

Unsupervised learning of distributions on binary vectors using two layer networks

Yoav Freund
David Haussler

UCSC-CRL-94-25
June 22, 1994

Baskin Center for
Computer Engineering & Information Sciences
University of California, Santa Cruz
Santa Cruz, CA 95064 USA

ABSTRACT

We present a distribution model for binary vectors, called the influence combination model and show how this model can be used as the basis for unsupervised learning algorithms for feature selection. The model is closely related to the Harmonium model defined by Smolensky [RM86][Ch.6]. In the first part of the paper we analyze properties of this distribution representation scheme. We show that arbitrary distributions of binary vectors can be approximated by the combination model. We show how the weight vectors in the model can be interpreted as high order correlation patterns among the input bits. We compare the combination model with the mixture model and with principle component analysis. In the second part of the paper we present two algorithms for learning the combination model from examples. The first algorithm is based on gradient ascent. Here we give a closed form for this gradient that is significantly easier to compute than the corresponding gradient for the general Boltzmann machine. The second learning algorithm is a greedy method that creates the hidden units and computes their weights one at a time. This method is a variant of projection pursuit density estimation. In the third part of the paper we give experimental results for these learning methods on synthetic data and on natural data of handwritten digit images.

1. Introduction

Suppose that we are given a large (unordered) set of binary vectors and that we wish to find the types of correlations and redundancies that exist between the bits in these vectors. We assume that each binary vector is of the form $\vec{x} = \{x_1, \dots, x_n\} \in \{\pm 1\}^n$, and that each vector is generated independently at random according to some unknown distribution on $\{\pm 1\}^n$. Such an assumption is natural, for instance, when each instance consists of (possibly noisy) measurements of n different binary attributes of a randomly selected object. Our interest is in cases where the dimension n of the vectors is large, say $n > 50$. One example of this type of scenario is when the instances are binary images of handwritten digits, where each bit corresponds to the black or white color of a single pixel in the image. The correlations that we expect to see in this case correspond to the fact that the values of neighboring pixels or pixels that lie along lines or curves are strongly dependent on each other.

Knowledge of the correlations between different bits of the binary vector is useful when we want to use a set of measurements for various classification and prediction tasks. The idea that features that are useful for classification can be deduced from the distribution of typical inputs is the basis of several existing algorithms for unsupervised learning. One type of algorithm selects projections of the input based on Principle Component analysis [San89, Oja89]. Another type of algorithm clusters data based on an assumption that the underlying distribution is a mixture of Gaussians [EH81, Now90]. The combination model presented in this paper is related to both of these lines of work and has some advantages over each of them.

If we find a good model of the distribution, we can tackle other interesting learning problems, such as the problem of estimating the conditional distribution on certain components of the vector \vec{x} when provided with the values for the other components (a kind of regression problem), or predicting the actual values for certain components of \vec{x} based on the values of the other components (a kind of pattern completion task). In the example of the binary images presented above, this would amount to the task of recovering the value of a pixel whose value has been corrupted. We can often also use the distribution model to help us in a supervised learning task. This is because it is often easier to express the mapping of an instance to the correct label by using “features” that are correlation patterns among the bits of the instance. For example, it is easier to describe each of the ten digits in terms of patterns such as lines and circles, rather than in terms of the values of individual pixels, that are more likely to change between different instances of the same digit.

The process of learning an unknown distribution from examples is usually called *density estimation* or *parameter estimation* in statistics, depending on the nature of the class of distributions used as models. There has been considerable work on density/parameter estimation for distributions on real vector spaces (see e.g. [DH73]), and less on binary vector spaces. The most popular mainstream statistics models for distributions on $\{\pm 1\}^n$ for large n appear to be small mixtures of Bernoulli product distributions¹ [EH81, Now90], and models in which only k -wise dependencies between the components of the input are allowed,

¹A Bernoulli product distribution is a distribution over binary vectors in which each component is chosen independently of the rest.

for some $k \ll n$ [Fre87, CS89]. Newer and more exciting models include Bayes networks [Pea88] and Markov random fields [Pea88, GG84, Gem86]. In the neural network area, both Hopfield nets [Hop82] and Boltzmann machines [AHS85] can be used as models of probability distributions on $\{\pm 1\}^n$ for relatively large n . We will look at a class of models defined by a special type of Boltzmann machine.

Hopfield networks, Boltzmann machines and Markov random fields are all based on the statistical physics concepts of energy and local interaction between units whose state is binary.² The models defined by Hopfield networks and Boltzmann Machines are special cases of the more general Markov random field model. The units in a Hopfield network correspond to the bits of the binary vectors and the interaction between units are restricted to symmetric pairwise interactions. Boltzmann machines also employ only pairwise interactions, but in addition to the units that correspond to bits of the data vectors, commonly called the *input* units, there are *hidden* units, which correspond to unobserved binary variables. These hidden units interact with the input units and generate correlations between the vector bits that the input units represent. The distribution of the binary vectors generated by the Boltzmann Machine is defined as the marginal distribution induced on the state of the input units by the Markov random field over all units, both observed and hidden.

While the Hopfield network is relatively well understood, it is limited in the types of distributions that it can model. On the other hand, Boltzmann machines are universal in the sense that they are powerful enough to model any distribution (to any degree of approximation), but the mathematical analysis of their capabilities is often intractable. Moreover, the standard learning algorithm for the Boltzmann machine, a gradient ascent heuristic to compute the maximum likelihood estimates for the weights and thresholds, requires repeated stochastic approximation, which results in unacceptably slow learning. Many methods have been proposed to speed up learning in Boltzmann machines. One of these methods is the mean-field approximation [PA87]. In Section 2.2 we shall see some relations between one of our learning rules and the mean field approximation.

In our research we have attempted to narrow the gap between Hopfield networks and Boltzmann machines by finding a model that will be powerful enough to be universal, yet simple enough to be analyzable and computationally efficient.³ The model that we use in this work is essentially a Boltzmann Machine whose connection graph is bipartite. There are two types of nodes: “input” nodes and “hidden” nodes. Each input node is connected to each of the hidden nodes, and no other connections exist. We call

²Informally, a Markov random field consists of a set of random variables that are connected as nodes in a graph. The distribution of each random variable is determined by the value of its neighbors. In other words, given the value of all the neighbors of random variables, the value of the random variable is independent of the state of the rest of the random variables. The Markov process is a special case of the Markov field in which each random variable corresponds to a specific time step and its neighbors are the random variables that correspond to the previous and the succeeding time steps. In general, Markov-field distributions have a canonical description that is based on the concept of interaction energy.

³Recent work on modeling correlations by hidden units has also been done by Radford M. Neal [Nea90]. In his work he gives a different variant of the Boltzmann Machine algorithm that uses distribution models similar to Judea Pearl’s Bayes Networks [Pea88, GP87]. His model is superior to the Boltzmann Machine in the sense that the connection weights are interpreted as conditional probabilities, which is a more accessible interpretation than local energy interactions. The learning algorithms that Neal used are based on stochastic approximation. The question of whether a two-layer model of this type has universal representation capabilities is open.

this model the *influence combination machine*, or, for short, the combination machine. We refer to the distribution that is defined on the binary vectors by the combination machine as the *combination model*. This type of Boltzmann machine was previously studied by Smolensky in his *harmony theory* [RM86][Ch.6]. In his work he discusses several possible ways of using this type of model for solving problems in Artificial Intelligence. In our work we concentrate on the mathematical properties of the model and on efficient algorithms for learning the model from random instances.

The combination machine consists of two types of units: input units, each of which holds one component of the input vector, and hidden units, representing hidden variables. There is a weighted connection between each input unit and each hidden unit, and no connections between input units or between hidden units (see Figure 2.1). The presence of the hidden units induces dependencies, or correlations, between the variables modeled by input units. To illustrate the representational power of the combination model, consider the distribution of people that visit a specific coffee shop on Sunday. Let each of the n input variables represent the presence (+1) or absence (-1) of a particular person that Sunday. These random variables are clearly not independent. For example, if Fred's wife and daughter are there, it is more likely that Fred is there as well, if you see three members of the golf club, you expect to see other members of the golf club, if Bill is there, you are unlikely to see his ex-wife Brenda there, etc. This situation can be modeled by a combination model in which each hidden variable represents the presence or absence of a social group. The weights connecting a hidden unit and an input unit measure the tendency of the corresponding person to be associated with the corresponding group. In this coffee shop situation, several social groups may be present at the same time, exerting a combined influence on the distribution of customers. In Sections 2.3 and 2.4 we discuss why the combination model is better for describing this type of distributions than popular models such as the mixture model and principal components methods.⁴

We show that the combination model is a universal model in the sense that any probability distribution on $\{\pm 1\}^n$ can be represented by a combination model with n input units to within any desired accuracy. Then we show that the standard Boltzmann machine learning rule, when applied to a combination model, can be computed in closed form, instead of using random sampling techniques. Thus we get a faster learning algorithm than the standard Boltzmann rule that is also exact. The computational complexity of the learning rule is exponential in the number of hidden units. However, under certain natural conditions we show that there exists a good approximation that requires only polynomial time.

We then explore the relationships between the distributions generated by the combination model and those studied in Projection Pursuit density estimation [Hub85, FWS84, Fri87]. We show that the search for hidden variables that have a strong influence on the input distribution can be interpreted as a search for projections of the input that have a non-Normal marginal distribution. Based on this observation, we propose a learning algorithm based on exploratory projection pursuit for the combination model. This method is a greedy method that adds a single hidden unit at a time to the model. The time complexity of this method is linear in the number of hidden units compared to the exponential complexity of the gradient based method. However, while the gradient based method is guaranteed to converge to a local minimum in the model space, the projection pursuit method does not have this guarantee.

⁴Noisy-OR gates have been introduced in the framework of Bayes Networks to allow for such combinations [Pea88]. However, using this in networks with hidden units has not been studied, to the best of our knowledge.

We conclude this paper with results of some experiments. The first set of experiments compare the two learning algorithms on synthetically generated data, and demonstrate their advantages and deficiencies. The second set of experiments compare the performance of the combination model to that of the mixture model and demonstrate the difference in the type of distribution representations that they generate.

2. The influence combination distribution model

2.1 Notation

For the most part, we use standard algebraic notation in our formulas. Elements from the n -dimensional spaces R^n and $\{-1, +1\}^n$ are denoted by vectors \vec{x}, \vec{y}, \dots . We denote by $\|\vec{x}\|_1, \|\vec{x}\|_2$ the l_1 and l_2 norms of \vec{x} , and by $\vec{x} \cdot \vec{y}$ the dot product between two vectors. We use the standard hyperbolic trigonometric functions

$$\sinh(x) = \frac{e^x - e^{-x}}{2}, \quad \cosh(x) = \frac{e^x + e^{-x}}{2}, \quad \tanh(x) = \frac{\sinh(x)}{\cosh(x)}.$$

We denote the natural base logarithm by “ln”. Finally, we use the function $\text{logistic}(x) = 1/(1 + \exp(-x))$ that is commonly used in the definition of Boltzmann Machines.

2.2 The Model

In this section we present the combination machine and the corresponding distribution model, which is the influence combination distribution model. The combination machine is a simple Connectionist type model which is a special case of the Boltzmann Machine [AHS85]. As we shall see, the simplicity of this special case makes it easier to analyze than the general Boltzmann machine and allows the use of more efficient learning algorithms. At the same time, the model is still powerful enough to approximate any distribution of binary vectors.

To model a distribution on $\{\pm 1\}^n$ we use a machine with $n + m$ units. There are two types of units, n *input* units, each of which represents a single bit in the random vector, and m *hidden* units, whose role is the create correlation between the values of the input units. These units are connected in a bipartite graph as illustrated in Figure (2.1).

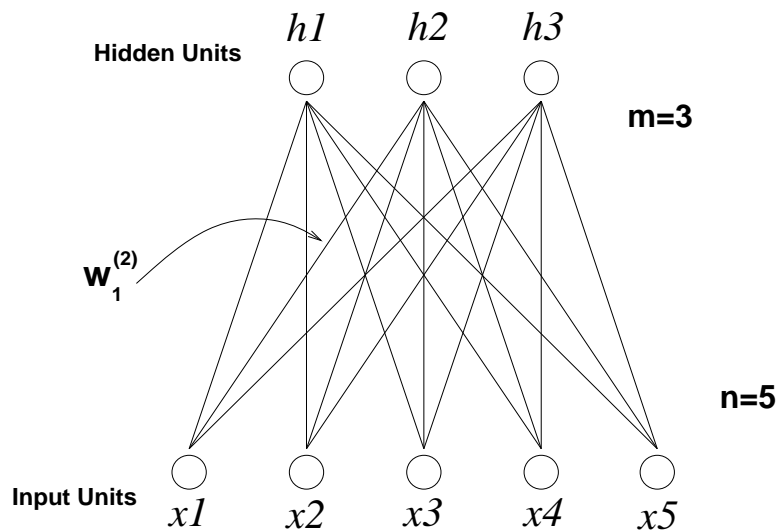


Figure 2.1: The bipartite graph of the combination model

The random variables represented by the input units each take values in $\{+1, -1\}$, while the hidden variables, represented by the hidden units, take values in $\{0, 1\}$. The *state* of the machine is defined by the values of these random variables. We denote by $\vec{x} = (x_1, \dots, x_n) \in \{\pm 1\}^n$ the state of the input units, and by $\vec{h} = (h_1, \dots, h_m) \in \{0, 1\}^m$ the state of the hidden units.

There are $m(n+1)$ real-valued parameters associated with the machine. Each particular setting of these parameters defines the *parameter vector* of the machine. Each parameter vector defines a distribution on the states of the machine. Summing over the state of the hidden units we get a distribution on the input units, which is the influence combination distribution defined by the particular parameter vector. There are two variants of the combination model, which we call the binary valued and the real valued combination machines. While we are mostly interested in the binary model, the real valued model is a useful approximation in some cases.

The parameters are all real-valued and are defined as follows. There is a *weight* parameter associated with each edge in the bipartite graph. We denote by $\omega_j^{(i)}$ the weight of the edge connecting the i th hidden unit to the j th input unit. We also use $\vec{\omega}^{(i)}$ to denote the vector of all n weights associated with the i th hidden unit.¹ There is a *bias* parameter associated with each hidden unit. We denote the bias of the i th hidden unit by $\theta^{(i)} \in R$. The complete parameter vector of a binary combination model is denoted by $\phi_B = \{(\vec{\omega}^{(1)}, \theta^{(1)}), \dots, (\vec{\omega}^{(m)}, \theta^{(m)})\}$. For a given ϕ_B , the *energy* of a state of the combination machine is defined as

$$E(\vec{x}, \vec{h} | \phi_B) = - \sum_{i=1}^m (\vec{\omega}^{(i)} \cdot \vec{x} + \theta^{(i)}) h_i \quad (2.1)$$

and the probability of a state is defined to be

$$Pr(\vec{x}, \vec{h} | \phi_B) = \frac{1}{Z_B} e^{-E(\vec{x}, \vec{h} | \phi_B)} \quad \text{where} \quad Z_B = \sum_{\vec{x}, \vec{h}} e^{-E(\vec{x}, \vec{h} | \phi_B)}.$$

We find it useful to define the “combined weight” of a particular state of the hidden units as the sum of the weight vectors corresponding to the hidden units whose state is 1:

$$\vec{\omega}(\vec{h}) = \sum_{i=1}^m h_i \vec{\omega}^{(i)}$$

Plugging the definition of the energy into the definition of Z_B , we get that

$$Z_B = \sum_{\vec{x}, \vec{h}} \exp \left(\sum_{i=1}^m (\vec{\omega}^{(i)} \cdot \vec{x} + \theta^{(i)}) h_i \right).$$

Expanding the sum in the exponent we get that

$$Z_B = \sum_{\vec{h} \in \{0,1\}^m} \left(\exp \left(\sum_{i=1}^m h_i \theta^{(i)} \right) \sum_{\vec{x} \in \{-1,+1\}^n} \exp(\vec{x} \cdot \vec{\omega}(\vec{h})) \right).$$

¹In [RM86][Ch.6], binary connection weights are used, here we use real-valued weights.

Expanding the sum over $\vec{x} \in \{-1, +1\}^n$, we get that

$$Z_B = \sum_{\vec{h} \in \{0,1\}^m} \left(\exp\left(\sum_{i=1}^m h_i \theta^{(i)}\right) \prod_{j=1}^n (\exp(\vec{\omega}(\vec{h})_j) + \exp(-\vec{\omega}(\vec{h})_j)) \right),$$

where $\vec{\omega}(\vec{h})_j$ denotes the j th component of $\vec{\omega}(\vec{h})$. Using the definition of $\cosh(x)$, we can rewrite the last expression as

$$Z_B = 2^n \sum_{\vec{h} \in \{0,1\}^m} \left[\exp\left(\sum_{i=1}^m h_i \theta^{(i)}\right) \prod_{j=1}^n \cosh(\vec{\omega}(\vec{h})_j) \right]. \quad (2.2)$$

Note that the trivial model, in which there are no hidden units, defines the uniform distribution over the state vectors \vec{x} . In the general case the probability distribution over possible state vectors on the input units is given by

$$Pr(\vec{x}|\phi_B) = \sum_{\vec{h} \in \{0,1\}^m} Pr(\vec{x}, \vec{h}|\phi_B) = \frac{1}{Z_B} \sum_{\vec{h} \in \{0,1\}^m} \exp\left(\sum_{i=1}^m (\vec{\omega}^{(i)} \cdot \vec{x} + \theta^{(i)}) h_i\right) \quad (2.3)$$

By separating the sum over \vec{h} into a sum over all \vec{h} such that $h_m = 0$ and a sum over all \vec{h} such that $h_m = 1$, we can rewrite Equation (2.3) in the following form:

$$Pr(\vec{x}|\phi_B) = \frac{1}{Z_B} \left(e^0 + e^{\vec{\omega}^{(m)} \cdot \vec{x} + \theta^{(m)}} \right) \sum_{\{h_1, \dots, h_{m-1}\} \in \{0,1\}^{m-1}} \exp\left(\sum_{i=1}^{m-1} (\vec{\omega}^{(i)} \cdot \vec{x} + \theta^{(i)}) h_i\right)$$

Repeating this manipulation for all m components of \vec{h} we get that

$$Pr(\vec{x}|\phi_B) = \frac{1}{Z_B} \prod_{i=1}^m \left(1 + e^{\vec{\omega}^{(i)} \cdot \vec{x} + \theta^{(i)}} \right). \quad (2.4)$$

Equation (2.4) is a simple closed form representation of the distribution defined by the parameter vector ϕ_B . Notice that the hidden unit variables, h_i , are not explicitly present in this formula. Each factor in the product is associated with one hidden unit in the corresponding machine. This product form is particular to the combination model, and does not hold for general Boltzmann machines. Product form distribution models have been used for density estimation in Projection Pursuit [Hub85, FWS84, Fri87]. We shall look further into this relationship in Section 3.2.

In some of the following discussion we shall find it useful to use a variant of the combination model that defines distributions over the whole real space R^n , i.e. to allow each input to have any real-value instead of limiting it to only $+1$ and -1 . The structure of the machine is the same, we keep the hidden variables $\{0,1\}$ -valued, and the distribution is defined in a similar way, but the energy function has an extra term that is necessary for ensuring that the distribution can be normalized. This term corresponds to each input unit having a connection of strength -1 to itself. To differentiate between the binary and the real-valued models we subscript quantities relating to the real-valued model by R . The energy of a particular state of the real-valued model is given by

$$E(\vec{x}, \vec{h}|\phi_R) = - \left(\sum_{i=1}^m (\vec{w}^{(i)} \cdot \vec{x} + \theta^{(i)}) h_i \right) + \frac{1}{2} \|\vec{x}\|_2^2, \quad (2.5)$$

which produces the following distribution over the R^n :

$$Pr(\vec{x}|\phi_R) = \sum_{\vec{h} \in \{0,1\}^m} Pr(\vec{x}, \vec{h}|\phi_R) = e^{-\frac{1}{2}\|\vec{x}\|_2^2} \frac{1}{Z_R} \prod_{i=1}^m \left(1 + e^{\vec{w}i \cdot \vec{x} + \theta^{(i)}}\right), \quad (2.6)$$

where

$$\begin{aligned} Z_R &= \int_{R^n} \sum_{\vec{h} \in \{0,1\}^m} \exp\left(-E(\vec{x}, \vec{h}|\phi_R)\right) d\vec{x} \\ &= \sum_{\vec{h} \in \{0,1\}^m} \int_{R^n} \exp\left(-\frac{1}{2}\|\vec{x}\|_2^2 + \sum_{i=1}^m (\vec{w}i \cdot \vec{x} + \theta^{(i)})h_i\right) d\vec{x} \\ &= (2\pi)^{n/2} \sum_{\vec{h} \in \{0,1\}^m} \exp\left[\sum_{i=1}^m h_i \theta^{(i)} + \frac{1}{2}\|\vec{\omega}(\vec{h})\|_2^2\right], \end{aligned} \quad (2.7)$$

$$(2.8)$$

using the integral of the Gaussian distribution.

We discuss the relation between the real-valued and the binary-valued model in Section 2.6.

2.3 Discussion of the model

The right hand side of Equation (2.4) has a simple intuitive interpretation. The i th factor in the product corresponds to the hidden variable h_i and is an increasing function of the dot product between \vec{x} and the weight vector of the i th hidden unit. Hence an input vector \vec{x} will tend to have large probability when it is in the direction of one of the weight vectors $\vec{w}i$ (i.e. when $\vec{w}i \cdot \vec{x}$ is large), and small probability otherwise. This is the way that the hidden variables can be seen to exert their "influence"; each corresponds to a preferred or "prototypical" direction in space. The bias parameter $\theta^{(i)}$, together with the length $\|\vec{w}i\|_2$ of the weight vector, control the strength of the influence of the i th hidden variable in comparison with the other hidden variables, as well as its "width", i.e. how close in direction \vec{x} has to be to $\vec{w}i$ before it significantly influences its probability. Increasing either $\|\vec{w}i\|_2$ or $\theta^{(i)}$ increases the strength of the influence of the hidden unit. Decreasing $\theta^{(i)}$ and, at the same time, increasing $\|\vec{w}i\|_2$, decreases the "width" of the influence, making the influence of the i th hidden unit more restricted to input vectors whose direction is very close to the direction of $\vec{w}i$. This is true for both the binary-valued and the real-valued combination models.

Equation (2.3) shows that the combination model can be written as a mixture of 2^m distributions of the form

$$\frac{1}{Z(\vec{h})} \exp\left(\sum_{i=1}^m (\vec{\omega}^{(i)} \cdot \vec{x} + \theta^{(i)})h_i\right),$$

where $\vec{h} \in \{0,1\}^m$ and $Z(\vec{h})$ is the appropriate normalization factor. Each of these distributions is a product of n Bernoulli distribution, i.e. the x_j is drawn independently at random and attains a value of -1 or $+1$ with probabilities $\text{logistic}(-2\vec{\omega}(\vec{h})_j)$ and $\text{logistic}(+2\vec{\omega}(\vec{h})_j)$ respectively, which implies that the mean of x_j in according to this distribution is $\tanh(\vec{\omega}(\vec{h})_j)$. We shall refer to this type of distribution as a "Bernoulli product distribution". The combination model is a mixture of 2^m Bernoulli product distributions, each corresponding to a setting of \vec{h} and each having a mixture coefficient $Z(\vec{h})$.

It is interesting to compare the class of combination models to the standard class of models defined by a mixture of Bernoulli product distributions. The same bipartite graph described in Figure (2.1) can be used to define a standard mixture model. Assign each of the m hidden units a weight vector \vec{w}_i and a probability p_i such that $\sum_{i=1}^m p_i = 1$. To generate an example, choose *one* of the hidden units according to the distribution defined by the p_i 's, and then choose the vector \vec{x} according to $P_i(\vec{x}) = \frac{1}{Z_i} e^{\vec{w}_i \cdot \vec{x}}$, where Z_i is the appropriate normalization factor so that $\sum_{\vec{x} \in \{\pm 1\}^n} P_i(\vec{x}) = 1$. We thus get the distribution

$$P(\vec{x}) = \sum_{i=1}^m \frac{p_i}{Z_i} e^{\vec{w}_i \cdot \vec{x}} \quad (2.9)$$

This form for presenting the standard mixture model emphasizes the similarity between this model and the combination model. A vector \vec{x} will have large probability if the dot product $\vec{w}_i \cdot \vec{x}$ is large for some $1 \leq i \leq m$ (so long as p_i is not too small). However, unlike the standard mixture model, the combination model allows more than one hidden variable to be +1 for any generated example. This means that several hidden influences can combine in the generation of a single example, because several hidden variables can be +1 at the same time.

To see why this is useful, consider two examples. First, consider the coffee shop example given in the introduction. At any moment of time it is reasonable to find *several* social groups of people sitting in the shop. The combination model will have a natural representation for this situation, while in order for the standard mixture model to describe it accurately, a hidden variable has to be assigned to each combination of social groups that is likely to be found in the shop at the same time. Similarly, when we want to represent the distribution of binary images of digits, it is reasonable to assume that each specific image contains *several* patterns, such as lines and curves. Of course, the whole digit can be perceived as a pattern, in which case the mixture model is the relevant distribution model. However, we claim that it is often more appropriate to represent each digit image as a combination of patterns rather than a single pattern. In other words, we claim that, for typical sets of images of digits, the maximal likelihood combination model will have larger likelihood than a mixture model with the same number of parameters. In Section 4 we give experimental evidence to support this claim. In cases where this claim is correct the combination model is exponentially more succinct than the standard mixture model, and naturally captures the underlying product structure of the distribution. Of course, if the space of hidden variables does not have a product structure of this type, then the combination model is no better than the standard mixture model.

In analogy with the binary combination model, the real-valued combination model can be shown to represent a mixture of 2^m symmetric Gaussian distributions. From Equation (2.6) we get that for the empty case, $m = 0$, where there are no hidden variables present, the distribution is a symmetric Gaussian by definition. When a single hidden variable is present, the distribution becomes

$$\begin{aligned} Pr(\vec{x} | \phi_R) &= e^{-\frac{1}{2} \|\vec{x}\|_2^2} \frac{1}{Z} \left(1 + e^{\vec{w}_1 \cdot \vec{x} + \theta^{(1)}} \right) = \frac{1}{Z} \left(e^{-\frac{1}{2} \|\vec{x}\|_2^2} + e^{-\frac{1}{2} \|\vec{x}\|_2^2 + \vec{w}_1 \cdot \vec{x} + \theta^{(1)}} \right) = \\ &= \frac{1}{Z} \left(e^{-\frac{1}{2} \|\vec{x}\|_2^2} + e^{-\frac{1}{2} \|\vec{x} - \vec{w}_1\|_2^2 + \frac{1}{2} \|\vec{w}_1\|_2^2 + \theta^{(1)}} \right). \end{aligned}$$

This is a mixture of two Gaussians, both of which have spherical symmetry. They differ only in the location of the mean, which is $\vec{0}$ for the first component and \vec{w}_1 for the second component, and in their relative

probabilities (mixture weights). Each additional hidden unit has the effect of transforming the previous distribution into a mixture of two distributions, one is the previous distribution, and the other is the original distribution shifted by \vec{w}_i (See Figure 2.2).

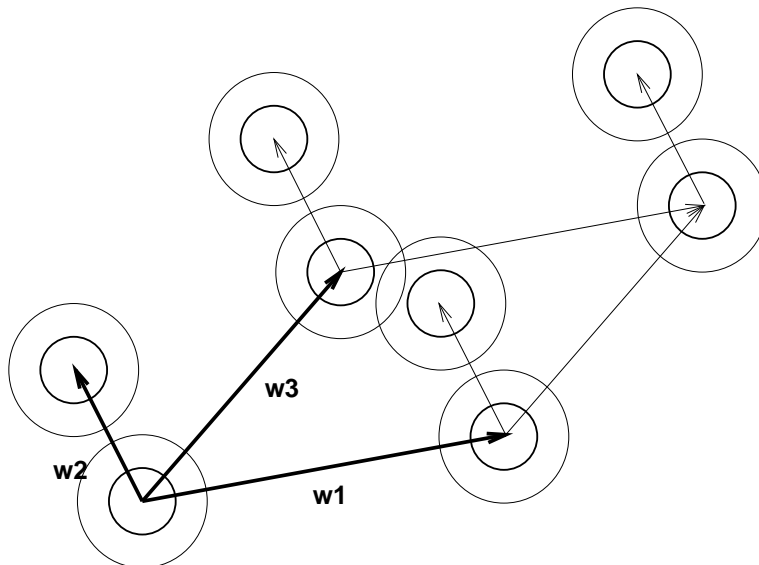


Figure 2.2: The distribution over R^2 generated by a (real valued) combination model with three hidden units. Each pair of concentric circles denotes a single Gaussian distribution. The distribution defined by the combination model is a mixture of these eight Gaussians.

In the general case the combination model with m hidden units is equivalent to a mixture of 2^m Gaussians whose expected values are located at the combined weight, $\vec{\omega}(\vec{h})$, corresponding to each of the 2^m possible states of \vec{h} .² This interpretation of the real-valued model will be used in Section (3.5) in a Projection Pursuit algorithm for learning the combination model.

2.4 Comparison with principal components analysis

Principal Component Analysis (PCA) is a popular method for the analysis of high order correlations (see e.g. [Jol86]). Many algorithms for unsupervised learning are based on this method, among them some learning rules for neural networks [San89, Oja89]. The method is based on the covariance matrix, which measures pairwise correlations among input bits. The main assumption underlying the method is that the low dimension projections of the data that retain the largest amount of information are those projections that have the largest variance. One justification of this assumption is that if the data has a simple enough distribution such as a Gaussian distribution then the reconstruction of the original input from its projections is optimal for this choice of projections. The directions with largest variance are equal to the directions of the eigenvectors of the covariance matrix that have the largest eigenvalues.

²Compare this to the mixture of Bernoulli products whose expected values are $\tanh(\vec{\omega}(\vec{h}))$. A more detailed comparison of the two models will be given in Section 2.6.

The neural network implementation of PCA is usually a two layer network with the same architecture as the combination model. The learning rule, however, is different, and tries to make the weight vectors of the hidden units equal to eigenvectors of the covariance matrix of the input. The outputs of the hidden units are thus projections of the data (or a nonlinear transformation of such projections).

This type of network is capable of representing each input as a combination of correlation patterns. In this sense it is as powerful as the combination model and does not suffer from the deficiencies of mixture models described in the previous section. However, as this method of analysis is based only on the second order correlations among pixels it necessarily ignores part of the structure of the distribution. In the combination model, on the other hand, each hidden unit can represent correlations of arbitrary order. We claim that some natural distributions have strong high order correlation and that taking into account only the second order correlations ignores some of the most important information available in the distribution. In Section 4 we shall give some experimental evidence to support this claim.

2.5 Universality of the model

Despite its limited connectivity, it is not hard to show that the class of binary combination models is universal in the sense that for every n and every distribution on $\{\pm 1\}^n$ there is a combination model with n input units that approximates that distribution to within any desired accuracy. The argument is similar to an argument for the same claim regarding the class of mixtures of Bernoulli product distributions.

Assume first that the distribution we want to estimate is $Pr(\vec{x}) = p$ for $\vec{x} = (1, 1, \dots, 1)$ and $Pr(\vec{x}) = \frac{1-p}{2^n-1}$ for $\vec{x} \neq (1, 1, \dots, 1)$. Here we need only one hidden unit. We define $q = \frac{p(2^n-1)}{1-p}$ and choose $\vec{\omega}^{(1)} = (a, a, \dots, a)$ and $\theta^{(1)} = -na + \ln(q-1)$, where $a = \frac{1}{2} \ln(q-1) + \frac{1}{2} \ln(1/\epsilon)$. We get the following values for $f(\vec{x}) := 1 + e^{\vec{\omega}^{(1)} \cdot \vec{x} + \theta^{(1)}}$.

If $\vec{x} = (+1, +1, \dots, +1)$, then $f(\vec{x})$ is equal to q . For a vector where exactly one component is equal to -1 and all the rest are $+1$, $f(\vec{x})$ is equal to $1 + \epsilon$, and for a vector \vec{x} which has k components that are equal to -1 , $f(\vec{x})$ is equal to $1 + (q-1)(\epsilon/(q-1))^k \leq 1 + \epsilon^k$. By setting ϵ small enough we can make $1 + e^{\vec{\omega}^{(1)} \cdot \vec{x} + \theta^{(1)}}$ arbitrarily close to 1 for all $\vec{x} \neq \vec{x} = (+1, +1, \dots, +1)$. Normalizing the distribution to sum to 1 we can get a distribution that is arbitrarily close to the desired distribution.

To approximate an arbitrary distribution, we multiply 2^n factors, each approximating a distribution that is highly concentrated on a single setting of \vec{x} and almost uniform on all other settings. By appropriate choice of the parameters we can approximate the arbitrary distribution closely for each value of \vec{x} . Of course this requires exponentially many hidden units, but this is unavoidable since it requires an exponential number of parameters to specify an arbitrary distribution over $\{\pm 1\}^n$ in any reasonable parametric model.

Of course, we are interested in cases where the distribution of the data can be represented well by a *small* combination model. While a general distribution might require many hidden units to model it, distributions that are encountered in nature are often simple, and can be modeled well by a model that has only a small number of hidden units. In Section 4 we show that the distribution of images of handwritten digits can be approximated well by a combination model with few hidden units.

2.6 Relations between the binary-valued and the real-valued models

Two variants of the combination model were introduced in Section 2.2, the binary-valued model (Equations (2.1) to (2.4)) and the real-valued model (Equations (2.5) to (2.8)). The binary-valued model is the natural model for representing distributions of binary vectors, and thus, ideally, we would like to use only this model. On the other hand, the real-valued model has properties that make it possible to use more efficient learning algorithms to learn it. As we show in this section, the real-valued model is an approximation of the binary model when the weights are all small. Thus we can use the algorithms for the real-valued model to find an approximate parameter vector of the binary model.

The real-valued model defines a density function, in contrast with the binary model, which defines a point mass distribution. However, the ratio between the densities assigned by a real-valued model to any pair of points in $\{-1, +1\}^n$ is equal to the ratio of the probabilities assigned to the same points by a binary model with the same parameters. This is because the factor of $e^{-\frac{1}{2}\|\vec{x}\|_2^2}$ in the density function is equal to $e^{-n/2}$ for all vectors in $\{-1, +1\}^n$.

This does not mean that the maximum-likelihood parameter vector for a given set of examples is equal for both models. This is because the normalization factors Z_B and Z_R are different for each of the two cases. However, as we shall now see, when the weight vectors $\vec{\omega}$ are small the normalization factors are very close to each other.

Recall Equation (2.2):

$$Z_B = 2^n \sum_{\vec{h} \in \{0,1\}^m} \left[\exp \left(\sum_{i=1}^m h_i \theta^{(i)} \right) \prod_{j=1}^n \cosh \left(\vec{\omega}(\vec{h})_j \right) \right].$$

The Taylor expansion of $\cosh(x)$ around $x = 0$ is:

$$\cosh(x) = 1 + \frac{x^2}{2!} + \frac{x^4}{4!} + \frac{x^6}{6!} + \dots$$

thus the first order approximation of Z_B for small values of $\omega_j^{(i)}$ is:

$$Z_B \approx 2^n \sum_{\vec{h} \in \{0,1\}^m} \left[\exp \left(\sum_{i=1}^m h_i \theta^{(i)} \right) \prod_{j=1}^n \left(1 + \frac{1}{2} \left(\vec{\omega}(\vec{h})_j \right)^2 \right) \right]$$

On the other hand, note that Equation (2.8) can also be written as:

$$Z_R = (2\pi)^{n/2} \sum_{\vec{h} \in \{0,1\}^m} \left[\exp \left(\sum_{i=1}^m h_i \theta^{(i)} \right) \prod_{j=1}^n \exp \left(\frac{1}{2} \left(\vec{\omega}(\vec{h})_j \right)^2 \right) \right],$$

The Taylor expansion of $\exp(x)$ around $x = 0$ is

$$\exp(x) = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \dots$$

thus the first order approximation of Z_R for small values of $\omega_j^{(i)}$ is:

$$Z_R \approx (2\pi)^{n/2} \sum_{\vec{h} \in \{0,1\}^m} \left[\exp \left(\sum_{i=1}^m h_i \theta^{(i)} \right) \prod_{j=1}^n \left(1 + \frac{1}{2} (\vec{\omega}(\vec{h})_j)^2 \right) \right] \approx \left(\frac{\pi}{2} \right)^{n/2} Z_B .$$

The fact that the two models differ by a constant factor is of no consequence when looking for the maximal-likelihood parameter vector, because this constant factor disappears in the derivative of the log of the likelihood.

The difference between the two approximations is of the order of $\|\vec{\omega}(\vec{h})\|_2^4$. Thus if $\|\vec{\omega}(\vec{h})\|_2$ is much smaller than 1 the approximation is reasonable.

There is another way in which the two models can be compared. In Section 2.3 we have shown that both the real-valued and the binary-valued combination models are equivalent to mixture models. The real-valued model is equal to a mixture of 2^m Gaussian distributions. Each mixture component corresponds to a setting of \vec{h} and has an expected value of $\vec{\omega}(\vec{h})$. Similarly, the binary-valued combination model is equivalent to a mixture of 2^m Bernoulli product distributions, each of which has an expected value of $\tanh(\vec{\omega}(\vec{h}))$. When $\vec{\omega}(\vec{h})$ is small $\tanh(\vec{\omega}(\vec{h})) \approx \vec{\omega}(\vec{h})$.

It is easy to show that every one dimensional projection of a Gaussian distribution generates a Normal marginal distribution. Thus the marginal distribution that is generated by the real-valued combination model is a mixture of normal distributions. Diaconis and Friedman [DF84] have shown that, in some sense, *most* “well-behaved” distributions generate a marginal distribution that is close to normal when projected on a randomly chosen direction. In particular, the uniform distribution on the 2^n binary vectors in $\{-1, +1\}^n$ generates, with very high probability, a marginal distribution that is close to the normal distribution, when the projection direction is chosen uniformly at random from the n dimensional sphere, and n is large. In Appendix A, we show that this is also true for Bernoulli product distributions, if the distributions of the individual coordinates are not too biased. Thus, under reasonable assumptions, the marginal distribution that is generated by the binary valued combination model is also mixture of normal distributions.

In addition, if the weight vectors, \vec{w}_i , are short, then the mixture coefficients and the means of the mixture components of the two models are close, which implies that the projections of the distributions defined by the real-valued model and the binary valued model with the same parameters are very close to each other. We use this correspondence in our analysis of the projection pursuit learning methods, which are based on properties of projections of the data.

3. Learning the model from examples

3.0.1 Learning by gradient ascent on the log-likelihood

We now suppose that we are given a sample consisting of a set S of vectors in $\{\pm 1\}^n$ drawn independently at random from some unknown distribution. Our goal is to use the sample S to find a good model for this unknown distribution using a combination model with m hidden units, if possible. The method we investigate here is the method of maximum likelihood estimation using gradient ascent. The goal of learning is reduced to finding the set of parameters for the combination model that maximizes the (log of the) probability of the set of examples S . In fact, this gives the standard learning algorithm for general Boltzmann machines [AHS85]. For a general Boltzmann machine this would require stochastic estimation of the parameters. As stochastic estimation is very time-consuming, the result is that learning is very slow. In this section we show that stochastic estimation need not be used for the combination model.

From Equation (2.4), the log of the likelihood of a sample of input vectors $S = \{\vec{x}^{(1)}, \vec{x}^{(2)}, \dots, \vec{x}^{(N)}\}$, given a particular setting $\phi_B = \{(\vec{\omega}^{(1)}, \theta^{(1)}), \dots, (\vec{\omega}^{(m)}, \theta^{(m)})\}$ of the parameters of the model is:

$$\text{log-likelihood}(\phi_B) = \sum_{\vec{x} \in S} \ln Pr(\vec{x} | \phi_B) = \sum_{i=1}^m \left(\sum_{\vec{x} \in S} \ln(1 + e^{\vec{\omega}^{(i)} \cdot \vec{x} + \theta^{(i)}}) \right) - N \ln Z_B. \quad (3.1)$$

Taking the gradient of the log-likelihood results in the following formulas. For the bias parameters we get:

$$\frac{\partial}{\partial \theta^{(i)}} \text{log-likelihood}(\phi_B) = \sum_{\vec{x} \in S} \frac{1}{1 + e^{-(\vec{\omega}^{(i)} \cdot \vec{x} + \theta^{(i)})}} - N \sum_{\vec{x} \in \{\pm 1\}^n} Pr(\vec{x} | \phi_B) \frac{1}{1 + e^{-(\vec{\omega}^{(i)} \cdot \vec{x} + \theta^{(i)})}} \quad (3.2)$$

and for the j th component of $\vec{\omega}^{(i)}$

$$\frac{\partial}{\partial \omega_j^{(i)}} \text{log-likelihood}(\phi_B) = \sum_{\vec{x} \in S} x_j \frac{1}{1 + e^{-(\vec{\omega}^{(i)} \cdot \vec{x} + \theta^{(i)})}} - N \sum_{\vec{x} \in \{\pm 1\}^n} Pr(\vec{x} | \phi_B) x_j \frac{1}{1 + e^{-(\vec{\omega}^{(i)} \cdot \vec{x} + \theta^{(i)})}} \quad (3.3)$$

The purpose of the clamped and unclamped phases (also called action and sleep phases) in the Boltzmann machine learning algorithm is to approximate these two expressions. The first term in each expression corresponds to the clamped phase, and the second one to the unclamped, or sleep phase. In general Boltzmann Machines, this estimation is performed using stochastic methods. However, here the clamped term is easy to calculate, it requires summing a logistic type function over all training examples. The same term is obtained by making the mean field approximation for the clamped phase in the general algorithm [PA87], which is exact in this case. It is more difficult to compute the sleep phase term, as it is an explicit sum over the entire input space, and within each term of this sum there is an implicit sum over the entire space of states of hidden units in the factor $Pr(\vec{x} | \phi_B)$. However, again taking advantage of the special structure of the combination model, we can reduce this sleep phase gradient term to a sum only over the states of the hidden units. Recall Equation (2.2):

$$Z_B = 2^n \sum_{\vec{h} \in \{0,1\}^m} \left[\exp\left(\sum_{i=1}^m h_i \theta^{(i)}\right) \prod_{j=1}^n \cosh(\vec{\omega}(\vec{h})_j) \right].$$

A similar derivation gives that

$$\Pr(\vec{h}|\phi_B) = \frac{\exp(\sum_{i=1}^m h_i \theta^{(i)}) \prod_{j=1}^n \cosh(\vec{\omega}(\vec{h})_j)}{\sum_{\vec{h}' \in \{0,1\}^m} \left[\exp(\sum_{i=1}^m h'_i \theta^{(i)}) \prod_{j=1}^n \cosh(\vec{\omega}(\vec{h}')_j) \right]} \quad (3.4)$$

The second term of the derivative w.r.t. $\theta^{(i)}$ is $\partial/\partial\theta^{(i)} \ln Z_B = (\partial/\partial\theta^{(i)} Z_B)/Z_B$. As $\theta^{(i)}$ appears only once in Z_B , we get that:

$$\frac{\partial}{\partial\theta^{(i)}} \log\text{-likelihood}(\phi_B) = \sum_{\vec{x} \in S} \frac{1}{1 + e^{-(\vec{\omega}^{(i)} \cdot \vec{x} + \theta^{(i)})}} - N \sum_{\vec{h} \in \{0,1\}^m} \Pr(\vec{h}|\phi_B) h_i. \quad (3.5)$$

Similarly, for each component of \vec{w}_i , we use the fact that $d \cosh(t)/dt = \tanh(t)$, to get that

$$\frac{\partial}{\partial\omega_j^{(i)}} \log\text{-likelihood}(\phi_B) = \sum_{\vec{x} \in S} x_j \frac{1}{1 + e^{-(\vec{\omega}^{(i)} \cdot \vec{x} + \theta^{(i)})}} - N \sum_{\vec{h} \in \{0,1\}^m} \Pr(\vec{h}|\phi_B) h_i \tanh(\vec{\omega}(\vec{h})_j) \quad (3.6)$$

The formulas for the gradients of the log likelihood for the real-valued model are very similar. A derivation similar to the one used to derive Equation (3.4), gives us that, for the real-valued model

$$\Pr(\vec{h}|\phi_R) = \frac{\exp(\sum_{i=1}^m h_i \theta^{(i)} + \frac{1}{2} \|\vec{\omega}(\vec{h})\|_2^2)}{\sum_{\vec{h}' \in \{0,1\}^m} \left[\exp(\sum_{i=1}^m h'_i \theta^{(i)} + \frac{1}{2} \|\vec{\omega}(\vec{h}')\|_2^2) \right]}. \quad (3.7)$$

Using this equation we get that

$$\frac{\partial}{\partial\theta^{(i)}} \log\text{-likelihood}(\phi_R) = \sum_{\vec{x} \in S} \frac{1}{1 + e^{-(\vec{\omega}^{(i)} \cdot \vec{x} + \theta^{(i)})}} - N \sum_{\vec{h} \in \{0,1\}^m} \Pr(\vec{h}|\phi_R) h_i \quad (3.8)$$

and for each j

$$\frac{\partial}{\partial\omega_j^{(i)}} \log\text{-likelihood}(\phi_R) = \sum_{\vec{x} \in S} x_j \frac{1}{1 + e^{-(\vec{\omega}^{(i)} \cdot \vec{x} + \theta^{(i)})}} - N \sum_{\vec{h} \in \{0,1\}^m} \Pr(\vec{h}|\phi_R) h_i \vec{\omega}(\vec{h})_j \quad (3.9)$$

Equations (3.5-3.4) are very similar to Equations (3.8-3.7). The differences are in the partial derivative of the normalization factors, Z_B and Z_R , with respect to the weight vectors. Note that the equations for the real-valued model are simpler. As was discussed in Section (2.6), the normalization factors for the real and binary models are very close to each other when the weight vectors \vec{w}_i have small l_2 norm. Thus although the equations for the maximal likelihood solutions differ, the solution of the real-valued model are approximate solutions for the binary model and vice versa.

The time required to compute Equations (3.5) and (3.6), (or Equations (3.8) and (3.9) is $O(|S|n + 2^m)$. Thus, if m is small compared with the size of the sample S , then the computation time is linear in the number of training example and in the size of the input vector, which is reasonable. However, for large m it might not be possible to compute all 2^m terms. There is a way to avoid this exponential explosion if we can assume that a small number of terms dominate the sums. If, for instance, we assume that the probability that more than k hidden units are active (+1) at the same time is negligibly small we can get a good approximation by computing only $O(m^k)$ terms. In the extreme case where we assume that only one hidden unit is active at a time (i.e. $k = 1$), the combination model essentially reduces to the standard mixture model as discussed in Section 2.3. For larger k , this type of assumption provides a middle ground between the generality of the combination model and the simplicity of the mixture model. In the next section we show how the gradient of the real-valued model can be approximated when m is large.

3.1 Approximating the gradient

One possible approach to estimating the gradient when m is large is to search for the larger terms in Equations (3.8,3.9) and ignore the smaller ones. We now show that in the case of the real-valued model the problem of locating the large terms is equivalent to a simple geometric problem. Although this problem is NP-hard in the general case it might typically be easy in the cases that we encounter in real life problems.

Recall Equation (3.7)

$$Pr(\vec{h}|\phi_R) = \frac{\exp(\sum_{i=1}^m h_i \theta^{(i)} + \frac{1}{2} \|\sum_{i=1}^m h_i \vec{w}i\|_2^2)}{\sum_{\vec{h}' \in \{0,1\}^m} \left[\exp(\sum_{i=1}^m h'_i \theta^{(i)} + \frac{1}{2} \|\sum_{i=1}^m h'_i \vec{w}i\|_2^2) \right]}$$

We would like to estimate which of the vectors \vec{h} correspond large terms. i.e. we would like to find all \vec{h} such that $g(\vec{h}) = \sum_{i=1}^m h_i \theta^{(i)} + \frac{1}{2} \|\sum_{i=1}^m h_i \vec{w}i\|_2^2$ is large. Define the following matrix notation. We use \vec{x} to denote a column vector and \vec{x}^T to denote its transpose, i.e. a row vector. We define

$$\Omega = \begin{pmatrix} (\vec{w}1)^T \\ (\vec{w}2)^T \\ \vdots \\ (\vec{w}m)^T \end{pmatrix}$$

$$\vec{\theta} = (\theta_1, \theta_2, \dots, \theta_m)^T$$

Using this notation we define $g(\vec{h})$ as

$$g(\vec{h}) = \vec{h} \cdot \vec{\theta} + \frac{1}{2} \|\vec{h}^T \Omega\|_2^2,$$

and rewrite Equation (3.7) as

$$Pr(\vec{h}|\phi_R) = \frac{\exp(g(\vec{h}))}{\sum_{\vec{h}' \in \{0,1\}^m} \exp(g(\vec{h}'))}.$$

Rearranging $g(\vec{h})$ we get

$$g(\vec{h}) = \frac{1}{2} \|\vec{h}^T \Omega + \vec{\theta}^T (\Omega \Omega^T)^{-1} \Omega\|_2^2 - \frac{1}{2} \vec{\theta}^T (\Omega \Omega^T)^{-1} \vec{\theta},$$

assuming that $\Omega \Omega^T$ is not singular.

The second term is a constant and is eliminated by the normalization. We can therefore ignore it. The first term corresponds to the distance between a sum of a subset of the weight vectors and the fixed vector $\vec{\theta}^T (\Omega \Omega^T)^{-1} \Omega$. The problem of finding the settings of \vec{h} for which $g(\vec{h})$ is largest translates to the problem of finding a subset of a given set of vectors which is furthest away (in the regular Euclidean distance) from a given fixed vector.

It is not clear how hard this computation problem is in the general case. If the vectors are orthogonal then the problem is easy. In this case the set of all 2^m vector combinations defines the corners of a rectangular box. If the dot product, $\vec{x} \cdot \vec{w}i$, is equal to $\|\vec{w}i\|_2/2$ for all i , then \vec{x} is in the center of the box. Any deviation from equality for a particular index i determines whether the vector corresponding vector,

$\vec{w}i$, is in the subset whose sum is furthest from \vec{x} . In general, one of the closest subset-sums is equal to $\vec{h}^T \Omega$, where each coordinate of \vec{h} is defined by:

$$h_i = \begin{cases} 1 & \text{if } \vec{w}i \cdot \vec{x} \leq \frac{1}{2} \|\vec{w}i\|_2 \\ 0 & \text{otherwise} \end{cases}$$

A promising direction for further research is to find methods that can solve this problem efficiently in the general case. Such methods would compute an approximation to the gradient by computing only the largest terms in the sum that defines it.

3.2 Projection Pursuit methods

A statistical method that has a close relationship with the combination model is the Projection Pursuit (PP) technique [Hub85, FWS84, Fri87]. In this section we give a short overview of the technique, show how it relates to the combination model, and present a learning algorithm for the combination model based on Projection Pursuit methods. This algorithm is a greedy algorithm that generates the hidden units one by one. It avoids the exponential blowup of the standard gradient ascent technique, and also has that advantage that the number m of hidden units is estimated from the sample as well, rather than being specified in advance.

3.3 Overview of Projection Pursuit

Many methods for analyzing high dimensional data study the first and second order statistics of the data, which are the mean vector and the covariance matrix. Principal components analysis is an example of such a method. Such methods necessarily ignore the structure of the distribution that is not reflected in the first and second order statistics, which may be an important part. Projection Pursuit methods can sometimes find this important high-order structure.

The distribution model over R^n with the largest entropy for a given average and covariance is a Gaussian distribution. Thus one natural definition of the information ignored by the second order analysis is the deviation of the empirical distribution from the Gaussian distribution. Low order linear projections have been traditionally used by researchers in their efforts to understand high dimensional distributions. As all projections of a Gaussian distribution produce a Normal marginal distribution. Thus, if a projection of a distribution generates a marginal distribution that is very different from the normal distribution, this is an indication that the projection contains information about the distribution that does not exist in its covariance matrix. Such a projection may be called an “interesting” projection. There are various “projection indices” defined in the PP literature to measure how interesting a particular projection is, and many of these indices relate directly to the deviation of the marginal distribution from a Normal distribution. Projection Pursuit methods locate the low dimensional projections in which the projection index is largest, i.e. those projections that are most interesting.

Originally, PP was used to suggest projection directions as an aid for the manual exploration of high dimensional data via two or three dimensional projections. Later PP became a complete method for

statistical data analysis, using repeated search for interesting projections to generate n -dimensional density estimations. The search for a description of the distribution of a sample in terms of its projections can be formalized in the context of maximal likelihood density estimation in the following way [Fri87]. Define $p_0(\vec{x})$ to be the initial estimate of the density over R^n , i.e. the Gaussian density with appropriate mean and covariance. Define G to be a family of functions from R to R and A to be the set of vectors of length 1, i.e. $A = \{\vec{\alpha} \in R^n \mid \|\vec{\alpha}\|_2 = 1\}$. Using these we define the n th order projection estimates to be the following set of densities

$$\mathcal{PP}_m = \left\{ \frac{1}{Z} p_0(\vec{x}) \prod_{i=1}^m g_i(\vec{\alpha}^{(i)} \cdot \vec{x}) \mid \vec{\alpha}^{(i)} \in A; g_i \in G; Z = \int_{R^n} p_0(\vec{x}) \prod_{i=1}^m g_i(\vec{\alpha}^{(i)} \cdot \vec{x}) d\vec{x} \right\} \quad (3.10)$$

The log-likelihood of a specific density $p \in \mathcal{PP}_m$ with respect to a sample $S = \{\vec{x}^{(1)}, \vec{x}^{(2)}, \dots, \vec{x}^{(N)}\}$, where $\vec{x}^{(i)} \in R^n$ is defined, in the standard way, to be

$$LL(p|S) = \sum_{\vec{x} \in S} \ln p(\vec{x}) .$$

The goal of Projection Pursuit is to find a series of approximations: $p_1 \in \mathcal{PP}_1, p_2 \in \mathcal{PP}_2, \dots, p_m \in \mathcal{PP}_m$ that have maximally increasing log-likelihood. The first approximation, p_0 , is the Gaussian density itself, and the $(i+1)$ -st approximation is generated by adding a factor $g_i(\vec{\alpha}^{(i)} \cdot \vec{x})$ to the i th approximation. The vector $\vec{\alpha}^{(i)}$ is called the i th projection of the data.

The projection index is a function of $\vec{\alpha}^{(i)}$ that is a heuristic measure of the anticipated contribution of a factor involving the projection $\vec{\alpha}^{(i)}$ to the likelihood of the model. Given a choice of $\vec{\alpha}^{(i)}$, the *optimal* choice of the function $g_i(\cdot)$ in terms of maximizing the likelihood is the following [FWS84]. Define $p_i^{\vec{\alpha}^{(i)}}(t)$ to be the marginal density on R generated by projecting the density p_i on the direction $\vec{\alpha}^{(i)}$. Similarly define $\hat{p}^{\vec{\alpha}^{(i)}}(t)$ to be an approximation to the marginal density generated by projecting the true density on the direction $\vec{\alpha}^{(i)}$, estimated empirically using the sample S .¹ Then the optimal choice for $g_i(\cdot)$, in terms of maximizing the likelihood of the model, is

$$g_i(t) = \frac{\hat{p}^{\vec{\alpha}^{(i)}}(t)}{p_i^{\vec{\alpha}^{(i-1)}}(t)} \quad (3.11)$$

As the optimal choice of $g_i(\cdot)$, for a given choice of $\vec{\alpha}^{(i)}$ is simple to calculate. The main problem of designing a projection pursuit method is finding a good projection index whose calculation can be performed efficiently. Various projection indices have been discussed in the literature [Hub85, Fri87]. Selection of a direction that has a high projection index is usually performed using gradient following methods. After a local maximum of the projection index has been found, the index function is altered to prevent the search from finding the same direction again, and a search for a direction with a high projection index is started from a different starting point.

¹Note that the marginal density is a one dimensional function, thus the number of samples needed for estimating it is relatively small. In this way projection pursuit avoids, to some degree, the infamous ‘‘curse of dimensionality’’ in the estimation of the distribution of high dimensional data.

The search for new projection directions can be simplified if instead of altering the projection index function, the sample is altered in a way that previously found interesting projections $(\vec{\alpha}^{(1)}, \vec{\alpha}^{(2)}, \dots, \vec{\alpha}^{(i-1)})$ are made to appear uninteresting, i.e. Normally distributed. So called “structure removal” methods have been devised towards this goal [Hub85, Fri87]. These methods alter the sample in such a way that a specific single projection that has been interesting is made uninteresting while all orthogonal projections are left unchanged. Put in another way, suppose that some density $p \in \mathcal{PP}_m$ has high likelihood with respect to a given sample, and that one of the factors in p is $g_1(\vec{\alpha}^{(1)} \cdot \vec{x})$. Then removing the structure corresponding to $g_1(\vec{\alpha}^{(1)} \cdot \vec{x})$ means transforming the sample into a sample for which $p(\vec{x})/g_1(\vec{\alpha}^{(1)} \cdot \vec{x})$, which is a model in \mathcal{PP}_{m-1} , has high likelihood.

To summarize, most iterative projection pursuit methods share the following common structure:

- **Initialization**

Set S_0 to be the input sample.

Set p_0 to be the initial density (Gaussian).

- **Iteration**

Repeat the following steps for $i = 1, 2 \dots$ until all projections of S_i are almost Normal.

1. Find a direction $\vec{\alpha}^{(i)}$ for which the projection index of the projection of S_{i-1} is maximized.
2. Approximate the actual marginal density in the direction $\vec{\alpha}^{(i)}$ by finding a close fit to the density of the projection of the sample S_{i-1} . Set $g_i(\cdot)$ to be the ratio between this approximation and the marginal density produced on $\vec{\alpha}^{(i)}$ by p_{i-1} , using Equation (3.11).
3. Set S_i to be S_{i-1} with the structure defined by the factor $g_i(\vec{\alpha}^{(i)} \cdot \vec{x})$ removed. This makes the projection of S_i on $\vec{\alpha}^{(i)}$ uninteresting, and all of the orthogonal projections remain equal to that of S_{i-1} .
4. Set $p_i(\vec{x})$ to be $p_{i-1}(\vec{x})g_i(\vec{\alpha}^{(i)} \cdot \vec{x})$.

Notice that in this method the functions g_i are chosen in such a way that the product $\prod_{i=1}^m g_i(\vec{\alpha}^{(i)} \cdot \vec{x})$ is normalized for each m and there is no need for an additional normalization term Z , as appears in the definition of \mathcal{PP}_m in Equation (3.10).

Projection Pursuit has proved itself successful in some experiments [Fri87]. However, the search for best density is performed in a greedy manner and might not succeed in finding the optimal density in \mathcal{PP}_m . While there is quite a large body of research on the representational power of projection pursuit models, little is theoretically known about reliability of the associated learning algorithms, such as the one presented above.

3.4 Projection Pursuit and the combination model

Recall Equation (2.6), which describes the density generated by the real-valued combination model:

$$p(\vec{x}) = e^{-\frac{1}{2}\|\vec{x}\|_2^2} \frac{1}{Z_R} \prod_{i=1}^m \left(1 + e^{\theta_i + \vec{w}^i \cdot \vec{x}}\right).$$

Using the following definitions we see that this class of models is a special case of the class of models presented in Equation (3.10).

$$p_0(\vec{x}) = (2\pi)^{-n/2} e^{-\frac{1}{2}\|\vec{x}\|_2^2} = \mathcal{N}(0, 1)$$

$$\vec{\alpha}^{(i)} = \frac{\vec{w}i}{\|\vec{w}i\|_2}$$

$$G = \left\{ g : R \rightarrow R \mid g(t) = \frac{1}{Z} \left(1 + e^{\theta_i + t\|\vec{w}i\|_2} \right) ; Z \in R \right\}$$

It is clear that, under these definitions, $p(\vec{x})$ is a function in \mathcal{PP}_m . In the next section we present a greedy algorithm for learning the combination model that is based on this relation.

A similar relationship holds for the binary model. However, we have not managed to find a good structure removal procedure for the binary-valued model. We thus present an algorithm for learning the real-valued model and, based on the relations given in Section 2.6, we claim that the solutions that we find for the real-valued model are approximate solutions for the binary-valued model.

There are two main differences between our work and previous work on using exploratory projection pursuit algorithms for estimating distributions. The first difference is that while our model defines a distribution on all R^n , our data-points are taken from $\{-1, +1\}^n$. However, as discussed at the end of Section 2.6, the projections of the binary vectors generate marginal distributions that are close to Normal, similarly to the distributions we expect from real-valued data.

The second difference is that the family of functions G from which the g_i s are taken is a very restricted set of functions. This is unlike standard PP techniques, in which the functions g_i are chosen from some very broad family, such as some family of spline functions. This means that, in our case, any single function $g \in G$ might be far from adequate for describing the marginal distribution on some direction $\vec{\alpha}^{(i)}$ and several factors with the same $\vec{\alpha}$ might be needed. This, in turn, has the effect that eliminating the structure generated by a single factor does not amount to transforming the marginal distribution on the corresponding projection so that it becomes completely uninteresting. As most structure elimination techniques do exactly that, they are unfit in the context of learning the combination model.

3.5 PP algorithm for learning the combination model

In this section we present a variant of PP that is a learning algorithm for the combination model. Our algorithm combines the search for an interesting projection direction, $\vec{\alpha}$, with the search for the corresponding projection function, $g(\cdot)$. The algorithm searches for the optimal factor by maximizing the likelihood of a single factor model with respect to the (possibly altered) sample. After such a factor is found, the algorithm alters the examples in such a way that the structure encoded in the factor is eliminated, and subsequent searches will find different factors.

The algorithm is thus based on two elements. The first element is a method for finding a maximal likelihood combination model with a single hidden unit. This method serves both for finding a projection direction, and for finding the function $g_i(\cdot)$ associated with this direction. The second element is a structure removal procedure. We shall describe the two elements in turn.

We have previously described how gradient ascent can be used for finding model with highest log-likelihood. However, for the special case where there is only a single hidden unit in the model, a much faster method can be used. This method is an Expectation-Maximization (EM) method [DLR77]. EM is a general method for estimating the parameters of distribution models that have both observable and unobservable random variables. This method achieves extremely fast convergence when used for estimating a mixture of product distributions.²

The Expectation Maximization method is based on iterative improvement of the estimates of the maximal likelihood values of the model parameters. It starts with some initial guess of the parameters ϕ_{init} , and proceeds by iterating the following two steps. It can be shown [DLR77], that each of these iterations improves the likelihood of the parameters.

1. Using the old setting of the parameters, ϕ_{old} , as if they were the actual parameters, some statistics of the joint distribution of the hidden and the observable variables are calculated.
2. The old setting of the parameters, ϕ_{old} , is replaced with a new setting of the parameters ϕ_{new} , which is the most likely setting of the parameters given the values of the statistics calculated in step 1. These new parameters are used as the old parameters in the following iteration.

To see how this method is implemented for the problem of estimating the parameters of a real-valued combination model with a single hidden unit let us calculate the maximal likelihood setting of the parameters assuming that we are given a sample S' , of size N , in which each element describes the value of both the observable random variables, \vec{x} , and the unobservable random variable h . The log likelihood is

$$\begin{aligned} LL(\theta, \vec{\omega} | S') &= \sum_{(h, \vec{x}) \in S'} \ln P(\vec{x}, h | \theta, \vec{\omega}) = \sum_{(h, \vec{x}) \in S'} \ln \frac{\exp(h(\theta + \vec{\omega} \cdot \vec{x}))}{Z_R} \\ &= \sum_{(h, \vec{x}) \in S'} h(\theta + \vec{\omega} \cdot \vec{x}) - N(2\pi)^{n/2} \ln \left(1 + \exp \left(\theta + \frac{1}{2} \|\vec{\omega}\|_2^2 \right) \right) \end{aligned}$$

Taking the derivative of the log-likelihood with respect to the parameters and equating to zero to find the optimal setting of the parameters, we get the following equations. From the derivative w.r.t. θ we get that

$$\sum_{(h, \vec{x}) \in S} h = N \text{logistic} \left(\theta_{\text{opt}} + \frac{1}{2} \|\vec{\omega}_{\text{opt}}\|_2^2 \right), \quad (3.12)$$

and from the gradient w.r.t. $\vec{\omega}$ we get that

$$\sum_{(h, \vec{x}) \in S} h \vec{x} = \vec{\omega}_{\text{opt}} N \text{logistic} \left(\theta_{\text{opt}} + \frac{1}{2} \|\vec{\omega}_{\text{opt}}\|_2^2 \right) \quad (3.13)$$

Notice that if we divide the sums on the left hand side of Equations (3.12) and (3.13) by N , we get the definition of the empirical estimates of $E(h)$ and of $E(h\vec{x})$, which we shall denote by $\hat{E}(h)$ and $\hat{E}(h\vec{x})$. Solving Equations (3.12) and (3.13) for the values of the optimal parameters, we get that:

$$\vec{\omega}_{\text{opt}} = \frac{\hat{E}(h\vec{x})}{\hat{E}(h)}, \quad (3.14)$$

²It is not easy to implement EM directly on the complete combination model, because although this distribution can be expressed as a mixture of product distributions, the parameters that define the mixture components are coupled.

and

$$\theta_{\text{opt}} = -\ln \frac{1 - \hat{E}(h)}{\hat{E}(h)} - \frac{1}{2} \|\vec{\omega}_{\text{opt}}\|_2^2. \quad (3.15)$$

We thus see that the statistics that we need to estimate in the first step of the EM iteration are $\hat{E}(h)$ and $\hat{E}(h\vec{x})$. These statistics can be directly calculated from the sample S' , as this sample includes both \vec{x} and h . However, given a setting of the parameters, we can compute the distribution of h for any setting of \vec{x} , and thus calculate the desired statistics.

The implementation of the EM method for the combination model with a single hidden unit is thus as follows. We start with an initial setting of the parameters: $(\vec{\omega}_{\text{init}}, \theta_{\text{init}})$ and proceed by iterating the following two steps on the given sample $S = \langle \vec{x}_1, \vec{x}_2, \dots, \vec{x}_N \rangle$

1. In the Expectation calculation step the current parameters $(\vec{\omega}_{\text{old}}, \theta_{\text{old}})$ are used as if they describe the correct input distribution. Given this description and a particular setting of the input units, \vec{x} , we can compute probability that each hidden unit is 0 or 1 given any setting of the observable vector \vec{x} :

$$\Pr(h_i = 1 | \vec{x}, \vec{\omega}_{\text{old}}, \theta_{\text{old}}) = \text{logistic}(\vec{\omega}_{\text{old}} \cdot \vec{x} + \theta_{\text{old}}) \vec{x}.$$

Using this equation and the sample S , it is possible to compute the following estimates:

$$\hat{E}(h\vec{x}) = \frac{1}{N} \sum_{\vec{x} \in S} \text{logistic}(\vec{\omega}_{\text{old}} \cdot \vec{x} + \theta_{\text{old}}) \vec{x}$$

$$\hat{E}(h = 1) = \frac{1}{N} \sum_{\vec{x} \in S} \text{logistic}(\vec{\omega}_{\text{old}} \cdot \vec{x} + \theta_{\text{old}})$$

2. In the Maximization step, new parameters $(\vec{\omega}_{\text{new}}, \theta_{\text{new}})$ are calculated using Equations (3.14) and (3.15). The new parameters $(\vec{\omega}_{\text{new}}, \theta_{\text{new}})$ are used as the old parameters $(\vec{\omega}_{\text{old}}, \theta_{\text{old}})$ in the following iteration.
3. The iteration terminates when the difference between $(\vec{\omega}_{\text{new}}, \theta_{\text{new}})$ and $(\vec{\omega}_{\text{old}}, \theta_{\text{old}})$ becomes insignificant.

We now present the structure removal procedure. In the analysis of the real-valued model in Section (2.3) we have shown that the addition of a hidden variable has the effect of replacing the previous distribution by a mixture of two distributions, the first of which is equivalent to the previous, and the second is a shifted copy of the previous distribution, shifted by the weight vector \vec{w}_i that corresponds to the hidden unit. The shifted copy corresponds to the case in which $h_i = 1$ while the unshifted one corresponds to the case where $h_i = 0$. For each data point we compute the probability, p , that $h_i = 1$. We then flip a random coin whose bias is p and, according to the outcome of the coin flip, either keep the example as it is or subtract \vec{w}_i from it. This has the effect of shifting the shifted copy, which corresponds to $h_i = 1$ to coincide with the unshifted copy, which corresponds to $h_i = 0$. In this way the structure encoded by the

hidden unit is eliminated from the empirical distribution. Details are described below.

- **Initialization**

Set S_0 to be the input sample.

Set p_0 to be the initial distribution (Gaussian).

- **Iteration**

Repeat the following steps for $i = 1, 2 \dots$ until no single-variable combination model has a significantly higher likelihood than the Gaussian distribution with respect to S_i .

1. Perform an EM procedure to maximize the log-likelihood of a single hidden variable model on the sample S_{i-1} . Denote by θ_i and \vec{w}_i the parameters found by this procedure, and create a new hidden unit with associated binary random variable h_i with these weights and bias.
2. Transform S_{i-1} into S_i using the following structure removal procedure.
For each example $\vec{x} \in S_{i-1}$ compute the probability that the hidden variable h_i found in the last step is 1 on this input:

$$P(h_i = 1) = \left(1 + e^{-(\theta_i + \vec{w}_i \cdot \vec{x})}\right)^{-1}$$

Flip a coin that has probability of “head” equal to $P(h_i = 1)$. If the coin turns out “head” then add $\vec{x} - \vec{w}_i$ to S_i else add \vec{x} to S_i .

3. Set $p_i(\vec{x})$ to be $p_{i-1}(\vec{x})Z_i^{-1} \left(1 + e^{\theta_i + \vec{w}_i \cdot \vec{x}}\right)$, where $Z_i = \sum_{\vec{x}} p_{i-1}(\vec{x}) \left(1 + e^{\theta_i + \vec{w}_i \cdot \vec{x}}\right)$.

4. Experimental work

We have carried out several experiments to test the performance of unsupervised learning using the combination model. The goals of these experiments is to show that the combination model is a useful one and to compare the performance of the different learning algorithm that we have developed.

The first set of experiments compares the two learning methods for the combination model presented in this paper. The first is the gradient ascent method, and the second is the projection pursuit method. The experiments in this set were performed on synthetically generated data. The input consisted of 4,000 binary vectors of 64 bits that represent 8×8 binary images. The binary vectors are synthesized using a combination model with 10 hidden units whose weights were set as in Figure (4.1,a). Each square in this image denotes a single real valued parameter,¹ the matrix corresponds to the weight vector, and the rectangle above the matrix corresponds to the bias parameter θ . We shall refer to each random binary vector as an instance.

The ultimate goal of the learning algorithms was to retrieve the model that generated the instances, which we call the “target” model. However, this goal is generally not achievable. The first reason is that the optimal model is not unique, i.e. there usually are other combination models that generate the exact same distribution as the target model, or a distribution that is very close to it. For example, a permutation of the hidden units does not change the distribution defined by the model. As we have found out in the experiments, other simple transformations of the target model produce models that are almost as good as the target model. Another reason that we cannot retrieve the exact target is that the parameter vector of the target is real valued, and thus cannot be exactly identified by a finite number of instances. The third reason is that our algorithms are not guaranteed to find the optimal model for the given data. The gradient ascent algorithm is only guaranteed to locate a local maximum of the likelihood, and the Projection Pursuit algorithm is only guaranteed to increase the likelihood of the model with each additional hidden unit.

While the difference between the parameter vectors of the learned model and of the target model is usually large, their performance as models of the random instances is similar. We measure this performance using three different error measures. Each error measure defines a way of computing the error of a combination model with respect to a set of instances. We have measured these errors for the target model and for each of the learned models. Each measurement was taken both with respect to the instances that were used for learning (the “training” instances) and with respect to an independent test set of 4000 instances.

We now describe each of the three measures of error that we have used:

- *Average log-loss*

Each learned distribution model defines a probability distribution, P , on the space of images. A popular measure of the distance between P and the actual distribution Q is the cross entropy, which is defined as $-\sum_x(Q(x) \log P(x))$. The cross entropy is minimized when $P = Q$, and is then equal to the entropy of Q . The cross entropy can be estimated by taking the average value of minus log

¹The results are given using Hinton diagrams [RM86], i.e. positive values are displayed as full rectangles, negative values as empty rectangles, and the area of the rectangle is proportional to the absolute value.

of the probability that the model assigns to each instance in the sample. This measure of error is also called the log-loss error. We scale the error so that the uniform distribution model, that assigns equal probability to all instances, has an expected error of 1. The log-loss error is hard to compute for large combination models, which is why we use it only in the experiments on synthetic data in which we use only 10 hidden units in the models.

- *Single bit completion*

We estimate the average number of mistakes made by the model when it is used to predict the value of single bits of the instances. More precisely, the mistakes it makes when used to predict the value of each single bit in each of the instances in the sample, when given the values of all the other bits of that instance. The combination model defines a probability for any possible instance. The prediction is defined as the value of the bit that corresponds to the more probable instance. We estimate this average number by choosing at random 5 bit locations for each instance in the sample.

- *Input reconstruction*

We estimate the quality of the combination model as an input representation scheme. For each instance (x_1, \dots, x_n) we compute the most probable state of the hidden units. This state can be seen as an encoding of the instance. One way of defining the quality of this encoding scheme is to measure how much additional information is required to reconstruct the instance from the state of the hidden units alone. Each state of the hidden units defined a Bernoulli product distribution over the images. The additional information that is required to encode a particular instance is the log of one over the probability assigned to the instance. As the distribution is a Bernoulli product, this can be written as the following sum:

$$H(\vec{x}|\vec{h}) = \frac{1}{2} \sum_{i=1}^n [(1 + x_i) \log_2 p_i + (1 - x_i) \log_2(1 - p_i)] ,$$

where p_i is the independent probability of the i th input bit to be +1 given the hidden state, which is equal to

$$p_i = \text{logistic} \left(\sum_{j=1}^m \omega_i^{(j)} h_j \right)$$

This measure of error is scaled so that it measures the additional information that is required per input bit.

All experiments used a test set and a separate training set, each containing 4000 examples. The gradient ascent method is based on the binary distribution model. It typically needed about 1000 epochs to stabilize.² In the projection pursuit algorithm, 4 iterations of EM per hidden unit proved sufficient to find a stable solution. The results are summarized in the following table and in Figure (4.1).³

²The algorithm used a standard momentum term (see [HKP91], page 123) to accelerate the convergence.

³The difference between the measurements of the quality of the true model on the test set and on the training set are due to the random fluctuations between the two sets of examples. These differences provide an indication of the accuracy of our measurements.

	log-loss		single bit prediction		input reconstruction	
	train	test	train	test	train	test
gradient ascent for 1000 epochs	0.399	0.425	0.098	0.100	0.311	0.338
projection pursuit	0.893	0.993	0.119	0.114	0.475	0.480
Projection pursuit followed by gradient ascent for 100 epochs	0.411	0.430	0.091	0.089	0.315	0.334
Projection pursuit followed by gradient ascent for 1000 epochs	0.377	0.405	0.071	0.082	0.261	0.287
true model	0.401	0.396	0.077	0.071	0.286	0.283

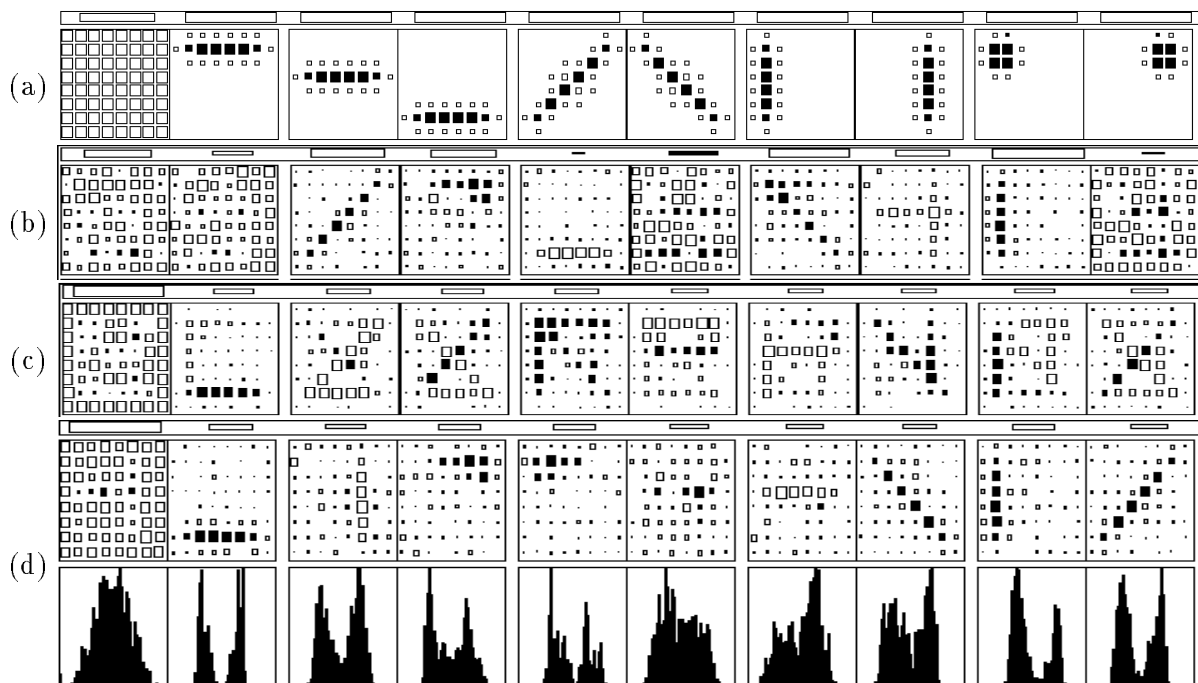


Figure 4.1: The weight vectors of the models in the synthetic data experiments. Each matrix represents the 64 weights of one hidden unit. The range of the weights is $[-6, +6]$ with the large white squares representing the value 6. The square above the matrix represents the units bias. positive weights are displayed as full squares and negative weights as empty squares, the area of the square is proportional to the absolute value of the weight. (a) The weights in the model used for generating the data. (b) The weights in the model found by gradient ascent alone. (c) The weights in the model found by projection pursuit alone. (d) The weights in the model found by projection pursuit followed by gradient ascent. For this last model we also show the histograms of the projection of the examples on the directions defined by those weight vectors; the bimodality expected from projection pursuit analysis is evident.

The best learning result was achieved by starting with the projection pursuit algorithm then using the parameter vector that was learned as a starting point for the gradient ascent algorithm. The final result of this combination is presented in Figure 4.1(d), together with the corresponding projections of the data

along the directions defined by the weight vectors. We can see that there is a close correspondence between the weight vectors in the learned model and the vectors in the target model described in Figure 4.1(a). Counting from left to right, the weight vectors of units 1,2,8,9, and 10 in the learned model are almost identical to the weight vectors of units 1,4,6,7, and 5 in the target model. Units 3 and 7 in the learned model are close to the negation of units 8 and 3 in the target model, and units 4 and 5 in the learned model are combinations of units (10,2) and (9,2) of the target model respectively. There is no exact correspondence of the biases. As we see from the table, the performance of the learned model is almost as good as that of the target model according to all three measures. We thus conclude that reversing the sign of weight vectors and combining them can sometimes create a different combination model whose corresponding distribution is very similar.

When the gradient ascent model is used to learn by itself (Figure 4.1(b)), it tends to get stuck in local minima, as can be seen in the table. It is also a very slow method, both because of the large number of iterations that is required and because each iteration requires complex calculations. The fact that the local search process is stuck in a sub-optimal solution can be seen in the weight vectors of the learned model in that four of the weight vectors (those of units 1,2,6,10, counting from the left) have no clear correspondence to any of the weight vectors in the target model.

The Projection Pursuit method is very fast, but its results are weaker than those of the gradient ascent method by itself. It tends to find a model whose weight vectors correspond to various combinations of the weight vectors of the target model and their negations. The performance of the results of projection pursuit are similar to those of the gradient method in the single bit prediction measure and in the input reconstruction measure. On the other hand, the performance of the Projection pursuit model in terms of the likelihood of the model that it generates is very poor. The reason is that the data that we use is generated by a binary valued combination model, while the projection pursuit model is based on a real valued combination model. The difference between these two models is large, because the weights that are used in the target model are in the range $[-6, +6]$. As we have shown in Section 2.6, the binary model and the real valued model are approximately equal when the weights are small. To show that this is indeed the source of the error, we repeated the previous experiments using a target model with the weight vectors divided by a factor of 7, so that now all the weights are in the range $[-6/7, +6/7]$. The results are summarized in the following table

	log-loss		single bit prediction		input reconstruction	
	train	test	train	test	train	test
True Model	0.939	0.941	0.36	0.36	0.86	0.87
gradient ascent for 400 epochs	0.937	0.944	0.36	0.37	0.86	0.87
projection pursuit	0.964	0.966	0.38	0.39	0.92	0.92
Projection pursuit followed by gradient ascent for 400 epochs	0.935	0.943	0.36	0.37	0.86	0.87

We see that in this case, the likelihood of the model found by the projection pursuit algorithm is similar

to that of the other models. Because in this case the weights are so small, the difference between the distribution defined by the model and the uniform distribution is small, as is reflected in the measures of accuracy. However, the difference from the uniform distribution is statistically significant. The combination of the two learning algorithms was able to retrieve the weights of the target model almost as well as in the previous experiment (see Figure 4.2).

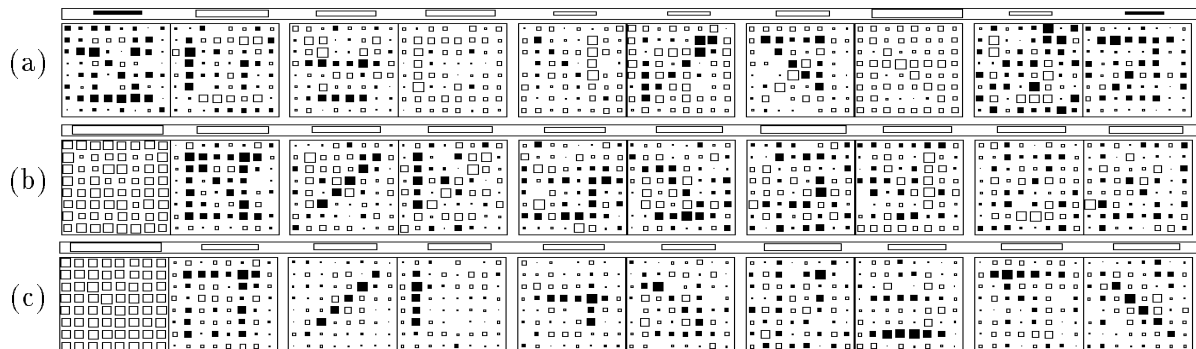


Figure 4.2: The weight vectors of the models in the synthetic data experiments. The target target is the same as in the previous experiment but the range of the weights is divided by a factor of 7, so that the largest white squares represent the value of $6/7$. (a) The weights in the model found by gradient ascent alone. (b) The weights in the model found by projection pursuit alone. (c) The weights in the model found by projection pursuit followed by gradient ascent.

In the second set of experiments we compare the performance of the combination model to that of the mixture model. The comparison uses real world data extracted from the NIST handwritten data base.⁴ Examples are 16×16 binary images (see Figure (4.3)). There are 500 examples in the training set and 500 in the test set. We use 45 hidden units to model the distribution in both of the models. Because of the large number of hidden units we cannot use gradient ascent learning and instead use projection pursuit. For the same reason it was not possible to compute the likelihood of the combination model and only the other two measures of error were used. Each test was run several times to estimate the accuracy of our measurements.

For learning a mixture model we use an incremental version of EM. We start with a model with a single Bernoulli product distribution and run EM until the method converges. We then take a mixture of two Bernoulli product distributions, each of which is initialized to be a slight random perturbation of the single Bernoulli product. We then let EM run on this model until it converges, and then we split each component into two in a similar way. Continuing in this fashion we repeatedly double the size of the model.⁵

The final errors of many runs of these algorithms, starting from different initial weights, are summarized in the table below. The errors of two representative runs are given in Figures 4.6 and 4.7. A sample of the final weight vectors of the learned combination model and mixture model are given in Figures 4.4 and 4.5

⁴NIST Special Database 1, HWDB Rel1-1.1, May 1990.

⁵When 32 units are to be split, only the first 13 of them are split, to give the final number of 45 mixture components.

respectively. A complete list of all of the 45 weight vectors for each model are given in Figures 4.8 and 4.9.

	single bit prediction		input reconstruction	
	train	test	train	test
Product distribution	0.29 ± 0.01	0.30 ± 0.01	0.78 ± 0.01	0.80 ± 0.01
Mixture model	0.19 ± 0.01	0.26 ± 0.01	0.55 ± 0.01	0.70 ± 0.01
combination model	0.19 ± 0.01	0.20 ± 0.01	0.60 ± 0.01	0.64 ± 0.01

The first line in this table, named “Product distribution” summarizes the performance of a simple distribution model that assumes that the pixels are distributed according to a Bernoulli product distribution. The reconstruction of the input, in this case, is simply the fixed reconstruction in which each bit is set to its more probable value. The performance of this model provides a baseline with respect to which we can compare the performance of the other distribution models whose goal is to capture dependencies between the pixels. We see that the performance of the combination model is significantly better than that of the mixture model on the test set. The difference is especially significant when compared to the baseline of the Product distribution model. Also, we see that the difference between the performance on the test set and on the training set, i.e. the over-fitting, is much smaller for the combination model.

A qualitative comparison between the weight vectors found by the two models confirms the expected advantage of the combination model in describing combinations of correlations. While the typical weight vectors of the mixture model (see Figure (4.5)), which is a sample out of Figure (4.8)) look very much like an average prototype of a specific digit, the weight vectors of the combination model relate to more local features, such as lines and curves (see Figure (4.4)), which is a sample out of Figure (4.9)). This relates to fact that the mixture model relates each example with the single weight vector that is most similar to it, while the combination model relates each example with a combination of its weights.

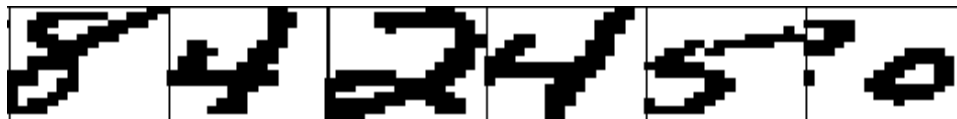


Figure 4.3: A few examples from the handwritten digits sample.

As the experiments on synthetic data have shown that PP does not reach optimal solutions by itself we expect the advantage of the combination model over the mixture model to increase further by using improved learning methods. Of course, the combination model is a very general distribution model and is not specifically tuned to the domain of handwritten digit images, thus it cannot be compared to models specifically developed to capture structures in this domain. However, the experimental results support our claim that the combination model is a simple and tractable mathematical model for describing distributions in which several correlation patterns combine to generate each instance.

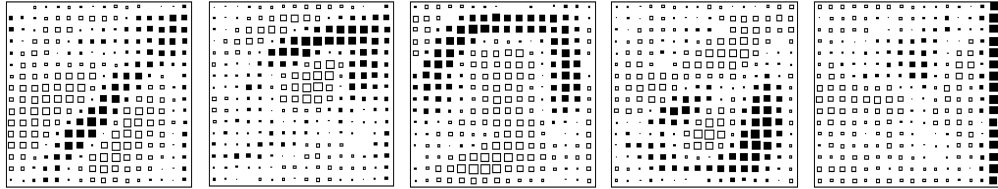


Figure 4.4: Typical weight vectors found by the combination model

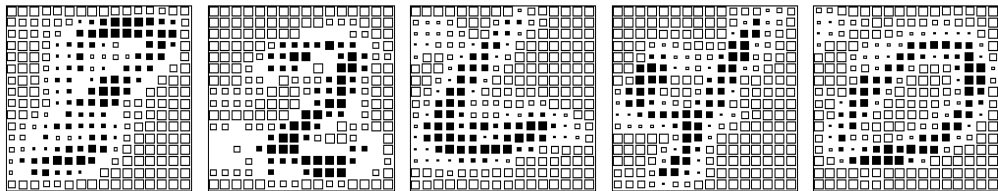


Figure 4.5: Typical weight vectors found by the mixture model

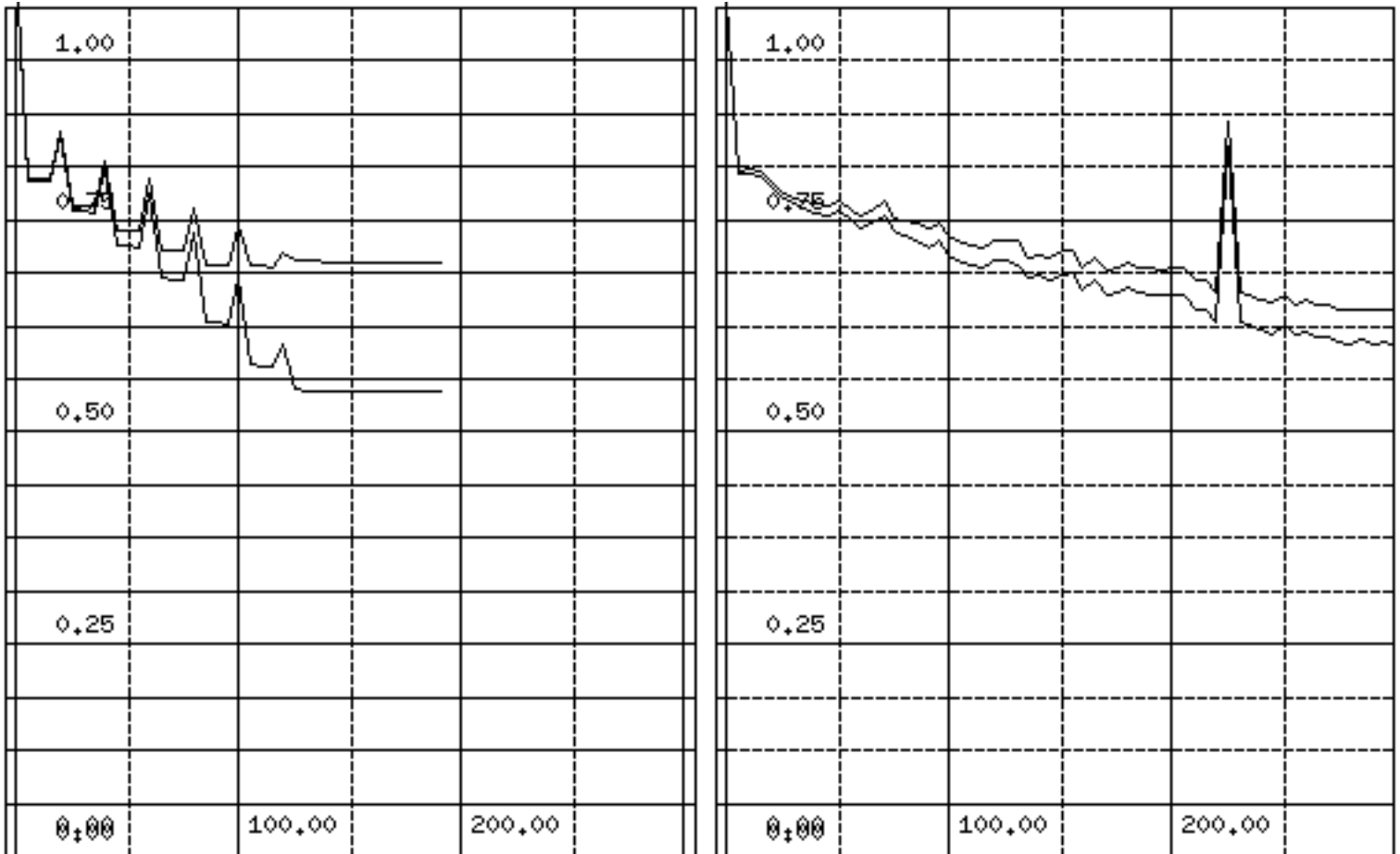


Figure 4.6: A comparison of the input reconstruction error on 16×16 pixel digit images. This error measures the average amount of additional information that is required for reconstructing the input from the state of the hidden units. The information is measured in bits per pixel. The higher and lower curves in each graph describe the error on the test set and on the training set respectively. The graph on the left describes the error of the mixture model as a function of the number of training iterations (epochs). The number of mixture components is doubled every 20 iterations. There is a spike in the error immediately following the doubling, as a result of the added randomization. The graph on the right describes the error of the combination model as a function of the number of iterations. (The spike in the graph around iteration 230 is a side effect of a “backfitting” stage that has not proven to be useful.)

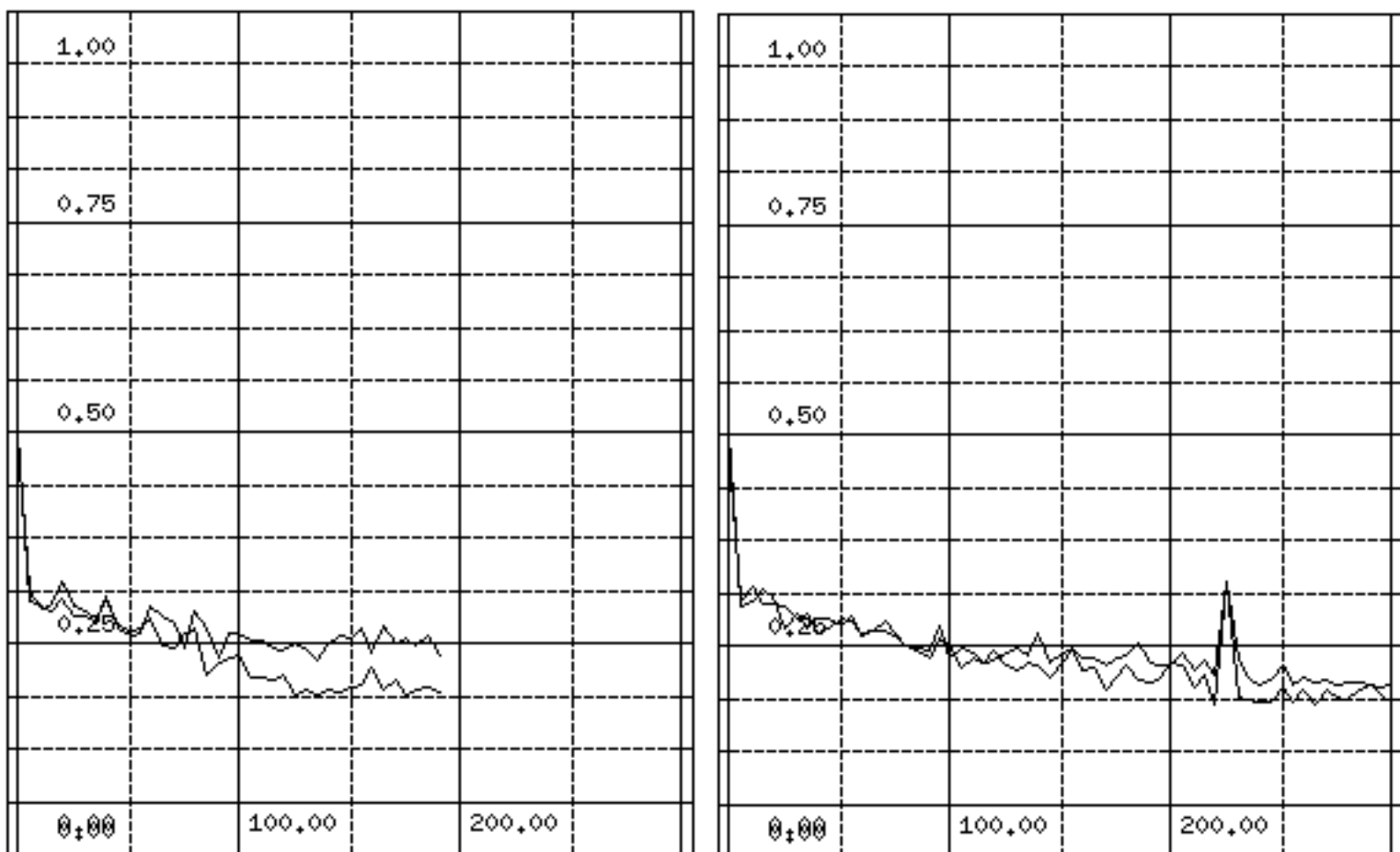


Figure 4.7: A comparison of the single bit completion error on 16×16 pixel digit images. The error measures the probability of a mistake in predicting a random single missing bit in the image, using the distribution model and the values of all the rest of the pixels. The higher and lower curves in each graph describe the error on the test set and on the training set respectively. The graph on the left describes the error of the mixture model as a function of the number of training iterations (epochs). The number of mixture components is doubled every 20 iterations. The graph on the right describes the error of the combination model as a function of the number of iterations. (The peak in the graph around iteration 230 is a side effect of a “backfitting” stage that has not proven to be useful.)

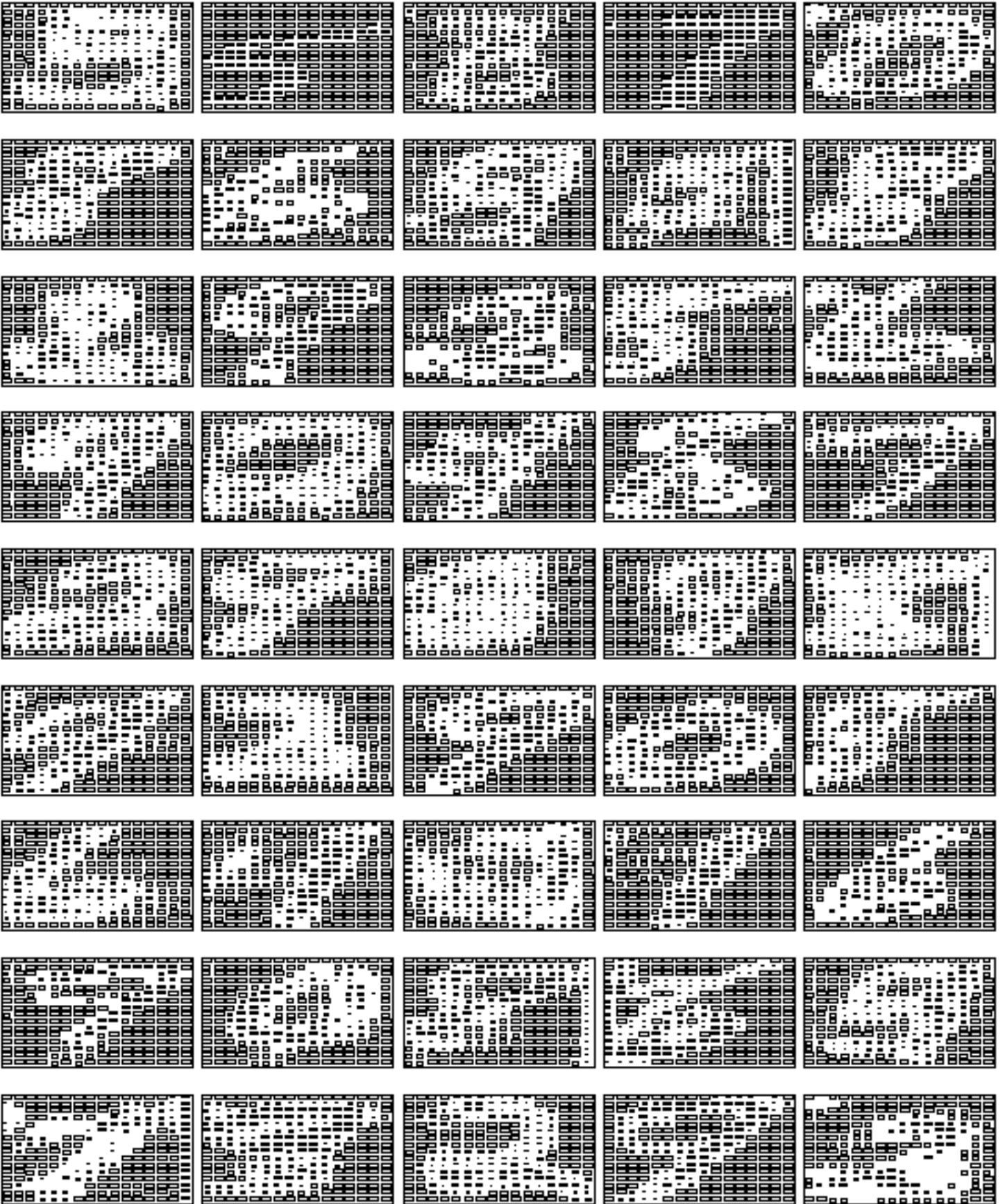


Figure 4.8: The weight vector, or image templates, found by the the mixture model

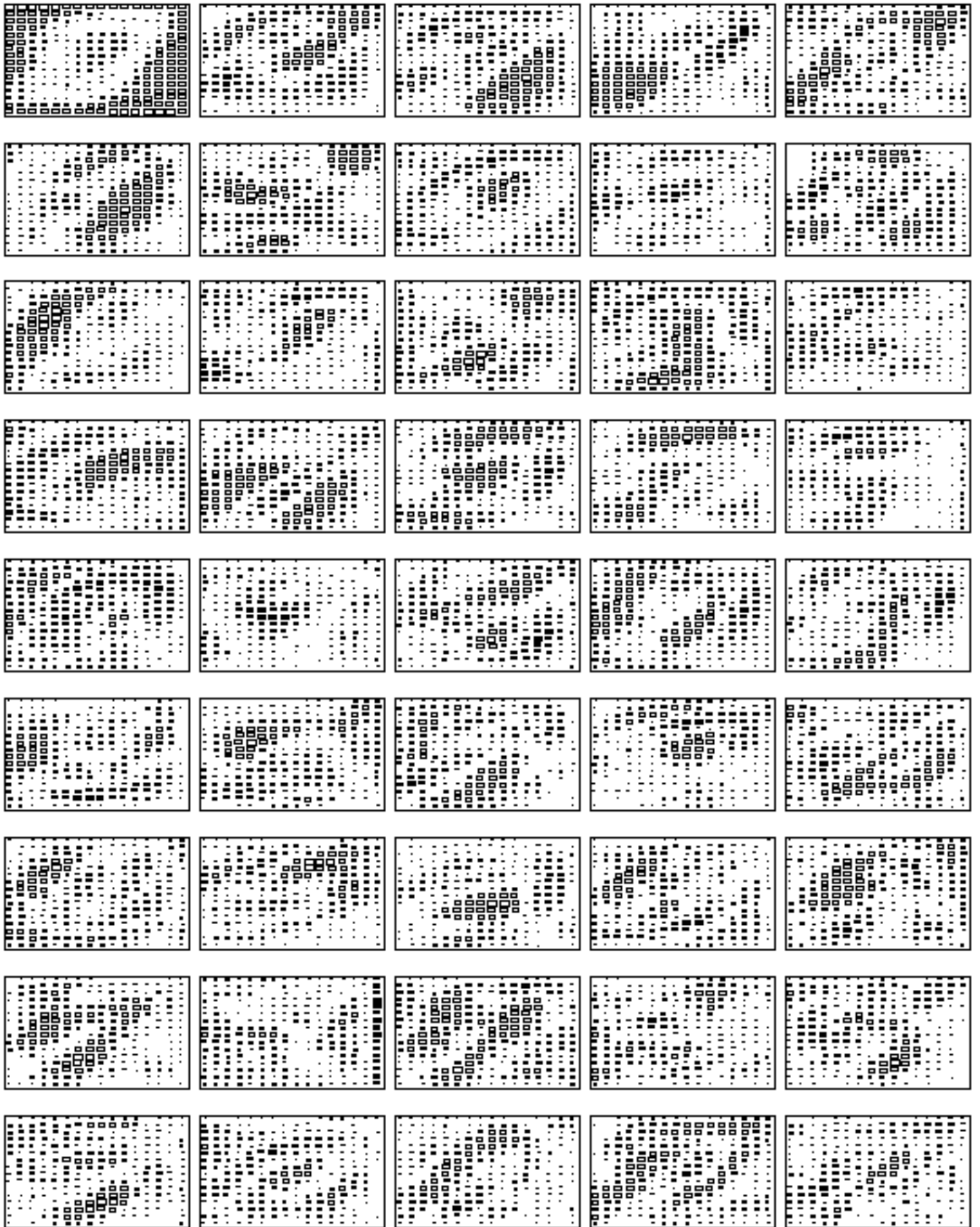


Figure 4.10: The weight vector or image templates found by the the mixture model

References

- [AHS85] D. H. Ackley, G. E. Hinton, and T. J. Sejnowski. A learning algorithm for Boltzmann machines. *Cognitive Science*, 9:147–169, 1985.
- [CS89] D. R. Cox and E. J. Snell. *Analysis of binary data*. Chapman and Hall, 1989.
- [DF84] P. Diaconis and D. Freedman. Asymptotics of graphical projection pursuit. *Annals of Statistics*, 12:793–815, 1984.
- [DH73] R. O. Duda and P. E. Hart. *Pattern Classification and Scene Analysis*. Wiley, 1973.
- [DLR77] A.P. Dempster, N.M. Laird, and D.B. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Roy. Statist. Soc. B*, 39:1–38, 1977.
- [EH81] B.S. Everitt and D.J. Hand. *Finite mixture distributions*. Chapman and Hall, 1981.
- [Fre87] D. H. Freeman. *Applied Catagorical Data Analysis*. Marcel Dekker, 1987.
- [Fri87] J. H. Friedman. Exploratory projection pursuit. *J. Amer. Stat.Assoc.*, 82(397):599–608, March 1987.
- [FWS84] J. H. Friedman, W.Stuetzle, and A. Schroeder. Projection pursuit density estimation. *J. Amer. Stat.Assoc.*, 79:599–608, 1984.
- [Gem86] Stuart Geman. Stochastic relaxation methods for image restoration and expert systems. In D.B. Cooper, R.L.Launer, and D.E. McClure, editors, *Automated Image Analysis: Theory and Experiments*. Academic Press, 1986.
- [GG84] S Geman and D Geman. Stochastic relaxations, Gibbs distributions and the Bayesian restoration of images. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 6:721–742, 1984.
- [GP87] Hector Gefner and Judea Pearl. On the probabilistic semantics of connectionist networks. Technical Report CSD-870033, UCLA Computer Science Department, July 1987.
- [HKP91] John Hertz, Anders Krogh, and Richard G. Palmer. *Introduction To The Theory Of Neural Computation*. Addison Wesley, 1991.
- [Hop82] J.J. Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proc. Natl. Acad Sci. USA*, 79:2554–2558, April 1982.
- [Hub85] P.J. Huber. Projection pursuit (with discussion). *Ann. Stat.*, 13:435–525, 1985.
- [Jol86] I.T. Jolliffe. *Principle Component Analysis*. New York: Springer-Verlag, 1986.
- [Nea90] Radford M. Neal. Learning stochastic feedforward networks. Technical report, Department of Computer Science, University of Toronto, November 1990.
- [Now90] S. Nowlan. Maximum likelihood competitive learning. In D. Touretsky, editor, *Advances in Neural Information Processing Systems*, volume 2, pages 574–582. Morgan Kaufmann, 1990.
- [Oja89] E. Oja. Neural networks, principle components, and subspaces. *Int. J. Neural Systems*, 1(1):61–68, 1989.
- [PA87] Carsten Peterson and James R. Anderson. A mean field theory learning algorithm for neural networks. *Complex Systems*, 1:995–1019, 1987.

- [Pea88] Judea Pearl. *Probabilistic Reasoning in Intelligent Systems*. Morgan Kaufmann, 1988.
- [RM86] D. E. Rumelhart and J. L. McClelland. *Parallel Distributed Processing: Explorations in the Microstructure of Cognition. Volume 1: Foundations*. MIT Press, Cambridge, Mass., 1986.
- [San89] T.D. Sanger. Optimal unsupervised learning in a single-layer linear feedforward neural network. *Neural Networks*, 2:459–473, 1989.
- [Ser80] R. J. Serfling. *Approximation Theorems of Mathematical Statistics*. John Wiley & Sons, 1980.

Appendix A. Projection distributions of the binary combination model.

In this section we use results from [DF84] to show that the projections of the binary combination model are very similar to those of the real-valued combination model when the weight vectors are small. As has been discussed in Section (2.3), the binary combination model distribution can be viewed as a mixture of 2^m generalized binomial distributions. We call these binomial distributions *binoms*. Each binom corresponds to a particular setting of the hidden vector \vec{h} and to a single Gaussian component in the real-valued model. We shall show that although the distribution of the binoms are very different from the corresponding Gaussians, their projections onto almost any direction are very similar. This implies that the projections of the binary-valued combination model are very similar to those of the real-valued combination model. Because Projection pursuit methods depend only on properties of the projections of the distribution, it is a valid approximation to use the real-valued combination model for learning distributions generated by a binary-valued combination model.

The mixture coefficients of the binoms are $Pr(\vec{h}|\phi)$ as defined in Equation (3.4). The mean of the binom corresponding to \vec{h} is $\mu(\vec{h}_i) = \tanh(\sum_{i=1}^m h_i \omega^{(i)})$ where by $\tanh(\vec{x})$ we denote the application of \tanh to each component of \vec{x} . If the weight vectors $\omega^{(i)}$ are all small then $\tanh(\sum_{i=1}^m h_i \omega^{(i)}) \approx \sum_{i=1}^m h_i \omega^{(i)}$, and we get that the means of the binoms are very close to the means of the corresponding Gaussians. Next we show that under mild assumptions, the projection of each binom is very close to a Gaussian.

Diaconis and Freedman [DF84] discuss conditions under which most projections of high-dimensional data sets are close to Gaussian. Their analysis considers large sets of points taken from high dimensional spaces. These points are not assumed to be generated by a distribution. Instead, the conditions for Gaussianity of the projection are given as geometric relations among the points. These relations must hold in the limit where both the dimension of the space and the size of the sets tends to infinity. We shall show that if the weight vectors of the combination model are generated by some distribution then, with high probability, samples generated by each binom have the required geometric properties and thus most of their projections are close to Gaussians.

We follow most of the notation used in [DF84]. Let $\vec{x}_1, \vec{x}_2, \dots, \vec{x}_N$ be vectors in R^n , this is the data set. Suppose that n, N and the data set all depend on some common index ν , and that as ν tends to infinity, so do n and N . Let S_{n-1} be the unit sphere in R^n and let γ be chosen uniformly at random from S_{n-1} . Theorem 1.1 in [DF84] states that if the following conditions hold, then the empirical distribution of $\gamma \cdot \vec{x}_i$ converges weakly to the normal distribution $\mathcal{N}(0, \sigma^2)$ in probability, as $\nu \rightarrow \infty$. Where “weak convergence” is convergence as a measure on R and “in probability” is w.r.t. the uniform distributions on S_{n-1} .

The required conditions follow. There must exist some finite and positive σ^2 such that for any positive ϵ , the following limits hold as ν tends to infinity,

$$\left| \{1 \leq j \leq N : \left| \|\vec{x}_j\|_2^2 - \sigma^2 n \right| > \epsilon n \} \right| / N \rightarrow 0 \quad (\text{A.1})$$

$$\left| \{1 \leq j, k \leq N : |\vec{x}_j \cdot \vec{x}_k| > \epsilon n \} \right| / N^2 \rightarrow 0 \quad (\text{A.2})$$

Where $\#$ denotes the cardinality of a set. The first condition intuitively means that vectors are almost all of almost the same length. The second condition means that most pairs of vectors are close to orthogonal.

We are interested in projections of samples generated by the combination model, as these are random samples, we would like to show that the geometric conditions hold with probability one. Suppose we have a sequence of binomial distributions over binary cubes of increasing dimension: $\{-1, +1\}, \{-1, +1\}^2, \dots, \{-1, +1\}^n, \dots$. Each distribution is fully specified by its mean vector: $\vec{\mu}_1 \in [-1, +1], \vec{\mu}_2 \in [-1, +1]^2, \dots, \vec{\mu}_n \in [-1, +1]^n, \dots$. Suppose that we have a sample from each distribution and that the sample size increases with the dimension n of the space: $\langle \vec{x}_1^1 \rangle, \langle \vec{x}_1^2, \vec{x}_2^2 \rangle, \dots, \langle \vec{x}_1^n, \dots, \vec{x}_n^n \rangle, \dots$. We would like to show that random projections of these samples produce empirical marginal distributions that are very close to Gaussian distributions with a probability that goes to 1 as $n \rightarrow \infty$. However, it is not hard to construct sequences of mean vectors such that this will not happen. For instance, if $\vec{\mu} = \{0, +1, \dots, +1\}$, then the distribution is concentrated in the two points $\{-1, +1, \dots, +1\}$, and $\{+1, +1, \dots, +1\}$, and all projections of this distribution will also be concentrated on two points.

We prove that the desired asymptotic conditions hold with probability 1 if the mean vectors $\vec{\mu}_n$ are selected in the following way. Assume there is some distribution \bar{P} on $[-1, +1]$ and that each component of each $\vec{\mu}_n$ is drawn independently at random from this distribution. For this to hold for the mixture components of the combination model it is enough to assume that the components of the weight vectors in the model underlying the data are chosen independently at random.

Theorem A.0.1: *Suppose that a sequence of vectors of increasing dimension:*

$$\vec{\mu}_1 \in [-1, +1], \vec{\mu}_2 \in [-1, +1]^2, \dots, \vec{\mu}_n \in [-1, +1]^n, \dots$$

is randomly drawn by selecting each component of each vector according to some distribution \bar{P} over $[-1, +1]$.

Each vector $\vec{\mu}_n$ defines a distribution over $\{-1, +1\}^n$ in which the components are independent and the expected value is $\vec{\mu}_n$. Suppose that for each n we draw n vectors from this distribution, and that from each random vector we subtract the mean, $\vec{\mu}_n$.

Suppose that for each n we draw a vector \vec{w} uniformly at random from the n dimensional unit sphere, project the n random vectors on the direction defined by \vec{w} and assign each of the points in the projection a probability mass of $1/n$. In this way we create, for each n , a discrete distribution over the reals.

*With probability one, over all the random choices that create the sequence of distributions, there exists $\sigma \geq 0$ such that the sequence of distributions converges weakly to the normal distribution $\mathcal{N}(0, \sigma^2)$.*¹

Proof: We prove the theorem by showing that the conditions of Theorem 1.1 in [DF84] hold with probability one.

The proof of the condition A.1 is a simple application of the Markov bound. We wish to show that for some σ and for any $\epsilon, \delta > 0$:

$$\lim_{n \rightarrow \infty} P(\#\{1 \leq j \leq n : ||\vec{x}_j||_2^2 - \sigma^2 n| > \epsilon n\} > \delta n) = 0$$

¹Weak convergence means that for any measurable set A , the probability assigned to A by the sequence of distributions converges to the probability of the limit distribution.

The n examples are independent, thus as n increases the fraction of the vectors that obey the condition becomes very close to the probability of obeying the condition. Thus it suffices to show that for a randomly chosen example \vec{x}

$$\lim_{n \rightarrow \infty} P(|\|\vec{x}\|_2^2 - \sigma^2 n| > \epsilon n) = 0$$

the squared length of a vector is a sum of the squares of its components. As the components are chosen independently at random according to the mean vector $\vec{\mu}_n$ and as the components of $\vec{\mu}_n$ are chosen independently at random according to \bar{P} we get that the average length of \vec{x} is $n(1 - \int_{-1}^{+1} x^2 d\bar{P}(x))$. The variance of each term is at most 1. Thus defining σ^2 to be $1 - \int_{-1}^{+1} x^2 d\bar{P}(x)$ and using Markov bounds we get that

$$P(|\|\vec{x}\|_2^2 - \sigma^2 n| > \epsilon n) \leq \frac{n}{(\epsilon n)^2} = \frac{1}{n\epsilon^2}.$$

and as n increases the probability decreases to zero as desired.

The proof of condition (A.2) is a bit more involved, because in this case the n^2 pairs that are checked for the condition are not independent. However, using the theory of U-statistics [Ser80][Chap. 5] their behavior can be related to that of independently drawn pairs. We wish to show that for any $\epsilon, \delta > 0$:

$$\lim_{n \rightarrow \infty} P(\#\{1 \leq j, k \leq N : |\vec{x}_j \cdot \vec{x}_k| > \epsilon n\} > \delta n^2) = 0$$

first observe that when $j = k$ the condition will most often not hold, as we have just proved that the squared length of a vector is concentrated around $\sigma^2 n$. However we can ignore this set as it is a vanishing fraction of the n^2 pairs. It is thus sufficient to prove that

$$\lim_{n \rightarrow \infty} P(\#\{1 \leq j, k \leq n; j \neq k : |\vec{x}_j \cdot \vec{x}_k| > \epsilon n\} > \delta n(n-1)) = 0$$

Using the notation of [Ser80] we define

$$h(\vec{x}, \vec{y}) = \begin{cases} 1 & \text{if } |\vec{x} \cdot \vec{y}| > \epsilon n \\ 0 & \text{otherwise} \end{cases}$$

and observe the corresponding U-statistic, that is a random variable defined over samples of size n :

$$U(\vec{x}_1, \dots, \vec{x}_n) = \frac{2}{n(n-1)} \sum_{1 \leq i < j \leq n} h(\vec{x}_i, \vec{x}_j)$$

This random variable is exactly the cardinality of the set of pairs that have a dot product larger than ϵn divided by $n(n-1)$. Our goal is thus reduced to proving that the probability of a sample for which U is too large is small. We do that by using Markov inequality. The fact that U is an unbiased statistic means that the average of U is equal to the average of $h(\vec{x}, \vec{y})$ when \vec{x} and \vec{y} are chosen independently at random. In other words it is equal to the probability that two randomly chosen vectors have a dot product larger than ϵn . We shall denote that probability by t . The variance of U can be related to the variance of $h(\vec{x}, \vec{y})$ by using Lemma A. from page 183 of [Ser80].

$$\text{Var}(U(\vec{x}_1, \dots, \vec{x}_n)) \leq \frac{2}{n(n-1)} [2(n-2)\zeta_1 + \zeta_2] \leq \frac{4}{n}\zeta_2$$

Where ζ_2 is simply the variance of $h(\vec{x}, \vec{y})$ when \vec{x} and \vec{y} are chosen independently at random. As $h(\vec{x}, \vec{y})$ is either 0 or 1, its variance is $t(1-t)$. Putting the bound on the variance into the Markov bound we get:

$$P [n(n-1)U(\vec{x}_1, \dots, \vec{x}_n) > \delta(n(n-1))] \leq P [|U(\vec{x}_1, \dots, \vec{x}_n) - t| > \delta - t] \leq \frac{4t(1-t)}{n(\delta-t)^2}$$

It is easy to see that

$$t = P(|\vec{x} \cdot \vec{y}| > \epsilon n) \leq \frac{4}{\epsilon^2 n}$$

thus $\lim_{n \rightarrow \infty} t = 0$ and we get that the desired probability goes to zero, which completes the proof. ■