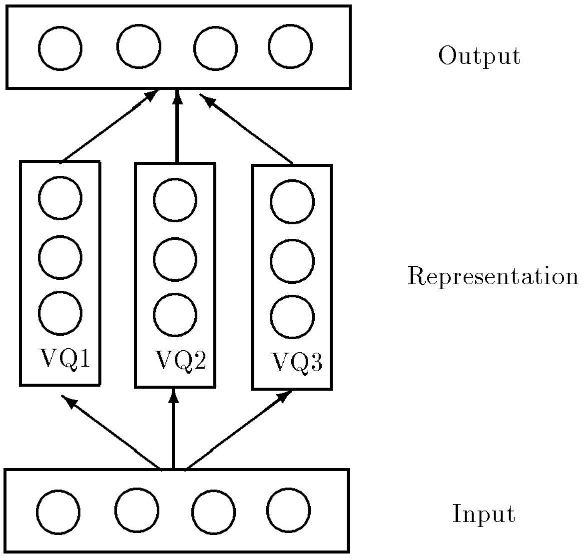

Figure 1.5: The network architecture for the Cooperative Vector Quantization algorithm described in Chapter 5. Within each of the three vector quantizers that comprise the representation layer, one unit is active per input case. These units then cooperate to reconstruct the input.

is the one mentioned above which assumes that the data is composed of a number of independent processes, each of which has several possible values. In standard *vector quantization*, a single unit comes to represent each input vector; in Cooperative Vector Quantization (CVQ), exactly one unit per pool comes to represent each input. This new scheme can lead to substantial savings in the number of hidden units. For example, in standard VQ, N hidden units can represent at most N items; in CVQ, if we divide the hidden units into pools of size k, we can represent $k^{N/k}$ items, provided these items have the factorial structure that allows each item to be reconstructed from N/k "factors." This algorithm can be seen as a more structured version of the previous algorithm, where on a single input, rather than individual units each being active or inactive and cooperating to reconstruct the input, now a single unit from within a pool is active and participates in the collective reconstruction.

The method tends to form representations in which the pools of units come to represent separate components of the input ensemble. Simulations demonstrate that this method allows networks to learn more efficient representations for the second dataset from the previous chapter. In addition, we show that this algorithm learns efficient representations of spline images by developing units with local receptive fields that form piece-wise linear reconstructions of the spline. We also present experimental results demonstrating the network's ability to learn the number of pools necessary to sufficiently represent the dataset. The formulation of this algorithm in the MDL framework enables the network to find highly efficient, informative representations of the dataset.

Developing population codes

Most current unsupervised learning methods, as well as the algorithms described in the previous two chapters, tend to remove redundancy in the activities of the hidden units in an attempt to minimize the activity cost $\mathcal{A}$. As a result, the methods are highly dependent on each unit functioning properly in order to represent a set of inputs.

An appealing alternative style of representation system has more distributed, *coarse-coded* representations in which information is represented by a whole population of active units. Here the activity of any single unit is not critical for representing an example, yielding more fault-tolerant and gracefully degrading solutions. Distributed representations possess the important property of *stability*—a small change in the input produces a small change in the representation.

Another feature of a representation that is appropriate for a wide variety of input sets is for it to reflect the underlying dimensions of variability in the input. Inputs may vary along several different dimensions. For example, a car may be represented by dimensions such as sportiness and fuel-economy, as well as spatial dimensions such as position and size. One sort of representation that captures these parameters is known as *topographic*, which means that points nearby in some underlying dimensions of variability in the input set will map to points physically nearby in the representation. In addition, topographic representations are biologically realistic, as dimensions such as position, orientation, and color are represented in topographic fashion in striate cortex of mammals. Individual neurons have been shown to respond to a limited range of stimuli along some of these dimensions, and nearby cells respond to similar inputs.

In this thesis, we use the term *population code* to refer to a representation where a unit may not have a physical position, but it has some associated position in the underlying dimensions of variability of the input. Population codes also appear to be characteristic of the brain. For example, recent studies have found that in monkey motor cortex, the direction of movement in 3D space of the monkey's arm is encoded by the activities of populations of cells, each of which represents a somewhat different direction (Georgopoulos, Schwartz and Kettner, 1986). Similarly, in superior colliculus, the direction of motion of the monkey's eyes in 2D space is encoded by a population of tuned cells (Lee, Rohrer and Sparks, 1988). A population code thus has the redundant, stable and fault-tolerant qualities of distributed representations along with the ability to represent positions in some underlying structure.

A population-code representation system, however, would seemingly be difficult to develop in the MDL framework as formulated above, as minimizing the cost of describing the internal representations discourages redundancy. If we alter the way in which these representations are encoded for transmission to the receiver, however, we can devise a novel learning algorithm based on the framework.

The algorithm described in this chapter proposes a less direct method of specifying the representation units' activities. This method allows a population-code representation to develop, and also encourages this representation to correspond to the underlying dimensions of variability. We make the network define a space to code the inputs and make it cheap to send a representation that respects the topology of this space. Each representation unit is assigned a particular point in an *implicit space*; this space is intended to come to represent a space that is not the input space, but rather the multidimensional space of variability underlying the inputs. Any input corresponds to a single point in this implicit space. Once the network has learned this space, each unit responds to a set of inputs that correspond to a local neighborhood in implicit space. Each input can therefore be represented by activating all the units that represent very similar inputs. Ideally, the population of active cells that represent a particular instance will form a smooth bump of activity in implicit space, in which the activation level falls off in a standard way as we move away from the center. The sender need only transmit the center of the bump in order to completely describe the pattern of activity. The cost of encoding the representation unit activity is the cost of describing the center of the bump plus the cost of describing how the actual activities differ from the desired smooth bump of activity. Thus the prior ideas as to what makes a good representation are reflected in the network structure and the encoding scheme.

Simulations demonstrate that the method learns to construct population codes for 2D shape instantiations, where each training image contains an instance of a particular shape. In the first task, the underlying dimensions of variability are the (x, y)-position of the shape, and the 2D implicit space comes to correspond to these two dimensions. The second task also varies the orientation of the shape, and the 3D implicit space again corresponds to the three dimensions of variability. Simulations also demonstrate that this method is capable of forming flexible and interesting structures in the implicit space, as the network develops a discontinuous topology when presented with different input classes.

1.4.4 Conclusions and future directions

The final chapter summarizes the primary contributions of this thesis, and describes directions of future research based on extensions and improvements of the work presented herein.

Chapter 2

A Framework Based on Minimizing Description Length

An important theme in this thesis is that each unsupervised learning algorithm entails, implicitly or explicitly, a particular encoding scheme, and that unsupervised learning can be viewed as the search for a set of good representations within a given encoding scheme.

In this chapter we show how this search for good representations can be cast in terms of an information-theoretic principle based on minimizing description length. We present some of the information theory background behind this principle, and discuss links between this information-theoretic approach and a Bayesian probabilistic approach. We then present some previous applications of the MDL principle, in the areas of machine learning, computer vision, and neural networks. Finally we introduce a new application of MDL to unsupervised learning, and describe some of the important components of the resulting learning framework.

2.1 Minimal-length information-preserving representations

A starting place for this general method of comparing alternative representations is the intuitive idea that simpler models are better than more complex ones. The 14th-century English philosopher William of Ockham stated the principle that "Plurality must not be posited without necessity." This principle, known as Ockham's Razor, is commonly applied in science, and is usually thought of as a heuristic guide for eliminating overly complicated hypotheses.

Determining which hypothesis is simplest is often no simple matter. One possible criterion is the number of degrees of freedom. But this definition is still underspecified. For example, several models can describe a set of points in a plane. The first is to describe each point independently, simply by specifying its coordinates. A second model may be as points along a curve, while a third model may group them into clusters. Comparing the simplicity of these models with respect to the data is not straightforward.

We set the problem up in terms of a communication model, where the goal is to transmit a set of data from a sender to a receiver. The sender must first communicate the type of model that can be used to describe the data. She first needs to specify the type of model, as well as its actual parameters; even amongst the different models in the set of planar points example, there are many different possibilities. The curves may be quadratic or sinusoids; the clusters may be overlapping Gaussians or distinct Voronoi regions. All of this complicates comparing the simplicity of different models.

In addition, *noise* plays an important role here. If the data is generated by a stochastic system, then a key element of the model is the noise in the data with respect to the model. Even in deterministic systems, the model will often be incomplete, or may not exactly match the underlying system. Here one also needs to describe the deviations from the models. The fit of the models to the actual data must therefore also be taken into account in a simplicity measure.

The information-theoretic answer to this question of simplicity is to reduce each model to its most basic form: a set of bits. This answer converts the heuristic idea of Ockham's Razor into a more principled approach. The measure of simplicity here is *the number of bits in the description that some computationally effective procedure requires in order to reproduce the observations.* The advantage of this formulation in terms of *reconstruction* is that one can specify procedures to reconstruct both the deviation of the data from the model, and the model itself. This leads to a natural trade-off between the number of bits to specify the model and the bits to specify the observations given the model.

The principle of searching for the model that permits the shortest encoding of the data, together with the model, is known as the Minimum Description Length (MDL) principle (Rissanen, 1985). MDL fits into an area of statistics known as model selection. Related work includes that of Akaike (1974), who suggested a model selection criterion that minimized the negative logarithm of the likelihood of the data given the model, and then added on a model complexity penalty based on the number of parameters in the model. Rissanen described this work within the framework of a communication model, where these terms are unified into a single encoded message that must be decoded in order to reproduce the data. The sender takes as input a set of observations, and sends an encoded message that the receiver decodes and uses to produce the output. When the output of the receiver is identical to the input, then the message is called an information-preserving description of the input.

The output of the sender is a message written in the description language $\mathcal{L}$ of the receiver. This message conveys both the model M, and the data D when encoded such that M can decode it; this second term can be seen as the errors, or *residuals*, i.e., aspects of the data not predicted by the model. The MDL approach attempts to minimize the length of this encoded message:

$$|\mathcal{L}(M, D)| \;=\; |\mathcal{L}(M)| \;+\; |\mathcal{L}(D \text{ using } M)| \tag{2.1}$$

2.2 MDL from a Bayesian perspective

The notion of comparing models based on simplicity can equivalently be expressed from a Bayesian perspective. The goal is to infer a model M from a set of observations D. Bayes's rule states that the posterior probability of a model is:

$$p(M|D) \;=\; \frac{p(D|M)\; p(M)}{p(D)} \tag{2.2}$$

The most plausible model given the data is then inferred by comparing the posterior probabilities of all models:

$$\text{argmax}_M[p(D|M)\; p(M)] \;=\; \text{argmax}_M[\log p(M) \;+\; \log p(D|M)] \tag{2.3}$$

The link between these two objectives (Equations 2.1 and 2.3) is provided by Shannon's *optimal coding theorem* (Shannon, 1948). This theorem states that the ideal code, generated from an optimal description language $\mathcal{L}^*$ based on a known probability distribution over discrete events, for a value x of a variable X will have an expected length given by:

$$|\mathcal{L}^*(x)| = -\log p(X = x) \tag{2.4}$$

A deeper relationship linking Bayesian analysis, MDL, and Ockham's Razor can be seen by considering the case of parameterized models. The intuitive idea is that more constrained models—

models with fewer parameters, and restricted parameter ranges—are simpler. A complex, general model can fit a wide range of datasets, while a simpler model can fit a narrower range of observations, and is more readily falsified. This relationship between model generality and number of parameters is related to what is known as the *bias-variance dilemma* in statistics: introducing too many parameters into the model incurs high variance, while restricting the model to too few parameters incurs high bias in the set of possible solutions (Geman, Bienenstock and Doursat, 1992). The Bayesian approach, like MDL, offers one approach to this dilemma by formalizing the intuitive idea in Ockham's Razor—a complex model is only preferred when its predictions are sufficiently more accurate—as an inference rule.

This can be made more concrete by considering the case where a model has a set of parameters $\mathbf{w}$. The probability of data given a model can be computed by integrating over possible values of $\mathbf{w}$:

$$p(D|M) \;=\; \int p(D|\mathbf{w}, M)\; p(\mathbf{w}|M)\; d\mathbf{w} \tag{2.5}$$

For many problems it is common for the posterior distribution for $\mathbf{w}$ to have a sharp peak at the most probable parameter vector $\mathbf{w}^*$, so this integral can be approximated by its height at $\mathbf{w}^*$ times its width $\Delta\mathbf{w}$:

$$p(D|M) \;\approx\; p(D|\mathbf{w}^*, M)\; p(\mathbf{w}^*|M)\; \Delta\mathbf{w} \tag{2.6}$$

The first term in this product is the maximum likelihood fit that the model can achieve; the second term ($p(\mathbf{w}^*|M)\, \Delta\mathbf{w}$) is an "Ockham factor" (Gull, 1989; MacKay, 1992b) which penalizes the model for having parameters $\mathbf{w}^*$. Here $p(\mathbf{w}^*|M)$ is the prior distribution for the parameter vector $\mathbf{w}$, and $\Delta\mathbf{w}$ is the posterior uncertainty in $\mathbf{w}$. A complex model with many parameters, each with many possible values, will have a broad $p(\mathbf{w}^*|M)$ and hence a small Ockham factor. A simpler, more constrained model will have a sharper prior. For the more complex model to be favored over the simpler one, it must achieve a much better fit to the data. Thus Bayesian model comparison techniques choose between alternate models by trading off this measure of the simplicity of a model against the data misfit.

MDL methods suggest the same trade-off on this problem. Different models will be compared based on the code length of $p(D|M)$. For a given model this involves minimizing $-\log p(D|\mathbf{w}^*, M)$, the number of bits in the ideal shortest message that encodes the data D using $\mathbf{w}^*$ and M, plus $-\log p(\mathbf{w}^*|M)$, the bits required to encode these parameters. MDL model comparison then selects the model that best balances the trade-off between these two terms.

2.3 Coding and probability distributions

Shannon's theorem describes the optimal code if the true probability distribution for a set of discrete alternatives is known. In general, however, one does not know which is the true distribution; an implementation of MDL typically necessitates approximations in evaluating the length of the ideal shortest message.

Constructing codes from a description language based on the wrong probability distribution will always take more bits, on average, so selecting an appropriate probability distribution for the codes is a key aspect of MDL applications. One approach is to tabulate statistics in order to estimate the true underlying probability distribution. An alternative is to select a particular probability distribution based on prior knowledge about the input, and encode as if the input conforms to that distribution. In either case, as the selected distribution approaches the actual one, one can code optimally by defining $\mathcal{L}^*$ to generate codes according to Equation 2.4. The key point is that selecting a distribution is equivalent to selecting a code, and that MDL allows the comparison of various choices based on the code length of the observations.

2.3.1 Example 1: Encoding parameters using universal priors

The first example of coding we consider concerns the original form of the MDL criterion (Rissanen, 1978). Here we return to the problem where the class of models is parameterized probability distributions whose parameters $\mathbf{w}$ are real values about which one has no specific prior knowledge (hence the name *universal priors*). The code here consists of two distinct parts: the parameter values $\mathbf{w}$ and the data $\mathbf{x}|\mathbf{w}$. The cost of encoding the data is simply the code length $|\mathcal{L}^*| = -\log P(\mathbf{x}|\mathbf{w}^*)$, where $\mathbf{w}^*$ is the maximum likelihood estimate for $\mathbf{w}$ (as in Section 2.2). To encode real-valued parameters by a finite binary string, they must be truncated to a finite precision; the cost of encoding the parameters then depends on the precision q to which they are specified.

An important point is to include the cost of truncating these parameter values into the total code length; the increase in code length of the data using this truncated $\hat{\mathbf{w}}^*$ must be traded off against the bits saved in describing $\mathbf{w}^*$. When n, the number of observations, is asymptotically large, by optimizing the precision q we find that the optimal code length for each component of $\hat{\mathbf{w}}^*$ is $q = -\log \frac{1}{\sqrt{n}}$. The total optimal code length for the data $\mathbf{x}$ with a k-component truncated parameter vector $\hat{\mathbf{w}}^*$ is given to within terms of order $O(\log n)$ by:

$$|\mathcal{L}^*(\mathbf{x})| \;=\; -\log P_{\hat{\mathbf{w}}^*}(\mathbf{x}) + \frac{k}{2}\log n \tag{2.7}$$

In the case where prior information fixes the number of parameters k, this criterion simplifies to the standard maximum likelihood estimation criterion. But in other cases, this second term in Equation 2.7 plays a crucial role, allowing the length to be minimized over the number of parameters to find the optimal model as well as its complexity.

2.3.2 Example 2: Encoding with a Gaussian distribution

In the first example, we considered how to encode quantized model parameters using universal priors. In many cases, however, prior information can be incorporated to estimate the appropriate probability distribution of the messages to be encoded.

One standard distribution that is assumed (often implicitly) for the purpose of coding is a Gaussian. For example, if we assume that the residual errors $\mathbf{r}$ for the data points (the second term in Equation 2.1) are independent and normally distributed about a zero mean, then the density function is:

$$P(\mathbf{r}) \;=\; \prod_{i,c} p(r_i^c) \;=\; \prod_{i,c} (\sqrt{2\pi\sigma^2})^{-1} \exp(-r_i^{c\,2}/2\sigma^2) \tag{2.8}$$

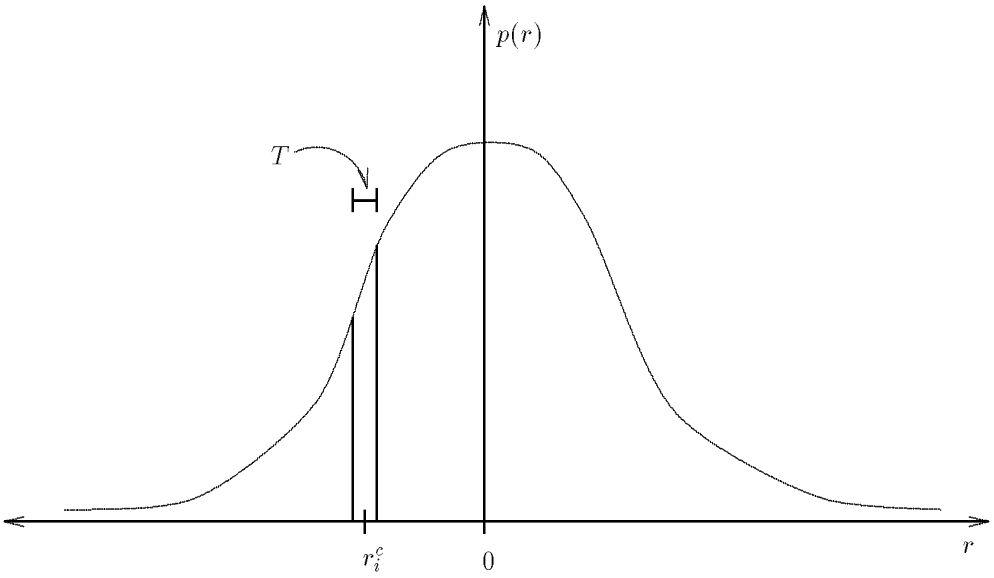

Figure 2.1: This figure illustrates the trapezoidal approximation to the probability mass in an interval of width T around a particular value. The mass approximately equals the product of T and the height under the Gaussian density function at r_i^c.

where r_i^c is the ith component of the residual for the cth datum, and σ^2 is the variance of the Gaussian.

The cost in bits of coding using this distribution can be computed from the optimal coding theorem (Equation 2.4). First, however, we must take care of the fact that we are coding with a continuous distribution. Rather than encoding based on the probability of a particular discrete value, the encoding from a continuous probability distribution must be constructed based on the volume under the density function within a small neighborhood of a value. That is, we must integrate the density function over an interval of values in order to compute the probability *mass* of the interval.

Here we make a *trapezoidal* approximation to estimate the probability mass in an interval of width T around a particular value of r. This allows the mass to be described as the height under the density function at r times the width of the interval (see Figure 2.1), so $\text{mass}(r_i^c) \approx p(r_i^c)T$. The code length is then:

$$|\mathcal{L}^*(x)| \approx -\sum_{i,c} \log[p(r_i^c)T] = \sum_{i,c} {r_i^c}^2/2\sigma^2 + N\log(\sqrt{2\pi\sigma^2}) - N\log T \tag{2.9}$$

where N is the product of the number of residual components and data items. If we take σ and T to be constants, then this results in the familiar summed-squared-error cost function, where the residuals are the deviations between the observed values and those values predicted by the model.

Note that this code-length measure is now expressed in units of $\log_2 e$ bits, called *nats*. For the remainder of the thesis, since these units are linearly related, we will use the term bits for both.

We can select an optimal value of the variance of the Gaussian by minimizing the code length with respect to σ^2. This produces the maximum likelihood value: $N\sigma^2 = \sum_{i,c} {r_i^c}^2$. Using this value in Equation 2.9 results in the following code-length (where K is a constant):

$$|\mathcal{L}^*(x)| \approx N(\log \frac{\sigma^2}{T} + K) \tag{2.10}$$

If we interpret T as the *tolerance* in the encoding of r, where a small value for T corresponds to a high required accuracy in the encoding, then this equation states that the code-length of a value optimally encoded using a Gaussian distribution depends on the logarithm of the ratio of the Gaussian variance to this tolerance. This has the desired intuitive relationship where demanding high accuracy relative to the data variance costs more bits, while an imprecise value is cheap. Note that we can also minimize the cost by shrinking the variance of the data; we will frequently return to this point below.

2.3.3 Sending stochastic codes

We have thus far been assuming that the message contains particular values, such as parameter values, or values under a Gaussian distribution. The sender and receiver have previously agreed upon a description language $\mathcal{L}^*$, which the sender is using to encode these values, and the receiver to decode. In many instances, however, including some explored in the remainder of this thesis, we are interested in a more general case where the sender wishes to communicate codes stochastically selected from a probability distribution across possible values.

We can treat this case using some standard tools from information theory. We first assume that corresponding to the agreed-upon description language $\mathcal{L}^*$ is a prior probability distribution $P(x)$. We then assume that the sender is selecting values from a second distribution $Q(x)$, which we will call the *posterior* probability distribution. We would like to reduce the distance between these two distributions. One such distance measure is the *asymmetric divergence*, or information gain, measure proposed by Kullback (1959).[1] If we consider the two distributions as competing hypotheses, where the hypothesis corresponding to $Q(x)$ is the correct one, then we would like to know the expected amount of information gained in favor of $Q(x)$ over $P(x)$ as a result of observations from $Q(x)$. Kullback showed that this equals:

$$G(P,Q) = \int Q(x) \log \left[\frac{Q(x)}{P(x)}\right] dx = \int Q(x) \left[(-\log P(x)) - (-\log Q(x))\right] dx \tag{2.11}$$

The measure $G(P,Q)$ is the expected number of additional bits required to code a value x drawn from the posterior distribution $Q(x)$ using the prior $P(x)$ rather than the posterior distribution. This is related to the entropy difference between the two distributions. We can see this by examining the two components within the square brackets in Equation 2.11; when these two terms match, no bits are required. This occurs if and only if $P = Q$. Otherwise, $G(P,Q)$ will always be positive. We will use this measure frequently throughout the thesis.

As a sanity check, we note that if the posterior distribution $Q(X)$ is a delta function around a

[1] Note that this is technically not a distance measure, as it is not symmetric, and the triangle inequality does not hold. The asymmetry is appropriate here, as the two distributions are not equivalent; the measure thus describes the distance from the incorrect to the correct distribution.

particular value x, then this distribution has no entropy, and the integral simplifies to $G(P,Q) = -\log P(X=x)$. We thus recover Shannon's optimal coding theorem for this simple case.

We describe this idea of stochastically selecting codes in more detail below (Section 2.5.4), and show that it can lead to a more efficient encoding than simply selecting the most compact code. This integration over possible codes is an example of what Rissanen terms *stochastic complexity* (Rissanen, 1989).

2.4 MDL applications

Before describing how the Minimum Description Length principle may be applied to unsupervised learning, we first describe some recent MDL applications. We consider examples from machine learning, computer vision, and neural networks. This range of examples demonstrate how a variety of prior beliefs can be incorporated into the framework through the class of models and the encoding scheme. While the applications use different forms of MDL, they all employ the basic notion of trading off model complexity against the error codes for data not described by the model. In each case, a simplistic model fails to capture the essence of the mechanism generating the data, leading to bigger error code lengths; while an overparametrized model may contain a lot of redundancy, making it too sensitive to minor irregularities to give accurate predictions of future data.

2.4.1 Machine learning

Here we consider an example of applying MDL to a standard problem in machine learning: inferring decision trees from examples for classification. The goal is to construct a decision tree that takes a set of objects, represented as collections of attribute-value pairs, and classifies them into a fixed set of classes, where a leaf of the tree corresponds to a particular class. The 'best' decision tree is the one with the smallest possible error rate (mis-classification) on previously unseen objects. This situation is an example of the general *over-fitting* problem, in that a perfect tree, which classifies each given object correctly, may not classify new objects as well as an imperfect, smaller decision tree. The MDL principle is naturally suited to this problem, as it trades off the classification ability of a decision tree against its complexity.

Quinlan and Rivest (1989) describe an MDL measure on decision trees where the tree that minimizes this measure is proposed as the 'best' decision tree to infer from the data. They propose an efficient coding technique for the tree by splitting it into 4 parts: tree structure, attribute names and values, and default classes at leaves. The unexplained data here are exceptions to the classifications produced by the encoded tree; these are also efficiently encoded. Note that this is an instance of supervised learning, since the goal is not to reconstruct the data but instead to reproduce the classification. For each item, various efficient encoding schemes are proposed and compared, with the choice being made based on the type of information, e.g., whether the classification problem is binary or multi-valued. The search for the 'best' tree is a heuristic two-phase algorithm, where a tree is first incrementally built up, creating a decision node by trading off its cost against the savings in terms of improved classification. The second phase prunes the tree by eliminating the children of decision nodes whenever this improves the total encoding cost. Thus, the MDL principle is used in both phases, originally to guide the selection of decision nodes, and then to prune the tree. Quinlan and Rivest found that this technique generally out-performed one of the best decision tree procedures. A natural extension to this technique would be to utilize MDL to choose which encoding scheme is appropriate for a given problem.

2.4.2 Computer vision

Image interpretation is another area to which minimum description length methods have been applied. Leclerc (1989) proposes an MDL technique for partitioning an image into regions. His models are drawn from the class of piecewise-polynomial images, i.e., images composed of regions whose intensity function is a two-dimensional polynomial. The image is described as a combination of these piecewise polynomials with additive independent noise with a known distribution. He suggests a succession of increasingly complex encoding schemes for both these terms. The code length of the message, which allows the receiver to reconstruct the intensity image, is linear in the number of polynomial coefficients per region, which encourages a simple, sparse partitioning. The coding scheme also describes how many coefficients are different between neighboring regions, which also encourages continuity between regions. Thus the scheme contains an implicit prior for the partitioning task.

MDL has also been applied to part segmentation (Pentland, 1989). The goal of this system is to recover accurate part descriptions from binary, silhouette images. A vocabulary of blob-like shapes is used to represent potential parts in an image. The system searches for the most likely model, i.e., the collection of these part instances that best account for the image pixel values. The problem is framed in terms of code length by trading off the cost of specifying a pixel value alone versus the cost of specifying a part plus the errors in terms of erroneously encoded pixels it introduces. Pentland uses a very simple encoding scheme, where each pixel is worth one bit, and the code length of each part is some arbitrary constant k. Encoding the pixels independently again indicates an implicit prior that favors segmenting using large parts which can explain several pixels.

2.4.3 Neural networks

In neural networks, minimum description length methods have been applied primarily to the problem of determining the appropriate network size for a given dataset. The ability of a network to generalize from its training set to unseen data is strongly dependent on the number of units and weights in the network (Baum and Haussler, 1989). Several methods have been proposed for determining the optimal network architecture as part of the learning process. These methods generally determine the number of units and weights by introducing extra *regularizing* terms into the network objective function which penalize large networks and large weights, e.g., (Hinton and Sejnowski, 1986; Weigend, Rumelhart and Huberman, 1991). For example, a "weight decay" term is commonly added to the standard summed-squared error function in a supervised back-propagation network in the hope of finding a smoother or simpler network:

$$E \;=\; \sum_{c,k}(y_k^c - d_k^c)^2 \;+\; \alpha \sum_{ij} w_{ij}^2 \tag{2.12}$$

where d_k^c is the desired value of output unit k on training case c, and α is a weighting parameter. One component of the partial derivative of E with respect to a weight is then proportional to the weight itself. Since the weights are typically adjusted in the opposite direction to this partial derivative, this weight decay term encourages smaller weights, or ideally, weight elimination.

MDL, or Bayesian methods have recently been applied to this problem in order to describe a more principled approach to network size selection. MacKay (1992a) describes how Bayesian methods can be used to find the most probable value of α in Equation 2.12. His method compares the "evidence" for different α's on the basis of the weights $w^*(\alpha)$ that minimize E for a particular value of α. The evidence is an "Ockham factor" (see Section 2.2), which roughly describes how sensitive E is to small changes in $\mathbf{w}$ in the neighborhood of the minima $w^*(\alpha)$. The optimal α maximizes the evidence. This quantity also describes how coarsely $\mathbf{w}^*$ can be specified, so this approach could equivalently be formulated in the MDL framework.

A second example of applying MDL ideas to neural networks involves the work of Nowlan and Hinton (1992). They propose a more sophisticated form of weight decay term where a mixture of Gaussians is fitted to the weights of the network. The standard regularization term in Equation 2.12 can be derived as the cost of specifying a weight under the model where the weights are independently generated by a zero-mean Gaussian with variance of $1/\alpha$. For many problems, such as translation-invariant recognition, improved network performance can be achieved by constraining particular subsets of the weights to share the same value. Nowlan and Hinton accomplish this by fitting multiple Gaussians to the weights, allowing the network to decide which weights should be tied together. The standard summed-squared-error function is then augmented with this new second term describing the cost of specifying a weight using the mixture:

$$E = \sum_{c,k}(y_k^c - d_k^c)^2 - \alpha \sum_i \log[\sum_j \pi_j p_j(w_i)] \tag{2.13}$$

where p_j is the Gaussian density of component j of the mixture, and π_j its mixing proportion. This more complex error function can again be seen as an MDL criterion, where the model class has been changed from independent zero-mean Gaussians to a mixture of Gaussians.

Both of these examples use continuous probability distributions to model both the weights and the data-misfit. Kendall and Hall (1993) propose an MDL approach to network construction where the model (network weights) and data-misfit are encoded over a discrete space. The code length of the network parameters is computed by forming a histogram of the weight values; since the task is assumed to be a classification, the code length of the data-misfit is simply the cost of specifying which training cases are incorrectly predicted by the model. The authors use a genetic optimization algorithm to minimize the total description length. They find that on some simple problems, this method succeeds in finding optimal network architectures.

2.5 Applying MDL to Unsupervised Learning

The previous section describes some examples of applying minimum description length methods to neural networks. Each of these applications involves supervised networks, where the model parameters are the weights of the network, and the goal of the decoder is to find the desired outputs of the network. The important point from the standpoint of traditional neural networks is that MDL provides a method of deciding when complexity in the model, whether in the effective number of weights or the precision required to describe a particular weight, is worth the added description cost. Applying MDL allows this model complexity to be expressed in the same units as the data-misfit, which provides a principled way of trading off these terms as well as allowing diverse models to be compared on the same footing.

This section describes a novel application of minimum description length ideas to the unsupervised learning problem in neural networks.[2] The resulting framework lies at the core of the thesis. We first provide an introduction to this framework, then show how it yields an interesting perspective on existing unsupervised algorithms, including many of those discussed in Chapter 2. We also present a Bayesian perspective on this new framework, and provide some detailed examples of how this perspective can be applied to some unsupervised learning problems. Finally, we describe an elaboration of the framework that will be important in the following chapters.

2.5.1 A new framework

In the first chapter (Section 1.4.1), we introduced a new framework for unsupervised learning based on a communication model. We can now view this model in terms of some of the other MDL applications described above.

In MDL for supervised learning, the message describes the weights of the network and the errors at the output layer. The receiver can see the input, so the input cost is zero. Given these pieces of information, the decoder can re-build the network and use it to recover the desired outputs on each training case.

In unsupervised learning, however, there are no desired outputs, only the inputs, or observations. But we can turn the problem around to say that the goal is for the decoder to reproduce the input, so that the input is now the desired output.[3] The question is how to find representations, or codes that are capable of generating the various inputs.

We begin by assuming that the input can be described as the result of some generative procedure. For example, a visual image can be described as having been generated by light falling on a scene which contains particular objects. One possible approach to the learning problem then is to directly find the representations that can generate the observations. However, this is a difficult learning problem. Many different models could generate a given data point, so the search space is large; in addition, we do not know the generating function. We have a given observation y^*, which we assume is generated by some function g, such that $y^* = g(\mathcal{A}^*)$. Our goal is to find the representation $\mathcal{A}^*$ that is capable of generating y^*. The problem is that neither the function g, nor its inverse (if it exists) is known *a priori*. Jordan and Rumelhart (1992) describe the difficulty of learning this kind of *generative mapping* for a different problem, sensorimotor learning.

We adopt the approach of allowing the system to learn the representations $\mathcal{A}$ by training a back-

[2] It should be noted that others have applied MDL to specific unsupervised learning problems, notably clustering, e.g., (Segen, 1990).

[3] This approach is related to the work on Pattern Theory (Grenander, 1982; Mumford, 1992), where the main idea is that in order to recognize patterns, a system should learn to synthesize them.

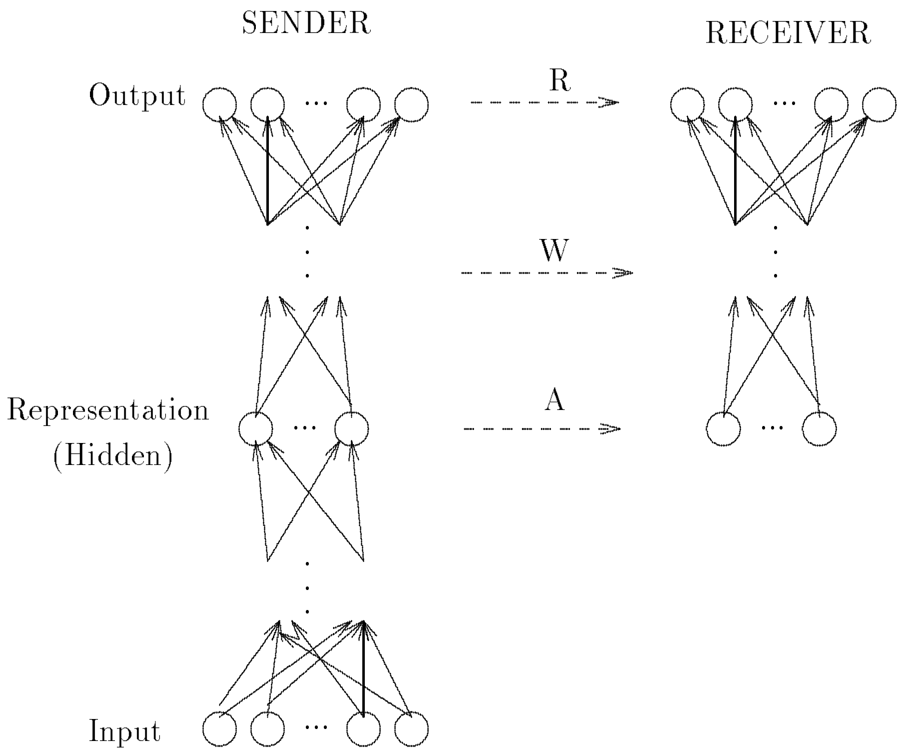

Figure 2.2: This figure shows the information that the sender must communicate to enable the receiver to recover the input. The sender uses a self-supervised network; the receiver can construct the top half of this network based on the weight values $\mathcal{W}$, and then uses the activities $\mathcal{A}$ of the representation, or hidden units, together with the reconstruction errors $\mathcal{R}$ to reconstruct the input. The cost of encoding each of these three terms is the objective function for the sender's self-supervised network. Ignoring the weight-cost, the sender trains the network by back-propagating the derivatives of the reconstruction-cost $C_{\mathcal{R}}$ from the output units, and then adding in the derivatives of the representation-cost $C_{\mathcal{A}}$ at the representation layer.

propagation network to reproduce its inputs. In this self-supervised network, the activity vectors of the hidden units are the representations $\mathcal{A}$, and the outgoing weights implement the function g. This approach suggests a different view of self-supervised networks, or autoencoders: the weights from the inputs to the hidden units facilitate the search for a generative model of the inputs.

Once we frame the problem in terms of a self-supervised network, we can easily apply MDL. The receiver requires three items in order to be able to recover each element of the training set (see Figure 2.2). The first element is the set of activities $\mathcal{A}$ of the representation units in the network across the training set; these are the codes that the network assigns to each case. The second element is the set of weights $\mathcal{W}$ connecting these representation units to the output units. The receiver can apply the transformation specified by these weights to $\mathcal{A}$ to recover the output of the self-supervised network for each case. This acts as a prediction for the input. So the third element is the reconstruction error $\mathcal{R}$—the misfit between the output and input. The receiver uses this misfit then to recover the actual inputs.

The cost associated with encoding each of these three terms—$\mathcal{A}$, $\mathcal{W}$, and $\mathcal{R}$—then comprise the total description length of the message:

- The **representation-cost** $C_{\mathcal{A}}$ is the number of bits required to communicate the representation $\mathcal{A}$ that the algorithm assigns to each input vector.

- The **model-cost** $C_{\mathcal{W}}$ is the number of bits required to specify how to reconstruct input vectors from these representations (e.g., the hidden-to-output weights $\mathcal{W}$ in Figure 2.2).

- The **reconstruction-cost** $C_{\mathcal{R}}$ is the number of bits required to fix up any errors $\mathcal{R}$ that occur when the input vector is reconstructed from its representation.

Several important aspects of this formulation should be emphasized. First, the communication model is only a device that we use to derive an MDL objective function which can be used to train a self-supervised network. We are not actually interested in sending the 3-part message, but rather in training a network to develop good representations of a dataset. Second, the network may have an arbitrary number of layers between the input and representation layers, and between the representation and output layers. In the rest of the thesis, we refer to the network as if it only contains these three layers, but the formulation applies directly to deeper networks. Finally, unlike earlier applications of MDL to neural networks (see Section 2.4.3), this method entails encoding only the set of weights $\mathcal{W}$ between the representation and output layers, as opposed to all of the weights of the network. This is due to the fact that we encode the activities of the representation units, which avoids the need to communicate the input-to-representation weights.

2.5.2 Bayesian perspective

Just as standard MDL can be viewed from a Bayesian perspective, we can usefully view the MDL unsupervised learning framework from this perspective as well. We let $\mathcal{W}$ be the set of weights from the hidden-to-output units, $\mathcal{A}$ be the activities of the representation units on a training case, and X be the input vector on a training case. Bayes' Rule states that:

$$p(\mathcal{W}, \{\mathcal{A}\}|\{X\}) = \frac{1}{p(\{X\})} p(\{X\}|\mathcal{W}, \{\mathcal{A}\})\, p(\{\mathcal{A}\}|\mathcal{W})\, p(\mathcal{W}) \tag{2.14}$$

We assume that $p(\{X\}) = 1$, so:

$$p(\mathcal{W}, \{\mathcal{A}\}|\{X\}) = p(\{X\}|\mathcal{W}, \{\mathcal{A}\})\, p(\{\mathcal{A}\}|\mathcal{W})\, p(\mathcal{W}) \tag{2.15}$$

The three terms on the right-hand side of this equation correspond to probabilities for the three parts of the message that allows the receiver to recover the input. The third term is the probability of the set of outgoing weights $\mathcal{W}$. Since the representation activities do not depend on their outgoing weights, the second term, $p(\{\mathcal{A}\}|\mathcal{W})$, is just the probability of the representation activities $\{\mathcal{A}\}$. The first term, $p(\{X\}|\mathcal{W}, \{\mathcal{A}\})$, is the probability of the input given these other two; since the receiver can use $\mathcal{W}$ and $\{\mathcal{A}\}$ to compute the network output, this amounts to the probability of the desired output of the network (the input) given its actual output.

Corresponding to each of the three parts of the message is therefore an all-important prior distribution. This prior distribution embodies the assumptions about the likely form of the relevant term as well as the encoding scheme. We discuss this in the following section.

If we assume that the sender and receiver agree on these three prior distributions, the total cost C of the message is the sum of the reconstruction-cost, the representation-cost, and the model-cost:

$$\begin{aligned} C &= -\log p(\mathcal{W}, \{\mathcal{A}\}|\{X\}) \\ &= -\log p(\{X\}|\mathcal{W}, \{\mathcal{A}\}) - \log p(\{\mathcal{A}\}) - \log p(\mathcal{W}) \\ &= C_{\mathcal{R}} + C_{\mathcal{A}} + C_{\mathcal{W}} \end{aligned} \tag{2.16}$$

2.5.3 Implementing the MDL objective function

The total cost C can be used as the objective function of a self-supervised network. Various learning algorithms can be defined by selecting different forms of prior distributions for these three terms. It is therefore in the specification of these priors that assumptions and knowledge about the structure of a dataset may be expressed. For example, if we know that the values are binary, a beta distribution will be more appropriate than a Gaussian. In addition, we can exercise control over the types of representations that the network will develop by selecting the parameters in these various distributions. For example, if one chooses a Gaussian coding prior for the reconstruction error (as described in Section 2.3.2), then by adjusting the variance of this coding Gaussian, we manipulate the degree to which this term will be emphasized relative to the other terms in the cost function. Thus, the variance acts as a regularizer, trading off representational accuracy against succinctness. The number of parameters in these prior distributions that are user-defined (as opposed to adaptively determined by the algorithm) determines how much additional control we have in shaping the learned representations. These priors and their parameters are described in more detail in Chapters 4-6 below.

With respect to the basic cost function $C = C_{\mathcal{R}} + C_{\mathcal{A}} + C_{\mathcal{W}}$, the algorithms introduced in this thesis concentrate on the first two terms. The third term describes the cost of communicating the weights of the network. Note that this can be done once before the training set is seen, while the other two terms must be specified for each case. Thus, in the high data limit, the model-cost ($C_{\mathcal{W}} = -\log p(\mathcal{W})$) is negligible. We will therefore ignore this term in the remainder of the thesis.

The representation-cost only depends on the activities of the hidden, or representation units. The algorithms described in the thesis are all implemented by back-propagating the derivative of the reconstruction-cost from the output units, and then adding in the derivatives of the representation-cost at the hidden layer, in order to compute the derivatives of the cost function with respect to the weights (see Figure 2.2).

2.5.4 An elaboration: Communicating stochastic representations

We describe various alternative forms for the two costs—representation and reconstruction—in the following chapters. In this section we present some simple examples for each, and focus on an elaboration that is particularly relevant to encoding the representation-cost.

The reconstruction-cost $C_{\mathcal{R}}$ we generally use is the common summed-squared-error, which corresponds to the example of coding based on a Gaussian distribution described in Section 2.3.2. This coding scheme is applied to the network by assuming that the sender and receiver agree on a prior consisting of an independent zero-mean Gaussian for each output unit. The variance of each Gaussian can either be fixed, or as was discussed in that section, the sender may optimize it based on the observed residuals (the difference between the output and desired output, or corresponding input of the relevant unit, computed on each training case). In either case, the sender must communicate these residuals, and the resulting cost is a function of the sum of the squares of the residuals.

In this case, we imagine that the sender is communicating a single value, here the error for a particular output unit. Often, however, the sender may actually wish to communicate values based on a probability distribution over a set of possible values. This is especially relevant when we consider networks in which the units are stochastic, or at least represent stochastic quantities. For example, assume that a stochastic binary unit j has an associated binary state $S_j = \{0, 1\}$. Then the unit's activity level $y_j \in [0, 1] = p(S_j = 1)$ is the expected activity of j across stochastic samples. We would then like to communicate the binary activity of unit j according to this probability distribution.

This concept of communicating values from probability distributions is particularly important with respect to describing the representations formed by the network. In many situations it is appropriate to encode the activity representations $\mathcal{A}$ as discrete-valued. However, searching is much more difficult in a discrete space rather than a continuous one. One reason for this is that many search algorithms rely on gradient information, which is not available in non-continuous spaces. We would therefore like to convert the discrete values, such as the binary states S_j, to continuous ones, such as the expected values y_j. Then the encoding scheme must take into account the fact that these values represent probability distributions.

In addition, at the beginning of the search for succinct, informative representations for an input set, we would like the network to be able to represent its lack of knowledge as to a good representation for a particular training case. Since the search is not guaranteed to get rid of this uncertainty, in order to make the problem well-defined, we need to take into account this uncertainty when calculating the cost of communicating the representation; encoding stochastic representations allows us to do this. Thus stochastic representations are generally more informative and easier to search through than single-valued representations.

In order to estimate the cost of communicating representations based on a distribution across $\mathcal{A}$, we apply the asymmetric-divergence measure (Equation 2.11) as described in Section 2.3.3. Staying with the previous example, we assume that the representation units are independent Bernoulli variables. The sender and receiver agree on a prior distribution P for these variables. In the simplest case, P is also a Bernoulli distribution with an expected value of 0.5, indicating no prior bias to the unit being on or off: $P = \{0.5, 0.5\}$. On each training instance, the sender uses the self-supervised network to compute the expected activity y_j of each representation unit j. This activity value then defines the posterior distribution $Q = \{y_j, 1 - y_j\}$. The additional cost of communicating values from Q using the agreed-upon prior P rather than Q is thus:

$$\begin{aligned} C_{\mathcal{A}} &= G(P, Q) = \sum_a Q(a) \log \frac{Q(a)}{P(a)} \qquad (2.17) \\ &= Q(S_j = 1) \log \frac{Q(S_j = 1)}{P(S_j = 1)} \; + \; Q(S_j = 0) \log \frac{Q(S_j = 0)}{P(S_j = 0)} \\ &= y_j \log y_j - y_j \log .5 + (1 - y_j) \log(1 - y_j) - (1 - y_j) \log(1 - .5) \end{aligned}$$

$$= \quad 1 - H_Q$$

where a indexes the possible activity values of unit j.

This simple example illustrates the computational benefits of communicating stochastic representations. The cost of describing this distribution includes a *reward* equal to the entropy of the posterior distribution (H_Q in Equation 2.17). The important effect of this reward to the sender is to encourage high-entropy distributions across representations, which aids the search for good representations by smoothing the search-space.

We will see in the following chapters that other, more complex kinds of distributions over representations can also be represented and communicated. In each case, the cost function may be derived from the asymmetric divergence measure.

A more intuitive argument may clarify why the sender obtains a reward when she communicates stochastic representations. The intuition is that the precise specification of a representation for an input will require more bits than a less precise description. If we assume a coding prior that makes two representation vectors equally likely, then when the vectors are equally good representations for the input, this representation should be cheaper to encode (by one bit) than one in which only one is good. Since it does not matter which representation we communicate, we can save bits by being vague.

This intuition can be made precise by considering the flow of information in our communication model. We assume that the sender is already communicating a large number of random bits to the receiver, and we want to compute the *additional* cost of communicating the input vectors. The sender uses the random bits to stochastically choose a sample representation a_0 from the posterior distribution $Q(\mathcal{A})$. She allows the receiver to reconstruct the input by sending a_0 and the corresponding reconstruction error. The sample representation is encoded using the prior distribution $P(\mathcal{A})$, which allows the receiver (who only knows P and not Q) to decode it. Assuming that the representations are discrete-valued, the cost of this encoding is:

$$-\log P(\mathcal{A} = a_0) + C_{\mathcal{R}}(\mathcal{A} = a_0) \tag{2.18}$$

where $C_{\mathcal{R}}(\mathcal{A} = a_0)$ is the reconstruction-cost based on that representation.

The sender pays this cost for each example in the training set, allowing the receiver to reconstruct each input example. Once the receiver has collected all of these actual inputs, he knows the inputs and outputs of the network, and can run the same training algorithm used by the sender to calculate the rest of the weights in the network, namely the weights from the input layer to the representation layer. On each case then, the receiver can recover the actual posterior distribution $Q(\mathcal{A})$ for the representation activities. Since he already knows which representation was chosen, he can figure out the random bits that the sender used in selecting the sample a_0 from $Q(\mathcal{A})$. He thereby recovers the cost of selecting a_0:

$$-\log Q(\mathcal{A} = a_0) \tag{2.19}$$

So, allowing for the fact that these random bits have been successfully communicated, the true expected combined cost is equivalent to the expectation of the difference between Equations 2.18

and 2.19 over the posterior distribution:

$$C = \sum_a Q(a) \left[\log Q(a) - \log P(a) + C_R(a)\right] \tag{2.20}$$

The total cost is therefore the sum of the asymmetric divergence $G(P, Q) = Q(a) \log[Q(a)/P(a)]$ and the expected reconstruction-cost. This latter term brings out the fact that we must fully take into account the effects of stochastic activity levels for the representation units. Noise in the representation units translates to noise in the output units of the network. We will see in the following chapters that this leads to an important trade-off between bits saved in imprecisely specifying $\mathcal{A}$ versus bits required in $\mathcal{R}$ to make up for the increased reconstruction error.

The key idea here is that this stochastic communication of representations leads to more efficient codes than the standard approach of simply choosing the representation with the smallest reconstruction error. The stochastic method wins based on the entropy of the posterior distribution, which corresponds to the intuitive notion of less precise codes being cheaper.

Finally, as a further elaboration of the simple example described above, imagine that we adapt the prior distributions for the stochastic representation units. Rather than just assuming an expected value of 0.5, we may save bits by actually calculating the expected values of each unit and using these values in our coding. In this way, since some units will likely be on more often than others, we can save bits by making the prior distribution have an expected value $\overline{y}_j$ that corresponds to the expected value of that unit across the training set. The cost then becomes:

$$C_A = -y_j \log \overline{y}_j - (1 - y_j) \log(1 - \overline{y}_j) - H_Q$$

This cost is minimized if the actual activity y_j matches the expected activity $\overline{y}_j$.

Note that this modification introduces a non-standard definition of a prior distribution, where the prior is a function of the data. We then have two levels of prior distributions. The first corresponds to a Bayesian *hyper-prior* (Jaynes, 1986), specifying probability distributions for the parameters of the second, the actual *coding-prior*. The hyper-priors together with the data then determine the best parameters of this coding-prior. The parameters of the coding-prior also have to be communicated from the sender to the receiver, who only knows the hyper-prior. Since there are relatively few of these parameters as compared to the other message elements, however, we generally do not include the cost of specifying these parameters in the optimization.

Chapter 3

Unsupervised Learning: Representations and Algorithms

In this chapter we review some basic forms and underlying ideas in the area of unsupervised learning. We first discuss basic motivations and general types, then focus on the particular algorithms that are directly relevant to the work described in the remainder of the thesis.

3.1 Why unsupervised learning?

In an unsupervised learning network, there is no feedback from the environment as to whether the outputs of the network are correct. The network must discover for itself regularities, correlations, or structure in the input data and represent them in the output.

This *task-independent* type of learning is useful in many situations. Interesting and difficult mapping problems generally involve several logical processing steps between the input and desired system output, and therefore call for hierarchical, multi-layer networks. The standard method of supervised learning, back-propagation, is slow in multi-layer networks. This is largely due to problems with parameter interactions, as the effect of modifying one weight can potentially depend on every other weight in the network.

One way of alleviating this problem is to perform one or more rounds of unsupervised learning on the input patterns of a large problem, sequentially extracting structure a layer at a time. Training some layers simply based on the outputs of the immediately preceding layer allows for a much more efficient construction of deep, hierarchical networks.

More generally, identifying regularities in the input data may ease the complexity of further processing. In addition, the representations formed by unsupervised learning procedures are typically of a lower dimensionality than the input. This *dimensionality-reduction* can be very important in avoiding the curse of dimensionality, which states that the required number of samples grows exponentially with the dimensionality of the space, so the amount of training data required to discover a desired mapping is impractical in high-dimensional spaces. Dimensionality-reduction may therefore be useful as a precursor to both supervised, and other unsupervised learning procedures.

Finally, another standard criticism of supervised learning procedures is their lack of biological plausibility. It is unreasonable to assume that an organism is provided with a direct training signal for every action. In addition, multi-layer back-propagation requires that specific kinds of very detailed information would have to be carried in cortical projections, and it would have to propagate through several cortical layers.

3.2 General background

This section presents a general overview of unsupervised learning methods. Excellent surveys of unsupervised techniques can be found in (Becker, 1989; Hertz, Krogh and Palmer, 1991), so we only introduce some of the main paradigms and briefly highlight the basic approaches. Particular relevant algorithms are described in detail in the following section.

The general goal of unsupervised learning is to construct new representations of the input data that capture its underlying structure. The form of these representations greatly depends on the architecture of the network as well as the training algorithm. These algorithms typically embody, implicitly or explicitly, assumptions about the nature of the structure underlying the input data. We can therefore view and characterize different unsupervised techniques by the sorts of representations they produce.

3.2.1 A spectrum of desired representations

A key dimension along which various representations can be described is their degree of locality. At one extreme along this dimension lies a purely local representation, in which a single unit represents an entire data vector. The other extreme is a fully distributed representation, in which all units are used to represent each data vector.

The advantage of a local, or "grandmother-cell" representation is that several data items can be represented simultaneously. In addition, it is typically easier to learn with local representations, because fewer parameters are involved. Also, fully distributed representations suffer from crosstalk when more than one item is active, and they require a higher degree of connectivity. The advantages of a fully distributed, partially-redundant representation, however, are that various combinations of representation variables offer higher accuracy in describing a single data vector (Hinton, 1986) and many more vectors can be separately represented for the same number of units. Also, purely local representations are costly in terms of hardware, as they require a unit for every possible datum, and are not robust to degradation or failure, as losing a unit can entirely corrupt the representation of a data item.

Unsupervised learning methods can be divided into two broad classes based on these two extremes of desired representation locality. The two classes can be visualized by considering the data distribution, where each datum of length N is a point in some N-dimensional space. The first set of algorithms, *projection methods*, attempt to define interesting directions in this space on which the data points can be projected. Projection methods generally produce fully distributed representations, as each data point is described as a combination of the component directions. These methods can be implemented in a 2-layer (input and output) network, where the weight vector into a given output unit constitutes a projection axis; computing the inner product of the input and the weight vectors then projects the data onto these component directions.

The second set of algorithms are *kernel methods*. These methods typically involve some form of clustering, where the input is approximated by one of a fixed set of points. This method can be implemented in a network where the N-dimensional weight vector into a given unit in the next layer is viewed as a point in the input space. Data points are then associated with particular unit(s) according to their distance to the weight vector of that unit. Units that are far away from a data point play no role in representing that point. Kernel methods typically produce local representations.

3.2.2 Projection methods and PCA

Many of the standard unsupervised learning methods are related to the common statistical method of *principal components analysis (PCA)*. This method computes the principal components, or the eigenvectors of the covariance matrix (e.g., the covariance of the pixels in an image). The corresponding eigenvectors represent a hierarchy of orthogonal coefficients where the vectors corresponding to the largest eigenvalues account for the greatest part of the covariance. This method, also known as the Karhunen-Loève transform, is equivalent to maximizing the information content of the output signal where that signal has a Gaussian distribution, or alternatively, minimizes information loss as measured in RMS error (Linsker, 1988). The method can be used to form *compact codes* for each input, as typically only the M most significant components, or the eigenvectors corresponding to the first M eigenvalues are retained, and the dimensionality of each input sample is reduced by projecting the N-dimensional input onto this M-dimensional subspace. Note that this projection is linear, and is thus limited in the types of structure it can capture; below we discuss a non-linear analog of PCA.

Hebbian learning

Several learning rules have been described which will compute these principal component directions, or a rotated subspace of these directions. The first set uses a *Hebbian* learning rule, where the strength of a weight connecting two units is proportional to the coactivity of the units. This learning rule is directly based on maximizing the information content of the network. In the Gaussian case, the amount of information in a Gaussian random variable is proportional to the logarithm of its variance, so maximizing the variance is equivalent to maximizing the information content of the signal. The objective can be stated as maximizing the output variance, V. To show how this objective leads to a simple learning rule, with several important variations, we begin with a simple case with one linear output unit. Denote the input vector to that unit $\mathbf{X}$, the weight vector into the unit $\mathbf{w}$, and its output Y.

$$V \;=\; < Y^2 > \;=\; < (\mathbf{w}^T \mathbf{X})^2 > \;=\; \mathbf{w}^T \mathbf{Q} \mathbf{w} \tag{3.1}$$

where $\mathbf{Q}$ is the input covariance matrix.[1] Maximizing the objective V by gradient descent on the weights $\mathbf{w}$ produces the following Hebbian learning rule:

$$\Delta w_i \;=\; \epsilon \frac{\partial V}{\partial w_i} \;\propto\; \epsilon < X_i Y > \tag{3.2}$$

where ϵ is a learning rate. Note that this rule alone will allow $\mathbf{w}$ to grow unboundedly. Additional constraints must be added in order to limit the magnitude of $\mathbf{w}$.

One particularly elegant solution to this problem was proposed by Oja (1982). He proposed a modified learning rule that makes the weight vector approach unit length:

$$\Delta w_i \;=\; \epsilon Y (X_i - Y w_i) \tag{3.3}$$

This second term is a form of *weight decay*, which limits the magnitude of the weight vector. Oja proved that this learning rule not only maximizes the variance of the output unit, but it will also allow $\mathbf{w}$ to be the first principal component of $\mathbf{Q}$, except when this direction is orthogonal to the initial weight vector.

Oja (1989) extended this approach to a network with multiple output units. In this case, the

[1] This description assumes that both the input and the output have zero mean. This simplifies the discussion, but the ideas still hold for the more general case.

weight decay term depends on the activities of the other units in the output layer:

$$\Delta w_{ji} = \epsilon Y_j (X_i - \sum_{k=1}^{M} Y_k w_{ki}) \tag{3.4}$$

where w_{ji} is the weight from unit i to unit j. This learning rule maximizes the sum of the variance of each output unit, which is equivalent to maximizing the information content of the output signals under the assumption that the signal is composed of independent Gaussians. The resulting weight vectors span the subspace of the first M principal components of $\mathbf{Q}$. Sanger (1989) described a variation on this learning rule which allows the weight vectors to converge to exactly the first M eigenvectors, in descending eigenvalue order. In Sanger's algorithm, the weight vectors are computed sequentially, which allows the network to perform a Gram-Schmidt orthogonalization of the input correlation matrix.

Self-supervised networks

A second form of network architecture that produces fully distributed representations is the *self-supervised*, or *autoencoder* network discussed in the previous chapters (see Figure 1.2). Here we describe the general background and utility of autoencoders. In an autoencoder, the goal is to store a collection of patterns so that they can be reconstructed as accurately as possible (Hinton, 1989a). The network is typically trained by back-propagation, where the desired output of a unit is the activity of the corresponding input unit. The simplest such network contains a single hidden layer, which typically consists of $M < N$ units (where N is the number of input and output units). This layer acts as a bottleneck, forcing the network to find a compact, invertible encoding for each input vector. These codes are fully distributed, in that each code unit participates in each encoding.

These networks are clearly useful for data compression; they may also be used for pattern completion, where one of the patterns is retrieved when only part of it is presented. In addition, the bottleneck can force the network to learn interesting features of the input distribution which permit generalization to patterns that were not in the training set.

Self-supervised networks with linear units are closely related to PCA. Cottrell, Munro, and Zipser (1989) empirically studied the performance of self-supervised networks on image compression. They found that the 16 hidden units spanned a subspace that was very close to that spanned by the first 16 principal components of the input space. The primary difference was that the amount of information conveyed, i.e., the variance, was approximately the same for all of the hidden units. Individual hidden units were not allocated to each principal component, but the set of units spanned the same space. Baldi and Hornik (1989) analytically showed that the error surface in a linear 3-layer self-supervised network has no local minima, and has a minimum corresponding to the projection onto the subspace generated by the first principal vectors of the input covariance matrix.

Non-linear self-supervised networks can learn representations that differ from PCA. Zipser (1986) showed that non-linear autoencoders can discover more local representations. Using two hidden units to encode an ensemble of images, each consisting of a single Gaussian spot in varying positions, the hidden units form a distributed code, each unit conveying the position along an orthogonal dimension. With more hidden units, the representation becomes increasingly local, with each hidden unit having a limited receptive field.

3.2.3 Kernel methods

The second category of basic unsupervised learning algorithms tend to form more local representations. Kernel methods attempt to associate a given input case with a particular class. Thus one way of distinguishing these two categories is that in projection methods, the units *cooperate* to represent the input, while in kernel methods, the units *compete*. A network associated with a kernel method is generally a simple two-layer network, input and output, and each output unit represents a different class. The goal is to classify or cluster the input data, so that similar inputs should be associated with the same output unit. The key point is that the network must discover the classes from the statistics of the inputs.

We discuss in detail several specific kernel methods in the following section. The algorithms are often variants of statistical clustering algorithms. Here we introduce the simplest form of kernel method—a *competitive learning* network where the output units compete to account for each input case. Each of M output units are directly connected to the N input units, and the total input to an output unit j is the distance between an input vector $\mathbf{X}$ and its incoming weights $\mathbf{w_j}$. The network then computes a winner-take-all (WTA), so that the output unit with the smallest input has activity 1, while the others have activity 0.

There are a number of different ways of implementing this WTA competition. The simplest way is to use a non-linear activation function that properly sets the activities. Rumelhart and Zipser (1985) use this method, and propose the following learning rule, where only the incoming weights of this winning unit j are modified by:

$$\Delta w_{ji} \;=\; \epsilon\left(\frac{X_i}{\sum_k X_k} - w_{ji}\right) \tag{3.5}$$

This rule accomplishes two important goals. Stability is achieved by moving the winning unit's weights closer to the input case, which makes it likely that the same unit will win the competition for that case the next time. Diversification is achieved by normalizing the input magnitude X_i, which prevents the weight magnitude of $\mathbf{w_j}$ from increasing without bound and dominating the competition for all inputs.

A second way of implementing a form of WTA competition (i.e., with a few winners) is by lateral inhibitory links amongst the units in the output layer. This type of lateral connectivity can be interpreted as inducing a geometrical arrangement of the output units. In von der Malsburg's model (1973), spatial locality in this *induced space* is enforced through fixed lateral connectivity profiles that roughly form a "Mexican hat", i.e., decaying strength over distance, with positive links to the closer units, and negative links to units further away. The connections from the input to output units are adapted by a Hebbian learning rule, where the coincident activation of the two cells causes an increment in the connection strength. The weights are normalized so that the total strength of the incoming weights remains constant, thereby preventing unbounded growth in synaptic strength. Von der Malsburg showed that when the system is trained with oriented line segments, the excitatory cells tend to develop receptive fields that look like oriented line segments that model the retina. In addition, adjacent cells tend to develop similar orientation preferences, and these preferences change smoothly across the surface. We discuss this model below in Section 3.3.3.

Even with this learning rule, an important problem with this simple form of competitive learning is that units with $\mathbf{w_j}$ far from any inputs might never win a competition, and will therefore be wasted. There are a number of ways of handling this problem of *dead units*, including penalizing frequent winners (Grossberg, 1976; Bienenstock, Cooper and Munro, 1982) and "softer" versions, where other units in addition to the winner have their weights updated, based on their distance to the input (Rumelhart and Zipser, 1985; Nowlan, 1990). Nowlan showed that soft competitive learning is superior to the traditional WTA form in terms of both performance and learning speed on two classification tasks, digit and vowel recognition. Softer versions allow more than one unit to stochastically participate in the representation of an input.

Kernel methods are useful for classification and categorization. The network must discover similarities among the inputs that are represented by a single unit, and differences in inputs represented by different units. This discovery of structure in the inputs should minimize interference for future learning, and permit more modular learning.

3.2.4 Discussion

An important point with respect to this thesis is that both kernel and projection methods are intrinsically tied to their particular encoding schemes, i.e., projection methods encode each projection axis independently, while kernel methods encode based on a 1-out-of-N scheme. While we would like to find an unsupervised method that is capable of extracting structure in the data, it is unlikely that a single encoding scheme will be able to do this for an arbitrary input distribution. Each of the approaches mentioned above can be expected to perform well on some distributions and poorly on others.

Projection methods can capture structure in input distributions when a few directions account for most of the variance. For example, if we think of the data as an ellipsoid, then PCA provides a method of finding the axes of the ellipsoid. Kernel methods are more appropriate when the data consists of isotropically distributed clusters, which would tend to be obscured by PCA and related methods. We therefore must make some *a priori* assumptions about the data structure, and the desired representation, when choosing a method for coding the input distribution.

3.3 Three forms of desired representations and algorithms

In this section we focus on the three types of desired representations that will be explored in the remainder of thesis. For each type, we describe some existing learning algorithms.

3.3.1 Minimally redundant representations

One type of representation that has been a common goal of unsupervised learning procedures is a *minimally redundant representation.* Barlow (1961; 1983; 1989) proposed the computational and biological utility of this representation scheme. He suggested that the redundant aspects of input patterns provide insight into the regularities in the environment; a good representation should capture and remove this redundancy from the input. One method of reducing redundancy is through *decorrelation*. Barlow and Földiák (1989) suggest a method of learning decorrelated representations, where each unit's output feeds back to the other inputs through an anti-Hebbian rule, so that correlated activity among the output is discouraged.

As the amount of redundancy decreases, the representation approaches a *factorial* representation, in which the probability of the activity of each representational element is independent. Factorial representations have several advantages. The prior probability of the occurrence of any input can be computed simply as the product of the probabilities of the individual features that represent it. In addition, since correlations have been removed from the representations, any dependence between two features indicates a novel situation (Barlow, Kaushal and Mitchison, 1989). Independent features can also make later learning easier. If the input units to a linear supervised learning network are uncorrelated, then the Hessian of its error function is diagonal, allowing the application of efficient methods for accelerated learning (Becker and Le Cun, 1989).

Various algorithms for discovering factorial representations have been proposed. A common theme in these algorithms is that of a *minimum entropy* representation: an invertible representation which minimizes the sum of the units' entropies. It turns out that minimizing the sum of the entropy of the representation units will lead to factorial codes; this is proved in Watanabe (1985). The intuitive idea is that correlations between the units indicates redundancy in the code. The total entropy in the codes equals the sum of the information in the code plus this redundancy. If we assume that the information in the representation is constant, then minimizing the total entropy in the code will therefore eliminate these correlations. Eliminating all of the redundancy will result in independent representation units, and hence a factorial code.

Barlow, Kaushal, and Mitchison (1989) describe a detailed example of this approach, considering the representation of keyboard characters in 7 binary digits. The average entropy in each bit i of the representation is:

$$H_i = -(P_i \log P_i + Q_i \log Q_i) \tag{3.6}$$

where P_i is the probability that ith bit is 1, and Q_i is $1 - P_i$. The average entropy of the characters is:

$$H_c = -\sum P_c \log P_c \tag{3.7}$$

where P_c is the probability of character c. Adding the 7-bit entropies will always be greater than or equal to H_c, due to correlations in the bit values. The two estimates will be equal if and only if the values are independent. Minimizing the sum of the H_i will therefore squeeze the redundancy out of the representations.

The authors suggest several methods for achieving this, and compare it to a test set. The first attempts to minimize the average probability of a bit being active; this produces a sum of bit entropies not far from the true character entropy. A second method that lists input state probabilities and numbers them in an ordinary binary sequence produces a lower, nearly optimal sum. Both of these compare quite favorably to the ASCII representation. While both of these techniques tend to approach factorial representations, neither quite achieves this goal. Minimizing the sum of bit probabilities will produce the lowest average number of active outputs, but it does not necessarily minimize the sum of bit entropies. The problem with the second method is that it will produce a good approximation to a factorial code only when the probabilities of the input states form an approximate geometric progression. While these methods involve heuristic serial search, Földiák (1990) showed how a network where the feed-forward connections are trained by a normalized Hebbian rule, and the feed-back connections by an anti-Hebbian rule can learn a representation that approximates a minimum-entropy code, as it reduces the statistical dependency between features while preserving information about the input.

Nowlan (1988) proposed a different method for discovering factorial representations. He employed a self-supervised network, where the requirement that the network reconstruct the input set forced the representations to retain the information in the data. The network consisted of stochastic binary units, where the activity level p_i of unit i indicates the probability that it is active. Independence among the representation units was encouraged by also minimizing the sum of the entropies of the hidden units of the network. With an additional constraint encouraging the hidden units to carry an equal amount of information, this method produced representations with decorrelated elements for some simple test cases. This method is discussed in more detail in Chapter 4.

An alternative method for finding factorial representations is proposed in Schmidhuber (1992). This scheme involves training a network consisting of several sub-networks. For each representation unit, a separate predictor sub-network is trained to predict the value of that unit given the other representation units' values. Meanwhile, a representation sub-network for each unit is trained to extract properties from the input that allow it to be *un*predictable. Moving a unit's value away from its predictions will move it away from its expected value; this forces it to be informative, and with continuous valued-units, also encourages it to be binary. This *predictability minimization* scheme thus encourages the units to represent environmental properties that are statistically independent from those represented by the other units. While this method entails a complicated training procedure which alternates between updating the predictor and representation sub-networks, simulations demonstrate that it produces near-factorial codes for some simple examples.

A related type of representation that has also been a common goal of unsupervised learning procedures is a *sparse distributed representation*, in which a set of binary units faithfully encompass the space of input data such that each unit participates in representing a limited subset of data items, and a few active units represent each data item. This representation is related to the idea of minimum entropy codes, since a minimum-entropy binary unit will either be typically active or inactive. If we assume the latter, then a minimum entropy code will also be sparse, i.e., a limited set of cells are active for any given item, and each cell is active for a limited set of data, there amount of redundancy in the coding is restricted.

This form of representation can be understood by viewing it in terms of the two extremes of coding represented by local and distributed representations. Distributed representations produce compact codes, using relatively few variables to represent the set of data items, while local representations use many variables, but only one is used to represent a particular input. Sparse distributed representations lie between these two. The hypothesis is that sparse distributed representation may share the faster learning properties and ability to represent multiple items simultaneously of local representations, and the precision and efficiency of distributed representations, without too much of the interference and abundance of connections of fully distributed representations. Furthermore, if a sparse distributed representation can also capture important underlying regularities in the task domain, it can lead to much better generalization.

Sparse distributed representations have also been suggested from the standpoint of describing visual coding—the response properties of cells in the mammalian visual pathway. Barlow posits that the purpose of natural image processing is "to represent visual scenes by activity of a sparse selection of reliable and nonredundant (i.e., independent) elements" (Barlow, 1983, p. 12). He points out that in sensory cortical areas, many neurons are arranged into retinotopic/topological maps, so that for any local stimulus, a few neurons will typically respond. He also suggests that neural cells in higher areas of cerebral cortex are less often active than inactive, and the number of neurons active for each stimulus is low. Field (1987) shows how the frequency selectivity of cortical cells are well-suited to the statistics of natural images, allowing an image to be represented in a few active cells. The high-order redundancy in pixel intensities across a set of images (i.e., correlations between pairs or triplets of pixels) is transformed into first-order redundancy in the form of the response distribution of the individual cells.

Several authors have argued the virtues of minimally redundant representations; most of these incorporate the idea of minimum entropy. A few unsupervised learning procedures have been proposed for discovering representations that fit this description, and have been demonstrated on a limited set of problems. In Chapter 4 we present a novel method for finding this sort of representation.

3.3.2 Fully local representations, with variations

A second popular form of desired representation is a local representation. Local representations are compact and discrete, which make them natural for data compression and communication. In Section 3.2.3 we focused on the simplest form of local representation, in which a single unit represented each input. Here we outline some important variations that have been proposed.

The standard neural network algorithm for forming local representations is competitive learning. This algorithm attempts to develop the best possible representations with only M discrete alternatives. There is typically additional structure in an input set that cannot be captured in a single round of this algorithm. For instance, if each input image contains a single horizontal or vertical line in a random position, the standard competitive algorithm would only be able to group the pixels of a line by some combination of position and shape, and would not discover these two separate dimensions that underlie the input set.

One method for extending local representations is to have several independent competitive pools, each of which may form its own partition of the input space. Rumelhart and Zipser (1985) showed that this scheme was able to learn to discriminate different shapes in a simplified version of the above task. Yet important modifications, including augmenting the input, were needed to prevent the natural tendency for different pools to all discover the *same* partitioning.

One method of forcing different pools to encode different components of the input is to apply them sequentially rather than simultaneously. Mozer (1991) proposed an Iterative Competitive Learning (ICL) algorithm, where after each stage of competition, the component that was captured in the representation at that stage is subtracted from the input pattern. On an image compression task, this method out-performed Cottrell, Munro, and Zipser's (1989) self-supervised network, when comparing the resulting signal-to-noise ratio at a variety of bits per pixel compression levels. As

Mozer points out, this advantage is not surprising, since unlike this other method, his approach does not separate the training procedure, which discovers the representation, from the quantization step, which turns the representation into a code suitable for transmission.

All of these algorithms are actually neural network implementations of conventional data compression schemes, collectively known as *vector quantization* methods (Gray, 1984). A vector quantizer (VQ) maps a sequence of continuous or discrete vectors into a digital sequence for transmission across a data channel. Vector quantizers are guaranteed to perform better than scalar quantizers, as this is a fundamental result of Shannon's rate-distortion theory; Shannon's theory, however, did not provide constructive design techniques for vector encoders.

In a VQ, the data is a sequence of random vectors $\mathbf{x}$. The sender produces a sequence of channel symbols $\mathbf{u}$, which is encoded and delivered to the receiver. The receiver then maps this sequence into the final reproduction sequence of vectors $\hat{\mathbf{x}}$. The quantizer is optimized by minimizing a distortion measure describing the cost of reproducing $\mathbf{x}$ from $\hat{\mathbf{x}}$. We can readily view these elements in terms of the MDL framework presented in Chapter 2. This distortion measure corresponds to the reconstruction error, while the channel symbols $\mathbf{u}$ correspond to the representation activities.

We can also easily relate competitive networks to vector quantizers. A feedforward network corresponds to a memoryless VQ, where each successive input is encoded using a rule that does not depend on past inputs or outputs; each channel symbol corresponds to an output unit in a competitive network, and the symbol's *codeword* is the weight vector into that unit. The standard competitive learning algorithm corresponds to the standard LBG algorithm (Linde, Buzo and Gray, 1980) for optimizing memoryless VQs, where each training instance is encoded using a nearest-neighbor rule, which minimizes most distortion measures, and then each symbol's codeword is updated to be the centroid of all training vectors which mapped to that symbol.

Variations on competitive learning algorithms also have their counterparts in VQ techniques. The problem of choosing good initial weights corresponds to choosing an initial codebook. One common approach in both cases is to initialize using some random set of the vectors in the training set. Other variations permit reduced memory or complexity. One example is a tree-searched VQ, where a tree is built from the root down, as the training sequence is recursively divided and the algorithm run separately on each node until the leaves contain the actual reproduction codewords. Mozer's ICL algorithm implements a different variation of VQ known as multi-step VQ. This is a tree-searched VQ where the residual vectors of one layer provide the training sequence for the next. This provides the reduction in computation of the tree-searched VQ, since the input vector is encoded in stages, while the residual encoding decreases the required number of codewords.

The novel training algorithm presented in Chapter 5 of this thesis bears some relation to another VQ technique, known as *product vector quantization*. In a product VQ, a collection of codebooks are defined, and the total codebook is the set of all possible concatenations of words from these separate codebooks. A product VQ may yield a smaller computational complexity than an ordinary VQ of the same dimension. This technique is useful when there are different aspects of the input vector that one might wish to code separately because of different effects. Gray (1984) gives an example of a product VQ where separate but interdependent codes are used to code the shape and gain of a waveform. Splitting the encoding into these two aspects yields savings on search, as if R_s bits are devoted to shape and R_g to the gain, then only $2^{R_s} + 2^{R_g}$ distortions need be computed rather than the $2^{R_s+R_g}$ that a standard VQ would require, and both yield the same minimum distortion codeword. We discuss product VQs in more detail in Chapter 5.

3.3.3 Population-coded representations

A third form of representation that is a popular goal for unsupervised learning techniques is a topographic map. In a topographic map, information is represented by a population of active units; it is therefore a sparse distributed representation. The fundamental characteristic of a topographic map is that the spatial organization of the representation units in the map reflects structure in the input set, as similar points in the input map to nearby patterns of activity in the representation.

Defined in this manner, topographic representations possess several important characteristics. They are stable, which means that small changes in the input produce small changes in the representation. In addition, they are fault-tolerant, in that the activity of any one unit is not critical for representing an example.

The motivation for topographic representations once again stem also from biology, as topographic maps are one of the major principles of organization in sensory systems (Churchland and Sejnowski, 1992). In mammalian striate cortex, adjacent neurons have adjacent receptive fields, and the neurons collectively constitute a fractured map of the retina. Body location is topographically represented in the somatosensory system, as is frequency in the auditory system. In other cortical areas, more abstract properties are represented topographically. In bat auditory cortex, for example, frequency differences and time delays are represented in a topographic fashion (Suga, 1984), as are amplitude differences and time delays in the inferior colliculus of the barn owl (Konishi et al., 1988), allowing precise sound localization. In each of these cases, some evidence exists that the axon routes and synapses are not genetically hardwired, but instead are activity-dependent. So one needs a mechanism of creating appropriate topographic maps based on activity.

An early model for developing topographic maps was proposed by Willshaw and von der Malsburg (1976). It is very similar to von der Malsburg's model for orientation column development, which was described in Section 3.2.3. A threshold on the output units ensured that only a few fired at a time, while additional lateral connections within the output layer in the form of a Mexican hat (i.e., short-range excitatory and longer-range inhibitory) forced neighboring units to respond similarly while more distant outputs responded differently. The system was trained with simple Hebbian learning and weight renormalization. In each input image, a pair of neighboring inputs was active; the system learned a form of retinotopic map.

An alternative method for discovering topographic representations is to create the topology through the learning rule directly rather than through the form of lateral connections. This is the approach adopted in the Kohonen algorithm (1982), which is a computational simplification of the Willshaw-von der Malsburg model. The input to a Kohonen network consists of a few continuous-valued inputs; the model uses a simple 2-layer network, where the output units compete in a WTA fashion for each input. In addition, the outputs live in a space which has a pre-defined topology. Units that are neighbors of the winner also update their weights. The learning rule is as follows:

$$\Delta w_{ji} = \epsilon F(j, j^*)(X_i - w_{ji}) \tag{3.8}$$

where j is the winning output unit and $F()$ is the neighborhood function, which equals one if $j = j^*$ and falls off with the distance between the two units in the output array. Note that this is a simple modification of the standard competitive learning update rule (see Equation 3.5). Hence outputs that are neighbors within the given topology of the network modify their weights *in input space* in a concerted way, such that they win the WTA competition for nearby inputs.

The neighborhood function F is typically a radially-symmetric exponential function, such as a Gaussian. In practice, it has been found useful for convergence to start off with a wide range for F, and gradually reduce it during learning. This allows the network initially to form a crude ordering of the space, and then refine it with respect to the inputs. In addition, the learning rate ϵ is typically reduced during learning until at value 0 learning stops.

A map is then formed in the output space that reflects the region of the space in which inputs occur. For example, a set of two-valued inputs describing the coordinates of points lying along a curve may be mapped onto a linear array of outputs, where each output unit responds to points within a small region of path lengths s. This mapping in output space reflects the underlying dimensions of variability in the input, so long as those dimensions are explicit in the few-dimensional input space, and often performs a form of dimensionality-reduction. In addition, the net performs a form of *density modelling*, as the density of output units reflects the input probability in that region of space.

A related algorithm is the *elastic net* algorithm (Durbin and Willshaw, 1987), where the topographic map is now considered to be an elastic net, or ring, where the ends are joined. Two forces act on the representation units in this ring; the first draws the units to the inputs, and the second draws together neighboring units on the ring. For example, the inputs might be the cities in a Travelling Salesman Problem, where the goal is to find the shortest tour that visits each city once. The elastic ring represents the tour, so the first force ensures that the tour visits each city, and the second attempts to minimize tour length. This algorithm is similar to the Kohonen algorithm, except that rather updating representation units according to their distance to the closest unit to the input, they are instead updated based on their own proximity to the input, normalized across all the representation units. The elastic net then relies on the second force, which draws a representation unit towards its neighbors, to maintain the topology.

Kohonen networks have been applied to many different tasks (see (Kohonen, 1989) for a review). We mention two applications here. Obermayer, Ritter, and Schulten (1991) applied Kohonen maps to model the development of retinotopy and orientation and ocular dominance in the striate cortex. The input to the network was a 5-dimensional vector describing (x, y)-position, preferred orientation and tuning strength, and ocular dominance; the output was a 2-dimensional map intended to represent the cortical surface. The algorithm generates maps which compare favorably with detailed anatomical data. Also, the topographic maps formed by these networks are often useful for further processing. For example, Kohonen, Mäkisara and Saramäki (1984) sampled Finnish speech using a set of 15 frequency channels, and formed a two-dimensional map as output. This map developed structure that reflected the phonemes in the speech, as nearby units encoded similar phonemes. This map was then used for speech processing, as words were recognized as paths through this representation.

One disadvantage of Kohonen's model is the need to specify the topological structure of the map in advance. Several methods for learning these neighborhood relations have been proposed. The "neural gas" method of Martinetz and Schulten (1991) forms neighborhood relations by building connections between the most and second-most active output units on each input case. This allows the formation of arbitrary topologies, possibly with disconnected regions of different dimensionalities. A drawback of this method is that the learning of the topology is separate from the learning of the feedforward connections; this makes the topology useful only for further learning, and not as an integrated aspect of learning the topographic representation. Fritzke (1991) proposed a method where output units are added and removed from the network in order to minimize the quantization error. This also allows the formation of disconnected regions in output space. This method dynamically selects the size of the map as well, but there is not a well-defined procedure for determining when and where new cells should be created.

A second disadvantage of the Kohonen model is the selection of a single unit to represent each input. This is antithetical to the sparse distributed nature of the desired representation, and is a poor approximation to the effects of lateral interactions. A smoother map could be formed if each input was actually represented by the population of units that respond to it. An alternative to the WTA competitive learning framework is a self-supervised network where the set of active hidden units cooperate to reconstruct each input. Saund (1989) added a separate error term to the reconstruction error that encouraged the pattern of activity of the hidden units to form a "scalar set", i.e., where this pattern describes a particular scalar value. The activity of the units can be seen as samples of a unimodal smearing function, such as a Gaussian, centered at the location along the interval corresponding the intended scalar value. Simulations demonstrate that this method is capable of forming one-dimensional topographic representations of a curve based simply on points drawn from the curve. The method, however, requires that the input also be represented as a scalar set, and separate sets of hidden units are needed for each variable.

In Chapter 6 we describe a related method for forming topographic representations. This method also overcomes some of these disadvantages of the Kohonen model, and it is compared to these other methods below.

3.4 Unsupervised algorithms and the MDL framework

In this chapter, we reviewed a variety of types of representations that are formed by unsupervised algorithms. The underlying goal in each of these algorithms is to find succinct yet informative representations of a dataset. In both kernel and projection methods, the aim is to develop a low-dimensional description of each data point. Whether the algorithm is clustering or finding principal components, it attempts to identify and preserve the few key aspects while removing noise and irrelevant detail. These reduced descriptions of the data can be expected to aid any further processing.

The various unsupervised learning methods involve different ways of building structure and *a priori* expectations into the representations and learning algorithms. The common theme, however, is that we would like to limit the complexity of the representation without losing important information in the data. The MDL criteria allows us to decide when complexity in the representation is not worth the improvement in the fidelity of the representation to the data.

Many of these unsupervised learning algorithms can thus be understood using the framework presented in the previous chapter. For example, the set of learning algorithms based on principal components analysis (PCA) (see Section 3.2.2) can be viewed as a version of MDL in which we limit the representation-cost by only having a few representation units, and we ignore the model-cost and the accuracy with which the representation activities must be coded. This can be easily seen once one observes that a self-supervised linear network will perform a version of PCA. Minimizing the reconstruction error allows the M representation units to span the space of the first M principal components of the input distribution.

Self-supervised networks have recently been extended to perform a non-linear analog of PCA, by making the units non-linear and adding layers between the inputs and the representation layer as well as between the representation layer and the outputs (Kramer, 1991; Oja, 1991; Usui, Nakauchi and Nakano, 1991). These networks extract "principal manifolds" of the input distribution. These algorithms can also be viewed in MDL terms, as can a recent variation. DeMers and Cottrell (1993) propose a method for dynamically determining the dimensionality of the representation. They apply a greedy algorithm that successively eliminates representation units by attempting to drive a chosen unit's variance to zero. A steadily-increasing penalty coefficient is applied to the unit's variance; if its variance can be driven to zero while maintaining an upper limit to the reconstruction error, then the unit can be excised from the network, and the process repeats.

This algorithm readily translates into the MDL framework, if the prior distribution on a unit's activities $\mathcal{A}$ is taken to be a Gaussian with a variance equal to the actual variance of the unit's activity, and the penalty coefficient describes accuracy to which the activity must be specified, or the inverse of the tolerance in Equation 2.10. Thus the representation-cost of a unit is taken to be the ratio between its variance and the tolerance with which its values must be specified. Increasing the penalty coefficient on a low-variance unit is then equivalent to decreasing this tolerance, which amounts to a prior distribution for the accuracy as a function of variance. This interpretation suggests an extension to their scheme where the representation-costs are traded off against reconstruction-costs for all units simultaneously rather than sequentially, thereby avoiding some of the local minima problems of the greedy sequential algorithm. In addition, the algorithm can also be extended to include the model-cost bits saved in describing the weights of a deleted zero-variance unit. Adopting the MDL viewpoint thus makes the inherent trade-offs in this scheme explicit, and suggests potential extensions and generalizations.

The other major class of unsupervised learning algorithms—kernel methods (see Section 3.2.2)—can also be seen in MDL terms. For example, in competitive learning, or vector quantization, the code is the identity of the winning hidden unit, so by limiting the system to N units we limit the average representation-cost to at most $\log_2 N$ bits. The reconstruction error is proportional to the squared difference between the input vector and the weight-vector of the winner, and this is what competitive learning algorithms minimize. The model-cost is usually ignored. Hierarchical versions of competitive learning, such as Iterative Competitive Learning (Mozer, 1991), or multi-step VQ, extend the basic idea by making the representation have several components. Here the representation-cost is still fixed, but the representational power is increased by allowing several units to encode each input. One could imagine further elaborations of the basic algorithms, particularly with respect to automatically determining the appropriate number of units in a vector quantizer, by adopting the MDL framework and making the trade-offs explicit.

A key point here is that the representations produced by standard VQ contain very little information about the input (at most $\log_2 N$ bits). To get richer representations we must allow many hidden units to be active at once and to have varying activity. While PCA algorithms allow this, they generally assume a fixed representation-cost, and ignore the accuracy to which the representation activities must be specified. The following chapters describe novel unsupervised learning algorithms that can be obtained by considering various alternative methods of communicating these activities.

Chapter 4

Developing Non-Redundant Binary Representations

The first novel unsupervised algorithm defined within this framework is simple; nonetheless, it successfully captures the structure in a range of datasets. In addition, this simple version introduces several key ideas that form the basis of the other algorithms presented in the thesis.

In the first section of this chapter, we motivate the representation scheme that underlies the algorithm. The second section contains a detailed description of the algorithm, explicitly specifying the network architecture and unit activation functions, and deriving the cost function for the network from the framework. We also discuss two variations of the basic algorithm, based on slightly different methods of coding the representation activities. In the third section, we describe experimental results of applying the resulting networks to some simple datasets. We specify the network parameters and performance measures, and then present results for different values of these parameters. The chapter concludes by relating this algorithm to previous approaches.

4.1 Motivations

This first algorithm is based on the minimally redundant representation scheme described in Section 3.3.1. The idea in this scheme is that each representation unit corresponds to an underlying feature of an input example, and each example can be described as a combination of several of these features. The features are taken to be independent. If any given example has only a few features, then it can be represented by a few active binary units; this leads to a sparse distributed representation for each input, in which the units corresponding to the relevant features are active, and the rest are inactive.

The key to the utility of this, and any form of representation is its suitability for the given dataset. Here, the working hypothesis is that in many domains, the data can be described as combinations of a few items out of a wide vocabulary of items. For instance, a jazz solo can often be described as a series of "licks", where each soloist typically has a wide repertoire of these licks, and selects a few for any given solo. The individual notes in a lick are often redundant, or irrelevant in terms of representing the solo. Similarly, features of an object, such as two halves of a chair, are often redundant, while a scene can be described as a set of these objects. Objects are collections of highly correlated properties, what Barlow (1989) termed 'suspicious coincidences'. He proposed a *non-redundant* form of representation, where the representation captures and removes these redundant features from the input. He likened the variables used in this type of representation to nouns in a language: there are many nouns, each occurs infrequently, and something can be represented by two or more nouns.

The representational scheme of this algorithm—sparse distributed representations comprised of independent elements—has several appealing properties. A sparse distributed representation enjoys some of the computational benefits of both local and distributed representations. As in a local code, several data items can be represented simultaneously, assuming that their features do not significantly overlap, and the representation does not suffer from crosstalk when more than one item is present, leading to greater capacity. In addition, a representation where only a few units are active for each input pattern can simplify and speed up subsequent processing, as it leads to a lower degree of connectivity. As in a distributed representation, however, the various combinations of representation variables offer higher resolution in describing a single data vector (Hinton, McClelland and Rumelhart, 1986) and many more vectors can be encoded for the same number of representation variables. Distributed representations are also more tolerant with respect to noise and damage, and can lead to generalization to novel inputs composed of the same features. Furthermore, if a sparse distributed code can also capture important underlying regularities in the task domain, it can lead to much better generalization.

These representations are also appealing from the standpoint of biological plausibility. In the lower cortical areas, we know that many neurons are arranged into retinotopic/topological maps, so that for any local stimulus, a few neurons will typically respond. Less is known about the response properties of cortical cells beyond the early stages. Barlow (1983) states that neural cells in higher areas of cerebral cortex are less often active than inactive, and the number of neurons active for each stimulus is low. He speculates that if each of these cells fires infrequently, in order to transmit information in the visual image, each cell must convey a lot of information when it does fire.

A second significant aspect of the representational scheme is that the representation variables, or units, are independent, or at least minimally redundant. That is, the activity of one unit is unpredictable from the activity of another. Each unit corresponds to an instantiation of a feature, and these features are assumed to occur in various combinations and permutations across the set of inputs so that they are relatively independent. Transforming the data into a representation with minimal redundancy can speed up further learning (Becker, 1989). Completely eliminating the redundancy leads to truly *factorial* representations, in which the representational elements are statistically independent. This is a particularly efficient form of representation, since the probability of the occurrence of any input is simply the product of the probabilities of the corresponding representational elements. Also, the detection of dependence between two elements in a factorial representation indicates a novel association (Barlow, 1989).

4.2 An MDL algorithm for non-redundant representations

In our MDL framework, a key idea is that in order to form a non-redundant representation, we simply adopt the most straightforward encoding scheme in which each representation unit's activity is encoded independently. We expect this scheme to achieve our objective since once we are devoting bits to encode each unit's activity, the network should make use of each bit, and make the units *informative*, i.e., participate in forming accurate reconstructions of the data. The set of representation units cooperate to reconstruct each input.

This description introduces three general themes. First, the encoding scheme is based on an assumption about the representation unit activities; here this assumption is that the units are independent. This assumption then determines the types of representations that the network will develop, as the search for cheap codings will encourage representations that conform to this assumption. In other words, since we decide to encode the units' activities independently, the algorithm will tend to eliminate redundancy in the representation, as this will be a waste of bits. The optimal result will then be factorial representations.

In addition to this pressure from the activity cost to form non-redundant representations, the goal of minimizing reconstruction cost provides a second source of pressure encouraging the representation units to diversify. If two units are doing the same thing, i.e., have the same inputs and weights, then the network is wasting bits in terms of both the activity and reconstruction costs. The algorithm will therefore be likely to attempt to minimize the encoding cost by recruiting one of the units to help represent some other part of the input space.

An alternative method of lowering the encoding cost is to spend fewer bits encoding the activities of a redundant unit. This is the third theme: the algorithm can adaptively determine how many bits are used for each unit independently. While the network architecture is pre-determined, a unit can effectively be eliminated by setting all its incoming weights (excluding the bias) to zero, so that it has the same activity level for each input. Thus the network may save bits by eliminating one of the two identical units.

The basic idea of encoding each representation unit's activity independently leaves some flexibility as to how this encoding is done. Before describing the encoding scheme, we first present the network architecture and activation functions, and introduce a key aspect in the network formulation of the algorithm.

4.2.1 Noisy representations

In this algorithm, we adopt a representational scheme that favors binary representations, where a few units are active and the rest are not. A straightforward way of formulating these representations is to use binary threshold units in the representation, or hidden layer of the reconstruction network. For each input pattern, a subset of the representation units will be on, i.e., have activity levels of one, and the rest will all be off, i.e., have activity levels of zero.

Rather than formulating the problem in terms of these binary representations, we instead convert it to a continuous representation. Our strategy is to associate with each input pattern, not a single binary vector, but rather a probability distribution over the ensemble of possible binary representations. The advantages of this approach were described in detail above (see Section 2.5.4). We review the main points here.

Formulating the problem in terms of these "noisy" representations has two main advantages. First, it is very difficult to search through a space of discrete representations. Many search algorithms rely on gradient information, which is not available in non-continuous spaces. If we select a single binary representation for a given input, then choosing a new one creates sudden changes in the error landscape. Converting the discrete problem into a continuous one facilitates the search, creating a smooth space in which the algorithm can search for good encodings.

The second advantage is that a noisy representation enables the network to save bits in describing the representation of a given case. The intuition here is that the precise specification of a representation for an input will require more bits than a less precise description. If we assume a coding prior that makes two binary vectors equally likely, then when the vectors are equally good representations for the input, this representation should be cheaper to encode (by one bit) than one in which only one is good. This intuition is borne out in the encoding scheme for these noisy representations, which is discussed below.

In order for the network to contain noisy representations, we adopt a simple probabilistic interpretation of the representation units. These units are stochastic binary variables, i.e., associated with each representation unit j is y_j^c, a real value between zero and one, which is the probability that it has value one on case c:

$$y_j^c \;=\; pr(S_j^c = 1) \;=\; \sigma(\sum_i w_{ji} y_i^c) \tag{4.1}$$

where S_j^c is the random binary state of unit j, w_{ji} is the weight from input unit i to unit j, and σ is the nonlinear sigmoid function $\sigma(x) = (1 + e^{-x})^{-1}$.

An autoencoder network containing these type of representation units can be run stochastically, using Monte Carlo techniques to sample from the units' probability distributions given above. Yet this approach is time-consuming (particularly during learning), since each unit must be sampled many times for each training case in order to obtain an accurate model of the probability distribution across the ensemble of binary representations. An alternative is to approximate this distribution.

For some architectures, however, we can compute the required quantities exactly, without any approximations or stochastic sampling. The networks we describe in this chapter have such an architecture: Each network has a fully-connected 3-layer architecture (shown in Figure 1.4), and the output units are linear. Output unit k then has a real-value y_k^c on case c, which is the expected value of the weighted sum of the random binary states of the representation units:

$$y_k^c \;= <Y_k^c> = <\sum_j w_{kj} S_j^c> = \;\sum_j w_{kj} y_j^c \tag{4.2}$$

By adopting this formulation, we convert the discrete search problem to a continuous one. An input pattern gives rise a set of real-valued activities in the representation layer, which describes a probability distribution across the representation units; the value of a representation unit describes the probability that that unit will participate in a given binary representation transmitted to the receiver. This probability distribution in turn gives rise to a probability distribution across the output units, which is also represented as a set of real-valued activities. These real values provide the algorithm with a continuous space in which to search for good encodings. The goal then is to maximize the likelihood of reconstructing the input, subject to the description length constraints.

Two important points emerge from this discussion. The first is that by adopting this probabilistic approach, the network is in effect considering a range of possible representations for a given pattern. This leads to the second point, which concerns the independence of the sender and receiver in our communication model. The sender is using this neural network approach to develop good representations, and is free to formulate the problem in any way so long as the receiver can recover the original data.

4.2.2 Coding reconstruction error

We derive the objective function for the network based on the MDL framework presented in Chapter 2. The goal is to minimize a cost function that is the sum of two terms. The *activity cost* $C_{\mathcal{A}}$ describes the cost of encoding the representations of the dataset, here captured in the activities of the representation units. The *reconstruction cost* $C_{\mathcal{R}}$ describes the cost of encoding the misfit between the data and the network's predictions for the data.

The reconstruction cost $C_{\mathcal{R}}$ involves encoding the *reconstruction error*: the differences between the values at the output and input layers of the network. In this algorithm, we assume a mean-zero, fixed-variance Gaussian prior probability distribution to encode this error (see Section 2.3.2). We assume that each output unit is encoded independently on each training case. In a standard autoencoder, the resulting reconstruction cost (ignoring constant terms) equals the sum-squared difference between the desired (input) and actual outputs of the network:

$$C_{\mathcal{R}} = \sum_{k,c} (y_k^c - d_k^c)^2 / 2V_{\mathcal{R}} \tag{4.3}$$

where d_k^c is the desired value of output unit k on case c (which equals the value of the corresponding input unit); y_k^c is the actual value of the unit; and $V_{\mathcal{R}}$ is the fixed variance of the coding Gaussian.

In this network, however, instead of considering the actual value y_k^c of each output unit k, we also take into account the stochastic nature of the representation unit activities. Since the value of each such unit j is a random variable S_j^c, the value of an output unit k is also a random variable Y_k^c. We would now like to estimate the *expected* reconstruction cost, rather than the reconstruction cost of the expected values.

Since Y_k^c is a linear function of a set of independent[1] random variables S_j^c (see Equation 4.2), we can calculate the first two moments of Y_k^c based on the first two moments of the set of S_j^c's. Equation 4.1 shows that the expected value of the binomially-distributed variable S_j^c is y_j^c; its variance is given by:

$$\begin{aligned} var(S_j^c) &= < (S_j^c - y_j^c)^2 > \\ &= y_j^c(1 - y_j^c)^2 + (1 - y_j^c)(y_j^c)^2 \\ &= y_j^c(1 - y_j^c) \end{aligned} \tag{4.4}$$

We have already seen (Equation 4.2) how the mean of each Y_k^c is a weighted sum of the representation unit activities. The variance of Y_k^c can also be computed directly from the set of y_j^c's and the weights:

$$V_k^c = var(Y_k^c) = \sum_j w_{kj}^2 var(S_j^c) = \sum_j w_{kj}^2 y_j^c (1 - y_j^c) \tag{4.5}$$

We use these values to calculate the expected reconstruction cost $C_{\mathcal{R}}$ for the network, which now takes into account the stochastic representations:

$$\begin{aligned} C_{\mathcal{R}} &= < \frac{1}{2V_{\mathcal{R}}} \sum_{k,c} (Y_k^c - d_k^c)^2 > \\ &= \frac{1}{2V_{\mathcal{R}}} \sum_{k,c} < ((Y_k^c - y_k^c) + (y_k^c - d_k^c))^2 > \\ &= \frac{1}{2V_{\mathcal{R}}} \sum_{k,c} < (Y_k^c - y_k^c)^2 > + (y_k^c - d_k^c)^2 + 2(< Y_k^c > - y_k^c)(y_k^c - d_k^c) \end{aligned}$$

[1] The expected values of the S_j^c's are not independent over the set of training cases, but the fluctuations within a training case are independent, i.e., on any given stochastic sampling trial, each unit would be sampled independently according to its own distribution.

$$= \frac{1}{2V_{\mathcal{R}}}[\sum_{k,c}(y_k^c - d_k^c)^2 + V_k^c] \tag{4.6}$$

The effect of this reconstruction cost is not only to force the network to capture the input data (i.e., make $y_k^c \approx d_k^c$), but also to encourage the assignment of a single binary representation for each input. The penalty on the variance of an output unit (V_k^c) translates into a penalty on the variance of representation units' activities by Equation 4.5. This variance can be eliminated if the expected value of the unit is binary (i.e., $y_j^c \approx 0$ or 1). If y_j^c is binary, then unit j will be constant for the various representations of input c, and if all representation units are binary, there will be a single binary representation for that input, and the stochastic representation reverts back to a discrete one.

Note that this derivation becomes considerably more complicated if Y_k^c is a nonlinear function of the set of S_j^c's. This is an example of an architecture in which either stochastic sampling or an approximation must be used. In some cases, the expected values and variances of the output units, and the reconstruction cost can be approximated. For instance, if we assume that many units affect X_k^c, the total input to unit k, then by the central limit theorem, the distribution of X_k^c is approximately normal. The output of the unit can then be described as a nonlinear transformation of a Gaussian random variable. In order to speed up the implementation of such a scheme, we have used a pre-computed look-up table that stores the requisite expected values for various values of the mean and variance of X_k^c. In the Discussion section below, we discuss the relationship between this approximation and the mean-field approximation, a standard approximation used for stochastic systems.

4.2.3 Coding representation activities

Up to this point, we have described the architecture and activation functions for the network. The previous section described the first half of the cost function—the reconstruction cost. In this section, we present the other cost term—the cost of encoding the representation activities.

In a standard autoencoder, the middle layer has fewer units than the other layers. This layer acts as a bottleneck, forcing the network to find a compact, invertible encoding for each input vector. These representations are fully distributed, in that each representation unit participates in each encoding. If each unit has a fixed cost for encoding its activity, then minimizing the number of representation units minimizes the description length.

This approach is similar to that of PCA, where a compact representation is obtained by keeping only the first few maximally informative, mutually orthogonal components. Indeed, an autoencoder will perform a version of PCA if the output units are linear.

The representational scheme in this algorithm, however, favors sparse binary representations, in which only a few units participate in encoding a pattern. Our coding scheme is based on this assumed form of representation. As was described in Chapter 2, constructing the coding scheme based on an assumed representation form will encourage the network to find these types of representations, since they will necessarily be cheap to encode.

In this case, the representation units are coded independently, and it is therefore optimum for the networks to use these units independently. In this way the scheme discourages redundancy between the activities of the units. In addition, since we expect sparse representations, where units will typically be inactive, the coding scheme makes it much cheaper to say that a representation unit is off rather than on.

Adopting the MDL approach thus removes the need for a narrow hidden layer, as this architectural constraint is now taken care of by a more flexible constraint on unit activities. Making it cheap to describe inactive units allows the representation layer to contain more units than the input or output layer, which fits the representational scheme of a large vocabulary in which a small subset describes any given input.

Variation 1: Coding with a fixed global base rate

The first version of a non-redundant coding scheme assumes that there is a fixed *base probability* b that is the same for each representation unit; it describes the prior probability that a representation unit will be active. We assume that the sender and receiver both know b.

On each case c, the sender transmits the binary values S_j^c for each representation unit j. The sender selects these values from the binomial probability distribution $Q_j^c = \{y_j^c,\ 1-y_j^c\}$, and encodes them from the prior distribution $P = \{b,\ 1-b\}$. The cost of this encoding is the expected additional information required to specify values from Q using P, or the asymmetric divergence between these two distributions. This situation was described in Section 2.5.4. The expected activity cost is then:

$$\begin{aligned} C_{\mathcal{A}1} &= \sum_{j,c} G(P, Q_j^c) = \sum_{j,c,a} Q_j^c(a) \log \frac{Q_j^c(a)}{P(a)} \\ &= \sum_{j,c} Q_j^c(S_j = 1) \log \frac{Q_j^c(S_j = 1)}{P(S_j = 1)} \ + \ Q_j^c(S_j = 0) \log \frac{Q_j^c(S_j = 0)}{P(S_j = 0)} \\ &= \sum_{j,c} y_j^c \log(y_j^c / b) + (1 - y_j^c) \log((1 - y_j^c)/(1 - b)) \end{aligned} \tag{4.7}$$

where a indexes the possible activity values of unit j.

This cost can be understood by viewing the flow of information between the sender and receiver. We assume that the sender is already communicating a large number of random bits to the receiver and we want to compute the *additional* cost of communicating the inputs. For each input case c, there are a number of possible representations. The sender uses the random bits to select a binary value from Q_j^c, and encodes it using P. The expected cost of this encoding for each unit j and case c is $-y_j^c \log b - (1 - y_j^c) \log(1 - b)$. The original expected cost of the message is then this encoding cost plus the cost of the random bits. Once the receiver has received the full set of representation activities, and used them to reconstruct the input, he can run the same learning algorithm that the sender used, and can then recover each posterior distribution $Q_j^c = \{y_j^c,\ 1 - y_j^c\}$. He can then recover the random bits that the sender used to select the binary values S_j^c from this distribution. Since the receiver gets all of the random bits back, the sender has succeeded in sending both the representations and the bits while only actually transmitting the coded binary values. So the total cost of the message is the difference between the original coding cost and the cost of the bits, which equals the summed entropy of the posterior distributions.

This representation activity cost per unit is a convex function of its activity (see Figure 4.1).

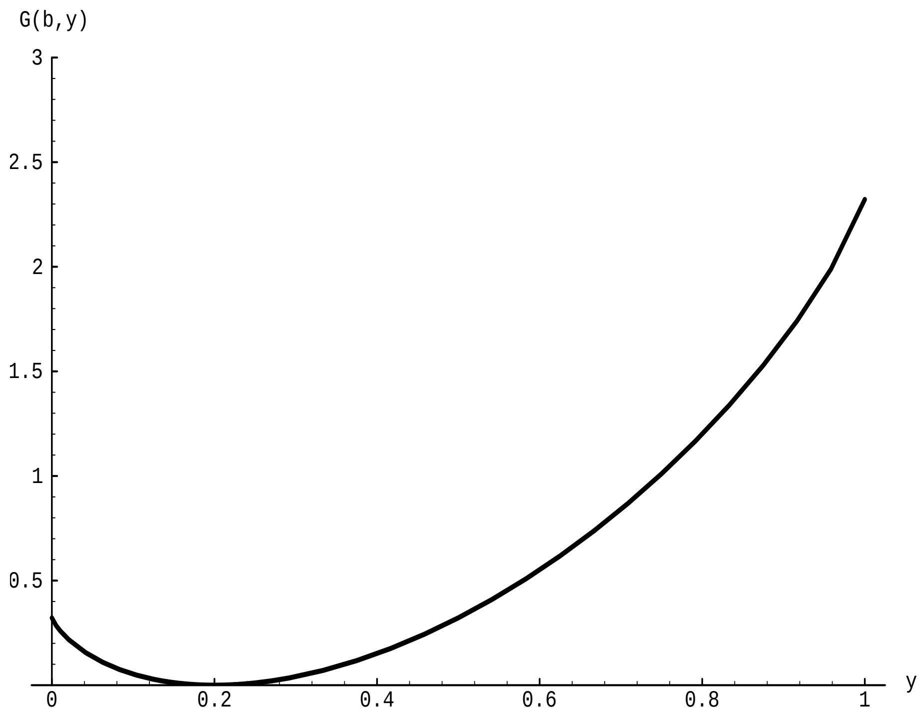

Figure 4.1: This plot shows the asymmetric divergence $G(P, Q)$ between two binomial distributions, $P = \{b, 1-b\}$ and $Q = \{y, 1-y\}$. Here $b = 0.2$.

If the unit's activity on a case equals the base rate ($y_j^c = b$), then the cost is zero. It increases in both directions as y_j^c approaches its bounds of 0 and 1. We therefore control the type of encodings that the network discovers by manipulating this parameter b. In order to form sparse distributed representations, we specify a low value for b, thereby making the encoding cost of each unit smaller when it is off. A unit that is firmly off ($y_j^c \approx 0$) has an encoding cost of $-log(1-b)$. When b is low, this cost approximately equals b. Meanwhile, a unit that is firmly on for a given input ($y_j^c \approx 1$) has an encoding cost of $-log(b)$. Since this produces a big encoding cost, this active unit must significantly contribute towards lowering the reconstruction cost.

Under this encoding scheme, the network has the freedom to make some units free by making them conform to their prior. A unit that has activity b on each case will cost no bits. This allows the network to effectively determine an appropriate vocabulary of representation units for a given dataset.

Variation 2: Coding with adaptive and local base rates

A variation on this encoding scheme allows the representation units to collectively determine a global base rate. A poor choice of b may make it difficult to find good representations for a given dataset. We can avoid the need for this imposed, fixed parameter by adapting the base rate.

The cost of coding this new b is negligible, since it need only be sent once, before the representation activities on each case are transmitted. The cost term in this case is the same as in the variation above (Equation 4.7), except that now b is adapted as well.

Encoding each representation activity based on a single b involves an implicit prior: each representation unit has the same base rate. It may be possible to find a cheaper encoding of the activities if we let each representation unit j form its own base rate b_j.

If we find the b_j that minimizes $< G(P_j, Q_j^c) >$ where $P_j = \{b_j,\ 1 - b_j\}$ and $Q_j^c = \{y_j^c,\ 1 - y_j^c\}$, it turns out that the optimal individual base rate is simply that unit's mean activity: $b_j = \overline{y_j}$. One obtains this simply by minimizing $C = -\sum_c y_j^c \log b_j + (1 - y_j^c)\log(1 - b_j)$ with respect to b_j.

We now adopt a different coding strategy for the activities. We encode the individual base rates $\overline{y_j}$ with respect to a global prior b, and we then encode each posterior distribution Q_j^c with respect to $\overline{y_j}$. Since there is now a separate base rate for each representation unit, we include the cost of sending these rates in the cost function. The approach we have adopted for encoding $\overline{y_j}$ is to assume a Gaussian prior with a mean at b and a fixed variance V_b.[2] The resulting cost for this encoding scheme is given by:

$$C_{\mathcal{A}2} \;=\; \sum_j (\overline{y_j} - b)^2/2V_b + \sum_{j,c} y_j^c \log(y_j^c/\overline{y_j}) + (1 - y_j^c)\log((1 - y_j^c)/(1 - \overline{y_j})) \qquad (4.8)$$

This second variation allows the representation units to have different mean activity rates averaged over the training set without paying a large price on each example. Nevertheless, a price is paid for having a base rate that differs greatly from the global base rate; this price maintains the sparseness of the representations (i.e., discouraging a unit from being on for most examples) while allowing more flexibility.

4.2.4 Discussion

The complete objective function for the network is the sum of the reconstruction and representation activity costs:

$$C \;=\; C_{\mathcal{R}} + C_{\mathcal{A}} \qquad (4.9)$$

The cost function for the first variation described above is:

$$\begin{aligned} C_1 &= C_{\mathcal{R}} + C_{\mathcal{A}1} \\ &= \frac{1}{2V_{\mathcal{R}}}\Big[\sum_{k,c}(y_k^c \;-\; d_k^c)^2 + V_k^c\Big] + \\ &\quad \sum_{j,c} y_j^c \log(y_j^c/b) + (1 - y_j^c)\log((1 - y_j^c)/(1 - b)) \end{aligned} \qquad (4.10)$$

We can understand how these terms in the objective function correspond to the representational scheme. The squared-error term forces representation units to be informative, while the output variance term combines with the asymmetric-divergence to form sparse, non-redundant representations. The effect of the asymmetric-divergence is actually to make the activity of each representation

[2] Note that the individual base rates must be communicated just once per training set. In the high data limit, therefore, the cost of this encoding is small relative to the second term in Equation 4.8. In addition, since y_j is a stochastic binary variable, a Gaussian is not the appropriate prior for coding the base rates. Yet we have included this first term, and chosen a Gaussian coding prior, for empirical reasons discussed below.

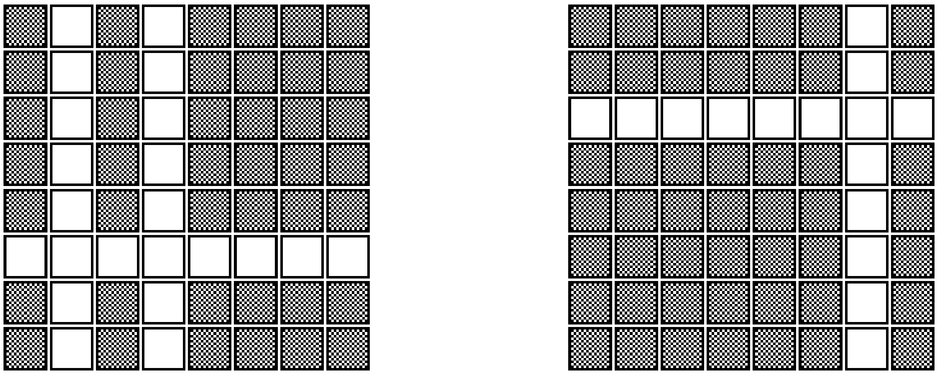

Figure 4.2: Two sample inputs from the horizontal/vertical lines dataset. Each 8x8 image contains a few (typically 0-3) of the 16 possible lines. In this figure, as in the other figures, a white square corresponds to a large positive value (1.0 here), while a gray square corresponds to a value of zero.

unit on each case to equal b. The variance term, however, encourages the activities to be binary. The combined result is to force most units to be firmly inactive on each case, while a few pay the activity cost to be fully active in order to reconstruct the input. A key point is that these active representation units *cooperate* to reconstruct each input. In addition, even though the algorithm involves noisy representations, the network typically ends up with a single binary representation for each input.

The second variation cost function can be interpreted similarly, with the additional flexibility in each representation unit's mean activity:

$$\begin{aligned} C_2 &= C_{\mathcal{R}} + C_{\mathcal{A}2} \\ &= \frac{1}{2V_{\mathcal{R}}}[\sum_{k,c}(y_k^c \; - \; d_k^c)^2 + V_k^c] + \\ & \sum_{j,c} y_j^c \log(y_j^c/\overline{y_j}) + (1-y_j^c)\log((1-y_j^c)/(1-\overline{y_j})) + \sum_j (\overline{y_j} - b)^2/2V_b \end{aligned} \tag{4.11}$$

4.3 Experimental results

4.3.1 Datasets

We tested this algorithm on several sets of training examples. Here we present the results of experiments with two types of datasets.

Horizontal and vertical lines

The first type of input images contained horizontal and/or vertical lines. There are $2N$ possible lines per NxN image; we selected images such that only a few such lines appeared in each image. The images we used in the experiments described here were 8x8 binary images. Each pixel that was part of a line had a value of 1, while others were 0. Two example images are shown in Figure 4.2.

We used two different methods of generating these images. The first was a deterministic method, producing all possible images containing up to M lines per image. This yielded a sum of $\binom{2N}{0}$ up to

$\binom{2N}{M}$ images, or 137 images for $M = 2, N = 8$, and 697 images for $M = 3$. We call these *enumerated* datasets, since they enumerate the possible images.

The second method was stochastic, where each image contains a random number of lines. Each of the 16 lines occurs independently with probability p in each image, where p is chosen to be some small number, such as 0.15. This method produces some images with many lines in it. The size of this *random* training set was 500 images.

Dual components

The second type of dataset used in testing this algorithm had some additional structure in the way it was generated. Each input was a binary vector consisting of two halves. Each half contains one of M possible binary vectors, yielding a total of M^2 input patterns. In these experiments, the input was a 22-bit vector, and each 11-bit half had a vocabulary of 13 vectors in which 3 units had an activity of 1 and the other 8 were 0. The network was trained on all but one of the 169 pairings of these vectors. We return to this point below.

4.3.2 Performance measures

It is generally difficult to adjudge the performance of an unsupervised algorithm on a task, since the goal is to find a good representation, which can be a subjective measure. In our case we have defined an objective measure, and are using that to learn good representations. Yet we may also desire another method of evaluating the quality of the resulting representations.

The datasets described above were designed to make the judgement of the performance of the algorithms clear. Each of these datasets was generated with some degree of underlying structure. We would like an unsupervised algorithm to discover this structure, to find the true causes of the data. The motivation for this was discussed in Chapter 1. The basic idea is that exposing the generator will facilitate both generalization to novel examples in the domain as well as any further processing of the data. We can therefore simply examine the weights into the representation units to find if they correspond to the components of the generator. For example, in the horizontal/vertical lines task, we would like individual representation units to come to correspond to particular lines, so that any image containing a few lines can be represented by a few active representation units. The weights should also be revealing for the dual-component dataset, where a representation unit should correspond to one of the 13 vectors occurring on one half of the input. These types of sparse representations will capture the underlying structure, the true causes of the datasets.

In addition to examining the weights of the network, we employ two other methods to measure the algorithm's performance. The first considers the *generalization* ability of the network after training. We can apply the same MDL measure on which the network was trained to assess its representations of a novel test set of data. This test set data must exhibit similar structure to the training set examples. For the lines, we use a test set of novel examples generated by the the random method. In the dual-component dataset, we leave out one pair when training the network and measure the reconstruction error on that example after training. The measure we use then is the *generalization length*: the expected number of bits required to describe one of these test examples.

The second method evaluates the quality of the representation with respect to the goal of redundancy-reduction. Following Barlow, Kaushal, and Mitchison (1989), we define a measure of the higher-order redundancy in a binary representation as:

$$R = [H_{\mathbf{b}} - H_P]/H_P \tag{4.12}$$

where H_P is the true entropy of the input distribution, and $H_{\mathbf{b}}$ is the sum of the bit entropies, defined by $H_{\mathbf{b}} = -\sum_i p_i \log p_i + q_i \log q_i$. Here p_i is the probability that bit i has value 1 over the set of inputs, and $q_i = 1 - p_i$. An intuitive interpretation of R is that it describes the expected ratio of the redundant bits in the representation to the required number of bits. It can be shown (Watanabe, 1985; Barlow, Kaushal and Mitchison, 1989) that R is always positive, and zero if and only if $\mathbf{b}$, the set of binary representations, is factorial. This means that the capacity of the bits of the representation is greater than the inputs they represent except when the representation is factorial.

In our algorithm, we will see that the algorithm empirically tends to ultimately assign to each input example a representation as a binary vector. In addition, the inputs in the datasets described above are also binary. We can therefore estimate the redundancy-reduction achieved by our algorithm by comparing the redundancy R in the input units versus the representation units. The probability that representation unit j has value 1 over the set of inputs is given by $\overline{y_j}$, and the sum of bit entropies in the representation is:

$$H_{REP} = -\sum_j \overline{y_j} \log \overline{y_j} + (1 - \overline{y_j}) \log(1 - \overline{y_j}) \tag{4.13}$$

We can estimate the redundancy in the input by simply counting the number of input units in each binary example, and taking this to be H_{IN}.[3] The true entropy of the input distribution can be calculated based on the generators we have defined. For instance, in the stochastic lines example, since the lines are generated independently, the true entropy H_P is simply the sum of the entropies of the individual lines: $H_P(p) \approx 16 * (-p \log p - (1-p)\log(1-p)); H_P(.15) \approx 9.76$ bits. The redundancy of the representation and input can then be computed by Equation 4.12, where $H_{\mathbf{b}}$ is H_{REP} and H_{IN}, respectively. For example, the redundancy in the input for the larger enumerated lines dataset (containing 697 examples) is $(64 - 10.66)/10.66 = 5.00$.

In the dual-component dataset, containing all 169 cases, the true entropy of the input is:

$$H_{P_{dvq}} = 2 \sum_{i=1}^{13} p_i \log p_i \tag{4.14}$$

where p_i is the cross-case probability of input i. Since the p_i's are equiprobable, $p_i = 1/13$ and $H_{P_{dvq}} = 2 \log 13 = 7.4$ bits, so $R_{IN_{dvq}} = (26 - 7.4)/7.4 = 2.65$.

[3] This is equivalent to computing the sum of bit entropies in the input assuming that each input unit has an expected activity of 0.5 across the training set. Alternatively, we could compute the sum of bit entropies based on the actual expected activities of the input units.

4.3.3 Parameters

Our new algorithm contains relatively few parameters. First, the number of representation units can be considered as a parameter in the network. One of the ideas in this algorithm is that the network can have many representation units, since instead of relying on the architecture, here the encoding scheme can produce compact representations. This idea is borne out in the experiments discussed below, as we give the network many representation units and it learns to use as many as it needs. Thus the number of representation units is not a critical parameter, so long as the network contains enough to find the input structure.

A second parameter is the variance of the Gaussian used in reconstruction ($V_{\mathcal{R}}$ in Equation 4.6). This term acts as a regularizer, controlling the trade-off between the reconstruction and representation activity costs. It could be optimized within the MDL framework (cf., (MacKay, 1992a)). Instead, we have chosen a value for $V_{\mathcal{R}}$ following a limited set of tests where the description length after learning for several different values were compared. As $V_{\mathcal{R}}$ approaches an upper bound, the network can ignore reconstruction and minimize description length simply by making efficient representations. In the case of the fixed base probability objective (Equation 4.11), it can ignore the input and adjust the bias into each representation unit so that that unit always has an activity of b. Conversely, as $V_{\mathcal{R}}$ approaches zero, the network acts as a standard nonlinear reconstruction network, ignoring the cost of coding the representation activities. It turns out that the network is not very sensitive to this parameter, so long as it is comfortably between these bounds.

The third parameter in this network is the base probability b. This parameter controls the cost of encoding the representation activities on each training example (see Figure 4.1). We will see that the primary advantage of the second variation of this algorithm is its insensitivity to this parameter relative to the first version.

We train the networks described below using a conjugate gradient optimization technique. The simulations were performed using the Xerion simulator package. The results we present are for networks that have been fully trained, i.e., the cost function has reached a minimum. Several simulations with each parameter setting were performed, and the best results are reported. The algorithm typically did not get stuck in local minima; some exceptions are mentioned below.

4.3.4 Results on the lines data

Encoding with a global base probability

The first MDL algorithm, with a fixed, global base probability b, learned the structure of the lines datasets. After training a network with 24 representation units using a low value of b ($b = 0.04$), the network learns to almost perfectly reconstruct the input data. The expected description length of a test example from the larger enumerated set is 16.3 bits, with less than 0.5 a bit for the reconstruction. The activity cost is nearly 16 bits since an active unit exacts a high price ($\approx -\log(.04) = 4.65$ bits). The learning requires 178 weight-update iterations of the conjugate gradient routine. Figure 4.3 shows the incoming and outgoing weights for the representation units after training on the random lines training set.

The network finds analogous weights on all three training sets. Since the weights correspond to

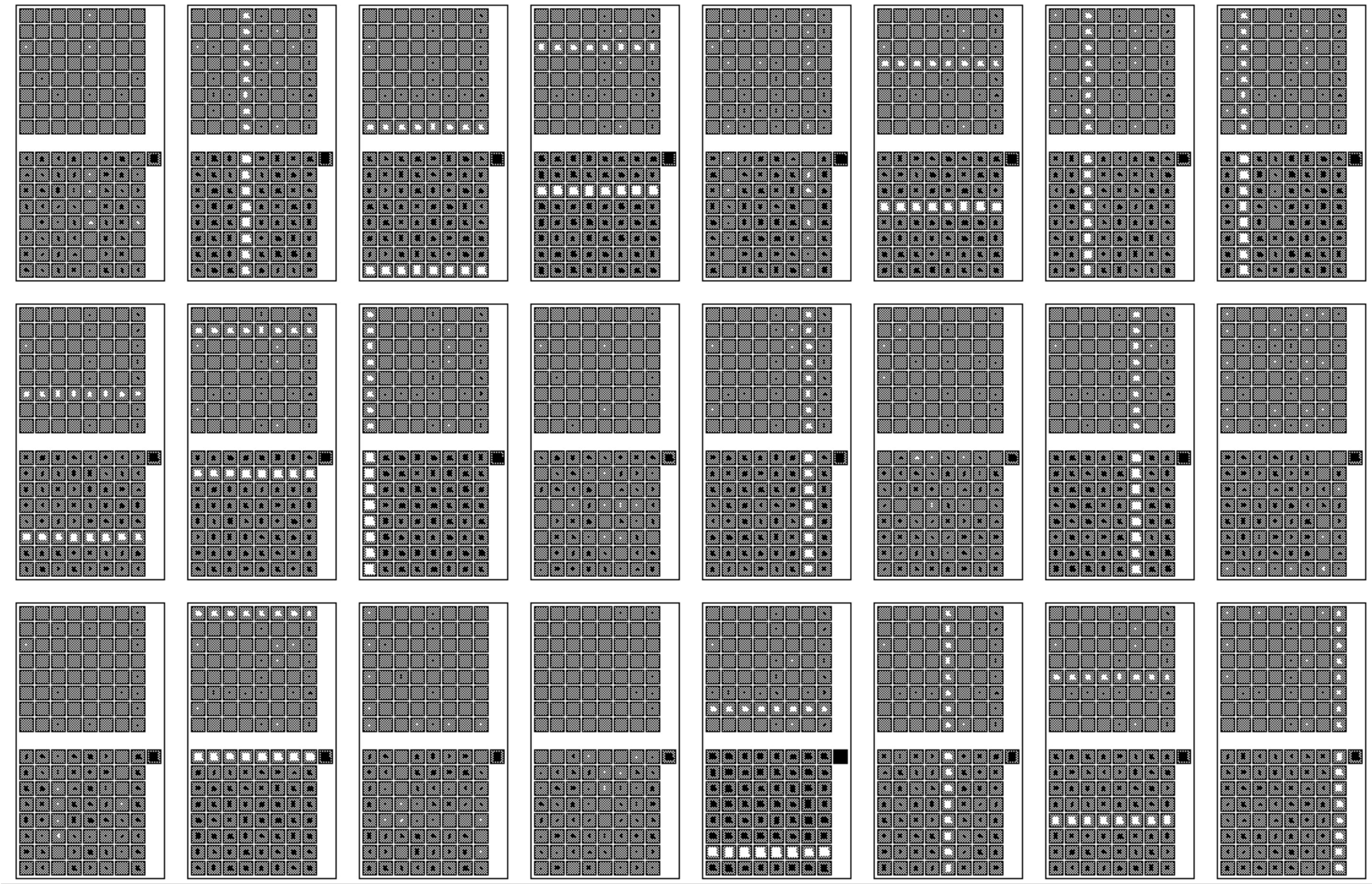

Figure 4.3: This figure shows the incoming and outgoing weights from the 24 representation units after learning using the first cost function. Here, and in all subsequent weight diagrams, the color of the blob indicates the sign of the weight (white is positive, black is negative), and the blob size indicates the weight magnitude. Each unit is fully connected to the output and input layers, here representing an 8x8 image. Each unit's weights are enclosed inside a box. The single square to the right of each of the 24 weight sets corresponds to the bias weight for that representation unit. Note that the network learns to make the outgoing weights resemble the incoming weights.

the independent features from which the input was generated, we would expect this network to have minimized redundancy. Indeed, for this network, R_{REP} initially equals 2.11, but after training, it equals 0.01.

This figure demonstrates several salient aspects of the algorithm. Only 16 representation units are required to optimally encode the 16 possible lines in each image. The network begins with random weights into all 24 representation units. In the final solution, however, only 16 of these 24 units have significant incoming and outgoing weights. The other units learn to ignore the input and simply adjust their biases so that they are free; i.e., their activity on each case matches b. The performance of the algorithm does not change much as more representation units are added to the network, but the learning speed decreases to 120 weight updates. For example, a network with 30 representation units trained on the same task has a generalization length of 16.8 bits, as a slight decrease in reconstruction error is offset by a slight rise in activity cost. The extra units ease the

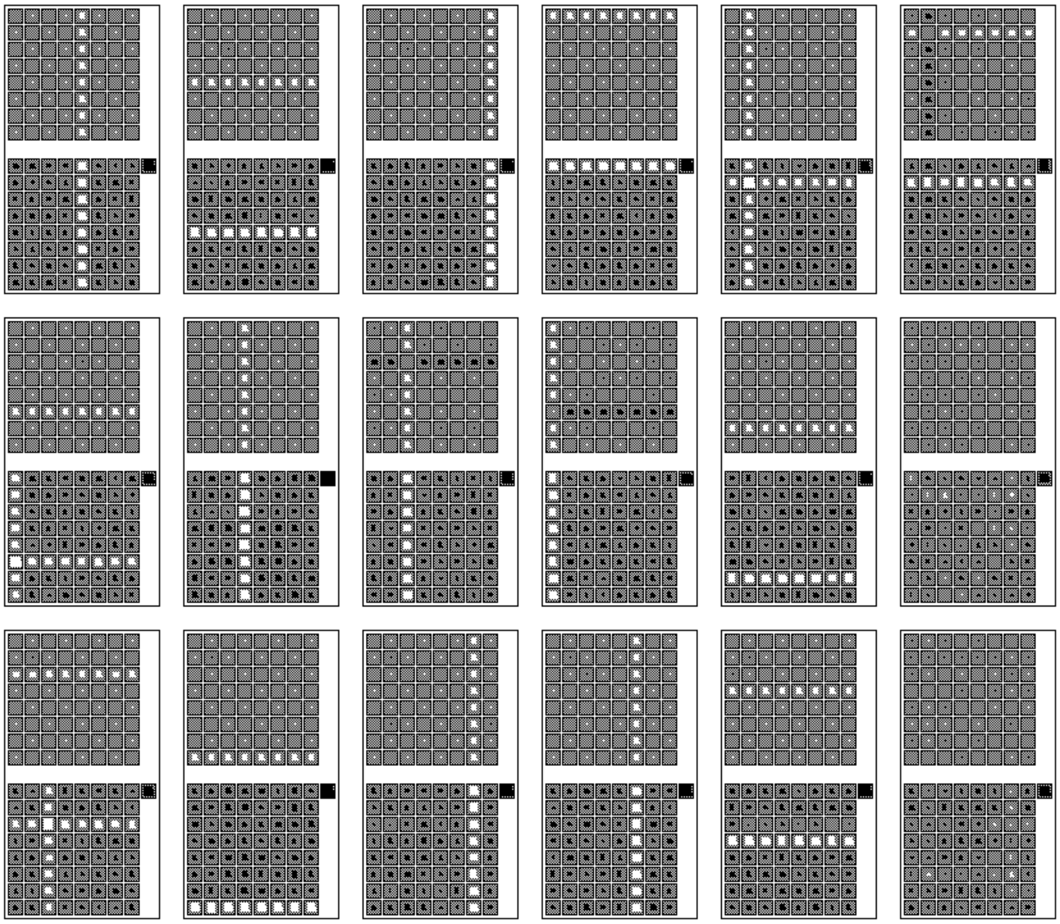

Figure 4.4: This figure shows the incoming and outgoing weights for each of the network's 18 representation units after learning using the first cost function. This network is stuck in a local minimum, where three pairs of units each respond to the same two lines. For example, the last two weight sets in the first row have a line in common. Note that the outgoing weights of these units nevertheless ensure good reconstruction of the inputs, as the activity of the right-hand unit of the pair cancels out the contribution from the other unit. The generalization length is 24.2 bits.

search, and the algorithm can then make these extra units cost nothing. In effect, the algorithm learns an appropriate architecture for the task.

When the network has less than 16 representation units, the algorithm produces a mixed encoding of the dataset, where some representation units respond to a pair of lines rather than a single line. This also tends to be a local minimum in which the algorithm gets stuck occasionally even when there are 16 or slightly more representation units. Figure 4.4 shows the incoming weights for a network with 18 representation units after training on the random lines training set. Note that two units have similar weights, and hence similar response properties. The resulting representation has a higher redundancy, due to the higher entropy of these units. The representation redundancy for this network is 0.16.

This algorithm is sensitive to the choice of the base probability parameter b. A low value of b encourages the representation units to have an activity close to zero on each case. This aids in learning the sparse, informative representations, as the high activity cost means that an active unit must significantly contribute to the reconstruction. As we increase the value of b, the sparseness of the learned representations gradually degrades.

Encoding with a local, adaptive rate

The second version of the MDL algorithm uses a more flexible method of encoding the activities of the representation units (see Equation 4.12). In effect, each unit's activity is encoded based on its own mean activity across the training set, which itself is encoded with respect to a global base rate. This algorithm has the advantage of increased flexibility in its coding.

This flexibility is both a blessing and a curse. If the coding prior for the base probabilities is broad, i.e., V_b in Equation 4.8 is high, each unit is completely free to choose its own base rate. This leads to the optimization getting stuck often in local minima. If we make the constraint tight by decreasing this parameter ($V_b = 0.01$), then this algorithm performs very similarly to the first algorithm. The residual flexibility in base rate proves useful, however, on the stochastic dataset. Here the base rates of the 16 lines are different due to sampling, and this algorithm outperforms the first version. The increased flexibility makes it cheap for the representation units to have slightly different means. Figure 4.5 shows the weights for a network containing 18 representation units trained on the stochastic dataset. This network successfully found the underlying generator.

4.3.5 Results on the dual-component dataset

On the dual-component task, both networks successfully found the underlying structure in the dataset. Figure 4.6 shows the incoming and outgoing weights into the 30 representation units of a network trained using the first version of the MDL algorithm on this task. The optimization required 244 weight-update iterations. The parameter settings were identical to those in the previous network: $b = 0.04$, $V_{\mathcal{R}} = 0.02$. This network perfectly reconstructed the remaining test case (reconstruction error of 0.0008). In the final representation, 26 of the units are each active for exactly 1 of every 13 training cases, while the other 4 units have a consistent activity of b. The generalization length in this case approximately equals the entropy of this representation, which is 10.53 bits; the redundancy $R_{REP} = (11.14 - 7.4)/7.4 = 0.51$.

Note that even though the network has learned the individual features underlying the input set, the resulting redundancy, while a considerable reduction from the redundancy of the input ($R_{IN} = 2.65$), indicates that it has still not discovered a non-redundant representation. The remaining redundancy turns out to be an inevitable consequence of applying this representational scheme to this dataset. While the units' activities are encoded independently, they are not actually independent, since if the unit representing the appropriate 1 of the 13 possible values on one half of the input is active, the other units are necessarily inactive. We will return to this issue in the following chapter.

4.3.6 Standard approaches

Standard unsupervised learning methods are unable to discover the structure in these datasets. Principal Components Analysis (PCA) finds some combination of the inputs which obscures their underlying components. Recalling that a linear reconstruction network performs a version of PCA,

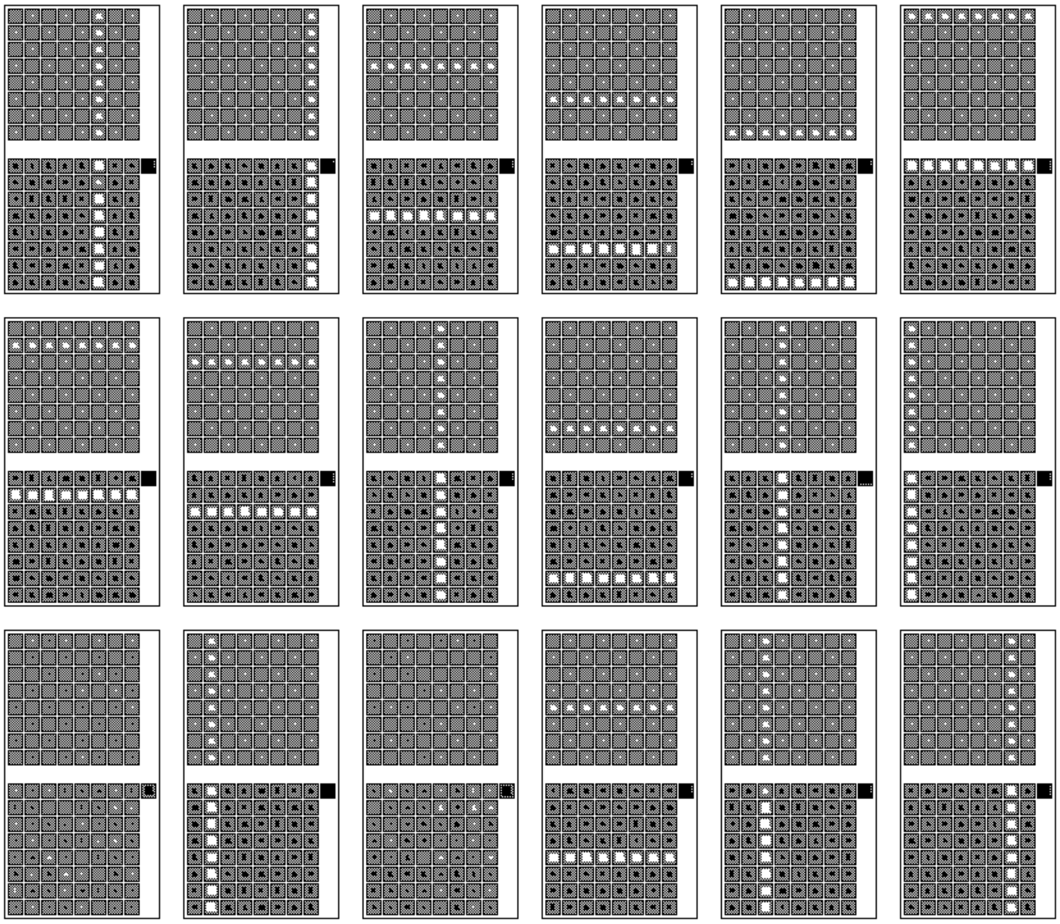

Figure 4.5: This figure shows the incoming and outgoing weights for each of the network's 18 representation units after learning using the second cost function. This network has allocated a representation unit to each of the 16 independent lines in the input set. The representation redundancy for this network is 0.01.