Learning to Disentangle the Complex Causes of Data

by

Tony Butler-Yeoman

A thesis
submitted to the Victoria University of Wellington
in fulfilment of the
requirements for the degree of
Master of Science
in Computer Science.

Victoria University of Wellington
2017

Abstract

The ability to extract and model the meaning in data has been key to the success of modern machine learning. Typically, data reflects a combination of multiple sources that are mixed together. For example, photographs of people's faces reflect the subject of the photograph, lighting conditions, angle, and background scene. It is therefore natural to wish to extract these multiple, largely independent, sources, which is known as *disentangling* in the literature. Additional benefits of disentangling arise from the fact that the data is then simpler, meaning that there are fewer free parameters, which reduces the curse of dimensionality and aids learning.

While there has been a lot of research into finding disentangled representations, it remains an open problem. This thesis considers a number of approaches to a particularly difficult version of this task: we wish to disentangle the *complex causes* of data in an *entirely unsupervised setting*. That is, given access only to unlabeled, entangled data, we search for algorithms that can identify the generative factors of that data, which we call *causes*. Further, we assume that causes can themselves be complex and require a high-dimensional representation.

We consider three approaches to this challenge: as an inference problem, as an extension of independent components analysis, and as a learning problem. Each method is motivated, described, and tested on a set of datasets build from entangled combinations of images, most commonly MNIST digits. Where the results fall short of disentangling, the reasons for this are dissected and analysed. The last method that we describe, which is based on combinations of autoencoders that learn to predict each other's output, shows some promise on this extremely challenging problem.

Contents

Chapter 1

Introduction

1.1 Disentangling Coherent Causes

Real-world data is often created by the interaction of several complex processes, such as the subject of a photograph and the lighting conditions when that photograph was taken. If this is the case, a natural representation of the data is to model each cause separately, along with their method of interaction. We wish to build models that find these *disentangled representations* of the data.

Learning high quality abstract representations of data has emerged as a critical factor in the success of modern machine learning algorithms [7], and the advantage of finding disentangled representations is significant. Not only can the *causes* of the data be manipulated individually, such as generating a given face under new lighting conditions, but, by decomposing the data correctly, the representation itself becomes very efficient. For example, if data is generated from 1000 faces under 1000 lighting conditions, then there are a million possible combinations of faces and lighting. A model that correctly disentangled these causes would only need to represent 2000 possibilities, each face and each lighting condition, rather than all million combinations. More generally, modelling causes and their interactions may simply be the correct representation of data. In fact Bengio

et al. [8] note that "one could even say the ultimate goal of AI research is to build machines that can understand the world around us, i.e., disentangle the factors and causes it involves".

While a significant body of literature on learning disentangled representations exists, it remains an open problem. In particular, most successful algorithms operate in a supervised or semi-supervised setting.

This thesis explores several methods for a particularly difficult version of this task: *learning complex and coherent disentangled representations in an entirely unsupervised setting*. The 'unsupervised' portion of this statement means the learner has access to nothing more than a dataset of entangled examples; no labels associated with any cause are available, and the data cannot be grouped so as to hold one cause fixed while the other varies (such as the same face under many lighting conditions). The 'complex and coherent' portion of the statement refers to the ability to represent causes with a model that is as complex as required, in contrast with some successful unsupervised disentanglers, in which causes are limited to scalar values [11]. This is perhaps the most difficult form of the disentangling problem: the model must discover and model complex causes of data with no more information than the composite data itself.

This is an ambitious aim, and so we will limit our scope somewhat. We will primarily, though not entirely, focus on the case of causes that interact linearly. Additionally, we will only consider the case of two causes.

1.2 Goals

The overall goal of this thesis is a scientific exploration into new methods for disentangling complex causes of data in a fully unsupervised setting. We intend to explore three approaches to this problem, each with their own motivations and specific goals as follows.

1. Take a Bayesian perspective and view disentangling as an *inference* problem, and construct energy models for the task. This approach is motivated by the appealing theoretical properties of Restricted Boltzmann Machines (RBMs). Specifically, we will:

    - Describe an algorithm for disentangling based on energy models, including a motivation of the design decisions of the model, and a discussion of potential alternatives.
    - Evaluate and analyse the model's performance.

2. Explore the relation between disentangling and Independent Components Analysis (ICA). This is motivated by the clear connection between finding independent *components* and finding independent *causes*. Specifically, we will:

    - Analyse the applicability of standard ICA to the disentangling problem, and options for extending it.
    - Develop an algorithm using ICA to perform disentangling.
    - Test that algorithm on simple data and analyse its limitations.

3. View disentangling as a *learning* problem, and construct unsupervised neural networks for the task. This is motivated by the state-of-the-art results achieved by neural networks on related disentangling problems. Specifically, we will:

    - Develop, test, and analyse a modification to the architecture of a standard autoencoder that implicitly encourages disentangling.
    - Explore candidate loss functions that explicitly penalise entanglement.
    - Develop, test, and analyse a disentangling algorithm whose loss explicitly penalises entanglement.

1.3 Credit

Credit for each contribution of this thesis is as follows.

- The core model presented in chapter 3 is largely the work of Frean and Marsland. The subsequent development, testing, and analysis are primarily my contribution.
- The analysis performed in chapter 4 was discussed jointly, and refined and performed by me.
- The content of chapters 6, 8, and 9 are primarily my contribution.

1.4 Roadmap

This thesis is organised into three parts, each focused on one of the three approaches outlined in the goals. Due to the different concepts required for the three topics, the background and related work relevant to each approach is presented in the first chapter of each part.

Part I develops the Broadnet in chapter 3, which is an energy model based on RBMs. While this model is superficially appealing, unfortunately it suffers from issues that ultimately invalidate it as an approach to disentangling. This is detailed and analysed in chapter 4. Part II considers the link between disentangling and finding independent components. We find that a limited connection between the concepts can be made, and develop a post-processing algorithm based on it, but this does not lead to a generally applicable algorithm. Part III explores the application of unsupervised neural networks to the problem. Chapter 8 shows that a small structural change to an autoencoder yields successful, but inconsistent, disentangling of linearly entangled causes. Finally, chapter 9 then proposes a model using adversarial networks that improves the consistency of these results, and leads to the first step towards non-linear disentangling.

Part I

Disentangling with Energy Models

Chapter 2

Background on Energy Models

The first part of this thesis develops and analyses a model for disentangling based on Restricted Boltzmann Machines (RBMs). This chapter provides the necessary background on generative modelling and RBMs, and discusses related work.

2.1 Generative Models

A *generative model* is a probabilistic description of how data is generated. It is described by a set of random variables $\mathbf{x} = (x_1, \ldots, x_n)$, and a joint probability distribution over them, $p(\mathbf{x})$. This distribution encodes both the uncertainty present in the model and the dependence structure of the random variables. The joint often contains quantities that are not random variables, which are known as the *parameters* Θ. Because the joint is parameterised by Θ it is more correctly written $p_\Theta(\mathbf{x})$, though the subscript is commonly omitted.

In abstract, there are two operations of interest on a generative model. The first is *inference*, which is the process of calculating the distribution over latent (unknown) variables given observed variables. The second is *learning*, which involves optimising the parameters of the model to maximise some quantity, often the probability of the dataset under the model.

Generative models come in two kinds: causal and acausal models. A causal (or directed) model has a joint that models direct causality between variables. Each random variable x_i has a set of 'parents' $\mathcal{P}_i$, and the conditional distribution $p(x_i \mid \mathcal{P}_i)$ fully describes that variable. If the values of all parents are known, then x_i can be sampled. A more intuitive view of causal models is as a directed acyclic graph, where nodes are random variables and edges describe conditional dependence [42]. In other words, in a causal model the joint distribution factors into a product of conditional distributions over each variable. This is the power of causal models: with the right dependence structure, an unwieldy joint can be factored into a compact representation comprised of conditional distributions.

In contrast, acausal models have a joint distribution that does *not* factor into conditional distributions over each variable. This makes the class of models very general and well-suited to describing cases where there is no natural causality between variables. Acausal models are most commonly described by their *unnormalised* joint, $p^*(\mathbf{x})$, which can be normalised as follows to find the true joint

$$p(\mathbf{x}) = \frac{p^*(\mathbf{x})}{\int p^*(\mathbf{x} = \mathbf{X})\, d\mathbf{X}}. \tag{2.1}$$

The denominator of equation 2.1 normalises p^*, and is known as the *partition function*, Z. The normalised joint probability of an acausal model is generally intractable to compute, which can be seen in the need to integrate (or sum, in the discrete case) over all variables in order to calculate the partition function. To disambiguate, $p(\mathbf{x})$ will often be referred to as the full or true joint.

Undirected models are also known as *energy models*, and have an associated energy:

$$E = -\log p^*(\mathbf{x}). \tag{2.2}$$

This separation of the full joint into an energy and a partition is natural, in particular because inference can be performed by only considering the

energy while learning requires evaluating the full joint. This will be discussed further in later sections.

In any generative model, it is often useful to divide the variables $\mathbf{x}$ into two sets: the *observed* (or visible) variables $\mathbf{v}$, and the *latent* (or hidden) variables $\mathbf{h}$. The observed variables are those that represent the data and their values are almost always known. The latent variables include all other variables in the model. Their values are generally not known, and can intuitively be viewed as an 'hypothesis' about the data. With this convention we can define some useful terminology. The *prior*, $p(\mathbf{h})$, describes the belief about an hypothesis in the absence of data. The *likelihood*, $p(\mathbf{v} \mid \mathbf{h})$, describes how likely a particular observation is given an hypothesis. Finally, the *posterior*, $p(\mathbf{h} \mid \mathbf{v})$, models how likely an hypothesis is given an observation. The posterior of a model is often the main quantity of interest, but it is commonly difficult to calculate. Unless otherwise stated, 'inference' generally refers to posterior inference.

2.2 Markov Chain Monte Carlo

It is often the case that computing or sampling from some distribution $p(\mathbf{x})$ cannot be performed exactly. This is a concern for inference in generative models, which requires doing exactly that. The *Markov Chain Monte Carlo* (MCMC) algorithm addresses this problem by providing a way to draw samples that are, asymptotically, from $p(\mathbf{x})$.

A Markov chain is a random process that transitions between different states over time. It is initialised to some (often arbitrary) state $\mathbf{x}^0$, and at each step selects a new state to transition into from a *transition distribution*. The initial state of the Markov chain affects its behaviour in future time steps, but the magnitude of this effect lessens as the chain is run for more steps. Every Markov chain has a stationary distribution, which describes its behaviour in the absence of any effect from its initial state. When the behaviour of a Markov chain very closely matches its stationary distribu-

tion, that chain is said to have *mixed*. Depending on the complexity of the transition matrix and the quality of the initial state, mixing can take a very long time.

Gibbs sampling [48] is a commonly used MCMC algorithm, which finds samples from $p(\mathbf{x})$ by constructing a Markov chain with $p(\mathbf{x})$ as its stationary distribution [48]. This is performed by, at each time step, selecting a variable $x_i \in \mathbf{x} = (x_1, \ldots, x_n)$ and updating its value by drawing a sample from its *full conditional* distribution:

$$x_i^{t+1} \sim p\left(x_i \,\middle|\, \mathbf{x}_{-i}^t\right), \tag{2.3}$$

where $\mathbf{x}_{-i}$ represents all variables except x_i. In other words, hold all variables but x_i fixed, and draw a new value for x_i's state. It can be shown that the Markov chain underlying this process has $p(\mathbf{x})$ as its stationary distribution [21]. This is useful, as the full conditionals are often known and easy to calculate. Variables do not need to be updated in any particular or consistent order, so long as every variable is updated on every iteration. If the full conditional distributions of two (or more) variables are independent, then those variables can be updated simultaneously. This is known as block sampling. Gibbs sampling is a special case of the more general Metropolis-Hastings [51, 29] MCMC algorithm. This more general algorithm is not necessary for our work, however, and will not be described further.

A MCMC algorithm will eventually produce samples distributed according to $p(\mathbf{x})$ if and only if the underlying Markov chain is *ergodic* [37]. A chain is ergodic if all states with non-zero probability can be reached from all other states with non-zero probability in a finite number of steps. However, few guarantees can be made about the mixing properties of the Markov chain, that is, *how long* it must be run before samples approximately match $p(\mathbf{x})$. The chain's mixing time is determined by the shape of the transition distribution, and pathological cases that cause very slow mixing can easily be constructed. Two examples are when islands of high

probability are connected only by low probability paths, or when high-dimensional distributions have almost flat probability [42].

2.3 Explaining Away

Explaining away [54] is the name given to a phenomenon in generative models in which latent variables that are *independent in the prior* become *dependent in the posterior*. In other words, independent causes of data become dependent when data is observed.

The canonical example of this effect [48] is the 'burglars and earthquakes' problem. Suppose a house alarm can be triggered by either an earthquake or a burglary, and in the absence of an alarm, burglaries and earthquakes are independent events. This leads to the causal generative model shown in figure 2.1. If one knows the house alarm has been triggered, then either an earthquake or a burglary is likely to have occurred. However, if one *also* knows that an earthquake has occurred, this 'explains away' the activation of the alarm, and *reduces* the likelihood that a burglary has occurred. In other words, the independent causes (burglary and earthquake) have become dependent once the data (alarm) is observed.

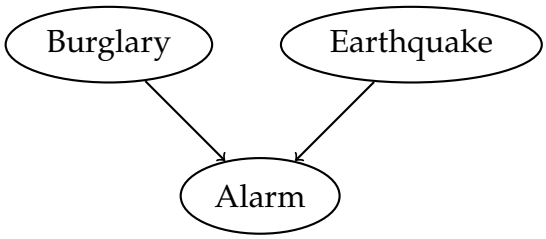


Figure 2.1: The classic example of explaining away. Two variables, burglary and earthquake, can both cause an alarm to be triggered. Without knowledge of the alarm, they are independent. With knowledge that the alarm is active, and that one of the causes has occurred, the likelihood of the other cause is reduced.

2.4 Restricted Boltzmann Machines

Restricted Boltzmann Machines (RBMs) are a popular type of undirected model developed in an early form by Smolensky [59], and later refined by Freund and Haussler [20]. A RBM is composed of binary variables, known as units, structured in two layers: the hidden layer with units h_j, and the visible layer with units v_i. We will universally use the subscript j to index hidden units and i to index visible units. These are connected in a bipartite graph structure; the two layers are fully inter-connected with weights W, but there are no connections between units in the same layer. This structure can be seen in figure 2.2.

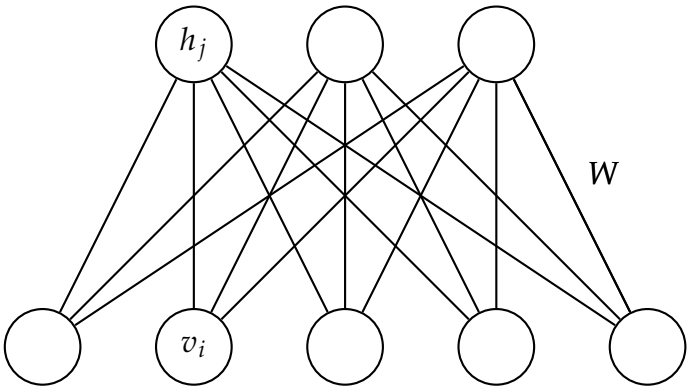


Figure 2.2: Diagram of the RBM model.

The connections between layers form a single matrix of weights W_{ji}, in which each row represents the connections of one hidden unit. Each hidden unit also has a bias parameter b_j. Visible units traditionally also have a bias parameter, but the work in the following chapters does not make use of these so they are omitted here. The unnormalised joint of a RBM is

$$\log p^*(\mathbf{v}, \mathbf{h}) = \sum_{j,i} h_j W_{ji} v_i + \sum_j b_j h_j. \tag{2.4}$$

The partition function is

$$Z = \sum_{\mathbf{v},\mathbf{h}} p^*(\mathbf{v}, \mathbf{h}), \tag{2.5}$$

where the sum over $\mathbf{v}, \mathbf{h}$ denotes the sum over the set of all possible binary assignments to the hidden and visible units, of size $2^{|\mathbf{h}|+|\mathbf{v}|}$. In the context of a RBM, we will refer to the quantity $p(\mathbf{h})$ as the *marginal*, rather than the prior. While it uses the same notation, $p(\mathbf{h})$ cannot be correctly thought of as a prior, because it shares parameters (the weights matrix W) with the likelihood.

RBMs are particularly useful, and of interest to the problem of disentangling, for their dependence structure. If the RBM were a directed model, all hidden units would be *independent* in the prior, but *dependent* in the posterior due to explaining away. In fact, the actual dependence structure is precisely the opposite; hidden units are dependent in the marginal, but independent in the posterior. Note also that visible units are independent in the likelihood.

The outcome of this is twofold. First, both the posterior and likelihood factor into distributions over each variable:

$$p(\mathbf{h} \mid \mathbf{v}) = \prod_j p\left(h_j \,\middle|\, \mathbf{v}\right) \qquad p(\mathbf{v} \mid \mathbf{h}) = \prod_i p(v_i \mid \mathbf{h}). \tag{2.6}$$

This allows for efficient sampling of either layer of units given the other. Second, the fact that the marginal does not factor allows it to model a complex distribution on the hidden units.

The observation that the marginal can model a complex distribution facilitates a more intuitive view of the model. RBMs are machinery for translating between a distribution in the visible space and a distribution in the hidden space. The marginal is not factorial by design, so that it can mirror a complex data distribution. The translation between distributions is bidirectional, however, and a single set of weights is responsible for both producing an informative hidden distribution, and reproducing the original data distribution. This creates a tension between the 'likelihood terms' and the 'posterior terms' of the joint.

2.5 Learning in RBMs

This sections discusses learning in RBM models with a focus on Contrastive Divergence (CD) [31, 34], the most commonly-used learning algorithm. As with most generative models, learning in RBMs proceeds by gradient descent on the probability of the dataset. Unfortunately, simply computing the gradient is intractable, as it requires computing the partition function. This motivates the need for an *approximate* gradient descent method. We will eventually arrive at CD by first rewriting the log probability in a useful form, then detailing a sampling algorithm, and then combining them to find gradient estimates.

2.5.1 Rewriting the Log Probability

Supposing $\mathcal{D}$ is a dataset containing N elements, we wish to compute

$$\begin{aligned}\frac{\partial}{\partial W_{ji}} \log p(\mathcal{D}) &\propto \frac{\partial}{\partial W_{ji}} \frac{1}{N} \sum_{\mathbf{v}\in\mathcal{D}} \log p(\mathbf{v}) \\ &= \frac{\partial}{\partial W_{ji}} \frac{1}{N} \sum_{\mathbf{v}\in\mathcal{D}} (\log p^*(\mathbf{v}) - \log Z) \\ &= \frac{\partial}{\partial W_{ji}} \left(\frac{1}{N} \sum_{\mathbf{v}\in\mathcal{D}} \log p^*(\mathbf{v}) \right) - \frac{\partial}{\partial W_{ji}} \log Z. \end{aligned} \tag{2.7}$$

Consider the two terms individually. The first term is simply the expectation of $\log p^*(\mathbf{v})$ over the dataset. We will use the notation $\langle \cdot \rangle_{a\sim b}$ to denote the expectation of $\cdot$ with a distributed according to b. The first term can then be rewritten as

$$\frac{\partial}{\partial W_{ji}} \left(\frac{1}{N} \sum_{\mathbf{v}\in\mathcal{D}} \log p^*(\mathbf{v}) \right) = \left\langle \frac{\partial}{\partial W_{ji}} \log p^*(\mathbf{v}) \right\rangle_{\mathbf{v} \sim \text{ data}} \tag{2.8}$$

The second term can be manipulated into a similar form as follows.

$$\frac{\partial}{\partial W_{ji}} \log Z = \frac{1}{Z} \frac{\partial}{\partial W_{ji}} Z$$

Substitute the definition of Z from equation 2.5.

$$
\begin{aligned}
&= \frac{1}{Z} \frac{\partial}{\partial W_{ji}} \sum_{\mathbf{v},\mathbf{h}} p^*(\mathbf{v},\ \mathbf{h}) \\
&= \frac{1}{Z} \sum_{\mathbf{v},\mathbf{h}} \frac{\partial}{\partial W_{ji}} p^*(\mathbf{v},\ \mathbf{h}) \\
&= \frac{1}{Z} \sum_{\mathbf{v},\mathbf{h}} \left[p^*(\mathbf{v},\ \mathbf{h}) \frac{\partial}{\partial W_{ji}} \log p^*(\mathbf{v},\ \mathbf{h}) \right] \\
&= \sum_{\mathbf{v},\mathbf{h}} \left[p(\mathbf{v},\ \mathbf{h}) \frac{\partial}{\partial W_{ji}} \log p^*(\mathbf{v},\ \mathbf{h}) \right]
\end{aligned}
$$

Finally, apply the definition of expectation.

$$
= \left\langle \frac{\partial}{\partial W_{ji}} \log p^*(\mathbf{v},\ \mathbf{h}) \right\rangle_{\mathbf{v},\ \mathbf{h} \sim p(\mathbf{v},\ \mathbf{h})} \tag{2.9}
$$

Note that each of these terms is a different expectation over the gradient of log unnormalised joint. This has a simple form:

$$
\frac{\partial}{\partial W_{ji}} \log p^*(\mathbf{v},\ \mathbf{h}) = h_j\ v_i. \tag{2.10}
$$

Substituting equations 2.8, 2.9, and 2.10 into equation 2.7, the gradient calculation can be rewritten as follows.

$$
\frac{\partial}{\partial W_{ji}} \log p(\mathcal{D}) \quad = \left\langle h_j v_i \right\rangle_{\substack{\mathbf{v} \sim \text{data} \\ \mathbf{h} \sim p(\mathbf{h} \mid \mathbf{v})}} - \left\langle h_j v_i \right\rangle_{\mathbf{v},\ \mathbf{h} \sim p(\mathbf{v},\ \mathbf{h})} \tag{2.11}
$$

These two terms are known as the Hebbian and anti-Hebbian steps of learning, respectively, and have an appealing interpretation.

If learning is considered to be the act of shaping the distribution over hidden units, then the Hebbian term attempts to place probability mass such that the data is represented well, and the anti-Hebbian term attempts to make the posterior as flat as possible. This has the effect of encouraging all data to be associated with a high probability optimum in hidden space, and discouraging any high probability hidden optimum to be disassociated from the data. When the posterior and marginal are in agreement,

the data distribution is perfectly represented in the hidden space, the Hebbian and anti-Hebbian terms are equal, and learning ends.

In terms of computational efficiency, the Hebbian term can be easily computed because the posterior is factorial. However, the anti-Hebbian step requires drawing samples from the joint, which is not factorial and cannot be computed efficiently. Instead, we turn to MCMC sampling to draw approximate samples from a Markov chain, which can then be used to approximate the expectation of the anti-Hebbian step.

2.5.2 Sampling from the Joint

Following the procedure from section 2.2, the full conditional distribution on each variable must be calculated, which will be shown here for hidden units. First, some notation is required. Let $\mathbf{h}_{-j}$ denote the set of all hidden units except h_j, and let $\mathbf{h}_{j \leftarrow a}$ denote the set of all hidden units, where the value of h_j is fixed at a.

Consider the full conditional $p\left(h_j = 1 \,\middle|\, \mathbf{v}, \mathbf{h}_{-j}\right)$, denoted p_j for convenience. This can be expanded to

$$\begin{aligned} p_j &= p\left(h_j = 1 \,\middle|\, \mathbf{v}, \mathbf{h}_{-j}\right) \\ &= \frac{p^*\left(\mathbf{h}_{j \leftarrow 1}, \mathbf{v}\right)}{p\left(\mathbf{h}_{-j}, \mathbf{v}\right) Z}. \end{aligned} \tag{2.12}$$

Now take the ratio of probabilities between the two possible states of h_j. Note that the probability of h_j equaling 0 is $1 - p_j$. Hence, the denominator of equation 2.12 cancels in the ratio. An important consequence of this is that the partition function Z does *not* need to be calculated to perform

posterior inference. This leaves

$$\frac{p_j}{1-p_j} = \frac{p^*\Big(\mathbf{h}_{j\leftarrow 1}, \mathbf{v}\Big)}{p^*\Big(\mathbf{h}_{j\leftarrow 0}, \mathbf{v}\Big)}$$

$$\implies p_j = \frac{1}{1 - \frac{p^*\Big(\mathbf{h}_{j\leftarrow 1}, \mathbf{v}\Big)}{p^*\Big(\mathbf{h}_{j\leftarrow 0}, \mathbf{v}\Big)}}. \tag{2.13}$$

It is convenient to perform this calculation in log space. Let $\sigma(\cdot)$ be the logistic sigmoid function, defined as

$$\sigma(x) = \frac{1}{1+e^{-x}}. \tag{2.14}$$

Equation 2.13 can be re-written in log space using a sigmoid as follows.

$$\begin{aligned} p_j &= \sigma\left(\log \frac{p^*\Big(\mathbf{h}_{j\leftarrow 1}, \mathbf{v}\Big)}{p^*\Big(\mathbf{h}_{j\leftarrow 0}, \mathbf{v}\Big)}\right) \\ &= \sigma\Big(\log p^*\Big(\mathbf{h}_{j\leftarrow 1}, \mathbf{v}\Big) - p^*\Big(\mathbf{h}_{j\leftarrow 0}, \mathbf{v}\Big)\Big). \end{aligned}$$

Substituting the definition of the energy from equation 2.4, all terms except those involving h_j cancel, yielding

$$p_j = \sigma\left(\sum_i W_{ji} v_i + b_j\right). \tag{2.15}$$

This completes the Gibbs update for a single hidden unit. The Gibbs update for visible units is derived in a similar fashion, and is

$$p(v_i = 1 \mid \mathbf{v}_{-i},\ \mathbf{h}) = \sigma\left(\sum_j h_j W_{ji}\right). \tag{2.16}$$

Because all hidden units are independent in the posterior, and all visible units are independent in the likelihood, the Gibbs updates for all units within a layer can be performed in parallel. This leads to an efficient block Gibbs sampling algorithm: first sample all hidden units, then sample all visible units, and continue for as long as desired.

2.5.3 Contrastive Divergence

In brief, the CD algorithm estimates the gradient of the log probability of the dataset by computing equation 2.11 using the MCMC sampling procedures described previously to approximate the expectations over the posterior and the joint.

In more detail, equation 2.11 requires expectations over the posterior and the joint. CD first draws a sample from the exact posterior and computes the Hebbian update. This sample is then used as the initial state for a Markov chain performing block updates defined by equations 2.15 and 2.16. This chain is run for k steps, where k is a hyperparameter, and the resulting visible and hidden assignments are used to compute the anti-Hebbian update. This is also known as CD-k. A diagram of the Markov chain is provided in figure 2.3.

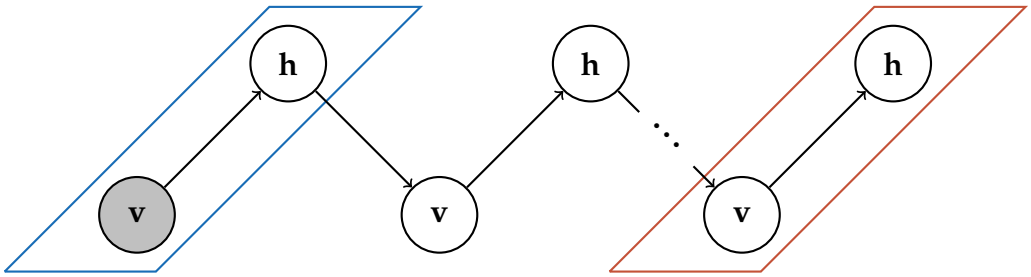


Figure 2.3: Diagram of CD-k learning in a RBM. The first value of $\mathbf{v}$ is shaded to indicate it is fixed by the data while the rest are found by the model. Each arrow denotes a sample taken from $p(\mathbf{h} \mid \mathbf{v})$ or $p(\mathbf{v} \mid \mathbf{h})$. The blue box (left) indicates the samples of $\mathbf{v}$ and $\mathbf{h}$ used to compute the Hebbian update, and the red box (right) indicates the samples used to compute the anti-Hebbian update.

It is clear that CD-k is not computing the gradient, as the anti-Hebbian expectation over the joint is replaced by a single sample from a Markov chain that may not have mixed. This is the key idea of CD: effective learning does not necessarily require a correct gradient, but rather a gradient

that is correct enough to move the model's parameters in the right direction. Indeed, the most commonly used version of this algorithm is CD-1, in which the Markov chain is run for only one step.

In practice, the performance of CD is improved by using mean-field values, i.e. the values p_j rather than a Bernoulli sample, for the Hebbian and anti-Hebbian updates. This results in an exact computation for the Hebbian step, and has the effect of reducing noise in the anti-Hebbian step that provides no additional benefit [30].

Several modifications on CD learning exist, the most popular being Persistent Contrastive Divergence (PCD) [63]. Under this scheme, the Hebbian phase of learning remains unchanged, but the samples used in the anti-Hebbian phase are drawn from a collection of Markov chains whose state persists between iterations of learning. These chains are intended to provide a better estimate of the partition function, by remaining closer to a mixed state at all times. On the assumption that each learning iteration only changes the joint by a small amount, then the persistent chains should be able to re-mix in a small number of steps.

2.6 RBM Variants

The basic RBM structure has been extended in a variety of ways, some of which are presented here. Our work will primarily make use of Gaussian RBMs.

Gaussian RBMs (GRBMs) [33] are the most simple extension of RBMs to model real-valued visibles. The energy of a GRBM is

$$\log p^*(\mathbf{v}, \mathbf{h}) = \sum_{ji} h_j W_{ji} \frac{v_i}{\sigma} - \frac{1}{2\sigma^2} \sum_i v_i^2 + \sum_j b_j h_j, \tag{2.17}$$

where σ is a variance parameter. Note that we use the symbol 'σ' for two concepts: $\sigma(x)$ refers to the logistic sigmoid function, and σ refers to a variance. The energy of a GRBM differs from the energy of a standard RBM

only in the addition of a quadratic term, and a division of all occurrences of the visible units by σ. GRBMs are often defined with a *vector* of variance parameters, σ_i, but for our uses the variance parameters are mostly constant. The addition of the quadratic term leaves the posterior unchanged, but results in the following likelihood. If $\phi = \mathbf{h}W$, then

$$p(\mathbf{v} \mid \mathbf{h}) \propto \exp\left(\frac{-(v_i - \phi_i)^2}{2\sigma}\right), \tag{2.18}$$

that is, a Gaussian centred on ϕ with variance σ. While the procedure for sampling the visible units given the hiddens has changed, the learning rule remains the same as in a standard RBM.

ReluRBMs [52] provide a clever way of introducing continuous hidden units. Each hidden unit is replaced by an infinite number of copies sharing the same weight and bias, but with a stepped offset added to the bias. This is close to a closed form.

$$\sum_{k=1}^{N} \sigma(W\mathbf{v} + \mathbf{b} - k + 0.5) \approx \log\left(1 + e^{W\mathbf{v}+\mathbf{b}}\right) \tag{2.19}$$

The infinite number of copies of a particular binary unit approximates a smoothed ReLU unit, allowing for positive, real-valued states. This is shown to significantly improve the performance of RBMs, particularly when used in conjunction with Gaussian visible units.

Mean-Covariance RBMs (mcRBMs) [32] extend GRBMs. They are motivated by the observation that hidden units effectively model the means of input data, but poorly capture the covariance structure between visible units. A mcRBM partitions the hidden units into two types: mean units, which are standard binary hidden units, and covariance units, which model pairwise interactions between visible units. The mean units contribute a Gaussian with complex mean and diagonal covariance, while the covariance units contribute a zero-mean Gaussian with possibly complex covariance. While the model is trained with CD as per a standard RBM,

the covariance matrix of $p(\mathbf{v} \mid \mathbf{h})$ is no longer diagonal, and requires a matrix inverse to calculate. This presents a problem for learning, as this inverse must be computed on every learning iteration in order to sample the likelihood [6]. For any reasonably large number of visible units, this is intractable. An alternative hybrid Monte Carlo training algorithm has been proposed, which avoids calculating the matrix inverse, but presents problems for training [13, 6].

Finally, Spike-and-Slab RBMs (ssRBMs) [13] replace each hidden unit of a GRBM with the product of a pair of variables: a binary 'spike' and real-valued 'slab'. The most recent iteration of the model, the μssRBM [5], includes two weight matrices which allow it to model some covariance in the visible units as in a mcRBM. This gives rise to a somewhat complex energy function. Let $\mathbf{v}$ denote the visible units, $\mathbf{h}$ the 'spike' hidden units, $\mathbf{s}$ the 'slab' hidden units, and w_j and b_j denote the weight vector and bias associated with spike and slab units j. Let Φ be the additional weights matrix, α_j a scalar parameter that penalises large values of s_j, Λ a diagonal matrix that penalises large visible unit values, and μ_j a scalar parameter that models the mean of unit s_j. The energy of the model is:

$$E(\mathbf{v}, \mathbf{s}, \mathbf{h}) = -\sum_{j=1}^{N} \mathbf{v}^T w_j s_j h_j + \frac{1}{2} v^T \left(\Lambda + \sum_{j=1}^{N} \Phi_j h_j \right) \mathbf{v} + \frac{1}{2} \sum_{j=1}^{N} \alpha_j s_j^2 - \sum_{j=1}^{N} \alpha_j \mu_j s_j h_j - \sum_{j=1}^{N} b_j h_j + \sum_{j=1}^{N} \alpha_j \mu_j^2 h_j.$$

Unlike the mcRBM model, standard CD learning can be performed efficiently in a ssRBM.

2.7 What is a Cause?

Before we discuss related work, it is useful to have a more formal definition of an 'independent cause' to hand. The intuitive meaning is fairly clear: a

cause is one of the independent processes, such as faces or lighting, that create the data. To formalise this notion, we tentatively define an independent cause as a partition of latent variables such that the two partitions are *statistically independent*. This definition fails to capture the nuance of the problem, and will be developed further throughout the thesis. Nevertheless, we will take it as a working definition for the time being.

2.8 Related Work

This section discusses work related to the disentangling algorithm proposed in the next chapter. Broadly, most existing algorithms for disentangling causes use one of three methods: bilinear models, energy models, or neural networks. This section discusses work related to bilinear models and energy models. Approaches based on neural networks are discussed in chapter 8.

2.8.1 Bilinear Models

One of the first approaches to disentangling factors of variation is the bilinear model, proposed by Tenenbaum and Freeman [61], which presents a tool for separating 'style' from 'content'. Style and content refer to two underlying factors of the data, such as letters and their handwriting style, or spoken words and the speaker's accent. In a symmetric bilinear model, elements of a data vector $\mathbf{x}$ are modelled by

$$x_i = \mathbf{a}^T W_i \mathbf{b}, \tag{2.20}$$

where $\mathbf{a}$ is a latent style vector, $\mathbf{b}$ a latent content vector, and W_i a matrix of weights. The collection of matrices W_i, one for each i, forms a map from the content and style space to the data space which is *bilinear*; if one of the vectors is held fixed, the model is linear in the other. However the content and style vectors interact multiplicatively to generate the data, allowing for complex interaction between the two models. This approach has

been extended to n causes in multi-linear models [65] through the use of tensors, and has been specialised for translation invariance [28] and image sequences [53]. On data such as handwritten digits and faces, bilinear and multilinear models have been shown to separate causes such as style and content; or face, lighting, and pose. However training this class of models requires an *ordered* matrix of data containing all combinations of all causes. This amounts to a strongly supervised problem, which is a simpler task than the one we wish to solve.

2.8.2 RBM Models

RBM-based models have a history of usage in disentangling problems, most commonly in the form of 'higher-order' energy models that involve extra latent parameters affecting the hidden units. This section discusses two of the most successful RBM-based disentanglers.

Higher-Order Spike and Slab RBMs

An extension of Spike and Slab RBMs, Higher-Order Spike and Slab RBMs (hossRBMs) [16] add partitioned latent variables that interact multiplicatively. The energy function of hossRBMs is complex, and the model is better described by its parameters. The hidden units are grouped into k stacked layers, with each stack gated by a latent variable f_k. Two groups of spike and slab hidden units are used, g and h, which interact multiplicatively. Along with v visible units, this results in a four-dimensional ($v \times k \times g \times h$) tensor of parameters. To reduce the size of the parameter space, a block-sparsity pattern is imposed on g and h: each g parameter is non-zero only across one row, and each h parameter is non-zero only down one column.

hossRBMs are trained without labels, but on datasets where all combinations of the two factors of variation are present. On the Toronto Face Database [60], there is some specialisation of the two sets of units (see fig-

ure 4 of Desjardins et al. [16]) into emotion and identity, but there is room for improvement.

Disentangling Boltzmann Machines

Disentangling Boltzmann Machines (DisBMs) [55] are a model for disentangling factors of variation that makes use of a partitioned hidden layer like the one proposed in section 3.1. Starting with a standard RBM, the hidden units are partitioned into two groups. In addition to the standard RBM energy terms, multiplicative factors between all three groups of units (two groups of hiddens, one group of visibles) are added. These factors result in the two groups of hiddens being dependent in the marginal, making posterior inference and Hebbian-phase learning intractable. Instead, a variational approximation is used. The model can be used with full labels, 'partial' labels that only explain one of two factors of variation, or no labels at all. The authors note that the model has poor performance when no labels are used, but achieves "modest" disentangling when labels are used.

ORBM

The ORBM model, proposed by Godfrey, Frean, and Marsland [23], is the most closely related model to the one proposed in section 3.1. The ORBM models two coherent, independent causes of *binary* data by constructing an undirected model from two RBMs. More formally, the marginal is that of two RBMs, and the likelihood of a given visible is the sigmoid of the sum of the two causes' outputs. The fact that the likelihood is nonlinear in each cause creates the need for a 'correction term', which is intractable to calculate but can be well approximated.

The ORBM model achieved some success in the context of inference when RBMs are each pre-trained on a cause, but has not been tested on full learning, i.e. disentanglement.

Chapter 3

The Broadnet Model

This chapter describes the Broadnet, an RBM-based algorithm for disentangling complex causes. Section 3.1 motivates the use of RBMs, section 3.2 presents the model, section 3.3 develops its inference and learning algorithms, section 3.4 discusses the model and explores some alternatives, and section 3.5 provides experimental results.

3.1 Setting and Motivation

The task of disentangling coherent and complex causes has broad scope. For that reason we will simplify the setting somewhat and model data that is generated by two independent causes that are individually complex, but which interact simply. Specifically, we will make the following assumptions.

- The data is real-valued.
- There are exactly two causes of the data.
- The two causes interact *additively*.

It is reasonable to question whether RBMs are the right tool for this task, as their results are mostly inferior to modern neural networks. However the

RBM model has a key property which is present in neither directed generative models nor neural networks: not only are the hidden units dependent in the marginal, but they are also *discouraged* from becoming factorial during learning. In other words, the model is capable of describing a complex latent distribution and is also discouraged from having units represent independent factors of variation. This would seem an ideal property for a model of a coherent cause.

The inclination of an RBM to model a coherent cause provides the motivation for combining several RBMs to form a model of multiple causes.

3.2 Model

This section describes the Broadnet model, and will begin with a general overview. We construct a model for disentangling by placing several RBMs in parallel, such that each has a set of hidden units, but share a common set of visible units. We name this model the 'Broadnet', to show that its complexity lies in the 'width' of several RBMs, in contrast with the structure of a deep net [25]. We intend for each of these RBMs to model a single complex cause.

We will focus on the case of two causes, leading to the model visualised in figure 3.1. As we wish to model real-valued data, the visible units will be real-valued, making each RBM a GRBM. Each visible unit will be an element-wise sum of the contribution from each GRBM, thus mirroring the assumptions about how elements of the data are generated.

By rights, each constituent GRBM in the Broadnet model should have its own noise parameter and visible bias vector, however we simplify the Broadnet architecture by using a single noise parameter σ for all causes and omitting the visible biases altogether. We treat σ as a hyperparameter that is not learned, and so use of a shared value does not couple the causes. It has been observed [30] that learning in GRBMs is often made more difficult by the presence of visible biases, and experiments suggest that this

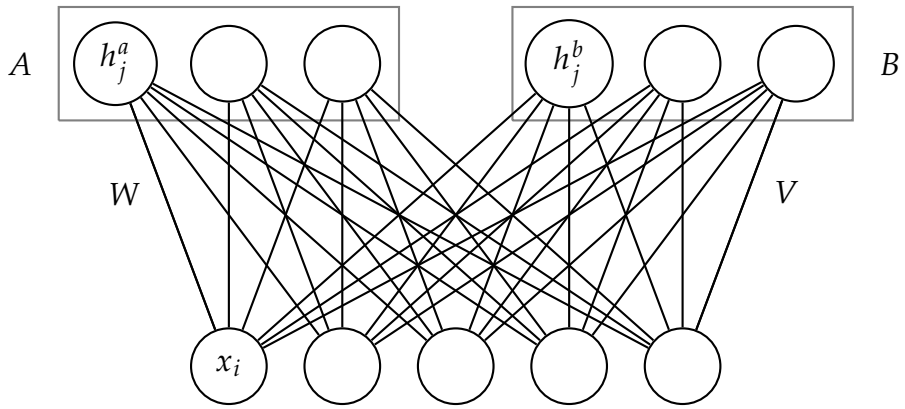


Figure 3.1: Diagram of the Broadnet model. The hidden units of each cause, A and B, are shown boxed.

problem is exacerbated in Broadnets. Due to this we will never use visible biases, and they are omitted from the derivation for ease of exposition. Hence, the parameters of a Broadnet are two weight matrices and two hidden bias vectors, and the variables are two hidden vectors and one visible vector.

In similar fashion to other energy model-based disentanglers [16, 55], Broadnets are intended to disentangle the causes of data via inference. While the two causes are independent in the marginal, they are dependent in the posterior due to explaining away. This dependency results in the need for an MCMC inference procedure based on trading residuals between the two networks. The procedure resembles 'negotiation' for responsibility of components of the data. We hypothesise that the benefit to the joint produced by each GRBM modelling a single cause, enabled by the process of negotiation, will drive the Broadnet to disentangle.

3.2.1 Terms and Notation

In the context of a GRBM, we use W to denote the weights matrix, $\mathbf{b}$ the bias, $\mathbf{h}$ the hidden state and $\mathbf{v}$ the visible state. In the context of a Broadnet, the notation is used as follows.

- When combined in a Broadnet, each GRBM is referred to as a 'cause'. These are individually referred to as the A and B causes.

- W and V denote the weights matrices of the A and B causes respectively.

- $\mathbf{b}^a$ and $\mathbf{b}^b$ denote the hidden bias vectors of the A and B causes respectively.

- $\mathbf{h}^a$ and $\mathbf{h}^b$ denote the state of the hidden units of the A and B causes respectively.

- ϕ^a and ϕ^b denote the reconstructions from each cause, that $\phi^a = \mathbf{h}^a \cdot W$ and $\phi^b = \mathbf{h}^b \cdot V$.

- $\mathbf{x}$ denotes the state of the visible units.

- $\mathcal{D}$ denotes the input dataset as a whole.

- σ is the noise parameter. Note that σ is used to denote noise, and $\sigma(x)$ the logistic sigmoid function.

Some mathematical shorthand will also be used.

- If $p(x)$ is a distribution then the statement $\sum_x p(x)$ is more correctly written $\sum_X p(x = X)$. However, when the variable and its assignment have the same name, we will use the simpler form.

- Whenever a norm's subscript is left unspecified, it refers to the 2-norm squared: $\|\cdot\| = \|\cdot\|_2^2$.

- In general, a or b superscripts *never* refer to powers, but to variables related to causes A or B.

3.2.2 Derivation

This section provides a more rigorous derivation of the Broadnet (BN) model. Like other generative models, the Broadnet model is described by its joint. First, observe that the joint can be decomposed into a likelihood term and a marginal term, as in equation 3.1, which will be derived individually. This is not a factorisation in the sense of a directed generative model, as the terms share parameters.

$$p_{\text{BN}}\left(\mathbf{x},\ \mathbf{h}^a,\ \mathbf{h}^b\right) = p_{\text{BN}}\left(\mathbf{x} \,\middle|\, \mathbf{h}^a,\ \mathbf{h}^b\right) p_{\text{BN}}\left(\mathbf{h}^a,\ \mathbf{h}^b\right) \tag{3.1}$$

A key model decision is that each cause is independent in the marginal, so the second term can be factored into the product of each cause's marginal.

$$p_{\text{BN}}\left(\mathbf{x},\ \mathbf{h}^a,\ \mathbf{h}^b\right) = p_{\text{BN}}\left(\mathbf{x} \,\middle|\, \mathbf{h}^a,\ \mathbf{h}^b\right) p_{\text{BN}}(\mathbf{h}^a)\ p_{\text{BN}}\left(\mathbf{h}^b\right) \tag{3.2}$$

As each individual cause is a GRBM, the marginal on a cause $p_{\text{BN}}(\mathbf{h}^a)$ is simply a GRBM's marginal $p_{\text{GRBM}}(\mathbf{h})$. Hence, we must derive two terms: the likelihood of a Broadnet and the marginal on a GRBM.

Likelihood. The likelihood term describes how the visible pattern is generated given the two hidden patterns. Since we wish to model real valued data with additive causes, the natural choice is to model the likelihood as a Gaussian centred on the sum of the causes. If $\phi^a = \mathbf{h}^a W$ and $\phi^b = \mathbf{h}^b V$ then

$$\begin{aligned} p_{\text{BN}}\left(\mathbf{x} \,\middle|\, \mathbf{h}^a,\ \mathbf{h}^b\right) &= \mathcal{N}\left(\phi^a + \phi^b,\ \sigma\right) \\ &= \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{1}{2\sigma^2} \sum_i \left(\mathbf{x}_i - \phi_i^a - \phi_i^b\right)^2\right). \end{aligned} \tag{3.3}$$

Marginal. Next, we must find the marginal induced by a single GRBM. Recall that the unnormalised joint of a GRBM is defined as

$$\log p^*_{\text{GRBM}}(\mathbf{h},\ \mathbf{v}) = \frac{1}{\sigma} \sum_{ji} h_j W_{ji} v_i - \frac{1}{2\sigma^2} \sum_i v_i^2 + \mathbf{b} \cdot \mathbf{h}, \tag{3.4}$$

where σ is the noise parameter. Define $\phi = \mathbf{h}W$, the reconstruction of the data from hiddens $\mathbf{h}$ and perform some rearrangement of the unnormalised log-joint:

$$\begin{aligned}
\log p^*_{\text{GRBM}}(\mathbf{h},\ \mathbf{v}) &= \frac{1}{\sigma}\sum_i \phi_i v_i - \frac{1}{2\sigma^2}\sum_i v_i^2 + \mathbf{h}\cdot\mathbf{b} \\
&= \frac{-1}{2\sigma^2}\sum_i \Big[v_i^2 - 2\phi_i v_i \sigma\Big] + \mathbf{h}\cdot\mathbf{b} \\
&= \frac{-1}{2\sigma^2}\sum_i \Big[(v_i - \phi_i\sigma)^2 - \phi_i^2\sigma^2\Big] + \mathbf{h}\cdot\mathbf{b} \\
&= \frac{-1}{2\sigma^2}\sum_i (v_i - \phi_i\sigma)^2 + \frac{1}{2}\sum_i \phi_i^2 + \mathbf{h}\cdot\mathbf{b}. \qquad (3.5)
\end{aligned}$$

Exponentiating $\log p^*_{\text{GRBM}}(\mathbf{h},\ \mathbf{v})$ finds the unnormalised joint, and including the partition Z results in the normalised joint. Equation 3.6 shows this, and is rearranged to emphasise that the joint is a product of Gaussians, and some terms not involving v_i.

$$p_{\text{GRBM}}(\mathbf{h},\ \mathbf{v}) = \frac{1}{Z}\prod_i \left[\exp\left(\frac{-(v_i - \phi_i\sigma)^2}{2\sigma^2}\right)\right] \exp\left(\frac{1}{2}\sum_i \phi_i^2 + \mathbf{h}\cdot\mathbf{b}\right) \qquad (3.6)$$

Importantly, observe from its definition in equation 2.5 that the partition Z is not a function of v_i. This allows the visible units to be integrated out:

$$\begin{aligned}
p_{\text{GRBM}}(\mathbf{h}) &= \int_{-\infty}^{\infty} p(\mathbf{v},\mathbf{h})\ \ d\mathbf{v} \\
&= \int_{-\infty}^{\infty} \frac{1}{Z}\prod_i \left[\exp\left(\frac{-(v_i - \phi_i\sigma)^2}{2\sigma^2}\right)\right] \exp\left(\frac{1}{2}\sum_i \phi_i^2 + \mathbf{h}\cdot\mathbf{b}\right)\ d\mathbf{v} \\
&= \frac{1}{Z}\int_{-\infty}^{\infty} \prod_i \left[\exp\left(\frac{-(v_i - \phi_i\sigma)^2}{2\sigma^2}\right)\right]\ d\mathbf{v}\ \ \exp\left(\frac{1}{2}\sum_i \phi_i^2 + \mathbf{h}\cdot\mathbf{b}\right) \\
&= \frac{1}{Z}\prod_i \left[\int_{-\infty}^{\infty} \exp\left(\frac{-(v_i - \phi_i\sigma)^2}{2\sigma^2}\right)\ dv_i\right]\ \ \exp\left(\frac{1}{2}\sum_i \phi_i^2 + \mathbf{h}\cdot\mathbf{b}\right) \\
&= \frac{1}{Z}\sqrt{2\pi\sigma^2}\exp\left(\frac{1}{2}\sum_i \phi_i^2 + \mathbf{h}\cdot\mathbf{b}\right). \qquad (3.7)
\end{aligned}$$

Joint. Finally, the marginals and the likelihood can be combined to form the joint. This leads to the following log unnormalised probability, which is denoted J for convenience.

$$J = \frac{-1}{2\sigma}\left\|x - \phi^a - \phi^b\right\| + \frac{1}{2}\|\phi^a\| + \frac{1}{2}\left\|\phi^b\right\| + \mathbf{h}^a \cdot \mathbf{b}^a + \mathbf{h}^b \cdot \mathbf{b}^b \tag{3.8}$$

The partition function is simply the product of the factors in equations 3.3 and 3.7 outside of the exponential or, equivalently, the definition in equation 2.5 applied to equation 3.8. Denoting the partition in each of the two marginals by Z^a and Z^b,

$$\begin{aligned} Z &= \frac{Z^a Z^b}{\sqrt{2\pi\sigma^2}} \\ &= \sum_{\mathbf{h}^a,\, \mathbf{h}^b} \int_{-\infty}^{\infty} J \; d\mathbf{x}. \end{aligned} \tag{3.9}$$

This completes the derivation of the Broadnet model.

3.2.3 Properties of the Model

The construction of the Broadnet model has several notable properties. By construction, the two constituent GRBMs are *independent in the marginal*, but the units within each GRBM are *dependent in the marginal*. From a generative perspective, this dependence structure is ideal for a model of two complex causes: the networks model independent concepts, but the units within a network model related concepts.

In contrast, the two constituent GRBMs are *dependent in the posterior*, $p_{\mathrm{BN}}\left(\mathbf{h}^a, \mathbf{h}^b \,\middle|\, \mathbf{x}\right)$. This is due to explaining away between the two causes; each unit in one cause is dependent on all units in the other cause given the data. This dependence is desirable, as it captures the entanglement of the data: the potentially difficult problem of calculating the posterior represents the work of disentangling the causes, and is why the model is said to disentangle via inference. This dependence also means the posterior cannot be computed in one step, in contrast to a GRBM.

Fortunately, several other quantities are easily computed. As with a GRBM, the likelihood $p_{\text{BN}}\left(\mathbf{x} \,\middle|\, \mathbf{h}^a, \mathbf{h}^b\right)$ can be computed in one step, as visible units are independent givens the hiddens. Additionally, units within a cause are independent given both the data and the units in the other cause. As such, the posterior over a single cause $p_{\text{BN}}\left(\mathbf{h}^a \,\middle|\, \mathbf{h}^b, \mathbf{x}\right)$ can be computed in one step. These properties are used in the following section to create tractable posterior and joint inference algorithms.

3.3 Inference and Learning

This section describes the algorithms used in training a Broadnet. We begin with posterior and joint inference, and then present the learning algorithm.

3.3.1 Posterior Inference

As discussed previously, posterior inference is made difficult by the dependence between $\mathbf{h}^a$ and $\mathbf{h}^b$ given $\mathbf{x}$. Instead of computing the posterior exactly, as in a GRBM, we use a block Gibbs sampling algorithm to generate samples from the posterior.

Following the procedure for Gibbs sampling in section 2.2, we first find the full conditional distribution on each hidden unit $p\left(h_j^a = 1 \,\middle|\, \mathbf{x},\ \mathbf{h}^b,\ \mathbf{h}_{-j}^a\right)$, denoted p_j for convenience. This derivation is similar to a standard RBM's, taking advantage of that fact that units within a cause are independent, and that the partition does not depend on the value of h_j^a.

$$\frac{p_j}{1-p_j} = \frac{p\left(h_j^a = 1 \,\middle|\, \mathbf{x},\ \mathbf{h}^b,\ \mathbf{h}_{-j}^a\right)}{p\left(h_j^a = 0 \,\middle|\, \mathbf{x},\ \mathbf{h}^b,\ \mathbf{h}_{-j}^a\right)}$$

$$= \frac{p^*\left(h_j^a = 1 \,\middle|\, \mathbf{x},\ \mathbf{h}^b\right)}{p^*\left(h_j^a = 0 \,\middle|\, \mathbf{x},\ \mathbf{h}^b\right)}$$

Solving for p_j gives a familiar form.

$$p_j = \frac{1}{1 - \frac{p^*\left(h_j^a=1 \,\middle|\, \mathbf{x},\, \mathbf{h}^b\right)}{p^*\left(h_j^a=0 \,\middle|\, \mathbf{x},\, \mathbf{h}^b\right)}} \tag{3.10}$$

This can be converted into log space with the inclusion of a sigmoid.

$$p_j = \sigma\Big(\log p^*\left(h_j^a = 1 \,\middle|\, \mathbf{x},\, \mathbf{h}^b\right) - \log p^*\left(h_j^a = 0 \,\middle|\, \mathbf{x},\, \mathbf{h}^b\right)\Big) \tag{3.11}$$

All terms not related to h_j^a cancel inside the log, yielding the posterior in equation 3.12. This process is symmetric for h_j^b, which is also presented below.

$$\begin{aligned} p\left(h_j^a = 1 \,\middle|\, \mathbf{x},\, \mathbf{h}^b\right) &= \sigma\left(W_{j\cdot}(\mathbf{x} - \phi^a)\right) \\ p\left(h_j^b = 1 \,\middle|\, \mathbf{x},\, \mathbf{h}^a\right) &= \sigma\left(V_{j\cdot}\left(\mathbf{x} - \phi^b\right)\right) \end{aligned} \tag{3.12}$$

With the full conditional distribution in hand, Gibbs sampling can be performed. Due to the fact that units within a cause are independent, each whole cause can receive a Gibbs update at once. This results in a block Gibbs sampling algorithm, presented in algorithm 1.

The *iterative, residual-driven nature* of posterior inference makes for an appealing approach to disentangling. To infer the state of a cause, $\mathbf{h}^a$, one calculates the residual from the other cause, $\mathbf{x} - \phi^b$. The standard GRBM posterior calculation is then performed, but with the data term replacing the residual term as seen in 3.12. This leads to an inference algorithm in which each cause receives the components of the data not yet explained by the other cause, and attempts to explain them as well as possible. Throughout inference, the two networks 'trade' these residuals in order to model the data, in a process we liken to *negotiation of responsibility for the data*. This negotiation is intended to perform the work required for posterior inference, made difficult by explaining away, and thereby disentangle the data.

Algorithm 1 Broadnet posterior inference

NB. The algorithm is presented as starting inference with the A network, but the initial network should be chosen at random to avoid bias and the algorithm adjusted accordingly.

parameters: Markov chain length l.

input: data element $\mathbf{x}$.

$\mathbf{x}^a \leftarrow$ sample from $\mathcal{N}(\mathbf{x} \cdot W,\ \sigma)$
$\mathbf{h}^a \leftarrow$ sample from Bernoulli($\sigma(\mathbf{x}^a)$)

for l iterations **do**

$\mathbf{x}^b \leftarrow$ sample from $\mathcal{N}((\mathbf{x} - \mathbf{x}^a) \cdot V,\ \sigma)$
$\mathbf{h}^b \leftarrow$ sample from Bernoulli$\Big($ $\sigma\Big(\mathbf{x}^b\Big)$ $\Big)$

$\mathbf{x}^a \leftarrow$ sample from $\mathcal{N}\Big(\Big(\mathbf{x} - \mathbf{x}^b\Big) \cdot W,\ \sigma\Big)$
$\mathbf{h}^a \leftarrow$ sample from Bernoulli($\sigma(\mathbf{x}^a)$)

return most recent samples $\mathbf{h}^a$ and $\mathbf{h}^b$, or their mean-field values.

Algorithm 2 Broadnet joint inference

parameters: Markov chain length k.

input: initial values of $\mathbf{h}^a$ and $\mathbf{h}^b$, likely the output of algorithm 1.

for k iterations **do**

$\mathbf{x}^a \leftarrow$ sample from $\mathcal{N}(\mathbf{h}^a \cdot W,\ \sigma)$
$\mathbf{h}^a \leftarrow$ Bernoulli sample from Bernoulli($\sigma($ $W \cdot \mathbf{x}^a + \mathbf{b}^a$ $)$)

$\mathbf{x}^b \leftarrow$ sample from $\mathcal{N}\Big(\mathbf{h}^b \cdot V,\ \sigma\Big)$
$\mathbf{h}^b \leftarrow$ sample from Bernoulli$\Big(\sigma\Big($ $V \cdot \mathbf{x}^b + \mathbf{b}^b$ $\Big)\Big)$

return most recent samples $\mathbf{x}^a$, $\mathbf{h}^a$, $\mathbf{x}^b$, $\mathbf{h}^b$, or their mean-field values.

3.3.2 Joint Inference

Fortunately, sampling from the joint of a Broadnet is no harder than from the joint of a GRBM. Because the two causes are independent when the data is not fixed, separate Markov chains for each cause can be run independently. These Markov chains are identical to those used in GRBM joint sampling. Algorithm 2 provides details of drawing these samples.

3.3.3 Learning Algorithm

Learning in Broadnets proceeds by gradient descent, but suffers the same difficulty as standard RBMs: calculating the gradient in a Broadnet requires calculating the gradient of the partition, which is intractable. We circumvent this problem in the same manner as standard RBMs, and use a Contrastive Divergence-based learning algorithm to estimate the gradient.

Analogous to the derivation in section 2.5, the gradient of the log probability of the dataset $\mathcal{D}$ with respect to some parameter θ can be rewritten in terms of expectations. Let the subscript 'posterior' denote $\mathbf{x} \sim$ data and $\mathbf{h}^a,\ \mathbf{h}^b \sim p\left(\mathbf{h}^a,\ \mathbf{h}^b \,\middle|\, \mathbf{x}\right)$, and the subscript 'joint' denote $\mathbf{x},\ \mathbf{h}^a,\ \mathbf{h}^b \sim p\left(\mathbf{x},\ \mathbf{h}^a,\ \mathbf{h}^b\right)$. The joint can then be rewritten as follows.

$$\frac{\partial}{\partial\theta}\log p(\mathcal{D}) \;=\; \left\langle \frac{\partial}{\partial\theta}\, J \right\rangle_{\text{posterior}} - \left\langle \frac{\partial}{\partial\theta}\, J \right\rangle_{\text{joint}} \tag{3.13}$$

The derivatives of J with respect to a weight W_{ji} and a bias h^a_j are

$$\begin{aligned} \frac{\partial}{\partial W_{ji}}\, J &= h^a_j\left(\mathbf{x}_i - \phi^b_i\right) \\ \frac{\partial}{\partial h^a_j}\, J &= h^a_j. \end{aligned} \tag{3.14}$$

The derivatives with respect to V_{ji} and h^b_j are symmetric. Substituting into equation 3.13 yields the learning rules in equation 3.15. These equations are simplified by the fact that, in the case of sampling from the joint, $\mathbf{x} =$

$\phi^a + \phi^b$ and so $x_i - \phi_i^b = \phi_i^a$.

$$\begin{aligned}
\frac{\partial}{\partial W_{ji}} \log p(\mathcal{D}) &= \left\langle h_j^a \left(x_i - \phi_i^b \right) \right\rangle_{\text{posterior}} - \left\langle h_j^a \, \phi_i^a \right\rangle_{\text{joint}} \\
\frac{\partial}{\partial V_{ji}} \log p(\mathcal{D}) &= \left\langle h_j^b \left(x_i - \phi_i^a \right) \right\rangle_{\text{posterior}} - \left\langle h_j^b \, \phi_i^b \right\rangle_{\text{joint}} \\
\frac{\partial}{\partial h_j^a} \log p(\mathcal{D}) &= \left\langle h_j^a \right\rangle_{\text{posterior}} - \left\langle h_j^a \right\rangle_{\text{joint}} \\
\frac{\partial}{\partial h_j^b} \log p(\mathcal{D}) &= \left\langle h_j^b \right\rangle_{\text{posterior}} - \left\langle h_j^b \right\rangle_{\text{joint}}
\end{aligned} \tag{3.15}$$

These learning rules are the same as in a GRBM, save for the replacement of x_i with $x_i - \phi_i^b$ or $x_i - \phi_i^a$. Here again the residual-based nature of the model is clear: each network receives a gradient only from parts of the input not explained by the other cause.

Using the procedures for finding posterior and joint samples detailed in the previous two sections, CD learning can be applied to this update rule to find an estimate of the true gradient. A key difference between GRBM and Broadnet learning is that both posterior and joint samples are estimated by Markov chains, which is shown diagrammatically in figure 3.2, to be compared with figure 2.3. The final algorithm for a step of learning is provided in algorithm 3.

Algorithm 3 Broadnet learning algorithm

input: data element $\mathbf{x}$.

$\mathbf{h}^{a+}$, $\mathbf{h}^{b+} \leftarrow$ mean-field posterior samples from algorithm 1
$\mathbf{h}^{a-}$, $\mathbf{h}^{b-}$, $\mathbf{x}^{a-}$, $\mathbf{x}^{b-} \leftarrow$ mean-field joint samples from algorithm 2
Perform learning updates according to equation 3.15.

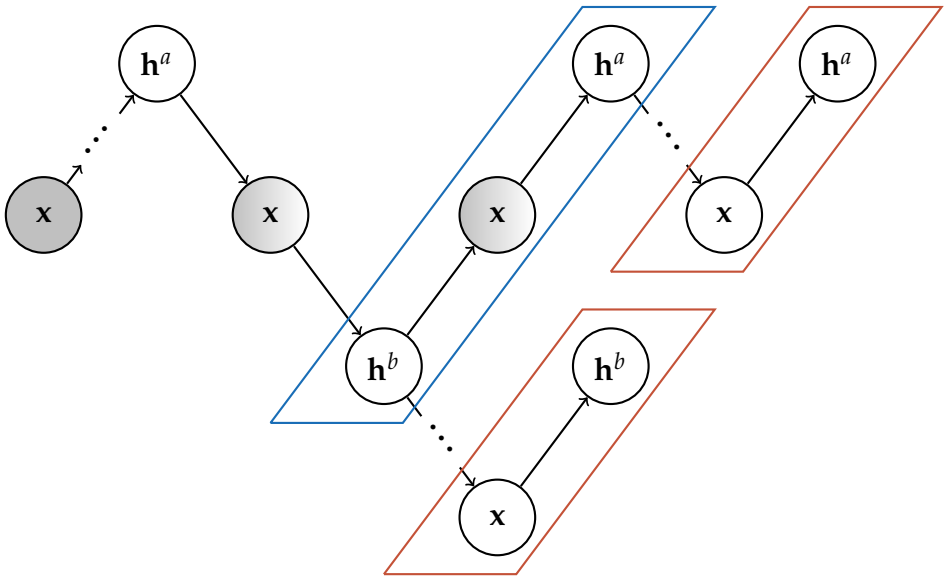


Figure 3.2: Diagram of CD-k learning in a Broadnet. The first $\mathbf{x}$ node is shaded to denote it is fixed to the data, and the other $\mathbf{x}$ of the Hebbian phase are half-shaded to denote $\mathbf{x} - \phi$. The nodes within the blue box are used for the Hebbian update, and the nodes within the red boxes are used for the anti-Hebbian update.

3.4 Discussion

With the Broadnet model and training algorithm now detailed, we can take a broader view of its properties as a model of disentanglement. The model has several attractive traits: in the marginal the causes are independent, and units within a cause are dependent. This is consistent with the idea of complex, coherent, and independent causes. In the posterior, units within a cause are independent, allowing for efficient sampling, while the causes are dependent, providing opportunity for negotiation. Both inference and learning have an appealing similarity to their standard RBM counterparts, with occurrences of the data being replaced by a residual from the other cause.

The core justification for Broadnet's disentanglement ability is the restricted dependence structure of the hidden units. The model contains no explicit incentive towards disentangling, but rather relies on the advantages brought about by representing independent causes in each model, which are recovered by the residual-trading inference algorithm. This algorithm performs the non-trivial work of reversing the entanglement of the causes.

Broadnets introduce one extra hyperparameter in addition to those of a standard GRBM: the length of the Markov chain used to collect posterior samples. Due to the presumed need for negotiation between the two causes, we expect this Markov chain to require several iterations, making the inference process slower than in a GRBM, though not intractable. It is unclear how long the posterior and joint chains need to be but, as a conservative estimate, both chains are generally run for 10 iterations.

3.4.1 Alternative RBMs

While GRBMs have been chosen as the basis of the Broadnet model, they are known to have limited representation power [13] and are often difficult to train. Ideally, they could be replaced with one of several popular improvements on the basic GRBM model, but few are suitable for use in a Broadnet. This section discusses some of the more common GRBM variants and their applicability to the Broadnet model.

ReluRBMs [52] show a significant improvement in performance and, in practice, ease of training when compared to GRBMs. This improvement can be attributed to the fact that hidden units are now effectively continuous, increasing representation power. Unfortunately this property also makes ReluRBMs unsuitable for our purpose, because inference must be performed on a hidden space that is rectilinear in W. Consider the behaviour of a Markov chain producing either joint or posterior samples. On each iteration of the chain, unit activations are multiplied by W. If it is run for n steps, the initial hidden activation is multiplied by W^n, the weights

matrix raised to the power of n. Unless all eigenvalues of W are exactly 1, this results in hidden values either vanishing or exploding. This issue exists in standard ReluRBMs as well, but is avoided by only performing CD-1 training. This is not an option in a Broadnet, as it relies on long Markov chains.

Mean-Covariance RBMs (mcRBMs) [32] also significantly improve the representation power of GRBMs, but it is not clear how to incorporate their inference procedure into a Broadnet. Our inference algorithm relies on sampling the likelihood of a single cause to approximate the posterior, but likelihood calculations are intractable in these alternative models. Their training algorithm avoids direct likelihood calculations by instead sampling from $p(\mathbf{x})$ [6] using a hybrid Monte-Carlo method, which rules out the possibility of residual-trading inference algorithm of the kind presented here. While a model for disentangling could be constructed with mcRBMs, it would be too far a departure from standard inference for us to consider.

Spike-and-Slab RBMs (ssRBMs) [13, 5] are a more recent modification to RBMs, and are noted as a "drop-in replacement" [13] for GRBMs. ssRBMs do not have problems with their model that prevent their use in a Broadnet, although the success of ssRBMs has been focused on convolutional models and data, with which we do not wish to complicate our model. These are the best candidate replacement for GRBMs, though our experiments indicate that, in practice, they can be as tricky to train as GRBMs. Initial success with Broadnets made from GRBMs would motivate us to pursue the use of ssRBMs further.

Finally, a clear issue with GRBMs is their requirement that the magnitude of their weights not only reconstruct the data well, but provide the correct amount of uncertainty in hidden activations. This is an example of the tension between likelihood and posterior terms as discussed in section 2.4: if the magnitude of the data is increased 10-fold, then increasing the weights accordingly results in $W\mathbf{v}$ increasing 100-fold, significantly alter-

ing the stochasticity of $p(\mathbf{h} \mid \mathbf{v})$. One solution would be to modify GRBMs to only reconstruct the direction of a data vector, not its magnitude. This could be achieved by constructing a model where

$$p(\mathbf{v} \mid \mathbf{h}) = \int_0^\infty \mathcal{N}(c \cdot W\mathbf{v},\ \sigma)\ \ dc. \tag{3.16}$$

Considering a simplified case with $\sigma = 1$ and no hidden or visible biases, this leads to the following log unnormalised joint.

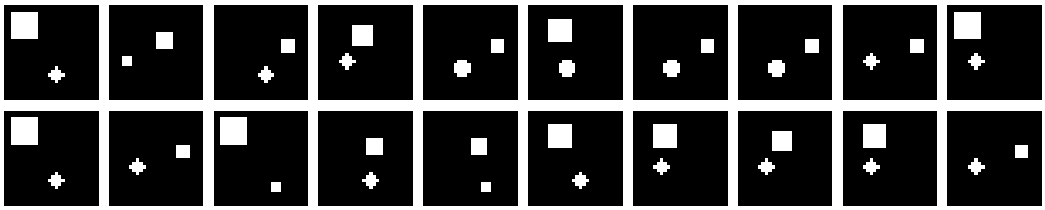
$$\log p^*(\mathbf{v}, \mathbf{h}) = k + \sqrt{\frac{\pi}{\|\mathbf{v}\|}} + \frac{\|\mathbf{h}W\mathbf{v}\|}{\|\mathbf{v}\|} \tag{3.17}$$

Unfortunately, due to the quadratic term of the energy, neither posterior nor likelihood inference can be performed in one step. Therefore, the inference algorithm in a Broadnet made from these models would require all units to be sampled sequentially, rather than in blocks, making it very slow. It appears that most models where magnitude is somehow controlled will suffer from this problem.

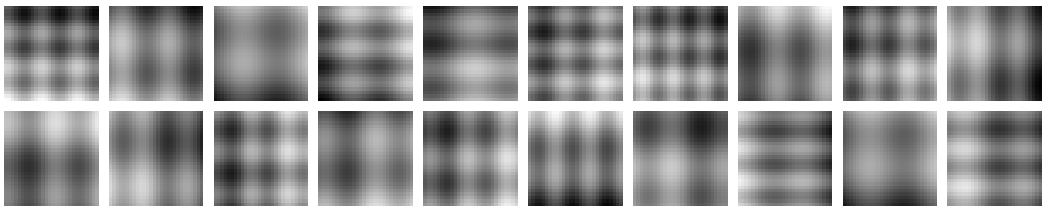
3.5 Results

With the Broadnet fully derived, this section presents some results on three tests of varying difficulties. In each test, a Broadnet is trained on data derived from MNIST. Broadnets have also been trained on several toy datasets with simple causes, some examples of which are shown in figure 3.3. Results on this data yield no additional information to the more difficult cases, and so are omitted.

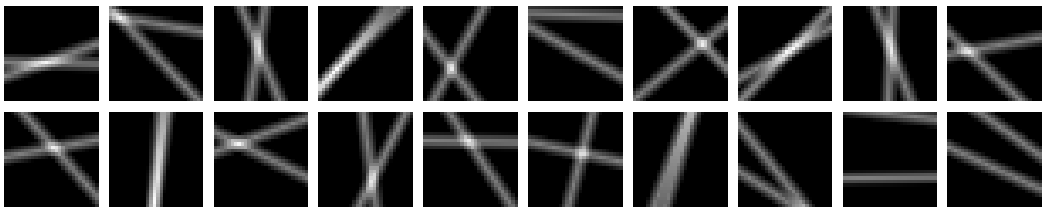
The first test trains on 1000 MNIST 5's added to a linear gradient across the image. The gradient is determined by only one parameter, the angle of increase across the image. The digits and gradients are both normalised into the range $[0, 1]$ before being added together. Figure 3.4 provides examples of this data, as well as the weights and reconstructions after training.



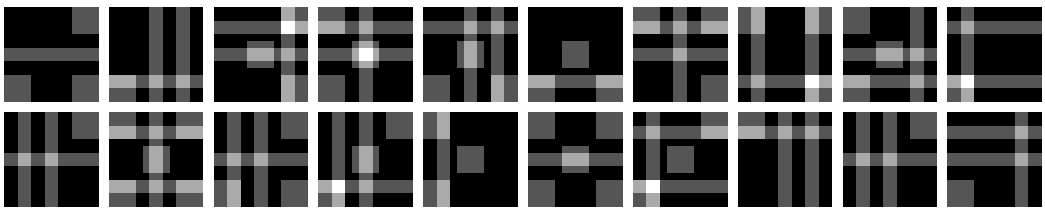
(a) A small circle and square independently take one of five locations and sizes.



(b) A combination of sine waves, generated from a single random variable, is added to a linear gradient.

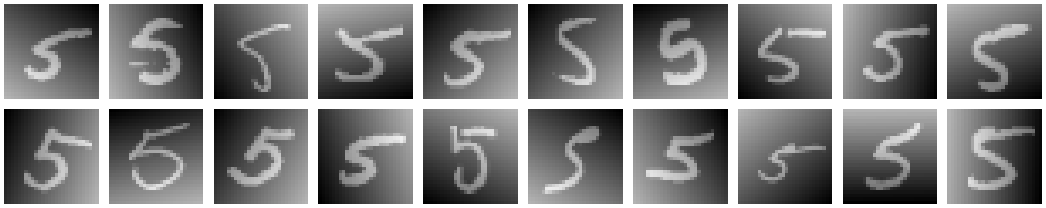


(c) Two lines are generated, the offset and orientation of each line are generated from a single random variable.

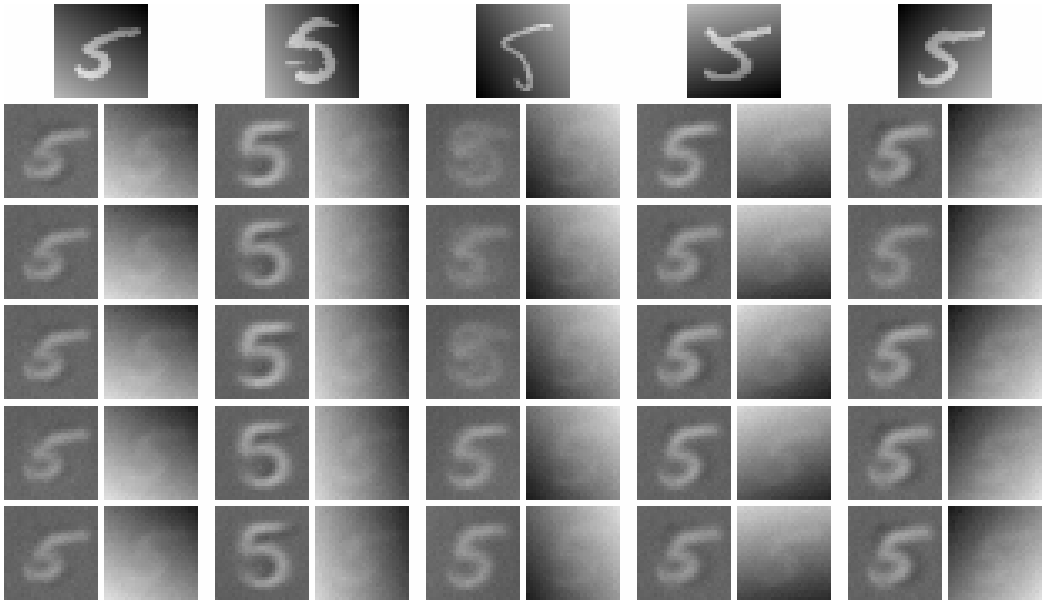


(d) A pattern of bars, in six locations, and blocks, in five locations, are generated independently and added together. The requirement of an odd number of each entangles the causes.

Figure 3.3: A sample of toy datasets used for testing. The experiments of section 3.5 have been performed on these datasets, but results do not differ from the more complex MNIST-based data and so are not presented.



(a) Samples from input data.



(b) Example reconstructions.

Figure 3.4: The first example of training a Broadnet on simple two-cause data. Each input image, examples of which are shown in figure (a), contains a 5 from MNIST added to a linear gradient. Figure (b) shows five examples of the reconstructions throughout several steps of inference. The top row displays the input image, and each pair of columns displays image as reconstructed by one of the two causes. The rows display the reconstructions throughout the first five steps of inference. Despite the simplistic representation of a 5, the causes appear to be fairly well disentangled.

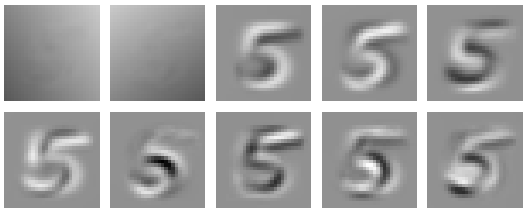


Figure 3.5: Top 10 principal components of the data shown in figure 3.4 (a). The first two principal components mostly represent the linear gradient, and the remaining components represent the digit.

These show promising disentangling; one cause makes a poor representation of a 5, and the other a good representation of a gradient with only the shadow of a 5 in it. However, two factors complicate matters.

First, this test case is less difficult than it appears at first sight. As shown in figure 3.5, the gradient is in fact entirely represented by the first two principal components of the data, and the following components represent increasing detail of a 5. This means the outcome of the model has been to imperfectly learn the first two principal components in one cause, and perhaps the first three in the other; a less impressive feat than learning independent causes.

Second, training is not robust to hyperparameters, and several tweaks must be made for a successful outcome. Each cause of the data has been normalised to the range $[-1, 1]$, rather than $[0, 1]$. Additionally, the model is extremely vulnerable to what could be called the 'mean-optimum', in which all units represent a small fraction of the data mean, and none of the variance. To combat this, an abnormally small minibatch size must be used, no greater than 5. In addition, contrary to the standard recommendation in RBM models [30], the Hebbian and anti-Hebbian updates must use samples instead of mean-field unit activations. This introduces extra noise that aids in escaping the mean optimum. If any of these tweaks are not used the model will fail to train.

The second test trains on 1000 MNIST 3's added to MNIST 7's that have been rotated by 90 degrees, both normalised to $[-1, 1]$, as shown in figure 3.6 (a). This is a much harder task than the previous one, and the causes do not factor nicely into principal components. Significant effort has been put towards finding a learning setup in which some sign of disentangling occurs, but unfortunately this has not succeeded. Our efforts focus on the important hyperparameters of learning: minibatch size, learning rate, value of the noise parameter, Markov chain length, number of hidden units, initial bias value, initial weight scale, momentum coefficient, and regularisation. We test both L1 regularisation, penalising the mean absolute weight value, and L2 regularisation, penalising the mean squared weight value.

To provide evidence for the scope of this problem, a large grid search has been performed. The tested combinations are listed in table 3.1 and, while detailed results are not presented, no run disentangled. These tests do not encompass all learning setups manually tested, as some restrictions had to be made for computational practicality.

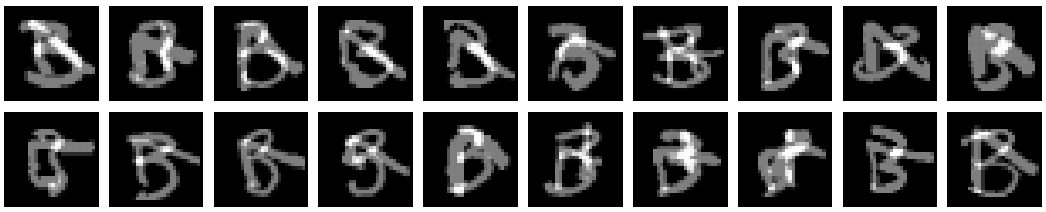
Detailed results of each test from table 3.1 are omitted. Instead, figure 3.6 (b) presents reconstructions from the test whose reconstructions of each network individually had the smallest mean squared error with the data's constituent digits, across the whole dataset. Acknowledging that mean squared error is not necessarily the correct measure of disentangling, reconstructions from each network were also manually checked; none were noticeably better than the one presented.

Additionally, several other tweaks to the learning setup have been tested, some of which will be discussed briefly.

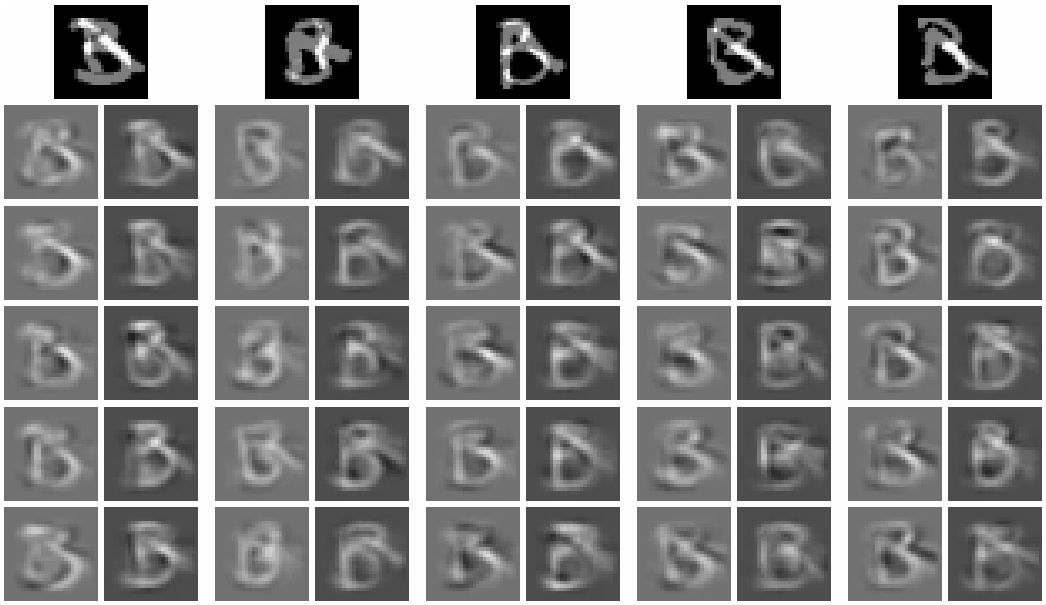
Bias initialisation. We have tested different bias initialisation methods, including random and constant initialisation, as well as ensuring all biases are initially negative. This aims to solve the tendency for units to be mostly active at initialisation, which can lead to degenerate solutions. Empirically, early learning of high-quality hidden biases is one of the most important factors for successful training. While careful

Table 3.1: Combinations of learning setups tested for disentanglement in a grid search.

Hyperparameter	Range of settings
Input data type	Trivial with 2 visibles or a subset of MNIST with 784 visibles.
Input data range	$[-n, n]$ and $[0, n]$ for $n \in \{0.1, 1, 5\}$
Learning rate	0.00001 or 0.01
Momentum	0 or 0.9
Optimiser	SGD
Minibatch size	1, 32, or 512
Regularisation	L1 or L2 with coefficient 0.01
Noise parameter	0, 0.1, or 1
(anti-)Hebbian chain length	10
Hidden units per cause	10 or 1000
Bias initialisation	Constant values 0 or -1, or uniform in ranges $[-1, 1]$ or $[0, 1]$.
Weight initialisation	Uniform in ranges $[-n, n]$ for $n \in \{0.01, 0.1\}$.



(a) Samples from input data.



(b) Reconstructions after training.

Figure 3.6: Second example of training a Broadnet, using data containing two digits added together. Sample inputs are shown in figure (a). The model consistently finds an entangled optimum after some training, such as the one shown in figure (b), which is the lowest error model from the grid search specified in table 3.1. Figure (b) shows the reconstructions over several iterations of inference in the same manner as figure 3.4.

initialisation makes training to a meaningful solution more reliable, it has no effect on the disentanglement of those solutions.

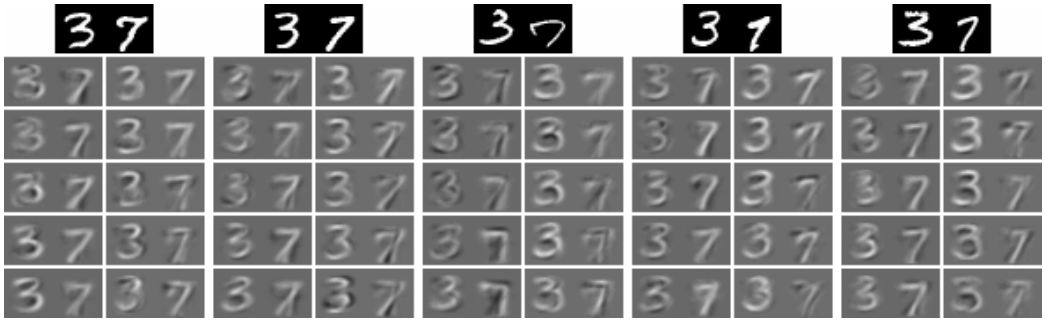
Learning rates. We have tested using separate learning rates for the weights and biases. This is motivated by the same observation as the previous point: learning hidden biases quickly results in hidden units that better model variation in the data. Empirically, biases can tolerate significantly higher learning rates than weights can, which improves training, but does not cause disentanglement.

Adaptive optimisers. We have tested using adaptive gradient optimisers, often with high learning rates and large amounts of momentum on an annealing schedule. This is intended to allow the model to avoid low-quality local optima that may be entangled. As with the previous tweaks, this leads to better training, but no disentanglement.

Data preprocessing. We have tested normalising the causes of the dataset (e.g. individual digits) to different ranges, such as $[-n, n]$ and $[0, n]$, as well as whitening the data. These measures attempt to allow complex posterior distribution to be created more easily. Results indicate that whitening the data lessens the model's vulnerability to the mean-optimum, likely due to the lack of visible biases. Any method that ensures input data has both positive and negative pixels helps prevent hidden units degenerating to always being active, which allows for a more complex posterior and better representations. None of these improvements result in disentangling.

The third test uses a simpler dataset, in which the causes are complex, but only trivially entangled. This aims to remove any potential problems created by the difficulty of finding a disentangled representation, and test whether the model is drawn towards disentangling at all. The data is created by simply placing two MNIST digits next to each other, as seen in figure 3.7 (a). A similar suite of tests was performed to those shown in table

(a) Samples from input data.



(b) Reconstructions after training.

Figure 3.7: Third example of training a Broadnet, using data with complex causes that are trivially 'entangled'. Each image contains an MNIST '3' and an MNIST '7' placed side-by-side, as seen in figure (a). Despite the fact that disentangling is trivial on this dataset, the model consistently converges to an entangled optimum. Figure (b) shows such an optimum, presented in the same manner as in figure 3.4.

3.1. Reconstructions from the model with the best mean squared error are shown in figure 3.7 (b). Even in the case where disentangling is trivial, the learning algorithm appears incapable of finding a disentangled optimum.

The almost completely negative outcome on even easy test cases casts doubt on the Broadnet as a model for disentangling. This raises a question, why don't Broadnets disentangle? This is the subject of the next chapter.

Chapter 4

Understanding Broadnets

The previous chapter introduced the Broadnet model and algorithms for disentangling data which, unfortunately, appears to have several fundamental issues that prevent successful disentangling. This chapter will explore these issues.

Datasets

Most of the experiments in this chapter have a common setup, which will be described here. With the exception of some examples using toy models, all models have 784 visible units and are trained on 1000 images derived from MNIST. Each image contains two causes added together: an upright digit, either a '0' or '1', and a digit rotated 90 degrees, either a '2' or '4'. Each cause's image is normalised to the range $[0, 1]$, so the resulting image is in the range $[0, 2]$. Some examples are shown in figure 4.1. Several experiments in this chapter construct Broadnets from two individually trained GRBMs. In this case, each is trained on one of the causes of the data.

Almost every experiment in this chapter has also been performed on several simpler datasets, some of which were presented in the previous chapter in figure 3.3. The results on these datasets are not specifically discussed, as they are almost all consistent with the results on the more complex data in figure 4.1.

(a) Left cause (b) Right cause (c) Both causes

Figure 4.1: Example images from the most commonly used dataset.

4.1 Calculating the Full Joint

Energy models are most commonly reasoned about using the unnormalised joint, as the full joint requires computing the partition function. However, the true joint allows for insight into the behaviour of the model that is difficult to attain otherwise, and several experiments in this chapter will make use of it. This section will briefly derive an analytic form of the loss. Recall that the unnormalised joint and partition are

$$\begin{aligned} p^*\left(\mathbf{x}, \mathbf{h}^a, \mathbf{h}^b\right) &= \exp\left(-\frac{1}{2}\left\|\mathbf{x} - \left(\phi^a + \phi^b\right)\right\| + \frac{1}{2}\|\phi^a\| + \frac{1}{2}\left\|\phi^b\right\| + \mathbf{h}^a \cdot \mathbf{b}^a + \mathbf{h}^b \cdot \mathbf{b}^b\right) \\ Z &= \sum_{\mathbf{h}^a,\, \mathbf{h}^b} \int_{-\infty}^{\infty} p^*\left(\mathbf{x}, \mathbf{h}^a, \mathbf{h}^b\right) d\mathbf{x}. \end{aligned} \tag{4.1}$$

The integral in the partition function Z is a Gaussian integral and so is analytic, leading to the following equation:

$$\begin{aligned} &\int_{-\infty}^{\infty} p^*\left(\mathbf{x}, \mathbf{h}^a, \mathbf{h}^b\right) d\mathbf{x} \\ &= \int_{-\infty}^{\infty} \exp\left(-\frac{1}{2}X^2 + \left(\phi^a + \phi^b\right)X - \frac{1}{2}\phi^a\phi^b + \mathbf{h}^a \cdot \mathbf{b}^a + \mathbf{h}^b \cdot \mathbf{b}^b\right) d\mathbf{x} \\ &= \sqrt{2\pi}\exp\left(\frac{1}{2}\|\phi^a\| + \frac{1}{2}\left\|\phi^b\right\| + \mathbf{h}^a \cdot \mathbf{b}^a + \mathbf{h}^b \cdot \mathbf{b}^b\right). \end{aligned} \tag{4.2}$$

The true joint can then be found, as

$$p\left(\mathbf{x}, \mathbf{h}^a, \mathbf{h}^b\right) = \frac{p^*\left(\mathbf{x}, \mathbf{h}^a, \mathbf{h}^b\right)}{Z} = \frac{\exp\left(-\left\|\mathbf{x} - \left(\phi^a + \phi^b\right)\right\| + \|\phi^a\| + \left\|\phi^b\right\| + \mathbf{h}^a \cdot \mathbf{b}^a + \mathbf{h}^b \cdot \mathbf{b}^b\right)}{\sum_{\mathbf{h}^a,\, \mathbf{h}^b} \sqrt{2\pi} \exp\left(\frac{1}{2}\|\phi^a\| + \frac{1}{2}\|\phi^b\| + \mathbf{h}^a \cdot \mathbf{b}^a + \mathbf{h}^b \cdot \mathbf{b}^b\right)}. \tag{4.3}$$

In this form the true joint can be computed exactly, as can the log evidence as follows.

$$\log p(\mathcal{D}) = \sum_{\mathbf{x} \in \mathcal{D}} \log \left[\sum_{A,B} p\left(\mathbf{x}, \mathbf{h}^a, \mathbf{h}^b\right) \right] \tag{4.4}$$

While the true joint can be calculated, it is exponentially costly to do so due to the sum over all hidden configurations.

The form of the joint given in equation 4.3 is somewhat unwieldy, and is more easily interpreted when decomposed into a likelihood and a marginal term.

$$p\left(\mathbf{h}^a, \mathbf{h}^b\right) = \frac{\exp\left(\frac{1}{2}\|\phi^a\| + \frac{1}{2}\left\|\phi^b\right\| + \mathbf{h}^a \cdot \mathbf{b}^a + \mathbf{h}^b \cdot \mathbf{b}^b\right)}{\sum_{\mathbf{h}^a, \mathbf{h}^b} \exp\left(\frac{1}{2}\|\phi^a\| + \frac{1}{2}\|\phi^b\| + \mathbf{h}^a \cdot \mathbf{b}^a + \mathbf{h}^b \cdot \mathbf{b}^b\right)}$$
$$p\left(\mathbf{x} \,\middle|\, \mathbf{h}^a, \mathbf{h}^b\right) = \exp\left(-\frac{1}{2}\left\|\mathbf{x} - \phi^a - \phi^b\right\|\right) \tag{4.5}$$

The true joint and log probability of the data can then be written as

$$p\left(\mathbf{x}, \mathbf{h}^a, \mathbf{h}^b\right) = p\left(\mathbf{x} \,\middle|\, \mathbf{h}^a, \mathbf{h}^b\right) p\left(\mathbf{h}^a, \mathbf{h}^b\right)$$
$$\log\, p(\mathcal{D}) = \sum_{\mathbf{x} \in \mathcal{D}} \log \frac{1}{\sqrt{2\pi}} \sum_{\mathbf{h}^a, \mathbf{h}^b} \left[p\left(\mathbf{x} \,\middle|\, \mathbf{h}^a, \mathbf{h}^b\right) p\left(\mathbf{h}^a, \mathbf{h}^b\right) \right]. \tag{4.6}$$

As with a standard RBM, this partition is not a factorisation in the sense of a directed generative model, as the two factors share parameters. Nevertheless the factors have intuitive interpretations. The likelihood term is simply responsible for ensuring the data is reconstructed well, while the marginal

term must apportion probability mass so as to mirror the data distribution. Much like an RBM, in combination these factors can be seen as requiring that the model translates between the data and hidden distributions, using a shared set of parameters for each translation.

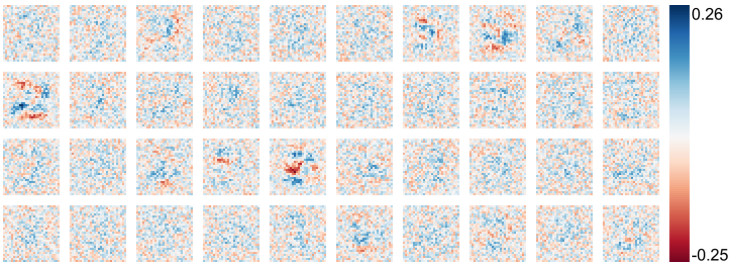
4.2 The Importance of Anti-Hebbian Initialisation

This section briefly details an important point in the Broadnet training algorithm: the hidden states used to initialise the two negative phase Markov chains must come from the reconstruction of each individual GRBM. They cannot be started at random values, or both initialised to the visible state of one network. Doing so all but guarantees that the model will eventually diverge, an example of which is shown in figures 4.2 and 4.3. This is a counterintuitive outcome, and we will show that it is caused by a combination of bad mixing behaviour causing biased anti-Hebbian samples and the Broadnets' sensitivity to noise.

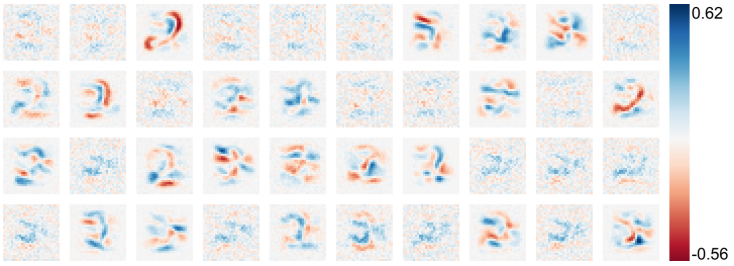
GRBMs can have bad mixing behaviour

Both GRBMs and Broadnets have the potential for bad mixing behaviour during the anti-Hebbian Markov chain. In particular, they risk getting 'stuck' in a state with all hidden units inactive regardless of the joint distribution. This can occur any time the initial visible pattern results in all hidden units being inactive. This behaviour will be discussed in the context of a GRBM with hidden units $\mathbf{h}$, visible units $\mathbf{v}$, and weights W. Superscripts $\mathbf{h}^{(t)}$ and $\mathbf{v}^{(t)}$ are used to denote the iteration of the Markov chain.

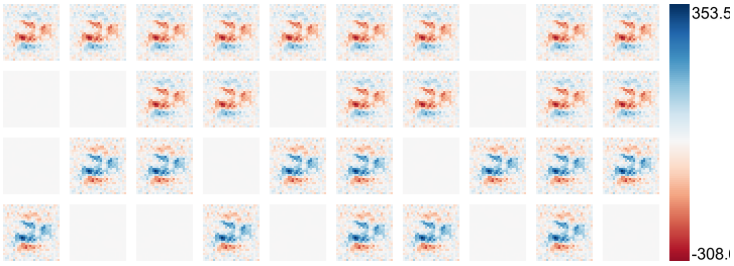
An observation related to bad mixing is that almost all well-trained GRBMs (and any other RBM-model with binary hidden units) have relatively large negative hidden biases. This is necessary to compensate for the effect of the logistic sigmoid function. Suppose a GRBM has all hid-



(a) Epoch 10



(b) Epoch 75



(c) Epoch 120

Figure 4.2: Typical pattern of training when using incorrect anti-Hebbian initialisation. Each figure visualises the weights at a particular epoch, with the first two rows representing W and the last two representing V. By epoch 75 the model has learned a reasonable, though entangled, representation of the data. By epoch 120, the model has diverged. See also figure 4.3.

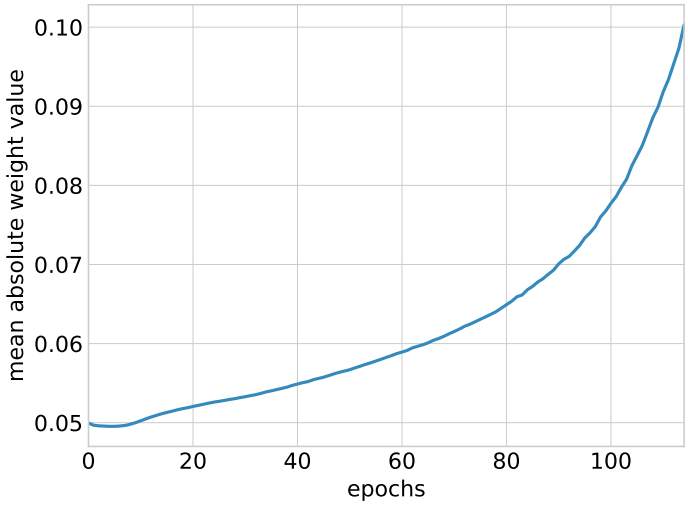


Figure 4.3: Plot of the mean absolute value of a weight in the network from figure 4.2 throughout training. Only the first 115 epochs are shown, as the final epochs diverge rapidly.

den biases set to 0, and a visible pattern $\mathbf{v}$ is presented that is *orthogonal* to W_j, the weights vector associated with unit h_j. The mean-field value of $p\left(h_j \,\middle|\, \mathbf{v}\right)$ is then $\sigma(0) = \frac{1}{2}$. This is clearly undesirable, as a unit that does not contribute to representing $\mathbf{v}$ will be activated half the time. This problem is solved by negative hidden biases.

Consider the anti-Hebbian phase of a GRBM, in which the Markov chain is initialised with a visible pattern $\mathbf{v}^{(0)}$ such that all elements of $W\mathbf{v}^{(0)}$ are non-positive, that is, $\mathbf{v}^{(0)}$ is not well modelled by any of the hidden units. So long as the hidden biases have relatively large negative values, this will lead to all hidden units $\mathbf{h}^{(0)}$ being sampled as inactive with very high probability. Without visible biases, this will lead to the subsequent $\mathbf{v}^{(1)}$ samples having mean-field values of 0. If the variance parameter of the likelihood is relatively small, then the resulting hidden values $\mathbf{h}^{(1)}$ are likely to also be entirely inactive. This cycle continues until a visible sample is lucky enough to overcome the hidden bias and activate at least one hidden unit.

Our experiments indicate that one needs to use $\sigma \geq 5$ for a Markov chain run for 20 iterations to reliably overcome this bad initialisation on GRBMs trained on data in the range $(-1, 1)$ or $(0, 1)$. This value of σ is too large for effective training on data of this magnitude, as the signal-to-noise ratio of the likelihood samples is very low. It should be noted that this issue cannot be solved by changing the magnitude of the data, as the learned weights and biases scale accordingly.

A consequence of this property is that bad initialisations of the anti-Hebbian Markov chain produce biased samples of the joint.

Broadnets are sensitive to noise

Suppose a weight W_{ji} erroneously *increases* in magnitude via the noise introduced by CD. In a standard RBM, the next iteration of learning would correct the error. However in a Broadnet, *both* networks receive a residual and correct the error. This means W_{ji} will decrease in magnitude, and the corresponding weights $V_{\cdot i}$ in the other network will increase in magnitude. The net result is that, once the two networks have re-aligned, all of W_{ji} and $V_{\cdot i}$ have slightly increased in absolute magnitude. In short, the learning dynamics in a two-cause model make parameters sensitive to noise.

This property can be seen experimentally. We construct a simple Broadnet with one hidden unit in each cause, and one visible unit, and train on the simple dataset $\mathcal{D} = [0, 3, 5, 8]$. This network is initialised with the following parameter settings. For demonstration purposes, a bad initialisation is used intentionally.

$$\begin{aligned} W &= 0.5 & V &= -0.5 \\ a &= 0 & b &= 0 \end{aligned} \tag{4.7}$$

Figure 4.4 shows a snapshot of several iterations of training, in which one weight increases in size and both weights move to compensate.

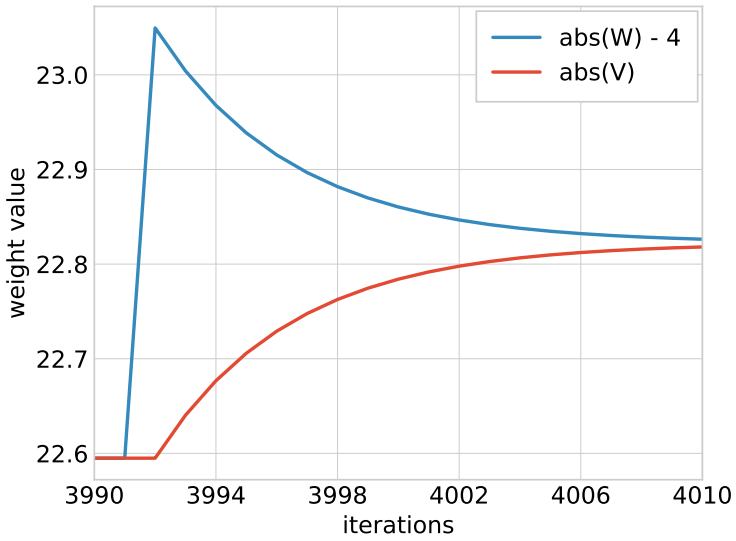


Figure 4.4: Example of a Broadnet's sensitivity to noise. The network is initialised as in equation 4.7, and trained on a simple dataset. The figure shows a snapshot of the weights over 20 iterations part-way through training. The weights have been adjusted for display by taking the absolute value, and adding a constant such that they are (almost) the same value at iteration 3990. After the W weight spikes due to the noise at iteration 3992, the sensitivity of the model to noise can be seen: *both* weights adjust to compensate for the noise, resulting in an overall increase in both weights.

Broadnets diverge with bad anti-Hebbian initialisations

These two properties of Broadnets interact to make learning diverge. Bad anti-Hebbian initialisations bias the samples towards zero, introducing a *consistent error* to the anti-Hebbian update. In a standard GRBM, that error would be corrected for during the next iteration of learning, but the residual-based nature of Broadnet learning prevents it from being entirely corrected. As a result, errors that increase the absolute magnitude of the weights compound throughout learning, eventually leading to divergence.

4.3 The Greedy Inference Problem

The Broadnet model is constructed to disentangle data via inference, that is, the process of computing $p\left(\mathbf{h}^a, \mathbf{h}^b \,\middle|\, \mathbf{x}\right)$. By partitioning the hidden units the flow of information is restricted and the units that represent one coherent cause should be placed in the same network. This emphasis on inference means we expect to see the two causes 'negotiate' responsibility for an input that is not trivial to disentangle. Negotiation should take the form of lively Markov chains during inference, that are active for several steps. However, it appears that inference is almost entirely 'greedy'; each cause mostly ignores the changing residual from the other network, resulting in inference that is generally unchanged after one step. Given that this is true, there is little expectation that Broadnets will achieve proper disentangling.

This section presents some results detailing this unfortunate outcome. We first show that inference provides a relatively good approximation of the posterior, and then show that complex work is not performed by the Markov chain during inference.

4.3.1 Experiments

Training

In order to focus on the inference algorithm, and factor out any issues with learning, the Broadnets used in this section are not trained from random initialisation. Instead, they are constructed by initialising their parameters from two GRBMs, each of which is trained on one of the causes that creates the data, seen in figures 4.1 (a) and (b). We name these *GRBM initialised* Broadnets; they are also used extensively in section 4.4. The constituent GRBMs are trained for 400 epochs of CD-10 learning with the following hyperparameters: a learning rate of 0.01, minibatches of 64 images, and a noise parameter σ of 0.1.

Correctness

First, we must consider the possibility that the inference algorithm used in Broadnets is not functioning as expected, in that samples drawn from the Markov chain do not represent the true posterior well. Asymptotically, this is not a concern as any ergodic Markov chain explores its distribution correctly when run forever [37]. Our Markov chain is ergodic so long as the Broadnet's graph structure remains connected when all connections with zero weight are removed. This is almost certainly the case. In practice, however, a distribution can cause bad mixing behaviour that makes drawing representative samples difficult within reasonable timeframes. Two causes of bad behaviour are when distributions contain regions of high probability that are only connected via paths of very low probability, or when high dimensional spaces have near-constant probability [42]. If the Markov chain used in inference is badly behaved, then the reasonably short chain lengths used may lead to bad gradient estimates.

As theoretical analysis of the mixing behaviour of Markov chains is difficult, we present empirical evidence that the posterior is, in fact, well-approximated by the inference procedure. In the case of a small number of hidden units per cause, it is tractable to calculate the exact posterior for a given input pattern. The true posterior can then be compared with samples from the Markov chain. These tests use six hidden units per cause, yielding 4096 hidden unit configurations.

Figure 4.5 compares the true posterior with the Markov chain's estimate for a pre-trained Broadnet. The blue points represent the log frequency of 10000 samples, with 10 steps of burn-in, and 10 steps between samples. The results are positive: at least in this small case, the inference procedure appears to be correct.

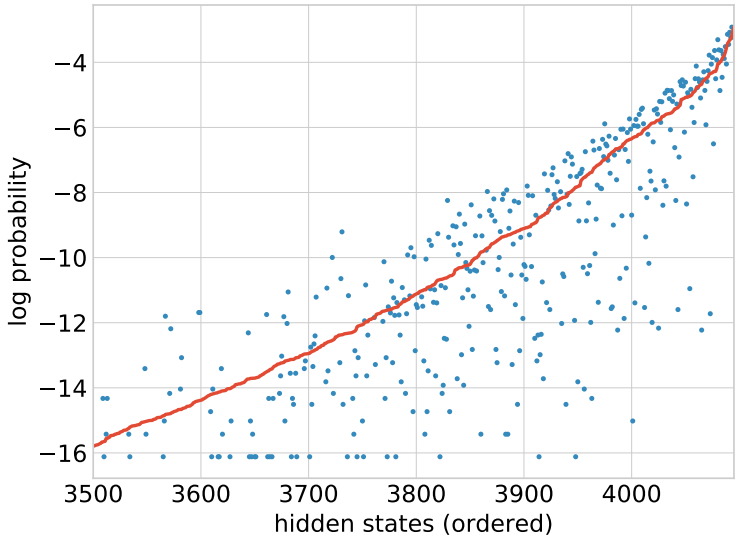


Figure 4.5: Comparison of the true posterior of a GRBM-initialised Broadnet and the sampling procedure. Each x value corresponds to one of 4096 possible hidden states, ordered by their posterior value. Only the most common 596 are shown, as all others are never sampled. The red line shows the true log-posterior values, and the blue points show the log-normalised-frequency of samples from each hidden state.

Time to Convergence

Next, we explore the more meaningful metric of *how long* the dynamics of inference take to settle into a steady state. When running any Markov chain, one can expect an initial period of high activity as the chain mixes, followed by a constant amount of activity once the chain has reached its equilibrium distribution. If negotiation is occurring in the inference Markov chain, we expect to initially see several iterations of abnormally high activity as the two networks trade responsibility for the data before the chain mixes. The activity in a Broadnet's inference chain can be measured by recording the difference in the mean-field hidden activations between successive steps of inference. Letting $\mathbf{h}^{a\,t}$ and $\mathbf{h}^{b\,t}$ represent the *mean-field* hidden activations after iteration t, define

$$\text{activity}_t = \sum_j \left|\mathbf{h}^{a\,t}_j - \mathbf{h}^{a\,t-1}_j\right| + \sum_j \left|\mathbf{h}^{b\,t}_j - \mathbf{h}^{b\,t-1}_j\right|, \tag{4.8}$$

where $|\cdot|$ is the absolute value. At $t = 0$, that is, before the first step of inference, let the activations all be 0.5.

This experiment uses GRBM-initialised Broadnet with 50 hidden units and a noise parameter $\sigma = 1$, trained on the MNIST-like data in figure 4.1. Figure 4.6 plots the value of equation 4.8 across 10 iterations of inference. As expected, the first iteration sees significant change to the activations of both networks. From the second iteration onwards, however, the activations change by a near-constant amount indicating that the chain has mixed. From this, we can draw the conclusion that inference in Broadnets is effectively single-step and negotiation does not occur. This removes the possibility of inference-based negotiation.

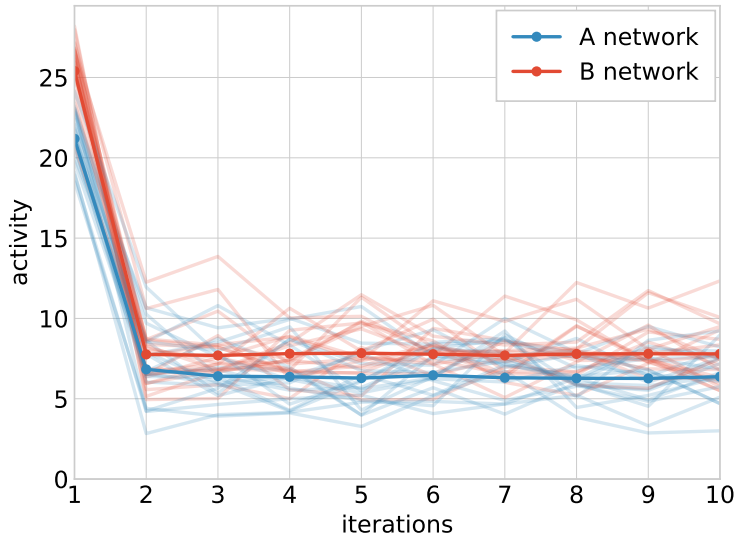


Figure 4.6: Measurement of activity during the first 10 iterations of the inference Markov chain. Activity is measured as in equation 4.8. Lightly coloured lines plot activity for 10 examples from the dataset, and the thick lines with dots plot the mean activity across all elements of the dataset.

Factors Affecting Inference

Inference is strongly affected by two factors: the model's noise parameter and the magnitude of the input patterns. This section will briefly investigate whether these factors play a significant role in the speed of inference.

When the noise parameter σ is large, then the downward step of Gibbs sampling is very noisy, resulting in more variation in the hidden units in the following upward step. Figure 4.7 shows the total activity per iteration (as in figure 4.6) for several settings of σ. The Broadnets used have 50 hidden units per cause, and are trained on data in the range $[-1, 1]$. Changing σ, unsurprisingly, determines the magnitude of the constant activity when at equilibrium, but also has a minor effect on convergence rate. As σ tends to 0 the amount of activity in the second iteration does increase; there is approximately one more unit (out of 100) changing state than would be expected at equilibrium. This does not qualify as negotiation.

In order to reconstruct the data well and thereby satisfy the likelihood term of the joint, the weights matrices W and V must produce reconstructions of a similar magnitude to the input data. Due to this, one would expect larger weights in a network trained on data in the range $[-2, 2]$ than one trained on data in $[-1, 1]$. However, as the size of the weights increases, $\sigma(W\mathbf{x})$ tends towards 0 or 1, decreasing the stochasticity of the hidden samples. Figure 4.8 shows the result of running inference in pre-trained Broadnets built from GRBMs trained on data that has been scaled into different ranges. Each constituent GRBM has 50 hidden units, and σ is set to 0.1 for all runs. As the magnitude of the data increases (green), the mean activity of the network at equilibrium decreases due to the decrease in stochasticity. Additionally, as the magnitude increases the second iteration of inference becomes very slightly more active than would be expected at equilibrium. On average, slightly less than one unit changes state. As with the previous experiment, this effect is minor enough to be ignored.

To summarise, if inference-based negotiation occurs, one would expect to see several iterations of abnormally high activity compared to when the

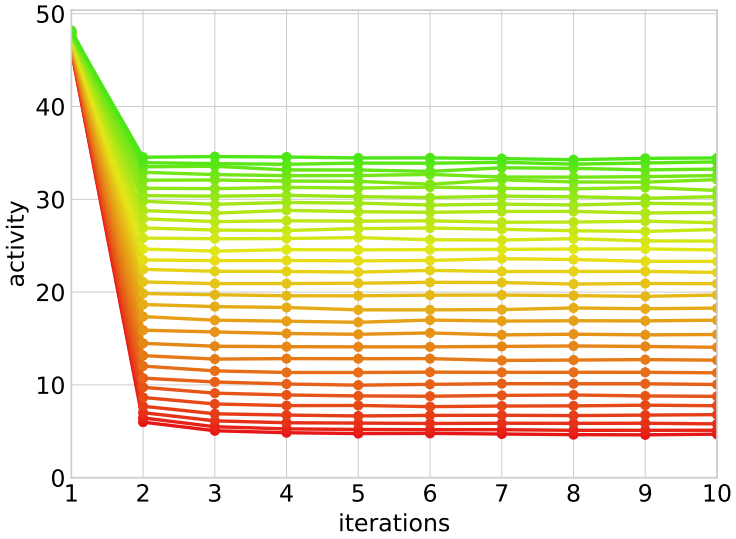


Figure 4.7: Measurement of activity for various σ across 10 iterations of inference. Activity is measured as the mean across the dataset of equation 4.8. Each of the 30 values of σ is represented by a colour, starting with red at $\sigma = 0$ and ending with green at $\sigma = 3$.

Markov chain has reached equilibrium. Empirically, the two major factors affecting the dynamics of inference have, at best, only a minor effect on activity of the Markov chain during its first few iterations. From this we can conclude that neither factor significantly changes the result that inference is largely a one-step process.

4.3.2 Discussion

The motivation for Broadnets rests, in large part, on the need for the two causes to 'negotiate' responsibility for the input, because the model contains no explicit incentive for disentanglement. Contrary to expectations, the Markov chains run during inference appear to perform no more work after the first step of inference than once they have converged. In combination with evidence that inference approximates the posterior fairly accu-

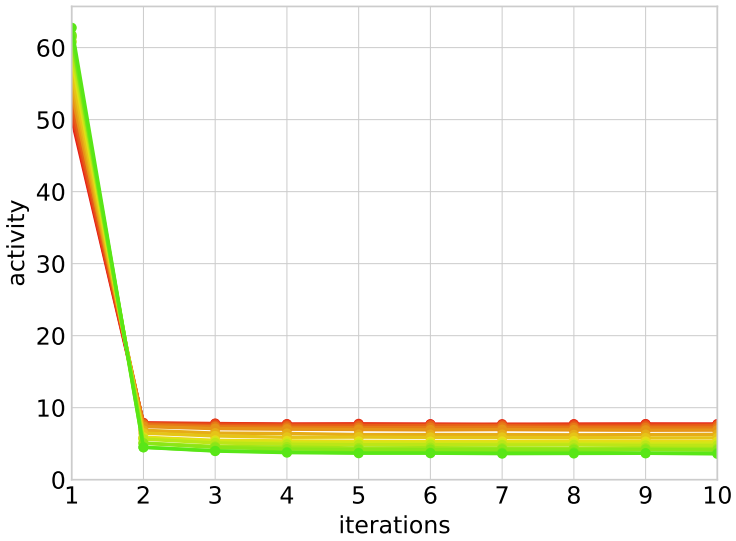


Figure 4.8: Measurement of activity using various data magnitudes across 10 iterations of Broadnet inference. For each plot, inference is performed with a Broadnet that is constructed from two GRBMs trained on a *scaled dataset*. This dataset is initially in the range $[-1, 1]$, and then multiplied by some scaling factor α. Each of the 20 plots uses a value of α indicated by the plot's colour, with red being $\alpha = 0.1$ and green being $\alpha = 5$. Activity is measured as the mean across the dataset of equation 4.8.

rately, this indicates that inference is simple enough to not require negotiation. This leaves us with little expectation of disentangling.

However, the fact that inference in Broadnets does not perform as expected does not imply that disentanglement is impossible. Perhaps even one-step inference in a model with partitioned hidden units provides incentive for disentangling. The following two sections will explore more damning problems of Broadnets.

4.4 The Exponential Optima Problem

The previous two sections have shown the inadequacies of Broadnet models in terms of inference, and of learning with contrastive divergence-based gradient estimates. However neither of these problems is necessarily insurmountable; simple inference does not necessarily imply lack of disentanglement, and alternative gradient estimate algorithms could be used. This section evaluates Broadnets purely as an optimisation problem, where gradient descent is performed on a loss, and draws conclusions about the complexity of the loss surface. This removes the complications added by gradient estimates and approximate samples.

Performing gradient descent limits our analysis to small cases because, like the joint itself, the gradient of the joint is analytic but exponentially costly to compute. For reference, the exact gradient is provided in equation 4.9. In practice it is tractable to learn with this exact gradient in models containing up to five hidden units per cause, which we will use for all models in this section.

$$\frac{\partial}{\partial W^a_{ji}} \log p\left(\mathbf{x}, \mathbf{h}^a, \mathbf{h}^b\right) =$$

$$h^a_j\left(x_i - \phi^b_i\right) - \frac{\sqrt{2\pi}}{Z} \sum_{\mathbf{h}^a,\, \mathbf{h}^b} \exp\left[\frac{1}{2}\|\phi^a\| + \frac{1}{2}\left\|\phi^b\right\| + \mathbf{h}^a \cdot \mathbf{b}^a + \mathbf{h}^b \cdot \mathbf{b}^b\right] \phi^a_i h^a_j \quad (4.9)$$

Training from Random Initialisation

The most simple use of gradient descent is to perform standard training: initialise a Broadnet at random weights, perform gradient descent until convergence, and observe whether a disentangled optimum is found. The log probability over epochs for 30 trial runs, along with example visualisations of the weights, are shown in figure 4.9.

The Broadnets in this experiment were trained for 50 epochs on the data in figure 4.1 (c), and used the following hyperparameters: A learning rate of 0.1, minibatches of size 64, Glorot initialised weights [22], and a σ value

of 1. Other settings were also tested, notably all learning rates within two orders of magnitude, all power-of-two minibatch sizes between 1 and 128, and manually set initial weight ranges of 0.1 and 1. None of these adjustments visibly improved the results, though many were significantly worse.

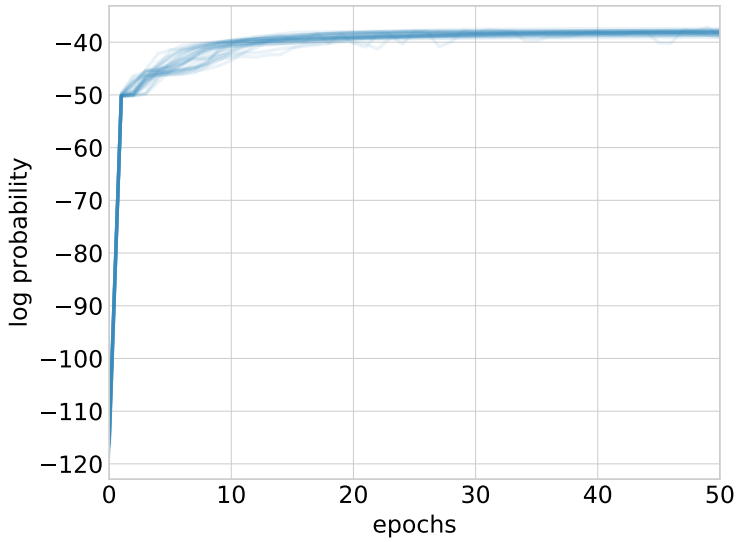
Figure 4.9 presents the log probability throughout training for each trial, as well as visualisations of selected networks. None of the 30 trained Broadnets exhibit any sign of disentangling, in either their weight visualisations or their reconstructions, though most find visibly different optima. Commonly, networks find an optimum in which one network becomes a bias while the other represents all variation in the data. Across the 30 runs, this happened 83% of the time. The training process of the models itself is also consistently fast, mostly reaching convergence after 20 epochs.

The lack of disentanglement under gradient descent is strong indication of the problems of Broadnets, but this is not a conclusive result in and of itself. It is uncertain whether this result generalises to networks with more hidden units. Testing this is impractical, as standard RBMs often use several thousand units to represent MNIST data well. In addition, the reason *why* the gradient descent does not find a disentangled optimum has yet to be answered.

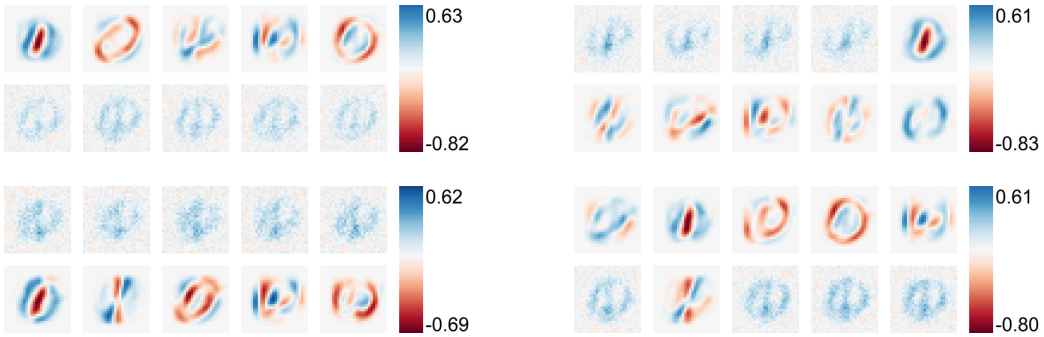
Optimality of the GRBM Initialisation

After the failure of gradient descent to uncover a disentangled optimum, it is reasonable to wonder whether a disentangled, GRBM initialised Broadnet is in fact an optimum at all. To test this, we take the same training setup as in the previous section, but initialise the Broadnet with two GRBMs that have been pre-trained with CD-1 learning on one of the causes, figures 4.1 (a) or (b), each. The results of 30 trial runs are shown in figure 4.10.

Each model's log probability sees a marginal increase throughout training that is consistent with the extra fine-tuning provided by gradient descent on a mostly converged model. The pre-training and post-training figures provided are representative of all runs, in that there is little dis-



(a) Log probability throughout training.



(b) Four sample visualisations of learned parameters.

Figure 4.9: Results of training small Broadnets from random initialisation with gradient descent. Each curve in figure (a) corresponds to the loss over training for one of 30 runs. Figure (b) shows four example visualisations of weight matrices from training. Each network is visualised with two rows of weights, the top row is W and the bottom row is V. The bar on the right of each image indicates the scale of the weights matrices.

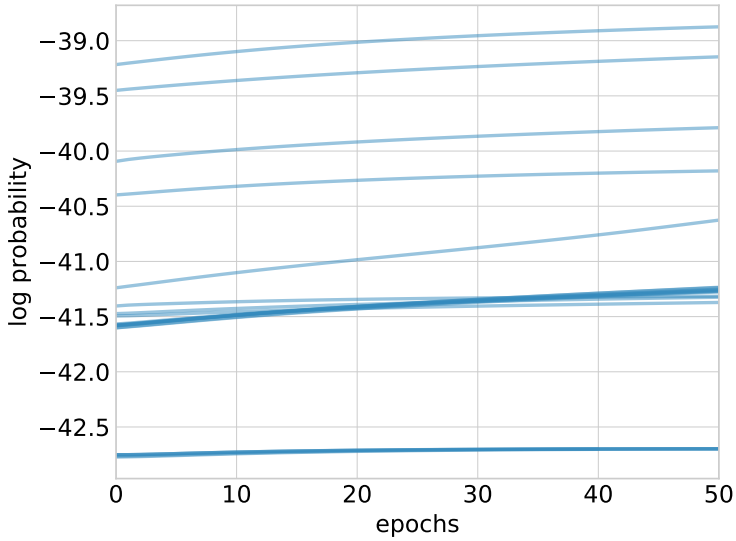
cernible difference after gradient descent. This suggests that a disentangled GRBM initialisation is indeed an optimum for a Broadnet, which leads to a question: if disentangled optima exist, why does gradient descent not find them?

Existence of Many Optima

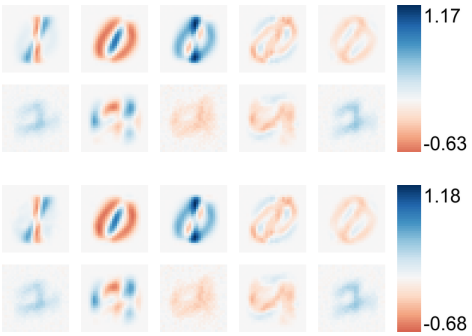
The previous sections have established that disentangled optima exist, but training with gradient descent does not find them. We hypothesise that, for every GRBM initialisation, there exist an exponentially large number of *other optima which are not disentangled*. Combined with the assumption that all disentangled Broadnet optima can be found by training GRBMs individually, this leads to an explanation of the bad convergence behaviour of Broadnets. This section presents a chain of reasoning and experimental evidence to support this hypothesis.

Our argument begins with a GRBM initialisation and then constructs an exponential number of unique, entangled optima from it. The experiments performed are computationally expensive, and so will only be performed on a single GRBM initialisation, the parameters of which are visualised in figure 4.11. This model was arrived at through a tweaked CD-k training procedure for each GRBM, to avoid the tendency for RBM models to underutilise their representational power by making some hidden units a bias. Normally models are trained with enough hidden units that this is not an issue; however, when training with only 10 units total we wish to make full use of the model. Each GRBM was trained for 400 epochs with CD-5. For the first 300 epochs, a learning rate of 0.1 is used, along with momentum with coefficient $\rho = 0.9$. For the final 100 epochs, the learning rate is cut to 0.01 and no momentum is used. This consistently results in all hidden units being used to represent variation in the data.

Starting with a GRBM initialised Broadnet and ignoring the original separation of units into two causes, we consider all possible partitions of



(a) Log probability throughout training.



(b) Example 1, before (top) and after training (bottom).

(c) Example 2, before (top) and after training (bottom).

Figure 4.10: Result of gradient descent on GRBM initialised Broadnets. Figure (a) shows the improvement to log probability for 30 runs of training, with each curve representing a run. Figures (b) and (c) visualise randomly selected examples of the weights of the model before and after training. Networks weights are visualised as in figure 4.9.

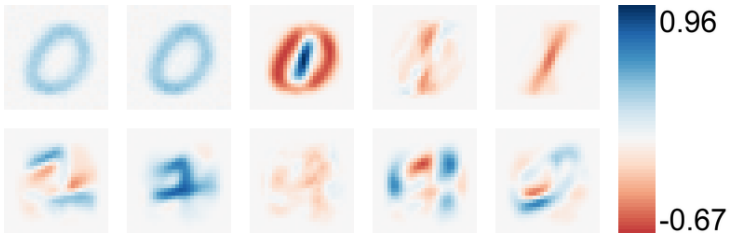


Figure 4.11: Visualisation of the GRBM initialised Broadnet used as the basis for all unit configuration experiments. The top row shows the W matrix and the bottom row the V matrix.

units into two equally-sized causes. More formally, let

$$U = \left[\begin{array}{c} W \\ \hline V \end{array}\right] \qquad \mathbf{c} = \left[\begin{array}{c|c} \mathbf{a} & \mathbf{b} \end{array}\right]. \tag{4.10}$$

Define $\mathcal{I} \subset \{1, \ldots, 2N\}$ as an index set such that $|\mathcal{I}| = N$, and further define $\mathcal{J} = \{1, \ldots, 2N\} \setminus \mathcal{I}$. The units are partitioned into two groups

$$W = \begin{bmatrix} U_{\mathcal{I}_1} \\ \vdots \\ U_{\mathcal{I}_N} \end{bmatrix} \qquad V = \begin{bmatrix} U_{\mathcal{J}_1} \\ \vdots \\ U_{\mathcal{J}_N} \end{bmatrix} \tag{4.11}$$

$$\mathbf{a} = [c_{\mathcal{I}_1} \ldots c_{\mathcal{I}_N}] \qquad \mathbf{b} = [c_{\mathcal{J}_1} \ldots c_{\mathcal{J}_N}],$$

which then form a Broadnet. Each value of $\mathcal{I}$ is called a 'unit configuration'. The experiments in this section will operate on the set of 120 possible unit configurations of the GRBM-initialised Broadnet whose parameters are visualised in figure 4.11. Our interest lies in the optima of the parameter space, and so we will first re-train each unit configuration to convergence. This provides a set of converged models throughout loss space, all based on the original disentangled optima.

Tangentially, a point of interest is that most unit configurations are not themselves optima before training. Changing the configuration does not affect the error term of the log probability under a corresponding configuration of A and B, but can significantly change the marginal term. As a

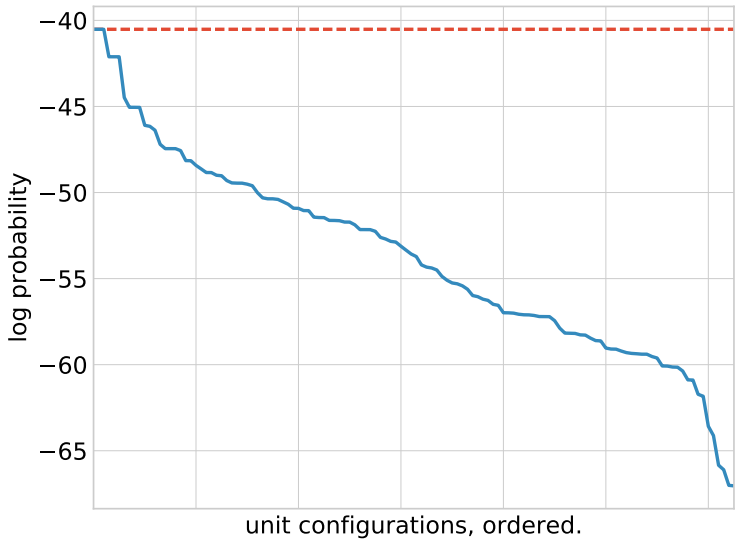


Figure 4.12: Log probability of data under all 120 possible unit configurations of a Broadnet at the disentangled optima, with no training after rearranging the units. Configurations are ordered by log probability. The dashed red line indicates the log probability of the original disentangled optima.

result, the log probability of a unit configuration is almost always much worse than the original configuration. To support this, the log probabilities prior to training are plotted in figure 4.12. Hence, re-training the unit configurations is a necessary step.

The following experiments require a means of comparing the similarity of two models. Two models can be compared while ignoring symmetry by finding the L2 distance between one model and the best permutation of the other model's units. That is, if $W^{(1)}$ and $V^{(1)}$ represent the parameters of one model and $W^{(2)}$ and $V^{(2)}$ the parameters of the other, define

$$\text{diff} = \min_i \left(\left\| \text{perm}_i\left(W^{(1)}\right) - W^{(2)} \right\| \right) + \min_i \left(\left\| \text{perm}_i\left(V^{(1)}\right) - V^{(2)} \right\| \right), \quad (4.12)$$

where $\text{perm}_i(X)$ is the i-th permutation of the rows of X. We call this measure the 'model difference'.

In summary, the following experiments show that, while each of the unit configurations is not immediately optimal, training them to convergence is a process of 'fine-tuning' that does not meaningfully change the weights. This implies that each model is a unique optima. We analyse four properties of the collection of trained units configurations: the quality of each optima, how many models have returned to the original optima, how many of the optima are unique, and how many are disentangled.

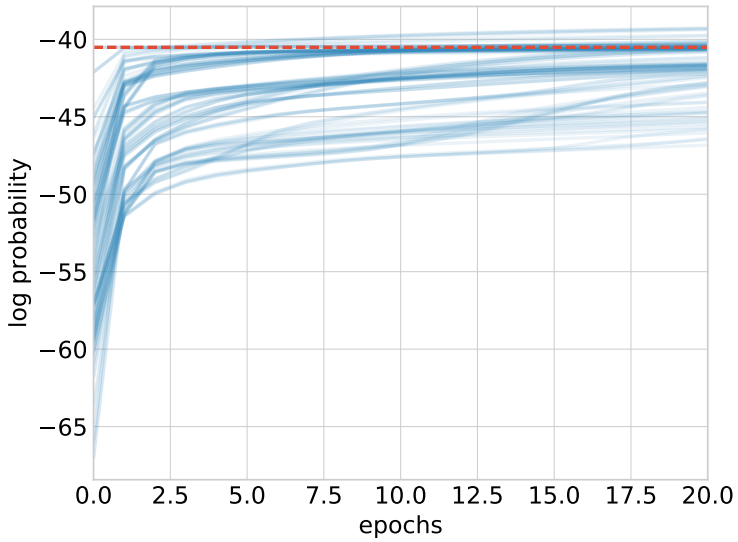
Optima quality. Figure 4.13 shows the log probability of each model throughout 20 epochs of training, as compared to the loss of the disentangled optima. While not identical, all optima have very similar log probability, and notably there are some optima that are *superior* to the original optima. Two of these high quality optima are compared with the original optima in figure 4.13, showing they are not disentangled.

Difference to original optima. Figure 4.14 shows the model difference between the disentangled optima and all unit configurations after training, and provides a comparison between the disentangled optima and the three closest post-training models. It is evident that none of these models return to the disentangled optima.

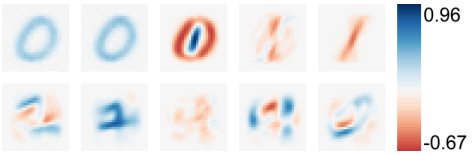
Number of unique optima. To measure the uniqueness of each optima, for each model we compute the minimum model difference between that model and all others. This is shown in figure 4.15, along with examples of the nearest pairs of models. None of the optima are near enough each other to be considered the same, implying that every unit configuration results in a unique model.

Number of disentangled optima. Finally, figure 4.16 presents examples of the parameters of some unit configurations pre- and post-training. While only a handful of the configurations are presented, they exhibit an important property: the post-training model is almost imperceptibly different from the pre-training model, and so are not disentangled. We have manually verified this is the case for all unit configurations.

From these experiments, we can draw the conclusion that the tested



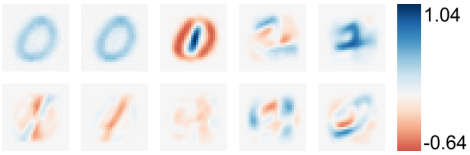
(a) Log probabilities throughout training.



(b) Original GRBM initialisation, as seen in figure 4.11.

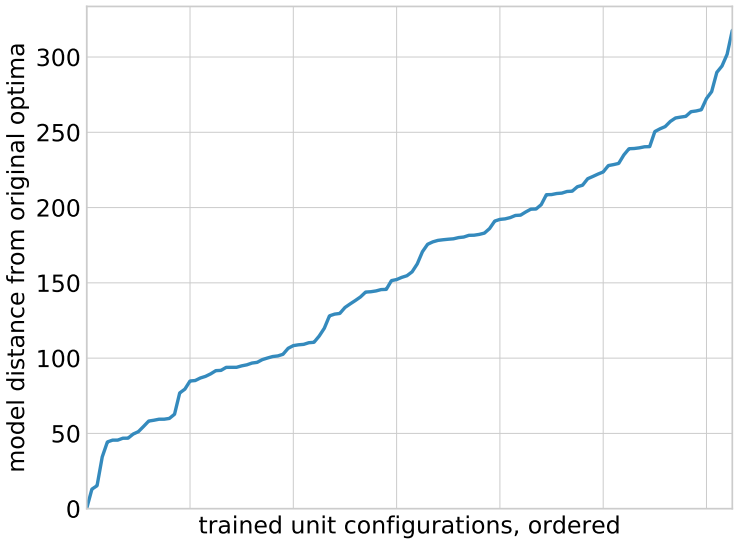
(c) Best unit configuration post-training. Units 4 and 5 of cause A are exchanged with units 3 and 5 of cause B.

(d) Second-best unit configuration post-training. Units 4 and 5 of cause A are exchanged with units 2 and 5 of cause B.

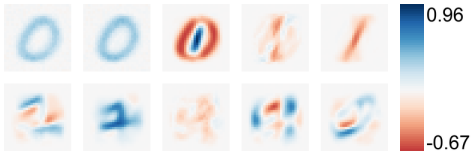


(e) Third-best unit configuration post-training. Units 4 and 5 of cause A are exchanged with units 1 and 2 of cause B.

Figure 4.13: Results of training each of the 120 unit configurations. Figure (a) shows the log probability over time, with each blue line representing a configuration and the dashed red line the log probability under the disentangled configuration. The remaining figures compare the disentangled configuration to the two configurations with highest log probability, showing they are not disentangled.



(a) Model distance to disentangled optima.



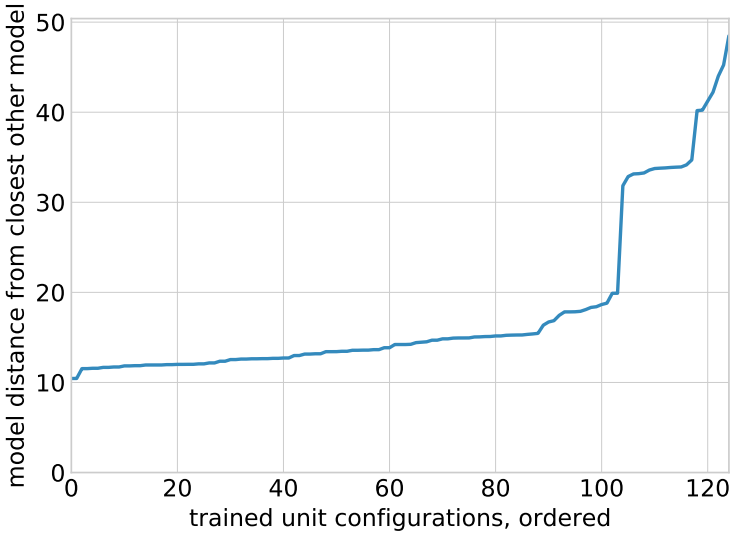
(b) Original disentangled optima, as seen in figure 4.11.

(c) Closest configuration, A unit 5 exchanged with B unit 3.

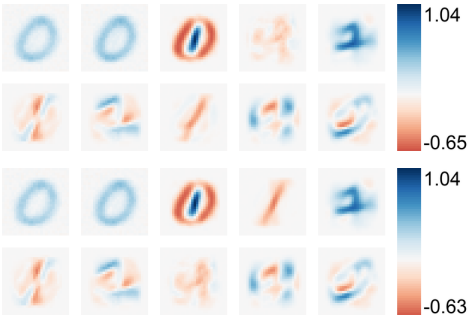
(d) Second closest configuration, A unit 4 exchanged with B unit 3.

(e) Third closest configuration, A unit 1 exchanged with B unit 5.

Figure 4.14: Comparison of disentangled optimum to all unit configurations post training. Figure (a) shows the model difference between the disentangled optima and all unit configurations post-training. The remaining images compare the weights of the disentangled optima to the three nearest configurations post-training.



(a) Distance between each trained unit configuration and its closest other model.



(b) Comparison of the closest pair of trained unit configurations.

(c) Comparison of the second closest pair of trained unit configurations.

Figure 4.15: Comparison of the distance between converged models. Figure (a) shows, for each model (x-axis), the minimum model distance to all other models, ordered by increasing distance. The remaining subfigures compare two of the closest pairs of models, showing they are sufficiently different to be considered different optima.

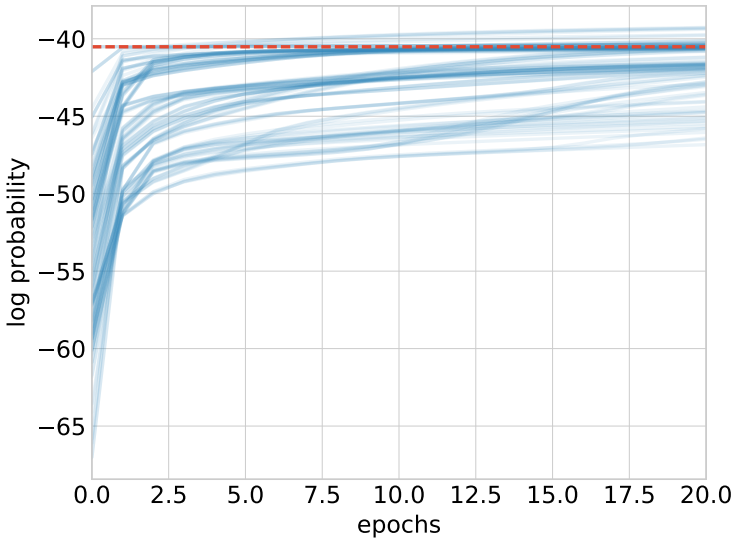


Figure 4.16: Log probability of all unit configurations throughout 20 epochs of training. Each blue line represents a model, and the red dashed line indicates the log probability of the original disentangled optimum. The majority of configurations achieve a similar log probability to the disentangled optimum, and several even find better optima.

GRBM initialisation can be used to construct an exponentially large number of unique optima that are not themselves disentangled, and are of a similar quality to the disentangled optimum. While these experiments have only been performed on one GRBM initialisation, it seems likely that this result generalises to all GRBM initialisations. If that is the case, we reach the conclusion that there exist exponentially more entangled optima than disentangled, GRBM initialised optima.

Finally, we must consider the possibility that there exist disentangled Broadnet optima that are *not* optima for the constituent GRBMs when trained only on one cause. These optima could not be constructed from the process detailed in this section, and so the evidence of an exponential number of associated unique optima would not apply. However, we deem this process unlikely for two reasons. First, it seems intuitively unlikely that, if each GRBM represents only one cause, then that GRBM is not an optima for that cause. Second, we have yet to observe any disentangled optima other than a GRBM initialisation, whether through training or manual configuration. For these reasons we believe this is unlikely.

While the hypothesis in this section has some unresolved assumptions, there is strong evidence for the hypothesis concerning why Broadnets do not disentangle: optimisation is fraught with so many entangled optima that finding a disentangled one is highly unlikely.

4.5 Attractiveness of the Disentangled Optimum

As an addendum, it is possible to gain some intuition for how *attractive* the disentangled optimum is. This is accomplished by observing how much the disentangled optimum can be modified before it converges to a *different* optimum when trained. As previously, start with a GRBM-initialised Broadnet. Instead of swapping units, select a unit j from the A cause and k from the B cause, and interpolate between them. That is, define the in-

terpolated model as

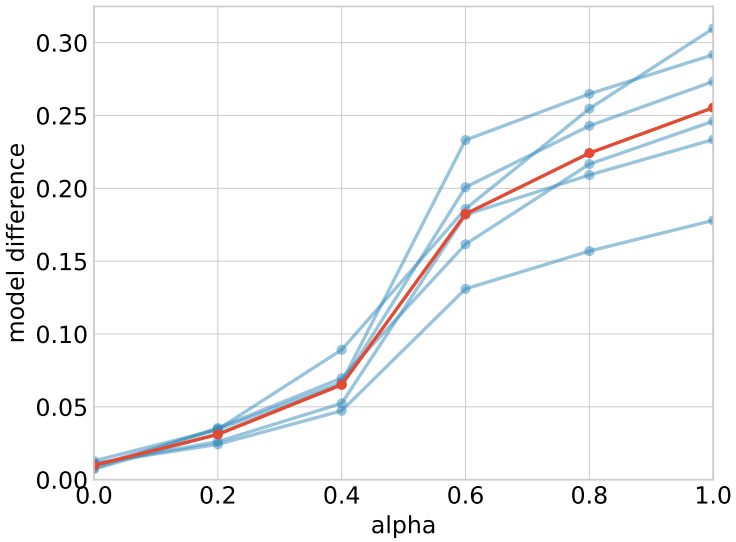
$$W'_j = \alpha W_j + (1-\alpha)V_k \qquad a'_j = \alpha a_j + (1-\alpha)b_k$$
$$V'_k = (1-\alpha)W_j + \alpha V_k \qquad b'_k = (1-\alpha)a_j + \alpha b_k. \tag{4.13}$$

By running gradient descent on models with varying values of α, we can find the point at which the model does not return to the original optima. Difference from the original optima is measured via model difference. Figure 4.17 presents the results of this process for six randomly selected pairs of units, one from each cause. The results show converged models only return to an optima similar to the original model when $\alpha < 0.5$. When $\alpha > 0.5$, the model converges to an optimum near the original model with the selected units entirely swapped. This can be seen in both the plot of model difference, and by inspecting the weights themselves. From this evidence, it seems likely that the disentangled optimum is not particularly attractive and, in particular, is no more attractive than the exponentially large number of other, entangled optima.

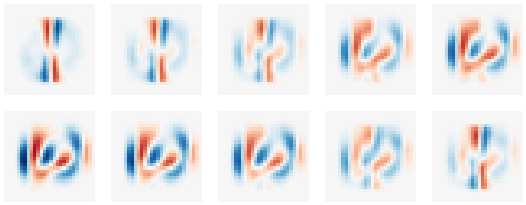
It is clear from figure 4.17 that trained models do not return exactly to the original optimum, or its swapped version. Instead, they converge to an entangled optimum that is a combination of the two. This is evidence that there are many more entangled optima than have been shown to exist in the arguments of this chapter. In fact, it seems likely that a *continuum* of optima exist.

4.6 Discussion

This chapter has investigated several properties of Broadnets in order to understand why they are incapable of disentangling additive data with complex causes. The first section described the importance of good anti-Hebbian initialisations to avoid a pathological case during learning. The second section showed that the inference procedure in GRBM initialised



(a) Model distance from original model over α values, for six pairs of units.



(b) Example of the filters of the selected unit pair as α is increased. Each column shows one value of α.

Figure 4.17: Comparison of the optima found by training interpolated models and the original disentangled optimum from which they were derived. Six randomly chosen pairs of units (one from each cause) are used to create interpolated models with five values for α according to equation 4.13, resulting in 30 models overall. Each model is then trained to convergence. In figure (a), each blue plot shows the model difference between the original model and the interpolated models created from one pair of units. The red plot shows the mean model difference for each value of α. Figure (b) shows an example pair of units, after convergence, for each α value tested.

networks does not perform a complex process of disentangling, which undermines much of the motivation for the Broadnet model. The third section argues that, in small models, the probability surface contains exponentially more entangled optima than disentangled optima, making successful training under gradient descent unachievable.

The generative properties of the Broadnet model are seemingly a perfect fit for the problem of disentangling: causes are independent in the marginal, inference provides a mechanism for meaningful computation, and learning is tractable. In light of this chapter's analysis, however, the prospects for Broadnets are dim: not only do the lacklustre results of inference put the motivation of the model into question, but learning is faced with the task of finding a not particularly attractive disentangled optimum, while avoiding an at least exponentially larger set of false optima of similar quality. To solve these issues would, in effect, reinvent the model.

From one perspective, these results show that Broadnets suffer from a number of practical issues. From another perspective, the magnitude of these problems suggests that Broadnets are simply not a model of independent causes at all. This approach was motivated by the propensity of RBMs to learn non-factorial posterior distributions, that is, distributions where units do not represent independent factors of the data. This could be interpreted as a model of a 'coherent cause'. If several RBMs are placed in parallel, one solution that enables this property is to make the fundamental independencies in the data lie *between* the constituent RBMs. Each RBM can then model a complex, non-factorial cause, and disentangling would result. However, this is only one of many ways to achieve non-factorial posteriors, which indicates the objective optimised by a Broadnet is much weaker than disentangling. We can conclude that Broadnets are perhaps a model of *coherent causes* but not *independent causes*, and are therefore not a model of disentangling.

With this conclusion in hand, we focus our efforts on other approaches to the disentangling problem, which are explored in the coming chapters.

Part II

Disentangling with ICA

Chapter 5

Background on ICA

The task of disentangling the underlying causes of data certainly bears some relation to blind source separation, and its principal algorithm Independent Components Analysis (ICA). Part II of this thesis explores the suitability of ICA to our task, and develops a new algorithm using it. This chapter presents the necessary background on information theory, simulated annealing, and ICA itself; the relationship between ICA and disentangling is then discussed.

5.1 Blind Source Separation

Blind Source Separation (BSS) [4] is a task closely related to disentangling, with origins in signals processing. It aims to recover several latent *sources*, known in our terminology as causes, that combine linearly to form a number of observed *signals*. The observed signals are denoted by $\mathbf{x}$, the unknown sources by $\mathbf{s}$, and the mixing matrix that combines them by A. In a simple setting without time or noise, this relationship is expressed by

$$\mathbf{x} = A\mathbf{s}. \tag{5.1}$$

The task of BSS is to recover A or A^{-1}. To simplify this description we will assume the number of sources and signals are equal, and that the mixing

matrix is invertible, though these requirements are often relaxed in practice.

5.2 Independent Components Analysis

ICA [35] is one of the most widely used algorithms for BSS problems. The basic formulation of ICA rests on a generative model expressed by equation 5.1 [36], but with one extra constraint: the sources are defined as being *statistically independent* given the data. As such, the rows of the mixing matrix are referred to as *independent components*. This restriction makes ICA interesting in the context of disentangling, as it is a model of the independent causes of data. It is important to note that statistical independence accomplishes more than uncorrelatedness, and is non-trivial to calculate. It is for this reason that all variants of the ICA algorithm must operate with statistics of a higher order than the covariance.

The standard derivation of ICA makes several assumptions about the data. The mixing matrix must be invertible, the data contains no noise and has zero mean and, most importantly, that sources are *non-Gaussian* [45]. In practice, some of these assumptions can be relaxed; ICA can be performed with a non-square mixing matrix and still functions well in the presence of noise. The zero mean requirement is also easy to fulfill, as data is almost always whitened prior to running ICA. The assumption that sources are non-Gaussian is the cornerstone of the ICA algorithm. In fact, it can be shown that the mixing matrix can be inferred if and only if this is a case [36]. Finally, note that ICA can only recover the mixing matrix up to scalar multiples and re-ordering of the independent components. The lack of ordering is an important part of the motivation for the coming algorithm, and is discussed further later in the chapter.

ICA can be derived from multiple perspectives, notably either as an algorithm to maximise non-Gaussianity, or the likelihood of a generative model [36]. The following sections summarise the first approach.

5.2.1 Maximising Non-Gaussianity

The key observation behind the ICA algorithm is that independent components should be less Gaussian individually than when they are summed, so long as they are themselves non-Gaussian. Consider an example where two signals $\mathbf{x} = [x_1\ x_2]$ are generated by $\mathbf{x} = A\mathbf{s}$. Let $y = w_1x_1 + w_2x_2$ be some linear combination of those signals. For most settings of $\mathbf{w} = [w_1, w_2]$, y is a linear combination of the sources $\mathbf{s}$, which are independent random variables. This is only not the case when $\mathbf{w}$ is a row of the inverse of the mixing matrix, that is, when y is one of the independent components.

The central limit theorem states that, in most circumstances, the sum of independent random variables is more Gaussian than the variables themselves. This is true in the example; any setting of $\mathbf{w}$ that leads to more than one source in $\mathbf{s}$ having a non-zero contribution to y must result in y being more Gaussian than if only one source made a contribution. Therefore, we can attempt to find settings of $\mathbf{w}$ that maximise the non-Gaussianity of y, and take these as the independent components.

If the independent components of data can be found by maximising non-Gaussianity, then a measure of non-Gaussianity is required. The following section briefly discusses two common choices.

5.2.2 Kurtosis and Negentropy

Perhaps the simplest estimate of non-Gaussianity is kurtosis. Kurtosis is the fourth moment of a distribution, which we will define as

$$\text{kurt}(x) = \mathbb{E}\left\{x^4\right\} - 3\left(\mathbb{E}\left\{x^2\right\}\right)^2, \tag{5.2}$$

where $\mathbb{E}\{\cdot\}$ denotes the expectation of $\cdot$ over the distribution. Intuitively, kurtosis measures the 'heaviness' of the tails of a distribution. A Gaussian has a kurtosis of 0, a positive kurtosis implies heavier tails than a Gaussian, and negative implies lighter tails. Most non-Gaussian distributions have non-zero kurtosis, making it a simple and suitable measure of Gaussianity.

However kurtosis is an imperfect approximation, in particular because it is vulnerable to outliers which can dominate and obscure more subtle signals of non-Gaussianity.

An alternative approach to maximising non-Gaussianity involves considering the entropy of the sources. It is known that, for a given variance, a Gaussian has the maximum entropy among all distributions of that variance. This property leads to a quantity called *negentropy*, denoted J, defined as

$$J = H\left(\mathbf{x}_{\text{gaussian}}\right) - H(\mathbf{x}), \tag{5.3}$$

where H is the entropy (see section 5.3) and $\mathbf{x}_{\text{gaussian}}$ is a Gaussian random variable with the same variance structure as $\mathbf{x}$. The negentropy is 0 if $\mathbf{x}$ follows a Gaussian distribution, and positive otherwise. This means maximising the negentropy of the source values can be used to maximise non-Gaussianity, in the same way as the kurtosis, and therefore find independent components.

However, calculating the entropy is generally intractable and so an approximation must be used. The most common approximation uses two 'non-polynomial moments', generally hyperbolic functions, providing a very cheap though not necessarily accurate estimate of the entropy. In practice though, negentropy is a much more robust measure than kurtosis.

5.2.3 The FastICA Algorithm

Both of these measures of non-Gaussianity are differentiable, meaning gradient ascent can be performed to maximise them. The situation is complicated, however, by the need to *orthogonalise* the rows of the mixing matrix at each step of gradient ascent, so as to ensure that no two sources come to represent the same independent component. This central idea, gradient ascent with orthogonalisation, leads to the FastICA algorithm. There are two major variants of FastICA, known as the *deflationary* and *symmetric* versions.

The deflationary FastICA algorithm iteratively finds individual independent components as follows. For the desired number of components, randomly initialise a unit norm vector $\mathbf{w}$ and perform gradient ascent to maximise non-Gaussianity. After each iteration, orthogonalise the current vector with all previously found independent components using Gram-Schmidt method. In contrast, the symmetric FastICA algorithm finds all independent components at once. This is achieved by gradient ascent on the entire mixing matrix, with a symmetric orthogonalisation step after each iteration.

5.3 Information Theory

Information theory [58] provides a general theory of communication and the information content of messages. This section will describe several of the fundamental quantities in information theory, which will be used throughout this chapter and the following two. All of these quantities concern the information or uncertainty of distributions, and can be defined in terms of either discrete or continuous random variables. Definitions will be given for the continuous case, as it is most applicable to the coming work, but some examples will be given in more intuitive discrete settings.

The *differential entropy*, which we will call the entropy, of a univariate distribution $p(x)$ intuitively measures the uncertainty of that distribution. Entropy is high when probability mass is spread across many values of x, and low when concentrated on a few. For example, if $p(x)$ were a discrete distribution then entropy is maximised when p is uniform and minimised when all probability mass is placed on one state. The entropy is defined as

$$H(x) = -\int p(x) \log p(x)\, dx. \tag{5.4}$$

Several variants of the entropy extend it to other types of probability distributions. The *joint entropy*, denoted $H(x, y)$ or more generally $H(\mathbf{x})$, extends

the concept to multivariate distributions. We will often relax the terminology and refer to the joint entropy simply as the entropy, as multivariate distributions are generally the object of interest. The conditional entropy, denoted $H(x \mid y)$, measures the uncertainty present in a variable when another is known. Definitions of these two quantities can be found in most literature on information theory and so are not presented here. These basic forms of entropy can be manipulated with the chain rule for entropies, showing they behave similarly to log probabilities.

$$H(\mathbf{x}, \mathbf{y}) = H(\mathbf{x} \mid \mathbf{y}) + H(\mathbf{y}) \tag{5.5}$$

The *mutual information* of two groups of variables $\mathbf{x}$ and $\mathbf{y}$ measures the amount of shared information present in the two groups of variables. It is defined as

$$I(\mathbf{x}; \mathbf{y}) = \int \int p(\mathbf{x}, \mathbf{y}) \log \frac{p(\mathbf{x}, \mathbf{y})}{p(\mathbf{x}) p(\mathbf{y})} \, d\mathbf{x} \, d\mathbf{y}. \tag{5.6}$$

More intuitively, it is the "reduction of uncertainty of one variable due to knowledge of the other" [14], and as such can be rewritten as

$$I(\mathbf{x}; \mathbf{y}) = H(\mathbf{x}) - H(\mathbf{x} \mid \mathbf{y}). \tag{5.7}$$

Mutual information is symmetric, non-negative, and zero if and only if $\mathbf{x}$ and $\mathbf{y}$ are statistically independent. This property makes mutual information of significant interest in the context of disentangling, as it provides a concrete measure of statistical independence.

Finally, mutual information can be generalised to *conditional mutual information*, denoted $I(\mathbf{x}; \mathbf{y} \mid \mathbf{z})$ which again describes the shared information present in $\mathbf{x}$ and $\mathbf{y}$ when $\mathbf{z}$ is known. It can be defined and characterised as follows.

$$\begin{aligned} I(\mathbf{x}; \mathbf{y} \mid \mathbf{z}) &= \int \int \int p(\mathbf{x}, \mathbf{y}, \mathbf{z}) \log \frac{p(\mathbf{x}, \mathbf{y} \mid \mathbf{z})}{p(\mathbf{x} \mid \mathbf{z}) p(\mathbf{y} \mid \mathbf{z})} \, d\mathbf{x} \, d\mathbf{y} \, d\mathbf{z} \\ &= H(\mathbf{x} \mid \mathbf{z}) - H(\mathbf{x} \mid \mathbf{y}, \mathbf{z}) \end{aligned} \tag{5.8}$$

5.4 Simulated Annealing

Simulated Annealing [41] is a stochastic optimisation algorithm useful in cases where several minima exist. It aims to minimise some *cost function* $\ell(\mathbf{x})$, where $\mathbf{x}$ are the modifiable variables, as follows. First, initialise $\mathbf{x}$ to a random state. Then, iteratively generate a *proposed state* $\mathbf{x}'$ from the current state $\mathbf{x}$. If the proposed state has a smaller cost than the current state, accept it as the new state. Otherwise, accept it with a probability related to the difference in costs. This stochasticity is intended to make the optimisation procedure less vulnerable to local minima by allowing the algorithm to climb out of minima.

In an effort to find a high quality minima, it is desirable to readily accept inferior proposed states at the start of optimisation, and gradually transition to only accepting states with a lower cost. Ideally, this allows the algorithm to explore the optimisation space during early iterations, and exploit a good minima during later iterations. This is achieved by the addition of a *temperature* variable, denoted T, that modifies the probability of accepting an inferior state. The temperature is initially high and is annealed towards zero over time in what is known as a *temperature schedule*.

These concepts are formalised as follows. Supposing T is the current temperature, a proposed state $\mathbf{x}'$ is accepted from a current state $\mathbf{x}$ with probability

$$\begin{aligned} p\left(\text{accept } \mathbf{x}'\right) &= \begin{cases} 1 & \text{if } \ell(\mathbf{x}') < \ell(\mathbf{x}) \\ \exp \frac{\ell(\mathbf{x}) - \ell(\mathbf{x}')}{T^t} & \text{otherwise} \end{cases} \\ &= \min\left\{1,\ \exp \frac{\ell(\mathbf{x}) - \ell(\mathbf{x}')}{T^t}\right\}. \end{aligned} \tag{5.9}$$

5.5 Applicability to Disentangling

ICA is the traditional algorithm most closely aligned to the task of disentangling; it finds independent components that combine linearly to gener-

(a) Entangled data.

(b) Visualisation of the independent components (rows of the mixing matrix) found by FastICA on the entangled data.

Figure 5.1: Example of running FastICA on linearly entangled data. Each component mainly represents part of one cause, either an upright or rotated digit. However, the components are not grouped according to their associated cause.

ate the data. However ICA cannot be seen as an algorithm for disentangling coherent causes. Each component found by ICA is indeed independent, but a single coherent cause can have several independent components and no distinction is made between components that should be attributed to different causes. Due to this, it is not possible to reconstruct only one of the causes. Figure 5.1 illustrates this by finding independent components of two MNIST digits added together. The result shows no distinction between components attributable to each digit.

We are unaware of any attempts to extend ICA to model independent, coherent causes.

Chapter 6

The Post-Processed ICA Algorithm

This chapter presents the main contribution of part two of the thesis: a new algorithm for disentangling complex causes that leverages ICA. Section 6.1 motivates the algorithm and provides an outline, which is then refined through sections 6.2 and 6.3. Section 6.4 presents experimental results, and section 6.5 concludes the chapter.

6.1 Motivation

This section motivates our simple adaptation of ICA to perform coherent-cause disentangling.

Given our previous work on the problem, a tempting approach to extending ICA to perform disentangling is to partition the sources. This could yield a model such as

$$\mathbf{x} = A\mathbf{s} + B\mathbf{r}, \tag{6.1}$$

where $\mathbf{s}$ and $\mathbf{r}$ are the two groups of sources intended to be causes. Unfortunately, this does not result in a realistic ICA-based algorithm; ICA does not compare sources to one another, but rather computes a quantity measuring

Gaussianity of each component individually. This makes partitioning the sources irrelevant, and leaves little room for disentangling causes *during* the ICA algorithm.

Instead, we consider a different approach. Two observations can be made of ICA's output: the independent components related to one cause are not grouped together, but if the components are truly independent then no component should contain significant contributions from *both* causes at once. Both of these properties can be seen in figure 5.1. It would seem that this second property in fact encompasses most of the work of disentangling; if no component contains contributions from both causes, then all that remains is to divide the components into meaningful groups.

To find a separation of independent components into causes, some measure of the quality of a separation is required. For this, we turn to information theory and consider the *shared information* between the values of the independent components. However this idea runs afoul of theory. If ICA performs perfectly then each of its independent components is indeed *statistically independent*, implying all components have zero mutual information, and making it a useless measure. In practice, however, finding truly statistically independent components of complex data seems a difficult task and one which ICA will solve imperfectly. If this is the case, it stands to reason that making two components statistically independent is easier when they are rightly attributed to different causes, rather than the same cause. This is because the causes interact linearly, while independent components within a cause must attempt to account for the nonlinear interactions present in complex data. Hence, this proposed mechanism relies on ICA performing more poorly in cases where correlations in the data are complex than it does in cases where the correlations are simple. When that is the case, examining the shared information between components should yield information on their membership of complex causes.

For a given separation of components, two simple information theoretic measures of quality present themselves; the shared information between

the two causes, or the shared information between units of the same cause. The first quantity can be measured by the *mutual information* between the two causes, which we would wish to minimise. The second case amounts to computing the *total correlation* [67] between all units within a cause, which we would wish to maximise. Neither of these options is clearly more useful than the other, but total correlation is at least as difficult to compute as mutual information, and has received less focus in the literature. Due to the general difficulty of computing information theoretic quantities on continuous variables (such as independent components), we favour the first option.

This leads to the following concept for an algorithm. Taking the output of ICA, assign membership of each 'independent' component to a cause by minimising mutual information. We view this as an optimisation problem over a set of binary *membership variables*, which we will optimise with simulated annealing. A diagram of the model is provided in figure 6.1.

6.2 Estimating Mutual Information

The algorithm outlined in the previous section involves minimising mutual information, but this is no simple task. In general, mutual information is intractable to compute and, in the case of continuous variables such as ours, requires an integral over all values as seen in equation 5.6. Therefore some approximation is required. Fortunately our optimisation method will only require *measuring* mutual information, rather than calculating its gradient. This section will select a tractable estimate that is suitable for our use case.

The simplest approximation is to discretise the variables into k equal-width bins, and then compute discrete mutual information. The accuracy of this measure depends on a careful setting of k [43, 18], and leads to noticeably worse results if used in the coming algorithm.

Instead, we will make use of the Kraskov MI estimator [43], which is based on density estimation via finding k-nearest neighbours and has min-

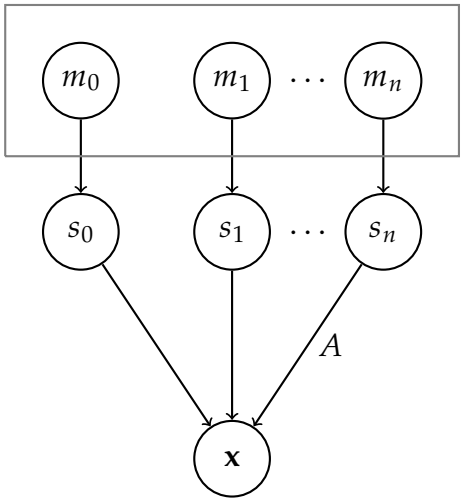


Figure 6.1: Diagram of the modified ICA model. Each m_i is a binary membership variable, each s_i is a source variable, and $\mathbf{x}$ is the observed data vector. Our proposed algorithm only modifies the membership variables, shown boxed. Note that while this has the appearance of a PGM, we do not interpret it as such.

imal bias. We also add very low-amplitude noise to points to prevent incorrect results when several data points have the same coordinates. It should be noted, however, that the Kraskov estimator assumes that the probability density is close to uniform in the neighbourhood of each data point. This assumption is violated in cases where the *joint* distribution over both causes lies on a low-dimensional manifold, an extreme example being when the causes have a functional relationship. Fortunately this is the opposite case to the one in which we require a high-quality estimate; when two causes are nearly disentangled there is almost no relationship between $\mathbf{a}$ and $\mathbf{b}$. Some numerical experiments indicate that the estimator becomes exact for independent distributions [43]. Hence, we can have some confidence that the Kraskov estimator behaves as expected where it counts.

Finally, note that while true mutual information cannot be negative, the Kraskov estimate can be and often is.

6.3 The Postprocessed ICA Algorithm

With an approximation of mutual information now chosen, we derive the algorithm itself. Our strategy, as outlined in section 6.1, is to perform simulated annealing to optimise a set of membership variables. We begin by defining a cost function on membership variables. Let S be a dataset size $\times$ source dimension matrix that represents the recovered source values of all observed vectors in the dataset. That is, if X denotes the dataset, $S = AX$. Denote the binary membership variables by $\mathbf{m}$. The cost is then defined as follows:

$$\ell(\mathbf{m}) = \hat{I}(S\mathbf{m};\ \ S(1-\mathbf{m})), \tag{6.2}$$

where $\hat{I}$ denotes the Kraskov mutual information estimator. In words, the cost function measures the estimated mutual information between the values of independent components assigned to cause A, and those assigned to cause B. The membership variables are not independent, and so simulated annealing is performed by iteratively updating individual variables m_j. This uses the standard simulated annealing update in equation 5.9, and occurs on a temperature schedule, with T^t denoting the temperature on iteration t. Letting $\mathbf{m}^t_{j\leftarrow k}$ denote the value of $\mathbf{m}$ at time t with m_j set to k, each membership variable is updated according to the following probability.

$$\begin{aligned} \alpha &= m_j^{t-1} \\ p\left(m_j^t = 1-\alpha\right) &= \min\left\{1,\ \exp\frac{\ell\left(\mathbf{m}^{t-1}_{j\leftarrow\alpha}\right) - \ell\left(\mathbf{m}^{t-1}_{j\leftarrow 1-\alpha}\right)}{T^t}\right\} \end{aligned} \tag{6.3}$$

We are using a simple decaying temperature schedule as follows. Supposing the algorithm begins at iteration 1,

$$T^t = \left(1 + \left\lfloor \frac{t}{|\mathbf{m}|} \right\rfloor\right)^{-1}. \tag{6.4}$$

The temperature is started at 0.1, rather than a higher value, because mutual information estimate values tend to only have small differences when a single membership variable is changed. Empirically this provides enough stochasticity. In sum, this leads to the algorithm presented in algorithm 4. Importantly, note that this algorithm modifies *only* the membership variables, the independent components themselves remain fixed. If the algorithm leads to a state where all components are attributed to the same cause, then the mutual information between the two causes is undefined. This is an error state of the algorithm, but has never been observed in practice and so we choose to ignore it.

Algorithm 4 The Postprocessed ICA algorithm

Perform ICA on the input, producing a matrix S of source assignments for each input.
Initialise binary vector $\mathbf{m}$ randomly.
Set $T \leftarrow 1$

for iterations $t \leftarrow 1$ to N **do**
    Randomly select membership variable m_j.
    Update the value of m_j according to equation 6.3.
    Update T according to equation 6.4.

6.4 Results

This section tests the Postprocessed ICA algorithm on some complex entangled data based on MNIST. The dataset consists of 5000 composite images, each containing the sum of an upright digit and a rotated digit as seen in figure 6.2. Prior to addition, the two constituent datasets are normalised into the range $[0, 1]$.

We present the results of the algorithm in two cases, which differ only in the number of sources ICA is instructed to find. The first case uses 50

(a) Left cause (b) Right cause (c) Input data

Figure 6.2: Example images from the tested dataset. Figure (c) shows the dataset, which is created by a linear combination of an upright 0, 1, or 2 MNIST digit with a rotated 3, 4, or 5 MNIST digit as in figures (a) and (b).

components and the second 200.

The 50 source test provides our first hint of success at fully-unsupervised disentangling of coherent causes. Figure 6.3 presents an example component separation found by the algorithm, along with data reconstructions using one or both causes. The postprocessing algorithm itself performs well: most components representing upright digits are grouped into cause A, and rotated digits into cause B. The single-cause reconstructions in figure 6.3 also generally show one digit dominating each reconstruction. However, the reconstructions with only 50 independent components are rather low quality. In addition, the single-cause reconstructions have somewhat messy background, seemingly components from the other cause are required to cancel out some artifacts.

These results are consistent across 30 trials of the algorithm. Figure 6.4 plots the mutual information score across iterations for 30 runs of the algorithm, each postprocessing a different run of ICA on the same data. The small mutual information suggests that the postprocessing is consistently successful at disentangling the units, and this has been confirmed by manual checking of the resulting separation.

However, the results change when the number of sources is increased to 200, as shown in figure 6.5. First, ICA itself now finds more visually impressive reconstructions of the input data, which contain sharply defined digit shapes. Unfortunately the postprocessing algorithm largely fails to separate causes, despite achieving a low mutual information estimate. While

(a) Recovered A components

(b) Recovered B components

(c) Data reconstructions. Each row is an input, and the four columns are: input, reconstruction with only A cause components, reconstruction with only B cause components, reconstruction with all components.

Figure 6.3: Results of the postprocessed ICA algorithm on the data from figure 6.2 when set to find 50 sources. The components are well-separated into two causes, but the performance of ICA with only 50 sources is unsatisfactory.

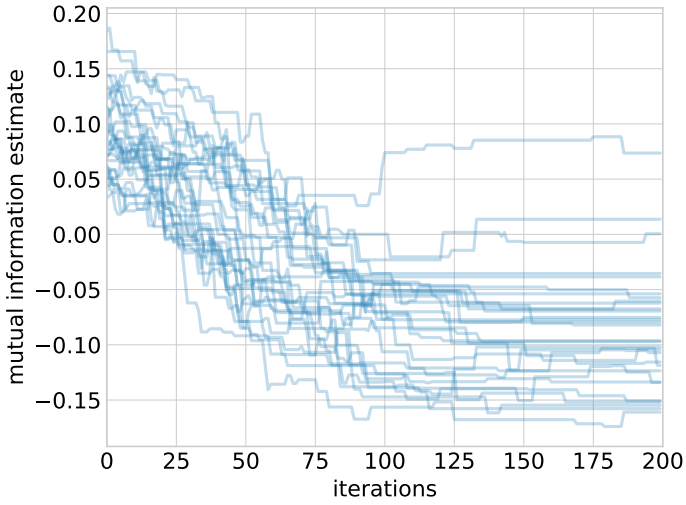
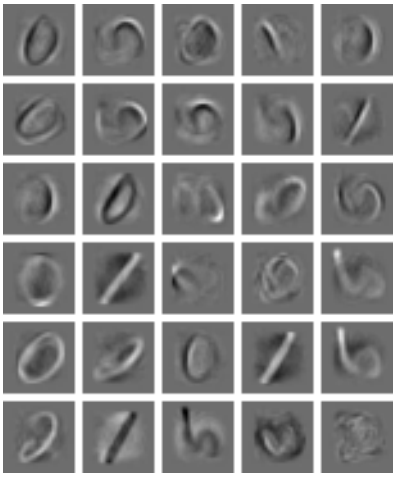


Figure 6.4: Estimated mutual information score across iterations for 30 runs of a PPICA algorithm. Each blue line represents a run. Generally, scores below zero result in visible disentangling.

some single-cause reconstructions in figure 6.5 (c) achieve a level of disentangling, such as in the third and tenth example, there is no clear grouping of independent components into their associated causes.

During the collection of the results in this section, we have found that the performance of the postprocessing algorithm is sensitive to the degree of convergence ICA has achieved. If ICA is run until the desired estimated entropy value is achieved minus a particular tolerance, disentanglement is best achieved when the tolerance is strict enough to ensure no component contains contributions from both causes, but relaxed enough to leave some shared information between components within the same cause. The previous experiments have been performed with a manually optimised tolerance within this range.

(a) Recovered A components



(b) Recovered B components

(c) Data reconstructions. Each row is an input, and the four columns are: input, reconstruction with only A cause components, reconstruction with only B cause components, reconstruction with all components.

Figure 6.5: Results of the postprocessed ICA algorithm on the data from figure 6.2 when set to find 200 sources. Figures (a) and (b) only show 30 randomly selected components attributed to each cause, of the 200 total. Compared to figure 6.3, ICA itself performs better but the components are no longer well separated into their true causes.

6.5 Discussion

The postprocessed ICA algorithm has shown our first success in disentangling complex causes in the linear case; the separation of components in figure 6.3 is compelling. However this success is limited, and disentanglement ability diminishes with the number of sources. This presents a problem for the algorithm as a whole, as ICA's simple linear model appears to require hundreds of components to represent moderately complex data well. Perhaps this is unsurprising, as ICA's success is commonly within the setting of signals processing, where sources and signals are often low dimensional compared to our test case.

Fortunately, the root cause of these shortcomings is clear. The main assumption made by the post-processing algorithm is that making two components statistically independent is noticeably more difficult when they are members of the same cause. When stringent convergence requirements are placed on ICA, all components become more statistically independent, and it is unsurprising that the noticeable distinctions in mutual information disappear. Of more interest is the case of a large number of sources. When many components are available, some components seemingly specialise into modelling specific subcomponents of the data, such as a particular stroke, as can be seen in figure 6.5. When components represent the data at this level of granularity, it is of no surprise that mutual information becomes uninformative.

While the limited ability to group independent components into complex causes is interesting, it does not lead to the general method for disentangling that we are searching for. As such, the next two chapters will explore different approaches based on neural networks.

Part III

Disentangling with Neural Networks

Chapter 7

Background on Neural Networks

Modern neural networks have risen to dominance on most tasks in machine learning [25] and are the forefront of representation learning problems [7, 24]. This motivates the topic of the final part: disentangling with unsupervised neural networks. This chapter provides background and related work.

7.1 Autoencoders

Autoencoders are the most common type of neural network used for unsupervised representation learning problems [7]. An autoencoder is composed of two neural networks: an encoder that maps the input, X, into a code, z, and a decoder that maps z to the output Y which is of the same dimension as X. The goal of the autoencoder is to reconstruct the input as well as possible, that is, make $Y = X$. However the z is generally of much lower dimension than the input, and so forms a 'bottleneck' through which only limited information can pass. Figure 7.1 provides a diagram of this network. In order to reconstruct the input well, the encoder must distill the most important features of the data into a high-quality *representation* in the code layer. Much of the work in modern autoencoders involves manipulating the structure of this representation.

More precisely, let $f(X)$ denote the encoder network and $g(z)$ the decoder network. Generally, f and g are standard dense feedfoward networks with ReLU nonlinearities. The *reconstruction loss* is defined as

$$\ell(X) = \|X - g(f(X))\|. \tag{7.1}$$

A standard autoencoder trains the parameters of f and g by minimising the reconstruction loss via gradient descent.

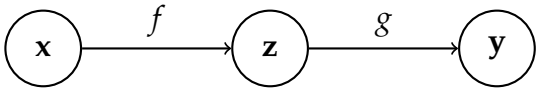


Figure 7.1: Diagram of an autoencoder. The input is **x**, the code **z**, and the output **y**. The function $f(\mathbf{x})$ denotes the encoder, and $g(\mathbf{z})$ the decoder.

The notation in this description, particularly figure 7.1, is common to all work in the coming chapters: **x**, **y**, and **z** denote the input, output, and hidden code, and f and g denote the encoder and decoder.

7.2 Adaptive Gradient Optimisers

In its most simple form, gradient descent entails iteratively moving parameters to minimise a loss. Supposing $W^{(t)}$ is some parameter of a model at time t and ℓ a loss, this can be written as

$$W^{(t)} \leftarrow W^{(t-1)} - \eta \frac{\partial}{\partial W} \ell, \tag{7.2}$$

where η is the learning rate. Generally, ℓ is computed on a minibatch of the data as full-batch learning is unnecessary and expensive. A momentum term is often added to the gradient update, which adds a fraction ρ of the gradient at time t to the gradient at time $t + 1$. This yields a modified

learning algorithm as follows.

$$\begin{aligned} g^{(t)} &= \frac{\partial}{\partial W}\ell + \rho\, g^{(t-1)} \\ W^{(t)} &\leftarrow W^{(t-1)} - \eta\, g^{(t)} \end{aligned} \tag{7.3}$$

In modern neural networks, however, training a model by gradient descent is rarely as simple as in the previous two algorithms. Most modern optimisation techniques fall into the category of *adaptive gradient optimisers*, which can be thought of as standard gradient descent, but with additional factors that dynamically control the learning rate throughout training. Though many such optimisers exist, we will only discuss RMSprop [64] and Adam [38] in detail.

RMSprop aims to provide a degree of normalisation to the gradient, by dividing it by a decaying average of past gradient magnitudes. Letting $\gamma < 1$ be the decay factor, RMSprop is defined as follows.

$$\begin{aligned} g^{(t)} &= \frac{\partial}{\partial W}\ell \\ v^{(t)} &= \left\| g^{(t)} \right\|^2 + \gamma\, n^{(t-1)} \\ W^{(t)} &\leftarrow W^{(t-1)} - \frac{\eta}{\sqrt{v^{(t)}} + \epsilon}\, g^{(t)} \end{aligned} \tag{7.4}$$

Adaptive Moment Estimation [38], known as Adam, can be thought of as RMSprop combined with momentum, where momentum takes the place of the decaying sum of previous gradient magnitudes [17]. This is achieved by estimating both the first and second moments of the gradient, denoted $v^{(t)}$ and $m^{(t)}$. These moment estimates are initially set to zero, resulting in a bias towards zero which, in particular, can adversely affect the early stages of learning [57]. This is compensated for by calculating bias-corrected ver-

sions, denoted $\hat{v}^{(t)}$ and $\hat{m}^{(t)}$. The full algorithm is as follows.

$$
\begin{aligned}
g^{(t)} &= \frac{\partial}{\partial W}\ell \\
m^{(t)} &= \beta_1 m^{(t-1)} + (1-\beta_1)\ g^{(t)} \\
v^{(t)} &= \beta_2 v^{(t)} + (1-\beta_2)\ \left(g^{(t)}\right)^2 \\
\hat{v}^{(t)} &= \frac{v^{(t)}}{1-\beta_2^t} \\
\hat{m}^{(t)} &= \frac{m^{(t)}}{1-\beta_1^t} \\
W^{(t)} &\leftarrow W^{(t)} - \frac{\eta}{\sqrt{\hat{v}^{(t)}}+\epsilon}\hat{m}^{(t)}
\end{aligned}
\tag{7.5}
$$

7.3 Adversarial Autoencoders

Adversarial autoencoders (AAEs) [49] are a method for applying a prior to the code layer of a standard autoencoder. This is achieved by playing an adversarial game between the encoder $f(\mathbf{x})$ and a *discriminator* network $d(\mathbf{z})$. The discriminator receives as input either a value of $\mathbf{z}$, or a sample from the desired prior $p(\mathbf{z})$, and its binary classification task is to distinguish between the two. In addition to minimising reconstruction error, the goal of the encoder is to fool the discriminator network. From the encoder's perspective, this is best achieved by making its code distribution indistinguishable from the prior $p(\mathbf{z})$. The architecture of an AAE is drawn in figure 7.2.

In more detail, the training algorithm of an AAE is as follows. On each iteration, perform a step of standard autoencoder learning: change the encoder and decoder parameters to minimise reconstruction error. Then, construct a *supervised* minibatch of $(\mathbf{z}, y)$ pairs, where half the elements are encodings of elements from the dataset paired with a $y = 1$ label, and half are samples from the prior paired with a $y = 0$ label. Perform a step of training to *minimise* the classification loss (often cross entropy) with the parameters

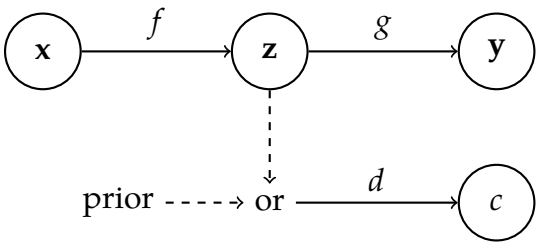


Figure 7.2: Diagram of an AAE. The f and g functions form a standard autoencoder. The discriminator network, denoted d, receives either encodings or samples from a prior, and outputs a classification c that attempts to distinguish between them.

of the *discriminator*. Finally, perform a step of training to *maximise* this error with the parameters of the *encoder*.

The AAE model is of interest in our context due to the concept of an adversarial game played between two networks.

7.4 Generative Adversarial Networks

Generative Adversarial Networks (GANs) [24] are an unsupervised network for synthesising data that is convincingly similar to the dataset. GANs learn through an adversarial game played between a *generator* network and a *discriminator* network. The discriminator receives as input either samples from the dataset, or output from the generator, and attempts to distinguish between them. The goal of the generator is to fool the discriminator. A diagram of this model is provided in figure 7.3. This general approach has yielded impressive results on a variety of tasks [9, 46, 69].

The details of the GAN learning algorithm are not important for our purposes, but two interesting properties are. First, GANs are a second example of an adversarial game played between two networks. Second, standard GANs are generation-only networks; one cannot easily encode data points into a code space.

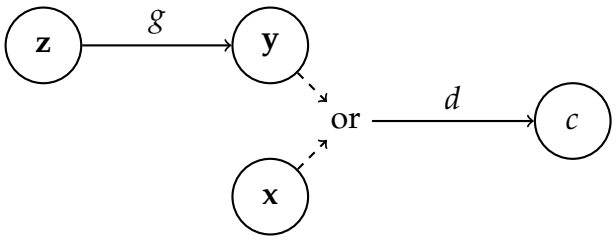


Figure 7.3: Diagram of a GAN. The code is denoted, $\mathbf{z}$, is mapped into the data space by generator g. The discriminator receives either output from the generator, $\mathbf{y}$, or elements of the data $\mathbf{x}$, and outputs a classification c that attempts to distinguish between them.

7.5 Related Work

This section will discuss neural networks that perform tasks related to ours.

Disentanglement problems have received attention from the computer graphics community, in order to produce latent codes that meaningfully represent a single component of a scene, such as lighting or pose [68]. The Deep Convolution Inverse Graphics Network (DC-IGN) [44] is a recent and successful example of disentanglement that is specialised towards disentangling the causes of three-dimensional scenes. It is based on a convolutional variational autoencoder, with a structure imposed on the code layer: before training, some code units are selected to represent components of the scene, such as pose or lighting, and some are left free to represent remaining variation in the image. This is named the 'graphics code'. A careful training procedure is used to disentangle these factors. On each iteration, a minibatch is selected in which only one of the pre-defined factors of variation changes. Code units describing that factor are trained as normal, and all other neurons receive a gradient towards the mean activation across that minibatch. Results on 3D faces and chairs show impressively disentangled and interpretable code variables, that successfully generate images with unseen cause combinations. However, the training regime requires complete and fully labelled data.

A significant body of work on finding disentangled representations in supervised or semi-supervised contexts exists. Much of this work focuses on finding 'invariant features', i.e. factors of variation that are relevant or irrelevant to a supervised signal. For example, the content of a handwritten digit is important for classification, but the handwriting style is irrelevant. Rifai et al. [56] train a semi-supervised convolutional network in which hidden units are split into two groups. Among other terms, the loss function includes an incentive for units in one group to be insensitive to directions of variation in the input for which units in the other group are sensitive. Cheung et al. [12] train semi-supervised variational autoencoders, where the code space is partitioned into a scalar y and a vector $\mathbf{z}$, where y is trained to represent all information relevant to the class label, and $\mathbf{z}$ the remaining variation in the input. A cross-covariance penalty is added to the loss to separate these two groups of variables. More recently, Mathieu et al. [50] construct a generative model that incorporates both generative adversarial networks and variational autoencoders to separate class-relevant and class-irrelevant factors. In summary, the algorithm trains by viewing two inputs with the same class label, and finding factors in one image that are either predictive or not predictive of the class label of the other. This yields a somewhat complex model, involving three uses of an encoder, four uses of a decoder, and two adversarial networks, but achieves impressive results. Kingma et al. [40] perform a similar task with modified variational autoencoders [39], which are supplied with the input data as well as a one-hot vector of class information. Adversarial autoencoders [49] can also be modified for a semi-supervised setting, and can learn to represent class information in a categorical code variable and other information in standard code units.

Chen et al. [11] propose an *unsupervised* approach to modelling factors of variation based on GANs. This approach, named InfoGAN, creates a GAN whose generator receives as input a code vector $\mathbf{z}$, along with a source of incompressible noise. The standard loss function of a GAN is modified

to include a term intended to maximise the mutual information between the code and the output. This is achieved by maximising a lower bound on mutual information, known as Variational Information Maximisation [3]. The effect of this extra term is to make each code unit represent an independent factor of variation in the data. This is the most successful work on unsupervised disentangling in the literature. However, this method is limited by the simplicity of its latent codes; each code unit z_i comes to represent a single factor of variation, and so the representation of a cause is limited to a single scalar (or categorical) value.

To summarise, a large body of work exists related to finding disentangled representations with neural networks. However, almost none of this work attempts to address our particularly difficult version of the task: fully unsupervised learning of high-dimensional, coherent factors of variation.

Chapter 8

Split Autoencoders

This chapter approaches the disentangling problem using autoencoders, with the central idea of encouraging disentangled representations through creating a split in the network architecture. Section 8.1 motivates the use of neural networks in general and, specifically, autoencoders. Section 8.2 develops a set of measures and test cases that will be used to benchmark disentangling models. Section 8.3 proposes the Additive Split Autoencoder model, and section 8.5 tests the model against our test suite. Finally, section 8.6 provides a conclusion and next steps.

8.1 Preliminaries

8.1.1 Motivation

Much of the motivation for Broadnets rested on the careful restriction of the flow of information: units within a cause are dependent and able to form complex distributions, while units between causes are independent and communicate only via 'negotiation'. While Broadnets have proven unsuccessful, this is largely due to issues in the practicalities of learning and does not necessarily discount this motivation as an important part of the disentangling problem. It stands to reason, then, that applying the intu-

ition of restricting the flow of information to a simpler and more successful learning algorithm may yield good results. This is the motivation for our first attempt at a disentangling autoencoder.

The Broadnet algorithm functioned on the idea that the posterior of one cause was highly dependent on the other. That is, knowing one cause as well as the data immediately allows the other cause to be found. This implies that disentangling is primarily an *inference problem*, and suggests an inference algorithm where better estimates of one cause allow for better estimates of the other. However, another perspective is available: the goal of disentangling is to find parameterisations such that each cause can be retrieved from the data with no knowledge of the other. In other words, disentangling is the task of finding parameters where causes are independent in the *posterior*, as well as the prior. This re-frames disentangling as a *learning problem*; finding a parameterisation is difficult but, once known, disentangling causes is simple.

8.1.2 Choice of Neural Network

Our disentangling models are based on standard autoencoders, but other choices for the underlying unsupervised neural network are available. Using Variational and Adversarial Autoencoders would allow for direct sampling of the code space, but would also increase the complexity of the models without contributing to the main objective of disentangling. GANs are a promising candidate, particularly due to their use in the InfoGAN model [11], which is one of the most impressive examples of disentangling in the literature. However GANs have a major drawback: hidden states can be sampled from the prior, but data cannot be encoded into a hidden state. Apart from being a useful feature for a disentangling network to have, this also makes quantitative comparison of different models difficult. GAN models are often compared by Gaussian Parzen window sampling [24] to find a lower bound on the log likelihood of the test set under the model. However, this has been noted to be a "deeply flawed" [49, 62] measure.

Moreover, in this context reconstruction quality is of far less importance than the level of disentangling. If data cannot be encoded into a hidden state, measuring disentanglement is difficult.

8.2 Benchmarking Models of Disentangling

This section discusses the experimental setup used to test the models in this chapter and chapter 9. Our metric of success is the level of disentangling achieved by these models, and other factors such as reconstructions error and training time are of secondary importance. As such, we require a robust setting in which to measure disentanglement.

Datasets

Over the next two chapters, we will benchmark our models on four datasets with increasingly complex methods of entanglement. MNIST has been chosen as the basis for most datasets as, despite being a mostly solved problem in a supervised setting, it is a common test for autoencoders [39, 49, 24, 47, 10, 66] and provides sufficient difficulty for disentangling. The base MNIST images contain entirely binary-valued pixels. Details of each dataset are listed below, and examples from each are shown in figure 8.1.

Simple MNIST. A trivially 'entangled' dataset of 25000 images, size 56×28 pixels. Each image contains two digits from MNIST placed side-by-side. This tests whether models are at all drawn towards disentangling, without the confounding factor of how *difficult* disentangling is.

Additive MNIST. Each image, size 28×28, contains two MNIST digits added together, and divided by 2 to stay within the $[0, 1]$ range. 25000 images total. This tests whether models can disentangle linearly combined causes where some work is required to separate causes.

Occlusive MNIST. Each image, size 28×28, contains two MNIST digits pixel-wise unioned together. 25000 images total. This tests whether models can disentangle non-linearly combined causes, a significantly more difficult task.

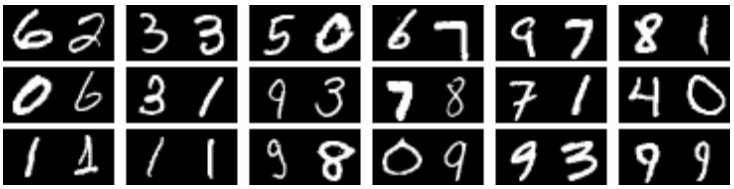
MNIST. The standard MNIST dataset. 50000 images total. This tests whether models can find independent factors of variation in natural data, where disentangling is a difficult task.

In the first three datasets, two MNIST digits are somehow combined. Care has been taken in these cases to ensure that each image in MNIST is used no more than once, resulting in a truly unsupervised problem, but a dataset of only half the size. In all datasets, prior to combination each component image is normalised into a range such that, after combination, images in the dataset are in the range $[0, 1]$.

Measuring Entanglement

In the literature, disentangling is commonly evaluated qualitatively through by-eye observations of whether individual variables can be changed to modify one factor of variation while others remain constant [12, 50, 55, 65, 11, 44]. This is less useful in a context where causes are high-dimensional, and is also less precise than a quantitative measure of disentanglement. Unfortunately, there are no standard quantitative benchmarks for evaluating disentangling [12, 50], so this section will propose some.

To gain a quantitative estimate of disentangling, we reuse the approach proposed in section 6.2 for the postprocessed ICA algorithm. Since the mutual information between causes is zero when they are statistically independent, we will use the Kraskov estimate [43] for mutual information to measure disentangling. Despite being more complex, the advantages over using a simple binning scheme are significant in this context; experiments indicate that the bias of discretised mutual information obscures a



(a) Simple MNIST

(b) Additive MNIST

(c) Occlusive MNIST

(d) MNIST

Figure 8.1: Example images from each dataset used to test disentangling.

surprising amount of detail of the mutual information, and makes comparison between models difficult. Additionally, the number of bins used plays enough of a role that it must be tuned almost on a per-experiment basis to achieve reasonable results. The Kraskov estimator suffers from none of these problems.

8.3 The Additive Split Autoencoder

This section presents a surprisingly simple tweak to a standard autoencoder that allows it to perform some disentangling on additive data.

We construct an Additive Split Autoencoder (ASAE) by placing two standard autoencoders in parallel, each supplied with the same data, and adding their outputs together to form the ASAE's output. More formally, if $\mathbf{x}$ is the input data, and $f^A(\mathbf{x})$ and $f^B(\mathbf{x})$ are standard autoencoders, we define an ASAE network as $\mathbf{y} = f^A(\mathbf{x}) + f^B(\mathbf{x})$. The two code layers are denoted $\mathbf{z}^A$ and $\mathbf{z}^B$, and the code units are referred to as 'causes' in this context. When discussing the output of each autoencoder individually they are denoted $\mathbf{y}^A$ and $\mathbf{y}^B$. See figure 8.2 for a diagram.

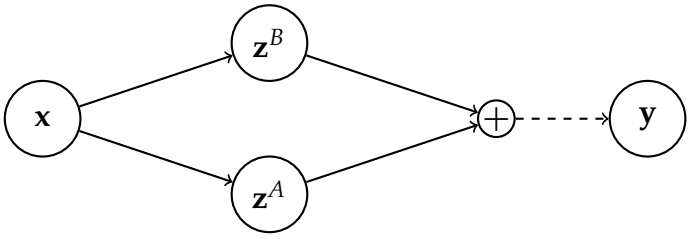


Figure 8.2: Diagram of the ASAE model. Each solid line denotes a neural network, and the small + node denotes element-wise addition. The dashed line indicates the transform has no parameters.

We hypothesise that the split nature of the network will enable disentangling, and that each constituent autoencoder will represent one cause. If this is the case, then causes can be reconstructed separately by taking

the output of f^A or f^B alone. The potential for disentangling in the ASAE model is based on the restricted flow of information between the two networks. In a standard autoencoder, one expects units in the code layer to be combined in a complex, nonlinear way by the decoder to produce output. In an ASAE, if two units are to both represent components of data requiring complex decoding, they must be within the same cause in order to participate in that decoding. In other words, the limitation of units in different causes being unable to 'talk' to each other should incentivise the codes to represent factors of variation that require no discussion. We expect that relatively deep networks will be required for this effect to be significant.

As an ASAE is only a minor change to a standard autoencoder, the options for the training algorithm are identical to a standard autoencoder.

8.4 Training Details

This section briefly outlines the training setup for the ASAE model. For all experiments, and for each split network, our architecture has a single 1000-unit hidden layer between the input and the code, and the code and the output. A ReLU is added after each hidden layer. In the same fashion as variational and adversarial autoencoders [39, 49], we take the sigmoid of the network's output, ensuring it is in the same range as the data. That is, the architecture of each constituent autoencoder is

$$\text{input size} \to 1000 \to \text{relu} \to N \to 1000 \to \text{relu} \to \text{input size} \to \text{sigmoid},$$

where N is the size of the code layer and is varied per experiment. Gradient descent details are as follows.

- RMSprop [64] is used to train the network. This is due to the tendency of vanilla SGD to only learn the mean input image when small code sizes are used.
- A learning rate of 0.001 is used, which is the largest learning rate (to within an order of magnitude) without convergence issues.

- Minibatches of 128 are used, but all multiples of 2 between 8 and 256 yield similar results.

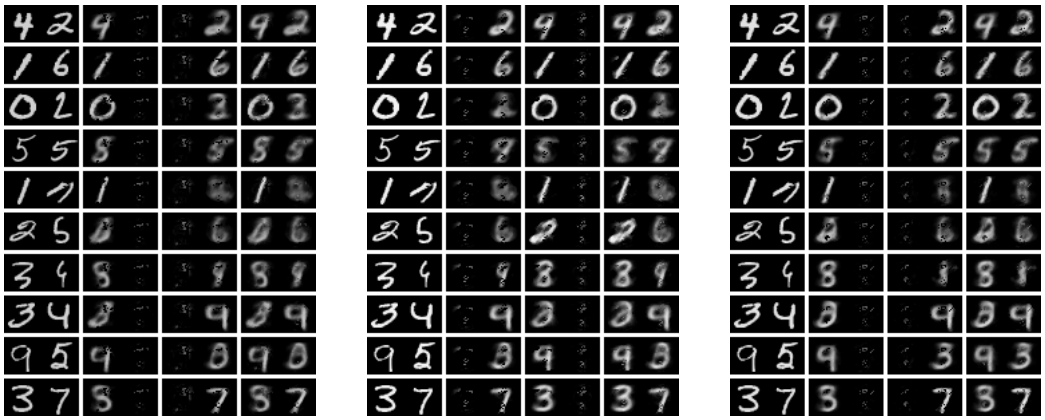
8.5 Results

This section presents results of the ASAE network on the Simple MNIST and Additive MNIST datasets. As the network can only model *additive* causes, we do not test on Occlusive MNIST or standard MNIST. In general, for each dataset we train 30 models, and present example reconstructions from three of them. The images used in the reconstructions are randomly selected from the dataset, but consistent across the three models. We also provide a plot of estimated mutual information throughout training for each run. Reconstruction error plots are not provided, as achieving state-of-the-art reconstructions is not our goal, and reconstruction quality is more easily evaluated by simply looking at reconstructions.

Simple MNIST with Small Causes

It appears that this simple network is successful at disentangling simple data. Figure 8.3 shows examples from three successfully trained ASAE networks using two units per cause. In every example, each network reconstructs one digit acceptably and reconstructs a constant value for the other. The digit reconstructions are imperfect, though they are not significantly worse than those achieved by a standard or variational autoencoder when a 2 unit code layer is used. As the model is so simple, and standard autoencoders certainly do not exhibit this behaviour, it is likely that the restricted flow of information resulting from a split network does indeed encourage the modelling of coherent causes.

Seemingly each network acts as a bias for the other, and the negative of the bias can be seen to some degree behind each well-reconstructed digit. This is perhaps unsurprising, since the bias does not require the shared information that likely binds together units modelling the same cause. This



(a) (b) (c)

Figure 8.3: Examples from three successfully trained two-unit ASAE networks on Simple MNIST. Each row is an example. The four columns are as follows: input images, reconstruction from left network $\mathbf{y}^A$, reconstruction from right network $\mathbf{y}^B$, reconstruction from both networks $\mathbf{y} = \mathbf{y}^A + \mathbf{y}^B$.

also holds with our definition of independent causes, as the two code layers are still representing independent things. While this situation is reminiscent of the diverging positive and negative weights in a Broadnet, no such issues were apparent during training.

However, despite several successful examples being presented in figure 8.3, convergence to a disentangled optimum is not guaranteed. Figure 8.4 shows the inter-cause mutual information estimate across training for 30 runs of an ASAE with the same setup as previously. There is a division between runs; some models converge to a mutual information near 0 after an initial spike while others stay relatively high. This division is exactly mirrored in by-eye analysis of the networks; the model in each run disentangles the data exactly when its mutual information is near 0, though the measure can get as high as perhaps 0.1 without visibly worse results. Hence, despite the success, even in this simple case there is clearly room for improvement. However the agreement between by-eye analysis and the

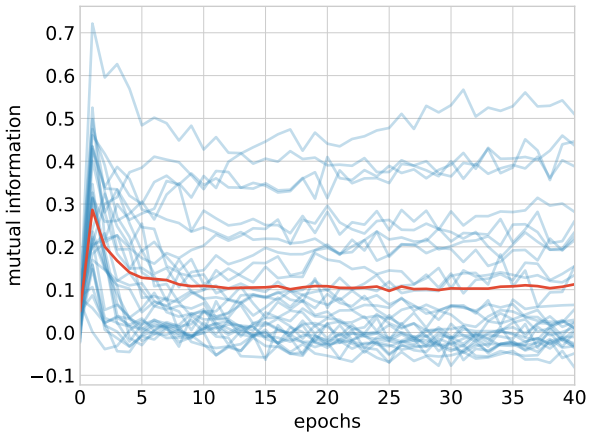


Figure 8.4: Mutual information estimate over epochs for 30 runs of training the two-unit ASAE network on Simple MNIST. Each blue line represents a run, and the red line the mean across all runs. Roughly, a final value of less than 0.2 corresponds to a 'limited success' in terms of visual disentangling, and a final value of less than 0.1 is a full success.

mutual information estimate provides some confidence that the estimate is a good measure of disentanglement.

Given that Broadnets suffered from optima that were at times higher quality despite being entangled, it is worth checking this is not the case in ASAEs. Figure 8.5 shows that mutual information and reconstruction error are positively correlated, that is, the better the disentangling the better the reconstruction.

Simple MNIST with Large Causes

While the results in the previous section are encouraging, our goal is to model *complex* and coherent causes, and using two units per cause only barely meets the definition of 'complex'. To show this, the same experiments as the previous section are performed, but with 10 units per cause.

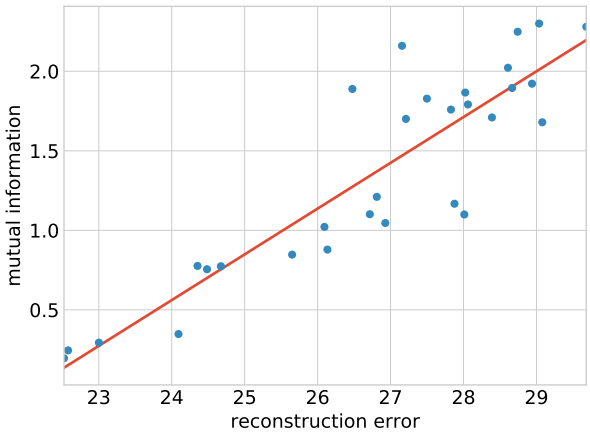


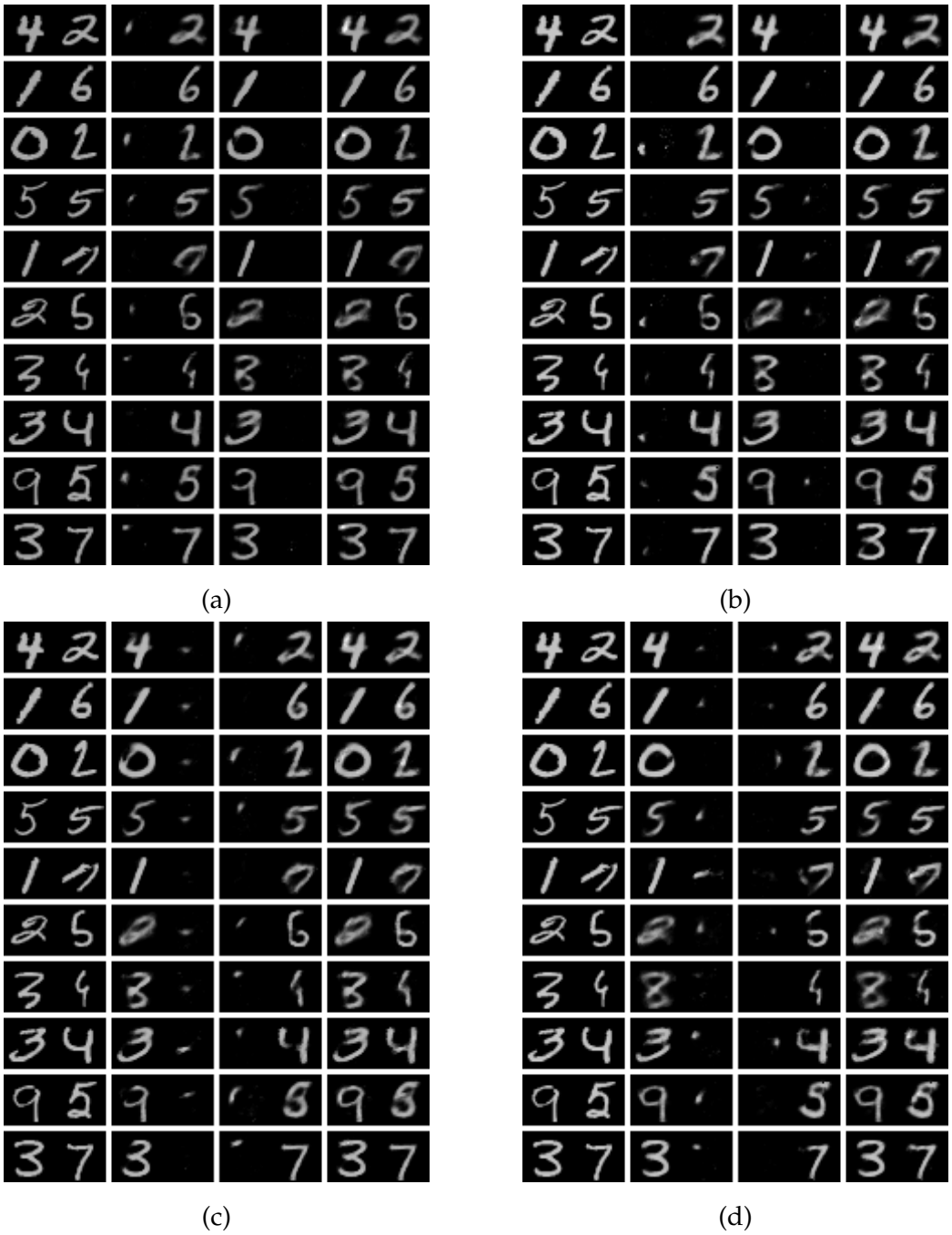
Figure 8.5: Mutual information estimate plotted against reconstruction error for 30 runs of the ASAE model on Simple MNIST. Each blue point is a run, and the regression line is shown in red.

Figure 8.8 shows the mutual information over time, figure 8.6 shows four successful runs, and figure 8.7 shows two failed runs. Unsurprisingly, the reconstructions are of significantly higher quality in all cases, but of the 30 runs, only four converged to optima which could be considered disentangled. In the cases where a entangled optima is found, each cause tends to represent different 'strokes' of each digit.

Additive MNIST

Lastly, we move to the more difficult Additive MNIST dataset. We use 10 units per cause, and perform the standard 30 runs. Figure 8.9 plots the mutual information estimate, and figure 8.10 provides reconstructions from three successful runs. The failure cases are not particularly informative, and so are omitted.

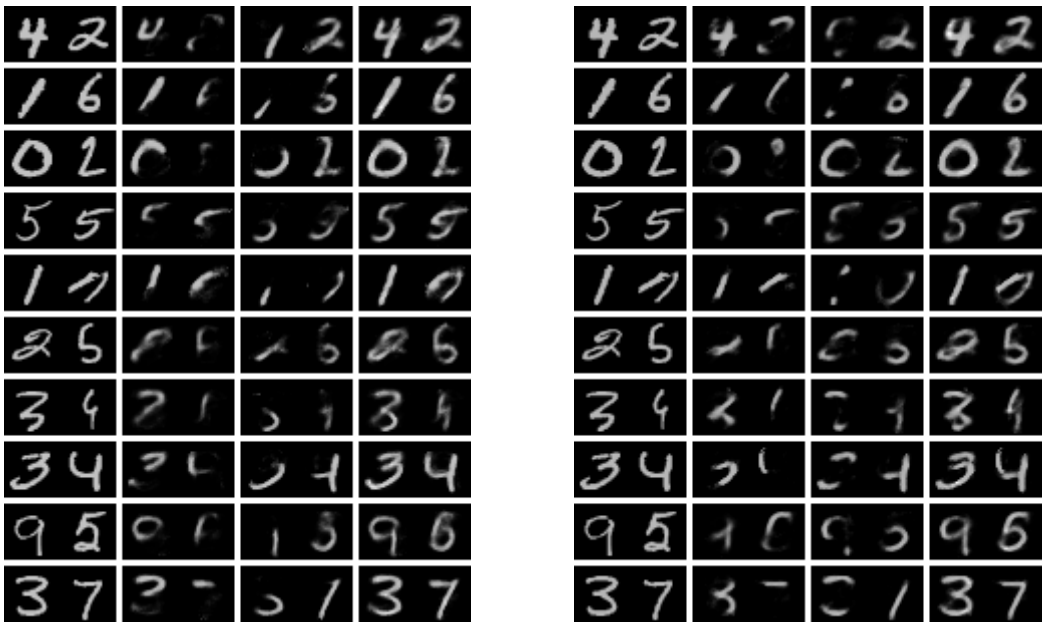
Clearly, the model is capable of finding disentangled representations: though the reconstructions are imperfect, two causes are recognisable as individual digits. Surprisingly, convergence to a disentangled optimum



(a) (b)

(c) (d)

Figure 8.6: Examples reconstructions from four successfully trained 10-unit ASAE networks on Simple MNIST. The runs shown have the four lowest mutual information estimates among all 30 runs. Each image has the same format as in figure 8.3; a row is an example, and the columns are: input, left cause, right cause, both causes.



(a) (b)

Figure 8.7: Example reconstructions from two failed runs of the 10-unit ASAE network on Simple MNIST. The runs shown have the two highest mutual information estimates among all 30 runs. Each image has the same format as in figure 8.3. In these examples, each cause can be seen to model some of the strokes composing each digit. Note that the reconstructions using both causes are visually less convincing than in the successful cases in figure 8.6, in large part because the strokes from different causes match up imperfectly.

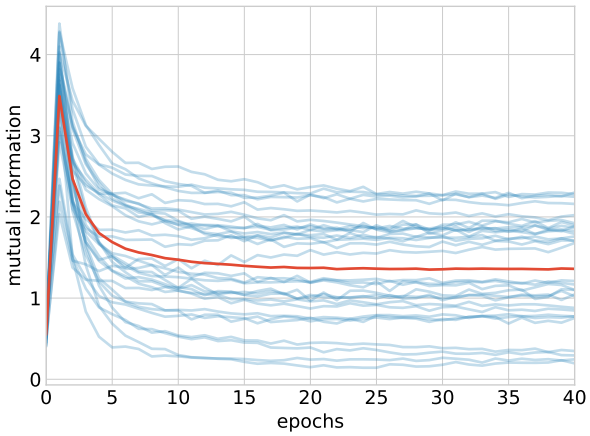


Figure 8.8: Mutual information estimate over epochs for the 30 runs of training the 10-unit ASAE network on Simple MNIST. Each blue line represents a run, and the red line the mean across all runs. The four runs with the lowest value are visually disentangled, the remaining runs exhibit varying degrees of failure.

is seemingly more common in this test case than on the Simple MNIST dataset. In figure 8.9, runs appear to split into two categories, one above the red line, and one below. Roughly, all runs with lower mutual information than the mean show some level of visual disentangling, though of course quality improves as mutual information decreases. It is also notable that no run achieves zero estimated mutual information.

8.6 Discussion

Negative Results

In the process of the previous experiments, many of the tweaks commonly applied to autoencoders were tested and found to, at best, have no effect. The most interesting results are related to encouraging sparsity in either the weights or activations. It seems reasonable that sparsity, particularly in

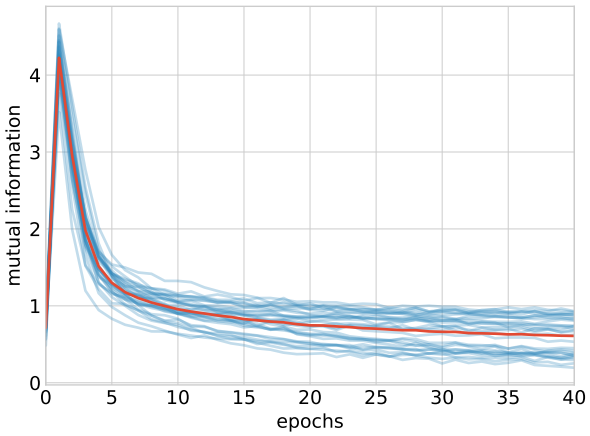


Figure 8.9: Mutual information estimate over epochs for the 30 runs of training the 10-unit ASAE network on Additive MNIST. Each blue line represents a run, and the red line the mean across all runs.

the encoders, would promote better disentangling by providing an explicit incentive for units in the code layer to each represent only small components of the input. Three methods well known for inducing sparsity were tested: L1 regularisation (applied to either weights or activations), denoising autoencoders, and dropout. In particular, using a denoising loss function has an extra appeal in this context: by adding noise, the networks may be encouraged to represent the signal as compactly as possible, perhaps leading to independent causes. Unfortunately none of these sparsity methods improved disentangling, and dropout yielded surprisingly bad results in all cases.

It is unclear *why* sparsity does not improve the results, but some conjecture can be made. As there is no explicit incentive towards disentangling in the loss function, it may be the case that adding noise or regularisation terms obscures the already fragile signal leading to disentangling. The case against dropout is more compelling; by requiring that units need not rely on other units, the representation is made more redundant and, to some extent, the entire concept of a 'complex and coherent cause' is defeated.

(a) (b) (c)

Figure 8.10: Examples reconstructions from three successfully trained 10-unit ASAE networks on Additive MNIST. The three runs shown were randomly selected from the runs with lower than average estimated mutual information. Each image has the same format as in figure 8.3. For display, the middle two columns have had their pixel values doubled, so that the brightness of each column is within the same range.

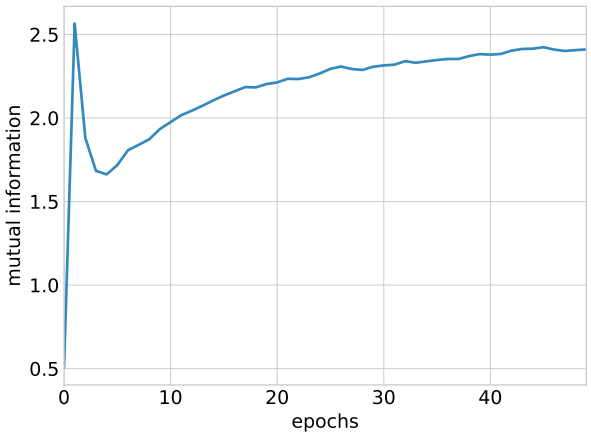


Figure 8.11: Example of a common outcome when training an ASAE network with a low learning rate. The plot shows the mutual information estimate across epochs. After the initial phase of rapid learning, the mutual information slowly rises as the network fine-tunes its optimum.

The Role of Optimisers

The RMSprop optimiser was chosen for these experiments, and used a somewhat higher learning rate than is usual on this data. This combination was arrived at experimentally, which led to two observations: lower learning rates lead to worse disentangling, and both SGD and Adam are consistently less successful than RMSprop.

The case for large learning rates is twofold. First, they prevent the optimiser from performing true gradient descent and add some unpredictability to the network's movements. Much like momentum, this may allow the network to skip over undesirable, entangled optima. Second, high learning rates work to prevent the process of fine-tuning that occurs once a network is trapped in an optima. It appears that this fine-tuning process often works to re-entangle the causes, which can be seen as a slow rise in the mutual information such as in figure 8.11.

The performance of RMSprop is commonly found to be superior to SGD (with or without momentum) [38, 1], particularly in the case of autoencoders [2]. As such the improved performance of RMSprop over SGD is unsurprising. The bad performance of Adam is surprising, however, as not only is it often found to perform at least as well as RMSprop [38, 1] it is also a fairly similar algorithm; it is effectively RMSprop with momentum and some correction terms. There is some evidence that, in autoencoders training on similar data, RMSprop performs a more extensive search across the non-linearities in the loss space than both SGD and Adam [2]. It is unclear whether this is the cause of RMSprop's success.

Conclusion and Next Steps

The ASAE architecture is a minor tweak to a standard autoencoder, and does not explicitly encourage finding disentangled causes. Nevertheless, the model performs full but unreliable disentangling on both easy and difficult datasets with additive causes. This behaviour is certainly not present in a standard autoencoder, and so can be attributed entirely to the restricted flow of information between the two networks. This validates one of the original motivations for the Broadnet: complex causes are best represented when all related units are allowed complex interactions, to the extent that units should self-organise across the 'information gap' between the two causes.

However, there is clearly room for improvement: the model does not always converge to a good optimum, and estimated mutual information never reaches zero on data that is difficult to disentangle. Moreover, the ASAE model is limited to representing causes that interact linearly, giving it a similar scope to ICA and making impossible the representation of the truly complex causes present in natural data.

While the results are somewhat positive in and of themselves, of more interest is the insight they provide into the behaviour of autoencoders on the disentangling problem. On all benchmarks, networks converged to

both entangled and disentangled optima, and that distinction is generally noticeable after the first few epochs of training. It is well known that neural networks are highly non-convex, and can have a number of optima that increases exponentially with depth [15]. In more traditional settings it appears that most optima to which a network converges are sufficiently high-quality that the non-convexity of the loss space is not an issue. The results of this chapter indicates that we are not so fortunate in the case of disentangling. Disentangled optima exist, and are seemingly higher quality, but the loss surface is polluted by many entangled optima.

In summary, the limitations of the ASAE model point to the difficulty of the *optimisation problem*, rather than flaws in the model itself. While the core idea of a split autoencoder is sound, the loss surface becomes sufficiently treacherous on difficult tasks that disentangling is inconsistent. This suggests a direction for further improvements: change the loss to remove the entangled optima from the loss surface. The following chapter explores a method for achieving this.

Chapter 9

Adversarial Networks

This chapter seeks to extend the work on split autoencoders from chapter 8, and progress towards a more powerful neural network-based model capable of consistently disentangling complex causes, even when the causes combine non-linearly. Our approach is to create a model with a loss function that *explicitly encourages disentangling*.

Section 9.1 discusses existing work relevant to constructing a disentangling loss function, and also explores and rejects some alternative options. Section 9.2 presents the new model itself, which is the main contribution of the chapter. Section 9.3 describes our experimental setup, and section 9.4 provides results and analysis on several benchmarks. Finally, section 9.5 discusses some interesting properties of the model and provides a conclusion.

9.1 Rejected Approaches and Related Work

This section discusses several candidate approaches for creating a 'disentangling loss function' that promotes independent causes. We will continue to use the definition of independent causes from section 2.7: causes are independent when they are statistically independent.

Existing Approaches

The scope for a penalty on statistical dependence is large. Most simply, it could take the form of a covariance penalty between causes, which would encourage them to be *uncorrelated*. This has been applied to semi-supervised autoencoders in the form of the XCov penalty [12], in order to separate class-relevant and class-irrelevant signal. The autoencoder model to which the XCov penalty is applied bears some similarity to ours, as the code layer is separated into two blocks. However, as discussed in relation to ICA, *uncorrelated* causes are not necessarily *independent* causes. Because covariance only models pair-wise interactions between units, it does not capture the full dependence structure of the data [36]. Our experiments indicate that having zero covariance is not sufficient for disentangling.

Another approach is to ignore (statistical) moments altogether, and to prevent units in different causes from being sensitive to the same input units. This is roughly the strategy used by Contractive Discriminant Analysis (CDA) [56], part of a pipeline for semi-supervised disentangling using autoencoders. The autoencoder used in CDA is very similar to the ASAE: the encoder and decoder are both split, and combine additively to form the output. However, their model uses tied weights in a one-layer network. The two causes are encouraged to represent different variation in the data by the following regularisation term. Supposing $\mathbf{z}^a$ and $\mathbf{z}^b$ are the causes and $\mathbf{x}$ the input,

$$\mathcal{J} = \sum_{jk} \left(\frac{\partial}{\partial \mathbf{x}} z_j^a \frac{\partial}{\partial \mathbf{x}} z_k^b \right)^2 . \tag{9.1}$$

This term puts a penalty on cases where two code units in different causes are sensitive to variations in the same input elements. The term is promising, because, if no code units are sensitive to the same input units, then the causes certainly represent different things. However, this method has two drawbacks. First, its performance appears to diminish as deeper networks are used. Second, in non-trivial entangling problems there are cases where *both* causes should be sensitive to a particular x_i, but this is strongly

penalised by $\mathcal{J}$. Both of these limitations are ameliorated in the original context of CDA by its use in a larger feature-extraction pipeline.

Maximum Mean Discrepancy

This section explores the use of Maximum Mean Discrepancy (MMD), also known as the two-sample test [26], as a disentangling loss. This is motivated by the fact that MMD can be seen as measuring the difference between *all* statistical moments of two sets of samples. This may enable it to succeed where simple covariance penalties fail.

MMD is a statistical estimate of whether two sets of samples are drawn from the same distribution. Let $\{\mathbf{x}_i\}_N$ and $\{\mathbf{y}_j\}_M$ be two sets of samples, both subsets of some data space $\mathcal{X}$. Let $\mathcal{F}$ be the class of functions $f : X \to \mathcal{R}$. The MMD statistic is:

$$\text{MMD} = \sup_{f \in \mathcal{F}} \left\| \frac{1}{N} \sum_i f(\mathbf{x}_i) - \frac{1}{M} \sum_j f\left(\mathbf{y}_j\right) \right\|. \tag{9.2}$$

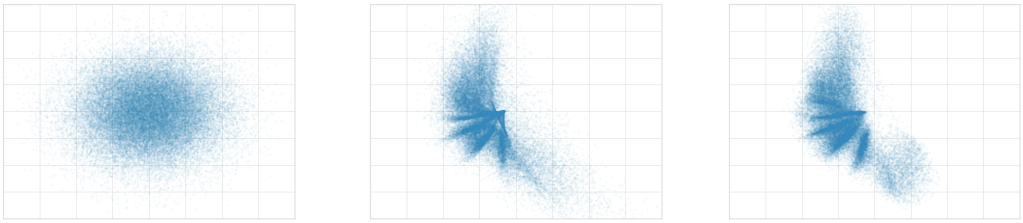
As noted by Li et al. [47], setting f as the identity function results in measuring the difference between sample means, and other settings measure the difference of higher order moments. The kernel trick can be applied to equation 9.2, most commonly with a Gaussian kernel, to find a closed and differentiable form. Under a Taylor expansion, this form yields a term for the difference between every statistical moment of the two sets of samples. As such, minimising MMD can be seen as minimising the difference between every statistical moment of the two sets of samples. MMD is non-negative and zero if and only if the two underlying distributions are equal [27].

Most recently, MMD has seen success in Generative Moment Matching Networks (GMMNs) [47], in which a network is constructed to take input drawn from a simple distribution, such as a Gaussian, and transform it into the data space. This network is trained to minimise MMD between the data and output of the network. If successful, the network is able to use

easily obtainable samples to produce a distribution over data that is indistinguishable from the dataset. Additionally, a similar model can be used to model the code layer of an autoencoder. In this case, an autoencoder is trained normally, and as a post-processing step a GMMN is trained to model the distribution of the code layer.

Given that MMD is a differentiable measure of the difference between two distributions, and considers all moments of those distributions, MMD is a superficially attractive measure of entanglement. This could be applied, for example, by learning to predict one cause's distribution from the other with a GMMN. However this is misguided. The ability or inability to construct one cause's distribution from the other has no bearing on whether *individual pairs* of samples from the two distributions are mutually informative. The view provided by MMD is too general, as it operates at the level of the entire distribution, whereas we require as detailed a view of the entanglement of single data points as possible.

This is exemplified by the fact that almost any distribution can be transformed into almost any other. If a GMMN is trained to construct the distribution of one cause given the other, it can almost always minimise its MMD loss and achieve a convincing reconstruction of the target distribution regardless of both the original distribution, and the level of disentanglement. An example of this ability is shown in figure 9.1. This shows the outcome of a GMMN learning the code distribution of a pre-trained autoencoder. The input to the GMMN is drawn from a spherical Gaussian, which is entirely uninformative of the complex code distribution of the autoencoder. Nevertheless, the GMMN can make a reasonable approximation of the code distribution and achieve a low MMD loss. The implication for the cause prediction setting is that a predictor can minimise its MMD loss regardless of the level of disentanglement.



(a) GMMN input, spherical Gaussian

(b) GMMN output, attempting to match the target

(c) Target code distribution from autoencoder

Figure 9.1: Example of a GMMN approximating an arbitrary code distribution well from spherical Gaussian input. As GMMNs are able to construct arbitrary distributions from uninformative input, MMD is not an effective measure of disentangling.

MeanNN Estimate

An ideal loss function would be one that directly measures the *mutual information* between the causes. The properties of mutual information are appealing; it takes into consideration the entire statistical structure, and is 0 if and only if the causes are statistically independent [36]. However, *optimising* mutual information is a fraught process as it requires a differentiable estimate. The most successful estimates are based either on discretising data or on k-nearest neighbour density estimates, including the Kraskov estimator used for *measuring* mutual information discussed in chapter 8. Neither of these operations are differentiable, making them inapplicable in the context of optimisation.

The MeanNN estimator [18] presents a clever way to find a differentiable mutual information estimate by taking the sum of individually non-differentiable kNN estimators. MeanNN is built on the key observation that a kNN estimator of differential entropy can use any value of k, and so an average over all k also produces an estimator. The main term in a kNN estimator is $\sum_i \log \epsilon_i$, where ϵ_i is the distance between x_i and its k-th

nearest neighbour. Taking the sum over all k simplifies this to the sum of the distances between all pairs of data points. This yields the following estimator.

$$H_{\text{mean}} = \text{const} + C \sum_{i \neq j} \log \left\| x_i - x_j \right\| \tag{9.3}$$

While H_{mean} is less accurate than a normal kNN estimator with a well-chosen k, it has the advantage of being differentiable. Applying an identity, the mutual information of two sets of variables X and Y can be estimated by

$$I_{\text{mean}}(X; Y) = H_{\text{mean}}(X) + H_{\text{mean}}(Y) - H_{\text{mean}}(X, Y). \tag{9.4}$$

MeanNN has in fact been used to approximate mutual information in a clustering setting [19]. Unfortunately, our experiments with MeanNN indicate that disentangling is not successful on anything but the most simple, low-dimensional datasets. We hypothesise that the estimator is simply not accurate enough to tease apart fine dependence structures in high dimensional data. This is consistent with the fact that the estimator appears remarkably smooth (see figure 2 in Faivishevsky and Goldberger [18]), and most of its success has been on relatively low-dimensional datasets.

9.2 Discriminative Disentangling Networks

This section presents the main contribution of this chapter: a neural network model for explicitly disentangling complex causes of unsupervised data, named the Discriminative Disentangling Network (DDN). We first describe the model and training algorithm in section 9.2.1, and then discuss some of the practicalities of training in section 9.2.2.

9.2.1 Description

In short, we draw inspiration from GANs and AAEs, and create an adversarial setting where some networks encode the data into causes, and some

networks try to predict one cause given the other.

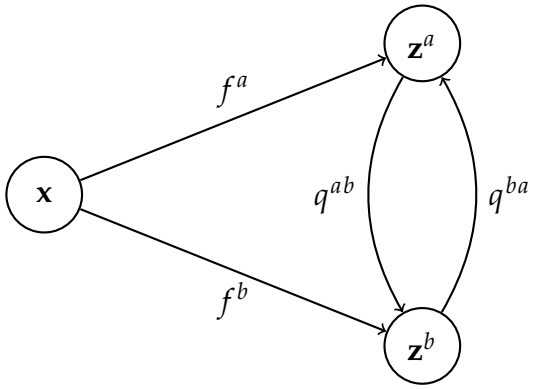


Figure 9.2: The basic DDN model. The input, $\mathbf{x}$, is received by two independent encoders, f^a and f^b, which output codes $\mathbf{z}^a$ and $\mathbf{z}^b$. The two predictive networks, q^{ab} and q^{ba}, take one code as input and try to predict the other.

Figure 9.2 provides a diagram of the model architecture. Begin with two *encoder* networks, denoted f^a and f^b, which receive input $\mathbf{x}$ and produce codes (also known as causes) $\mathbf{z}^a$ and $\mathbf{z}^b$. Beyond receiving the same input, the encoders are entirely independent from one another. Next, add two *predictor* networks, denoted q^{ba} and q^{ab}. Each network receives as input one of the causes, and attempts to predict the value of the other. q^{ba} takes $\mathbf{z}^b$ and produces $\mathbf{z}^a$, while q^{ab} takes $\mathbf{z}^a$ and produces $\mathbf{z}^b$.

Our goal is to train the predictor networks to take advantage of any information about the target cause that is present in the input cause. In tandem, we wish to train the encoder networks to fool their predictors. There is one infallible solution for the encoder networks: make the causes statistically independent. It is our hope that this process will find that solution. To achieve this, we define two *predictive losses* as

$$\begin{aligned} \ell^P_{ba}(\mathbf{x}) &= \left\| \, q^{ba}\left(\mathbf{z}^b\right) - \mathbf{z}^a \, \right\| \\ \ell^P_{ab}(\mathbf{x}) &= \left\| \, q^{ab}\left(\mathbf{z}^a\right) - \mathbf{z}^b \, \right\|, \end{aligned} \tag{9.5}$$

where $\mathbf{z}^a = f^a(\mathbf{x})$ and $\mathbf{z}^b = f^b(\mathbf{x})$. Each predictor network is trained to *minimise* its corresponding loss. Ideally, we would then train each encoder network to *maximise* the loss of its predictor. This would result in two simple min-max games being played between the four networks. However, this is not practical. The predictive losses are squared errors with no upper bound, and the most effective way for an encoder to maximise the error is to make its encodings very large. This invariably results in the networks diverging.

We solve this problem by defining alternative losses to be *minimised* (rather than maximised) by the encoder networks. These are named the *encoding losses* and are simply dot products.

$$\begin{aligned} \ell_a^E(\mathbf{x}) &= \mathbf{z}^a \cdot q^{ba}\left(\mathbf{z}^b\right) \\ \ell_b^E(\mathbf{x}) &= \mathbf{z}^b \cdot q^{ab}\left(\mathbf{z}^a\right). \end{aligned} \tag{9.6}$$

The encoder networks must be matched with their losses carefully: encoders attempt to fool the predictor for which their code is the *output*, not the *input*. That is, the parameters of f^a are used to minimise ℓ_a^E and the parameters of f^b are used to minimise ℓ_b^E. For completeness, equation 9.7 lists each set of parameters and their associated losses.

$$\begin{aligned} \text{parameters of } f^a: &\qquad \ell_a^E = \mathbf{z}^a \cdot q^{ba}\left(\mathbf{z}^b\right) \\ \text{parameters of } f^b: &\qquad \ell_b^E = \mathbf{z}^b \cdot q^{ab}\left(\mathbf{z}^a\right) \\ \text{parameters of } q^{ab}: &\qquad \ell_{ba}^P = \left\| q^{ba}\left(\mathbf{z}^b\right) - \mathbf{z}^a \right\| \\ \text{parameters of } q^{ba}: &\qquad \ell_{ab}^P = \left\| q^{ab}\left(\mathbf{z}^a\right) - \mathbf{z}^b \right\|. \end{aligned} \tag{9.7}$$

It is interesting to note that our model's adversarial setting has a key difference to that of AAEs and GANs: the networks are playing an adversarial game based on *regression*, rather than *classification*. This is the reason why we require different losses for the encoder and predictor, while AAEs and GANs do not. While regression is generally a more difficult task than classification, it is uncertain whether this makes 'solving' the adversarial game more difficult.

An Interpretation of the Encoding Losses

As a digression, the encoding losses have an appealing interpretation. Consider the encoding loss of encoder f^a and, for simplicity, let $\hat{\mathbf{z}}^a = q^{ba}\left(\mathbf{z}^b\right)$. The loss is a dot product, which can be rewritten as a quadratic with two terms removed like so:

$$\ell_a^E \;=\; \hat{\mathbf{z}}^a \cdot \mathbf{z}^a \;=\; -\frac{1}{2}\Big[\, \left\|\hat{\mathbf{z}}^a - \mathbf{z}^a\right\| - \left\|\hat{\mathbf{z}}^a\right\| - \left\|\mathbf{z}^a\right\| \,\Big].$$

During learning, the gradient of ℓ_a^E is taken with respect to parameters W of f^a. The $\|\hat{\mathbf{z}}^a\|$ term depends only on q^{ba} and f^b, not f^a. Hence,

$$\frac{\partial}{\partial W}\ell_a^E \;=\; \frac{1}{2}\frac{\partial}{\partial W}\Big[\, \left\|\mathbf{z}^a\right\| - \left\|\hat{\mathbf{z}}^a - \mathbf{z}^a\right\| \,\Big]. \tag{9.8}$$

As such, the encoding losses can be thought of as having two terms: a squared error to maximise, and a regularisation term to minimise. There is no need for a coefficient λ on the regularisation term, as $\lambda = 1$ is the *only* viable setting. If $\lambda < 1$, then the squared error grows faster than the regularisation and the model still explodes, and if $\lambda < 1$ then the regularisation grows faster than the squared error and the model collapses.

For a loose link to a more principled perspective, recall the view of disentangling as a learning problem, in which we aim to find posteriors encodings that are independent from one another given the data. In this context, we wish to minimise the *conditional mutual information* $I\left(\mathbf{z}^a; \mathbf{z}^b \;\middle|\; \mathbf{x}\right)$, rather than standard mutual information, as the data is known. Conditional mutual information can be rewritten as two entropy terms as follows.

$$I\left(\mathbf{z}^a; \mathbf{z}^b \;\middle|\; \mathbf{x}\right) = H\left(\mathbf{z}^a \;\middle|\; \mathbf{x}\right) - H\left(\mathbf{z}^a \;\middle|\; \mathbf{z}^b,\, \mathbf{x}\right) \tag{9.9}$$

If one supposes that $\|\mathbf{z}^a\|$ is an approximation of the entropy $H\left(\mathbf{z}^a \;\middle|\; \mathbf{x}\right)$, and $\|\hat{\mathbf{z}}^a - \mathbf{z}^a\|$ an approximation to the conditional entropy $H\left(\mathbf{z}^a \;\middle|\; \mathbf{z}^b,\, \mathbf{x}\right)$, then the encoder loss function could be considered an approximation of conditional mutual information.

The encoding losses in equation 9.6 are meant to prevent divergence when minimised, but are still not bounded below; an encoder could produce a code of the opposite sign to its predictor's output, yielding a negative loss, which would allow for divergence. Fortunately the dynamics of the model prevent this situation from ever occurring. So long as the predictive networks have a sensible training setup, a change in the sign of the encoder's output will be quickly followed by a change in sign of the predictor's output. At worst, if the predictor is incapable of any prediction at all, it will output 0. This process is very reliable, and divergence has never been observed.

Learning Without Reconstruction

The DDN model has a surprising feature: it contains no decoder, and its loss does not include a term related to reconstruction error. This the justification behind the 'discriminative' in DDN; in contrast to traditional generative modelling, as well as 'generative' neural networks such as variational autoencoders [39] and GANs, our training algorithm is unrelated to generating or even reconstructing data.

In order to provide useful output, we train decoder networks as a second stage of training after the training of the encoders and predictors is complete. Three decoder networks are trained: one receives both $\mathbf{z}^a$ and $\mathbf{z}^b$ as input, one receives only $\mathbf{z}^a$, and one receives only $\mathbf{z}^b$. The decoders are labelled g^{ab}, g^a, and g^b respectively, and are trained on the gradients of the following reconstruction errors.

$$\begin{aligned} \text{parameters of } g^a: &\qquad \ell_a^R(\mathbf{x}) = \left\| g^a\left(\mathbf{z}^a\right) - \mathbf{x} \right\| \\ \text{parameters of } g^b: &\qquad \ell_b^R(\mathbf{x}) = \left\| g^b\left(\mathbf{z}^b\right) - \mathbf{x} \right\| \\ \text{parameters of } g^{ab}: &\qquad \ell_{ab}^R(\mathbf{x}) = \left\| g^{ab}\left(\mathbf{z}^a, \mathbf{z}^b\right) - \mathbf{x} \right\| \end{aligned} \tag{9.10}$$

This extended network is drawn figure 9.3, and leads to the training procedure in algorithm 5. The significance of this architecture is that no assumptions are made about the method by which causes combine. As a result, the

model is in theory capable of representing causes that interact *non-linearly*, in contrast with the approaches from previous chapters, which sought only to model causes that combine linearly.

The two networks that only receive one cause allow for the causes to be reconstructed independently. However, the idea of reconstructing one cause without the other is not well-formed in general. For example, in the case of faces and lighting, a face cannot be reconstructed without *some* lighting condition. One would hope that, if $\mathbf{z}^a$ contains information about only faces and not lighting, a decoder from $\mathbf{z}^a$ would use some average lighting condition in order to make good reconstructions. Nevertheless, all three decoders are useful for example output.

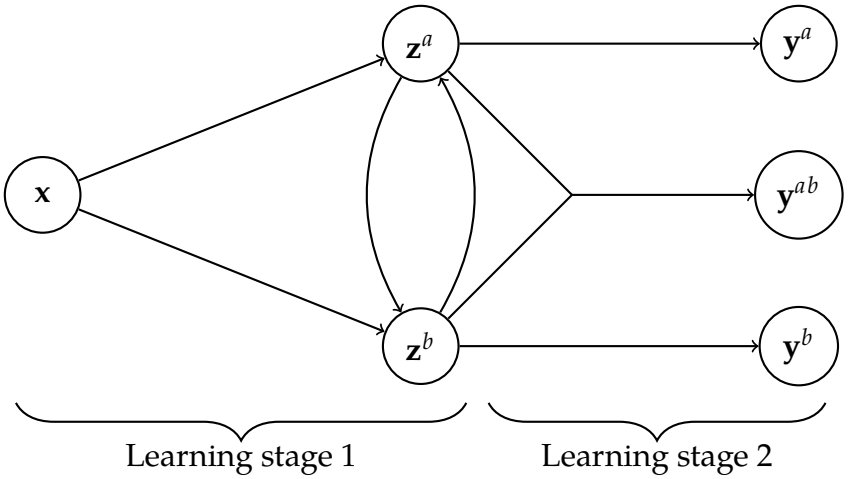


Figure 9.3: Diagram of the extended DDN model. The first stage of the model contains the encoders and predictive networks, and is identical to figure 9.2. The second stage of the model contains three decoders, all attempting to reconstruct the input data. One receives only $\mathbf{z}^a$, one receives only $\mathbf{z}^b$, and one receives both $\mathbf{z}^a$ and $\mathbf{z}^b$. As described in algorithm 5, these two stages are trained separately.

Given the fact that the encoder is not trained on reconstruction error, it is valid to question whether the encodings will contain sufficient informa-

tion about the input for the decoders to make good reconstructions. As a point of comparison, if a randomly initialised autoencoder has only its decoder trained, reconstructions are very bad. Surprisingly, training with the encoding loss actually does result in codes that are useful for reconstruction. To see why, consider various cases for f^a. Upon initialisation, f^a contains significant information about both causes, as can be seen by the high mutual information prior to training in the results in this chapter and the last. f^a cannot ignore the input completely, as that would result in a constant output that is trivially predictable. In order to make itself predictable, the only option available to f^a is to make its output *highly dependent* on some facet of the data ignored by f^b. Consequently, significant information about a cause is available in the encodings of a well-trained DDN. This does not guarantee those encodings will be *easy* for the decoders to use for reconstruction, or that they will model all components of a cause, but some reconstruction ability is likely.

Algorithm 5 The DDN training algorithm

hyperparameters: standard autoencoder hyperparameters.

learning stage 1: encoders and predictors
for some number of epochs **do**
    Generate a minibatch of data.
    Perform the four types of gradient updates listed in equation 9.7, which train the encoders and predictors.

learning stage 2: decoders
for some number of epochs **do**
    Generate a minibatch of data.
    Perform the three types of gradient updates listed in equation 9.10, which train the three decoders.

9.2.2 Practical Considerations

The previous section described the basic DDN model and training algorithm. Unsurprisingly, however, several 'tweaks' can or must be made to achieve good performance. This section details those tweaks.

Setting learning rates with care

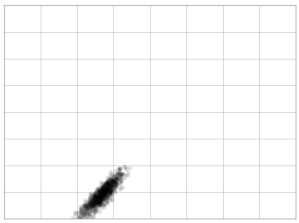
At the core of the learning algorithm are two min-max games played between f^a and q^{ba}, and f^b and q^{ab}. These are games of 'cat and mouse', where the predictor is trying to catch the encoder. It is vital that the predictor learns more quickly than the encoder, otherwise the encoder can simply evade the predictor by moving its encoding around the code space faster than the predictor can catch up. An example of this curious behaviour is shown in figure 9.4. Fortunately, preventing this issue is simple: make the learning rate of the predictor network larger than the learning rate of the encoder network. Experiments indicate that a predictor learning rate roughly twice as large as the encoder learning rate is sufficient, though we tend to increase by an order of magnitude.

End-to-end fine-tuning

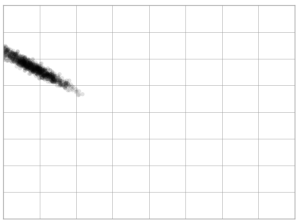
The two-stage training procedure in algorithm 5 has acceptable reconstruction quality, however it is not competitive with the reconstruction quality of well-trained standard autoencoders. To partially bridge this gap, a third 'end-to-end' training stage can be added, in which the encoders, predictors, and decoders are all trained at once. During this stage, we modify the encoding losses as follows. Supposing $g\left(\mathbf{z}^a, \mathbf{z}^b\right)$ is the decoder network,

$$\begin{aligned} \ell_a^E(\mathbf{x}) &= \mathbf{z}^a \cdot q^{ba}\left(\mathbf{z}^b\right) + \alpha \left\| g\left(\mathbf{z}^a, \mathbf{z}^b\right) - \mathbf{x} \right\| \\ \ell_b^E(\mathbf{x}) &= \mathbf{z}^b \cdot q^{ab}\left(\mathbf{z}^a\right) + \alpha \left\| g\left(\mathbf{z}^a, \mathbf{z}^b\right) - \mathbf{x} \right\|. \end{aligned} \tag{9.11}$$

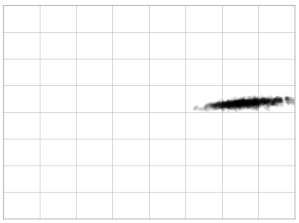
That is, we add in a reconstruction loss term with a small coefficient α. While α has not been carefully optimised, $\alpha = 0.1$ appears to be a good



(a) $\mathbf{z}^a$ at epoch 20

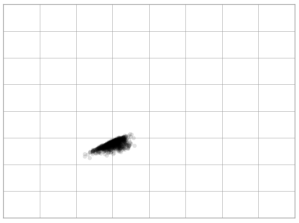


(b) $\mathbf{z}^a$ at epoch 22

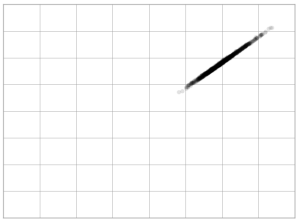


(c) $\mathbf{z}^a$ at epoch 24

(d) prediction of $\mathbf{z}^a$ at epoch 20



(e) prediction of $\mathbf{z}^a$ at epoch 22



(f) prediction of $\mathbf{z}^a$ at epoch 24

Figure 9.4: Visualisation of the outcome of using equal learning rates for the encoders and predictors. A DDN is trained with two code units per cause on Simple MNIST, using SGD with a learning rate of 0.01. Each figure in the top row plots the encoding of all elements of the dataset, and each figure in the bottom row plots the prediction of those encodings using the other cause's encodings. The encodings are two-dimensional as the causes have two units each.

The figures show the encoder evading the predictor by simply moving its encodings around the code space faster than the predictor network can catch up. This is possible because the two networks learn at the same speed, and is solved by ensuring the encoder's learning rate is smaller than the predictor's.

value. It is important to note that using the losses in equation 9.11 during the first stage of training yields *significantly worse results*; the reconstruction term interferes with disentangling, and mostly results in failed training. Therefore, this is best used a fine-tuning step.

Denoising

The speed and quality of disentangling both see a minor but consistent improvement when a denoising component is added to the *predictor* networks. That is, the predictive losses are modified to be:

$$\begin{aligned} \ell^P_{ba}(\mathbf{x}) &= \left\| q^{ba}\left(\mathbf{z}^b + \epsilon\right) - \mathbf{z}^a \right\| \\ \ell^P_{ab}(\mathbf{x}) &= \left\| q^{ab}\left(\mathbf{z}^a + \epsilon\right) - \mathbf{z}^b \right\|, \end{aligned} \tag{9.12}$$

where ϵ is i.i.d. noise. We use spherical Gaussian noise with $\sigma = 0.1$.

Algorithm 6 The extended DDN training algorithm, including tweaks

hyperparameters: standard autoencoder hyperparameters, end-to-end tuning coefficient α, and denoising parameter σ.

Perform the first two stages of learning as described in algorithm 5.

learning stage 3: end-to-end fine-tuning
for some number of epochs **do**
  Generate a minibatch of data.
  Perform gradient updates for the *encoders* as listed in 9.11.
  Perform gradient updates for the *decoders* as listed in 9.10.
  Perform gradient updates for the *predictors* as listed in 9.12.

9.3 Experimental Setup

This section details the training procedure used for all experiments in section 9.4. The network architecture varies slightly between experiments, in

particular the number of hiddens units per cause, denoted N, and the number of layers in each network, denoted by L with a subscript. With these two variables in mind, the two encoders have the architecture:

$$\text{input dimension} \to \underbrace{1000 \to \text{relu} \to \ldots \to 1000 \to \text{relu}}_{L_e \text{ layers total}} \to N. \tag{9.13}$$

The encoder has L total hidden layers, each followed by a ReLU. Each predictor network has the form:

$$N \to \underbrace{1000 \to \text{relu} \to \ldots \to 1000 \to \text{relu}}_{L_p \text{ layers total}} \to N. \tag{9.14}$$

Finally, each of the three decoder networks has the form:

$$N \text{ or } 2N \to \underbrace{1000 \to \text{relu} \to \ldots \to 1000 \to \text{relu}}_{L_d \text{ layers total}} \to \text{input dim} \to \text{sigmoid}. \tag{9.15}$$

In the case where the decoder has both causes as input, the input size is $2N$, otherwise it is N. As with the experiments on the ASAE model from chapter 8 we sigmoid the outputs, in a similar fashion to variational autoencoders and adversarial autoencoders, as our data is within the range $[0, 1]$. In sum, we will describe a particular architecture by specifying N, L_e, L_p, and L_d.

All networks are trained according to algorithm 6 unless specified. The RMSprop optimiser is used for all networks during all stages of learning. The encoders are trained with learning rate $1e-4$ and momentum 0.9, and the predictors with a learning rate $1e-3$ and momentum 0.9. The decoders are trained with same hyperparameters as the predictors. The denoising parameter σ is set to 0.1 and, during the fine-tuning stage, α is also set to 0.1. All weights use Glorot initialisation [22] and biases are initialised at zero. Training is performed with minibatches of 128 data points.

9.3.1 Evaluating Results

For all experiments, we evaluate the results with figures similar to those in the previous chapter. First, we provide a plot of the Kraskov mutual information estimate throughout training. We plot the mutual information throughout stages 1 and 3 of learning, that is, encoder learning and end-to-end fine-tuning. We omit the mutual information during decoder training, as it remains constant throughout this stage. Second, we select three of the 30 runs randomly, and present reconstructions from the three decoders: using only $\mathbf{z}^a$, only $\mathbf{z}^b$, or using both. As a second visualisation of the networks, we will also present 'cross plots'. These take 10 images from the dataset, encodes each, and then decodes all 100 pairings where $\mathbf{z}^a$ is taken from one image and $\mathbf{z}^b$ from another. These are presented in a grid.

We do not present a comparison between our method and others because, as previously discussed, we are unaware of any neural network-based disentanglers built for our task.

9.4 Results

This section tests the DDN model on the benchmarks described in section 8.2: simple MNIST, additive MNIST, occlusive MNIST, and finally MNIST.

9.4.1 Simple MNIST

Our first set of results are on the Simple MNIST dataset, which aims to show that the model is capable of disentangling when the work required is trivial. We train DDNs in the fashion described in section 9.3, with 10 hidden units per cause, and one hidden layer each for all encoders, decoders, and predictors. That is, $N = 10$, and $L_e = L_p = L_d = 1$. We perform 35 epochs of encoder-only training, 10 epochs of decoder-only training, and five epochs of end-to-end fine-tuning. Figure 9.5 shows the mutual infor-

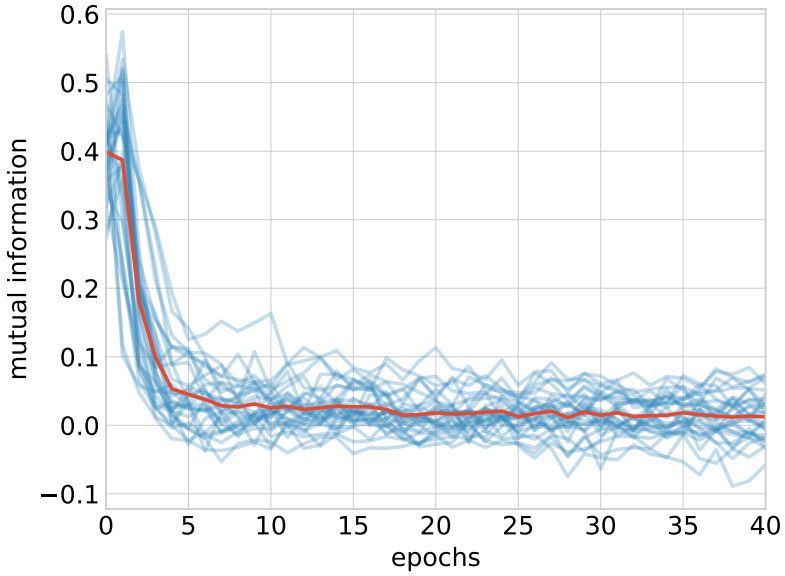


Figure 9.5: Estimated mutual information across 30 runs on the Simple MNIST dataset. Each blue plot is a run, and the mean is shown in red. The associated reconstruction and generation examples are provided in figures 9.7 and 9.6.

mation throughout training for all 30 runs, generation examples are provided in figure 9.6, and reconstruction examples are provided in figure 9.7.

Compared to the performance of the ASAE networks, these results are strongly positive. The DDN achieves disentangling on every run, indicating a consistency that was not present in the ASAE model. This is seen both in the mutual information estimate, and by-eye in the reconstructions and cross plot. It should be noted that the DDN model is succeeding in a more difficult setting than the ASAE as, unlike the ASAE, it has no built-in knowledge of the fact that causes combine additively. From this we can conclude that, at least in simple cases, the DDN model is capable of disentangling and is not vulnerable to being trapped in entangled optima like the ASAE network. While our focus is not on achieving state-of-the-art

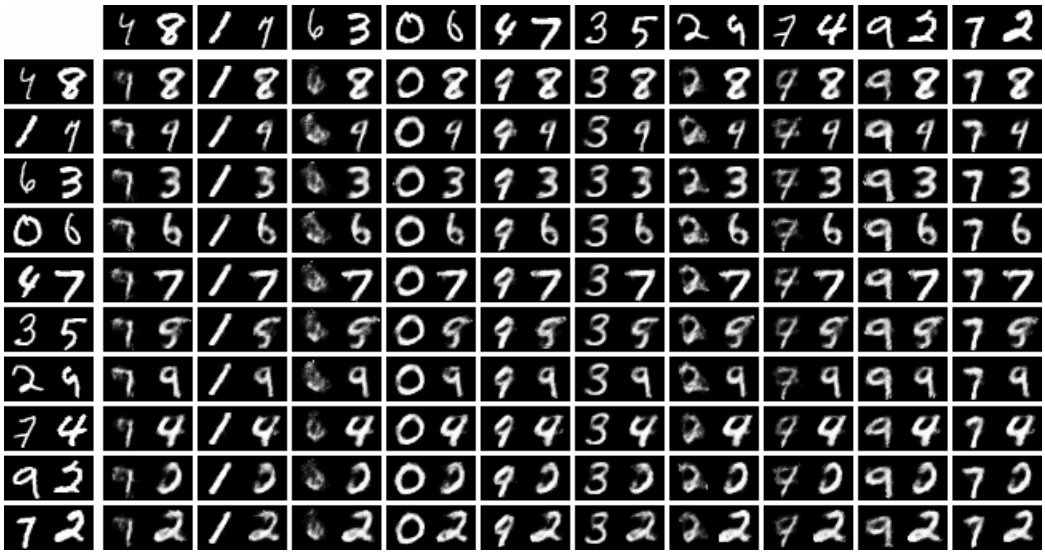


Figure 9.6: Generation examples from one model trained on the Simple MNIST dataset, shown as a cross plot. This shows the combinations of different causes from different inputs. The top-most row and left-most column show 10 input images each, and the central grid is comprised of reconstructions from the network. Each image in the grid is found by decoding $\mathbf{y}^{ab}$ using the $\mathbf{z}^a$ encoding of the corresponding input image in the top row and the $\mathbf{z}^b$ encoding of the corresponding image in the left column. Hence, all reconstructions in a given column are created from the same $\mathbf{z}^a$ value, and all in a given row share the same $\mathbf{z}^b$ value. The associated mutual information plots and reconstruction examples are provided in figures 9.5 and 9.7.

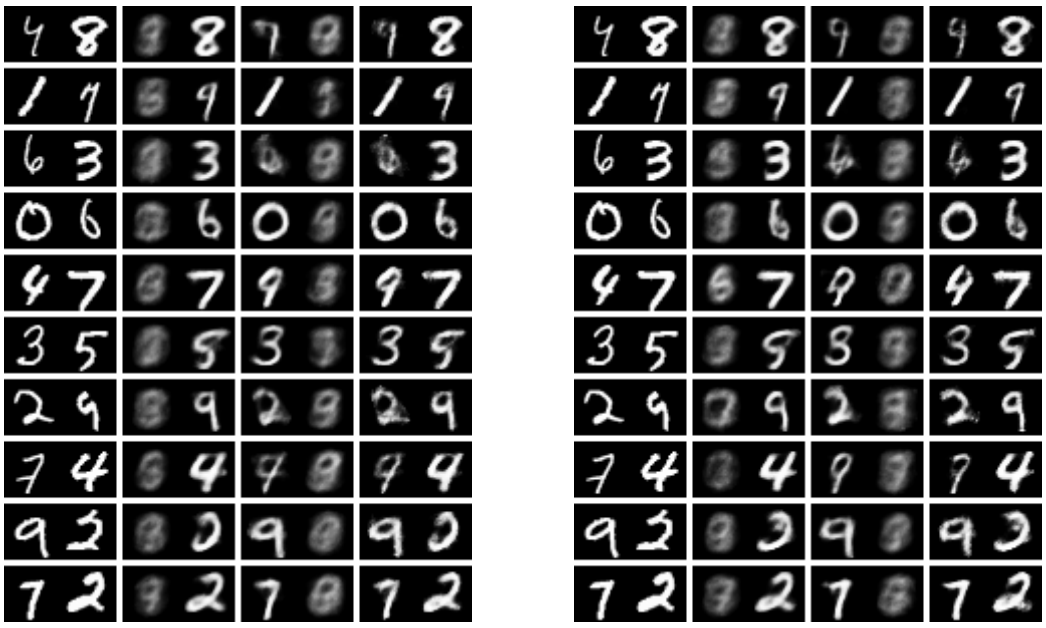


Figure 9.7: Sample reconstructions from two randomly selected runs on the Simple MNIST dataset. For each run, the first column is the input, the second column is the reconstruction $\mathbf{y}^a$ that uses only $\mathbf{z}^a$, the third column is the reconstruction $\mathbf{y}^b$ that uses only $\mathbf{z}^b$, and the fourth column is the reconstruction $\mathbf{y}^{ab}$ that uses both $\mathbf{z}^a$ and $\mathbf{z}^b$. The associated mutual information plots and generation examples are provided in figures 9.5 and 9.6

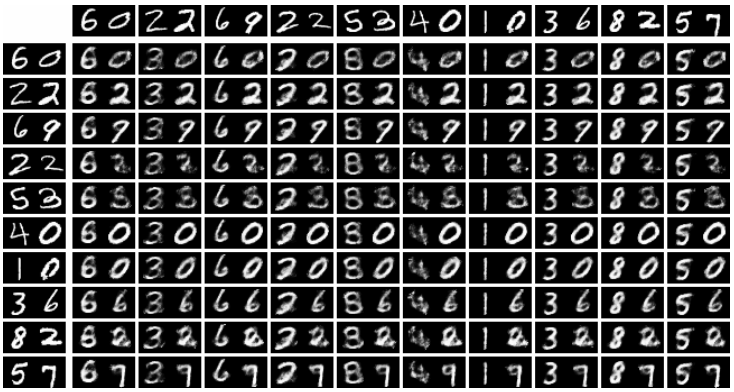


Figure 9.8: Cross plot for a trained DDN model on Simple MNIST *without* end-to-end fine-tuning, to be compared with figure 9.6. The disentangling is similarly as good as when fine-tuning is used, but reconstructions are of a somewhat lower quality.

reconstruction, the results in figure 9.7 are loosely comparable with the reconstructions from an equivalent standard autoencoder.

As a point of comparison, the previous test has been repeated *without end-to-end training*, and a cross plot is presented in figure 9.8. In this case, the mutual information is minimised in a near-identical fashion to the previous results, but the reconstructions are of a slightly lower quality. This is consistent with the motivation for including the end-to-end training step.

9.4.2 Additive MNIST

Next, we test the DDN model on Additive MNIST. This is the first true test of the model's disentangling ability, because the causes have non-trivial (but still linear) entanglement. The networks again use the setup from section 9.3, with 10 hidden units per cause, encoders and predictors with two hidden layers, and decoders with one hidden layer. That is, $N = 10$, $L_e = 2$, $L_p = 2$, and $L_d = 1$. We perform 60 epochs of encoder-only learning, 10 epochs of decoder-only learning, and 40 epochs of end-to-end fine-tuning, leading to the changing character of the mutual information plots. Figures

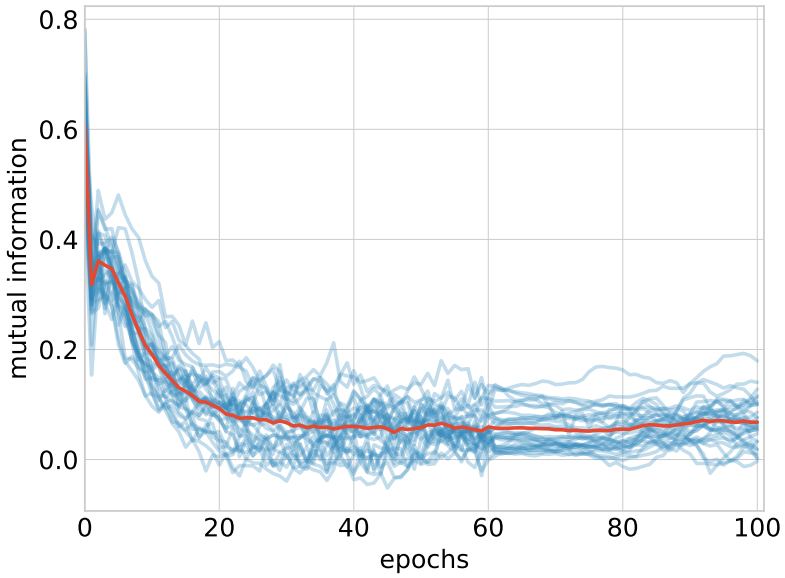


Figure 9.9: Estimated mutual information across 30 runs on the Additive MNIST dataset. Each blue plot is a run, and the mean is shown in red. The first 60 epochs shown are encoder-only training, and the final 40 are end-to-end fine-tuning, explaining the changing character of the curves before and after epoch 60. The associated reconstruction and generation examples are provided in figures 9.11 and 9.10.

9.9, 9.10, and 9.11 present results of 30 runs of training.

These results are imperfect but positive. Across all runs, the individual digits in the data are clearly visible in the reconstructions from each cause, and the cross plot shows structure in both the rows and columns. Similarly, every run achieves a small mutual information between causes according to the Kraskov estimate. This is a significant improvement over the results of the ASAE model, both in terms of the quality and reliability of disentangling. At least in the case of linear combinations, the DDN shows some promise in the unsupervised disentangling of complex causes.

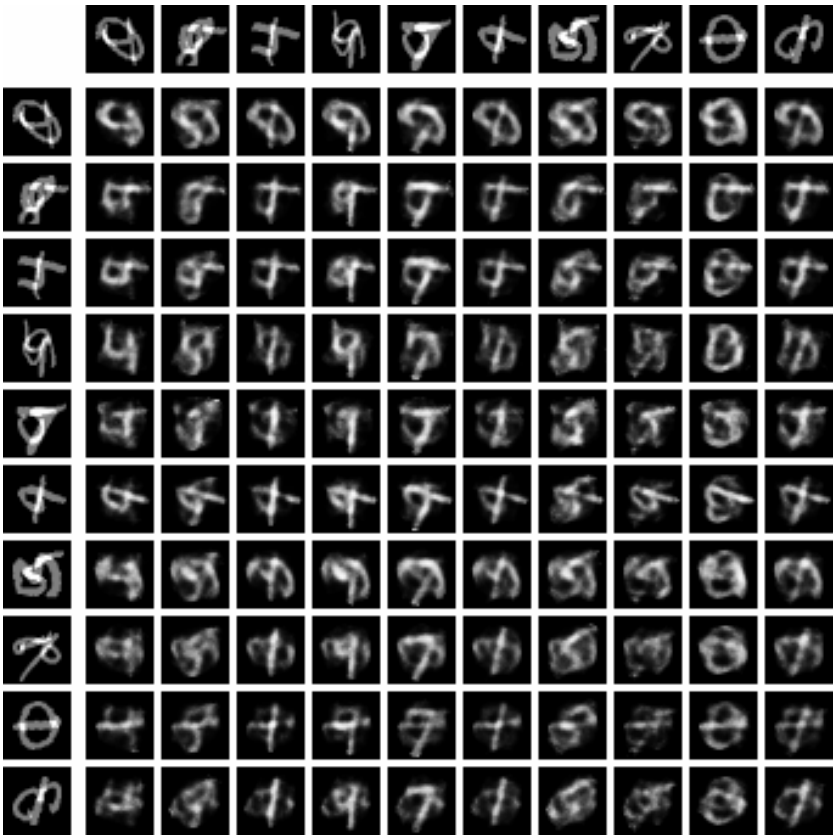


Figure 9.10: Generation examples from one model trained on the Additive MNIST dataset. These are shown in a cross plot, see the caption of figure 9.6 for explanation. The associated mutual information plots and reconstruction examples are provided in figures 9.9 and 9.11.

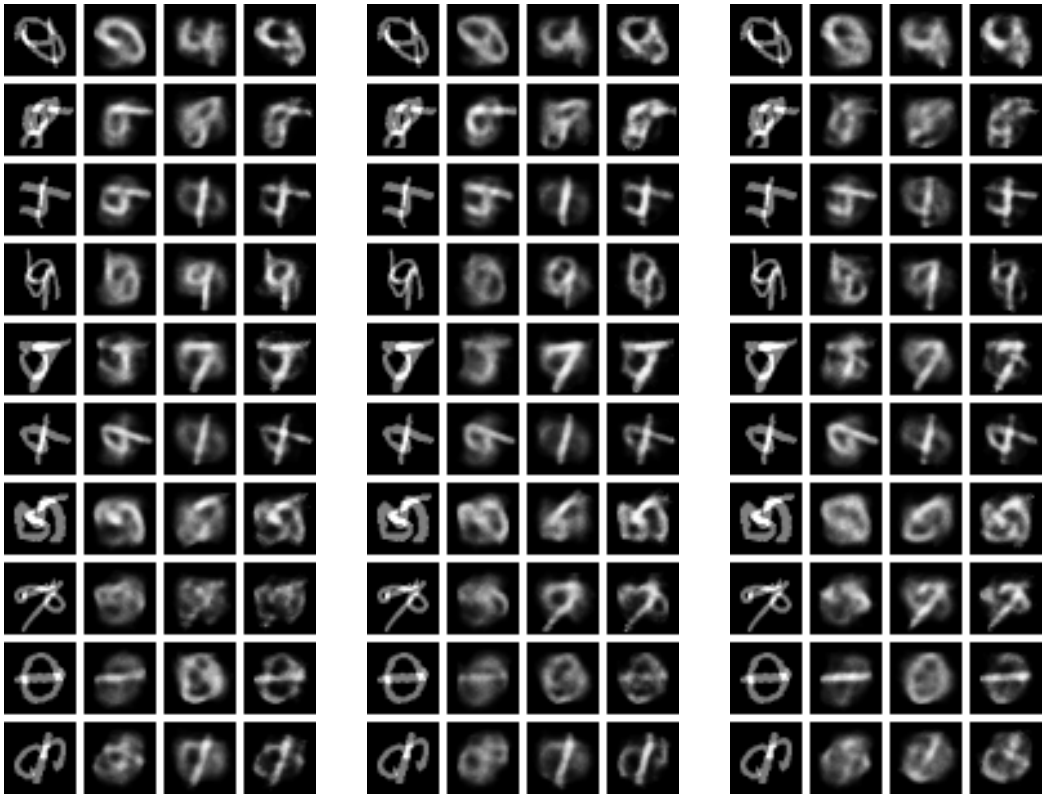


Figure 9.11: Reconstruction examples from three randomly selected runs on the Additive MNIST dataset. See the captions of figure 9.7 for explanation. The associated mutual information plots and generation examples are provided in figures 9.9 and 9.10.

Interestingly, the fine-tuning stage of learning impacts the character of mutual information curves in a way not seen in the Simple MNIST experiment. The predictive loss has unsteady convergence properties, as the pairs of networks seemingly do not stop responding to one another even when an optima is found. Including a reconstruction error term appears to help smooth the loss, as well as cause a slight increase in mutual information.

9.4.3 Occlusive MNIST

Our third experiment benchmarks the model using the Occlusive MNIST dataset. This is a significantly more challenging task than the previous two, as it is the first dataset in which causes *combine non-linearly*. Moreover, the combination results in some loss of information about the causes. Our experiments use 10 units per cause, with two hidden layers in the encoder, and three in the predictor and decoder. That is, in the notation of section 9.3, $N = 10$, $L_e = 2$, $L_d = 3$, and $L_p = 3$. We perform 50 epochs of encoder-only learning, 10 epochs of decoder-only learning, and 50 epochs of end-to-end fine-tuning. The same figures as the previous experiments are produced in figures 9.12, 9.13, and 9.14.

Surprisingly, these results are also positive. The mean mutual information estimate at the end of training is slightly higher than in the additive test, but is still close to zero. The reconstructions from a single cause show that one cause is consistently modelled by each network, although quality suffers due to the fact that reconstructions from a single cause are not well-defined when causes interact non-linearly. The cross plot shows that cause information from two examples can be combined to form new images with some consistency, however the network does not handle the more difficult cases particularly well.

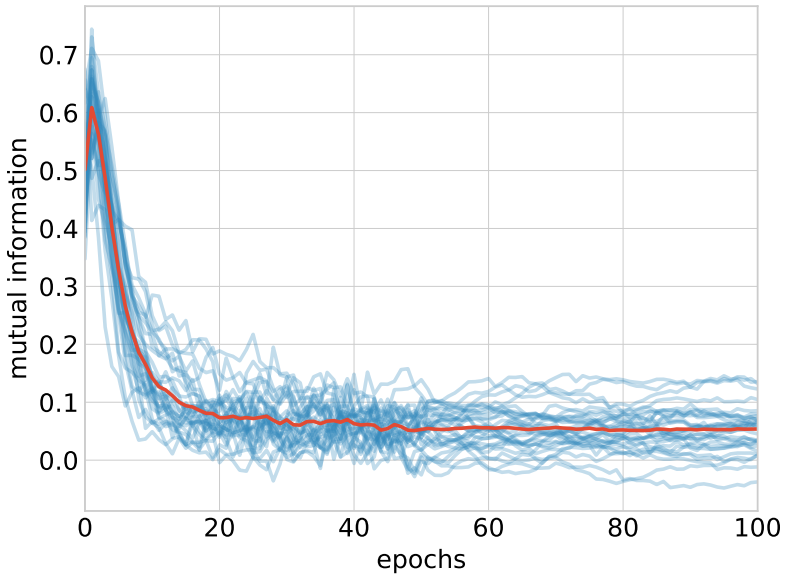


Figure 9.12: Estimated mutual information across 30 runs on the Occlusive MNIST dataset. Each blue plot is a run, and the mean is shown in red. The first 50 epochs shown are encoder-only training, and the last 50 are end-to-end fine-tuning. The associated reconstruction and generation examples are provided in figures 9.14 and 9.13.

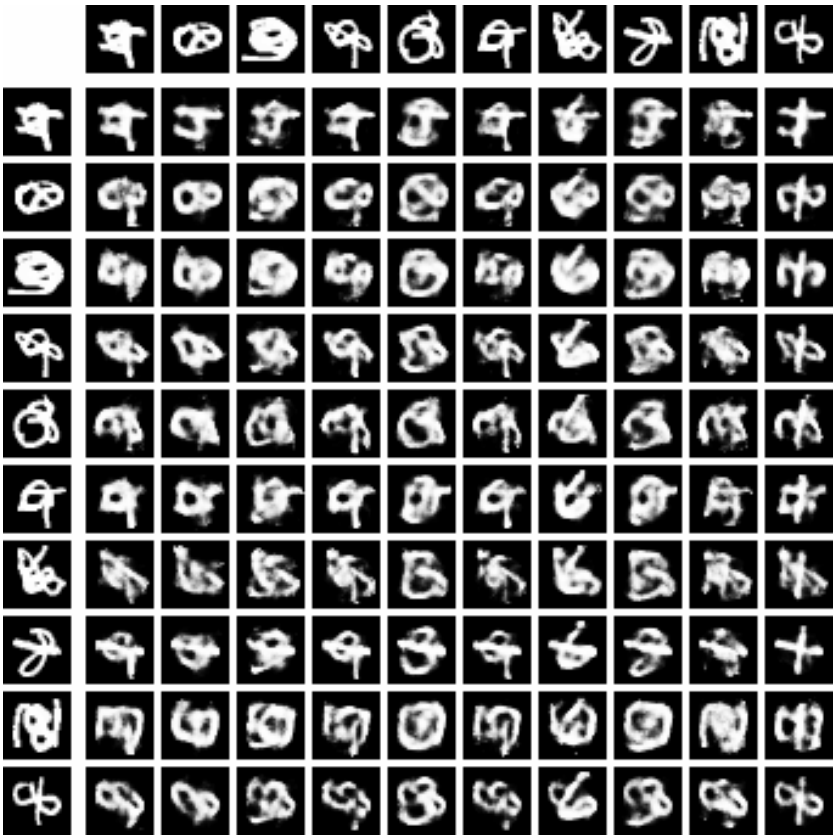


Figure 9.13: Generation examples from one model trained on the Additive MNIST dataset. These are shown in a cross plot, see the caption of figure 9.6 for explanation. The associated mutual information plots and reconstruction examples are provided in figures 9.12 and 9.14.

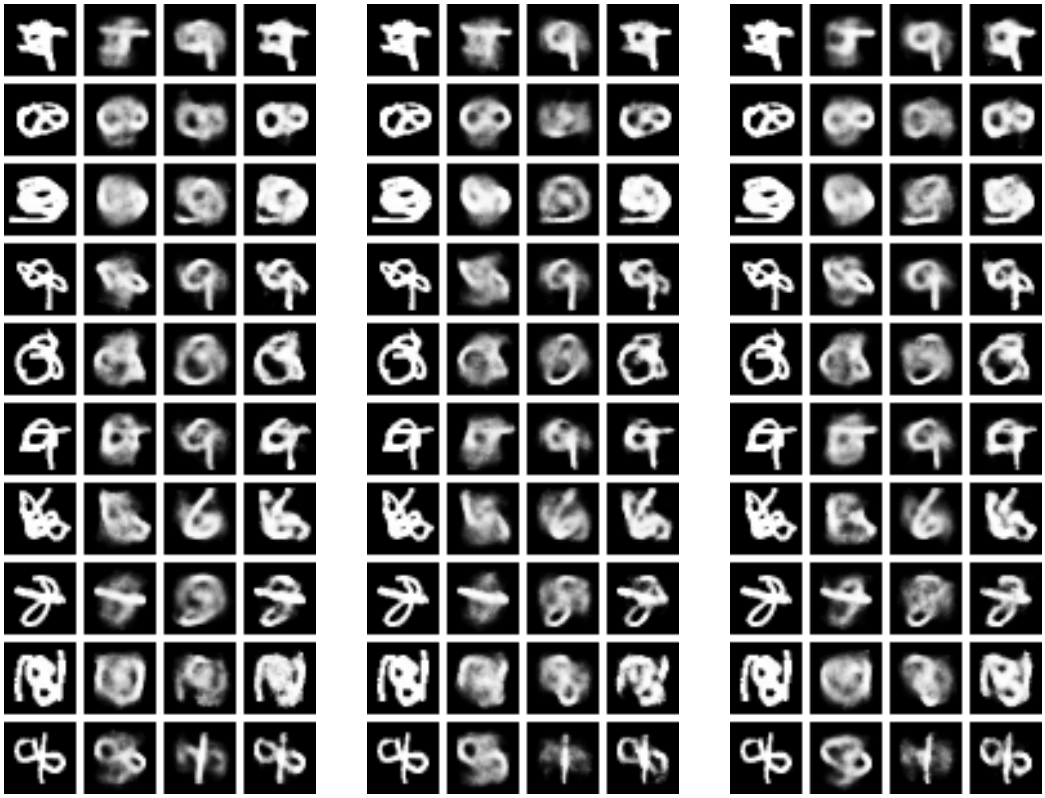


Figure 9.14: Reconstruction examples from three randomly selected runs on the Occlusive MNIST dataset. See the captions of figure 9.7 for explanation. The associated mutual information plots and generation examples are provided in figures 9.12 and 9.13.

9.4.4 MNIST

The final experiment runs on the standard MNIST dataset, with no modifications. As this is not a synthetic benchmark with known, ground-truth causes, the desired outcome of a disentangling algorithm is not entirely clear. However, in the literature, MNIST is most commonly decomposed into factors representing the digit itself, and the handwriting style it is drawn in. The latter factor is generally a combination of the slope of the digit and the width of the stroke. From one view this test is easier than Occlusive MNIST, because the method by which causes combine does not destroy nearly as much information. However, the fact that causes are far more entangled than in previous experiments presents a different challenge for the model.

We train a DDN network on MNIST with 10 hidden units, and two layer encoders, predictors, and decoders. That is, $N = 10$ and $L_e = L_p = L_d = 2$. We have found that our standard training setup is not particularly successful on this problem. However, results are improved by performing a large amount of end-to-end fine-tuning, throughout which the α hyperparameter is annealed to increase the contribution of reconstruction error over time. An initial stage of encoder-only learning is still required for a solution with low mutual information. In sum, our training procedure involves 20 epochs of encoder-only learning, 10 epochs to decoder-only learning, and 100 epochs of annealed fine-tuning.

Figure 9.15 provides an example cross plot from a DDN trained in this fashion. The results show minor success: the network has, to a degree, separated what would normally be label information from handwriting style, and is able to transfer these causes. However the 'style' cause models only whether strokes are thick or thin, and ignores finer style properties. This leads to many reconstructions looking similar regardless of the style cause. This network achieved a final estimated mutual information score of -0.01, and convergence to a low mutual information solution was consistent across five runs. While these are preliminary results of only limited

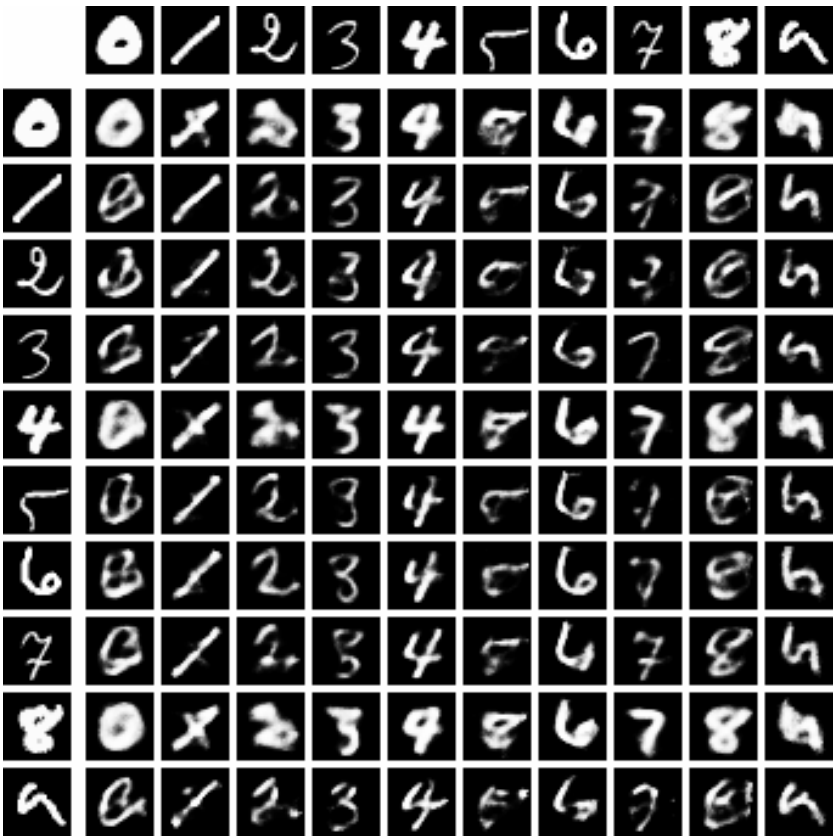


Figure 9.15: Generation examples on the standard MNIST dataset shown in cross plot. The examples used in this plot have been manually selected to display a range of digits and handwriting styles. The model achieves a minor degree of disentangling, but one cause primarily encodes whether the strokes are thick or thin rather than more complete style information.

success, they suggest that adversarial networks have the capacity to perform unsupervised disentangling of complex, non-linear causes in natural data.

9.5 Discussion

9.5.1 The Importance of Split Encoders

Since the DDN model is driven to disentangle by its loss function, rather than its architecture, it is reasonable to question whether two separate encoders are required. Alternatively, they could be replaced by a single encoder f that outputs both $\mathbf{z}^a$ and $\mathbf{z}^b$. This has been tested, and the split encoder is vital to the stability of training. The problem is as follows: if any parameters effect both $\mathbf{z}^a$ and $\mathbf{z}^b$, then a change to minimise ℓ_a^E has an unintended effect on $\mathbf{z}^b$, which has an unintended effect of q^{ba}, resulting in a change to ℓ_a^E that is not accounted for by gradient descent. In other words, gradient descent computes an incorrect gradient because it cannot account for the changes to the predictive networks. The consequence of this is highly unstable learning.

9.5.2 Insights into the model

Throughout the process of performing this chapter's experiments, several observations have been made about the conditions required for successful disentangling.

Empirically, the most important single factor in successfully training a DDN is selecting the right number and size of layers in the encoders, predictors, and decoders. Our observations can be encapsulated in three rules, two of which are intuitive and one surprising. The first rule is that the decoder networks generally must be deeper and more powerful than one would expect. For example, representing additive data was most successful with a decoder with two hidden layers. A likely explanation for this

is that the first stage of learning is not directly incentivised to make the information necessary for decoding easy for a decoder to find. This problem is lessened by end-to-end fine-tuning, but potentially is not entirely fixed.

The second rule is that the predictors must be at least as 'powerful' as the encoders. Intuitively, this is unsurprising. If the encoders have an advantage in representational power, they may be able to hide information about other causes in their encodings in such a way that a predictor cannot uncover. In the adversarial game played between two networks, the encoder has available to it a winning solution that is infallible regardless of the power of the predictor: zero mutual information between causes. By making the predictors as powerful as necessary, the encoder is unable to win by other means.

The third rule is troubling: encoders have a 'Goldilocks zone' of representational power. If the architecture is too shallow *or* too deep then disentangling does not occur. The explanation for too-shallow networks is clear: a certain amount of power is required to extract disentangled representations from data generated by a complex or lossy entangling process. This is exemplified by the need for deeper encoders with Occlusive MNIST and standard MNIST than with the simple or additive datasets. However, if the encoder network is too deep, then the causes are not visually disentangled even when matched with a predictor of equal or greater depth. Concerningly, although by-eye disentangling does not occur, the mutual information estimate *is still nearly zero*. The fact that mutual information is minimised suggests that this is not a direct issue with training, such as a vanishing gradient problem in a too-deep network. Rather, we conjecture that this behaviour results from a fundamental complexity of the problem of disentangling.

Originally, independent causes were defined as a partitioning of latent factors such that the partition is *statistically independent* and, by implication, has zero mutual information. However, this definition does not capture the full scope and complexity of the disentangling problem, because com-

plex and independent causes of data *are themselves likely to have independent causes*. For example, in the case of faces and lighting, faces could be considered to be caused by an interaction of independent factors such as age and gender. More concretely, significant literature and the results of this chapter indicate that unmodified MNIST can be decomposed into 'style' and 'content' factors, that are largely independent. In that case, a dataset such as Additive MNIST can just as correctly be seen as the result of *four* independent causes, two style and two content, rather than *two* independent digits. If one is attempting to partition latent factors into two independent groups, a solution that partitions the data into a style cause and content cause is equally as valid as a solution that partitions into two digits. Nevertheless, only the partition into digits is intuitively correct. From the perspective of a model seeking only to find *some* independent partition of latent factors, what distinguishes the intuitively correct solution from all other possible combinations?

One answer is that grouping Additive MNIST into digits is the *simplest explanation of the data*. Phrased in computational terms, one could say it is a more *efficient* representation than modelling digits and modelling styles. Therefore, we conjecture that *a model is incentivised towards the intuitively correct partition of independent factors because it is the most efficient representation of the data*. This, finally, provides an explanation of behaviour seen when the encoder of a DDN is too deep: if the encoder is too powerful, the extra efficiency provided by the correct partition is meaningless, as the model is powerful enough to easily represent incorrect partitions. This leads to zero mutual information, but no obvious disentangling.

9.5.3 Conclusion

This chapter has presented the Discriminative Disentangling Network as a model for disentangling complex causes by using adversarial games to penalise predictability. In our experiments, the model is capable of consistently achieving very low estimated mutual information between causes, as

was its original goal. This translates into moderately successful by-eye disentangling. To the best of our knowledge, these are the first disentangling results for neural networks in an entirely unsupervised setting, where the causes are not limited in their complexity to a single scalar value. However, the DDN model is clearly not a complete solution to the disentangling problem. Rather, we present it as a proof-of-concept that shows the promise of disentangling with predictive adversarial networks. We believe that a more thorough analysis of the encoding loss as an approximation of conditional mutual information is the most promising direction to improving the algorithm further, but we leave this as future work.

Chapter 10

Conclusion

10.1 Summary

This thesis has explored the problem of unsupervised disentangling of the complex causes of data. This is a difficult task; successful disentangling requires teasing apart the fundamental independencies in data, using no information beyond the data itself. Indeed, when the independent causes of data are themselves comprised of independent causes, the problem is somewhat ill-defined.

We have investigated three methods for disentangling. Part I took a Bayesian approach, and viewed disentangling as an inference problem. This led to the development of the Broadnet model, which represents data with several RBMs in parallel. From a generative perspective, this approach has several appealing qualities as a model of independent causes of data. Unfortunately, analysis suggests that this perspective is flawed, and Broadnets are not capable of learning disentangled representations.

Part II analysed and built on the connection between ICA and our problem. This was motivated by the similarity of ICA's task to ours. As modifying ICA itself to find complex causes seems unlikely, we developed a post-processing algorithm on ICA's output intended to group independent components into their natural causes using an estimate of mutual information. This algorithm is successful to a degree in our test case: so long as the number of independent components is not too large, components representing one cause can be separated from another. However, when the data is modelled with too many components, the mutual information measure becomes uninformative, and the method breaks down. The fact that there is residual mutual information present in the solutions found by ICA, and that it can be used to cluster independent components into independent causes, is an interesting result. However the limitations of this method do not make for a general algorithm for finding disentangled representations.

Part III approached disentangling as a *learning* problem, and investigated the use of unsupervised neural networks. We first proposed the Additive Split Autoencoder model, which is a simple structural change to an autoencoder. Experiments showed that the model *is* capable of disentangling, and can achieve convincing reconstructions of linearly combined causes. This method shows promise, though the inconsistency of its convergence limits its usefulness.

Finally, part III also investigated methods for directly encouraging disentangled representations. Our chosen approach, the Discriminative Disentangling Network, uses a pair of adversarial games to make two representations of the data unpredictable from each other. The DDN is novel in that the majority of its learning does not make use of reconstruction error, and stands apart from our other work due to its ability to model complex, non-linear combinations of causes. This model has shown some success when applied to our test suite, in that it can consistently find representations with low mutual information. This translated to a degree of disentangling, even on natural data, but there is certainly room for improvement.

10.2 Conclusions and Future Work

Perhaps the most significant conclusion of this work is that unsupervised disentangling is extremely difficult to even define, let alone perform. The problem is significantly more complex than other independence-related tasks, such as finding independent components or factors of variation represented by a scalar. When causes are themselves composed of causes, there is little to differentiate the correct grouping beyond an argument for representational efficiency. As such, successful disentangling entails discovering and modelling the underlying generative process of data, certainly a more involved task than finding standard representations. As discussed by Bengio et al. [8], this could be considered one of the ultimate goals of AI, or at least representation learning.

Part of the value of unsupervised disentangling is its similarity to learning in the real world: causes are complex, and labels are not provided. However, it is unclear whether even humans are capable of this. In reality, we have ample exposure to data in which one cause is fixed while another varies; we often see the same scenes from multiple angles, for example. More concretely, on simple cases such as two MNIST digits unioned together, distinguishing the underlying causes is often impractical even with years of exposure to the individual digits.

Despite the singular difficulty of the task, our work provides some indications that *it is in fact possible*: the results of chapter 9 show promise for the use of adversarial networks in finding disentangled representations, even in the case of natural data with a complex entanglement process.

Aside from our core challenge, this work has raised a number of questions about the models and algorithms used. Part I made extensive use of GRBMs, which are infamous for their limited representational power and temperamental training. While many extensions, most notably ssRBMs, convincingly improve representational power, our experience is that they are similarly as tricky to train. This appears to be a common trait among

energy models, though it is unclear why.

The experiments in part III put into focus the complexities of using neural networks in practice. For example, split autoencoder networks are clearly vulnerable to entangled optima, which makes a case for using a gradient descent method that performs a degree of exploration, rather than strict gradient descent. However, the fact that RMSprop succeeds where Adam fails is mysterious, and suggests that the optimisation problem at hand is more complicated than it initially seems.

In the context of DDNs, even semi-successful disentangling requires a fine balance of almost every tunable attribute of the model: networks must be balanced in terms of their representational power and training speed, a specific optimiser must be used, and the different losses in the model must be introduced at different points throughout training. In part, this is likely indicative of deficiencies in the model itself. However, our experience also echoes the well-known stability issues with GANs. As the use of adversarial networks has only come to prominence recently, it seems likely that the dynamics of adversarial settings are generally not well understood.

We believe the most promising launching point for future work into unsupervised disentangling is, unsurprisingly, based on neural networks. More specifically, the preliminary success of adversarial networks suggests several avenues for further development. A more principled approach to autoencoder-like networks has potential; a concrete formulation of predictive networks as an approximation to mutual information may yield a loss function with fewer practical issues. Combining this with GANs could be productive, which may be made possible by recent work that enables one to find representations with GANs, as well as generate examples. From the understanding gained in this thesis we believe this is worthy of future research, and could well provide a solution to the unsupervised disentangling problem.

Bibliography

[1] Marcin Andrychowicz, Misha Denil, Sergio Gomez, Matthew W Hoffman, David Pfau, Tom Schaul, and Nando de Freitas. Learning to learn by gradient descent by gradient descent. In *Advances in Neural Information Processing Systems*, pages 3981–3989, 2016.

[2] David Balduzzi, Brian McWilliams, and Tony Butler-Yeoman. Neural Taylor approximations: Convergence and exploration in rectifier networks. *arXiv preprint arXiv:1611.02345*, 2016.

[3] David Barber and Felix V Agakov. The IM algorithm: A variational approach to information maximization. In *Advances in Neural Information Processing Systems*, pages 201–208, 2003.

[4] Adel Belouchrani, Karim Abed-Meraim, J-F Cardoso, and Eric Moulines. A blind source separation technique using second-order statistics. *IEEE Transactions on signal processing*, 45(2):434–444, 1997.

[5] Yoshua Bengio, Aaron C Courville, and James S Bergstra. Unsupervised models of images by spike-and-slab RBMs. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pages 1145–1152, 2011.

[6] Yoshua Bengio, Aaron C Courville, and Pascal Vincent. Unsupervised feature learning and deep learning: A review and new perspectives. *CoRR, abs/1206.5538*, 1, 2012.

[7] Yoshua Bengio, Aaron Courville, and Pascal Vincent. Representation learning: A review and new perspectives. *IEEE transactions on pattern analysis and machine intelligence*, 35(8):1798–1828, 2013.

[8] Yoshua Bengio, Grégoire Mesnil, Yann Dauphin, and Salah Rifai. Better mixing via deep representations. In *Proceedings of the 30th International Conference on Machine Learning (ICML-1)*, pages 552–560, 2013.

[9] David Berthelot, Tom Schumm, and Luke Metz. Began: Boundary equilibrium generative adversarial networks. *arXiv preprint arXiv:1703.10717*, 2017.

[10] Yuri Burda, Roger Grosse, and Ruslan Salakhutdinov. Importance weighted autoencoders. *arXiv preprint arXiv:1509.00519*, 2015.

[11] Xi Chen, Yan Duan, Rein Houthooft, John Schulman, Ilya Sutskever, and Pieter Abbeel. InfoGAN: Interpretable representation learning by information maximizing generative adversarial nets. In *Advances in Neural Information Processing Systems*, pages 2172–2180, 2016.

[12] Brian Cheung, Jesse A Livezey, Arjun K Bansal, and Bruno A Olshausen. Discovering hidden factors of variation in deep networks. *arXiv preprint arXiv:1412.6583*, 2014.

[13] Aaron C Courville, James Bergstra, and Yoshua Bengio. A spike and slab restricted boltzmann machine. In *Proceedings of the 14th International Conference on Artificial Intelligence Systems*, pages 233–241, 2011.

[14] Thomas M Cover and Joy A Thomas. *Elements of information theory*. John Wiley & Sons, 1991.

[15] Yann N Dauphin, Razvan Pascanu, Caglar Gulcehre, Kyunghyun Cho, Surya Ganguli, and Yoshua Bengio. Identifying and attacking the saddle point problem in high-dimensional non-convex optimization.

In *Advances in Neural Information Processing Systems*, pages 2933–2941, 2014.

[16] Guillaume Desjardins, Aaron Courville, and Yoshua Bengio. Disentangling factors of variation via generative entangling. *arXiv preprint arXiv:1210.5474*, 2012.

[17] Timothy Dozat. Incorporating nesterov momentum into adam. 2016.

[18] Lev Faivishevsky and Jacob Goldberger. ICA based on a smooth estimation of the differential entropy. In *Advances in Neural Information Processing Systems*, pages 433–440, 2009.

[19] Lev Faivishevsky and Jacob Goldberger. Nonparametric information theoretic clustering algorithm. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pages 351–358, 2010.

[20] Yoav Freund and David Haussler. Unsupervised learning of distributions of binary vectors using two layer networks. *Advances in Neural Information Processing Systems*, 1992.

[21] Stuart Geman and Donald Geman. Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images. *IEEE Transactions on pattern analysis and machine intelligence*, (6):721–741, 1984.

[22] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Aistats*, volume 9, pages 249–256, 2010.

[23] Max Godfrey. Richer Restricted Boltzmann Machines. Honours report, Victoria University of Wellington.

[24] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in Neural Information Processing Systems*, pages 2672–2680, 2014.

[25] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[26] Arthur Gretton, Karsten M Borgwardt, Malte Rasch, Bernhard Schölkopf, and Alex J Smola. A kernel method for the two-sample-problem. In *Advances in Neural Information Processing Systems*, pages 513–520, 2007.

[27] Arthur Gretton, Karsten M Borgwardt, Malte J Rasch, Bernhard Schölkopf, and Alexander Smola. A kernel two-sample test. *Journal of Machine Learning Research*, 13(Mar):723–773, 2012.

[28] David B Grimes and Rajesh PN Rao. Bilinear sparse coding for invariant vision. *Neural computation*, 17(1):47–73, 2005.

[29] W Keith Hastings. Monte carlo sampling methods using markov chains and their applications. *Biometrika*, 57(1):97–109, 1970.

[30] Geoffrey Hinton. A practical guide to training restricted Boltzmann machines. *Momentum*, 9(1):926, 2010.

[31] Geoffrey E Hinton. Training products of experts by minimizing contrastive divergence. *Neural computation*, 14(8):1771–1800, 2002.

[32] Geoffrey E Hinton. Modeling pixel means and covariances using factorized third-order Boltzmann machines. In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pages 2551–2558. IEEE, 2010.

[33] Geoffrey E Hinton and Ruslan R Salakhutdinov. Reducing the dimensionality of data with neural networks. *science*, 313(5786):504–507, 2006.

[34] Geoffrey E Hinton, Simon Osindero, and Yee-Whye Teh. A fast learning algorithm for deep belief nets. *Neural computation*, 18(7):1527–1554, 2006.

[35] Aapo Hyvärinen and Erkki Oja. Independent component analysis: algorithms and applications. *Neural networks*, 13(4):411–430, 2000.

[36] Aapo Hyvärinen, Juha Karhunen, and Erkki Oja. *Independent component analysis*, volume 46. John Wiley & Sons, 2004.

[37] John G Kemeny, James Laurie Snell, et al. *Finite markov chains*, volume 356. van Nostrand Princeton, NJ, 1960.

[38] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[39] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[40] Diederik P Kingma, Shakir Mohamed, Danilo Jimenez Rezende, and Max Welling. Semi-supervised learning with deep generative models. In *Advances in Neural Information Processing Systems*, pages 3581–3589, 2014.

[41] Scott Kirkpatrick, C Daniel Gelatt, and Mario P Vecchi. Optimization by simulated annealing. *science*, 220(4598):671–680, 1983.

[42] Daphne Koller, Nir Friedman, Lise Getoor, and Ben Taskar. Graphical models in a nutshell. *Introduction to statistical relational learning*, pages 13–55, 2007.

[43] Alexander Kraskov, Harald Stögbauer, and Peter Grassberger. Estimating mutual information. *Physical review E*, 69(6):066138, 2004.

[44] Tejas D Kulkarni, William F Whitney, Pushmeet Kohli, and Josh Tenenbaum. Deep convolutional inverse graphics network. In *Advances in Neural Information Processing Systems*, pages 2539–2547, 2015.

[45] Dominic Langlois, Sylvain Chartier, and Dominique Gosselin. An introduction to independent component analysis: InfoMax and FastICA

algorithms. *Tutorials in Quantitative Methods for Psychology*, 6(1):31–38, 2010.

[46] Christian Ledig, Lucas Theis, Ferenc Huszár, Jose Caballero, Andrew Cunningham, Alejandro Acosta, Andrew Aitken, Alykhan Tejani, Johannes Totz, Zehan Wang, et al. Photo-realistic single image super-resolution using a generative adversarial network. *arXiv preprint arXiv:1609.04802*, 2016.

[47] Yujia Li, Kevin Swersky, and Richard S Zemel. Generative moment matching networks. In *Proceedings on the International Conference on Machine Learning*, pages 1718–1727, 2015.

[48] David JC MacKay. *Information theory, inference and learning algorithms*. Cambridge university press, 2003.

[49] Alireza Makhzani, Jonathon Shlens, Navdeep Jaitly, Ian Goodfellow, and Brendan Frey. Adversarial autoencoders. *arXiv preprint arXiv:1511.05644*, 2015.

[50] Michael F Mathieu, Junbo Jake Zhao, Junbo Zhao, Aditya Ramesh, Pablo Sprechmann, and Yann LeCun. Disentangling factors of variation in deep representation using adversarial training. In *Advances in Neural Information Processing Systems*, pages 5041–5049, 2016.

[51] Nicholas Metropolis, Arianna W Rosenbluth, Marshall N Rosenbluth, Augusta H Teller, and Edward Teller. Equation of state calculations by fast computing machines. *The journal of chemical physics*, 21(6):1087–1092, 1953.

[52] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on machine learning (ICML-10)*, pages 807–814, 2010.

[53] Bruno A Olshausen, Charles Cadieu, Jack Culpepper, and David K Warland. Bilinear models of natural images. In *Electronic Imaging 2007*, pages 649206–649206. International Society for Optics and Photonics, 2007.

[54] Judea Pearl. Bayesian networks. *Department of Statistics, UCLA*, 2011.

[55] Scott Reed, Kihyuk Sohn, Yuting Zhang, and Honglak Lee. Learning to disentangle factors of variation with manifold interaction. In *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*, pages 1431–1439, 2014.

[56] Salah Rifai, Yoshua Bengio, Aaron Courville, Pascal Vincent, and Mehdi Mirza. Disentangling factors of variation for facial expression recognition. *Computer Vision–ECCV 2012*, pages 808–822, 2012.

[57] Sebastian Ruder. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, 2016.

[58] Claude Elwood Shannon. A mathematical theory of communication. *ACM SIGMOBILE Mobile Computing and Communications Review*, 5(1): 3–55, 2001.

[59] Paul Smolensky. Information processing in dynamical systems: Foundations of harmony theory. Technical report, DTIC Document, 1986.

[60] Josh M Susskind, Adam K Anderson, and Geoffrey E Hinton. The toronto face database. *Department of Computer Science, University of Toronto, Toronto, ON, Canada, Tech. Rep*, 3, 2010.

[61] Joshua B Tenenbaum and William T Freeman. Separating style and content. *Advances in Neural Information Processing Systems*, pages 662–668, 1997.

[62] Lucas Theis, Aäron van den Oord, and Matthias Bethge. A note on the evaluation of generative models. *arXiv preprint arXiv:1511.01844*, 2015.

[63] Tijmen Tieleman. Training restricted Boltzmann machines using approximations to the likelihood gradient. In *Proceedings of the 25th International Conference on Machine learning*, pages 1064–1071. ACM, 2008.

[64] Tijmen Tieleman and Geoffrey Hinton. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, 4(2), 2012.

[65] M Alex O Vasilescu and Demetri Terzopoulos. Multilinear analysis of image ensembles: Tensorfaces. In *European Conference on Computer Vision*, pages 447–460. Springer, 2002.

[66] Pascal Vincent, Hugo Larochelle, Isabelle Lajoie, Yoshua Bengio, and Pierre-Antoine Manzagol. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *Journal of Machine Learning Research*, 11(Dec):3371–3408, 2010.

[67] Satosi Watanabe. Information theoretical analysis of multivariate correlation. *IBM Journal of research and development*, 4(1):66–82, 1960.

[68] William Whitney. Disentangled representations in neural models. *arXiv preprint arXiv:1602.02383*, 2016.

[69] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. *arXiv preprint arXiv:1703.10593*, 2017.