A Model-Adaptive Universal Data Compression Architecture with Applications to Image Compression

by

Joshua Ka-Wing Lee

B.A.Sc. in Engineering Science, University of Toronto (2015)

Submitted to the Department of Electrical Engineering and Computer Science in partial fulfillment of the requirements for the degree of

Master of Science in Computer Science and Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2017

Author . . . .
Department of Electrical Engineering and Computer Science
May 19, 2017

Certified by . . . .
Gregory W. Wornell
Professor of Electrical Engineering and Computer Science
Thesis Supervisor

Accepted by . . . .
Leslie A. Kolodziejski
Professor of Electrical Engineering and Computer Science
Chair, Committee on Graduate Theses

A Model-Adaptive Universal Data Compression Architecture with Applications to Image Compression

by

Joshua Ka-Wing Lee

Submitted to the Department of Electrical Engineering and Computer Science
on May 19, 2017, in partial fulfillment of the
requirements for the degree of
Master of Science in Computer Science and Engineering

Abstract

In this thesis, I designed and implemented a model-adaptive data compression system for the compression of image data. The system is a realization and extension of the Model-Quantizer-Code-Separation Architecture for universal data compression which uses Low-Density-Parity-Check Codes for encoding and probabilistic graphical models and message-passing algorithms for decoding. We implement a lossless bi-level image data compressor as well as a lossy greyscale image compressor and explain how these compressors can rapidly adapt to changes in source models. We then show using these implementations that Restricted Boltzmann Machines are an effective source model for compressing image data compared to other compression methods by comparing compression performance using these source models on various image datasets.

Thesis Supervisor: Gregory W. Wornell
Title: Professor of Electrical Engineering and Computer Science

Acknowledgments

I would like to extend my sincerest thanks to the many people without whose support this work would not have been completed.

First and foremost, I would like to thank my advisor Professor Gregory Wornell for his guidance and teaching throughout my journey here at MIT. His advice has led me to the discovery of new ideas and fields that I would never have encountered otherwise. I owe much of my knowledge and passion for machine learning and information theory to him, from lessons learned in the classroom to those gleaned from our meetings, and without his inspiration this project would never have existed in the first place. I am grateful beyond words for all he has done for me to help me grow as a student and a researcher.

Much of the early work done is this thesis has been done in partnership with Angus Lai. As a research partner, he has helped me greatly both in developing the ideas that are central to the workings of the project as well as in implementing the project, especially in improving the runtime of our code. I would like to thank him for being a great research partner who has always been willing to do what needs to be done to make the project a success, and for helping make this project greater than the sum of its parts.

The inspiration for this work comes directly from Dr. Ying-Zong Huang, whose 2015 PhD dissertation forms the basis of our project. Dr. Huang has always been willing to provide us with advice that has opened our eyes to new possibilities and new ways of problem-solving, in addition to providing insights and data related to his previous work on his dissertation, and for that I will always be grateful.

In the later parts of my research, I had the opportunity to speak with Dr. Jerry Shapiro multiple times about his research into image modeling. Dr. Shapiro was generous enough to meet with me several times over the course of a few months to discuss my research, and his experience with image modeling has provided me with vital advice needed to move forward with my project many times, for which I am thankful.

In the process of our exploratory research, Angus and I had the opportunity to speak with Dr. Ulugbek Kamilov about his work on message-passing dequantization, which proved to be an important part of our research. His insights led to an important breakthrough in our research, and I would like to thank him for that.

I am also grateful to the SIA community for their support and advice, and for providing me with a friendly and intellectually-stimulating environment that has given me the drive and knowledge needed to complete this project.

Finally, I extend my deepest thanks and love to my parents, who have guided me along my path to achieving my dreams and given me the love, support, and teachings that has allowed me to persevere through hardships and find myself where I am today.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Data Compression is a long-studied problem of great importance, with applications in many fields. In recent years, developments in the field of machine learning has allowed for novel breakthroughs in solving this problem. One such breakthrough is the development of the Model-Quantizer-Code Separation (MQCS) Architecture for universal compression. In this thesis, we extend the concept of the MQCS Architecture to the problem of lossy compression of continuous-alphabet sources and explore the applications of this archiecture to the compression of images. We will also illustrate the benefits this architecture has over other methods of data compression.

1.1 Motivation

We begin with a motivating example to illustrate the necessity of the MQCS Architecture.

1.1.1 Example: A Tale of Two Standards

In 1987, the Joint Photographic Experts Group (JPEG) released an image compression standard for the lossless and lossy compression of natural images. The JPEG standard used a perceptual model of image fidelity to design a quantization scheme centered around the discrete cosine transform (DCT) [33]. This standard soon become the most used method for compressing image data, used in most computer systems as well as most digital cameras [31].

In 2000, the JPEG2000 standard was released by the same group [28]. The new standard used an improved image model exploited by the wavelet transform to provide compression ratios and reconstruction fidelities superior to that of the original JPEG standard as well as providing additional features such as scalability and progressive decoding.

However, the standard format used for image compression in most modern computer systems is still JPEG, with few cameras even supporting JPEG2000 encoding, despite the improvements that JPEG2000 provides [1]. One main reason why JPEG2000 never become a replacement for JPEG is the lack of backwards compatibility. JPEG and JPEG2000 use two completely different coding schemes, and as such require two different encoders and decoders. Switching over to the JPEG2000 standard would require re-encoding all existing JPEG images using the new codec and, in addition, most imaging software would need to continue to support both formats for a period of at least several years in the event that they encountered unconverted files. In general, the benefits provided by JPEG2000 was not considered to be worth the trouble of switching over to the new system.

Since the creation of the JPEG standard, many advances have been made in the field of image analysis and compression, and yet none of these advances have been leveraged in modern computer systems, with JPEG remaining the standard. The same reasons that prevented the proliferation of JPEG2000 continues to make JPEG the format of choices for image compression, as few wish to have to re-encode their images each time a better image model is developed.

1.1.2 A Universal Problem

The phenomenon observed in the history of image compression is not an isolated incident. In many applications where data must be compressed without a complete knowledge of the source from which the data is drawn from, it is difficult to leverage additional knowledge discovered about the source model *ex post facto*. Attempts to do so either lead to the creation of new standards which are only adapted by a small number of users, or which clutter decoding programs that need to be backwards-compatible with multiple formats.

1.1.3 Types of Compression and Joint Design

To get a better idea of where this problem comes from and where a solution might be found, let us briefly discuss the modern state of data compression. Most compression systems that exist today or are being researched can be divided into one of four domains based on two categories: model specificity and fidelity of reconstruction. Table 1.1 outlines the four categories.

Fidelity/specificity	Model Specific	Model Independent
Lossless	Entropy Coding	Universal Entropy Coding
Lossy	Rate-Distortion Coding	Universal Rate-Distortion Coding

Table 1.1: Types of Data Compression Problems

Lossless compression algorithms achieve compression with prefect reconstruction via entropy coding, while lossy compression uses rate-distortion coding for imperfect compression.

Model specific compression methods use domain knowledge in order to design compression algorithms specially tailored to the data model. Examples of this include image compression algorithms such as JPEG, JPEG2000, and PNG, as well as most audio and video codecs. In general, model specific compression methods use some form of data processing followed by an entropy coding step (e.g. Huffman coding, arithmetic coding), with an additional quantization step for lossy compression that are often integrated with the coding and processing steps. (e.g. EZW coding, Lloyd-max quantization).

Model independent (universal) coding does not require any prior knowledge about the source of the data being compressed, instead relying on universal methods of entropy coding (e.g. LZW coding, CTW coding).

Both model specific and model independent compression have their advantages and disadvantages. Model specific compression is, in general, very effective at compressing the type of data it is designed to compressed, and uses domain knowledge to greatly improve compression performance. However, model specific compression is hampered by the concept of joint design, where the data-processing, quantization, and coding steps must be designed jointly for maximum performance based on the source model. Figure 1-1 illustrates the concept of joint design.

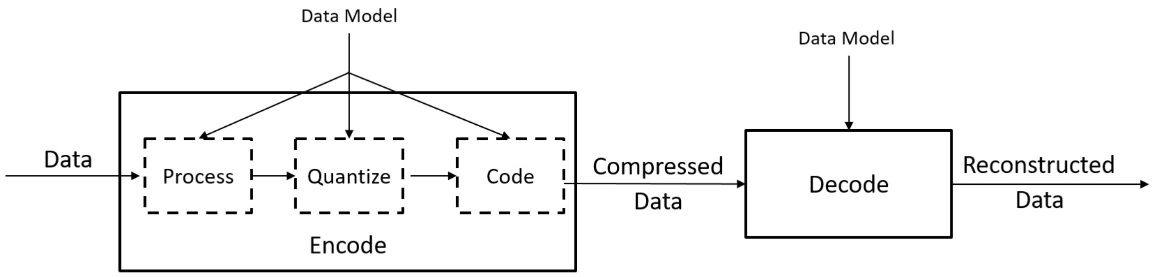


Figure 1-1: The joint design paradigm, where the data model is required in the construction of the encoder, and the processing, quantization, and coding steps are intertwined.

We can see this paradigm in action in JPEG coding, where the pre-processing step of transforming the data into the DCT domain is done because it can be coupled with a quantization procedure based on perceptual models of vision, which allows for effective run-length coding, which in turn can be effectively compressed using Huffman coding. Each part of the system is effective only because the other parts of the system are designed to work specifically with it and the overall source model. This makes the system rigid and makes is often impossible or very difficult to include new domain knowledge into an existing system. In addition, to design model-specific systems, one must already possess the domain knowledge, which means that the data cannot be encoded until enough data has been collected to understand the source model, which is not always optimal.

Model independent coding avoids these issues by using a universal code, which allows for the encoding of data without any prior knowledge of the source. However, universal codes are unable to leverage knowledge of the source model, and for complex models, require very long bitstreams to achieve effective compression.

Ideally, we would like to be able to leverage the benefits of both types of coding. To wit, we would like a universal encoding system that allows for data to be compressed without knowledge of the source model, and would also like a system that is able to integrate varying amounts of domain knowledge in order to improve compression performance.

1.1.4 Image Compression

One particular domain where this type of universal coding would be useful is in the compression of images. Image data tends to be highly structured in complex, non-linear ways and, given the ubiquity of this type of data and its importance in many applications, work is constantly being done to improve our ability to analyze and model images. As discussed in the motivating example, however, much of this work has not be used to benefit image compression systems, with JPEG continuing to dominate decades after its release despite all the advances in the field. With a universal, easily-upgradable system, it would be possible to finally create an image compression algorithm that can leverage current advances in imaging as well as future developments without sacrificing backwards-compatibility.

1.2 Previous Work

1.2.1 The Model-Quantizer-Code Separation Architecture

The problem of eliminating joint design from compression architectures was studied in-depth by Huang and Wornell [17] [16] and later by Lai [21]. Huang and Wornell proposed a source-agnostic universal data compression architecture known as the *Model-Quantizer-Code Separation (MQCS) Architecture.*

This system separates the data modelling aspects of the compression from the coding aspects, which allows for the use of universal codes for compression while still being able to integrate different source models in the decoding step for improved performance. The system also includes an optional quantization component that is independent from both source model and code which allows for lossy compression. An illustration of the separation paradigm can be found in Figure 1-2.

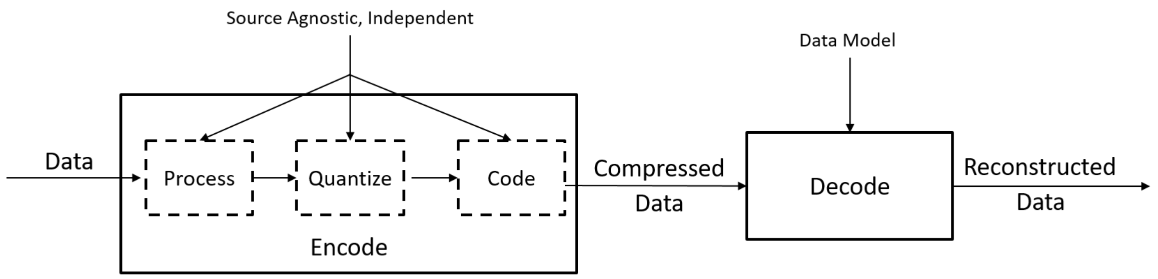


Figure 1-2: The separation paradigm, where the processing, quantization, and coding steps of the encoder are independently designed from one another and from the data model, with the data model used only to facilitate decoding.

The key contribution of the MQCS Architecture is its ability to repurpose any source model which can be formulated as a graphical model and leverage it to improve decoding performance and thus compression performance. A more detailed outline of the functionality of the MQCS Architecture will be given in Chapter 3.

1.2.2 Image Modeling

All image compression algorithms rely on some underlying model of how the image is generated. In this thesis, we focus on natural images; that is, pictures from the real world, which is the same subset that most image compression systems focus on, including JPEG and JPEG2000. Over the course of the past thirty years, many advances have been made in this regard.

The original JPEG standard used an image model whereby low-frequency components of the Discrete Cosine Transform (DCT) of the image were more significant than the high-frequency components [33]. The JPEG2000 standard refined this model with the wavelet transform, allowing some important high-frequency components to be conserved as well [28]. Later models were then developed, though never leveraged for practical image compression applications.

Gaussian and Laplacian image pyramids provide simple hierarchal methods for decomposing an image into component parts, allowing for image analysis and processing at multiple resolution levels [5]. Later, Restricted Boltzmann Machines (RBMs) were developed to model images using mixture models and perform feature detection on them [22]. More recently, significant work has been done in the field of deep learning with regard to the classification and generation of images [7] [27].

Deep Neural Networks have been used extensively for image classification [6], and the idea of stacking multiple layers of nodes have been extended to RBMs to create Deep Belief Networks, which consist of stacks of RBMs [13]. Finally, to provide scalability of these systems, convolutional networks have been developed for both Deep Neural Nets [20] and Deep Belief Nets [24], enforcing certain patterns in the connections between layers to reduce complexity.

1.3 Contributions

In this thesis, we extend the work done by Huang and Wornell and Lai in the development of the MQCS Architecture into practical applications of compression of continuous or large-cardinality alphabet sources. Specifically, we will look at the compression of natural image data. We will show how the MQCS Architecture can be used to compress image data using different source models for both bi-level and greyscale images.

We will begin by modeling the pixels of the image as being drawn from a Gaussian or Gaussian mixture distribution and explore different methods for determining this distribution and how the resulting model can be used in the MQCS Architecture. In particular, we will focus on the Restricted Boltzmann Machine and its variations, showing how these models can connected to the MQCS Architecture and deriving approximate expressions for message-passing in these models. Finally, we will explore the model-adaptive potential of the MQCS Architecture by looking at the compression of 2D and 3D bi-level image data and showing how the MQCS Architecture can be quickly modified with new source models to improve compression performance.

1.4 Thesis Organization

This thesis will cover both the overall design of the MQCS Architecture as well as its implementations on real-world data. In particular, the organization of the thesis will be as follows:

Chapter 2 will provide a brief overview of various basic concepts that serve as the basis of the MQCS Architecture, as well as an overview of the image models that will be used in the thesis. Chapter 3 will provide a description of the MQCS Architecture, including its use as an encoder or decoder for various types of data and how to leverage information about source models to improve compression performance via graphical modeling. Chapter 4 will explore the applications of the MQCS Architecture to the problem of image compression, including discussions of the results of experiments performed on various datasets. Finally, Chapter 5 will conclude the thesis and discuss potential for future work that can be done with the MQCS Architecture.

1.5 Notation

While we have endeavored to use the standard notations and terminology, the multi-disciplinary nature of the system coupled with the fact that some of the fields in question are quite new and developing quickly mean that there will often be multiple competing or contradicting sets of notation that we will encounter. As such, the following list outlines the notation we have chosen for this thesis unless otherwise noted.

- We use lower case s to denote a scalar or vector variable. For a vector variable, we denote the length using superscripts (e.g. s^n) when introducing the variable, with subscript indexing (e.g. s^n_i is the ith element of s^n). We also use the backslash to indicate all other elements in a vector (e.g. $s_{\backslash(i,j)}$ are all elements in s^n except for the ith and jth element).
- We use upper case A to denote a matrix, with size specified by superscripts $A^{n\times m}$ and indexing done using subscripts (e.g. A_{ij} is the (i,j)th entry of A).
- For iterative algorithms, we use bracketed superscripts to denote iteration number ($m^{(\tau)}$ is the value of m at the τth iteration).
- We use $\mathbb{R}$ to denote the set of real numbers and $\mathbb{Z}_p$ to denote the set of integers *mod p*.
- For algorithms, we use doubles slashes // to indicate the beginning of a comment.
- We use $\mathbb{1}\{x\}$ to denote the indicator function, which evaluates to 1 if the statement x is true, and zero otherwise. We also use $exp(x)$ to denote the exponential function e^x.

Chapter 2

Background

The development and implementation of the MQCS Architecture is an undertaking that spans multiple fields of study. The fundamental concepts behind the quantization and compression of data come from the field of information theory, with the actual codes used coming from coding theory. The developments of the decoder and decoding algorithms are drawn from the field of inference and probabilistic modeling. And finally, the models used for compression are mainly drawn from the field of machine learning. In this chapter, we will provide an overview of the key concepts needed for the understanding of the MQCS Architecture, including relevant definitions and results. Chapter 3 will expand on this background, showing how all the pieces discussed can be put together to form the compression architecture.

2.1 Probabilistic Graphical Models

Probabilistic Graphical Models (PGMs) are powerful tools used to model conditional dependencies between variables in a probabilistic distribution. A PGM represents a distribution as a graph, with nodes representing variables and edges representing relations between variables. There are a few types of graphical models, but in this thesis the focus shall be on undirected graphical models and factor graphs.

2.1.1 Undirected Graphical Models

An *undirected graphical model* is an undirected graph $G = (V, E)$ in which each node $v_i \in V$ represents a variable, and each edge $(v_i, v_j) \in E$ represents a conditional dependency. That is, for any two variables:

$$(v_i, v_j) \notin E \implies v_i \perp\!\!\!\perp v_j | v_{\setminus(i,j)} \tag{2.1}$$

By the Hammersley-Clifford theorem [18, Thm 4.2], any strictly positive probability distribution can be represented using this form, and the probability density function can be factored out into a product of functions over the cliques of the graph.

$$p(v) \propto \prod_{C \in \mathcal{C}} \psi_C(v_C) \tag{2.2}$$

Where $\psi_C(v_C)$ is the potential function of clique C and $\mathcal{C}$ is the set of all cliques in G.

One useful subset of undirected graphical models are *pairwise graphical models*. In a pairwise graphical model, the largest clique size is two, and as such the probability density function can be factored into a product of functions of two nodes:

$$p(v) \propto \prod_{(i,j) \in E} \psi_{i,j}(v_i, v_j) \tag{2.3}$$

Pairwise models are especially attractive due to their simplicity and the ease in which inference can be performed over them, as well as the fact that each edge represents a distinct potential function.

2.1.2 Factor Graphs

In addition to undirected graphical models, another PGM of interest is the *Factor Graph (FG)* [18, Sec 4.4]. A factor graph is a bipartite graph $G = (V, F, E)$ with two sets of nodes. Variable nodes $v_1, ..., v_N \in V$ represent variables in the distribution and factor nodes $f_1, ..., f_M \in F$ represent functions. The probability density function of the distribution represented by a factor graph is given by:

$$p(v) \propto \prod_{i=1}^{M} f_i(v_{N(f_i)}) \tag{2.4}$$

Where $N(f_i)$ is the neighborhood of f_i or the set of variable nodes adjacent to f_i.

Both undirected graphical models and factor graphs are useful tools for representing probability distributions, and both will be used in this thesis.

2.1.3 Inference in PGMs: Belief Propagation and Loopy Belief Propagation

Once a model for the probability distribution has been determined, the next step is to perform inference over the model to extract the necessary information from it. In this case, our goal is to determine the marginal distribution of a variable or multiple variables. In the case where the variables are drawn from a discrete alphabet source, marginalization can be computed using a brute-force method by summing over all variables except for the variable being isolated:

$$p(v_i) = \sum_{v_{\setminus i}} p(v_1, ..., v_N) \tag{2.5}$$

For continuous alphabets, the summation would be replaced by integration over the variables. Direct computation of this marginal is exponential in the number of nodes, and is thus impractical for the purposes of this thesis. As such, approximate methods must instead be used. One such method is *Loopy Belief Propagation (LBP)*, also known as *message passing* [18, Sec 10.3]. LBP is an iterative algorithm in which each node sends a "message" along its edges to its adjacent nodes, with these messages updated each iteration. Messages are local functions of the incoming messages into the node from the previous iteration of the algorithm as well as any potential functions between the node and its adjacent nodes. The LBP algorithm for factor graphs and pairwise models are given in Algorithms 1 and 2. Note that since each message is only a function of the messages into the source node, the potential of the source and destination nodes, and the potential between the source and destination nodes, this algorithm can easily be expanded to graphical models that contain a mixture of factor graphs and pairwise models by simply using the appropriate message computation for each pair of node.

Algorithm 1: Loopy Belief Propagation for Factor Graphs

```
Input: Graph $G = (V, F, E)$
Output: Marginal probabilities $\hat{p}_i(v_i)$ for all nodes $v_i$ in $V$
Initialize messages $m^{(0)}_{i \to j}(v_j)\ \forall (i,j) \in E$ to some nonzero values.
$n \leftarrow 0$
repeat
    $n \leftarrow n + 1$
    for all edges $(i,j) \in E$ do
        if $i \in V$ and $j \in F$ then
            $m^{(n)}_{i \to j}(v_i) \leftarrow \prod_{k \in N(i) \setminus j} m^{(n-1)}_{k \to i}(v_i)$ //Variable to Factor message update
        else if $i \in F$ and $j \in V$ then
            $m^{(n)}_{i \to j}(v_j) \leftarrow \sum_{v_{k \in N(i) \setminus j}} \left( f_i(v_{N(i)}) \prod_{k \in N(i) \setminus j} m^{(n-1)}_{k \to i}(v_k) \right)$ //Factor to
             Variable message update
    end
until convergence of messages;
//Combine final messages and compute marginal probabilities
for all variables $v_i \in V$ do
    $\hat{p}_i(v_i) \leftarrow \prod_{j \in N(i)} m^{(n)}_{j \to i}(v_i)$
    Normalize $\hat{p}_i(v_i)$ to sum to one
end
return $\hat{p}_i(v_i)\ \forall i \in V$
```

Once again, for continuous-valued variables, summations would be replaced by integration.

Given enough iterations of the computation of these messages, a reasonable approximation of the marginal distribution can be computed from the incoming messages into a node. In addition, if the graph is a tree, then this algorithm reduces

Algorithm 2: Loopy Belief Propagation for Pairwise Undirected Graphical Models

```
Input: Graph $G = (V, E)$ and node and edge potential functions
        $\phi_i(v_i)\ \forall i \in V, \quad \psi_{i,j}(v_i, v_j)\ \forall (i,j) \in E$
Output: Marginal probabilities $\hat{p}_i(v_i)$ for all nodes $v_i$ in $V$
Initialize messages $m^{(0)}_{i\to j}(v_j)\ \forall (i,j) \in E$ to some nonzero values.
$n \leftarrow 0$
repeat
    $n \leftarrow n + 1$
    for all edges $(i,j) \in E$ do
        $m^{(n)}_{i\to j}(v_j) \leftarrow \sum_{v_i} \left( \phi_i(v_i)\psi_{i,j}(v_i, v_j) \prod_{k \in N(i)\setminus j} m^{(n-1)}_{k\to i}(v_i) \right)$
    end
until convergence of messages;
//Combine final messages and compute marginal probabilities
for all variables $v_i \in V$ do
    $\hat{p}_i(v_i) \leftarrow \prod_{j \in N(i)} m^{(n)}_{j\to i}(v_i)$
    Normalize $\hat{p}_i(v_i)$ to sum to one
end
return $\hat{p}_i(v_i)\ \forall i \in V$
```

to the *Belief Propagation* or *Sum-Product algorithm*, and converges in a number of iterations equal to the width of the tree minus one.

The time complexity of one iteration of LBP is $\mathcal{O}(|E||\mathcal{X}|^{d^*})$ for Factor Graphs and $\mathcal{O}(|E||\mathcal{X}|^2)$ for Pairwise Models, where $|\mathcal{X}|$ is the alphabet size and d^* is the maximum degree of any factor node in the graph [18, Sec 10.3]. Thus, with a reasonable bound on the degrees of the factor nodes and a source model that converges reasonably quickly (such as the ones used in this thesis), LBP can be computed very efficiently compared to the exponential time of exact inference.

2.2 Low Density Parity Check Codes

Low Density Parity Check (LDPC) codes were originally developed by Gallager for use in channel coding. LDPC codes are examples of *error-correcting linear codes*, which use parity check bits to correct for corruption in the codewords [9].

In the binary case, a linear code maps a source sequence of length $(n-k)$ into a codeword of length n by adding k parity check bits to the sequence, where $k < n$. These codes are defined by a linear transform

$$L : \mathbb{Z}_2^{n-k} \to \mathbb{Z}_2^{n} \tag{2.6}$$

This transform may be characterized by a *generator matrix* $G \in \mathbb{Z}_2^{(n-k)\times n}$ such that

$$L(s) = G^T s \tag{2.7}$$

An equivalent characterization is the *parity-check matrix* $H \in \mathbb{Z}_2^{k \times n}$ satisfying

$$HG^T = 0 \tag{2.8}$$

From which the parity-check bits x^k can be computed from the codeword s^n by

$$x = Hs \tag{2.9}$$

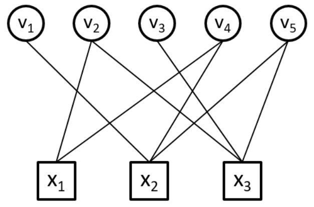
LDPC codes are linear codes with sparse parity check matrices H, that is, the row and column weights of H are negligible as n grows to infinity. LDPC codes have been shown to approach capacity and are thus popular for use in channel coding [15].

2.2.1 Modeling LDPC Codes

LDPC codes can be modeled as bipartite graphs, with one set of nodes representing the initial source sequence, and the other representing the parity check bits [2]. Specifically, we can model LDPC codes as a factor graph, with each bit in the source sequence represented by a variable node and each parity-check bit represented by a factor node. An edge exists between a factor node f_i and a variable node v_j if the variable node is involved in the computation of the parity-check bit; that is, the (i, j)th entry of H is non-zero. The functions f_i of the factor nodes impose the parity-check constraints on the source sequence.

$$f_i(v_{N(i)}) = \mathbb{1}\{x_i == \sum_{j \in N(i)} v_j\} \tag{2.10}$$

Where $x = Hv$ are the parity-check bits. Figure 2-1 shows an example of an LDPC code and its associated factor graph.



$$x = Hv$$

$$H = \begin{bmatrix} 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 & 1 \end{bmatrix}$$

Figure 2-1: A factor graph of a LDPC code where $k = 3$ and $n = 5$. The associated factor potential functions are as follows: $f_1(v_2, v_4) = \mathbb{1}\{x_1 == v_2 + v_4\}$, $f_2(v_1, v_4, v_5) = \mathbb{1}\{x_2 == v_1 + v_4 + v_5\}$, $f_3(v_2, v_3, v_5) = \mathbb{1}\{x_3 == v_2 + v_3 + v_5\}$

2.3 Image Models

When attempting to create a suitable model for image data, the first challenge faced is invariably the problem of alphabet size. A truecolor image specifies the color of each pixel using twenty-8four bits, for an alphabet size of $2^{24} \approx 1.68 \times 10^7$ and even greyscale images use eigth bits per pixel, for an alphabet size of $2^8 = 256$. Even with a pairwise model, Message Passing scales quadratically with alphabet size, and the complexity of modeling interactions between individual bits in an image can often blow-up, to say nothing of the fact that differences in how intensities are coded can lead to vastly different relationships between bits.

As such, it is often more useful to model pixels as continuous alphabet sources, and to apply parameterizable continuous probability distributions to model them. This can greatly reduce the complexity of analysis, and in addition, for natural images, continuous-alphabet distributions fit the image data distributions better since natural images are representations of a continuous, real-world phenomena. They also provide a more intuitive understanding than bitwise modeling.

2.3.1 Gaussian Graphical Models

The simplest and easiest to implement a model for continuous image data is to assume that the data is normally distributed and use a *Gaussian Graphical Model* [18, Sec 7.3]. In a Gaussian Graphical Model, we assume that the variables (pixels) $v_1, ..., v_n$ obey a Gaussian distribution $N(\mu, \Sigma)$ with mean $\mu \in \mathbb{R}^n$ and covariance matrix $\Sigma \in \mathbb{R}^{n \times n}$. If we rewrite the distribution in information form $N^{-1}(h, J)$, where $J = \Sigma^{-1}$ and $h = J\mu$, we have the following conditional independence statement:

$$J_{ij} = 0 \iff v_i \perp\!\!\!\perp v_j | v_{\setminus(i,j)} \tag{2.11}$$

From this, we can easily construct a pairwise undirected graphical model for any Gaussian distribution as follows. Let $G = (V, E)$ be the undirected graphical model for the Gaussian distribution p_{v^n} with information form parameters h, J. Then

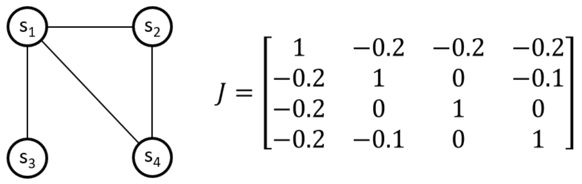
$$(v_i, v_j) \in E \iff J_{ij} \neq 0 \tag{2.12}$$

And the potential functions are

$$\phi_i(v_i) = exp(h_i v_i - \frac{1}{2} J_{ii} v_i^2) \tag{2.13}$$

$$\psi_{i,j}(v_i, v_j) = exp(-\frac{1}{2} J_{ij} v_i v_j), \ \ \forall J_{ij} \neq 0 \tag{2.14}$$

In this thesis, we will only be using pure Gaussian graphical models from independent distributions, and as such all graphs of p_{v^n} will be empty graphs (i.e. have no edges).



s1
s2
s3
s4
$J = \begin{bmatrix} 1 & -0.2 & -0.2 & -0.2 \\ -0.2 & 1 & 0 & -0.1 \\ -0.2 & 0 & 1 & 0 \\ -0.2 & -0.1 & 0 & 1 \end{bmatrix}$

Figure 2-2: A pairwise graphical model representing a Gaussian distribution.

2.4 Image Models

2.4.1 Restricted Boltzmann Machines

In many cases, a purely Gaussian model may be a poor model for image data. One class of models which can capture a more complex class of distributions are *Restricted Boltzmann Machines (RBMs)* [18, Sec 4.4], which have been used extensively as both generative and discriminative models for image classification, feature detection, and denoising [30] [22] [29]. An RBM is a bipartite graph consisting of two sets of variable nodes: visible nodes $v_1, ..., v_n$, which generally correspond to observed data, and hidden nodes $h_1, ..., h_m$, which are used to model latent factors that determine the value of the visible nodes. Since the graph is bipartite, visible nodes are independent of one another conditioned on the hidden nodes and vice-versa, which allows for very easy Gibbs sampling of the model, since one can alternately sample the hidden and visible layers conditioned on the other set until convergence.

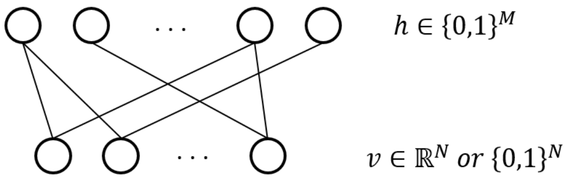


Figure 2-3: A pairwise graphical model representing a Restricted Boltzmann Machine.

There are multiple types of RBMs, but in this thesis we will focus on *Gauss-Bernoulli RBMs (GRBMs)* and *Bernoulli-Bernoulli RBMs (BRBMs)*. In a GRBM, the visible variables are continuous-valued and the hidden variables are binary-valued, with the following distribution:

$$p(v^n, h^m) \propto exp\left(-E((v^n, h^m))\right) \tag{2.15}$$

$$E(v^n, h^m) = \frac{1}{2}\sum_{i=1}^{n}\frac{(v_i - b_i)^2}{\sigma_i^2} - \sum_{j=1}^{m} c_j h_j - \sum_{i,j}^{n,m} W_{ij} v_i h_j \tag{2.16}$$

Where $\sigma^n, b^n, c^m, W^{n\times m}$ are parameters of the model and $E(v^n, h^m)$ is the energy function of the RBM [30]. The conditional distributions of v and h are

$$p(v_i|h) \sim N(\mu_i, \sigma_i^2) \tag{2.17}$$

$$\mu_i = b_i + \sigma_i^2 \sum_j W_{ij} h_j \tag{2.18}$$

$$p(h_j = 1|v) = \frac{1}{1 + exp\left(-\sum_i W_{ij} v_i - c_j\right)} \tag{2.19}$$

That is, the GRBM models a Gaussian Mixture Model with parameters defined by the GRBM parameters and each configuration of hidden nodes creating a different Gaussian distribution.

In a BRBM, the visible variables and the hidden variables are binary-valued, with the following distribution:

$$p(v^n, h^m) \propto exp\left(-E((v^n, h^m))\right) \tag{2.20}$$

$$E(v^n, h^m) = -\sum_{i=1}^{n} b_i v_i - \sum_{j=1}^{m} c_j h_j - \sum_{i,j}^{n,m} W_{ij} v_i h_j \tag{2.21}$$

Where $b^n, c^m, W^{n\times m}$ are parameters of the model. The conditional distributions of v and h are

$$p(v_i = 1|h) = \frac{1}{1 + exp\left(-\sum_j W_{ij} h_j - b_i\right)} \tag{2.22}$$

$$p(h_j = 1|v) = \frac{1}{1 + exp\left(-\sum_i W_{ij} v_i - c_j\right)} \tag{2.23}$$

In this case, hidden variables can be used to model underlying features that affect the values of the visible variables [30].

For both types of models, the standard method for training RBMs is to use a gradient descent method. The update rules for the parameter set $\Theta = (b^n, c^m, W^{n\times m})$ is

$$\Theta^{(t)} = \Theta^{(t-1)} + \eta \frac{dL(\Theta; v)}{d\Theta}\bigg|_{\Theta^{(t-1)}} \tag{2.24}$$

Where η is the learning rate and $L(\Theta; v)$ is the average log-likelihood of the data v and the parameters Θ. Using the Constrastive Divergence technique, we have that

$$\frac{dL(\Theta; v)}{d\Theta} = -\mathbb{E}_{data}(E(v, h, ; \Theta)) + \mathbb{E}_{model}(E(v, h; \Theta)) \tag{2.25}$$

$\mathbb{E}_{data}(\cdot)$ is the expectation of the data over the model, which can be evaluated by taking an empirical average of the energy function over all the training data, and $\mathbb{E}_{model}(\cdot)$ is the expectation over the model, which can be calculated by performing Gibbs sampling on the RBM a finite number of times to produce an "average" realization of the data and then evaluating the expression using that realization [12]. This algorithm can be used to train both BRBMs and GRBMs by using the correct energy functions and parameter sets.

2.4.2 Ising Models

For the compression of bi-level image data, one model which has been shown to be somewhat effective at modeling certain classes of images is the *Ising Model*, an extension of the Markov Chain to higher dimensions used to model interactions between adjacent nodes in space. The homogeneous Ising Model in two dimensions $v^{n \times m}$ is defined over a lattice grid $G = (V, E)$ of size $n \times m$ (see Figure 2-4), with potential functions

$$\phi_i(v_i) = [1 - p; p](v_i) \tag{2.26}$$

$$\psi_{i,j}(v_i, v_j) = \begin{bmatrix} q & 1-q \\ 1-q & q \end{bmatrix} (v_i, v_j), \ \ \forall (i,j) \in E \tag{2.27}$$

Where p and q are model parameters characterizing the individual probabilities of each variable as well as the transition probability between adjacent variables, respectively [18, Sec 4.4]. We also note that this is a pairwise undirected graphical model, which makes message passing simple, with the maximum node degree and the maximum clique size both being very small constants (4 and 2, respectively).

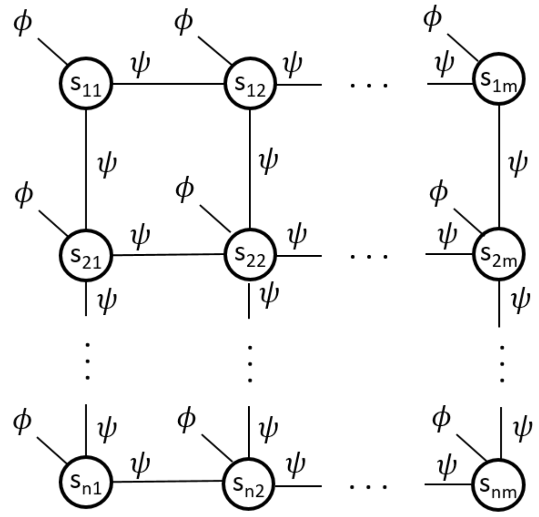


Figure 2-4: An $n \times m$ lattice graph, which is the structure of the pairwise graphical model representing an Ising distribution.

Chapter 3

The Model-Quantizer-Code Separation Architecture

Equipped with the fundamental concepts underlying the MQCS Architecture, we are now prepared to describe its operation. Since the MQCS Architecture is modular, this chapter will show its full construction component by component, from a simple lossless binary compressor to a complete architecture which can be used to compress almost any type of data.

3.1 The Binary Lossless Model-Code Separation Architecture

We begin by looking at the problem of losslessly compressing a stream of binary digits. Without any prior knowledge of where this data stream comes from, we cannot design a specific compression scheme for it, and must thus default to using universal codes such as LZW coding, which is guaranteed to eventually reach entropy but may require unrealistically long bitstreams before achieving a compression rate close to the best possible rate. On the other hand, if we know the source of the data, we can use more efficient coding techniques such as Huffman Coding. However, for most of these techniques, if more information is ever gained about the source, the entire compression system must be redesigned from scratch.

To solve this issue of flexibility, the *Model-Code Separation (MCS) architecture* uses LDPC codes for universal compression, followed by Message Passing for decoding. Specifically, the encoder and decoder are designed as follows.

The encoder requires an input sequence $s^n \in \mathbb{Z}_2^n$ to compress and a randomly-generated parity-check matrix $H \in \mathbb{Z}_2^{k \times n}$. The output of the encoder are the parity-check bits:

$$x^k = Hs^n \tag{3.1}$$

x^k comprises the compressed output of the encoder. Thus, we see that the encoder compresses n bits to k bits, for a compression rate of k/n.

The decoder attempts to reconstruct the original sequence s^n given the parity-check bits x^k. Since the mapping from s^n to x^k is many-to-one for a given H, the decoder requires additional information in order to determine the correct reconstruction. This is provided in the form of a source or data model p_{s^n}, which gives the probabilities of each source sequence occurring. p_{s^n}, as a probability distribution, can be modeled using a PGM of some sort and thus, by combining x^k, H, and p_{s^n}, we can construct a single graphical model to model the information present in the decoder (see Figure 3-1).

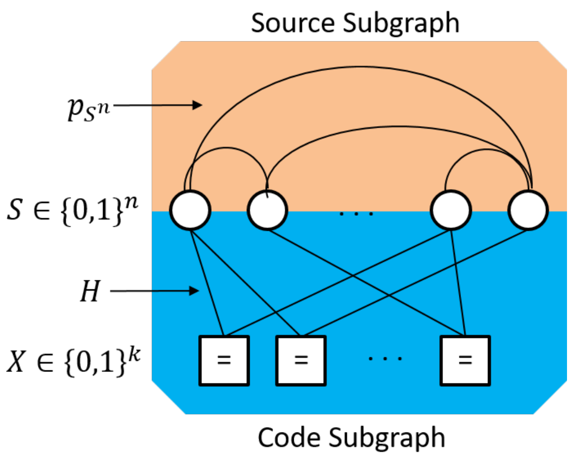


Figure 3-1: The Model-Code-Separation graphical model for binary alphabet sources used by the decoder, containing both the source and code subgraphs

In particular, we can divide the graph into two distinct subgraphs: the code subgraph and the source subgraph. The code subgraph is a factor graph which imposes the parity-check constraints set by H and x^k, forcing the source sequence to hash to the correct parity-check values.

The source subgraph contains the information about the data model given in p_{s^n}. Depending on the form p_{s^n} takes, we may use a factor graph, undirected graphical model, or pairwise graphical model to represent the distribution. In each case, this portion of the graph provides information on which configurations of the source sequence are more likely to occur, which is important in disambiguating between possible reconstructions.

When both subgraphs are combined, we obtain a distribution whose probabilities are nonzero only for certain valid sequences depending on the code graph, and where the likelihood of the valid source sequences differ based on the source graph. Our goal, then, is to determine the source sequence of highest probability which satisfies the parity-check constraints. This can be achieved by running LBP on the graph until convergence, then taking the output marginal probabilities $\hat{p}_i(s_i)$, and outputting the maximal probability sequence

$$\hat{s}_i = \arg\max_{s_i} \hat{p}_i(s_i), \quad i = 1, ..., n \tag{3.2}$$

Given that k/n exceeds the entropy rate of $\hat{p}_i(s_i)$, and with sufficiently well-designed H, each instance of x^k should map to only one typical sequence of s^n [15], and thus the maximal probability sequence $\hat{s}^n$ returned will almost certainly be the correct source sequence, thus achieving perfect reconstruction and thus lossless compression at rate k/n.

3.2 Non-Binary Alphabets: the Complete Model-Code Separation Architecture

While the compressor described in Section 3.1 can achieve universal lossless compression with source-adaptive decoding of binary-alphabet data, it is still unable to handle sequences drawn from larger-alphabet sources. To compensate for this, we introduce a *translator* $T(s^n)$ into our architecture.

Consider a source sequence s^n of length n where each element is drawn from a finite alphabet $\mathcal{X}$ of size p (WLOG, we can assume that $\mathcal{X} = \mathbb{Z}_p$, so that $s^n \in \mathbb{Z}_p^n$). Then the translator is an element-wise operator which maps each element s_i in s^n to a sequence of b bits $z_{bi}, z_{bi+1}, ..., z_{bi+b-1}$. For this thesis, we will assume that p is a power of 2, and thus can be represented with $b = \log p$ bits. Thus, T maps $s^n \in \mathbb{Z}_p^n$ to $z^{nb} \in \mathbb{Z}_2^{nb}$, and

$$(z_{bi}, z_{bi+1}, ..., z_{bi+b-1}) = t(s_i) \tag{3.3}$$

Where $t(s)$ is a single instance of the function $T(s^n)$ applied to a single scalar element. Examples of $t(s)$ include binary codes and Gray codes. Because $T(s^n)$ is an element-wise operator, we can easily model it using a factor graph. Specifically, we can add a series of factor nodes $t_1, t_2, ..., t_n$, with each factor node imposing the translator constraints in a similar fashion to how the factor nodes in the code graph enforce the parity-check constraints. That is, for a factor node t_i connected to s_i and $\{z_{bi}, z_{bi+1}, ..., z_{bi+b-1}\}$,

$$f_{t_i}(s_i, z_{bi}, z_{bi+1}, ..., z_{bi+b-1}) = \mathbb{1}\{t(s_i) == \{z_{bi+k}\}_{k=0}^{b-1}\} \tag{3.4}$$

With the addition of this translator, we can now provide an updated system for the lossless compression of data drawn from any finite alphabet. The encoder now takes in an input sequence $s^n \in \mathbb{Z}_p^n$, passes it through $T(s^n)$ to produce $z^{nb} \in \mathbb{Z}_2^{nb}$. z^{nb} is then passed through the parity-check matrix $H \in \mathbb{Z}_2^{k \times nb}$ to produce the parity-check bits x^k, which is the compressed output of the encoder.

The decoder attempts to reconstruct the original sequence s^n given the parity-check bits x^k in much the same way as in the binary case, except this time there are an extra layer of translator factor nodes to account for, as shown is Figure 3-2. In any case, the decoding algorithm functions the same way, running LBP over the entire graph until convergence, calculating the output marginal probabilities $\hat{p}_i(s_i)$, and outputting the maximal probability sequence

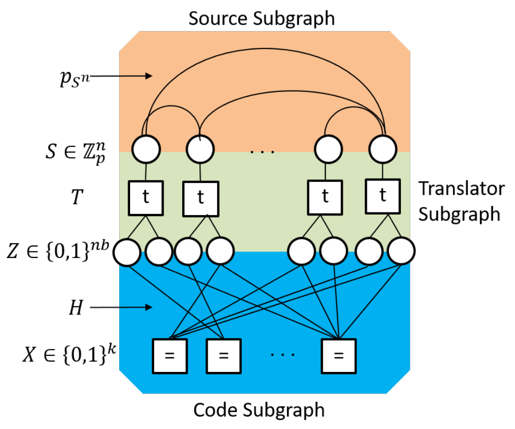


Figure 3-2: The Model-Code-Separation graphical model for non-binary alphabet sources used by the decoder, containing both the source and code subgraphs as well as the translation layer

$$\hat{s}_i = \arg\max_{s_i} \hat{p}_i(s_i), \quad i = 1, ..., n \tag{3.5}$$

Once again, given that k/n exceeds the entropy rate of $\hat{p}_i(s_i)$, and with sufficiently well-designed H and T, each instance of x^k should map to only one typical sequence of s^n, and thus the maximal probability sequence $\hat{s}^n$ returned will almost certainly be the correct source sequence, thus achieving perfect reconstruction and thus lossless compression at rate k/n.

This system is thus able to compress any discrete source in a lossless manner with no prior knowledge of the data model required at the encoder. In addition, the decoder is modular and, in the event that a better source model is discovered that better models the source distribution, it can be easily integrated into the decoder to improve compression performance by replacing the existing source graph with the newly-discovered source graph and eliminating parity-check bits to account for the increased compression rate.

3.3 Lossy Compression: the Model-Quantizer-Code Separation Architecture

While the above system is able to universally compress data in a lossless manner, it does not support lossy compression. In order to extend the MCS Architecture to lossy compression, we add one more component to the system: the quantizer.

We define a *quantization function* $Q(s^n)$ that maps the source sequence s^n to a quantized sequence u^m in a many-to-one fashion. In general, the quantization function can be broken down into a set of scalar functions $q_j(s^r)$ of a small number of elements of the source sequence. That is:

$$u_j = q_j(s^r_{q_j}), j = 1, ..., m \tag{3.6}$$

Where $s^r_{q_j}$ is a subset of s^n and $r \ll n$. Depending on the choice of $Q(s^n)$, we can see how this quantization function can achieve lossy encoding. And then, by applying an appropriate translator T to u^m we can produce a binary sequence z^{mb}, which can then be used as the input to the parity-check matrix H to produce a compressed output x^k, we can achieve lossy compression.

In order to decode, we once again model the entire system as a PGM and run message passing to determine an estimate $\hat{s^n}$ of the source sequence which maps to the given parity check bits. To model the quanitzer, we add an extra layer of nodes Q between the translator T and the source model p_{s^n}, which consists of a set of factor nodes $q_1, ...q_m$ that impose the quantizer constraints of the system

$$f_{q_i}(s^r_{q_i}, u_i) = \mathbb{1}\{q_i(s^r_{q_i}) == u_i\} \tag{3.7}$$

For the case of binary lossy compression, one choice of $Q(s^n)$ that has been shown to be effective is the *Low Density Hashing Quantizer (LDHQ)* [8]. The LDHQ $Q(s^n)$ consists of a set of independent functions $q_1(s^r_1), ..., q_m(s^r_m)$, with each s^r_i consisting of a random subset of s^n of size r. The encoding function is a geometric hash

$$q_i(s^r_i) = \mathbb{1}\{\delta(s^r_i, v^r_i) \leq \frac{r}{2}\} \tag{3.8}$$

Where $\delta(u, v)$ is the Hamming distance between u and v and v^r_i is a randomly drawn binary vector of length r. By setting $m < n$, we can achieve lossy compression at a rate $\frac{n}{k}$ and a distortion determined by m.

For the case of compressing a real-valued source sequence, we have shown in joint work with Lai [21] that the uniform quantizer is an effective model-free quantizer. The uniform quantization function is an element-wise quantizer which, for each element of the source sequence s_i, outputs a uniformly quantized output

$$u_i = \lfloor \frac{s_i}{w} \rfloor \tag{3.9}$$

Where $\lfloor x \rfloor$ is the floor function (the largest integer smaller than x), and w is a width parameter used to control the rate and distortion of the reconstruction, with larger w corresponding to lower rate and higher distortion [10, Sec 3.2]. This quantizer has been shown to be effective for compressing iid and markov Gaussian sources [21]. An example of the complete decoder graph for the case of the uniform quantizer can be found in Figure 3-3.

3.4 Benefits of the Separation Architecture

Having described the architecture we are using for compression, we can now list the benefits provided by this system.

- *Model-independent encoding*: The MQCS Architecture requires only the source

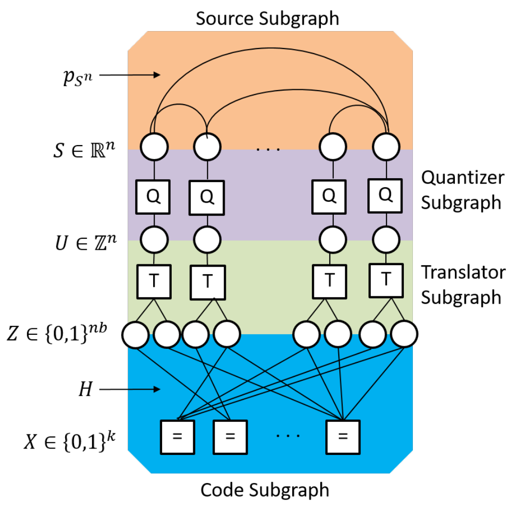


Figure 3-3: The Model-Code-Quantizer-Separation graphical model with uniform quantization for continuous alphabet sources used by the decoder, containing both the source and code subgraphs as well as the translation and quantization layers

sequence, parity-check matrix, and, if necessary, the translator and quantization functions in order to achieve compression. The parity-check matrices can be generated independently of the source, and the translator and quantizer require only knowledge of the source's alphabet size to design. Thus, we see that the MQCS Architecture is able to compress any type of data without knowledge of the source model, which is useful in applications for which data may be generated from an unknown system, such as experimental data from an experiment for which there has not yet been a reasonable hypothesis for how the data is produced.

- *Model-adaptive decoding*: While the encoder does not require any knowledge of the source model, the decoder does require this knowledge in order to complete the message-passing graph and disambiguate different source sequences that map to the same parity-check bits. However, since the source model is only required in the construction of the source graph, it can be easily be swapped out with a different source graph to better reflect the source model. This allows for easy upgrading of the system to include improved knowledge about the source model to improve decoder performance. The increase in decoder performance can allow us to discard parity-check bits, thus allowing us to store the data in a smaller-sized file and thus realize a better compression rate. In addition, it also allows for a single decoder to decode multiple sources of data, simply by using the appropriate source graph for each sequence to decode.

- *Modularity and flexibility*: The separated natured of the encoding and decoding systems also makes it much easier to redesign parts of the system. For example, if we are currently using a gray code to encode the source alphabet, and wish to switch to a different binary code, we can simply swap out the translator function in the encoder and decoder without altering any other part of the system. Or, if a better class of codes arise that outperforms LDPC Codes for compression, the code graph can easily be updated to use the new code. Thus, we never have to worry about "locking in" any part of the compression system, and we can quickly alter the encoder or decoder to work with different types of data or different variations on the system without altering the core algorithms being implemented.

- *Robustness*: As discussed by Lai [21], the MQCS Architecture also provides robustness to errors, since each parity-check bit is independent of the others. By adding more parity-check bits (i.e. decreasing the compression rate), we can allow for some of these bits to be corrupted while still correctly decoding the message with near certainty.

- *Speed and efficiency*: Compared to other compression methods, the compression of binary sources in this architecture is very fast, requiring only multiplication by a sparse matrix, which can be performed in linear time on a CPU or even faster on a dedicated chip with support for LDPC Codes. Depending on the choices of translator and quantizer functions, compressing a source sequence can be done extremely quickly, and since message-passing is a series of local updates, it can be computed in a distributed fashion for very rapid decoding as well.

3.5 Practical Considerations

Now that we have described the theoretical operations of the MQCS Architecture, there are some practical considerations that need to be discussed which will affect the implementation of this system in code.

3.5.1 Doping

As discussed in [15], often times the decoder cannot converge without some non-trivial initialization of the messages. One method of ensuring a good initialization is doping, in which a subset of the bits (the *doped bits*) are transmitted uncompressed, thus initializing the probabilities of these bits to zero or one. This provides a few points to "anchor" the message-passing algorithm and ensure that it converges to the correct reconstruction. We define the *doping rate* r_{dope} as the fraction of uncompressed bits sent.

$$r_{dope} = \frac{\#\quad of\quad dope\quad bits\quad transmitted}{total\quad \#\quad of\quad uncompressed\quad bits} \tag{3.10}$$

In general, different models require different doping rates, with more complex models requiring more doped bits. Practically, we can run the encoder and decoder using different rates to determine empirically what doping rate is optimal for convergence. The actual compression rate r_{total} is thus

$$r_{total} = r_{dope} + r_{code} \tag{3.11}$$

Where r_{code} is the code rate, or the fraction of parity-check bits to uncompressed bits.

3.5.2 Threshold Rates

When running the encoder/decoder at different rates, we observe three regimes in the decoder, summarized in Figure 3-4.

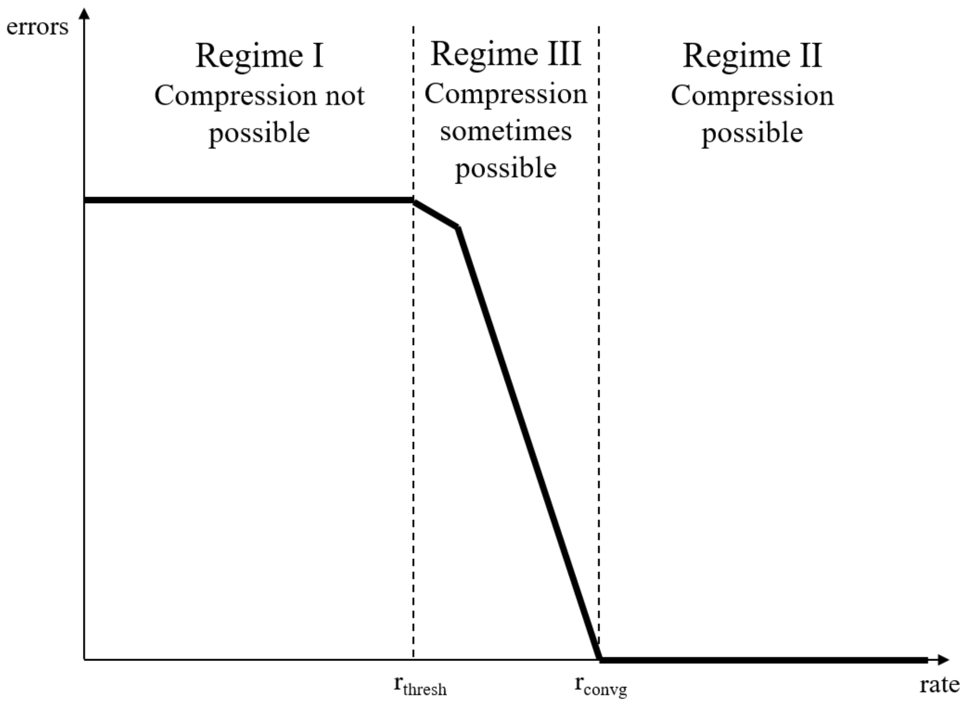


Figure 3-4: Plot of decoding errors as a function of compression rate illustrating the three regimes of operation in a typical application of the MQCS system.

- In the first regime, at low rates, LBP does not converge or has a high probability of converging to an incorrect reconstruction, due to not having enough parity-check bits to properly disambiguate between different possible reconstructions, and thus compression is not possible at these rate.
- In the second regime, at high rates, the system possesses a sufficient number of parity-check bits and LBP almost certainly converges to the correct reconstruction, thus making compression possible at these rates.
- The third regime is a transitional phase between the low-rate regime and the high-rate regime, in which some but not all instances of the system can be decoded correctly, due to variations in source sequences or parity-check matrix

patterns at a certain rate. In this regime, compression is sometimes possible, and with better codes or algorithm implementations, it is possible that compression could almost certainly be possible at these rates.

From these observations, we observe two rates of interest:

- r_{thresh}, the threshold rate at which compression first becomes possible (the transitional rate between the first and third regime). Practically, we define it as the point at which we observe a 10% success rate in compression.
- r_{convg}, the rate at which compression is almost always possible (the transitional rate between the third and second regime). Practically, we define it as the point at which we observe a 90% success rate in compression.

For the purposes of this thesis, we will be using r_{thresh} as the compression rate, as it reflects the potential ability of the system to compress data at a certain rate [15].

3.6 Summary

In this chapter, we have described the construction and operation of the MQCS Architecture for the compression of data. We have explained each component of the system and how to adapt the system to compress any type of data drawn from any source alphabet in either a lossy or a lossless fashion. We have also described the benefits of the system, which motivate our reasons for pursuing the investigation of this system. Finally, we touch on a few implementation details which will later be relevant. In the next chapter, we will explore the application of this architecture to the compression of image data.

Chapter 4

Image Compression

Now that we have described the MQCS Architecture and summarized its features and previous uses, we are now prepared to explore the usage of this architecture in practical applications. Specifically, we focus on one common use of data compression systems: image compression. Natural images tend to have very complex properties which also allow for a high degree of redundancy and thus can be greatly compressed. Furthermore, as discussed in Chapter 1, image compression systems tend to be very "locked-in," with almost no ability to adapt to new developments in image modeling, of which there are many.

4.1 Bi-Level Image Compression

We begin our investigation by looking at image data drawn from a small finite alphabet, in this case a binary alphabet where pixels can only take on a value of 0 (black) or 1 (white). Huang [15] has already shown that the 2D homogeneous Ising model is somewhat effective as a source model for compressing binary images. We extend this work by investigating an alternative source model which better captures the source model for images: the Restricted Boltzmann Machine.

4.1.1 RBMs for Bi-Level Image Compression

The RBM is a natural choice of a model to use for images, as previous work has shown it to be an effective generative model for images, with its ability to capture features of images with its latent nodes [30]. Since the pixels are binary valued, we use the BRBM as a source model for the images.

4.1.2 Message Passing in BRBMs

In order to use the BRBM in the MQCS Architecture, we must first develop the message passing equations for the model. Recall that for a BRBM with visible nodes v^n, hidden nodes h^m, and parameters $b^n, c^m, W^{n \times m}$, the associated probability distribution is

$$p(v^n, h^m) \propto exp\left(\sum_{i=1}^{n} b_i v_i + \sum_{j=1}^{m} c_j h_j + \sum_{i,j}^{n,m} W_{ij} v_i h_j\right) \tag{4.1}$$

From which we can derive the potential functions

$$\phi_{v_i}(v_i) = exp(b_i v_i), \quad i = 1, ..., n \tag{4.2}$$

$$\phi_{h_j}(h_j) = exp(c_j h_j), \quad j = 1, ..., m \tag{4.3}$$

$$\psi_{v_i,h_j}(v_i, h_j) = exp(W_{ij} v_i h_j), \quad i = 1, ...n, j = 1, ..., m \tag{4.4}$$

Using these potentials, we can easily evaluate the message passing equation

$$m^{(t)}_{v_i \to h_j}(h_j) = \sum_{v_i}\left(\phi_{v_i}(v_i)\psi_{v_i,h_j}(v_i, h_j) \prod_{h_k \in N(v_i) \setminus h_j} m^{(t-1)}_{h_k \to v_i}(v_i)\right) \tag{4.5}$$

By inserting the appropriate potential functions, explicitly evaluating the summations, and normalizing the messages, we obtain

$$m^{(t)}_{v_i \to h_j}(h_j) = \frac{1 + e^{W_{ij}} p_{v_i \to h_j}}{2 + (1 + e^{W_{ij}}) p_{v_i \to h_j}} \tag{4.6}$$

$$p_{v_i \to h_j} = exp\left(b_i + \sum_{k \in N(v_i) \setminus h_j} log\left(\frac{m^{(t-1)}_{k \to i}(1)}{m^{(t-1)}_{k \to i}(0)}\right)\right) \tag{4.7}$$

Similarly, the message passing equation from the hidden to the visible nodes is

$$m^{(t)}_{h_i \to v_j}(h_j) = \frac{1 + e^{W_{ji}} p_{h_i \to v_j}}{2 + (1 + e^{W_{ji}}) p_{h_i \to v_j}} \tag{4.8}$$

$$p_{h_i \to v_j} = exp\left(c_i + \sum_{k \in N(h_i) \setminus v_j} log\left(\frac{m^{(t-1)}_{k \to i}(1)}{m^{(t-1)}_{k \to i}(0)}\right)\right) \tag{4.9}$$

However, while these are the exact message-passing equations, in practice, they do not perform well in our compression system due to numerical issues, as many of the messages tend to take on values very close to 0 or 1, making it difficult to retain precision even in the log domain as the number of edges grows large.

As such, we use an approximate energy-based method which is much more robust to precision errors. Looking at the energy function for the BRBM, we observe that by conditioning on the visible layer

$$E(h_j | v^n) = -c_j h_j - \sum_{i=1}^{n} W_{ij} v_i h_j \tag{4.10}$$

If we know the independent marginal probabilities of each visible node $p_{v_1}(v_1)$, $p_{v_2}(v_2)$, ..., $p_{v_n}(v_n)$, then by taking a weighted average we can obtain the estimate of the energy of h_j

$$E_{h_j}(h_j) \approx \sum_{v^n} E(h_j|v^n) \prod_{i=1}^{n} p_{v_i}(v_i) = -c_j h_j - \sum_{i=1}^{n} W_{ij} p_{v_i}(1) h_j \tag{4.11}$$

Then, we can compute the marginal probability $p_{h_j}(h_j)$ by noting that an approximation of the probability of a node is given by

$$p(h_j) \propto exp(-E_{h_j}(h_j)) \tag{4.12}$$

And then normalizing to obtain

$$p_{h_j}(h_j = 1) \approx \frac{1}{1 + exp(-E_{h_j}(1))} = \frac{1}{1 + exp(-c_j - \sum_{i=1}^{n} W_{ij} p_{v_i}(1))} \tag{4.13}$$

Similarly, for the visible nodes

$$p_{v_i}(v_i = 1) \approx \frac{1}{1 + exp(-E_{v_i}(1))} = \frac{1}{1 + exp(-b_i - \sum_{j=1}^{m} W_{ij} p_{h_j}(1))} \tag{4.14}$$

Thus, we have the following approximate message passing algorithm for the BRBM source subgraph given in Algorithm 3.

This algorithm is much more robust and has shown to work much more effectively in empirical tests performed on the MNIST Database (described in the next section).

4.1.3 Compression of the MNIST Database

We demonstrate the effectiveness of the BRBM as a source model for compression by using it to compress data drawn from the MNIST Database, a database of images of handwritten digits of size 28x28 [23]. As a database of images which have clear patterns and shared features but still have a very large set of possible realizations, it is a natural choice for compression using the BRBM.

Experimental Setup

Since the MNIST Database is a greyscale image database, we begin by thresholding the values at the midpoint between black and white to create a binary image database. We then train a model on a subset of the dataset using the Contrastive Divergence method outlined in [12] in order to obtain the model parameters. Once we have our model, we test its effectiveness in the MQCS Architecture by constructing the encoder and decoder for our data source.

The encoder takes in the binary data and applies a randomly-generated LDPC code of some specified size to generate a sequence of parity-check bits which serve as the compressed output of the encoder. We also output a small percentage of the bits uncompressed as doped bits.

n.b. We used code provided in [25] for generating LDPC codes which removes 4-cycles in the code graph. Our previous work [21] has shown that 4-cycles interfere

```
Algorithm 3: Message Passing Updates for BRBM
  Input: Parameters $\Theta = (b^n, c^m, W^{n\times m})$, aggregate messages into subgraph
         from rest of graph $m^{(t-1)}_{G'\to v_1}, m^{(t-1)}_{G'\to v_2}, ..., m^{(t-1)}_{G'\to v_n}$, normalized marginal
         probabilities from previous iterations $p^{(t-1)}_{h_1}(h_1), p^{(t-1)}_{h_2}(h_2), ..., p^{(t-1)}_{h_m}(h_m)$
  Output: Marginal probabilities $p^{(t)}_{h_j}(h_j)$ for all hidden nodes $h_i$ in $h^n$, output
          messages from visible nodes to rest of graph
          $m^{(t)}_{v_1\to G'}, m^{(t)}_{v_2\to G'}, ..., m^{(t)}_{v_n\to G'}$
  //Update visible node probabilities
  for $i = 1, ..., n$ do
      $p_{v_i}(1) \leftarrow \frac{1}{1+exp(-b_i-\sum_{j=1}^{m} W_{ij}p^{(t-1)}_{h_j}(1))}$
      $p_{v_i}(1) \leftarrow \frac{p_{v_i}(1)m^{(t-1)}_{v_i\to G'}(1)}{p_{v_i}(1)m^{(t-1)}_{v_i\to G'}(1)+(1-p_{v_i}(1))(1-m^{(t-1)}_{v_i\to G'}(1))}$
  end
  //Update hidden node probabilities
  for $j = 1, ..., m$ do
      $p^{(t)}_{h_j}(1) \leftarrow \frac{1}{1+exp(-c_j-\sum_{i=1}^{n} W_{ij}p_{v_i}(1))}$
  end
  //Update visible node probabilities again and set them as output messages
  for $i = 1, ..., n$ do
      $m^{(t)}_{v_i\to G'}(1) \leftarrow \frac{1}{1+exp(-b_i-\sum_{j=1}^{m} W_{ij}p^{(t)}_{h_j}(1))}$
  end
  return $p^{(t)}_{h_j}(h_1)$, $j = 1, ..., m$ and $m^{(t)}_{v_i\to G'}(v_i)$, $i = 1, ..., n$
```

with message-passing performance and reduce the effectiveness of compression.

The decoder takes in the parity-check and doped bits, the parity-check matrix, and the BRBM model parameters and uses them to construct a graphical model of the data as in Figure 4-1. We then run loopy belief propagation to decode the original data, using the doped bits to initialize our messages. We run the algorithm for one hundred iterations or until it converges (that is, the marginal probability of any one bit does not change by more than a small threshold value between iterations) and threshold the output probability at 0.5 to obtain an estimated binary reconstruction of the data. In order to determine the compression rate, we run the encoder and decoder for multiple parity-check matrices at a variety of different coding rates and doping rates, with 10 different parity-check matrices tested for each image at each rate. We then extract the threshold and convergent compression rates by looking for the total rates at which the source sequence can first be successfully decoded 1/10 and 10/10 times, respectively.

We compare these values with the same system but with the source model in the decoder replaced with the non-homogeneous Ising Model (where each edge has a different potential instead of sharing a single potential parameter), which has been shown to outperform the general compression algorithm GZIP. We estimate the Ising

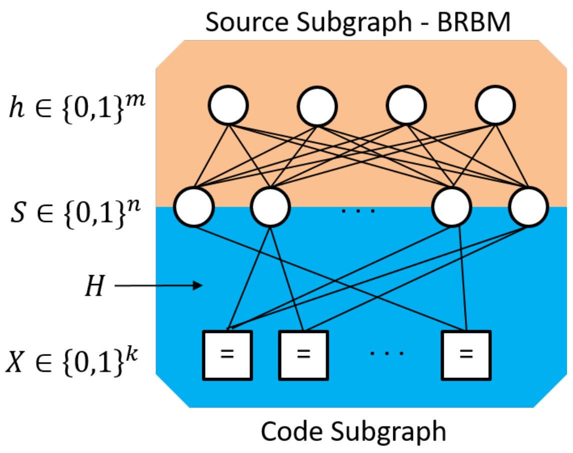


Figure 4-1: Decoding graph with BRBM source subgraph used for compressing MNIST database images.

Model parameters using an approximate Maximum Likelihood estimate based on the empirical ratios of transition pairs to total pairs of adjacent nodes. [32, Sec 6.1].

Results and Discussion

Figure 4-2 shows a comparison of the compression rate achieved for one hundred images randomly sampled from the non-training subset of the MNIST Database. We see that using the BRBM as a source model provides a modest but consistent gain in compression performance, thus showing that the decoder is able to leverage the additional information about the image provided by the features of the BRBM to improve its performance.

We can also see the compression benefits from the additional information provided by the features of the BRBM by comparing compression performance using different BRBM source models with varying quantities of latent variables, as illustrated in Figure 4-3. Up until a saturation point, more hidden nodes improve compression performance, suggesting that the decoder is able to utilize extra information stored in the additional nodes to improve compression performance.

As an aside, we note that since the testing set was separate from the training set, this model can in theory be used to compress an unlimited number of handwritten digit images, and thus the cost of storing the model itself in the memory per image compressed approaches zero.

4.2 Greyscale Image Compression

Having shown the effectiveness of our system for compressing bi-level images, we now move on to more complex images that better describe the types of images encountered commonly in real-life applications. In particular, we look at the compression of natural greyscale images. While not as complex as colored images, greyscale images

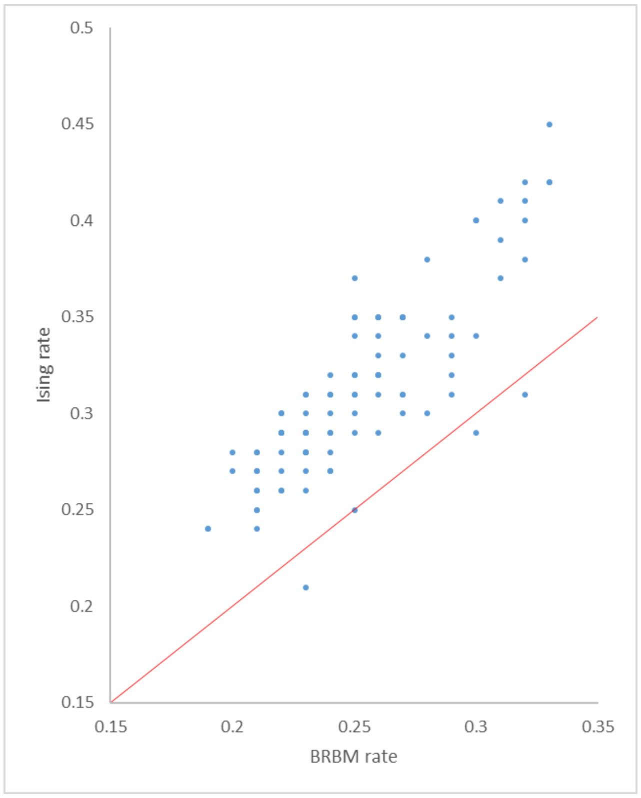


Figure 4-2: Threshold compression rate results of compressing 100 randomly selected images from the MNIST database using the separation architecture under the Ising model and under the BRBM model with 2000 hidden nodes with a dope rate of 0.05. The message-passing decoder was run for a maximum of 100 iterations. Note that the BRBM model almost always provides better compression than the Ising model.

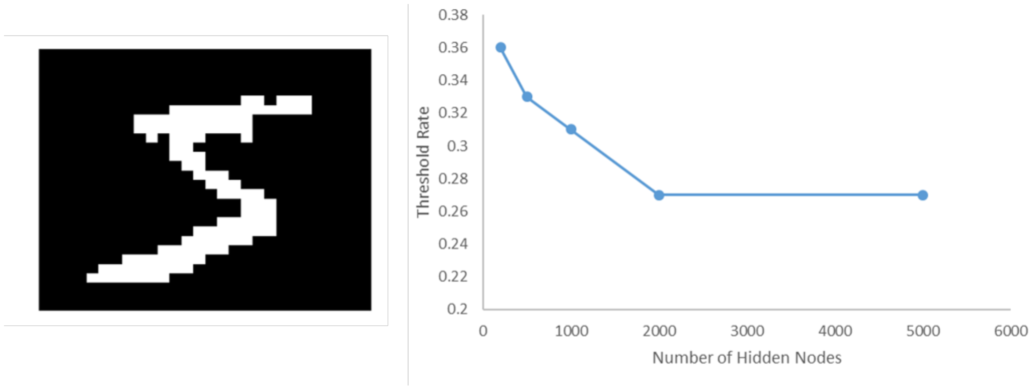


Figure 4-3: Threshold compression rate results (right) of compressing one randomly selected image from the MNIST database (left) under the BRBM model with varying number of hidden nodes with a dope rate of 0.05. The message-passing decoder was run for a maximum of 100 iterations. Note that the BRBM model performs better with more hidden nodes up until a saturation point.

still possess a great deal of complexity and contain richer features than bi-level images. They also allow us to explore the regime of lossy compression, as full-resolution greyscale images tend to require a large amount of memory to store losslessly.

4.2.1 RBMs for Greyscale Image Compression

Once again, in order to capture the complex features of natural greyscale images, we look to the RBM, this time utilizing the GRBM as a source model. While in theory it would be possible to use a BRBM to model the relationships between all the bits used in encoding all the pixels, a GRBM is a far better model to use due to the nearly continuous nature of the pixel distributions and the fact that GRBMs have a much lower complexity compared to modeling each individual bit of the image rather than the pixels.

4.2.2 Message Passing in GRBMs

As with the BRBM, we must first develop the message passing updates for the GRBM before we can use it in our compression scheme. However, unlike the BRBM, the exact message update expressions quickly becomes intractable due to an exponential scaling in time complexity with respect to the number of hidden nodes in message computation.

To see where this scaling arises, we begin by computing the messages from the hidden nodes to the visible nodes. Recall that for a GRBM with visible nodes v^n, hidden nodes h^m, and parameters $b^n, c^m, \sigma^n, W^{n\times m}$, the associated probability distribution is

$$p(v^n, h^m) \propto exp\left(-\frac{1}{2}\sum_{i=1}^{n}\frac{(v_i - b_i)^2}{\sigma_i^2} + \sum_{j=1}^{m} c_j h_j + \sum_{i,j}^{n,m} W_{ij} v_i h_j\right) \tag{4.15}$$

Which gives the following potentials

$$\phi_{v_i}(v_i) = exp\left(-\frac{1}{2}\frac{(v_i - b_i)^2}{\sigma_i^2}\right), \quad i = 1, ..., n \tag{4.16}$$

$$\phi_{h_j}(h_j) = exp(c_j h_j), \quad j = 1, ..., m \tag{4.17}$$

$$\psi_{v_i,h_j}(v_i, h_j) = exp(W_{ij} v_i h_j), \quad i = 1, ...n, j = 1, ..., m \tag{4.18}$$

The associated message passing equation for hidden nodes to visible nodes is:

$$m^{(t)}_{h_j\to v_i}(v_i) = \sum_{h_j}\left(\phi_{h_j}(h_j)\psi_{v_i,h_j}(v_i, h_j)\prod_{v_k\in N(h_j)\backslash v_i} m^{(t-1)}_{v_k\to h_j}(h_j)\right) \tag{4.19}$$

By inserting the appropriate potential functions, explicitly evaluating the summations, and rescaling the messages, we obtain

$$m^{(t)}_{h_j\to v_i}(v_i) = p_{h_j\to v_i}(0) + p_{h_j\to v_i}(1)exp(W_{i,j} v_i) \tag{4.20}$$

Where

$$p_{h_j \to v_i}(x) = \prod_{v_k \in N(h_j) \backslash v_i} m^{(t-1)}_{v_k \to h_j}(x), \quad x \in \{0,1\} \tag{4.21}$$

Using these messages, we can now attempt to compute the message updates from the visible to the hidden nodes.

$$m^{(t)}_{v_i \to h_j}(h_j) = \int\limits_{v_i} \left(\phi_{v_i}(v_i)\psi_{v_i,h_j}(v_i, h_j) \prod_{h_k \in N(v_i) \backslash h_j} m^{(t-1)}_{h_k \to v_i}(v_i) \right)$$
$$= \int\limits_{v_i} \left(exp\left(-\frac{1}{2}\frac{(v_i - b_i)^2}{\sigma_i^2} + W_{ij}v_i h_j \right) \prod_{h_k \in N(v_i) \backslash h_j} (p_{h_k \to v_i}(0) + p_{h_j \to v_i}(1) exp(W_{i,j}v_i)) \right) \tag{4.22}$$

Expanding this expression out yields an exponential number of squared exponential terms, making the message computation intractable. Thus, we instead derive a close approximation of the message passing updates by looking at estimates of marginal probabilities of nodes

We begin by approximating the visible node distributions as Gaussian distributions, an approximations which holds very well when there is a high degree of certainty as to which features are on or off. For the hidden nodes, we note that the probability of a single node is

$$p_{h_j}(h_j) \propto \int_{v^n} exp(-E(h_j, v^n)) = \int_{v^n} exp\left(c_j h_j + \sum_{i=1}^{n} \left(-\frac{1}{2}\frac{(v_i - \mu_{v_i})^2}{\sigma_{v_i}^2} + W_{ij}h_j v_i \right) \right) \tag{4.23}$$

Where μ_{v_i}, $\sigma^2_{v_i}$ are the marginal mean and variance of the visible node v_i. Evaluating this integral for $h_j = 0$ and $h_j = 1$ and then normalizing the terms yields

$$p_{h_j}(h_j = 1) = \frac{1}{1 + exp\left(-b_i - \sum_{i=1}^{n}\left(W_{ij}\mu_{v_i} + W_{ij}^2 \sigma_{v_i}^2\right)\right)} \tag{4.24}$$

For the visible nodes, we can use an analogous energy argument as in the BRBM case. The average energy of a visible node is given by

$$E_{v_i}(v_i) \propto \sum_{h^m} E(v_i|h^m) \prod_{j=1}^{m} p_{h_j}(h_j) = \frac{1}{2}\frac{(v_i - b_i)^2}{\sigma_i^2} - \sum_{j=1}^{m} W_{ij} p_{h_j}(1) v_i \tag{4.25}$$

As in the BRBM, we approximate the distribution of v_i by

$$p(v_i) \propto exp(-E_{h_j}(h_j)) \tag{4.26}$$

By completing the square on the energy function and then extracting the scale

Algorithm 4: Message Passing Updates for GRBM

```
Input: Parameters $\Theta = (b^n, c^m, \sigma^n, W^{n\times m})$, aggregate means and variances of
       the Gaussian messages into subgraph from rest of graph
       $\mu^{(t-1)}_{G'\to v_1}, \mu^{(t-1)}_{G'\to v_2}, ..., \mu^{(t-1)}_{G'\to v_n}, \sigma^{2(t-1)}_{G'\to v_1}, \sigma^{2(t-1)}_{G'\to v_2}, ..., \sigma^{2(t-1)}_{G'\to v_n},$, normalized
       marginal probabilities from previous iterations
       $p^{(t-1)}_{h_1}(h_1), p^{(t-1)}_{h_2}(h_2), ..., p^{(t-1)}_{h_m}(h_m)$
Output: Marginal probabilities $p^{(t)}_{h_j}(h_j)$ for all hidden nodes $h_i$ in $h^n$, output
        means and variances of the Gaussian messages from visible nodes to
        rest of graph $\mu^{(t)}_{v_1\to G'}, \mu^{(t)}_{v_2\to G'}, ..., \mu^{(t)}_{v_n\to G'}, \sigma^{2(t)}_{v_1\to G'}, \sigma^{2(t)}_{v_2\to G'}, ..., \sigma^{2(t)}_{v_n\to G'}$
//Update visible node probabilities
for $i = 1, ..., n$ do
    $\sigma^2_{v_i} \leftarrow \frac{1}{\frac{1}{\sigma^2_{v_i\to G'}} + \frac{1}{\sigma^2_i}}$
    $\mu_{v_i} \leftarrow \left(\frac{\mu_{v_i\to G'}}{\sigma^2_{v_i\to G'}} + \frac{b_i}{\sigma^2_i}\right)\sigma^2_{v_i}$
    $\mu_{v_i} \leftarrow \mu_{v_i} + \sigma^2_{v_i}\sum_{j=1}^m W_{ij} p^{(t-1)}_{h_j}(1)$
end
//Update hidden node probabilities
for $j = 1, ..., m$ do
    $p^{(t)}_{h_j}(1) \leftarrow \frac{1}{1+exp\left(-c_j - \sum_{i=1}^n \left(W_{ij}\mu_{v_i} + W^2_{ij}\sigma^2_{v_i}\right)\right)}$
end
//Update visible node probabilities again and set them as output messages
for $i = 1, ..., n$ do
    $\mu^{(t)}_{v_i\to G'} \leftarrow b_i + \sigma^2_i \sum_{j=1}^m W_{ij} p^{(t-1)}_{h_j}(1)$
    $\sigma^{2(t)}_{v_i\to G'} \leftarrow \sigma^2_{v_i}$
end
return $p^{(t)}_{h_j}(h_1)$, $j = 1, ..., m$ and $\mu^{(t)}_{v_i\to G'}, \sigma^{2(t)}_{v_i\to G'}$, $i = 1, ..., n$
```

factor, we obtain

$$p(v_i) \sim N(v_i; \mu_{v_i}, \sigma_i^2) \tag{4.27}$$

$$\mu_{v_i} = b_i + \sigma_i^2 \sum_{j=1}^{m} W_{ij} p_{h_j}(1) \tag{4.28}$$

Thus, we obtain the message passing algorithm for a GRBM given in Algorithm 4.

4.2.3 Compression of the Cifar-10 Database

We demonstrate the effectiveness of the GRBM as a source model for compression by using it to compress data drawn from the Cifar-10 Database, a database of labeled natural greyscale images of size 32x32 [19].

Figure 4-4: Sample images of the Cifar-10 image dataset.

Experimental Method

As with the MNIST Database, we begin by training the GRBM on a training subset of the images in the dataset to determine the model parameters using Constrastive Divergence. Once we have our model, we test its effectiveness in the MQCS Architecture by constructing the encoder and decoder for our data source.

The encoder takes in the greyscale image data and applies a uniform quantizer to the pixel values with a pre-determined quantizer width (the independent variable in our experiment). The bin labels produced are encoded using a gray code in order to produce a binary sequence, which is then passed through a randomly-generated LDPC code of some specified size to generate a sequence of parity-check bits which serve as the compressed output of the encoder. We also output a small percentage of the gray-coded bits uncompressed as doped bits.

The decoder takes in the parity-check and doped bits, the parity-check matrix, the gray coder and uniform quantizer functions, and the GRBM model parameters and uses them to construct a graphical model of the data as in Figure 4-5. We then run loopy belief propagation to decode the original data, using the doped bits to initialize our messages and approximating messages into the GRBM as Gaussian distributions, as in [21]. We run the algorithm for one hundred iterations or until it converges (that is, the mean value of any one pixel does not change by more than a small threshold value between iterations) and output the mean values of the pixel distributions as an estimated reconstruction of the data. In order to determine the compression rate, we run the encoder and decoder for multiple parity-check matrices at a variety of different coding rates and doping rates, with 10 different parity-check matrices used for each image at each rate. We then extract the threshold and convergent compression rates by looking for the total rates at which the source sequence can first be successfully decoded 1/10 and 10/10 times, respectively.

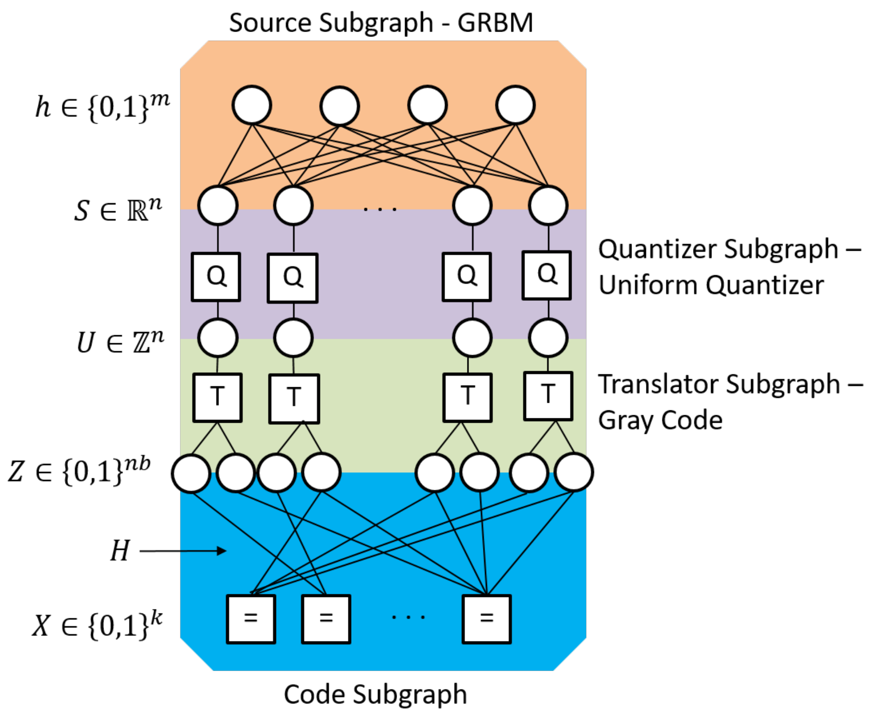


Figure 4-5: Decoding graph with GRBM source subgraph used for compression of Cifar-10 database images.

Once we have a convergent reconstruction, we compare its value to the that of the original image in order to determine the reconstruction MSE. This gives us a (rate, distortion) pair for a specific image given a specific quantizer width. By altering the width of the quantizer, we can produce different pairs and thus generate an estimate for the rate-distortion curve of a single image under a certain model.

We compare our results to MQCS compression using a model where the pixels are distributed as independent Gaussians (using empirical means and variances of the data) as well as with compression using the JPEG standard (that is, by encoding the data as JPEG images using MATLAB's built-in *imwrite()* function at various qualities, then determining the file size and distortions of the resultant images).

Results and Discussion

Figure 4-6 shows a comparison of the average rate-distortion curves achieved for ten images randomly sampled from the non-training subset of the Cifar-10 Database. We see that using the GRBM as a source model provides a consistent gain in compression performance compared to an independent Gaussian model, thus showing that the

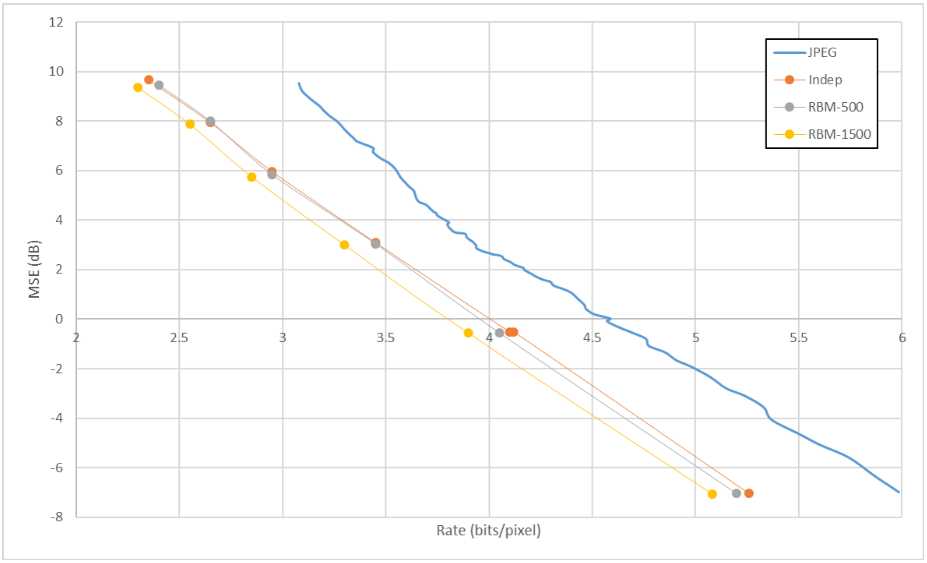


Figure 4-6: Average rate-distortion curve for the lossy compression of ten images drawn randomly from the Cifar-10 database, compressed using the MQCS Architecture with a GRBM source model with 1500 nodes (RBM-1500), a GRBM source model with 500 nodes (RBM-500), a GRBM source model with 1500 nodes (RBM-1500), an independent Gaussian source model (Indep), and compressed using the JPEG algorithm (JPEG). For the separation architecture, we allowed the message-passing algorithm to run up to 100 iterations for the decoder and used an optimal dope rate of 1 bit per pixel.

decoder is able to leverage the additional information about the image provided by the features of the GRBM to improve its performance, especially at higher rates.

We also see the compression benefits from the additional information provided by the features of the GRBM by comparing compression performance using different GRBM source models with varying quantities of latent variables, as illustrated in the same graph. Clearly, adding more hidden nodes up to a saturation point improves compression performance as it provides more features for the GRBM to model the images with, thus providing more information to the system.

As an aside, we note that, as with the MNIST case, since the testing set was separate from the training set, this model can in theory be used to compress an unlimited number of images, and thus the cost of storing the model itself in the memory per image compressed approaches zero. We also note that while we have outperformed the JPEG compression, the JPEG standard is designed to work with larger-sized images and thus could not bring the full benefits of its underlying source model to bear on this trial.

4.3 3D Image Compression

So far, we have focused on the compression of traditional two-dimensional image data. We now illustrate another benefit of the MQCS Architecture by extending the system to the compression of higher-dimensional data. Specifically, we will look at the compression of 3D bi-level image data.

As previously discussed, image compression algorithms are generally designed with a very specific underlying model in mind. In almost all cases, this model is specific to the dimensionality of the data; that is, images compression algorithms designed to compress 2D images (such as JPEG and PNG) can only compress 2D images, and do not support images of a different dimensionality. Since our compression algorithm is model-adaptive with model-independent encoding, it should be much easier to extend our system to compress 3D data in additional to 2D data in a way that allows us to leverage the benefits associated with the third dimension instead of treating the problem as one of compressing a set of independent 2D image "slices," as a purely 2D model of 3D data would do.

4.3.1 Compression of 3D Tomographic Data

We demonstrate our extension by compressing tomographic data. Tomographic imaging is a common imaging technique used to create 3D scans of physical volumes by imaging many 2D "slices" of the volume at various depths, creating a 3D grid of pixel values [11, Sec 2.1]. Specifically, our data is a volume of density measurements generated from a 3D X-ray tomography scan of a rat's femur. This type of image is a good example of a natural 3D image that is used in real-world applications. In order to apply bi-level image compression models to this data, we quantize the image data to one bit, thresholding at the median pixel intensity value to create a bi-level 3D scan showing key major features of the femur.

This data, as with most natural images, exhibits the property of having large, solid "patches" of black and white values, thus making it suitable for compression with the Ising model.

Experimental Method

We attempt to compress a 60x60x40 volume sampled from the center of a 10μm resolution scan of a femur bone of a one-day-old mouse taken from [4]. We compress this data in the same way as in Section 4.1, with a randomly-generated parity-check matrix of the appropriate size.

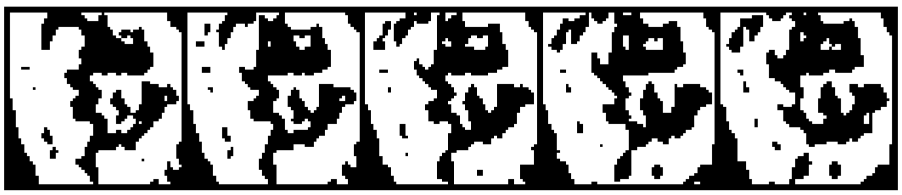


Figure 4-7: Five adjacent slices sampled from the X-ray tomography data used for the 3D compression experiment. Note the strong continuity is the third dimension.

Our decoder also functions similarly, constructing a PGM from the parity-check matrix, parity-check bits, and source graph, and then using message-passing to attempt to reconstruct the original image. To illustrate the added benefits of including a third dimension of correlation, we compare two different source models: a 3D homogeneous Ising Model and a series of 2D homogeneous Ising Models.

For the 3D Ising Model, we assume a homogeneous model similar to the 2D case described in Section 2.4.2, except over a 3D lattice (see Figure 4-8 (left)). The potentials remain the same; that is:

$$\phi_i(v_i) = [1-p; p](v_i) \tag{4.29}$$

$$\psi_{i,j}(v_i, v_j) = \begin{bmatrix} q & 1-q \\ 1-q & q \end{bmatrix}(v_i, v_j), \quad \forall (i,j) \in E \tag{4.30}$$

Where we use ML methods once again to determine the parameters p and q [32, Sec 6.1]. Since the same "patchy" property extends in the third dimension, we believe this extension of the Ising model would also be effective for 3D images of this type.

Our comparison model is a series of independent homogeneous 2D Ising Models, with the width and depth dimensions correlated but the height dimension independent (see Figure 4-8 (right)). For fairness, we assume that all layers share the same parameters, so that both the 2D and 3D models have the same amount of parameter information available to describe the models. We attempt compression of this volume using both these sources, as well as using the GZIP algorithm (using MATLAB's built-in *gzip()* function on a zig-zag scan of the pixels), a universal compression algorithm.

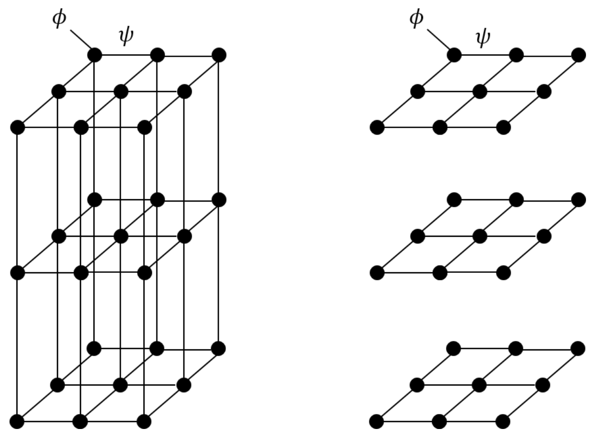


Figure 4-8: A $3 \times 3 \times 3$ three-dimensional lattice graph (left) and three 3×3 two-dimensional lattice graphs stacked on top of each other (right), which are examples of the two structures of the pairwise graphical models representing their respective Ising distributions. Note that in the stacked 2D case, each layer is independent of all other layers.

Results and Discussion

As seen in Table 4.1, the 3D Ising Model produces the best compression, beating out both the 2D Ising Model and the GZIP algorithm. Thus, we have shown for this

Compression Type	**Compression Rate**
MQCS with 3D Ising Model	0.305
MQCS with 2D Ising Model	0.339
GZIP	0.582

Table 4.1: Compression results for 60x60x40 3D tomography image. For the MQCS Architecture, threshold rates were used with a 0.125 doping rate and a maximum of 300 message-passing iterations.

example that the extra information provided by the extra correlation in the third dimension can be leveraged by our architecture to improve compression performance.

However, despite the heavy degree of correlation between pixels in the third dimension ($q \approx 0.92$), the compression gain between the two Ising Models is quite small. The primary reason for this is likely the limitations of the LDPC code when used as a data compressor. Lai [21] has noted that the optimal node degree for each parity-check bit is 3, but at low rates (specifically, rates below $\frac{1}{3}$), it is impossible to construct a code that has an average factor degree of 3 and where each bit to be compressed is connected to at least one parity-check factor node. Thus, as the compression decreases, the average node degree must by necessity increase, thus making the LDPC code less effective [26]. From this data, we believe that the code rate soft limit where the LDPC code begins to severely under-perform for compression is around 0.17 to 0.18.

4.4 Summary and Remarks

In this chapter, we have explored three different realizations of the MQCS Architecture for three different applications. In the first two, we have shown that Restricted Boltzmann Machines are effective source models for compression due to their ability to store information about image features in its latent variables. In the third, we showed how our architecture was able to easily leverage additional dimensional information in order to improve compression performance.

In these three applications, we have illustrated the flexibility and adaptability of this compression architecture. We were able to modify the architecture to compress each data source using only minor modifications or additions to the encoder and decoder, none of which altered the fundamental algorithms used in the system (hashing for encoding and message-passing for decoding). Within each application, we were able to rapidly "upgrade" the system by simply swapping out the old source graph with a newer, more effective one, never needing to change the encoder (such that both the old and new decoder could reconstruct using the same compressed data) or any other part of the decoder outside of the source graph.

There are still many questions that need to be answered about the effectiveness of these models. The Restricted Boltzmann Machine has been shown to be more effective than the Ising Model for compressing images drawn from the MNIST and Cifar-10 databases, but it is difficult to pinpoint the exact properties that make it more effective and, by extension, what types of datasets would be more suited to the RBM compared to the Ising Model. Since RBMs extract global features and Ising Models only model local (adjacent) dependencies, it follows logically that any image in which distant pixels are correlated would benefit greatly from using the RBM over the Ising Model, especially in cases where the pixels in between might not be correlated with one another. For example, a database of faces would not have the "patchy" property that makes Ising Models useful, but distant correlations for features such as eyes or hair could easily be captured using RBMs. More work will need to be done in determining when these models are most useful and ultimately when they can outperform existing compression algorithms.

As for the 3D data, since RBMs do not make any dimensionality assumptions, it should be possible to obtain the same gains in the 3D case as in the 2D one. The primary issue is in the number of visible and hidden nodes such an RBM would require. Since the number of pixels scales with the cube of the length of the 3D volume being compressed instead of the square as in the 2D case, even a small image (such as the 60x60x40 volume used in Section 4.3.1) can have a very large number of pixels, making training a very time-consuming problem. In addition, since the RBM is a complete bipartite graph, the number of edges in the model would be very high, and as such decoding would take much longer and be more prone to numerical errors from the sheer number of computations required. However, this scaling also presents a great opportunity; since 3D images tend to be much larger than 2D images, there is a greater need for effective compression of these types of data. This can be seen in the ease of which these types of data can be shared: while it is relatively simple to download large amounts of 2D images from various databases available online, many 3D image databases are still locked behind permission systems due to the strain of transmitting the large amounts of data associated with 3D images. There is a real need for effective compression of these kinds of data for the sake of better dissemination of information vital in many fields of study, ranging from medicine to geology [4] [3].

Chapter 5

Conclusions and Future Work

We finish this thesis with a brief summary of the work presented, followed by a discussion of possible extensions for our current work. We will then conclude with some remarks about the current state of research in this field.

5.1 Summary

In this thesis, we have presented a number of applications for the Model-Code-Quantizer Separation Architecture. We have described the operation of the MQCS Architecture, including practical considerations affecting the implementation of the system. We then presented three examples which illustrate the benefits of this system, showing how the architecture can quickly be modified to adapt to different data models in order to provide effective, source-adaptive compression of image data. We also demonstrate the effectiveness of the RBM as a compression tool by leveraging its ability to act as a feature-based generative model to store information about the source model and use that information to increase the effectiveness of data compression.

5.2 Future Work

While our examples have illustrated the potential of these systems, there is still much work to be done in realizing this potential. Some of these have already been discussed in Section 4.4, but there are still many other theoretical questions about the effectiveness of the system to be answered, as well as implementation-based optimizations to create more efficient realizations of the architecture. Finally, there are still a wealth of models and data sources to be explored, each offering the opportunity for significant gains in compression.

5.2.1 Quantizer Selection

In the greyscale-image case, we utilized the uniform-quantizer to quantize the data by treating it as a continuous source. While this may be effective, in the lower bit regime uniform quantizers have been shown to perform much worse compared to other methods for compressing Gaussian sources, and it follows that this deficit would be reflected in image compression applications. The main benefit that is provided by the uniform quantizer is that it is universal, and thus can be used for any source without any prior knowledge of the model. There are two approaches to mitigating this loss.

- Using a *Low-Density Hashing Quantizer* - while the Gaussian-based models naturally arise when studying natural images, it is also possible to use the bit representations of the pixels as the source sequence instead of the overall pixel value. Then, the source sequence would be purely binary, allowing us to use the LDHQ as a quantizer, which does not have the gap that the uniform quantizer possesses [15]. To mitigate the issues of complexity associated with modeling individual bits, we can use a sequential decoding method, in which we encode each bit plane separately, limiting the edges of the graphical model so that bits can only be connected other bits in the same bit plane or pixel. Thus, we can decode each bit plane individually, using the known values of the previous bitplanes as source model inputs for decoding the next bit plane [21].

- Using a specialized quantizer - If we are willing to break the Model-Quantizer separation paradigm, we could also use a quantizer more specially designed for compressing images. For example, the JPEG standard defines a DCT-based non-uniform quantizer as part of its encoding scheme [33], while the JPEG2000 uses a wavelet transform followed by EZT coding [28]. By using these pre-processing and quantization schemes, we can better extract important features of images, possibly allowing for more efficient encoding and decoding. Since the JPEG scheme is based on linear transformations and and uniform quantization, it is easy to model in our system and allows for relatively simple message-passing. The JPEG2000 scheme would lend itself better to bit-based models due to the nature of the EZT coding, but could still yield models that expose complex interactions between pixel values which can be leveraged for better coding. However, this would violate the separation of model and quantizer, and forces a "lock-in" of the quantizer, which may be undesirable since better quantizer structures may emerge in time.

5.2.2 Full-Resolution Image Compression

The images compressed in this thesis are all "thumbnail-sized" images, of sizes smaller than 50x50 pixels. In most applications, images tend to be much larger, on the order of hundreds or thousands of pixels wide. At these scales, RBMs begin to become impractical, as they generally require a number of hidden nodes on the same order of magnitude as that of the number of visible nodes, which would results in graphs with trillions of edges, making training and storage of these models incredibly costly in terms of time and memory. There are a few methods for effectively scaling the concepts of RBMs for full-resolution image compression.

- We can split the image into individual "thumbnail-sized" patches and then train the RBM on all the patches instead of all the images. Then, the decoder can decode the image using a model in which each patch is an independent RBM. This provides a great deal of scalability, with model complexity and decode time scaling linearly with image size, but loses the ability to capture relationships between patches.

- We can use a Convolutional Deep Belief Network (CDBN) to model the image. The CDBN is an extension of the RBM to large, spatially-correlated data, providing the same scaling and feature advantages as other types of deep convolutional models [24]. A CDBN consists of multiple stacked layers of "sliding" RBM filters which are much smaller than the image size. These filters relate a patch in the image with a set of hidden nodes, and by overlapping many copies of these filters across the entire image, we can model small features such as edges at any locations in the image in an efficient manner. Stacking multiple layers of these filters on top of one another also allows for deeper feature modeling, though at the cost of increased complexity. These models can be trained using similar Contrastive Divergence techniques as those used for training RBMs [14].

5.3 Concluding Remarks

It is well-known that the amount of data being generated worldwide is growing at an explosively fast rate, demanding a need for storage capacity that physical device technologies cannot keep up with. In order to bridge this gap, it will be necessary to develop better compression algorithms to store the data and make up the difference.

However, it is also the case that the data being created tends to be very structured, but in vastly different ways. While it may be clear that genetic sequencing data will be very different from the text of a novel, even small differences in data sources can result in very different data sets. Images of faces are very different from images of landscapes, for example, despite both being natural images. As we increase the granularity of the partitioning of our data, we find more and more classes of data, each with a unique underlying model. Trying to fit a general model over all these data sources will naturally lead to losses in compression effectiveness, a cost which we have borne without complaint for decades due to a lack of resources that could be devoted to designing a compression algorithm for each type of data and redesigning the system each time a better model arose. If we are to adapt to the rapid growth of data generation, we must acknowledge this cost and work towards mitigating it.

With this new model-free paradigm, we can finally begin to achieve that goal, providing a universal framework in which arbitrarily small classes of data may be compressed by leveraging the maximum amount of domain knowledge available. With our framework making developing new compression methods inexpensive, we may finally be able to treat the world of data not with a wide net designed to catch as many sources as possible, but with a series of precise hooks to perfectly capture the individual models at play. Only then will we able to stay afloat in the age of information.

Bibliography

[1] Michael Archambault. JPEG 2000: The Better Alternative to JPEG That Never Made it Big, Sep 2015.

[2] L Barnault and D Declercq. Fast decoding algorithm for ldpc over gf (2/sup q/). In *Information Theory Workshop, 2003. Proceedings. 2003 IEEE*, pages 70–73. IEEE, 2003.

[3] Doug Blankenship. Fallon forge 3d geologic model, 2016.

[4] Emely L Bortel, Georg N Duda, Stefan Mundlos, Bettina M Willie, Peter Fratzl, and Paul Zaslansky. High resolution 3d laboratory x-ray tomography data of femora from young, 1-14 day old c57bl/6 mice, 2015.

[5] Peter Burt and Edward Adelson. The laplacian pyramid as a compact image code. *IEEE Transactions on communications*, 31(4):532–540, 1983.

[6] Dan Ciregan, Ueli Meier, and Jürgen Schmidhuber. Multi-column deep neural networks for image classification. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pages 3642–3649. IEEE, 2012.

[7] Howard B Demuth, Mark H Beale, Orlando De Jess, and Martin T Hagan. *Neural network design*. Martin Hagan, 2014.

[8] Simone Ercoli, Marco Bertini, and Alberto Del Bimbo. Compact hash codes for efficient visual descriptors retrieval in large scale databases. *arXiv preprint arXiv:1605.02892*, 2016.

[9] Robert Gallager. Low-density parity-check codes. *IRE Transactions on information theory*, 8(1):21–28, 1962.

[10] Robert G Gallager. *Principles of digital communication*, volume 1. Cambridge University Press Cambridge, UK:, 2008.

[11] Gabor T Herman. *Fundamentals of computerized tomography: image reconstruction from projections*. Springer Science & Business Media, 2009.

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

[13] Geoffrey E Hinton. Deep belief networks. *Scholarpedia*, 4(5):5947, 2009.

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

[15] Ying-zong Huang. *Model-Code Separation Architectures for Compression Based on Message-Passing*. PhD thesis, Massachusetts Institute of Technology, 2014.

[16] Ying-zong Huang and Gregory W Wornell. A class of compression systems with model-free encoding. In *Information Theory and Applications Workshop (ITA), 2014*, pages 1–7. IEEE, 2014.

[17] Ying-zong Huang and Gregory W Wornell. Separation architectures for lossy compression. In *Information Theory Workshop (ITW), 2015 IEEE*, pages 1–5. IEEE, 2015.

[18] Daphne Koller and Nir Friedman. *Probabilistic graphical models: principles and techniques*. MIT press, 2009.

[19] Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. Cifar-10 (canadian institute for advanced research). 2009.

[20] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[21] Wai Lok Lai. A Probabilistic Graphical Model Based Data Compression Architecture for Gaussian Sources. Master's thesis, Massachusetts Institute of Technology, 2016.

[22] Hugo Larochelle and Yoshua Bengio. Classification using discriminative restricted boltzmann machines. In *Proceedings of the 25th international conference on Machine learning*, pages 536–543. ACM, 2008.

[23] Yann LeCun and Corinna Cortes. MNIST handwritten digit database. 2010.

[24] Honglak Lee, Roger Grosse, Rajesh Ranganath, and Andrew Y Ng. Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations. In *Proceedings of the 26th annual international conference on machine learning*, pages 609–616. ACM, 2009.

[25] Radford M. Neal. Sparse matrix methods and probabilistic inference algorithms. In *IMA Program on Codes, Systems, and Graphical Models*, 1999.

[26] Hosung Park, Seokbeom Hong, Jong-Seon No, and Dong-Joon Shin. Construction of high-rate regular quasi-cyclic ldpc codes based on cyclic difference families. *IEEE Transactions on Communications*, 61(8):3108–3113, 2013.

[27] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*, 2015.

[28] Athanassios N Skodras, Charilaos A Christopoulos, and Touradj Ebrahimi. Jpeg2000: The upcoming still image compression standard. *Pattern Recognition Letters*, 22(12):1337–1345, 2001.

[29] Nitish Srivastava and Ruslan R Salakhutdinov. Multimodal learning with deep boltzmann machines. In *Advances in neural information processing systems*, pages 2222–2230, 2012.

[30] Yichuan Tang, Ruslan Salakhutdinov, and Geoffrey Hinton. Robust boltzmann machines for recognition and denoising. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pages 2264–2271. IEEE, 2012.

[31] W3 Techs. Usage of JPEG for websites. *Usage Statistics of JPEG for Websites, May 2017*, 2017.

[32] Martin J Wainwright, Michael I Jordan, et al. Graphical models, exponential families, and variational inference. *Foundations and Trends® in Machine Learning*, 1(1–2):1–305, 2008.

[33] Gregory K. Wallace. The jpeg still picture compression standard. *Communications of the ACM*, pages 30–44, 1991.