ATOMIC ENERGY MODELS FOR MACHINE LEARNING: ATOMIC RESTRICTED BOLTZMANN MACHINES

by

Hasari Tosun

A dissertation proposal submitted in partial fulfillment
of the requirements for the degree

of

Doctor of Philosophy

in

Computer Science

MONTANA STATE UNIVERSITY
Bozeman, Montana

September, 2014

TABLE OF CONTENTS

TABLE OF CONTENTS – CONTINUED

ABSTRACT

Restricted Boltzmann Machines (RBM) are energy-based models that are successfully used as generative learning models as well as crucial components of Deep Belief Networks (DBN). The most successful training method to date for RBMs is the Contrastive Divergence method. However, Contrastive Divergence is inefficient when the number of features is very high and the mixing rate of the Gibbs chain is slow.

We develop a new training method that partitions a single RBM into multiple overlapping atomic RBMs. Each partition (RBM) is trained on a section of the input vector. Without an overlap between partitions, we can train RBM partitions in parallel. As a results, this method is significantly faster. In addition to its speed, our preliminary results demonstrate that the resulting model has better performance in terms of its generative power when compared to a regular RBM without partitions. We also experimented with a configuration where partitions have an overlap: each partitioned RBM shares some of hidden and visible nodes with its neighboring RBMs. In this case the generative power of the model increases significantly. The proposed research will evaluate this model in more detail.

First, we intend to evaluate the discriminative power of this model. We plan to develop a discriminative atomic RBM model and apply it to several datasets to determine the classification accuracy. Although RBMs are usually used as a feature extraction component for DBNs, a few extensions have been defined and researched with respect to training RBMs for classification tasks. The proposed research aims to train partitioned RBMs discriminatively for multiclass classification tasks.

Second, we intend to provide a theoretical foundation to explain why this model can represent deep features better. To this end, we examine the model characteristics such as sparseness and overfitting.

Third, we intend to apply our proposed algorithm to multi-modal datasets containing both text and images. We will determine the classification accuracy and the generative performance of the model. Specifically, we plan to investigate whether the text features can be used to successfully reconstruct images or not.

Finally, we intend to investigate how our proposed learning model can be applied to temporal (e.g, time series) and sequential (e.g.,clickstream) datasets. Thus, we aim to evaluate our model on a variety of temporal/sequential data sets. This prospectus reviews preliminary progress on these goals, discusses anticipated contributions of the research, and outlines the direction of future studies for continuation of this doctoral research.

CHAPTER 1
INTRODUCTION

1.1 Motivation

As the volume of data is increasing exponentially, the corresponding need for efficient learning algorithms is also increasing. In addition to the traditional Internet, in the Internet of Thing (IoT) where a variety of smart devices are connected to each other and to the Internet, the volume of generated data is immense. There are some basic characteristics of recent data: 1) it is collected from many sources, 2) it is very high dimensional, 3) it is complex with many latent variables, and 4) it is spatio-temporal. Representing such data efficiently and developing computationally efficient algorithms is a challenging task. Most of the training algorithms for learning are based on gradient descent with data likelihood objective function are intractable [1].

An example of data generated by users and sensors is images. Image pixel resolution is increasing continuously. An image taken by a cheaper camera with 2048×1536 pixels resolution has 3.1 million total pixels. Thus, without any preprocessing, the dimension of the image is 3.1 million. If one were to construct a neural network, the input layer needs to have 3.1 million neurons. If the network had a single hidden layer, also with 3.1 million neurons, then propagating information from one layer to next will involve 9×10^{12} calculations. This is even more prohibitive when deep neural networks with multiple hidden layers are used. Thus, current machine learning methods will not handle such datasets when the number of images are on the order of billions. For example, as of 2010, Google indexed approximately 10 billions images [2].

An immediate solution to overcome the time complexity of training algorithms is to distribute learning on many nodes for processing. However, in current Deep Neural Network (DBN) algorithms, to accomplish an optimization task on multiple machines, a central node is needed for communicating intermediate results. As a result, the communication becomes a bottleneck.

In addition to training and inference time complexity, representation is an issue: what is the best model to effectively represent features? The performance of machine learning methods is dependent on the choice of data representation. Recent studies show a representation that maximizes sparsity has properties of the receptive fields of simple cells in the mammalian primary visual cortex; receptive fields are spatially localized, oriented, and selective to structure at different spatial scales [3]. In a sparse representation, most of the extracted features will be sensitive to variations in data; thus, sparseness is a key component of good representation [4]. Moreover, there is a need for distributed representation where a concept is represented by many neurons and a neuron is involved in many concepts. Energy-based models such as Markov Random Fields, Boltzmann Machines, and Autoencoders have gained significant success. However, these algorithms, although tractable, are slow if data dimensionality is very high.

We recently implemented a partitioned Restricted Boltzmann Machine (RBM) and trained an individual RBM separately [5]. We quickly realized that partitioned RBMs are not only more accurate in terms of their generative power, they are also fast. Our preliminary results shows that they are also very accurate in terms of their discriminative power, namely in a classification task. This leads us to believe that a learning method with many partitioned small RBMs will be more accurate and efficient. Thus, we propose to research a theoretical foundation for atomic RBMs with many configurations to obtain better data representation and better performance.

1.2 Overview

This section describes the organization of the remaining chapters of this proposal and gives a brief overview of the focus of each chapter.

In Chapter 2, we review the background work common to this entire prospectus. We start by discussing learning methods such as Autoencoders and Restricted Boltzmann Machines (RBM). We briefly describe sparsity and work assessing the role of sparsity on the training performance. We also discuss RBMs in the context of Deep Belief Networks (DBN). Finally, we describe our Atomic Restricted Boltzmann Machine (AtomicRBM) algorithm.

In Chapter 3, we describes concepts related to energy-based models. Most of energy-based models rely on the Boltzmann distribution, the partition function, and the free energy. Thus, we describe these concepts and derive energy functions for the Boltzmann distribution. Finally, we describe our main algorithms and models. We provide details on Restricted Boltzmann Machines.

In Chapter 4, we discuss our proposed, special version of the Restricted Boltzmann Machine-AtomicRBM-that we develop as a generative and discriminative model. We provide algorithms for training AtomicRBMs. In the last section of the chapter, we briefly describe the discriminative Atomic RBM.

In Chapter 5, we show some preliminary experiments on a dataset of handwritten digits provided by the National Institute of Standards and Technology (NIST), and a subset of the NIST, the MNIST dataset. We evaluate our algorithm in terms of reconstruction error (generative).

In Chapter 6 we lay out the dissertation plan, describing the high-level goals necessary for completing this doctoral research.

In Chapter 7, we draw conclusions for our current experiments and summarize the expected contributions resulting from this work.

1.3 Research Questions and Anticipated Contributions

In this research, we plan to answer the following questions that will develop into a thesis.

1.3.1 Question I

Does an RBM that is composed of AtomicRBMs learn faster than a single monolithic RBM in terms of generative performance?

1.3.2 Contribution and significance of Question I

As we will show in Section 4, training AtomicRBMs is faster, and individual RBMs can be run in parallel. Thus, if we can demonstrate that AtomicRBMs have the same or better feature representation power as a monolithic RBM, AtomicRBMs will yield two major contributions: 1) a faster algorithm for handling high dimensional data, and 2) a novel algorithm for training RBMs under resource-bounded conditions; when there are limited computational resources, the algorithm can terminate early with a good generative performance. In Chapter 5, our preliminary results demonstrate that AtomicRBMs perform better than the single monolithic RBM in almost all stages of the training. Thus, it can be terminated at any stage with a comparable generative performance.

1.3.3 Question II

Does an RBM that is composed of AtomicRBMs have the same or better feature representation power as a single monolithic RBM in terms of generative and classification performance on multi-modal data?

1.3.4 Contribution and significance of Question II

Since our training algorithm joins multiple small RBMs during its training process, we hypothesize that AtomicRBM will represent multi-modal data better than traditional RBMs. For example, a dataset with images and text can be constructed naturally with multiple AtomicRBMs. In a simple case, an RBM can be composed of two AtomicRBMs, one representing the text and the other representing the image.

1.3.5 Question III

Are there advantages of using a different optimization methods in AtomicRBM as compared to Contrastive Divergence?

1.3.6 Contribution and significance of Question III

As our algorithm creates multiple small RBMs, it opens the door to many other optimization methods that would otherwise have performance implications. For example, particle swarm optimization (PSO) can be used to optimize model parameters for each partition in parallel. Developing other optimization methods for AtomicRBMs constitutes a significant contribution.

1.3.7 Question IV

What are the sparsity characteristics of an AtomicRBM?

1.3.8 Contribution and significance of Question IV

Sparsity is one of the key ingredients in feature representation. A sparse RBM is a model when trained, most of its weights are close to zero, and only a few are either highly positive or highly negative. In a sparse RBM, only a few neurons (hidden nodes) will be active at any given time. This enables the network to learn sparse representations. Researchers often limit the number of neurons that are active for a given input by explicitly adding a sparsity factor in the training process so that most of the weight between visible and hidden nodes are close to zero. However, we hypothesize that AtomicRBMs will have better sparsity characteristics; they will have natural sparsity because each AtomicRBM will be trained on a region of the dataset. Moreover, all of the sparsity metrics defined in the literature are defined for a single model. Defining and developing a sparsity measure for AtomicRBMs constitute a significant contribution.

1.3.9 Question V

Can an AtomicRBM be used efficiently to represent temporal data?

1.3.10 Contribution and significance of Question V

A temporal dataset, in addition to thousands of dimensions, has temporal complexity-data needs to be analyzed over a long time window. Thus, the complexity of training algorithms increases significantly due to the sequential nature of the data. In addition to modeling joint probability of hidden and visible layers, we must model the same joint probability for each time window. Moreover, there is a dependency between models in subsequent windows. For example, in order to model EKG signals and detect anomalies, one can use a sliding window to train a different RBM model for each time window (e.g., 15-minutes window). Each RBM will have a dependency on

the RBM in the previous window. As a result, there is a need for efficient generative models for representing temporal features. If an AtomicRBM can represent temporal features with less training time, it will yield a significant contribution in the domain of time series.

1.4 Notation

All notation used throughout this thesis is described in this section.

a, b, c, w	lowercase italic symbols for scalars
$\mathbf{a}, \mathbf{b}, \mathbf{c}, \mathbf{x}$	lowercase bold symbols for vectors
$\mathbf{W}$	uppercase bold symbols for matrices
$x\top$	by default, all vectors a column vectors $x\top$ denotes row vector
x_i	ith element of vector $\mathbf{x}$
$\mathbf{x}_i$	ith row vector of matrix $\mathbf{X}$
$\mathbf{x}_{ij}$	the ith row and jth column of matrix $\mathbf{X}$
$X(w)$, or X	denotes random variables

CHAPTER 2

BACKGROUND WORK

Today, most of the data generated by humans and devices is unlabeled. As a result, unsupervised neural nets recently became more popular. One basic unsupervised neural net is the Autoencoder [6]. An Autoencoder is a feed-forward neural net that predicts its own input. Using reconstruction error, it updates model parameters (weights and biases) to minimize reconstruction error. An Autoencoder often introduces one or more hidden layers that have lower dimensionality than the inputs so that it creates a more efficient code for representation [3]. It is shown by Hinton *et al.* that when hidden layer activations are constrained through sparsity, the the network composed of Autoencoders creates more efficient representations [7]. These types of Autoencoders are known as sparse Autoencoders. Often a corrupted version of the input is fed to the network to reconstruct the original input. These denoising Autoencoders are shown to represent data more accurately [8].

The other well known approach for feature representation is the Restricted Boltzmann Machine (RBM). An RBM is a type of Hopfield net and a restricted version of the general Boltzmann Machines (BM). A BM is a network of visible and hidden stochastic binary units where the network is fully connected. On the other hand, an RBM network is a bi-partite graph where only hidden and visible nodes are coupled. There are no dependencies among visible nodes or among hidden nodes. Hinton *et al.* developed an algorithm to train a BM in 1985 as a parallel network for constraint satisfaction [9].

The RBM model was first proposed by Smolensky [10] in 1986. Although training of the RBM was tractable, training was initially inefficient, and RBMs did not gain popularity for seventeen years until Hinton *et al.* developed Contrastive Divergence,

a method based on Gibbs Sampling [11]. Since then, RBMs are used as basic components of deep learning algorithms [12, 13, 6]. RBMs have also been successfully applied to classification tasks [14, 15, 16]. Moreover, RBMs have been applied to many other learning tasks including Collaborative Filtering [17].

As RBMs became popular, research on training them efficiently increased. Gibbs sampling is a Markov chain Monte Carlo (MCMC) sampling algorithm [18] for obtaining a sequence of samples from a posterior distribution. It is an iterative process that samples from a distribution closer to posterior distribution. The graph of states over which the sampling algorithm produces samples is called a "Markov Chain." As we describe in the next section, the Contrastive Divergence algorithm performs Gibbs sampling for k-steps to compute weight updates. However, for each data sample, it creates a new Markov Chain. In other words, the state of the Markov chain is not preserved for subsequent updates. Thus, Tieleman modified the Contrastive Divergence method by making Markov chains persistent [1]. In other words, the Markov chain is not reset for each training example. This has been shown to outperform Contrastive Divergence with one step, *CD-1*, with respect to classification accuracy. However, it does not address the problem of training speed. Brekal *et al.* introduced an algorithm to parallelize training RBMs using parallel Markov chains [19]. They run several Markov chains in parallel and treat this set of chains as a composite chain that generates samples from a distribution.

Generally Autoencoders and RBMs are used as components for deep learning [20, 13, 6, 21, 12, 22]. In other words, RBMs and Autoencoders are used to form a deep neural network model. It was shown that layerwise stacking of RBMs and auto encoders yielded better representation [23, 24]. Training is done layer-wise where each layer of the RBM or Autoencoder is trained individually. In the context of Deep Belief Networks (DBN), the DistBelief model and data parallelization framework was devel-

oped by [25]. Here, the DBN model is partitioned into parts. Overlapping parts then exchange messages. Moreover, models are replicated in different computation nodes and trained on different subsets of data to provide data parallelization. Although this algorithm is related to our work, we are primarily concerned with partitioning a single RBM into multiple small RBMs and combining small RBMs for learning features.

When RBMs are trained in an unsupervised fashion, there is no guarantee that the hidden layer representing learned features will be useful for a classification task. Thus, there is a wealth of research on RBMs in supervised or semi-supervised learning when labeled data is available [26, 27, 28, 16, 15]. Moreover, in addition to other metrics, we will use classification for comparing performance of our algorithms with existing algorithms.

CHAPTER 3
ENERGY MODELS

This chapter presents the theoretical foundation for energy-based models and presents existing energy models. We first present energy equations that are the foundation of *statistical mechanics*. Subsequently, we presents energy-based models inspired by statistical mechanics.

3.1 Boltzmann Distribution

A certain quantity of matter under thermodynamic study or analysis is called a *system*. When the thermodynamic system was viewed as black box, the Austrian physicist Ludwig Boltzmann(1844-1906) thought of a system in terms of atoms and molecules and described entropy (a measure of the number of specific ways in which a thermodynamic system may be arranged) in terms of possible disposition of atoms [29]. Boltzmann characterized thermodynamic systems with possible arrangements of atoms and molecules.

Definition 1. A microstate in statistical mechanics is identified by "a detailed particle-level description of the system." For example, microstates for a system with equal-volume parts and three distinguishable particles has $2^3 = 8$ different microstates as illustrated in Figure 3.1.

Definition 2. A macrostate in statistical mechanics consists of a set of microstates that can be described with a relatively small set of variables. Specifically, a macrostate is a thermodynamic or equilibrium state that can be defined in terms of variables energy (E), volume (V), and particle number (n). For example, Figure 3.1, shows

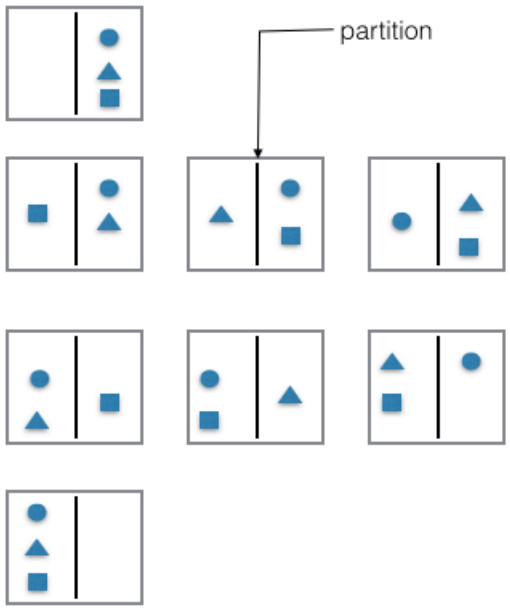

Figure 3.1: Micro states of two-partitions system with three distinguishable particles

a two-sided system where two particles are in the left partition and one particle is on the right. This can be considered as a macrostate, and this macrostate has three microstates.

Boltzmann defined special macrostates by the number of particles n_i that occupy a particular energy level i. These macro states are also called "occupation numbers." A set of occupation numbers, $n_1, n_2..., n_j$ defines a particular macrostate of the system. Suppose we have n copies of a system in a heat bath as show in Figure 3.2 where n is very large. Drawn from n, we want to know how many boxes occupy state i or energy level i. Since there are an infinite number of states, most of these states will have zero assignments. All boxes have the same average energy. Moreover, assume that the total energy is $E_{total} = n \times \bar{E}$ where $\bar{E}$ is the average energy. Thus, the number of ways that occupational numbers can be realized is

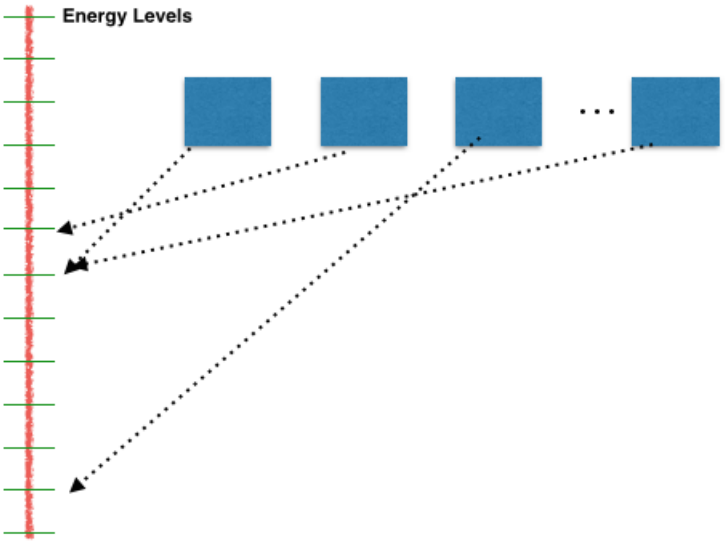

Figure 3.2: Energy levels or occupation numbers of N particles

$$\Omega = \frac{n!}{\prod_{i=1}^{n} n_i}.$$

The probability of any given box will be in state i is given by $p(i) = \frac{n_i}{n}$. Thus, we have following constraints:

$$\sum_{1}^{n} p(i) = 1 \tag{3.1}$$

$$\sum_{1}^{n} n_i E_i = n\bar{E} \tag{3.2}$$

where the average energy can be written in terms of probability as $\bar{E} = \sum_{i=1}^{n} p(i)E_i$. Instead of maximizing Ω, it is easier to maximize the $\log(\Omega)$. Thus, $\log(\Omega) = \log(n!) - \sum_{i=1}^{n} \log(n_i!)$. Using Stirling's approximation, $\log(n!) = n\log(n) - n$, we get

$$\log(\Omega) = -n \sum_{i=1}^{n} p(i) \log(p(i)) \tag{3.3}$$

In fact, $-\sum_{i=1}^{n} p(i)\log(p(i))$ is the entropy of a single system. To optimize (finding minimum or maximum) Equation 3.3 subject to the constraints defined in Equation 3.1 and Equation 3.2 and apply Lagrange Multipliers as $-\sum_{i=1}^{n} p(i)\log(p(i)) - \alpha\sum_{i=1}^{n} p(i) - \beta\sum_{i=1}^{n} n_i E_i = 0$, where α and β are Lagrange multipliers. The probability is obtained as:

$$p(i) = e^{-(1+\alpha)}e^{-\beta E_i}. \tag{3.4}$$

β is defined in terms of temperature as $\frac{1}{k_b T}$ where k_b is the Boltzmann constant and T is temperature. Historically, $e^{-(1+\alpha)} = \frac{1}{Z}$ where Z is the *partition function*. The partition function is the sum over all the microstates of the system. In probability theory, it is used as normalization constant. Using the constraint in Equation 3.1, we rewrite the partition function as:

$$Z(\beta) = \sum_{i=1}^{n} e^{-\beta E_i} \tag{3.5}$$

Thus, the probability can be written as:

$$p(i) = \frac{1}{Z}e^{-\beta E_i} \tag{3.6}$$

This equation is the Boltzmann Distribution. The partition function contains a great deal of information. For example, average energy, $\bar{E}$, can be written as $\bar{E} = -\frac{\partial \log Z}{\partial \beta}$. Following are important equations related to the partition function.

Free Energy: Free energy is "useful" work obtainable from a thermodynamic system at a constant temperature. It is also called Helmholtz Free energy:

$$A = -\frac{\log(Z)}{\beta}$$

Average energy of the system:

$$\bar{E} = -\frac{\partial \log(Z)}{\partial \beta}$$

Entropy of the system:

$$T \times S = E - A$$

$$S = \log(Z) - \beta \frac{\partial \log(Z)}{\partial \beta}$$

3.2 Restricted Boltzmann Machines

As discussed in Section 3.1, in statistical mechanics, the Boltzmann distribution is the probability of a random variable that realizes a particular energy level (Equation 3.6) [29]. In machine learning, β is usually set to 1, except in the context of algorithms such as simulated annealing. In simulated annealing the temperature T controls the evolution of the states of the system . The partition function, Z, is generally intractable to compute. However, when Z is computable, all other properties of the system such as entropy, temperature, etc. can be calculated.

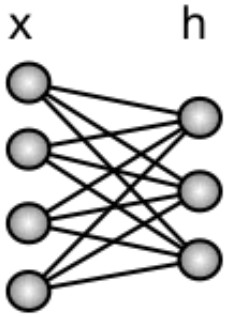

Figure 3.3: Restricted Boltzmann Machine

As a type of Hopfield Network, an RBM is a generative model with visible and hidden nodes as shown in Figure 3.3. There are no dependencies between hidden

nodes, or between visible nodes, thus an RBM forms a bipartite graph. The model represents a Boltzmann energy distribution [29], where the probability distribution of the RBM with visible ($\mathbf{x}$) and hidden nodes ($\mathbf{h}$) is given in Equation 3.7.

$$p(\mathbf{x}, \mathbf{h}) = \frac{\exp(-E(\mathbf{x}, \mathbf{h}))}{Z} \tag{3.7}$$

The conditional probability can be written in terms of the energy function as follows:

$$p(\mathbf{h}|\mathbf{x}) = \frac{\exp(-E(\mathbf{x}, \mathbf{h}))}{\sum_{\mathbf{h}} \exp(-E(\mathbf{x}, \mathbf{h}))} \tag{3.8}$$

The partition function defines configurations over all possible $\mathbf{x}$ and $\mathbf{h}$ vectors

$$Z = \sum_{\mathbf{x},\mathbf{h}} \exp(-E(\mathbf{x}, \mathbf{h})) \tag{3.9}$$

The probability of data $p(\mathbf{x})$ is obtained by marginalizing over the hidden vector $\mathbf{h}$.

$$p(\mathbf{x}) = \sum_{\mathbf{h}} p(\mathbf{x}, \mathbf{h}) = \sum_{\mathbf{h}} \frac{\exp(-E(\mathbf{x}, \mathbf{h}))}{Z} \tag{3.10}$$

The energy function of an RBM is then given as

$$E(\mathbf{x}, \mathbf{h}) = -\mathbf{h}^\top \mathbf{W}\mathbf{x} - \mathbf{b}^\top \mathbf{x} - \mathbf{c}^\top \mathbf{h}$$

Here, $\mathbf{b}$ and $\mathbf{c}$ are bias vectors on visible and hidden layers respectively.

We can rewrite the energy function as a sum over both hidden and visible nodes as:

$$E(\mathbf{x}, \mathbf{h}) = -\sum_{j}^{n} \sum_{k}^{m} h_j w_{jk} x_k - \sum_{k}^{m} b_k x_k - \sum_{j}^{n} c_j h_j.$$

where n is number of hidden nodes and m number of visible(input) nodes. $p(\mathbf{x})$ defined in equation 3.10 can be written as the number of configurations of size n :

$$p(\mathbf{x}) = \sum_{\mathbf{h}\in\{0,1\}^n} \exp(\mathbf{h}^\top \mathbf{W}\mathbf{x} + \mathbf{b}^\top \mathbf{x} + \mathbf{c}^\top \mathbf{h})/Z$$

$$\begin{aligned}
p(\mathbf{x}) &= \sum_{\mathbf{h}\in\{0,1\}^n} \exp(\mathbf{h}^\top \mathbf{W}\mathbf{x} + \mathbf{b}^\top \mathbf{x} + \mathbf{c}^\top \mathbf{h})/Z \\
&= \underbrace{\sum_{h_1\in\{0,1\}} \sum_{h_2\in\{0,1\}} \cdots \sum_{h_n\in\{0,1\}}}_{\mathbf{h}\in\{0,1\}^n} \exp(\mathbf{h}^\top \mathbf{W}\mathbf{x} + \mathbf{b}^\top \mathbf{x} + \mathbf{c}^\top \mathbf{h})/Z \\
&= \sum_{h_1\in\{0,1\}} \sum_{h_2\in\{0,1\}} \cdots \sum_{h_n\in\{0,1\}} \exp(\sum_{j=1}^{n} (h_j \mathbf{w}_j \mathbf{x} + b^\top \mathbf{x} + c_j h_j)/Z \\
&= \sum_{h_1\in\{0,1\}} \sum_{h_2\in\{0,1\}} \cdots \sum_{h_n\in\{0,1\}} \prod_{j=1}^{n} \exp(h_j \mathbf{w}_j \mathbf{x} + b^\top \mathbf{x} + c_j h_j)/Z \\
&= \prod_{j=1}^{n} \sum_{h_j\in\{0,1\}} \exp(h_j \mathbf{w}_j \mathbf{x} + b^\top \mathbf{x} + c_j h_j)/Z
\end{aligned} \tag{3.11}$$

Since $\mathbf{b}^\top \mathbf{x}$ is not dependent on $\mathbf{h}$ and it can be taken out of sum and product:

$$p(\mathbf{x}) = \exp(\mathbf{b}^\top \mathbf{x}) \prod_{j=1}^{n} \sum_{h_j\in\{0,1\}} \exp(h_j \mathbf{w}_j \mathbf{x} + c_j h_j)/Z$$

Replacing $h = 0$ and $h = 1$, we obtain

$$p(\mathbf{x}) = \exp(\mathbf{b}^\top \mathbf{x}) \prod_{j=1}^{n} (1 + \exp(\mathbf{w}_j \mathbf{x} + c_j))/Z$$

Using $\exp(\log(\mathbf{x})) = \mathbf{x}$ mathematical rule, $p(\mathbf{x})$ can be written as:

$$p(\mathbf{x}) = \exp(\mathbf{b}^\top \mathbf{x} + \sum_{j=1}^{n} \log(1 + \exp(\mathbf{w}_j \mathbf{x} + c_j)))/Z$$

Finally, $f(x) = \log(1 + \exp(\mathbf{x}))$ is the $softplus$ function. Writing the above function in term of $softplus$, we obtain the following equation:

$$p(x) = \exp(\mathbf{b}^\top \mathbf{x} + \sum_{j=1}^{n} \text{softplus}(\mathbf{w}_j \mathbf{x} + c_j))/Z \tag{3.12}$$

Inspired from statistical mechanics, the exponential term is $F(\mathbf{x}) = \mathbf{b}^\top \mathbf{x} + \sum_{j=1}^{n} \log(1 + \exp(\mathbf{w}_j \mathbf{x} + c_j))$ is called free energy. Thus,

$$p(\mathbf{x}) = \exp(-F(\mathbf{x}))/Z \tag{3.13}$$

3.2.1 Inference in RBM: conditional probability

Calculating $p(\mathbf{x}, \mathbf{h})$ is not tractable due to the partition function, Z. However, the conditional probability, $p(\mathbf{h}|\mathbf{x}) = p(\mathbf{x}, \mathbf{h}) / \sum_{h\prime} p(\mathbf{x}, \mathbf{h\prime})$, has rather a simple form. To differentiate from $\mathbf{h}$, we use $\mathbf{h\prime}$ to represents all hidden vectors (configurations) of size n. Using equation 3.7, we obtain

$$p(\mathbf{h}|\mathbf{x}) = \frac{\exp(\mathbf{h}^\top \mathbf{W}\mathbf{x} + \mathbf{b}^\top \mathbf{x} + \mathbf{c}^\top \mathbf{h})/Z}{\sum_{h\prime \in \{0,1\}^n} \exp(\mathbf{h\prime}^\top \mathbf{W}\mathbf{x} + \mathbf{b}^\top \mathbf{x} + \mathbf{c}^\top \mathbf{h\prime})/Z}$$

The Z and $\mathbf{b}^\top \mathbf{x}$ values cancels out. If we write equation in explicit sum over all indices, we obtain

$$p(\mathbf{h}|\mathbf{x}) = \frac{\exp(\sum_{j=1}^{n} h_j \mathbf{w}_j \mathbf{x} + c_j h_j)}{\underbrace{\sum_{h\prime_1 \in \{0,1\}} \sum_{h\prime_2 \in \{0,1\}} \cdots \sum_{h\prime_n \in \{0,1\}}}_{\mathbf{h\prime} \in \{0,1\}^n} \exp(\sum_{j=1}^{n} h\prime_j \mathbf{w}_j \mathbf{x} + c_j h\prime_j)}$$

Using $c^{[\sum_{n=s}^{t} f(n)]} = \prod_{n=s}^{t} c^{f(n)}$ rule, we rewrite it as:

$$p(\mathbf{h}|\mathbf{x}) = \frac{\prod_{j=1}^{n} \exp(h_j \mathbf{w}_j \mathbf{x} + c_j h_j)}{\sum_{h\prime_1 \in \{0,1\}} \sum_{h\prime_2 \in \{0,1\}} \cdots \sum_{h\prime_n \in \{0,1\}} \prod_{j=1}^{n} \exp(h\prime_j \mathbf{w}_j \mathbf{x} + c_j h\prime_j)}$$

Further we can rewrite the denominator as product:

$$p(\mathbf{h}|\mathbf{x}) = \frac{\prod_{j=1}^{n} \exp(h_j \mathbf{w}_j \mathbf{x} + c_j h_j)}{\prod_{j=1}^{n} \sum_{h\prime_j \in \{0,1\}} \exp(h\prime_j \mathbf{W}_j \mathbf{x} + c_j h\prime_j)}$$

The denominator can further be simplified using $j = 0$ and $j = 1$:

$$p(\mathbf{h}|\mathbf{x}) = \frac{\prod_{j=1}^{n} \exp(h_j \mathbf{w}_j \mathbf{x} + c_j h_j)}{\prod_{j=1}^{n} (1 + \exp(\mathbf{w}_j \mathbf{x} + c_j))}$$

Using the same product, we obtain

$$p(\mathbf{h}|\mathbf{x}) = \prod_{j=1}^{n} \frac{\exp(h_j \mathbf{w}_j \mathbf{x} + c_j h_j)}{(1 + \exp(\mathbf{w}_j \mathbf{x} + c_j))}$$

It turns out the term inside the product is a probability distribution. Thus, it must be $p(h_j|\mathbf{x})$ because of the Markov property. As result, the equation can be written as follows.

$$p(\mathbf{h}|\mathbf{x}) = \prod_{j=1}^{n} p(h_j|\mathbf{x})$$

Now, we can calculate $p(h_j = 1|\mathbf{x})$ as follows.

$$p(h_j = 1|\mathbf{x}) = \frac{\exp(\mathbf{w}_j \mathbf{x} + c_j)}{(1 + \exp(\mathbf{w}_j \mathbf{x} + c_j))}$$

If we multiply both numerator and denominator by $\exp(-\mathbf{w}_j\mathbf{x} - c_j)$, the conditional probability is simplified as:

$$p(h_j = 1|\mathbf{x}) = \frac{1}{(1 + \exp(-\mathbf{w}_j\mathbf{x} - c_j))}$$

The result is a sigmoid function where $\sigma(x) = \frac{1}{1+e^{-x}}$.

$$p(h_j = 1|\mathbf{x}) = \sigma(\mathbf{w}_j\mathbf{x} + c_j) \tag{3.14}$$

3.2.2 Training RBM: Stochastic Gradient Descent

To train an RBM so that it assigns high probability to data, a negative log-likelihood method is used. Using equation 3.7 the log-likelihood is calculated as follows.

$$\frac{-\partial \log(p(\mathbf{x}))}{\partial\theta} = \frac{\partial}{\partial\theta}\left(-\log\sum_h \frac{\exp(-E(\mathbf{x},\mathbf{h}))}{Z}\right)$$

where θ represents the model parameters (W, b, and c). To derive the gradient, first the log is expanded and Z is replaced with equation 3.9. Then, partial differentiation is carried out.

$$
\begin{aligned}
\frac{-\partial \log(p(x))}{\partial\theta} &= \frac{\partial}{\partial\theta}\left(-\log\sum_h \frac{\exp(-E(\mathbf{x},\mathbf{h}))}{Z}\right) \\
&= -\frac{1}{\partial\theta}\left(\sum_h \exp(-E(\mathbf{x},\mathbf{h})\right) + \frac{1}{\partial\theta}\left(\sum_{h,x} \exp(-E(\mathbf{x},\mathbf{h})\right) \\
&= \frac{1}{\sum_h \exp(-E(\mathbf{x},\mathbf{h}))}\sum_h \exp(-E(\mathbf{x},\mathbf{h})\frac{\partial E(\mathbf{x},\mathbf{h})}{\partial\theta} \\
&- \frac{1}{\sum_{h,x} \exp(-E(\mathbf{x},\mathbf{h}))}\sum_{h,x} \exp(-E(\mathbf{x},\mathbf{h}))\frac{\partial E(\mathbf{x},\mathbf{h})}{\partial\theta} \\
&= \sum_h p(h|x)\frac{\partial E(\mathbf{x},\mathbf{h})}{\partial\theta} - \sum_{h,x} p(\mathbf{x},\mathbf{h})\frac{\partial E(\mathbf{x},\mathbf{h})}{\partial\theta}.
\end{aligned} \tag{3.15}
$$

The first term is the expectation over $\mathbf{h}$, and the negative term is the expectation over both $\mathbf{h}$ and $\mathbf{x}$. Thus, the stochastic gradient becomes

$$
\frac{-\partial \log(p(x))}{\partial\theta} = \underbrace{E_h\left[\frac{\partial E(\mathbf{x},\mathbf{h})}{\partial\theta}|x\right]}_{\text{positive phase}} - \underbrace{E_{h,x}\left[\frac{\partial E(\mathbf{x},\mathbf{h})}{\partial\theta}\right]}_{\text{negative phase}} \tag{3.16}
$$

The gradient contains two terms that are referred as the *positive* and the *negative* terms respectively. The first term increases the probability of the training data by decreasing free energy, while the second term decreases the probability of a sample generated by the model. Computing the expectation over the first term is tractable; however, computing the second term it is not. Thus, Hinton introduced the Contrastive Divergence(CD) algorithm that uses Gibbs sampling to estimate the second term [11].

3.3 Contrastive Divergence

In this section, we describe the Contrastive Divergence (CD) Algorithm, the seminal algorithm for training Restricted Boltzmann Machines. CD provides a reasonable approximation to the likelihood gradient. The CD-1 algorithm (i.e, Contrastive Divergence with one step) is usually sufficient for many applications [1, 22]; however, for CD-1, resetting the Markov chain after each parameter update is inefficient because the model has already changed [1]. Alternatively, Tieleman modified the Contrastive Divergence method by making Markov chains persistent [1]; one more more persistent chains are kept during training.

To calculate updates in CD, we need to do the derivation of the equation 3.16 with respect to weight vector $\mathbf{W}$.

$$\begin{aligned}\frac{E(\mathbf{x},\mathbf{h})}{\partial w_{jk}} &= \frac{\partial}{\partial w_{jk}}\left(-\sum_j\sum_k h_j w_{jk} x_k - \sum_k b_k x_k - \sum_j c_j h_j\right) \\ &= -\frac{\partial}{\partial w_{jk}}\left(\sum_j\sum_k h_j w_{jk} x_k\right) \\ &= h_j x_k\end{aligned} \tag{3.17}$$

We can find the expectation of the positive term in equation 3.16 with respect to W_{jk} as follows:

$$\begin{aligned}E_{\mathbf{h}}\left[\frac{\partial E(\mathbf{x},\mathbf{h})}{\partial w_{jk}}|\mathbf{x}\right] &= E_{\mathbf{h}}\left[-h_j x_k|\mathbf{x}\right] \\ &= \sum_{h_j\in\{0,1\}} -h_j x_k p(h_j|\mathbf{x}) \\ &= -p(h_j=1|\mathbf{x})x_k\end{aligned} \tag{3.18}$$

Since the negative term in equation 3.16 is not tractable to compute in CD, an approximation $\tilde{\mathbf{x}}$ is obtained using Gibbs sampling. Thus, the resulting gradient for CD becomes $-p(h_j = 1|\tilde{\mathbf{x}})\tilde{x_k}$, similar to equation 3.18 for the positive phase. The $\mathbf{W}$ parameter update is carried out as follows:

$$\begin{aligned} w_{jk} &\leftarrow w_{j,k} - \alpha(\frac{E(x,h)}{\partial w_{jk}}) \\ &\leftarrow w_{jk} + \alpha(p(h_j = 1|\mathbf{x})x_k - p(h_j = 1|\tilde{\mathbf{x}})\tilde{x_k}) \end{aligned} \tag{3.19}$$

where $\alpha \in (0, 1)$ is a learning rate. Using the same technique, updates for all parameters take following form:

$$\begin{aligned} w_{jk} &\leftarrow w_{jk} + \alpha(p(h_j = 1|\mathbf{x})x_k - p(h_j = 1|\tilde{\mathbf{x}})\tilde{x_k}) \\ b_j &\leftarrow b_j + \alpha(x_j - \tilde{x_j}) \\ c_j &\leftarrow c_j + \alpha(p(h_j = 1|x) - p(h_j = 1|\tilde{x})) \end{aligned} \tag{3.20}$$

An alternative description of Contrastive Divergence algorithm is given by Bengio [22].

Finally, Algorithm 3.3 shows the pseudocode for training RBMs using the one step Contrastive Divergence method. The algorithm accepts a sample data instance and model parameters; weight vector ($\mathbf{W}$), visible layer bias vector ($\mathbf{b}$), hidden layer bias vector($\mathbf{c}$), learning rate (α).It updates model parameters as follows.

First, in the positive phase (lines 3-4), for all hidden nodes, we calculate the probability of the hidden node given the visible vector. In the negative phase (lines 5-7), for each hidden node, its probability is sampled from the model on line 5. Then, on line 6, for each visible node, its probability is calculated, conditioned on the sample

Algorithm CD-1($\mathbf{x}_i$, α)

1: **Input**: $\mathbf{x}_i$: data sample, α: learning rate.
2: **Model Parameters**: $\mathbf{W}$: weight vector, $\mathbf{b}$: bias vector on visible nodes, $\mathbf{c}$: bias vector on hidden nodes
Notation: $\mathbf{x} \sim p$ means $\mathbf{x}$ is sampled from p
Positive Phase:
3: $\mathbf{x}^0 \leftarrow \mathbf{x_i}$
4: $\mathbf{h}^0 \leftarrow \sigma(\mathbf{c} + \mathbf{W}\mathbf{x})$
Negative Phase:
5: $\tilde{\mathbf{h}}^{\mathbf{0}} \sim p(\mathbf{h}|\mathbf{x}^{\mathbf{0}})$
6: $\tilde{\mathbf{x}} \sim \mathbf{p}(\mathbf{x}|\tilde{\mathbf{h}}^{\mathbf{0}})$
7: $\mathbf{h}^1 \leftarrow \sigma(\mathbf{c} + \mathbf{W}\tilde{\mathbf{x}})$
Update parameters:
8: $\mathbf{b} \leftarrow \mathbf{b} + \alpha(\mathbf{x}^{\mathbf{0}} - \tilde{\mathbf{x}})$
9: $\mathbf{c} \leftarrow \mathbf{c} + \alpha(\mathbf{h}^{\mathbf{0}} - \mathbf{h}^{\mathbf{1}})$
10: $\mathbf{W} \leftarrow \mathbf{W} + \alpha(\mathbf{h}^{\mathbf{0}}\mathbf{x}^{\mathbf{0}} - \mathbf{h}^{\mathbf{1}}\tilde{\mathbf{x}})$

Algorithm 3.1: CD-1

probabilities for the hidden nodes. Finally, on line 7, the probabilities of the hidden nodes are calculated conditioned on the sampled vector for the visible nodes. The parameters are updated on lines 8-10. For $k > 1$, the positive and negative phases are repeated k times before the parameters are updated.

CHAPTER 4

ATOMIC RESTRICTED BOLTZMANN MACHINES

4.1 Atomic Restricted Boltzmann Machines

We propose a training method for RBMs that partitions the network into several overlapping subnetworks. With our method, training involves several partition steps. In each step, the RBM is partitioned into multiple RBMs as shown in Figure 4.1. We call these small RBMs "Atomic RBM." In this figure, the partitions do not overlap; we discuss the version with overlap later in this section. These atomic RBMs are trained in parallel with a corresponding partition of training data using CD-1. In other words, the feature vector is also partitioned, and each individual RBM is trained on a section of that feature vector. Once all partitions are trained, we generate another set of RBMs with fewer splits. For example, in Figure 4.1, we initially generate four RBMs. In the second step, we generate two, and final training occurs on the full RBM. It should be noted that the training process in all steps is over the same weight matrix. In other words, during training, each RBM partition updates a subsection of the weight vector defined between all visible nodes and hidden nodes.

The motivation behind our approach is that when RBMs are small, they can be trained with more training epochs. However, as we reduce the number of splits, training requires fewer epochs because model parameters are already optimized within various partitions. Therefore, less time is required to train the final RBM with less splits.

The pseudo-code for our training procedure is given in Algorithm 4.2. The algorithm accepts a list of configurations as input. Each configuration defines training properties of a layer for training; it describes the number of partitions in each layer,

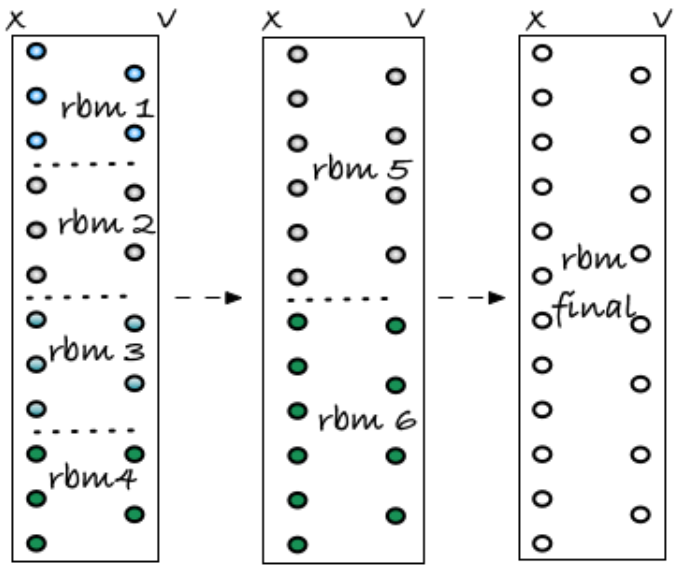

Figure 4.1: RBM Partitions

the learning rate, the number of training samples for each layer, and the number of iterations for each layer. On lines 2-3, it creates the weight matrix, the bias vector for the visible nodes, and the bias vector for the hidden nodes. Then, for each configuration it selects the training samples and calls *AtomicRBMUpdate* procedure (lines 5-8).

The *AtomicRBMUpdate* procedure creates a list of Atomic RBMs based on the configuration on line 1. For instance, if the configuration has 10 split points it will create a list of 10 RBMs. Each RBM will operate on a portion of $\mathbf{W}$, $\mathbf{b}$, and $\mathbf{c}$. Then for each training instance, it splits the training instance into the number of partitions (line 3). If $configuration.overlap$ is greater than 0, each split will share $configuration.overlap$ percent elements of the neighboring partition. Finally, it calls CD-1 algorithm to train an atomic RBM for the given data split and learning rate.

The *CreateAtomicRBM* procedure creates a list of atomic RBMs for a given configuration. On lines 2-3, it determines the number of visible and hidden nodes for

each partition. Then for each partition and RBM it creates, it calculates the base pointer for determining what part of **W**, **b**, and **c** this particular atomic RBM operates on (lines 6-7). Finally, on line 8, it creates an atomic RBM with this configuration.

Based on the intuition that neighboring RBMs may share some features (nodes), for overlapping partitions, we define similar partitions as described above. However, in this model, each partition has some percent of its nodes overlap with its neighboring partitions. As shown in the example in Figure 4.2, the RBMs are sharing two hidden and two visible nodes. Since nodes are shared, partitioned RBMs cannot be trained concurrently without some kind of message passing or synchronization. It should be noted that when trained sequentially, message passing between the partitions is not required.

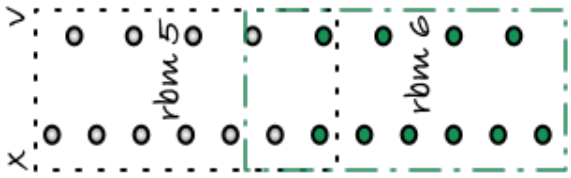

Figure 4.2: RBM With Overlapping Partitions

4.2 Discriminative Atomic RBMs

For discriminative Atomic RBMs, we propose a model where we train partitions as described in Section 4.1 in an unsupervised fashion with the exception of the last layer. As shown in Figure 4.3, class nodes are added-one node per class to the visible vector of the final layer. Class nodes are constructed by setting the node that represents the label to 1.0 and the rest to 0. Thus, we train the final layer in a supervised fashion.

To classify a new data sample, the input vector will be constructed with all class nodes set to 0. Then, the input vector will be propagated to the hidden layer using

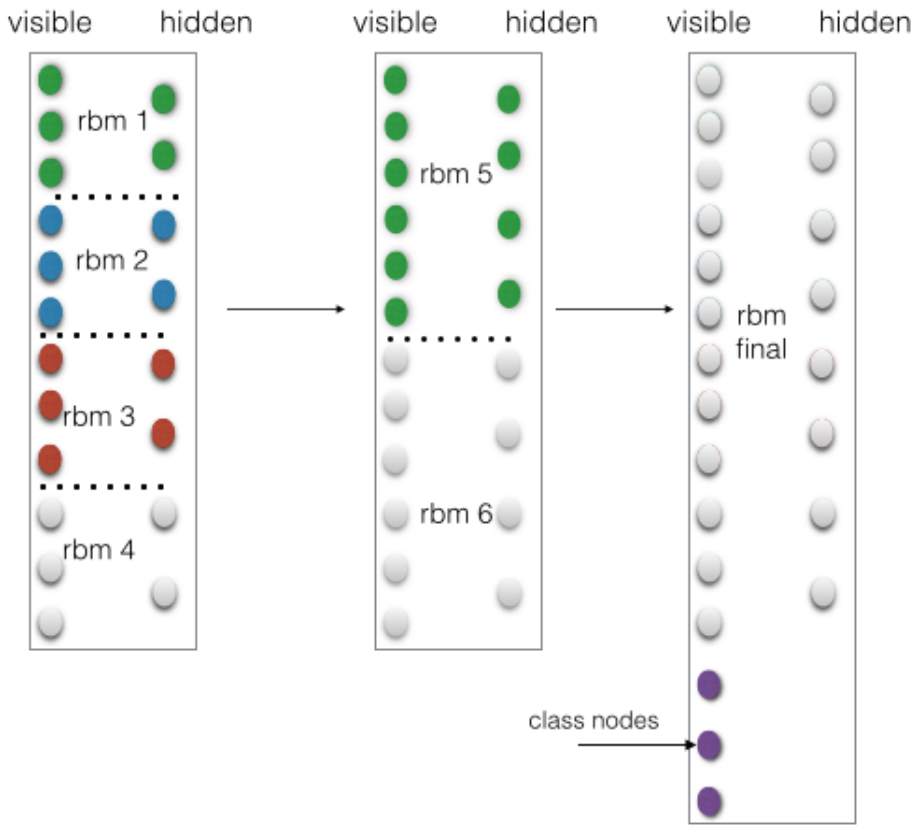

Figure 4.3: RBM Discriminative Atomic RBM

Equation 3.14. A new visible vector will be sampled from the model based on the activation of hidden nodes. Since the visible vector contains class nodes as well, the class nodes' values will be used to predict the class label. The class node with the highest activation value will determine the class label.

Algorithm AtomicRBM($\mathcal{L}$, **nvisible**, **nhidden**)

Input: $\mathcal{L}$ a list of configurations that describe splits and training instances for each network layer training. Each configuration contains the number of splits, the number of training samples, the learning rate, the overlapping rate. *nvisible*: the number of visible nodes. *nhidden*: the number of hidden nodes.
$W \leftarrow$ Create and initialize weight matrix for $nvisible \times hidden$ nodes
$b \leftarrow$ Create and initialize bias vector for visible layer
$c \leftarrow$ Create and initialize bias vector for hidden layer
for each l in $\mathcal{L}$:
    $\mathcal{X} \leftarrow$Training instances
    AtomicRBMUpdate(**l**, $\mathcal{X}$, W, b, c)
end for

Algorithm AtomicRBMUpdate(**l**, $\mathcal{X}$, **W**, **b**, **c**)

$rbmList \leftarrow$ **CreateAtomicRBM**($\mathbf{l}, W, b, c$)
for each x_i in $\mathcal{X}$:
    $splits \leftarrow$split x_i data instance into number of l.splits partitions
    for each rbm_i in $rbmList$:
        rbm_i.CD-1($splits(i), l.learningRate$)
    end for
end for

Algorithm CreateAtomicRBM(**l**, **W**, **b**, **c**)

Input: **l**: split configuration, W: weight matrix, b: visible layer bias vector, c: hidden layer bias vector
$nvsible \leftarrow l.visible/l.splits$
$nhidden \leftarrow l.hidden/l.splits$
$rbms$: Atomic RBM list
for i in $configuration.splits$:
    $vbase \leftarrow$base index in visible vector
    $hbase \leftarrow$base index in hidden vector
    $rbm_i \leftarrow$ new RBM of $\{W, b, c, vbase, hbase, nvisible, nhidden\}$
end for
return rbms

Algorithm 4.2: AtomicRBM Update Algorithm

CHAPTER 5

EXPERIMENTS

5.1 Experimental Setup

We measure the performance of our *AtomicRBM* algorithms using reconstruction error, which is defined to be the average pixel differences between the original and reconstructed images. To calculate reconstruction error, we first obtain the binary representation of the original image and the reconstructed image. Thirty (30) is chosen as the threshold for converting pixel values [0-255] to binary 0 or 1. Thus, pixel values greater than or equal to 30 are set to 1 while values less than 30 are set to 0. The image is reconstructed by sampling a hidden vector from the model for the given image, followed by sampling a visible vector based on the hidden activations. The resulting visible vector is multiplied by 255 and binarized as described above. Finally, the reconstruction error is calculated as follows.

$$error = \frac{\sum_i (image(i) \neq reconstructedImage(i))}{total\ pixels} \tag{5.1}$$

To have a fair comparison in terms of performance, rather using CPU time, we applied the following method. Since operations at each step of CD involve $visible\_nodes \times hidden\_nodes$ updates, we estimate that the total number of Markov chain calculations is

$$ChainOperations = visible\ nodes \times hidden\ nodes \times samples$$

Fewer chain operations translate into less CPU time.

For each epoch, we use a batch size of 10 images from the training samples. Thus, for 6,000 epochs, 60,000 samples are used for training. Unless stated otherwise, we use CD-1 for all training steps.

Moreover, we use each image sample once; unless stated otherwise, we ran our training algorithms on samples for one iteration only—at most, each sample is used only once. Each RBM-X represents a step with X partitions. Samples chosen for RBM-X are always from the first N samples of total images. RBM-1 represents the final model. For these experiments, partitions are trained sequentially. Thus, if we train them concurrently, the total *ChainOperations* will be lower because at each layer, atomic RBMs will run in parallel.

Finally, *Single RBM* represents a fully connected RBM that is used as a baseline for comparison. Since *Single RBM* perform better when the learning rate is 0.01, the learning rate 0.01 was selected for comparing algorithms.

5.2 Preliminary Results: MNIST dataset

The MNIST database (Mixed National Institute of Standards and Technology database) is a database of handwritten digits. The MNIST dataset is used for our experiments due to its wide use in evaluating RBMs. MNIST has 60,000 training samples and 10,000 test samples of images. Each image is 28×28 pixels corresponding to handwritten digits from 0 to 9.

The unpartitioned RBM has 500 hidden nodes and $28 \times 28 = 784$ visible nodes. Table 5.1 shows the results of our first experiment where the learning rate is set to 0.01 for all RBMs. As described later, we experimented with different learning rates. The training sample for each RBM is given in the *samples* column.

As compared to Single RBM, RBM-1 has significantly lower reconstruction error. The total *ChainOperations* for partitioned RBMs is also less than Single RBM. In the table, using a t-test, significant results with 99% confidence are shown in bold. *AtomicRBM*, after training on 20 partitions, significantly outperformed the *SingleRBM*. Furthermore, the total number of chain operations for *AtomicRBM* is substantially less than for *SingleRBM*.

Table 5.1: Reconstruction Errors

Configuration	Number of RBMs	Samples	Learning Rate	Reconstruction Error (%)	Chain Operations (10^9)
Single RBM	1	60000	0.01	3.85	23.52
RBM-28	28	60000	0.01	4.76	0.84
RBM-20	20	50000	0.01	4.03	0.98
RBM-15	15	40000	0.01	**3.19**	1.05
RBM-10	10	30000	0.01	**2.67**	1.18
RBM-5	5	25000	0.01	**2.29**	1.96
RBM-2	2	20000	0.01	**2.33**	3.92
RBM-1	1	20000	0.01	**2.36**	7.84
Total					**17.77**

Since we want fast convergence in the first step, in the following experiment we varied the learning rate to enable this. In layers with multiple partitions we kept the learning rate high, whereas when the number of split gets smaller we also lowered the learning rate. Results are shown in Table 5.2. The *AtomicRBM* with 99% confidence outperforms the Single RBM in all steps including the first partition, RBM-28 (with 28 partitions). Moreover, reconstruction errors are even lower compared to our previous experiment.

Using the same configuration above, we varied the learning rate (denoted *lr* in the results) for the Single RBM and RBM-1. Learning rates for other RBM-X are fixed as in the configuration given in Table 5.2. Reconstruction errors for different learning rates are given in Table 5.3. Results demonstrate that *AtomicRBM* is less sensitive

Table 5.2: Varied Learning Rate

Configuration	Number of RBMs	Samples	Learning Rate	Reconstruction Error (%)	Chain Operations (10^9)
RBM-28	28	60000	0.03	**3.23**	0.84
RBM-20	20	50000	0.03	**2.93**	0.98
RBM-15	15	40000	0.03	**2.66**	1.05
RBM-10	10	30000	0.025	**2.30**	1.18
RBM-5	5	25000	0.020	**2.08**	1.96
RBM-2	2	20000	0.010	**2.10**	3.92
RBM-1	1	20000	0.010	**2.10**	7.84
Total					**17.77**

to different learning rates as compared to the Single RBM with 99% confidence. This suggests that as long as the learning rate is kept higher when there are many partitions and lower when there are fewer partitions, *AtomicRBM* will learn data accurately– we do not have to find an optimum learning rate for the algorithm.

Table 5.3: Sensitivity to Learning Rate

	$lr = 0.03$	$lr = 0.01$	$lr = 0.005$	$lr = 0.0005$	$lr = 0.00005$
Single RBM	4.43	3.85	4.22	9.40	23.15
RBM-1	**3.45**	**2.10**	**1.95**	**1.83**	**1.92**

We also wanted to determine if overlapping partitions would affect the results. We ran our experiment with 5% overlap, which means that each RBM shares 5% of its neighbor's nodes (5% from the left neighbor and 5% from the right neighbor). We ran overlapping partitions sequentially. As shown in Table 5.4, reconstruction errors are even lower with only a modest increase in overhead in terms of *ChainOperations*.

Comparing overlapping with non-overlapping *AtomicRBM* algorithms using the *t-test*, results show that the overlapping algorithm outperforms the non-overlapping algorithm with 99% confidence in almost every stage. However, in the last stage, the results were not significantly different, as shown in Table 5.5. We hypothesize that since overlapping partitions have more connections in each partition, they will require more training samples. Moreover, for both configurations, there is an increase

Table 5.4: Overlapping Partitions

Configuration	Number of RBMs	Samples	Learning Rate	Reconstruction Error (%)	Chain Operations (10^9)
RBM-28	28	6000	0.03	**3.11**	0.90
RBM-20	20	5000	0.03	**2.76**	1.10
RBM-15	15	4000	0.03	**2.50**	1.18
RBM-10	10	3000	0.025	**2.19**	1.35
RBM-5	5	2500	0.020	**1.95**	2.27
RBM-2	2	2000	0.010	**1.92**	4.30
RBM-1	1	2000	0.010	**2.08**	7.84
Total					**18.94**

in reconstruction error in the last layer. Again, we speculate the last layer needs more training samples. Nonetheless, this also suggests that we can have a cut-off with two partitions.

Table 5.5: Non-overlapping vs. Overlapping Partitions

Configuration	Number of RBMs	Samples	Learning Rate	Overlapping Reconstruction Error(%)	NonOverlapping Reconstruction Error(%)	Overlapping Chain Operations (10^9)	NonOverlapping Chain Operations (10^9)
RBM-28	28	60000	0.03	**3.11**	3.23	0.90	0.84
RBM-20	20	50000	0.03	**2.76**	2.93	1.10	0.98
RBM-15	15	40000	0.03	**2.50**	2.66	1.18	1.05
RBM-10	10	30000	0.025	**2.19**	2.30	1.35	1.18
RBM-5	5	25000	0.020	**1.95**	2.08	2.27	1.96
RBM-2	2	20000	0.010	**1.92**	2.10	4.30	3.92
RBM-1	1	20000	0.010	2.08	2.10	7.84	7.84
Total						**18.94**	**17.77**

Finally, 10-fold cross validation results are given in Table 5.6. Rather than using the provided training and test data sets, we pooled all of the data and split samples into 10 equal size subsamples. One subsample was used as the validation data for testing and the remaining 9 subsamples were used for training. We repeated this process 10 times, rotating through the subsamples. It should be noted that the numbers of samples for partitioned RBMs are not equal (Table 5.6) because we wanted to keep the total time complexity of *AtomicRBM* to be no worse than the Single RBM. *AtomicRBM* outperforms Single RBM with 99% confidence. Moreover, overlapping RBMs have lower average reconstruction error as compared to non-overlapping ones.

Table 5.6: 10-Fold Cross Validation Results

Configuration	Number of RBMs	Samples	Learning Rate	**No Overlap**: Average Reconstruction Error (%)	**Overlap**: Average Reconstruction Error(%)	Chain Operations per fold (10^9) No-Overlap/Overlap
Single RBM	1	60000	0.01	4.06		21.12
RBM-28	28	60000	0.03	**3.32**	**3.32**	0.76/0.81
RBM-20	20	50000	0.03	**3.07**	**2.92**	0.88/1.00
RBM-15	15	40000	0.03	**2.68**	**2.62**	0.94/1.06
RBM-10	10	30000	0.025	**2.35**	**2.29**	1.06/1.21
RBM-5	5	25000	0.020	**2.12**	**2.09**	1.76/2.04
RBM-2	2	20000	0.010	**2.15**	**2.08**	3.53/3.88
RBM-1	1	20000	0.010	**2.18**	**2.14**	7.06/7.06
Total						**16.00/17.06**

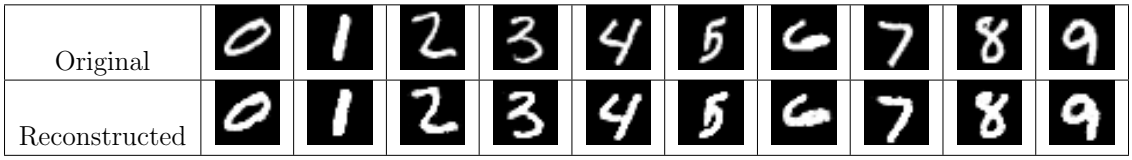

Figure 5.1: Original vs. Reconstructed Images

To visually compare the original images with some of our reconstructed images, we present some examples in Figure 5.1.

The learning behavior with respect to the number of training samples is given in Figure 5.2. We compare RBM-10 with Single RBM. After each training cycle where we add 10,000 more images, we tested the algorithms on 10,000 images. RBM-10 outperforms Single RBM with 99% confidence on all training steps.

As we described at the begining of the Section 5.2, so far we ran these experiments for one iteration only. To see how our learning method will behave with additional iterations, we ran *AtomicRBM* and Single RBM for 15 iterations. Results are shown in Table 5.7. Starting with RBM-10, *AtomicRBM* significantly outperforms Single RBM with 99% confidence. For *AtomicRBM*, on average, the error is approximately 5 pixels out of 28×28 pixels, whereas it is 10 pixels for Single RBM. Also, these results demonstrate that when the number of iterations are increased, the reconstruction error does significantly decrease.

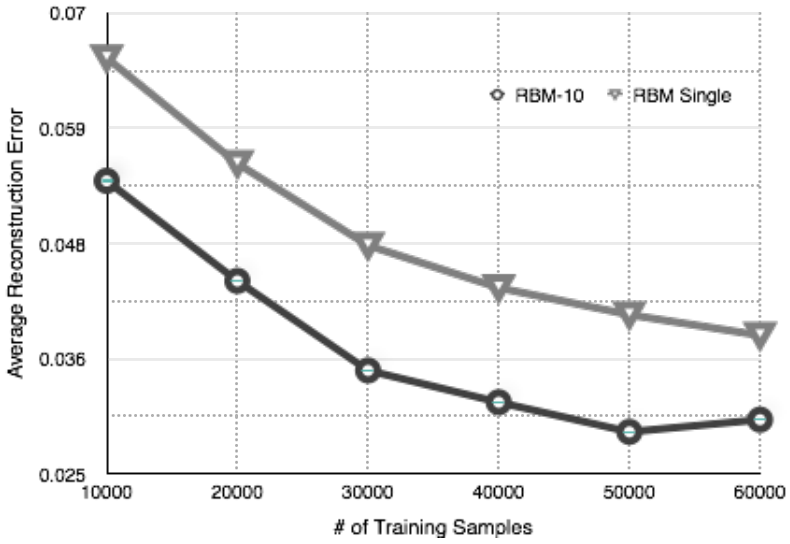

Figure 5.2: Reconstruction Error vs. Training Samples

5.3 Preliminary Results: NIST dataset

The *Special Database 19* dataset from the National Institute of Standards and Technology (NIST) is the official training dataset for handprinted document and character recognition from 3600 writers, including 810K character images and 402K handwritten digits. Unlike the MNIST dataset, images are 128 by 128 pixels. We wanted to have the same number of images as the MNIST dataset. Thus, we selected 62K images for training and testing. The dataset consists of 62 types of images for lowercase and uppercase letters, and numbers. Thus, in our dataset each type has 1,000 images. We used 10% for testing and 90% for training. Hold out results are shown in Table 5.8. Based on the t-test, *AtomicRBM* significantly outperforms Single RBM, again with substantially fewer chain operations.

Finally, *AtomicRBM* reconstruction error for each character is given in Table 5.9. Characters with highest reconstruction error are shown in bold. Thus, the average reconstruction error is lowest for I, i, and 1 and it is highest for W,Q and B.

Table 5.7: Training Iterations

Configuration	Number of RBMs	Samples	Learning Rate	Reconstruction Error (%)
Single RBM	1	60000	0.005	1.29
RBM-28	28	60000	0.03	1.75
RBM-20	20	30000	0.03	1.45
RBM-15	15	20000	0.03	1.35
RBM-10	10	20000	0.025	**1.08**
RBM-5	5	20000	0.02	**0.94**
RBM-2	2	20000	0.01	**0.75**
RBM-1	1	30000	0.005	**0.67**

Table 5.8: Training Characteristics with NIST dataset

Configuration	Number of RBMs	Training Samples	Learning Rate	Reconstruction Error (%)	Chain Operations (10^9)
Single RBM	1	62000	0.01	4.82	507.90
RBM-28	28	62000	0.03	**3.74**	19.92
RBM-20	20	50000	0.03	**3.65**	24.09
RBM-15	15	40000	0.03	**3.70**	25.19
RBM-10	10	30000	0.025	**3.69**	28.70
RBM-5	5	25000	0.020	**3.63**	47.78
RBM-2	2	20000	0.010	**3.67**	90.14
RBM-1	1	20000	0.010	**3.74**	163.8
Total					**399.62**

Table 5.9: Reconstruction Error per Character

Uppercase	Error	Lowercase	Error	Number	Error
A	3.93	a	3.55	0	2.95
B	**6.24**	b	4.14	1	*1.95*
C	2.8	c	2.68	2	3.55
D	5.22	d	4.49	3	4.01
E	3.34	e	2.79	4	3.67
F	3.61	f	4.14	5	4.07
G	5.68	g	5.15	6	3.35
H	4.25	h	3.35	7	3.2
I	*1.47*	i	*1.93*	8	4.49
J	4.47	j	3.2	9	3.97
K	4.89	k	4.15		
L	3.08	l	1.97		
M	4.26	m	4.69		
N	3.64	n	2.83		
O	2.58	o	2.69		
P	4.62	p	3.78		
Q	**6.62**	q	4.5		
R	3.53	r	2.37		
S	3.24	s	2.98		
T	3.08	t	3.48		
U	3.56	u	3.09		
V	3.18	v	2.87		
W	**6.92**	w	4.17		
X	4.38	x	3.19		
Y	3.75	y	3.37		
Z	5.04	z	3.57		

CHAPTER 6

DISSERTATION PLAN

The following section outlines high-level goals and a work schedule for progress on the dissertation.

1. Revise prospectus as starting point for dissertation, incorporating feedback from the comprehensive examination.

2. Complete literature review for all research questions.

3. Determine a multimodal dataset with text and images and apply our algorithm to find out classification accuracy. Adapt to handle multimodal data as necessary. Determine an appropriate venue and submit paper for publication.

4. Develop an alternative optimization methods for AtomicRBMs. Determine an appropriate venue and submit paper for publication.

5. Perform theoretical Analysis of Atomic RBM: Formalize and characterize why Atomic RBM works. Determine how accurately the algorithm captures hierarchical features. Identify learning characteristics such as sparseness, free energy, and overfitting per layer. Determine an appropriate venue and submit paper for publication.

6. Develop a temporal *AtomicRBM* and apply it to temporal (e.g, time series) and sequential (e.g.,clickstream). Determine a temporal dataset. Evaluate our algorithm on a variety of temporal/sequential data sets. Determine an appropriate venue and submit paper for publication.

7. Finish writing the dissertation.

8. Present and defend the dissertation in Fall, 2015

CHAPTER 7

CONCLUSIONS

We showed that our *AtomicRBM* training algorithm with small RBM partitions outperforms training full RBMs using CD-1. In addition to having superior results in terms of reconstruction error, *AtomicRBM* is also faster as compared to the single, full RBM. The reason that *AtomicRBM* is faster is due to having fewer connections in each training step. However, the reasons for the superior generative characteristics in terms of reconstruction error is not that obvious. We hypothesize that it is because in each training step, fewer nodes are involved and a small partition RBM settles in a low energy configuration more rapidly. As we move to other stages with less partitions, fewer training instances are needed to modify the energy configuration to obtain lower energy in the full network.

However, reconstruction error is not always a good indicator for a classification task. We intend to investigate the classification accuracy of this model. Moreover, we hypothesize that this model will represent multi-modal data and deep features more accurately. To find out the representation power of this model, we aim to study the theoretical foundation of *AtomicRBM*.

Our approach of atomic RBMs will open the door to many potential applications. Fast and accurate models are needed for event detection, anomaly detection, and temporal pattern mining applications. Thus, we intend to develop a temporal *AtomicRBM* that represents temporal or sequential features.

REFERENCES CITED

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

[2] Ooh! ahh! google images presents a nicer way to surf the visual web [online]. July 2010 [cited 8/9/2014].

[3] Bruno A Olshausen, i in. Emergence of simple-cell receptive field properties by learning a sparse code for natural images. *Nature*, 381(6583):607–609, 1996.

[4] Yoshua Bengio, Aaron Courville, Pascal Vincent. Representation learning: A review and new perspectives. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(8):1798–1828, 2013.

[5] Hasari Tosun, John W. Sheppard. Training restricted boltzmann machines with overlapping partitions. *Machine Learning and Knowledge Discovery in Databases,ECML PKDD*, Volume 8726 series *Lecture Notes in Computer Science*, pages 195–208. Springer, 2014.

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

[7] Christopher Poultney, Sumit Chopra, Yann L Cun, i in. Efficient learning of sparse representations with an energy-based model. *Advances in neural information processing systems*, pages 1137–1144, 2006.

[8] Pascal Vincent, Hugo Larochelle, Yoshua Bengio, Pierre-Antoine Manzagol. Extracting and composing robust features with denoising autoencoders. *Proceedings of the 25th international conference on Machine learning*, pages 1096–1103. ACM, 2008.

[9] David H Ackley, Geoffrey E Hinton, Terrence J Sejnowski. A learning algorithm for boltzmann machines. *Cognitive science*, 9(1):147–169, 1985.

[10] P. Smolensky. Parallel distributed processing: Explorations in the microstructure of cognition, vol. 1. David E. Rumelhart, James L. McClelland, CORPORATE PDP Research Group, editors, *Information Processing in Dynamical Systems: Foundations of Harmony Theory*, pages 194–281. MIT Press, Cambridge, MA, USA, 1986.

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

[12] Yoshua Bengio, Pascal Lamblin, Dan Popovici, Hugo Larochelle, i in. Greedy layer-wise training of deep networks. *Advances in Neural Information Processing Systems*, 19:153, 2007.

[13] Geoffrey Hinton, Ruslan Salakhutdinov. Discovering binary codes for documents by learning deep generative models. *Topics in Cognitive Science*, 3(1):74–91, 2011.

[14] George E Dahl, Ryan P Adams, Hugo Larochelle. Training restricted boltzmann machines on word observations. *Proceedings of the 29th International Conference on Machine Learning*, pages 679–686. ACM, 2012.

[15] Hugo Larochelle, Yoshua Bengio. Classification using discriminative restricted boltzmann machines. *Proceedings of the 25th International Conference on Machine Learning*, pages 536–543. ACM, 2008.

[16] Jérôme Louradour, Hugo Larochelle. Classification of sets using restricted boltzmann machines. *Uncertainty in Artificial Intelligence*, pages 463–470. AUAI, 2011.

[17] Ruslan Salakhutdinov, Andriy Mnih, Geoffrey Hinton. Restricted boltzmann machines for collaborative filtering. *Proceedings of the 24th International Conference on Machine Learning*, pages 791–798. ACM, 2007.

[18] Daphne Koller, Nir Friedman. *Probabilistic Graphical Models: Principles and Techniques*. MIT press, 2009.

[19] Philemon Brakel, Sander Dieleman, Benjamin Schrauwen. Training restricted boltzmann machines with multi-tempering: Harnessing parallelization. *Artificial Neural Networks and Machine Learning–ICANN 2012*, pages 92–99. Springer, 2012.

[20] Dong Yu, Michael L. Seltzer, Jinyu Li, Jui-Ting Huang, Frank Seide. Feature learning in deep neural networks - A study on speech recognition tasks. *CoRR*, abs/1301.3605, 2013.

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

[22] Yoshua Bengio. Learning deep architectures for ai. *Foundations and Trends in Machine Learning*, 2(1):1–127, 2009.

[23] Ruslan Salakhutdinov, Geoffrey E Hinton. Deep boltzmann machines. *International Conference on Artificial Intelligence and Statistics*, pages 448–455, 2009.

[24] Hugo Larochelle, Yoshua Bengio, Jérôme Louradour, Pascal Lamblin. Exploring strategies for training deep neural networks. *The Journal of Machine Learning Research*, 10:1–40, 2009.

[25] Jeffrey Dean, Greg Corrado, i in. Large scale distributed deep networks. *Advances in Neural Information Processing Systems*, pages 1232–1240, 2012.

[26] Guido Montufar, Jason Morton. Inductive principles for restricted boltzmann machine learning. *International Conference on Learning Representations*, 2013.

[27] Benjamin M. Marlin, Nando de Freitas. Asymptotic efficiency of deterministic estimators for discrete energy-based models: Ratio matching and pseudolikelihood. *CoRR*, abs/1202.3746, 2012.

[28] Benjamin M Marlin, Kevin Swersky, Bo Chen, Nando D Freitas. Inductive principles for restricted boltzmann machine learning. *International Conference on Artificial Intelligence and Statistics*, pages 509–516, 2010.

[29] Don S Lemons. *A student's guide to entropy*. Cambridge University Press, 2013.