INITIALIZING NEURAL NETWORKS USING RESTRICTED BOLTZMANN MACHINES

by

Amanda Anna Erhard

B.S. in Electrical Engineering, University of Pittsburgh, 2014

Submitted to the Graduate Faculty of
the Swanson School of Engineering in partial fulfillment
of the requirements for the degree of
Master of Science

University of Pittsburgh

2017

UNIVERSITY OF PITTSBURGH

SWANSON SCHOOL OF ENGINEERING

This thesis was presented

by

Amanda Anna Erhard

It was defended on

April 3, 2017

and approved by

Amro El-Jaroudi, Ph.D., Associate Professor
Department of Electrical and Computer Engineering

Steven Jacobs, Ph.D., Assistant Professor
Department of Electrical and Computer Engineering

Zhi-Hong Mao, Ph.D., Associate Professor
Department of Electrical and Computer Engineering

Thesis Advisor: Amro El-Jaroudi, Ph.D., Associate Professor
Department of Electrical and Computer Engineering

INITIALIZING NEURAL NETWORKS USING RESTRICTED BOLTZMANN MACHINES

Amanda Anna Erhard, M.S.

University of Pittsburgh, 2017

This thesis presents an approach to initialize the parameters of a discriminative feed- forward neural network (FFN) model using the trained parameters of a generative classification Restricted Boltzmann Machine (cRBM) model. The ultimate goal of FFN training is to obtain a network capable of making correct inferences on data not used in training. Selection of the FFN initialization is a critical step that results in trained networks with different parameters and abilities. Random selection is one simple method of parameter initialization. Unlike pretraining methods, this approach does not extract information from the training data, and the optimization does not guarantee that relevant parameters will result.

Pretraining methods train generative models such as RBMs that define model parameters by learning about the training data structure using information based on clusters of points discovered in the data. This proposed method uses a cRBM that incorporates the class information in pretraining, determining a complete set of non-random FFN parameter initializations. Eliminating random initializations is one advantage in this approach over previous pretraining methods. This approach also uniquely alters the hidden layer bias parameters, compensating for differences in cRBM and FFN structure when adapting the cRBM parameters to the FFN. This alteration will be shown to provide meaningful parameters to the network by evaluating the network before training. Depending on the number of pretraining epochs and the influence of generative and discriminative methods in hybrid pretraining, the hidden layer bias adjustment allows initialized and untrained models to achieve a lower error range than corresponding models without the bias adjustment.

Training FFNs with all parameters pretrained is capable of reducing the standard deviation of network errors from that of randomly initialized networks. One disadvantage of this proposed pretraining approach, as with many pretraining methods, is the necessity of two training phases compared to the single phase of backpropagation used for randomly initialized networks.

Keywords: machine learning, RBM, hybrid model, generative, discriminative, hybrid training, initializing feedforward networks, classification RBM.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

PREFACE

This thesis is dedicated to everyone who has believed in me, with special thanks to my family and to my advisor for their unfailing patience, wisdom, encouragement, understanding, and inspiration. In memory of my great-grandparents, Anna Margaret and Howard Louis Wasmer.

1.0 INTRODUCTION

Mathematical equations describe the structure of the physical world along with an array of tasks ranging from common chores like balancing a checkbook to quantum physics. Automating tasks will help with optimizing productivity, reducing risk, or providing information about an operating environment. Current examples illustrating progress toward achieving these goals include search engine websites, speech recognition by mobile devices, and facial recognition software used by authorities.

To automate a task, the steps for task completion should be translated into equations for computers to calculate. Some methods applied by humans to perform tasks are described naturally using words. For example, in distinguishing between a car and a motorcycle, a human may count the number of wheels. The number of wheels is one measured characteristic, or feature, that could be provided to a machine for identifying vehicle types. If this quantity is already known, conditional statements can be used for automatic identification. Developing equations quantifying the methods used by humans to identify the wheels of a vehicle may take much time. Those equations may hold under a subset of task conditions since full understanding about all use cases may be unknown. Multiple sets of equations would then be required.

As an example of dealing with multiple scenarios, different viewpoints of a vehicle affect the number of visible wheels. Viewing a car from a distance may reveal only two wheels. A human immediately knows to identify other features such as the number of doors and the relative size of the vehicle before reaching a conclusion. A machine would have to be instructed to search for additional features since, in this case, the number of wheels is not a distinguishing feature between cars and motorcycles. One example additional feature could be the number of doors or windows.

Pictures and audio recordings, which simulate humans' abilities to see and hear, are able to be inputs to a computer. The ability to make decisions using collected environmental information is also required for task automation. Image processing and speech recognition techniques can be applied to detect objects and words. Applying these techniques to tasks such as disease identification and speaker identification involves decision making by determining relevant features to accomplish the task. An ideal process involved in task automation should be capable of describing varied and intricate functions to a machine regardless of the plethora of possible situations. Training graphical models such as neural networks is one method able to distinguish between data sets collected from varied tasks.

The remainder of this chapter will provide a general introduction to feedforward neural network models (FFNs) in Section 1.1. Challenges presented in using these networks are illustrated in Section 1.2. This chapter will end by describing the structure for the remainder of the thesis.

1.1 FEEDFORWARD NEURAL NETWORKS (FFNS)

A neural network is a generalized mathematical concept that operates by weighting and combining features that are used to differentiate between types or classes of objects or to calculate or predict a desired value. There are different graphical configurations of neural networks. All are comprised of nodes and links where the base unit, a node with connections, is termed a neuron. A simpler model examined in this thesis is a feedforward neural network (FFN). The term feedforward describes how information flows through the network from the input nodes through the hidden layer nodes to the output nodes.

A sample FFN is provided in the Figure 1.1. Yellow circles represent the FFN input nodes, indicated mathematically using the vector $\mathbf{v}$. Throughout this document, bold lowercase letters will denote vectors, and bold capital letters will indicate matrices. Values in the vector $\mathbf{v}$ are visible or known to the user. Blue circles illustrate the FFN hidden nodes, represented in equations by the vector $\mathbf{h}$. These hidden node values are not directly provided to the user, but can be calculated. Red circles show the FFN output nodes, given as the vector values $\mathbf{e}_y$. The output nodes provide the value that the network calculates to

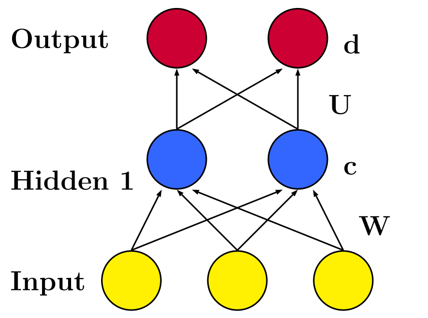


Figure 1.1: Feedforward neural network (FFN)

be the most appropriate given the inputs. The single direction arrows in Figure 1.1 show that this is an example of a feedforward neural network. This particular FFN has three input nodes, one hidden layer with two nodes, and two output nodes.

Each network has a single input layer and a single output layer, and the number of nodes for these layers is determined by the data. An input layer uses one node per feature measured from the data. The number of nodes for an output layer is determined by the problem application. Problem applications include classification and regression. In either application, the network provides a likely output value for each input. With classification, the network identifies one of a fixed number of groups to which the input may belong. With regression, the network supplies an answer that could take on any value such as when fitting a curve to a data set. The FFN models in this thesis are used for classification tasks. The number of hidden layers and the number of nodes for each hidden layer for neural networks is a design choice. In this thesis, one hidden layer is used for the FFNs, and the networks can also be termed as two-layer FFNs.

The generalized neural network model can be specified to operate on a particular dataset by changing the values of the parameters, known as weights and biases. The weights, $\mathbf{W}$, provide a matrix of values specifying the magnitude of the contribution of values from one layer to the next. The biases, $\mathbf{c}$, help determine typical node values even if all other nodes contribute no values. The process of changing network parameters to function for a particular application is known as training. More details about training are given in Chapter 2. After training, the neural network is expected to perform classification or regression accurately for previously unseen data inputs.

1.2 CHALLENGES WITH FFNS

The main goal that FFNs should fulfill is to provide reliable results of the most likely answer for data previously unseen during the training process. Directing design of the model structure and training process to achieve reliable results is difficult, mainly because of the vast number of unknown situations in which the network is expected to operate. Descriptions of specific situations may be possible to some extent for narrowly defined constraints for network operations. As the restrictions are loosened, this ability to provide situational knowledge decreases.

An unreliable metric to evaluate FFN models ' abilities to extrapolate successfully to new data is the quality of results that the model produces for the training dataset. Figures 1.2 through 1.6 show why this metric could be unreliable with a specific example for classification. Training data for the two classes are shown using 30 crosses and 22 circles.

A clear boundary between the two classes can be determined given these data, and one possible boundary, shown with a solid line, is shown in Figure 1.3. It is exaggerated to show why fitting a model exactly to training data could be problematic when the model is expected to operate also on new data.

In Figure 1.3, everything in the region above and to the left of the solid line is decided by the model to belong to Class 1. All data points below and to the right of the solid line are chosen by the model to belong to Class 2. In Figure 1.4, new data points, marked in magenta, have been added. There are a total of 37 crosses and 29 circles.

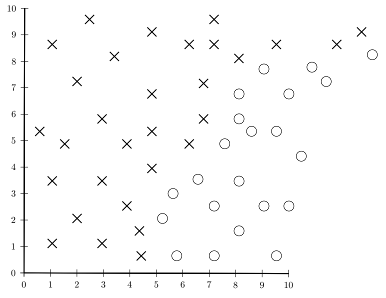


Figure 1.2: Example training data

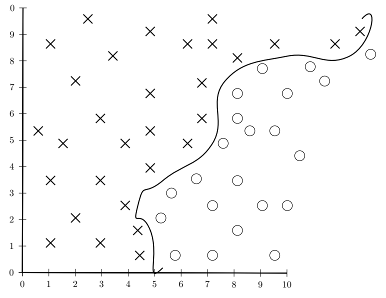


Figure 1.3: Exact training data classification

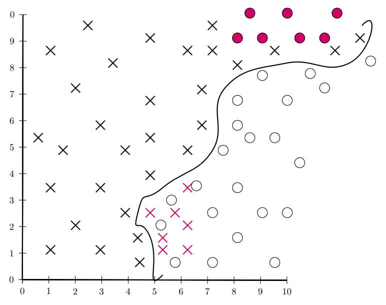


Figure 1.4: Adding test data: exact classification

Figure 1.4 shows a representation of a model's output. The example model shows that all seven of the newly added Class 1 points are marked as Class 2. All additional Class 2 points are misclassified as Class 1. The model has chosen incorrect classifications for all of the new points even though the boundary perfectly separates the original points. This is known as model overfitting. A different and simpler possible boundary is provided in Figure 1.5. Magenta points in these next few figures indicate misclassified data points.

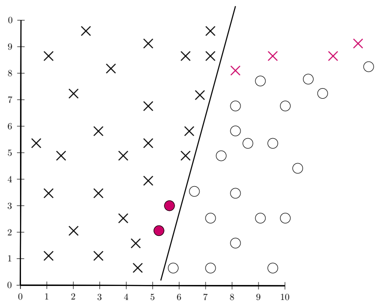


Figure 1.5: Approximate training data classification

With this alternate boundary, again all points to the left are determined by the model to belong to Class 1 and all points to the right are set as Class 2. This alternate boundary is not as well fit to the training data since six total points are misclassified. In Figure 1.6, the new data is added, and again magenta points are misclassified. Even though the alternate boundary is not as well formed to the training data, a total of eight points are misclassified instead of 14 as occurred with the original boundary.

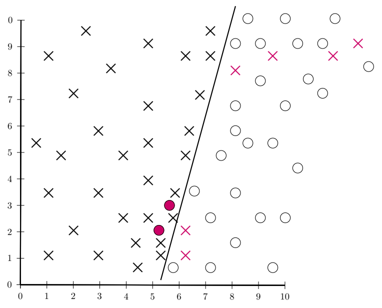


Figure 1.6: Adding test data: approximate classification

Different techniques have been explored to improve the generalization capability of FFN models. One method, early stopping, involves using part of the available data set for training and part to evaluate the model during training in order to end training when overfitting is detected [Bishop, 2006]. Methods studied in [Hinton, 2007], [Erhan et al., 2010], and [Larochelle et al., 2009] include initializing FFN models with trained parameters from different models. The training process for these different models is not the same as for FFN models. It has been shown that the initial parameter set assumed at the start of training FFNs impacts the ability of FFNs to generalize to previously unseen data [Erhan et al., 2010], [Larochelle et al., 2009], [Hinton, 2007], [Mohamed et al., 2009].

In this thesis, a Restricted Boltzmann Machine (RBM) model will be used to provide initial parameter values to the FFN. RBMs are described in detail in Chapter 2. Preliminary terms relevant for describing RBMs are explained initially. Derivations for creating the RBM model along with training are provided. Adaptations to both the RBM model and training methods so that these models can initialize FFNs are described in Chapter 3. Descriptions of methods using the RBMs to initialize FFNs are given in Chapter 4. Experiments and results are detailed in Chapter 5. Conclusions and extensions of these proposed FFN initialization methods are provided in Chapter 6. Additional derivations are provided in the Appendix.

2.0 RESTRICTED BOLTZMANN MACHINES

A Restricted Boltzmann machine (RBM) models or captures probability density functions (PDFs) of the observed data. These PDF functions characterize the variability of a dataset. Models such as these are known as generative models. When RBMs were first developed, they were trained using unlabeled data with unsupervised methods, explained below in Section 2.1. At that time, RBMs did not function equivalently to deterministic models like neural networks. Deterministic models use labeled data in a different method of training known as supervised training.

The purpose of this chapter is to provide a basic understanding of an RBM by explaining the terms and concepts introduced in the above paragraph. Section 2.1 provides fundamental background information necessary to the understanding of the RBM definition. Both the RBM definition as well as the derivation of the procedure for creation, are found in Section 2.2. A description of the different RBM functions is in Section 2.3. Definitions and derivations describing a common method for training RBMs are explained in Section 2.4.

Derivations in this chapter follow the notation presented. Vectors are represented by bold variables $(\mathbf{v}, \mathbf{h})$, and scalars are not bold (v, h). A sum over a bold variable $\sum_{\mathbf{v}} p(\mathbf{v})$ indicates that the sum can be expanded $\sum_{v_1} \sum_{v_2} \cdots \sum_{v_N} p(v_1, v_2, \cdots, v_N)$. In this example, $\mathbf{v}$ represents a vector of N variables, where each $v_i, i \in \{1, \cdots, N\}$ can take one of K values.

2.1 TYPES OF DATA, MODELS, AND TRAINING METHODS

The fundamental concepts explained in this section include labeled and unlabeled data, discriminative and generative models, and supervised and unsupervised training. Definitions for terms associated with PDFs are also provided as a precursor to discriminative and generative models. Usually, labeled data is used to train discriminative models with supervised training. Unlabeled data is commonly used to train generative models using unsupervised training.

2.1.1 Labeled and unlabeled data

The sources providing the information for dataset entries are known for a labeled dataset and unknown for an unlabeled dataset. In a dataset, there are P entries or observations. Each observation contains N different characteristics or measurements from a subset of sources. For example, consider a dataset shown in Table 2.1 containing three rows of information ($P = 3$), each with two columns of measurements ($N = 2$). The sources are types of motor vehicles. In this case, the measurements are the number of wheels and the maximum adult capacity. Examples of labels are: bus, car, and motorcycle. The assumed number of wheels and capacity for buses are six wheels and forty-nine adults, respectively. Cars have four wheels and are assumed to carry five adults. Motorcycles have two wheels, and the assumed capacity is two adults. The total amount of information known is the number of wheels, the maximum adult capacity, and the type of vehicle.

Table 2.1: Labeled dataset

No. of Wheels	Max. Capacity	Vehicle Type
4	5	car
6	49	bus
2	2	motorcycle

In contrast, if the same sample dataset is unlabeled, the number of wheels, and the maximum adult capacity would be provided for each of the P entries. To create an unlabeled dataset, either the source information can be removed from a labeled dataset, or the source information was never made available. Table 2.2 shows the creation of an unlabeled dataset from a labeled dataset.

Table 2.2: Unlabeled dataset

No. of Wheels	Max. Capacity
4	5
6	49
2	2

2.1.2 PDF terminology

Equation 2.1 shows possible information available from a general joint probability distribution function, $P(\boldsymbol{\mathcal{A}}, \boldsymbol{\mathcal{B}})$.

$$P(\boldsymbol{\mathcal{A}}, \boldsymbol{\mathcal{B}}) = P(\boldsymbol{\mathcal{A}}|\boldsymbol{\mathcal{B}})P(\boldsymbol{\mathcal{B}}) = P(\boldsymbol{\mathcal{B}}, \boldsymbol{\mathcal{A}}) = P(\boldsymbol{\mathcal{B}}|\boldsymbol{\mathcal{A}})P(\boldsymbol{\mathcal{A}}) \tag{2.1}$$

In this equation, the $[N \times 1]$ random vector, $\boldsymbol{\mathcal{A}}$, can take any of the possible values of measurements from a source. From the measurements provided in Section 2.1.1 the complete allowable set of values for either the number of wheels or for the maximum capacity belongs to the set of whole numbers $\boldsymbol{W}$. The prior information for $\boldsymbol{\mathcal{A}}$, $P(\boldsymbol{\mathcal{A}})$, indicates the possibility of obtaining each set of values of $\boldsymbol{\mathcal{A}}$.

The $[M \times 1]$ random vector, $\boldsymbol{\mathcal{B}}$, indicates all possible combinations used to group similar sets of measurements. The number of groups is denoted by M. With the example labeled dataset from Section 2.1.1, the groupings are known as classes, and the class values equal the labels. In Table 2.1 there are three classes ($M = 3$), each with a single observation. For the unlabeled dataset, the data could be separated into groups or clusters based on the maximum capacity. Observations 1 and 3 could be assigned to Cluster 1 since these observations have a lower maximum capacity in common. Cluster 2 with the higher capacity contains only Observation 2. In this situation, $M = 2$. Another possible separation or clustering could be by the number of wheels. In this case, Observations 1 and 2 would be in Cluster 1, since the number of wheels is greater than two. Observation 3 would be in Cluster 2. Again, $M = 2$. As with $P(\boldsymbol{\mathcal{A}})$, $P(\boldsymbol{\mathcal{B}})$ gives the prior information for $\boldsymbol{\mathcal{B}}$ which is the possibility of having information belonging to the group denoted by $\boldsymbol{\mathcal{B}}$.

The joint PDF, $P(\boldsymbol{\mathcal{A}}, \boldsymbol{\mathcal{B}}) = P(\boldsymbol{\mathcal{B}}, \boldsymbol{\mathcal{A}})$, provides all necessary information to describe completely the relationship of the dataset comprised of the observations $\boldsymbol{\mathcal{A}}$ and the groupings $\boldsymbol{\mathcal{B}}$. This illustrates how the set of measurements of the random vector $\boldsymbol{\mathcal{A}}$ relate to the possible groups given in the random vector $\boldsymbol{\mathcal{B}}$.

The term $P(\boldsymbol{\mathcal{A}}|\boldsymbol{\mathcal{B}})$ represents the likelihood, which determines the probability that an observation $\boldsymbol{\mathcal{A}}$ belongs to a given group $\boldsymbol{\mathcal{B}}$. Using the example dataset from Section 2.1.1, the likelihood can be understood as the chance of producing data that describes a motor vehicle with some number of wheels, x, given the class “car”. The probability of having a motor vehicle with four wheels given Class “car” should be greater than the probability of a motor vehicle with two wheels given Class “car”.

The other term $P(\boldsymbol{\mathcal{B}}|\boldsymbol{\mathcal{A}})$ is called the posterior and provides the most probable group $\boldsymbol{\mathcal{B}}$ using the data example of $\boldsymbol{\mathcal{A}}$. With the labeled dataset, in Table 2.1, the most likely class for a motor vehicle with four wheels carrying five people is “car”. Using the unlabeled dataset in Table 2.2, the likely cluster for a motor vehicle with four wheels and five occupants is Cluster 1 if the data is separated using the maximum capacity. Partitioning the data by the number of wheels would also place this observation in Cluster 1.

2.1.3 Generative models

Generative models learn to represent the PDF using a set of data known as the training data. The model learns to represent the entire set of training data, and so unlabeled data may be used for this training. The generative model applies the learned PDF to create new sets of sample values with the same characteristics as the training dataset [Bishop, 2006]. To do this, the model calculates the likelihood of an observation, given a grouping. This likelihood, $P(\boldsymbol{\mathcal{A}}|\boldsymbol{\mathcal{B}})$, is multiplied with the priors of the cluster, $P(\boldsymbol{\mathcal{B}})$ to generate the joint PDF, $P(\boldsymbol{\mathcal{A}}, \boldsymbol{\mathcal{B}})$ [Bishop, 2006].

Suppose a generative model is trained using the training data from Table 2.2. If the data was partitioned by the number of wheels, and Cluster 1, more than two wheels, was selected, the generative model could return a possible maximum capacity of 10 adults. The model has learned that vehicles with more than two wheels can hold more adults than those with two wheels. In the case where the data is separated using the maximum capacity and Cluster 2, the high number of occupants, is selected, the generative model could return eight wheels. The model associates an increased number of occupants with an increase in the number of required wheels. This second example assumes the wheel increments are always in twos.

2.1.4 Discriminative models

In contrast to generative models, discriminative models learn to classify training data by calculating the posteriors $P(\boldsymbol{\mathcal{B}}|\boldsymbol{\mathcal{A}})$. For discriminative models, the data priors are not taken into account, and the model now has to learn fewer parameters [Bishop, 2006]. As a result, the discriminative model cannot create new examples like the generative model. However, the discriminative model does specialize in separating data classes by only modeling the posterior probabilities. This specialization also results from applying labeled training data when learning the model parameters.

Suppose that a discriminative model was trained with the labeled data in Table 2.1. From these labels, there are similarities between buses, cars, and motorcycles since the number of wheels is increasing in increments of two. However, in this sample dataset, enough differences exist in the maximum capacity to separate linearly the three classes, and the model should distinguish successfully between the three types of motor vehicles.

2.1.5 Training methods

Supervised and unsupervised training are the two major methods used to teach models about the training dataset by calculating model parameters. As generative models, RBMs originally were trained using unsupervised methods [Larochelle et al., 2012], [Larochelle and Bengio, 2008]. However, adaptations can be made that allow RBMs to be taught with supervised training [Larochelle et al., 2012],[Larochelle and Bengio, 2008]. Both training methods will be explained as a precursor to application in the context of RBMs.

Regardless of the type of training applied, the data is split into a training set and a test set. The training set is used to learn the model parameters to specify a model appropriate for the chosen situation. Model performance calculated on the test set evaluates how well the model can be applied to data previously unseen during training. Evaluation on the training data set does not represent how well the model can extrapolate its understanding of the data space to fit unseen cases. The goal is to have the trained model apply its understanding of the familiar environment to previously unseen data sets and to extrapolate from learned knowledge to new situations. Therefore, models' knowledge about their operating environments is determined by characteristics of the training data.

2.1.5.1 Supervised training In supervised training the labels, or classifications, for each data point are known. The goal for supervised training is to calculate the model parameters, the weights and biases, that produce the minimum value of an error-based cost function. The error is determined by the difference between the model's output based on its current parametric state and the labeled data. The purpose of minimizing the error on the training data set is to teach the model to recognize the classes well. This error is calculated for each training example, and training may be repeated for multiple iterations. Multiple criteria are used to determine when training completes such as the number of iterations over the training data and the model error achieving a value less than a predetermined threshold ϵ. With low error on the training set, it is hoped that the model will provide similar results for new data seen in the use cases.

Two typical error functions denoted by E and employed with supervised training are least mean squares and cross-entropy. For the equations below, the model outputs are represented by $\mathbf{x}$, while $\mathbf{y}$ illustrates the corresponding label for a single training example. The training set consists of P examples, and the total number of classes is represented using C. The subscripts i denote values of the $\mathbf{x}$ vector for discrete classes, and the index in parenthesis (m) refers to the individual training examples.

Least mean squares is an approximation of the expected value of the squared model errors, summed over all classes C. The probability of each error is given the same weighting $(\frac{1}{P})$. It is expected that there are enough training examples P so that $\frac{1}{P}$ approximates the true weighting for the probability of each error, $p(E(m))$. This ensures that the approximation by using the average of the squared error approaches the true expected value of the squared error [El-Jaroudi and Makhoul, 1990]. The equation below shows that least mean squares calculates the Euclidean distance between the current model answer $\mathbf{x}$ and the correct label $\mathbf{y}$.

$$E = \frac{1}{P}\sum_{m=1}^{P}\sum_{i=1}^{C}(x_i(m) - y_i(m))^2 \tag{2.2}$$

Since each error is weighted equally, larger differences affect the magnitude of the error term much more than smaller errors [El-Jaroudi and Makhoul, 1990]. For example, set $P = 1$ and $C = 1$. If $y = 3$ and $x = 0.5$, the least mean squares error equals 6.25. If $y = 0.1$ and $x = 0.01$, the error is 0.0081. The absolute differences between x and y are 2.5 for Example 1 and 0.09 for Example 2.

Achieving error values on the order of 0.0081 for consecutive training examples signifies that the overall model error is decreasing favorably. In a classification setting, probability values are desired to be exact, and the difference between 0.1 and 0.01 could be problematic. With least mean squares in the situation of Example 2, the model parameters may not be adjusted as much as required for proper classification. A higher rate of misclassification for unseen data could result.

One benefit of least mean squares is that restrictions are not imposed on the function arguments $\mathbf{x}$ and $\mathbf{y}$ by calculating squared error. Therefore, both the model outputs and the difference between the model values and the labels are allowed to be subsets of all $\boldsymbol{R}$. Least mean squares is appropriate for training models to perform linear regression since negative errors between the model and the ideal line of best fit may occur.

The cross-entropy error for two classes ($y \in \{0, 1\}$) is the negative log of the Bernoulli likelihood function [Bishop, 2006]. The derivation begins with the Bernoulli likelihood. The index m indicates that the error is calculated for each training example for all $m \in \{1, \cdots, P\}$. Taking the negative log of the likelihood produces the cross-entropy function for binary classes.

$$p(x(m)) = x(m)^{y(m)}(1 - x(m))^{(1-y(m))} \tag{2.3}$$

In general, the logarithm of the likelihood function is used for simplifications such as turning products into sums. Taking the log of a PDF describing a combination of independent random variables can help make parameter optimization easier. Differentiating sums is much simpler than taking a derivative of products.

For values of $x(m) > 0$, the $x(m)$ that maximizes the function $f(x)$ also maximizes $\ln[f(x)]$. Therefore, the same parameters maximize $f(x)$ and $\ln[f(x)]$.

$$E(m) = -y(m)\ \ln[x(m)] - (1 - y(m))\ \ln[1 - x(m)] \tag{2.4}$$

Summing over all P training examples produces the overall scalar error E.

$$E = \sum_{m=1}^{P} E(m) = -\sum_{m=1}^{P} \Big(y(m)\ \ln[x(m)] + (1 - y(m))\ \ln[1 - x(m)] \Big) \tag{2.5}$$

The equation can be generalized to multiple classes by assuming $y_1(m) = y(m)$, $x_1(m) = x(m)$, $y_2(m) = (1 - y(m))$, and $x_2(m) = (1 - x(m))$ and by summing over all classes i in $\{1, \cdots, C\}$.

$$E = -\sum_{m=1}^{P} \sum_{i=1}^{C=2} \Big(y_i(m) \;\ln[x_i(m)] \Big) \tag{2.6}$$

For $C \geq 2$:

$$E = -\sum_{m=1}^{P} \sum_{i=1}^{C} \Big(y_i(m) \;\ln[x_i(m)] \Big) \tag{2.7}$$

Cross-entropy is suited for classification tasks because the equation will magnify small differences between the targets $\mathbf{y}$ and model outputs $\mathbf{x}$ to be sizable errors [El-Jaroudi and Makhoul, 1990]. By continuing training, the errors could be further reduced. This error magnification is caused by weighting the error calculations using the ratios instead of the differences between the model output and target value. The proof involves dividing each term by the minimum value possible for cross-entropy and is explained in the following paragraphs [Theodoridis and Koutroumbas, 2008].

The minimum possible value for the error function occurs when the model output equals the target output, as shown. In the proofs, two classes and one example are assumed. When $y = 0$:

$$\begin{aligned} & x(m) = y(m) = 0 \\ & y(m) \;\ln[y(m)] = 0 \;\ln(0) = 0 \\ & (1 - y(m)) \;\ln[1 - y(m)] = (1 - 0) \;\ln[1 - 0] = 0 \end{aligned} \tag{2.8}$$

For $y = 1$:

$$\begin{aligned} & x(m) = y(m) = 1 \\ & y(m) \;\ln[y(m)] = 1 \;\ln[1] = 0 \\ & (1 - y(m)) \;\ln[1 - y(m)] = (1 - 1) \ln[1 - 1] = 0 \end{aligned} \tag{2.9}$$

Subtracting out the general form for the minimum value of cross-entropy gives the following equations [Theodoridis and Koutroumbas, 2008]. This general form of cross-entropy is equivalent to that in Equation 2.7 since the values subtracted equal 0. Two classes are assumed ($C = 2$).

$$\begin{aligned} E = -\textstyle\sum_{m=1}^{P} \Big(& y(m)\ \ln[x(m)] + (1 - y(m))\ \ln[1 - x(m)] \\ & -y(m)\ \ln[y(m)] - (1 - y(m))\ \ln[1 - y(m)] \Big) \end{aligned} \tag{2.10}$$

$$E = -\sum_{m=1}^{P} \left(y(m)\ \ln\left[\frac{x(m)}{y(m)}\right] + (1 - y(m))\ \ln\left[\frac{1 - x(m)}{1 - y(m)}\right] \right) \tag{2.11}$$

The next step generalizes the equation for more than two classes ($C \geq 2$).

$$E = -\sum_{m=1}^{P} \sum_{i=1}^{C} \left(y_i(m)\ \ln\left[\frac{x_i(m)}{y_i(m)}\right] \right) \tag{2.12}$$

The last two equations show how the ratio of the model output to the target value, $\frac{\mathbf{x}}{\mathbf{y}}$, is incorporated into the error. These next two examples illustrate the error magnification and complement the examples in the section explaining least mean squares.

For these examples, it is assumed that $P = 1$ and $C = 1$ and Equation 2.12 is used. If $y = 3$ and $x = 0.5$, the cross-entropy error equals 2.33. If $y = 0.1$ and $x = 0.01$, the error is 0.10. The values for Example 1 and Example 2 using least mean squares are 6.25 and 0.0081, respectively. Cross-entropy minimizes larger errors and magnifies smaller errors [El-Jaroudi and Makhoul, 1990].

2.1.5.2 Unsupervised training For unsupervised training, the training data labels are unknown. As with supervised training, unsupervised training is used to change the model weights and biases. Unlike in supervised training, the purpose of the model is to represent the training data, and the cost function is known as an energy function. These cost functions are minimized based on methods of calculating similarities between data points. Common distance metrics include the inner product calculation and the Euclidean distance [Sarle, 2002].

Unsupervised training leads to organizing the data points by groups or clusters. However, the meaning of the clusters remains unknown. An approximation of the probability density function is achieved since clusters are identified by measures of center and spread. With unsupervised training, there are no restrictions on the relationships for the input data, and new relationships between the data points may be discovered. Another benefit of unsupervised learning is the elimination of the time required to produce the labeled training set. The model energy for each training example is calculated for unsupervised training, and training may be required to reuse those same data points over multiple iterations. Similar criteria as used in supervised training are used to end unsupervised training.

2.1.5.3 Supervised versus unsupervised training Figure 2.1 shows an example of a simple linearly separable data set with two features, x_1 and x_2, and two classes denoted by circles and squares. Figure 2.2 divides the data according to the goals of supervised learning, so the line exactly separates the class of circles from the class of squares.

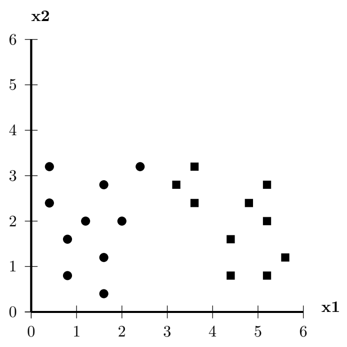


Figure 2.1: Example supervised training data set

Figure 2.3 illustrates the same data set as Figure 2.1, but the classes are unknown, and the data is shown using only circles. Figure 2.4 shows alternate methods of dividing the same data set. Since the class information is not used in subdividing the data, the training results depend on how well each point fits in each of the data groups. The difference in hyperplane placement for unsupervised training depends on the critical points shown in Figure 2.5.

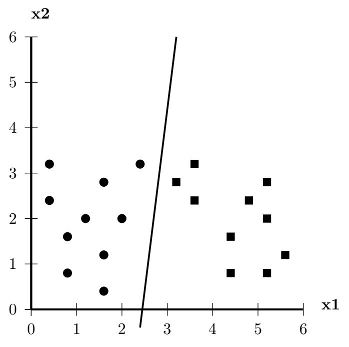


Figure 2.2: Example divided supervised training data set

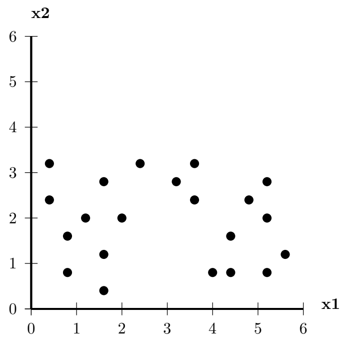


Figure 2.3: Example unsupervised training data set

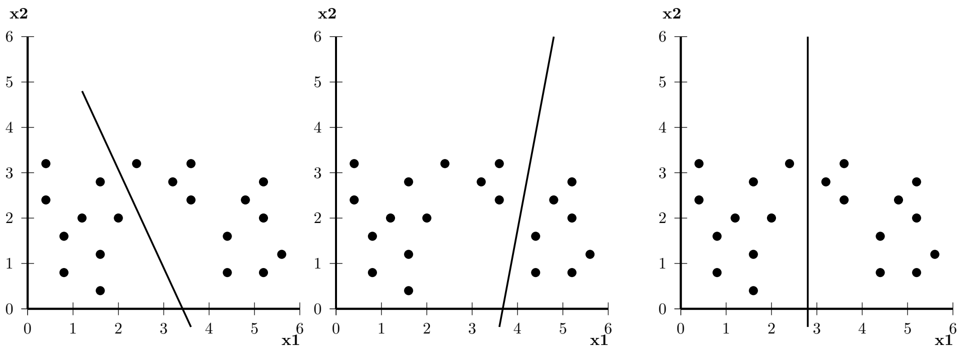


Figure 2.4: Examples of a divided unsupervised training data set

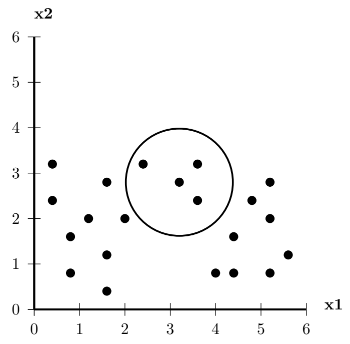


Figure 2.5: Highlighting four points

Unsupervised learning is useful because the shape of the data set is learned via methods such as k-means, mean-shift, maximum likelihood, and expectation maximization [Theodoridis and Koutroumbas, 2008], [Bishop, 2006]. The parameters calculated from unsupervised training are based on the data set's natural groupings and may be used to initialize models for supervised training.

2.2 DEFINITION OF AN RBM

The RBM has two types of nodes, visible and hidden [Fischer and Igel, 2012]. Visible nodes take inputs either directly from the data set or calculated probabilistically using a function of linear combinations of hidden node values. Hidden nodes produce values calculated stochastically from a function of a linear combination of the visible nodes. RBM parameters consist of one set of weights and two sets of biases. Unlike typical feedfoward NN models, the connections between the two sets of nodes are bidirectional. This characteristic provides the RBM with two distinct functions depending on the direction of information flow [Fischer and Igel, 2012]. This section describes the mechanics of RBMs such as the graphical structure and the steps necessary to create an RBM to describe a dataset. These details are provided before describing the different operations performed by an RBM (Section 2.3).

2.2.1 Graphical structure

A graphical model with undirected connections such as the RBM is known as a Markov random field (MRF) [Welling et al.]. The Markov property states that the values of variables currently being evaluated depend only on the immediate set of previously determined values. In reference to time, this property states that the values calculated at time t depend only on the values calculated at $t-1$ [Bishop, 2006]. For RBMs, this means that the visible node values depend on the hidden node values and vice versa.

A random field is a mathematical concept that applies random variables to a spatial arrangement. Random processes are limited to time variations between random variables, but random fields could have the random vectors relating to one another spatially [Chung, 2007]. For example, with RBMs, some relationships between the visible node values for the motor vehicle datasets are determined by vehicle size. Larger vehicles usually have more wheels and may carry more occupants. This illustrates that the relationships found by hidden nodes do not have to be time-dependent.

For an RBM, the letters **b** and **c** represent the bias vectors for the visible and hidden layers, respectively. The letter **W** represents the weight matrix connecting the visible and hidden layers. Assuming that there are N nodes in the visible layer and M nodes in the hidden layer the sizes of the **b** vector, **c** vector, and **W** matrix are indicated. The size of each layer's bias vector is determined by the number of nodes in that layer.

$$\begin{aligned} &\mathbf{b}, \mathbf{v} : [N \times 1] \\ &\mathbf{c}, \mathbf{h} : [M \times 1] \\ &\mathbf{W} : [N \times M] \end{aligned} \tag{2.13}$$

Figures 2.6 and 2.7 show an example RBM as well as a generalized model known as a Boltzmann machine. In these figures, the visible nodes are yellow, and the hidden nodes are blue. Double headed arrows illustrate the undirected connections between the nodes. In the structure of Figure 2.6, each hidden node is connected to all visible nodes and vice versa. No nodes within a single layer are connected (no hidden-hidden or visible-visible connections) [Fischer and Igel, 2012]. This graphical structure is classified as being bipartite, since there are two distinct groups of nodes in the graph, visible nodes and hidden nodes [Geoffrey Hinton, 2013]. The elimination of within layer connections is the source of the term "restricted" in the RBM name. In contrast, a generic Boltzmann machine (BM), shown in Figure 2.7, includes all possible connections between nodes [Fischer and Igel, 2012].

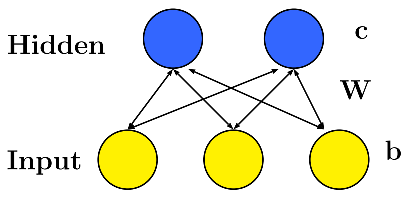


Figure 2.6: Restricted Boltzmann machine (RBM)

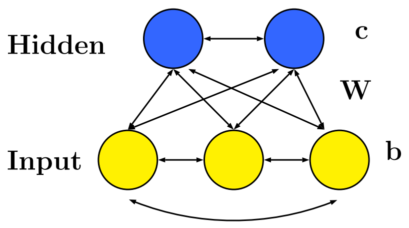


Figure 2.7: Boltzmann machine (BM)

The lack of physical connection between any two nodes indicates the absence of a relationship, and the nodes in a given layer are viewed as being statistically independent of one another. In the RBM there is not complete independence between nodes in any one layer since connections exist between a single node and all nodes in the opposite layer. Rather, nodes in a given layer are linked via the opposite layer [Fischer and Igel, 2012]. This property is known as conditional independence.

Due to conditional independence, evaluating $p(\mathbf{v}|\mathbf{h})$, the probability of a combination of visible node values given the hidden node values, or $p(\mathbf{h}|\mathbf{v})$, the probability of a combination of hidden node values given the visible node values, requires multiplication. Equations 2.14 and 2.15 illustrate the calculation of these conditional probabilities.

$$p(\mathbf{h}|\mathbf{v}) = \prod_{j=1}^{M} p(h_j|\mathbf{v}) \tag{2.14}$$

$$p(\mathbf{v}|\mathbf{h}) = \prod_{i=1}^{N} p(v_i|\mathbf{h}) \tag{2.15}$$

2.2.2 RBM PDFs

The values of each RBM node are described by a PDF. Usually all nodes in a given layer apply the same type of PDF, although each node could be modeled by a different type of distribution [Welling et al.]. Distributions from the exponential family such as Bernoulli, binomial, exponential, Gaussian, and Poisson are often chosen as node models since these distributions are commonly used in modeling data collected from nature [Jebara].

Initial efforts in machine learning with RBM models represented both the visible and the hidden node values with Bernoulli distributions [Hinton, 2010]. A common application for RBMs is image processing [Hinton, 2010],[Fischer and Igel, 2012]. For simplicity, black, white, or gray pixels were used for the images, restricting the pixel values to the range of $[0, 1]$. The pixel values were the RBM inputs using the visible nodes, so the allowed range of values matches with the Bernoulli distribution's absolute values and probability values, $[0, 1]$ [Hinton, 2010]. The hidden layer Bernoulli distributions were used for finding groups in the data [Hinton, 2010]. Calculated values for the hidden nodes represent the probabilities of visible node values possessing a given relationship among each other [Chen, 2011].

For applications where the data on the visible nodes should not be confined to be binary values, Gaussian or other distributions may be used [Welling et al.], [Hinton, 2010]. The derivation of the probability distribution function for a Bernoulli-Bernoulli RBM is provided in Section 2.2.2.3. A similar derivation for a Gaussian-Bernoulli RBM is available in Appendix A.2. These derivations are based on explanations of generalized RBM equations for the exponential distribution family from [Welling et al.]. The notation used was taken from both [Welling et al.] and [Jebara].

2.2.2.1 Generalized RBM derivation After selecting the distribution family for individual RBM nodes, chosen to match the types of data represented by each node, the overall RBM PDF is constructed. At this point, the property of conditional independence is used to combine individual PDFs via multiplication. Two products of terms are formed, one for the visible layer and one for the hidden layer. The term with the weights describing the interaction between the two layers is added by definition from the graphical model (Figure 2.6) [Welling et al.]. The connection weights indicate the proportion for which each node value affects the other layer's values.

The vector of visible nodes is shown using $\mathbf{v}$ with individual nodes denoted with subscript i, as v_i. Similarly for the vector of hidden nodes, $\mathbf{h}$ and subscript j, as with h_j, are used. It is assumed that there are N visible nodes and M hidden nodes. Applying the notation of [Welling et al.] in this set of equations, the $r_i(v_i)$ and $s_j(h_j)$ terms represent Lebesgue measures for the individual visible and hidden node PDFs [Blei, 2011]. In the context of both Bernoulli and Gaussian distributions, the effect of these terms is to help the PDF integrate to one as required. The number of parameters per PDF is indicated using the subscripts a for visible nodes and b for hidden nodes. The parameter sets for individual visible and hidden nodes are denoted using $\boldsymbol{\theta_i}$ and $\boldsymbol{\lambda_j}$. The PDF sufficient statistics for individual visible and hidden nodes are indicated using $\boldsymbol{f(v_i)}, \boldsymbol{g(h_j)}$. The logarithmic normalization factors, A_i, B_j, also ensure the distribution equations integrate or sum to one.

Below is the marginal PDF for the visible nodes.

$$p(\mathbf{v}) = \prod_{i=1}^{N} r_i(v_i) \exp\Big[\sum_a [\theta_{ia} f_a(v_i)] - A_i(\boldsymbol{\theta_i})\Big] \tag{2.16}$$

Equation 2.17 shows the marginal PDF for the hidden nodes.

$$p(\mathbf{h}) = \prod_{j=1}^{M} s_j(h_j) \exp\Big[\sum_b [\lambda_{jb} g_b(h_j)] - B_j(\boldsymbol{\lambda_j})\Big] \tag{2.17}$$

Equation 2.18 describes the joint PDF for the complete RBM without the normalization term [Welling et al.]. This value is usually intractable to compute, since the complexity of this term grows exponentially with the total number of nodes, $N + M$ [Fischer and Igel, 2012].

$$p(\mathbf{v}, \mathbf{h}) \propto \exp\Big[\sum_{i,a} [\theta_{ia} f_a(v_i)] + \sum_{j,b} [\lambda_{jb} g_b(h_j)] + \sum_{i,j,a,b} W_{ia}^{jb} [f_a(v_i) g_b(h_j)]\Big] \tag{2.18}$$

2.2.2.2 Exponential family derivations The generalized equation for all exponential family PDFs is provided below. These equations are shown for a single random variable, x_i, for simplicity. In this case, x_i could represent either the visible or hidden node values. This equation is the starting point for derivations to obtain the common forms of the Bernoulli or Gaussian PDFs.

$$p(x_i) = r_i(x_i) \exp\Big[\boldsymbol{\theta}_i^T \boldsymbol{f(x_i)} - A(\boldsymbol{\theta}_i)\Big] \tag{2.19}$$

Starting from the above equation, the specific parameters for the Bernoulli PDF are given in Equation 2.20. Substituting in the Bernoulli parameters leads to Equation 2.21.

$$\begin{aligned} & r_i(x_i) = 1 \\ & \boldsymbol{\theta}_i = \ln(\tfrac{\alpha_i}{1-\alpha_i}) \\ & \boldsymbol{f(x_i)} = x_i \\ & A(\boldsymbol{\theta}_i) = \ln(1 + e^{\boldsymbol{\theta}_i}) \end{aligned} \tag{2.20}$$

$$p(x_i) = [1] \;\exp\left[\;\ln\left(\frac{\alpha_i}{1-\alpha_i}\right)x_i - \;\ln\left(1+\exp\left(\ln\left(\frac{\alpha_i}{1-\alpha_i}\right)\right)\right)\right] \tag{2.21}$$

The steps to prove that the general exponential form can be derived from the above equation are found in A.1.1. Below is the typical form of the Bernoulli equation.

$$p(x_i) = \alpha_i^{x_i}(1-\alpha_i)^{1-x_i}; x_i \in \{0,1\} \tag{2.22}$$

A similar procedure can be followed for obtaining the Gaussian PDF, and so the Gaussian parameters are provided

$$\begin{aligned} r_i(x_i) &= \tfrac{1}{\sqrt{2\pi}} \\ \boldsymbol{\theta}_i &= \left[\tfrac{\mu_i}{\sigma_i^2}, \tfrac{-1}{2\sigma_i^2}\right] \\ \boldsymbol{f(x_i)} &= \left[x_i, x_i^2\right] \\ A(\boldsymbol{\theta}_i) &= \tfrac{\mu_i^2}{2\sigma_i^2} + \ln(\sigma_i) \end{aligned} \tag{2.23}$$

Substituting in the Gaussian parameters leads to the following equation.

$$p(x_i) = \frac{1}{\sqrt{2\pi}} \exp\left(\frac{x_i\mu_i}{\sigma_i^2} - \frac{x_i^2}{2\sigma_i^2} - \frac{\mu_i^2}{2\sigma_i^2} - \ln(\sigma_i)\right) \tag{2.24}$$

The steps to prove that the general exponential form can be derived from the Gaussian PDF are found in A.1.2. Equation 2.25 shows the typical form of the Gaussian equation.

$$p(x_i) = \frac{1}{\sqrt{2\pi\sigma_i^2}} \exp\left(-\frac{||x_i-\mu_i||^2}{2\sigma_i^2}\right); x_i \in \boldsymbol{R} \tag{2.25}$$

2.2.2.3 Bernoulli-Bernoulli RBM Using Equations 2.18 and 2.20 the joint PDF for a Bernoulli-Bernoulli RBM is derived as follows, and the derivation of the joint PDF for the Gaussian-Bernoulli RBM is provided in Appendix A.2. For this derivation, the number of PDF parameters is 1 for each layer ($a = 1, b = 1$).

$$\begin{aligned}\boldsymbol{\theta}_i &= \theta_{ia} = \ln\left(\tfrac{\alpha_i}{1-\alpha_i}\right)\\ \boldsymbol{\lambda}_j &= \lambda_{jb} = \ln\left(\tfrac{\beta_j}{1-\beta_j}\right)\end{aligned} \tag{2.26}$$

$$\begin{aligned}\boldsymbol{f(v_i)} &= f_a(v_i) = v_i\\ \boldsymbol{g(h_j)} &= g_b(h_j) = h_j\end{aligned} \tag{2.27}$$

Plugging into Equation 2.18 yields:

$$p(\mathbf{v},\mathbf{h}) \propto \exp\left[\sum_i \ln\left(\frac{\alpha_i}{1-\alpha_i}\right) v_i + \sum_j \ln\left(\frac{\beta_j}{1-\beta_j}\right) h_j + \sum_i \sum_j W_{ij} v_i h_j\right] \tag{2.28}$$

Set $b_i = \ln\left(\frac{\alpha_i}{1-\alpha_i}\right)$ and $c_j = \ln\left(\frac{\beta_j}{1-\beta_j}\right)$. This gives the final equation for the Bernoulli-Bernoulli joint PDF.

$$p(\mathbf{v},\mathbf{h}) \propto \exp\left[\sum_i b_i v_i + \sum_j c_j h_j + \sum_i \sum_j W_{ij} v_i h_j\right] \tag{2.29}$$

In the next step, the previous equation is rewritten in vector form.

$$p(\mathbf{v},\mathbf{h}) \propto \exp\left[\mathbf{b}^T\mathbf{v} + \mathbf{c}^T\mathbf{h} + \mathbf{v}^T\mathbf{W}\mathbf{h}\right] \tag{2.30}$$

2.2.2.4 The Boltzmann distribution The Boltzmann distribution is an energy based probability distribution, defined over a set of states [Emberly]. In Equation 2.31, E refers to an energy configuration, k_B is the Boltzmann constant, and T is the system temperature.

$$p(E) = \frac{e^{\frac{-E}{k_B T}}}{\sum_{E'} e^{\frac{-E'}{k_B T}}} \tag{2.31}$$

The decaying exponential term of the unnormalized Boltzmann distribution is illustrated in Figure 2.8. Showing the relationship between probability and energy, this figure indicates that the system mimics natural systems' behavior by applying a favorable bias toward low energy configurations. For RBM systems the $k_B T$ term is set equal to 1. The state energy becomes $E(\mathbf{v}, \mathbf{h})$ and depends on the values of the visible and hidden nodes for a given set of weights and biases.

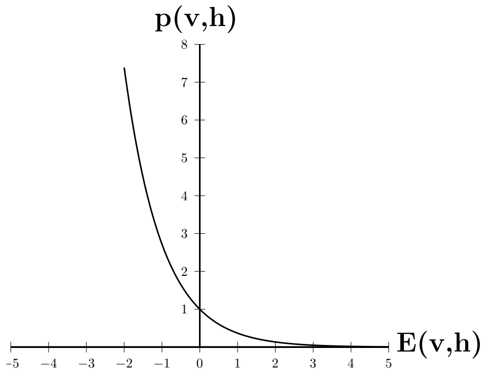


Figure 2.8: Unnormalized Boltzmann distribution

Equation 2.32 gives the joint PDF of the RBM for the combination of visible and hidden nodes. The Z term is used as a normalization factor to ensure that the discrete Bernoulli-Bernoulli PDF sums to one.

$$p(\mathbf{v}, \mathbf{h}) = \frac{e^{-E(\mathbf{v},\mathbf{h})}}{Z} \propto e^{-E(\mathbf{v},\mathbf{h})} \tag{2.32}$$

The normalizing Z term sums over all of the RBM states.

$$Z = \sum_{\mathbf{v}'} \sum_{\mathbf{h}'} e^{-E(\mathbf{v}',\mathbf{h}')} \tag{2.33}$$

Summing over the hidden nodes gives the marginal PDF for the visible nodes.

$$p(\mathbf{v}) = \sum_{\mathbf{h}} p(\mathbf{v}, \mathbf{h}) = \sum_{\mathbf{h}} \frac{e^{-E(\mathbf{v},\mathbf{h})}}{Z} \tag{2.34}$$

Obtaining the marginal hidden nodes' PDF requires summing over the visible nodes.

$$p(\mathbf{h}) = \sum_{\mathbf{v}} p(\mathbf{v}, \mathbf{h}) = \sum_{\mathbf{v}} \frac{e^{-E(\mathbf{v},\mathbf{h})}}{Z} \tag{2.35}$$

2.2.2.5 The energy equation

Comparing the exponential term argument of Equation 2.30 with that of the Boltzmann distribution definition in Equation 2.32 illustrates how to extract the energy function for the Bernoulli-Bernoulli RBM.

$$\begin{aligned} &p(\mathbf{v}, \mathbf{h}) \propto \exp[\mathbf{b}^T\mathbf{v} + \mathbf{c}^T\mathbf{h} + \mathbf{v}^T\mathbf{W}\mathbf{h}] \\ &p(\mathbf{v}, \mathbf{h}) \propto \exp[-E(\mathbf{v}, \mathbf{h})] \end{aligned} \tag{2.36}$$

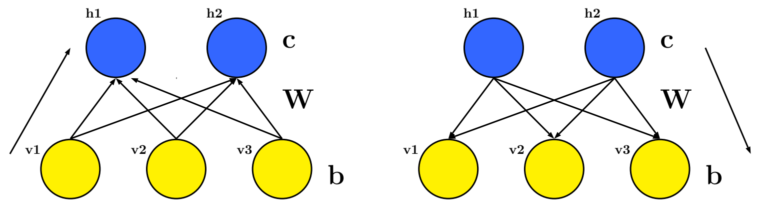
It can be shown that $-E(\mathbf{v}, \mathbf{h}) = \mathbf{b}^T\mathbf{v} + \mathbf{c}^T\mathbf{h} + \mathbf{v}^T\mathbf{W}\mathbf{h}$.

Summing over $i \in \{1, \cdots, N\}$ and $j \in \{1, \cdots, M\}$ produces the scalar equation.

$$E(\mathbf{v}, \mathbf{h}) = -\sum_{i=1}^{N} b_i v_i - \sum_{j=1}^{M} c_j h_j - \sum_{i=1}^{N}\sum_{j=1}^{M} v_i W_{ij} h_j \tag{2.37}$$

2.3 FUNCTIONS OF AN RBM

As mentioned previously, an RBM's bidirectional nature provides the model with two functions [Chen, 2011], [Fischer and Igel, 2012], [Hinton, 2010], [Welling et al.]. In general, the function that results when moving from the visible nodes to the hidden nodes can be termed as the forward operation, modeled by the conditional probability $p(\mathbf{h}|\mathbf{v})$. Correspondingly, the reverse direction from hidden nodes to visible nodes demonstrates the backward operation and is calculated with the conditional probability $p(\mathbf{v}|\mathbf{h})$. Figure 2.9 below illustrates the forward direction using the RBM on the left and the backward direction with the RBM on the right.



(a) Forward Direction (b) Backward Direction

Figure 2.9: Directions for RBM operations

Figure 2.10 gives a generic illustration of the RBM data space. As with a neural network, the N visible nodes determine the spatial dimensions, indicated by the axes v_1, v_2, and v_3. Each of the M hidden nodes creates a hyperplane. For this example, there are two hidden nodes and two hyperplanes. These hyperplanes separate the data space into four sections. Two sample data clusters, located in different quadrants, are represented using blue squares and pink diamonds. Applying the example from Table 2.2, these blue squares could represent vehicles with low maximum occupancy and fewer wheels, and the pink diamonds may indicate vehicles with higher maximum occupancy and more wheels.

The overall purpose of an RBM with Bernoulli hidden nodes is described by logistic regression, a mapping between data and binary-valued clusters. This section begins with logistic regression. Specifics for the forward operation and the backward operation follow.

2.3.1 Logistic regression

Logistic regression seeks to achieve a mapping between a set of values and a set of binary states, which match the sampled values of Bernoulli hidden nodes [Shalizi, 2012]. Therefore, for any RBM with Bernoulli hidden nodes, the relationship from the visible layer to the hidden layer can be represented using logistic regression. To account for uncontrolled, noisy input values, the most probable state should be indicated, explaining why hidden node values range from $[0, 1]$.

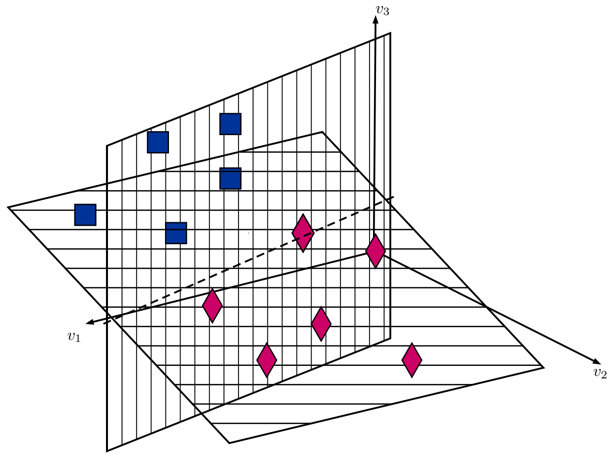


Figure 2.10: Graphical illustration of RBM functions

Therefore, logistic regression determines the expected probability-like value that any linear combination of noisy inputs belongs to a certain state [Shalizi, 2012]. The fractional value of the hidden node can be sampled by comparison to a random value from a uniform distribution to produce the final state decision.

Producing a mapping function from any continuous or discrete input to the desired output range of $[0, 1]$ requires a transition to occur between the two states, $\{0, 1\}$. From [Shalizi, 2012], it was desired to apply a linear function in some way to model the behavior of this mapping function as with linear regression. However, a linear function does not completely describe the situation. The outputs must tend to 0 and 1 asymptotically for extremely large absolute values of the inputs. Experimentation revealed that a larger change in the input is required to achieve a change of the same magnitude for the output towards the asymptotes [Shalizi, 2012]. The logistic transform $\ln\left(\frac{p(\mathbf{x})}{1-p(\mathbf{x})}\right)$ produces these desired behaviors and is set equal to a linear function $\beta_0 + \mathbf{x} \cdot \boldsymbol{\beta}$ [Shalizi, 2012]. This function, shown in Equation 2.38, has an infinite domain over $(-\infty, +\infty)$.

$$\ln\left(\frac{p(\mathbf{x})}{1-p(\mathbf{x})}\right) = \beta_0 + \mathbf{x} \cdot \boldsymbol{\beta} \tag{2.38}$$

The right side of this logistic function is a hyperplane in N dimensions, where $N + 1$ equals the number of parameters. The inputs come from the generic vector $\mathbf{x}$, and the hyperplane parameters are from the vector $\boldsymbol{\beta}$. When the hyperplane equals zero, both state probabilities, $p(\mathbf{x})$ and $1 - p(\mathbf{x})$, are equal, so both states have the same chance of being selected. When the value of the hyperplane is greater than zero, $p(\mathbf{x})$ is greater than $1-p(\mathbf{x})$ and vice versa when the hyperplane is less than zero.

Taking the exponent of each side cancels out the natural logarithm.

$$\left(\frac{p(\mathbf{x})}{1 - p(\mathbf{x})}\right) = e^{\beta_0 + \mathbf{x}\cdot\boldsymbol{\beta}} \tag{2.39}$$

Multiplying both sides by the denominator, and distributing the right hand side results in Equation 2.40.

$$p(\mathbf{x}) = e^{\beta_0 + \mathbf{x}\cdot\boldsymbol{\beta}} - p(\mathbf{x})e^{\beta_0 + \mathbf{x}\cdot\boldsymbol{\beta}} \tag{2.40}$$

Solving for $p(\mathbf{x})$ produces an equation resembling the Boltzmann distribution [Shalizi, 2012]. To match the Boltzmann distribution exactly, the energy function is assumed to be $E(\mathbf{x}) = -\beta_0 - \mathbf{x}\cdot\boldsymbol{\beta}$, and the allowed values for a single node x_i are assumed to be restricted to $\{0, 1\}$. This makes the denominator match the partition function Z.

$$p(\mathbf{x}) = \frac{e^{\beta_0 + \mathbf{x}\cdot\boldsymbol{\beta}}}{1 + e^{\beta_0 + \mathbf{x}\cdot\boldsymbol{\beta}}} \tag{2.41}$$

This function can be further reduced to match the definition of the sigmoid function, given in Equation 2.42 and shown in Figure 2.11.

$$\sigma(u) = \frac{1}{1 + e^{-u}} \tag{2.42}$$

The sigmoid function for logistic regression is used to calculate the probability $p(x)$ of one state over a second state and is derived from this equation by multiplying the numerator and denominator of Equation 2.41 by $e^{-\beta_0 - \mathbf{x}\cdot\boldsymbol{\beta}}$ [Shalizi, 2012].

$$p(\mathbf{x}) = \frac{1}{1 + e^{-(\beta_0 + \mathbf{x}\cdot\boldsymbol{\beta})}} = \sigma(\beta_0 + \mathbf{x}\cdot\boldsymbol{\beta}) \tag{2.43}$$

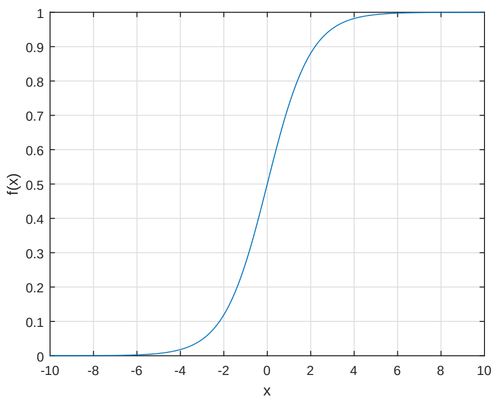


Figure 2.11: Sigmoid function

Also used in the hidden layers of NNs, the sigmoid function is an activation function that indicates the commonalities using binary values, calculated between values in the previous layers. For more than two states, the sigmoid function can be generalized to the softmax function as shown in Appendix A.5.

These commonalities calculated among the components of the input vector $\mathbf{x}$, are determined by the dot product, $\beta_0 + \mathbf{x} \cdot \boldsymbol{\beta}$, between the inputs and hyperplane parameters. The $\mathbf{x}$ vector is compared with $\boldsymbol{\beta}$, also the hyperplane normal vector. Resulting nonzero dot products indicate that $\mathbf{x}$ is not perpendicular to $\boldsymbol{\beta}$. For all positive dot products, $\mathbf{x}$ is oriented in a similar direction as $\boldsymbol{\beta}$. For all negative dot products, $\mathbf{x}$ is in the opposite direction of $\boldsymbol{\beta}$. If a dot product is zero, $\mathbf{x}$ is parallel to the plane and cannot be classified by it. Therefore, a differently oriented hyperplane would be required to classify $\mathbf{x}$.

2.3.2 The forward direction

The forward RBM function, similar to that of feedforward NNs, calculates commonalities between the input data values of the visible nodes [Chen, 2011]. This function corresponds to calculating the posterior values, $p(\mathbf{h}|\mathbf{v})$, which indicate belonging to different clusters, also known as latent factors [Chen, 2011]. The number of hidden nodes selected for the model equals the number of latent factors that will be discovered in the data. Since RBM data are unlabeled, the exact meanings of the latent factors are generally unknown [Chen, 2011].

For the example from Tables 2.1 and 2.2, the hidden node meanings are surmised from previous knowledge of the motor vehicle dataset labels. An RBM describing the data in Table 2.2 could have two visible nodes and one hidden node. Assuming hidden nodes take binary values, the zero state indicates that the visible data does not belong to a cluster, while a state of one indicates the opposite. The latent factor assumed for the example is vehicle size. Zero indicates a smaller vehicle size, and one indicates a larger vehicle size. Assuming the vehicle has two wheels and can carry two passengers would probably lead to a value of zero for the hidden node indicating smaller vehicle size. Figure 2.12 illustrates a possible configuration of the space for this specific example.

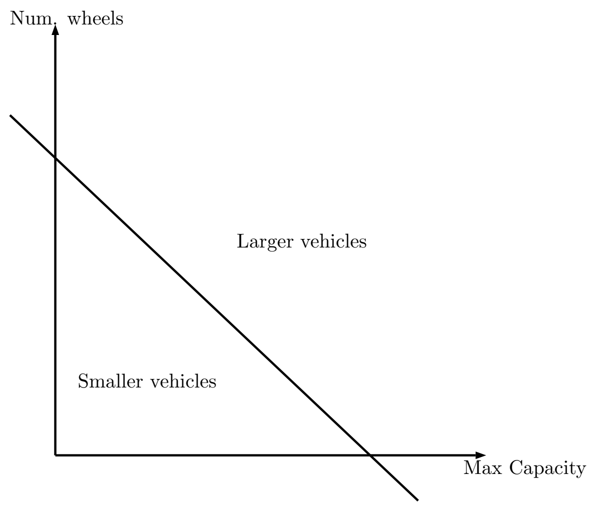


Figure 2.12: Potential hyperplane for vehicle sample space

2.3.2.1 Calculating $p(\mathbf{h}|\mathbf{v})$

The steps for determining the PDF for $p(\mathbf{h}|\mathbf{v})$ will be shown in this section. Initially, the equation for $p(\mathbf{h}|\mathbf{v})$ is derived from the definition of conditional PDFs.

$$p(\mathbf{h}|\mathbf{v}) = \frac{p(\mathbf{v}, \mathbf{h})}{p(\mathbf{v})} \tag{2.44}$$

Next, both the joint Boltzmann distribution and the marginal Boltzmann distribution for the visible nodes are plugged into the right hand side.

$$p(\mathbf{h}|\mathbf{v}) = \frac{\frac{1}{Z} e^{-E(\mathbf{v},\mathbf{h})}}{\frac{1}{Z} \sum_{\mathbf{h}'} e^{-E(\mathbf{v},\mathbf{h}')}} \tag{2.45}$$

The vector form of the Bernoulli-Bernoulli energy equation is applied, and the partition function can be canceled from both the numerator and the denominator as shown in Equation 2.46.

$$p(\mathbf{h}|\mathbf{v}) = \frac{\exp\left(-(-\mathbf{b}^T\mathbf{v} - \mathbf{c}^T\mathbf{h} - \mathbf{v}^T\mathbf{W}\mathbf{h})\right)}{\sum_{\mathbf{h}'} \exp\left(-(-\mathbf{b}^T\mathbf{v} - \mathbf{c}^T\mathbf{h}' - \mathbf{v}^T\mathbf{W}\mathbf{h}')\right)} \tag{2.46}$$

To be able to make further simplifications, the energy equation is transformed into its scalar equivalent form.

$$p(\mathbf{h}|\mathbf{v}) = \frac{\exp\left(\sum_{i=1}^{N} v_i b_i + \sum_{j=1}^{M} h_j c_j + \sum_{i=1}^{N}\sum_{j=1}^{M} W_{ij} v_i h_j\right)}{\sum_{\mathbf{h}'} \exp\left(\sum_{i=1}^{N} v_i b_i + \sum_{j=1}^{M} h'_j c_j + \sum_{i=1}^{N}\sum_{j=1}^{M} W_{ij} v_i h'_j\right)} \tag{2.47}$$

Common terms that can be canceled from the numerator and denominator are separated from the rest of the terms.

$$p(\mathbf{h}|\mathbf{v}) = \frac{\exp\left(\sum_{i=1}^{N} v_i b_i\right) \exp\left(\sum_{j=1}^{M} h_j c_j + \sum_{i=1}^{N}\sum_{j=1}^{M} W_{ij} v_i h_j\right)}{\exp\left(\sum_{i=1}^{N} v_i b_i\right) \sum_{\mathbf{h}'} \exp\left(\sum_{j=1}^{M} h'_j c_j + \sum_{i=1}^{N}\sum_{j=1}^{M} W_{ij} v_i h'_j\right)} \tag{2.48}$$

The common terms $\exp\left(\sum_{i=1}^{N} v_i b_i\right)$ are canceled. Properties of exponents are applied to turn exponents of sums into products of exponents, and the h_j variable is pulled out from the remaining terms. This step will lead to the proof that the hidden nodes are conditionally independent from one another.

$$p(\mathbf{h}|\mathbf{v}) = \frac{\prod_{j=1}^{M} \exp\left(h_j\left(c_j + \sum_{i=1}^{N} W_{ij} v_i\right)\right)}{\sum_{\mathbf{h}'} \prod_{j=1}^{M} \exp\left(h'_j\left(c_j + \sum_{i=1}^{N} W_{ij} v_i\right)\right)} \tag{2.49}$$

In this step, the vector sum $\sum_{\mathbf{h}'}$ is expanded over all M nodes. These steps will help to expand and then simplify the denominator.

$$p(\mathbf{h}|\mathbf{v}) = \frac{\prod_{j=1}^{M} \exp\left(h_j\left(c_j+\sum_{i=1}^{N} W_{ij} v_i\right)\right)}{\sum_{h'_1} \cdots \sum_{h'_M} \left[\exp\left(h'_1\left(c_1+\sum_{i=1}^{N} W_{i1} v_i\right)\right)\right] \cdots \left[\exp\left(h'_M\left(c_M+\sum_{i=1}^{N} W_{iM} v_i\right)\right)\right]} \tag{2.50}$$

The sums and products can be matched together for the individual hidden nodes. For example, when calculating the sum over h'_1, only h'_1 changes. The other hidden node variables, $h'_2, \cdots, h'_M$, are constants unaffected by the summation over h'_1 and can be pulled from the sum over h'_1. The same process is repeated for h'_2 through h'_M. A similar proof for integrals is provided in Appendix A.4.

$$p(\mathbf{h}|\mathbf{v}) = \frac{\prod_{j=1}^{M} \exp\left(h_j\left(c_j+\sum_{i=1}^{N} W_{ij} v_i\right)\right)}{\left[\sum_{h'_1} \exp\left(h'_1\left(c_1+\sum_{i=1}^{N} W_{i1} v_i\right)\right)\right] \cdots \left[\sum_{h'_M} \exp\left(h'_M\left(c_M+\sum_{i=1}^{N} W_{iM} v_i\right)\right)\right]} \tag{2.51}$$

To obtain an equation that illustrates the conditional independence of the hidden nodes, the product over j should be common for both the numerator and the denominator. With reorganizing the denominator sums and products in the previous step, a product for individual terms can be written and then pulled out of both terms.

$$p(\mathbf{h}|\mathbf{v}) = \prod_{j=1}^{M} \frac{\exp\left(h_j\left(c_j + \sum_{i=1}^{N} W_{ij} v_i\right)\right)}{\sum_{h'_j} \exp\left(h'_j\left(c_j + \sum_{i=1}^{N} W_{ij} v_i\right)\right)} \tag{2.52}$$

This next step expresses the two possible values $\{0,1\}$ for h'_j in the denominator for Bernoulli hidden nodes.

$$p(\mathbf{h}|\mathbf{v}) = \prod_{j=1}^{M} \frac{\exp\left(h_j\left(c_j + \sum_{i=1}^{N} W_{ij}v_i\right)\right)}{\exp\left((0)\left(c_j + \sum_{i=1}^{N} W_{ij}v_i\right)\right) + \exp\left((1)\left(c_j + \sum_{i=1}^{N} W_{ij}v_i\right)\right)} \tag{2.53}$$

Conditional independence is reached. The probability of an individual h_j value can be calculated, while the overall probability depends on the visible layer values.

$$p(\mathbf{h}|\mathbf{v}) = \prod_{j=1}^{M} \frac{\exp\left(h_j\left(c_j + \sum_{i=1}^{N} W_{ij}v_i\right)\right)}{1 + \exp\left(c_j + \sum_{i=1}^{N} W_{ij}v_i\right)} \tag{2.54}$$

Set $p(h_j|\mathbf{v}) = \frac{\exp\left(h_j\left(c_j + \sum_{i=1}^{N} W_{ij}v_i\right)\right)}{1+\exp\left(c_j+\sum_{i=1}^{N} W_{ij}v_i\right)}$ for simplification.

$$p(\mathbf{h}|\mathbf{v}) = \prod_{j=1}^{M} p(h_j|\mathbf{v}) \tag{2.55}$$

For any hidden node j, this equation can be further simplified to the sigmoid function when solving for $p(H_j = 1|\mathbf{v})$ and starting from Equation 2.54.

$$p(H_j = 1|\mathbf{v}) = \frac{\exp\left([1]\left(c_j + \sum_{i=1}^{N} W_{ij}v_i\right)\right)}{1 + \exp\left(\left(c_j + \sum_{i=1}^{N} W_{ij}v_i\right)\right)} \tag{2.56}$$

The general form of the sigmoid function results when multiplying both the numerator and the denominator by $\exp\left(-\left(c_j + \sum_{i=1}^{N} W_{ij}v_i\right)\right)$.

$$p(H_j = 1|\mathbf{v}) = \frac{1}{\exp\left(-\left(c_j + \sum_{i=1}^{N} W_{ij}v_i\right)\right) + 1} = \sigma\left(c_j + \sum_{i=1}^{N} W_{ij}v_i\right) \tag{2.57}$$

2.3.3 The backward direction

The backward RBM function, unlike feedforward NNs, is used to calculate the likelihood, $p(\mathbf{v}|\mathbf{h})$, and to generate new data. Following the same motor vehicle example, the hidden node value indicating a specific cluster would be provided. Values for the visible nodes are determined subsequently. Essentially, starting on one side of the hyperplane from Figure 2.12, a spot is chosen randomly. If the value for the hidden node is one, indicating larger vehicle size, one possible combination that the RBM may return for the visible nodes is four wheels and eight occupants. This combination of values was not in the unsupervised training set, illustrating that the RBM can create new data. In this case, the vehicle label could be guessed as sport utility vehicle (SUV) or van, given that the labels are already known for these data. The RBM may also return six wheels and forty-nine occupants, one of the training examples, labeled as bus.

2.3.3.1 Calculating $p(\mathbf{v}|\mathbf{h})$ For Bernoulli-Bernoulli RBMs, a complementary derivation exists to derive $p(\mathbf{v}|\mathbf{h})$ as for $p(\mathbf{h}|\mathbf{v})$. This derivation is provided in Appendix A.3.

2.4 UNSUPERVISED TRAINING OF AN RBM

This section will provide details about training RBMs using unsupervised methods, beginning with an overview for energy-based training. Subsequent sections delve into the specific terminology and define maximum likelihood, log likelihood, and gradient ascent before providing a general derivation of training. Specific equations relevant to Bernoulli-Bernoulli RBMs are described. Training challenges and methods to overcome those challenges will be detailed. This section will end by introducing contrastive divergence, one common simplification for RBM training.

2.4.1 Training overview

Since RBMs are energy based models, training involves minimizing the energy function for each training vector. Unlike NNs, which calculate intermediate model values deterministically, the RBM intermediate values can be determined stochastically [Hinton, 2010], [Fischer and Igel, 2012]. This behavior allows these generative models to produce new data points. The model weights are equivalent when moving from visible nodes to hidden nodes and back again to visible nodes. Without random behavior, calculating the $p(\mathbf{v}|\mathbf{h})$ would undo the calculation performed to obtain the $p(\mathbf{h}|\mathbf{v})$ and provide the initial values as the output.

The error function for generative models cannot be calculated in the same way as with discriminative models, because labels are not required for unsupervised training. RBM reconstructions are the visible node values that result after recalculating the visible node values from the hidden node values [Hinton, 2010]. Reconstruction error can be calculated between the data values input to the visible nodes and the reconstructions of the visible nodes [Hinton, 2010]. Its interpretation is not the same as with the error of deterministic models, since reconstructions are not required to equal the original data values, but reconstruction error is expected to decrease during training [Hinton, 2010]. However, according to [Hinton, 2010], it is not a reliable metric since the goal of RBM training is to minimize the difference between the training data and the model's reproduction of the complete PDF, based on the training data. Depending on how quickly the RBM parameters change on consecutive training iterations, the difference between the values of the data and the reconstructed visible nodes may be small [Hinton, 2010]. This does not indicate that training is successful, however, since this is not a measure to determine how well the current RBM produces an unbiased PDF that matches the training data using only the reconstruction error [Hinton, 2010]. This is unlike training with deterministic models, since decreasing model error on the training set directly corresponds to the model's increased ability to classify the training data.

During training, the energy function is minimized in the input data space at the training examples and increased elsewhere to encourage the RBM to produce values similar to but not necessarily equal to the training points [Geoffrey Hinton, 2013]. This helps the RBM to learn more than the PDF of the training data.

2.4.2 Maximum likelihood

Maximum likelihood (ML) is a method used to solve for the parameters of an assumed likelihood PDF, $p(\mathbf{v}|\boldsymbol{\theta})$, so that a set of data is described by the model as best as possible [Theodoridis and Koutroumbas, 2008]. This makes ML based methods natural approaches for training energy based models [Fischer and Igel, 2012]. Once the parameter set is optimized, the model should be able to produce the best representation of the data. Solving for these parameters is accomplished by finding the maxima for the positive log likelihood equation and minima for the negative log likelihood equation [Jebara]. These maxima or minima ensure the parameters are suitable to describe the training data PDF. The RBM parameters must be found via iterative gradient based optimization methods. Gradient ascent is used for the positive likelihood function, and gradient descent is applied with negative likelihood functions. To update the model parameters using gradient ascent, the change in parameters is added to the previous parameter estimate. The same change is subtracted from the previous parameter estimate for gradient descent. These gradient based methods, used to update model parameters, can be performed through three common methods: batch mode, stochastic (or on-line) mode, and mini-batch mode [Fischer and Igel, 2012]. A mode is selected by balancing trade-offs between estimation accuracy and calculation time. These specific methods are described later. In this section, use of gradient ascent will be assumed.

2.4.2.1 Log likelihood It was mentioned that the optimization function used in training is actually based from the log likelihood, not simply the likelihood function. The reasons for first taking the log of the likelihood function are the same as given when describing cross-entropy (Section 2.1.5.1). The likelihood function is typically complex, involving sums and ratios of exponential terms. Taking the derivative of this function with respect to each parameter from the set $\boldsymbol{\theta}$ would be cumbersome. Since the goal is maximization when using gradient ascent, a simpler function could be maximized in place of the likelihood function such as the natural log of the likelihood function.

Taking the log preserves the locations of the derivative's maxima while simultaneously changing multiplications to additions, divisions to subtractions and canceling some exponentials. Taking the derivative of a sum of linear terms is much simpler than applying the product rule. In the first step, the marginal Boltzmann PDF is substituted for the visible nodes.

$$L(\boldsymbol{\theta}|\mathbf{v}) = p(\mathbf{v};\boldsymbol{\theta}) = \sum_{\mathbf{h}} p(\mathbf{v},\mathbf{h}) = \frac{\sum_{\mathbf{h}} e^{-E(\mathbf{v},\mathbf{h})}}{\sum_{\mathbf{v}'}\sum_{\mathbf{h}'} e^{-E(\mathbf{v}',\mathbf{h}')}} \tag{2.58}$$

Apply the log to the likelihood function and use log properties to turn division into subtraction.

$$\ln\left(L(\boldsymbol{\theta}|\mathbf{v})\right) = \ln\left(\sum_{\mathbf{h}} e^{-E(\mathbf{v},\mathbf{h})}\right) - \ln\left(\sum_{\mathbf{v}'}\sum_{\mathbf{h}'} e^{-E(\mathbf{v}',\mathbf{h}')}\right) \tag{2.59}$$

Take the derivative with respect to $\boldsymbol{\theta}$.

$$\frac{\partial}{\partial\boldsymbol{\theta}}\ln\left(L(\boldsymbol{\theta}|\mathbf{v})\right) = \frac{\partial}{\partial\boldsymbol{\theta}}\left[\ln\left(\sum_{\mathbf{h}} e^{-E(\mathbf{v},\mathbf{h})}\right) - \ln\left(\sum_{\mathbf{v}'}\sum_{\mathbf{h}'} e^{-E(\mathbf{v}',\mathbf{h}')}\right)\right] \tag{2.60}$$

2.4.3 Gradient ascent

Gradient ascent is used to maximize concave cost functions such as the positive log likelihood, providing the optimal parameter set $\boldsymbol{\theta}^*$. The equation for updating the parameter set $\boldsymbol{\theta}$ at intermediate steps $t+1$ is shown [Fischer and Igel, 2012].

$$\boldsymbol{\theta}^{t+1} = \boldsymbol{\theta}^{t} + \Delta\boldsymbol{\theta}^{t} \tag{2.61}$$

The equation for the change in parameters $\Delta\boldsymbol{\theta}^t$ is provided [Fischer and Igel, 2012].

$$\Delta\boldsymbol{\theta}^{t} = \eta\left[\frac{\partial}{\partial\boldsymbol{\theta}^{t}}\left(\sum_{k=1}^{B}\ln\left[L(\boldsymbol{\theta}^{t}|\mathbf{v}_k)\right]\right) - \lambda\boldsymbol{\theta}^{t} + \nu\boldsymbol{\theta}^{t-1}\right] \tag{2.62}$$

In the equation above, the gradient $\frac{\partial}{\partial \boldsymbol{\theta}^t}\left(\sum_{k=1}^{B} \ln\left[L\left(\boldsymbol{\theta}^t|\mathbf{v}_k\right)\right]\right)$ is a vector specifying the direction and distance to take in the likelihood surface. The distance traveled in that direction is scaled by η. It is hoped that each update at time t will lead to a larger value of the function being maximized. This is not always true using mini-batch or on-line methods. Iterations over the training data are used to recalculate the direction of movement and reach the peak of the likelihood function. For iterative optimization methods, an iteration is considered to have occurred after calculating values on one subset of training examples, and an epoch occurs after calculating values on all N training examples. Multiple epochs are used so that the function value may become as large as possible while the function gradient is minimized.

In Equation 2.62, the log likelihood is summed over the batch size, $B \leq N$. The batch size is the number of examples used to calculate each cost function update. For batch gradient ascent, $B = N$, the total number of training examples. Stochastic, or on-line, gradient ascent sets $B = 1$, while mini-batch gradient ascent uses $1 < B < N$ [Hinton, 2010]. Using more information per update through a larger sum at time t is expected to provide a more accurate value for the gradient, indicating that a better vector may be discovered to reach larger values of the likelihood function. However, calculating derivatives over large sums is time consuming, and so faster training uses smaller batch sizes with less accuracy for the resulting vector at each time step.

The term $\lambda\boldsymbol{\theta}^t$ applies weight decay, which both keeps the weight values small and forces the cost function to move in the opposite direction as determined by the gradient [Fischer and Igel, 2012], [Hinton, 2010]. Keeping the weights small prevents the sigmoid function from saturating, which permits only small changes with subsequent updates. Since each update may not help the gradient to increase, moving in the opposite direction could help to prevent settling at local maxima in favor of higher maxima at future steps. While this tactic would prevent the cost function from reaching the absolute maximum of the likelihood, realistically, the absolute maximum is not expected to be found in time $t < \infty$. Also, reaching the absolute maximum of the cost function on the training set may prevent the model from appropriately clustering new examples. This technique is used to help the model overcome the error of the training set in modeling completely the desired environment. The parameter λ determines the proportion of weight decay applied.

The $\nu\boldsymbol{\theta}^{t-1}$ term is used to weight the previous parameter estimate, allowing the new estimate to contribute only smaller changes. This is known as applying momentum to the training process, and is used to prevent sudden and incorrect changes in the gradient from dominating the update direction. The amount of momentum used is determined by ν [Fischer and Igel, 2012].

2.4.4 Generic training derivation

In this section, the training equations are derived for a generic RBM [Fischer and Igel, 2012]. The derivation starts from Equation 2.60, reproduced below.

$$\frac{\partial}{\partial \boldsymbol{\theta}} \ln\left(L\left(\boldsymbol{\theta}|\mathbf{v}\right)\right) = \frac{\partial}{\partial \boldsymbol{\theta}}\left[\ln\left(\sum_{\mathbf{h}} e^{-E(\mathbf{v},\mathbf{h})}\right) - \ln\left(\sum_{\mathbf{v}'}\sum_{\mathbf{h}'} e^{-E(\mathbf{v}',\mathbf{h}')}\right)\right] \tag{2.63}$$

Derivative rules for logs, exponents, and sums are applied, and the derivative is taken with respect to the parameter set $\boldsymbol{\theta}$. The energy function must be known to complete the derivative.

$$\begin{aligned} \tfrac{\partial}{\partial \boldsymbol{\theta}} \ln\left(L(\boldsymbol{\theta}|\mathbf{v})\right) \quad = \quad & \left[\tfrac{1}{\sum_{\mathbf{h}'} e^{-E(\mathbf{v},\mathbf{h}')}} \times \textstyle\sum_{\mathbf{h}} e^{-E(\mathbf{v},\mathbf{h})} \times (-1)\tfrac{\partial E(\mathbf{v},\mathbf{h})}{\partial \boldsymbol{\theta}}\right] \\ - \quad & \left[\tfrac{1}{\sum_{\mathbf{v}'''}\sum_{\mathbf{h}'''} e^{-E(\mathbf{v}''',\mathbf{h}''')}} \times \textstyle\sum_{\mathbf{v}''}\sum_{\mathbf{h}''} e^{-E(\mathbf{v}'',\mathbf{h}'')} \times (-1)\tfrac{\partial E(\mathbf{v}'',\mathbf{h}'')}{\partial \boldsymbol{\theta}}\right] \end{aligned} \tag{2.64}$$

Terms are put into equivalent forms using joint and marginal PDFs and combined. The negatives are placed in front of each term.

$$\begin{aligned} \tfrac{\partial}{\partial \boldsymbol{\theta}} \ln\left(L(\boldsymbol{\theta}|\mathbf{v})\right) \quad = \quad & \left[-\textstyle\sum_{\mathbf{h}} \tfrac{p(\mathbf{v},\mathbf{h})}{p(\mathbf{v})} \times \tfrac{\partial E(\mathbf{v},\mathbf{h})}{\partial \boldsymbol{\theta}}\right] \\ & + \left[\textstyle\sum_{\mathbf{v}'}\sum_{\mathbf{h}'} p(\mathbf{v}',\mathbf{h}') \times \tfrac{\partial E(\mathbf{v}',\mathbf{h}')}{\partial \boldsymbol{\theta}}\right] \end{aligned} \tag{2.65}$$

The first term becomes a conditional PDF, and the last term remains as a joint PDF. The conditional PDF represents the distribution of the hidden nodes given the data vector $\mathbf{v}$. The joint PDF represents the distribution of the complete model. Taking the derivative of the energy function with respect to the model distribution provides a computational challenge in training [Fischer and Igel, 2012].

$$\begin{aligned}\tfrac{\partial}{\partial \boldsymbol{\theta}} \ln\left(L(\boldsymbol{\theta}|\mathbf{v})\right) &= \left[-\textstyle\sum_{\mathbf{h}} p(\mathbf{h}|\mathbf{v}) \times \tfrac{\partial E(\mathbf{v},\mathbf{h})}{\partial \boldsymbol{\theta}}\right] \\ &\quad + \left[\textstyle\sum_{\mathbf{v}'} \sum_{\mathbf{h}'} p(\mathbf{v}',\mathbf{h}') \times \tfrac{\partial E(\mathbf{v}',\mathbf{h}')}{\partial \boldsymbol{\theta}}\right]\end{aligned} \tag{2.66}$$

Each term matches the definition of an expected value function. The first term calculates the expected value of the energy function given the distribution of the hidden nodes determined from the visible nodes, $p(\mathbf{h}|\mathbf{v})$. The second term calculates the expected value of the energy function given the joint distribution of the model nodes, $p(\mathbf{v},\mathbf{h})$ [Fischer and Igel, 2012].

$$\frac{\partial}{\partial \boldsymbol{\theta}} \ln\left(L(\boldsymbol{\theta}|\mathbf{v})\right) = -\boldsymbol{E}_{p(\mathbf{h}|\mathbf{v})}\left\{\frac{\partial E(\mathbf{v},\mathbf{h})}{\partial \boldsymbol{\theta}}\right\} + \boldsymbol{E}_{p(\mathbf{v}',\mathbf{h}')}\left\{\frac{\partial E(\mathbf{v}',\mathbf{h}')}{\partial \boldsymbol{\theta}}\right\} \tag{2.67}$$

When calculating the terms in the above equation over a set of B examples in each batch of training examples $\mathbf{v} \in \mathfrak{D}$, additional notation is introduced [Fischer and Igel, 2012].

$$\frac{1}{B}\sum_{\mathbf{v}\in\mathfrak{D}} \frac{\partial}{\partial \boldsymbol{\theta}} \ln(L(\boldsymbol{\theta}|\mathbf{v})) = -\frac{1}{B}\sum_{\mathbf{v}\in\mathfrak{D}} \boldsymbol{E}_{p(\mathbf{h}|\mathbf{v})}\left\{\frac{\partial E(\mathbf{v},\mathbf{h})}{\partial \boldsymbol{\theta}}\right\} + \frac{1}{B}\sum_{\mathbf{v}\in\mathfrak{D}} \boldsymbol{E}_{p(\mathbf{v}',\mathbf{h}')}\left\{\frac{\partial E(\mathbf{v}',\mathbf{h}')}{\partial \boldsymbol{\theta}}\right\} \tag{2.68}$$

In Equation 2.69, the angle bracket notation ($<>$) indicates the average of the expected value, calculated with respect to the indicated PDF, over the batch of training examples $\mathbf{v} \in \mathfrak{D}$ [Fischer and Igel, 2012].

$$\left\langle \frac{\partial}{\partial \boldsymbol{\theta}} \ln(L(\boldsymbol{\theta}|\mathbf{v})) \right\rangle = -\left\langle \frac{\partial E(\mathbf{v},\mathbf{h})}{\partial \boldsymbol{\theta}} \right\rangle_{p(\mathbf{h}|\mathbf{v})} + \left\langle \frac{\partial E(\mathbf{v}',\mathbf{h}')}{\partial \boldsymbol{\theta}} \right\rangle_{p(\mathbf{v}',\mathbf{h}')} \tag{2.69}$$

The first term in this equation indicates the change in the energy function with respect to the conditional PDF of the hidden nodes. This term represents the data groupings that the model creates based on the training data batch $\mathbf{v} \in \mathfrak{D}$ that is input to the visible nodes.

The second term in Equation 2.69, derived from the partition function Z, represents the change in energy from the overall state of the model, including the visible nodes $\mathbf{v}$ and hidden nodes $\mathbf{h}$. Equation 2.69 is also shown as in Equation 2.70 [Hinton, 2010].

$$\left\langle \frac{\partial}{\partial \boldsymbol{\theta}} \ln(L(\boldsymbol{\theta}|\mathbf{v})) \right\rangle = -\left\langle \frac{\partial E(\mathbf{v},\mathbf{h})}{\partial \boldsymbol{\theta}} \right\rangle_{data} + \left\langle \frac{\partial E(\mathbf{v}',\mathbf{h}')}{\partial \boldsymbol{\theta}} \right\rangle_{model} \tag{2.70}$$

The change in the energy function due to the data pulls the likelihood function in the opposite direction as specified by the change in the energy function from the model [Geoffrey Hinton, 2013]. Training encourages the groupings found by the model to produce similar data to that on the visible nodes while suppressing the model's reproductions of the training data points [Geoffrey Hinton, 2013]. Theoretically, when the model's reconstruction of the visible nodes equals the state resulting from the training data on the visible nodes, the change in energy should be zero. Practically, the change will fall below a threshold but not reach absolute zero. Ideally, if the training data matches exactly the desired PDF of the data environment, a zero gradient indicates the model's ability to reconstruct the desired PDF perfectly.

2.4.4.1 Bernoulli-Bernoulli derivation

Derivatives of the Bernoulli energy function with respect to the parameters in the set $\boldsymbol{\theta} \in \{\mathbf{W}, \mathbf{b}, \mathbf{c}\}$ are shown. The Bernoulli energy equation in scalar form is given.

$$E(\mathbf{v},\mathbf{h}) = -\sum_{i=1}^{N} v_i b_i - \sum_{j=1}^{M} h_j c_j - \sum_{i=1}^{N}\sum_{j=1}^{M} W_{ij} v_i h_j \tag{2.71}$$

The derivative with respect to an individual weight parameter W_{ij} is taken.

$$\frac{\partial E(\mathbf{v},\mathbf{h})}{\partial W_{ij}} = \frac{\partial \left[-\sum_{i=1}^{N} v_i b_i - \sum_{j=1}^{M} h_j c_j - \sum_{i=1}^{N}\sum_{j=1}^{M} W_{ij} v_i h_j \right]}{\partial W_{ij}} = -v_i h_j \tag{2.72}$$

The derivative $\frac{\partial E(\mathbf{v},\mathbf{h})}{\partial W_{ij}}$ is plugged into $\left\langle \frac{\partial \ln(L(\boldsymbol{\theta}|\mathbf{v}))}{\partial W_{ij}} \right\rangle$.

$$\left\langle \frac{\partial \ln(L(\boldsymbol{\theta}|\mathbf{v}))}{\partial W_{ij}} \right\rangle = -\langle -v_i h_j \rangle_{data} + \langle -v_i h_j \rangle_{model} = \langle v_i h_j \rangle_{data} - \langle v_i h_j \rangle_{model} \tag{2.73}$$

The derivative with respect to an individual visible bias parameter b_i is taken.

$$\frac{\partial E(\mathbf{v},\mathbf{h})}{\partial b_i} = \frac{\partial \left[-\sum_{i=1}^{N} v_i b_i - \sum_{j=1}^{M} h_j c_j - \sum_{i=1}^{N}\sum_{j=1}^{M} W_{ij} v_i h_j\right]}{\partial b_i} = -v_i \tag{2.74}$$

The derivative $\frac{\partial E(\mathbf{v},\mathbf{h})}{\partial b_i}$ is plugged into $\left\langle \frac{\partial \ln(L(\boldsymbol{\theta}|\mathbf{v}))}{\partial b_i} \right\rangle$.

$$\left\langle \frac{\partial \ln(L(\boldsymbol{\theta}|\mathbf{v}))}{\partial b_i} \right\rangle = -\big\langle -v_i \big\rangle_{data} + \big\langle -v_i \big\rangle_{model} = \big\langle v_i \big\rangle_{data} - \big\langle v_i \big\rangle_{model} \tag{2.75}$$

The derivative with respect to an individual hidden bias parameter c_j is taken.

$$\frac{\partial E(\mathbf{v},\mathbf{h})}{\partial c_j} = \frac{\partial \left[-\sum_{i=1}^{N} v_i b_i - \sum_{j=1}^{M} h_j c_j - \sum_{i=1}^{N}\sum_{j=1}^{M} W_{ij} v_i h_j\right]}{\partial c_j} = -h_j \tag{2.76}$$

The derivative $\frac{\partial E(\mathbf{v},\mathbf{h})}{\partial c_j}$ is plugged into $\left\langle \frac{\partial \ln(L(\boldsymbol{\theta}|\mathbf{v}))}{\partial c_j} \right\rangle$.

$$\left\langle \frac{\partial \ln(L(\boldsymbol{\theta}|\mathbf{v}))}{\partial c_j} \right\rangle = -\big\langle -h_j \big\rangle_{data} + \big\langle -h_j \big\rangle_{model} = \big\langle h_j \big\rangle_{data} - \big\langle h_j \big\rangle_{model} \tag{2.77}$$

Each of Equations 2.73, 2.75, and 2.77 shows that the magnitude of the parameter update depends on the ability of the model to describe a PDF that is then trained to reproduce similar values as those calculated directly from the data. The overall goal of this optimization is to train the RBM to produce the PDF encoding the training data by minimizing the difference between the representation of the data PDF and that of the model. Ideally when the derivative remains at zero, the data PDF and model PDF are equal, and training ends.

2.4.5 Partition term

In these training equations, the second term is derived from the partition function, Z. The number of terms in the partition function for a Bernoulli-Bernoulli RBM is on the order of 2^{N+M} [Fischer and Igel, 2012]. Even though an RBM eliminates the within-layer connections, the number of total state combinations causes the partition term to be intractable [Fischer and Igel, 2012]. For example, for a Bernoulli-Bernoulli RBM with two visible nodes and one hidden node, the resulting partition function has eight terms, which agrees with the number of binary states from three nodes. The partition function must be estimated, and one common method is via Markov chain Monte Carlo (MCMC) sampling [Fischer and Igel, 2012],[Hinton, 2010]. Contrastive divergence (CD), developed by Geoffrey Hinton, approximates the MCMC process to decrease training time [Geoffrey Hinton, 2013]. Sections 2.4.6 and 2.4.7 summarize MCMC and Gibbs sampling, and CD is explained in Section 2.4.8.

2.4.6 Markov chain Monte Carlo (MCMC)

Markov chain Monte Carlo calculates samples from an approximation of a high dimensional PDF. The MCMC iterates through many states, and the Markov property simplifies calculations since the entire state history can be summarized by the output of the previous step. The chain must be initialized, and the initialization point can be chosen randomly [Geyer, 2011], [Fischer and Igel, 2012]. Using the equations of the desired PDF, calculations are repeated beginning from the random seed.

Since the PDF structure is applied to produce the output at each step, after many iterations, an approximation of the PDF variation is obtained. Stationarity, the point when further iterations will not change the PDF characteristics, is reached when the detailed balance condition is satisfied ($\pi(i)p_{ij} = \pi(j)p_{ji}$) [Fischer and Igel, 2012], [Geyer, 2011]. In the equation, $\pi(i)$ is taken to be the state of the chain at Step i, and p_{ij} is the transition probability matrix from Step i to Step j. Conversely, $\pi(j)$ is the state of the chain at Step j, and p_{ji} is the transition probability matrix from Step j to Step i. The detailed balance condition states that beginning at Step i and transitioning to Step j produces the same result as beginning at Step j and transitioning to Step i. This allows any value to be reached from any other value, ensuring that the chosen sample is unbiased.

2.4.7 Gibbs sampling

Gibbs sampling is a particular type of MCMC sampling where the conditional distributions are simulated [Geyer, 2011]. The requirements are that the complete conditional distributions, $p(\mathbf{x}_1|\mathbf{x}_2)$ and $p(\mathbf{x}_2|\mathbf{x}_1)$, can be calculated for each vector of variables $\mathbf{x}_1$ and $\mathbf{x}_2$ [Niemi, 2013]. For RBMs, $\mathbf{x}_1$ and $\mathbf{x}_2$ correspond to the vectors of visible and hidden nodes $\mathbf{v}$ and $\mathbf{h}$, respectively. With RBMs at time $j = 0$, the visible vector $\mathbf{v}$ is initialized with an example from the training dataset. For each subsequent time step, j, the following iterations are performed [Niemi, 2013], [Hinton, 2010], [Fischer and Igel, 2012].

$$\begin{aligned} &Sample\ \mathbf{h}^j \sim p(\mathbf{h}^j|\mathbf{v}^j) \\ &Sample\ \mathbf{v}^{j+1} \sim p(\mathbf{v}^{j+1}|\mathbf{h}^j) \end{aligned} \tag{2.78}$$

The property of conditional independence allows for block Gibbs sampling [Fischer and Igel, 2012]. With block sampling, the individual values of the entire vector $\mathbf{v}$ or $\mathbf{h}$ can be calculated at once. Therefore, all of the variables can be updated in two steps per iteration. Without conditional independence, $K = N+M$ calculations would be required to determine the values for all nodes per update [Fischer and Igel, 2012], [Niemi, 2013]. Below is an example of an update of the nodes without using block sampling for all $\mathbf{v} \in \{1, N\}$ and $\mathbf{h} \in \{1, M\}$. All other nodes' values are used to determine the value for a single node in each calculation.

$$\begin{aligned} v_1^j &\sim p\left(v_1^j|v_2^{j-1}, \cdots, v_N^{j-1}, h_1^{j-1}, \cdots, h_M^{j-1}\right) \\ &\cdots \\ v_N^j &\sim p\left(v_N^j|v_1^j, \cdots, v_{N-1}^j, h_1^{j-1}, \cdots, h_M^{j-1}\right) \\ h_1^j &\sim p\left(h_1^j|v_1^j, \cdots, v_N^j, h_2^{j-1}, \cdots, h_M^{j-1}\right) \\ &\cdots \\ h_M^j &\sim p\left(h_M^j|v_1^j, \cdots, v_N^j, h_1^j, \cdots, h_{M-1}^j\right) \end{aligned} \tag{2.79}$$

2.4.8 Contrastive divergence (CD)

Contrastive divergence approximates the MCMC sampling step by decreasing the number of steps needed to be run to achieve an approximation to the equilibrium joint distribution [Hinton, 2010]. This equilibrium joint distribution is the term from training that comes from the partition function as given by the second term in Equation 2.70. Instead of beginning the chain at a random point, the chain is initialized at one of the training data points. Next, the chain is run for K steps (usually in practice, $K = 1$) [Hinton, 2010]. Figure 2.13 illustrates the process.

Gibbs sampling is used to calculate the values of the hidden and visible nodes, alternately, starting with the training data on the visible nodes at $t = 0$. Due to the requirement of MCMC to run until achieving equilibrium, the ability of the reconstructed visible layer values to represent an unbiased model after K steps will depend on the model's ability to converge quickly to equilibrium [Fischer and Igel, 2012], [Hinton, 2010]. During the early stages, the reconstructed values are not expected to represent an unbiased model PDF well. These training approximations are used to represent the second term in Equations 2.73, 2.75, and 2.77.

When updating the weights, W_{ij}, the first term is equivalent to $< v_i h_j >^0$. This term uses the actual data values for the visible nodes and calculates hidden node values based on the training data. With CD, the reconstruction after K steps is given by $< v_i h_j >^K$. The expected value is implied, since the model can be approximated to have reached convergence after K steps, and the value after K steps is sampled as if from the desired joint PDF [Hinton, 2010].

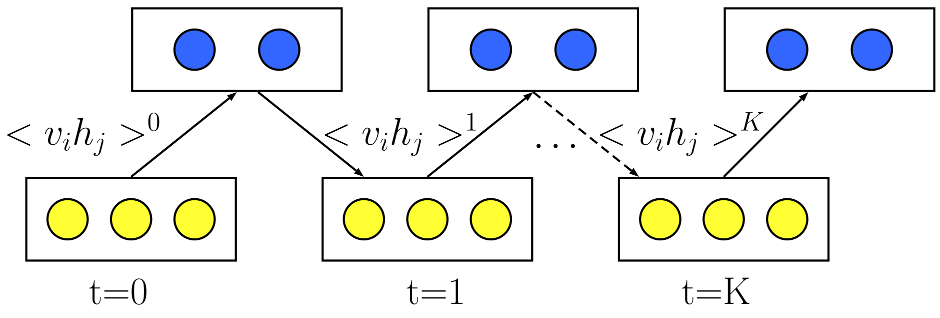


Figure 2.13: Illustration of contrastive divergence for weight updates

2.4.9 Why CD works

The goal of RBM training is to have the PDF produced by the RBM, $p(\mathbf{v})$, match that of the training data, $q(\mathbf{v})$, and this can be achieved via ML [Fischer and Igel, 2012], [Hinton, 2010]. The Kullback-Leibler (KL) divergence, which is non-negative and asymmetric, can be used to calculate the difference between two PDFs [Fischer and Igel, 2012], [Theodoridis and Koutroumbas, 2008]. This KL divergence equation is provided.

$$KL(q(\mathbf{v})||p(\mathbf{v})) = \sum_{\mathbf{v}} q(\mathbf{v}) \ln \left[\frac{q(\mathbf{v})}{p(\mathbf{v})}\right] \tag{2.80}$$

One bias that ML training attempts to minimize comes from the difference between $q(\mathbf{v})$ and $p(\mathbf{v})$. During training the model parameters, $\boldsymbol{\theta}$, are changed, and this modifies $p(\mathbf{v})$. As $p(\mathbf{v})$ produces a better representation of the training data, the natural log in the KL divergence approaches zero, which has the effect of reducing the bias [Fischer and Igel, 2012].

Applying the CD approximation to obtain a sample for the PDF of the partition term in the log likelihood equation introduces another potential source of bias, created by using a small and finite number of sampling steps, K. MCMC, when started from a randomly chosen point, produces PDF samples fairly at equilibrium, which occurs as K tends toward infinity [Fischer and Igel, 2012]. The CD training approximation can be described as the difference of two KL divergences as shown below [Fischer and Igel, 2012], [Hinton, 2010]. The PDF produced by the RBM model after K sampling steps is given by $p_K(\mathbf{v})$ [Fischer and Igel, 2012].

$$\begin{aligned} CD_K = \\ & KL(q(\mathbf{v})||p(\mathbf{v})) - KL(p_K(\mathbf{v})||p(\mathbf{v})) = \\ & \textstyle\sum_{\mathbf{v}} q(\mathbf{v}) \ln \left[\frac{q(\mathbf{v})}{p(\mathbf{v})}\right] - \sum_{\mathbf{v}} p_K(\mathbf{v}) \ln \left[\frac{p_K(\mathbf{v})}{p(\mathbf{v})}\right] \end{aligned} \tag{2.81}$$

The first term provides the bias reduced by ML training, and the second term shows the additional bias that could be incurred by the CD approximation [Fischer and Igel, 2012]. For K going to infinity, the MCMC model approaches stationarity, $p_K(\mathbf{v})$ approaches $p(\mathbf{v})$, and the bias from the CD approximation is eliminated.

$$CD_K = KL(q(\mathbf{v})||p(\mathbf{v})) - KL(p_K(\mathbf{v})||p(\mathbf{v})) = \sum_{\mathbf{v}} q(\mathbf{v}) \ln \left[\frac{q(\mathbf{v})}{p(\mathbf{v})} \right] \tag{2.82}$$

Fewer than infinite steps are required for actual CD implementation, since the MCMC chain is initialized at a training point, $\mathbf{v}$, instead of using a random value [Hinton, 2010].

3.0 GENERATIVE MODELS AS CLASSIFIERS

There are distinct goals in training generative versus discriminative models. Supervised learning of discriminative models focuses on the classification task by orienting hyperplanes for maximum class separability, and for this, discriminative models only need to learn the posterior values. Classification improvements are attributed to both using more separable features and in finding parameters that describe accurately where the classes should be separated for maximum model classification accuracy. From Chapter 2, RBMs are generative models that discover naturally occurring relationships between training data points by learning to represent the training data PDF. These relationships are reflected in the trained model's outputs and parameters, which define hyperplanes that separate unlabeled data clusters.

Unsupervised and supervised learning methods can be combined in an effort to discover relationships in the data that provide useful information for classification [Larochelle and Bengio, 2008], [Larochelle et al., 2012]. Initialization with trained generative models' parameters leads to performance improvements [Erhan et al., 2010],[Hinton, 2007],[Larochelle et al., 2009]. One type of standalone model combining supervised and unsupervised tasks is the classification RBM (cRBM), which learns the joint PDF between the data and the class labels [Larochelle and Bengio, 2008], [Larochelle et al., 2012]. RBMs without label information, as from Chapter 2 will be referred to as non-classification RBMs (nRBMs).

This chapter describes how RBMs are used for classification. Derivations of the cRBM model equations and training equations for generative RBMs (GRBMs), discriminative RBMs (DRBMs), and hybrid RBMs (HRBMs), all specific types of cRBMs, are included in this section. These cRBM models are incorporated in the proposed FFN initialization method applied in the experimental work of Ch 5.

3.1 ADAPTING RBMS FOR CLASSIFICATION

One method for applying RBMs to classification is to train one RBM per class [Hinton, 2010]. The free energy values calculated from individual RBMs can be applied in a separate softmax model for classification [Hinton, 2010]. One drawback to this method is that the features are not created specifically for classification purposes since feature generation and discrimination occur in separate training phases.

A method to directly link the classes with the data is by adapting the RBM structure to include the label information, forming the classification RBM (cRBM) [Larochelle et al., 2012], [Larochelle and Bengio, 2008], [Hinton, 2007]. Since the model learns the joint PDF between the data and the labels, the parameters resulting from cRBM training should be able to perform data generation and classification [Larochelle and Bengio, 2008], [Larochelle et al., 2012]. This method combines the training phases for model initialization and discrimination, which reduces the number of hyperparameters available to optimize model performance [Larochelle and Bengio, 2008], [Larochelle et al., 2012]. Classification RBM models may be trained using unsupervised, supervised, or hybrid training objectives [Larochelle et al., 2012],[Larochelle and Bengio, 2008]. The structural changes will be described in Section 3.1.1. Derivations illustrating the adapted equations are provided in Section 3.1.2.

3.1.1 Structural model adaptations

The original purpose of the RBM is to model the PDF of the visible nodes, $p(\mathbf{v})$. By including nodes to represent the class label information, the visible layer of the RBM is adapted to model the joint PDF of the data and classes, $p(\mathbf{v}, \mathbf{e}_y)$ [Larochelle et al., 2012], [Larochelle and Bengio, 2008]. In both Figures 3.1 and 3.2, the yellow circles represent the visible nodes, the blue circles indicate the hidden nodes, and the red circles contain the class labels. The parameter $\mathbf{W}$ indicates the weight matrix between the visible nodes and the hidden nodes. The parameters $\mathbf{U}$ provide the weight contributions of the individual hidden nodes to the classes. The bias parameters $\mathbf{b}$, $\mathbf{c}$, and $\mathbf{d}$ belong to the visible nodes, the hidden nodes, and the label nodes, respectively. Notation for the parameters is taken from both [Larochelle and Bengio, 2008] and [Larochelle et al., 2012].

Figure 3.1 illustrates the RBM model capable of performing classification [Larochelle and Bengio, 2008], [Larochelle et al., 2012]. In this model, the nodes originally designated as belonging to the visible layer are partitioned into two sets colored yellow and red. This is possible because all nodes within a layer are conditionally independent of one another. These additional label nodes change the meaning of the model as a whole, but not the meaning of the original set of visible nodes. The hidden nodes' meaning is adjusted to incorporate the information from both the visible and the label nodes.

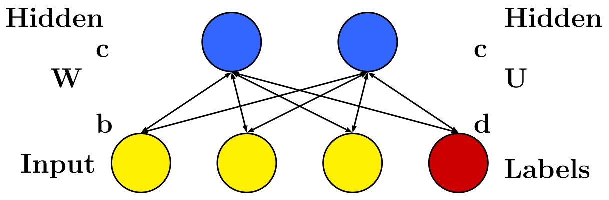


Figure 3.1: Classification RBM (cRBM)

One way to view the creation of the classification RBM is that nodes are added to the visible layer and reassigned to indicate labels. These label nodes form their own layer and are not considered to be part of the visible set of nodes. Figure 3.2 illustrates an equivalent configuration for the classification RBM where the label nodes are physically separated from the visible nodes. Just as in Figure 3.1, the relationship between the visible and the label nodes occurs only through the hidden nodes.

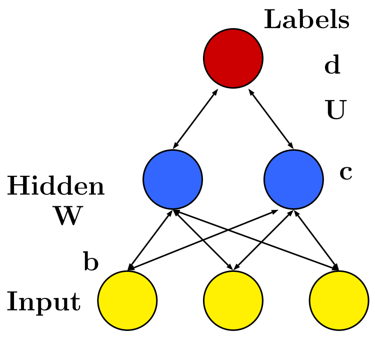


Figure 3.2: Alternate orientation for cRBM

The parameter dimensions are indicated:

$$\begin{aligned}
&\mathbf{b}, \mathbf{v} : [N \times 1] \\
&\mathbf{c}, \mathbf{h} : [M \times 1] \\
&\mathbf{W} : [N \times M] \\
&\mathbf{d}, \mathbf{e}_y : [C \times 1] \\
&\mathbf{U} : [M \times C] \\
&\mathbf{U}_k : [M \times 1]
\end{aligned} \tag{3.1}$$

The meanings for $\mathbf{b}$, $\mathbf{v}$, $\mathbf{c}$, $\mathbf{h}$, and $\mathbf{W}$ have not changed from Chapter 2. All equations and notations in this chapter are derived and adapted, respectively, from [Larochelle and Bengio, 2008], [Larochelle et al., 2012]. The vector $\mathbf{e}_y$ indicates the binary representation of a class label from one of C options. The scalar y_k is defined with the indicator function $y_k = (1_{y=k})_{k=1}^{C}$ and corresponds to the current class choice k. The notation for the vector set of label layer biases uses $\mathbf{d}$ while a specific bias for class k is noted using the scalar d_k. Similar notation is required for the weight matrix between the hidden nodes and the label nodes. The full weight matrix is indicated using $\mathbf{U}$, and the vector corresponding to class k is shown using $\mathbf{U}_k$.

3.1.2 Model equation adaptations

In this section, derivations for the equations describing the overall classification RBM are provided. The derivations begin with the energy equation, followed by the equations for all of the possible applicable conditional PDFs. As with the non-classification RBM, the PDF to determine the visible node values given the hidden nodes is unchanged, $p(\mathbf{v}|\mathbf{h})$. The classification RBM requires the conditional PDF equation for calculating the probability of the hidden nodes given the visible nodes and the label nodes, $p(\mathbf{h}|\mathbf{v}, \mathbf{e}_y)$. The classification RBM also requires equations to establish the link between the hidden nodes $\mathbf{h}$ and the label nodes $\mathbf{e}_y$, $p(y_k|\mathbf{h})$. Lastly, by accounting for all possible hidden node values that could occur, the values for the label nodes $\mathbf{e}_y$ can be determined directly from the visible nodes $\mathbf{v}$, $p(y_k|\mathbf{v})$. This is true for a number of classes that does not cause the sum over the classes in the denominator to become intractable [Larochelle and Bengio, 2008].

3.1.2.1 Classification energy equation

The energy equation for the non-classification RBM from Chapter 2 is reproduced in vector form. This applies when all visible and hidden nodes are described using the Bernoulli PDF.

$$E(\mathbf{v}, \mathbf{h}) = -\mathbf{b}^T\mathbf{v} - \mathbf{c}^T\mathbf{h} - \mathbf{v}^T\mathbf{W}\mathbf{h} \tag{3.2}$$

Energy is quantified using scalar values. With the cRBM, an indicator function is applied to the label nodes, meaning that a single class is chosen to equal one while all others are set to zero. In terms of the energy equation, this results in two additional scalar terms. One term is for the label node bias and the other provides the dot product between the hidden node vector $\mathbf{h}$ and the weights linking the hidden nodes to the indicated class, $\mathbf{U}_k$.

$$E(\mathbf{v}, \mathbf{h}, \mathbf{e}_y) = -\mathbf{b}^T\mathbf{v} - \mathbf{c}^T\mathbf{h} - \mathbf{v}^T\mathbf{W}\mathbf{h} - \mathbf{d}^T\mathbf{e}_y - \mathbf{h}^T\mathbf{U}\mathbf{e}_y \tag{3.3}$$

Using the indicator function for the labels explicitly shows the added scalar terms for the energy equation.

$$E(\mathbf{v}, \mathbf{h}, \mathbf{e}_y) = -\mathbf{b}^T\mathbf{v} - \mathbf{c}^T\mathbf{h} - \mathbf{v}^T\mathbf{W}\mathbf{h} - d_k - \mathbf{h}^T\mathbf{U}_k \tag{3.4}$$

The classification RBM energy equation can also be given in scalar form.

$$E(\mathbf{v}, \mathbf{h}, \mathbf{e}_y) = -\sum_{i=1}^{N} b_i v_i - \sum_{j=1}^{M} c_j h_j - \sum_{i=1}^{N}\sum_{j=1}^{M} v_i W_{ij} h_j - \sum_{k=1}^{C} d_k y_k - \sum_{j=1}^{M}\sum_{k=1}^{C} h_j U_{jk} y_k \tag{3.5}$$

Indicating a specific class $y = k$ will be shown by simplifying the sum to include only the scalar values associated with that class k. This occurs because y_k either equals one for class k or zero.

$$E(\mathbf{v}, \mathbf{h}, \mathbf{e}_y) = -\sum_{i=1}^{N} b_i v_i - \sum_{j=1}^{M} c_j h_j - \sum_{i=1}^{N}\sum_{j=1}^{M} v_i W_{ij} h_j - d_k - \sum_{j=1}^{M} h_j U_{jk} \tag{3.6}$$

3.1.2.2 Calculating $p(\mathbf{v}|\mathbf{h})$ In creating the classification RBM, the hidden and visible nodes' PDFs remain unchanged. Therefore, each hidden and visible node is described using a Bernoulli PDF. This means that the $p(V_i = 1|\mathbf{h})$ can be calculated as in Ch 2 using the sigmoid function. The derivation for obtaining the $p(\mathbf{v}|\mathbf{h})$ is provided in Section A.3.

3.1.2.3 Calculating $p(\mathbf{h}|\mathbf{v}, \mathbf{e}_y)$ In calculating the probability values for the set of hidden nodes, both the visible and the label nodes contribute. This is a similar concept as compared to the calculation for the PDF of the non-classification RBM hidden nodes. The difference is the addition of parameters required to distinguish between the visible and label nodes.

As with the PDF derivations in Chapter 2, the equation for $p(\mathbf{h}|\mathbf{v}, \mathbf{e}_y)$ begins with the definition of a conditional PDF.

$$p(\mathbf{h}|\mathbf{v}, \mathbf{e}_y) = \frac{p(\mathbf{h}, \mathbf{v}, \mathbf{e}_y)}{p(\mathbf{v}, \mathbf{e}_y)} = \frac{p(\mathbf{h}, \mathbf{v}, \mathbf{e}_y)}{\sum_{\mathbf{h}'} p(\mathbf{h}', \mathbf{v}, \mathbf{e}_y)} \tag{3.7}$$

The Boltzmann probability distribution is inserted into the equation.

$$p(\mathbf{h}|\mathbf{v}, \mathbf{e}_y) = \frac{\frac{1}{Z} \exp\left(-E(\mathbf{h}, \mathbf{v}, \mathbf{e}_y)\right)}{\frac{1}{Z} \sum_{\mathbf{h}'} \exp\left(-E(\mathbf{h}', \mathbf{v}, \mathbf{e}_y)\right)} \tag{3.8}$$

The partition terms are canceled from the fraction, and the vector form of the Bernoulli-Bernoulli energy equation is applied.

$$p(\mathbf{h}|\mathbf{v}, \mathbf{e}_y) = \frac{\exp\left(\mathbf{b}^T\mathbf{v} + \mathbf{c}^T\mathbf{h} + \mathbf{v}^T\mathbf{W}\mathbf{h} + \mathbf{d}^T\mathbf{e}_y + \mathbf{h}^T\mathbf{U}\mathbf{e}_y\right)}{\sum_{\mathbf{h}'} \exp\left(\mathbf{b}^T\mathbf{v} + \mathbf{c}^T\mathbf{h}' + \mathbf{v}^T\mathbf{W}\mathbf{h}' + \mathbf{d}^T\mathbf{e}_y + \mathbf{h}'^T\mathbf{U}\mathbf{e}_y\right)} \tag{3.9}$$

Terms without the hidden node vector $\mathbf{h}$ can be canceled from the numerator and the denominator.

$$p(\mathbf{h}|\mathbf{v}, \mathbf{e}_y) = \frac{\exp\left(\mathbf{c}^T\mathbf{h} + \mathbf{v}^T\mathbf{W}\mathbf{h} + \mathbf{h}^T\mathbf{U}\mathbf{e}_y\right)}{\sum_{\mathbf{h}'} \exp\left(\mathbf{c}^T\mathbf{h}' + \mathbf{v}^T\mathbf{W}\mathbf{h}' + \mathbf{h}'^T\mathbf{U}\mathbf{e}_y\right)} \tag{3.10}$$

The scalar energy equation is applied to enable further simplifications.

$$p(\mathbf{h}|\mathbf{v}, \mathbf{e}_y) = \frac{\exp\left(\sum_{j=1}^{M} c_j h_j + \sum_{i=1}^{N}\sum_{j=1}^{M} v_i W_{ij} h_j + \sum_{j=1}^{M}\sum_{k=1}^{C} h_j U_{jk} y_k\right)}{\sum_{\mathbf{h}'} \exp\left(\sum_{j=1}^{M} c_j h'_j + \sum_{i=1}^{N}\sum_{j=1}^{M} v_i W_{ij} h'_j + \sum_{j=1}^{M}\sum_{k=1}^{C} h'_j U_{jk} y_k\right)} \tag{3.11}$$

The common sum over h_j is pulled out from each of the terms.

$$p(\mathbf{h}|\mathbf{v},\mathbf{e}_y) = \frac{\exp\left(\sum_{j=1}^{M} h_j \left[c_j + \sum_{i=1}^{N} v_i W_{ij} + \sum_{k=1}^{C} U_{jk} y_k\right]\right)}{\sum_{\mathbf{h}'} \exp\left(\sum_{j=1}^{M} h'_j \left[c_j + \sum_{i=1}^{N} v_i W_{ij} + \sum_{k=1}^{C} U_{jk} y_k\right]\right)} \tag{3.12}$$

The next steps are the same as with the derivation for $p(\mathbf{h}|\mathbf{v})$ for the non-classification RBM in Section 2.3.2.1. The exponential over a sum is turned into a product of exponentials.

$$p(\mathbf{h}|\mathbf{v},\mathbf{e}_y) = \frac{\prod_{j=1}^{M} \exp\left(h_j \left[c_j + \sum_{i=1}^{N} v_i W_{ij} + \sum_{k=1}^{C} U_{jk} y_k\right]\right)}{\sum_{\mathbf{h}'} \prod_{j=1}^{M} \exp\left(h'_j \left[c_j + \sum_{i=1}^{N} v_i W_{ij} + \sum_{k=1}^{C} U_{jk} y_k\right]\right)} \tag{3.13}$$

The vector sum over $\mathbf{h}'$ is expanded, the denominator terms are separated into the individual sums, and the product of sums is taken just as in Equations 2.50 and 2.51.

$$p(\mathbf{h}|\mathbf{v},\mathbf{e}_y) = \frac{\prod_{j=1}^{M} \exp\left(h_j \left[c_j + \sum_{i=1}^{N} v_i W_{ij} + \sum_{k=1}^{C} U_{jk} y_k\right]\right)}{\prod_{j=1}^{M} \sum_{h'_j} \exp\left(h'_j \left[c_j + \sum_{i=1}^{N} v_i W_{ij} + \sum_{k=1}^{C} U_{jk} y_k\right]\right)} \tag{3.14}$$

The sum over $h'_j \in \{0,1\}$ is computed in the denominator, and the product over j is pulled from both the numerator and the denominator.

$$p(\mathbf{h}|\mathbf{v},\mathbf{e}_y) = \prod_{j=1}^{M} \frac{\exp\left(h_j \left[c_j + \sum_{i=1}^{N} v_i W_{ij} + \sum_{k=1}^{C} U_{jk} y_k\right]\right)}{1 + \exp\left(\left[c_j + \sum_{i=1}^{N} v_i W_{ij} + \sum_{k=1}^{C} U_{jk} y_k\right]\right)} \tag{3.15}$$

Since the denominator sums over the individual variable of interest h_j, a simplification can be made to show that the $p(\mathbf{h}|\mathbf{v},\mathbf{e}_y)$ is a product of the individual probability terms $p(h_j|\mathbf{v},\mathbf{e}_y)$.

$$p(\mathbf{h}|\mathbf{v},\mathbf{e}_y) = \prod_{j=1}^{M} p(h_j|\mathbf{v},\mathbf{e}_y) \tag{3.16}$$

The same simplifications as in Chapter 2 are applied to derive the sigmoid function when calculating $p(H_j = 1|\mathbf{v},\mathbf{e}_y)$.

$$p(H_j = 1|\mathbf{v},\mathbf{e}_y) = \sigma\left[c_j + \sum_{i=1}^{N} v_i W_{ij} + \sum_{k=1}^{C} U_{jk} y_k\right] \tag{3.17}$$

The equation for $p(H_j = 0|\mathbf{v}, \mathbf{e}_y)$ is provided.

$$p(H_j = 0|\mathbf{v}, \mathbf{e}_y) = \frac{1}{1 + \exp\left(c_j + \sum_{i=1}^{N} v_i W_{ij} + \sum_{k=1}^{C} U_{jk} y_k\right)} \tag{3.18}$$

3.1.2.4 Calculating $p(y_k|\mathbf{h})$

With the classification RBM, more than two classes are possible. As mentioned in Chapter 2, the generalization of the sigmoid to multiple classes results in the softmax. This is shown in calculating $p(\mathbf{e}_y|\mathbf{h})$.

The definition for conditional PDFs is applied, as usual.

$$p(\mathbf{e}_y|\mathbf{h}) = \frac{p(\mathbf{e}_y, \mathbf{h})}{p(\mathbf{h})} = \frac{\sum_{\mathbf{v}} p(\mathbf{e}_y, \mathbf{v}, \mathbf{h})}{\sum_{\mathbf{v}} \sum_{\mathbf{e}'_y} p(\mathbf{e}'_y, \mathbf{v}, \mathbf{h})} \tag{3.19}$$

The Boltzmann distribution is applied, and the partition function is immediately canceled from the numerator and the denominator. The vector form of the energy equation is applied.

$$p(\mathbf{e}_y|\mathbf{h}) = \frac{\sum_{\mathbf{v}} \exp\left(\mathbf{b}^T\mathbf{v} + \mathbf{c}^T\mathbf{h} + \mathbf{v}^T\mathbf{W}\mathbf{h} + \mathbf{d}^T\mathbf{e}_y + \mathbf{h}^T\mathbf{U}\mathbf{e}_y\right)}{\sum_{\mathbf{v}} \sum_{\mathbf{e}'_y} \exp\left(\mathbf{b}^T\mathbf{v} + \mathbf{c}^T\mathbf{h} + \mathbf{v}^T\mathbf{W}\mathbf{h} + \mathbf{d}^T\mathbf{e}'_y + \mathbf{h}^T\mathbf{U}\mathbf{e}'_y\right)} \tag{3.20}$$

The $\exp\left(\mathbf{c}^T\mathbf{h}\right)$ terms can be canceled out of the fraction. The terms with $\mathbf{v}$ can be pulled out in both the numerator and denominator.

$$p(\mathbf{e}_y|\mathbf{h}) = \frac{\left[\sum_{\mathbf{v}} \exp\left(\mathbf{b}^T\mathbf{v} + \mathbf{v}^T\mathbf{W}\mathbf{h}\right)\right]\left[\exp\left(\mathbf{d}^T\mathbf{e}_y + \mathbf{h}^T\mathbf{U}\mathbf{e}_y\right)\right]}{\left[\sum_{\mathbf{v}} \exp\left(\mathbf{b}^T\mathbf{v} + \mathbf{v}^T\mathbf{W}\mathbf{h}\right)\right]\left[\sum_{\mathbf{e}'_y} \exp\left(\mathbf{d}^T\mathbf{e}'_y + \mathbf{h}^T\mathbf{U}\mathbf{e}'_y\right)\right]} \tag{3.21}$$

These terms with $\mathbf{v}$ are common factors and are canceled.

$$p(\mathbf{e}_y|\mathbf{h}) = \frac{\exp\left(\mathbf{d}^T\mathbf{e}_y + \mathbf{h}^T\mathbf{U}\mathbf{e}_y\right)}{\sum_{\mathbf{e}'_y} \exp\left(\mathbf{d}^T\mathbf{e}'_y + \mathbf{h}^T\mathbf{U}\mathbf{e}'_y\right)} \tag{3.22}$$

The scalar versions for the remaining terms are applied along with the indicator function notation.

$$p(y_k|\mathbf{h}) = \frac{\exp\left(d_k + \sum_{j=1}^{M} h_j U_{jk}\right)}{\sum_{l=1}^{C} \exp\left(d_l + \sum_{j=1}^{M} h_j U_{jl}\right)} \tag{3.23}$$

This results in the softmax equation with an argument of $d_k + \sum_{j=1}^{M} h_j U_{jk}$, since this expression is used in determining the value for y_k.

3.1.2.5 Calculating $p(y_k|\mathbf{v})$ With this classification RBM, the $p(y_k|\mathbf{v})$ can be calculated without requiring the values of the hidden layer nodes. The starting conditional PDF equation is given. The notation for y_k and $\mathbf{e}_y$ is used interchangeably since ideally only one y_k value is non-zero. On the right side of the equation, the $\mathbf{e}_y$ is applied with the rest of the vector notation.

$$p(y_k|\mathbf{v}) = \frac{p(\mathbf{e}_y, \mathbf{v})}{p(\mathbf{v})} = \frac{\sum_{\mathbf{h}} p(\mathbf{e}_y, \mathbf{v}, \mathbf{h})}{\sum_{\mathbf{e}'_y} \sum_{\mathbf{h}} p(\mathbf{e}'_y, \mathbf{v}, \mathbf{h})} \tag{3.24}$$

The Boltzmann distribution is substituted into the equation, and the partition function is canceled out from the terms. In addition, the common $\exp\left(\mathbf{b}^T\mathbf{v}\right)$ terms are also canceled.

$$p(y_k|\mathbf{v}) = \frac{\sum_{\mathbf{h}} \exp\left(\mathbf{c}^T\mathbf{h} + \mathbf{v}^T\mathbf{W}\mathbf{h} + \mathbf{d}^T\mathbf{e}_y + \mathbf{h}^T\mathbf{U}\mathbf{e}_y\right)}{\sum_{\mathbf{e}'_y} \sum_{\mathbf{h}} \exp\left(\mathbf{c}^T\mathbf{h} + \mathbf{v}^T\mathbf{W}\mathbf{h} + \mathbf{d}^T\mathbf{e}'_y + \mathbf{h}^T\mathbf{U}\mathbf{e}'_y\right)} \tag{3.25}$$

Next, the scalar version of the energy equation is applied with the indicator notation for further simplifications.

$$p(y_k|\mathbf{v}) = \frac{\sum_{\mathbf{h}} \exp\left(\sum_{j=1}^{M} c_j h_j + \sum_{i=1}^{N}\sum_{j=1}^{M} v_i W_{ij} h_j + d_k + \sum_{j=1}^{M} h_j U_{jk}\right)}{\sum_{l=1}^{C}\sum_{\mathbf{h}} \exp\left(\sum_{j=1}^{M} c_j h_j + \sum_{i=1}^{N}\sum_{j=1}^{M} v_i W_{ij} h_j + d_l + \sum_{j=1}^{M} h_j U_{jl}\right)} \tag{3.26}$$

The terms with the label bias d_k are pulled away from the other terms, and the sum over the common factor h_j is pulled from the remaining terms.

$$p(y_k|\mathbf{v}) = \frac{\exp\left(d_k\right) \sum_{\mathbf{h}} \exp\left(\sum_{j=1}^{M} h_j [c_j + \sum_{i=1}^{N} v_i W_{ij} + U_{jk}]\right)}{\sum_{l=1}^{C} \exp\left(d_l\right) \sum_{\mathbf{h}} \exp\left(\sum_{j=1}^{M} h_j [c_j + \sum_{i=1}^{N} v_i W_{ij} + U_{jl}]\right)} \tag{3.27}$$

In both the numerator and the denominator, the terms involving the vector sum over the hidden nodes $\sum_{\mathbf{h}}$ can be simplified as in Equations 3.13 through 3.15. The sum over j in the exponential first becomes a product of exponentials over j. Then the individual products in the sum over $\mathbf{h}$ are transformed into the product of sums. Lastly, the h_j variable is summed over its set of values $\{0, 1\}$.

$$p(y_k|\mathbf{v}) = \frac{\exp\left(d_k\right) \prod_{j=1}^{M} \left[1 + \exp\left(c_j + \sum_{i=1}^{N} v_i W_{ij} + U_{jk}\right)\right]}{\sum_{l=1}^{C} \exp\left(d_l\right) \prod_{j=1}^{M} \left[1 + \exp\left(c_j + \sum_{i=1}^{N} v_i W_{ij} + U_{jl}\right)\right]} \tag{3.28}$$

3.1.3 Training RBMs as classifiers

The choice of objective function applied when training the classification RBM model determines its capabilities. In this section, the equations required for training classification RBMs as generative, discriminative, and hybrid models are derived in Sections 3.1.3.1, 3.1.3.2, and 3.1.3.3. For hybrid models, the contribution of the generative and discriminative objective functions is controlled by the α parameter. A fully discriminative cRBM occurs when $\alpha = 0.0$, and a fully generative cRBM occurs when $\alpha = 1.0$.

3.1.3.1 Generative classification RBMs (GRBMs) Just as with training the RBM in Chapter 2, the goal in training the classification RBM as a generative model is to maximize the log likelihood of a PDF. In this case, the PDF is the joint representation of the data and the labels $p(\mathbf{v}, \mathbf{e}_y)$. The derivation is similar to that given in Chapter 2.4.4. The CD_K approximation is applied to all of the derivative updates.

The initial equation uses the natural log of the probability function. The z subscript indicates an individual training example. The remaining derivations imply the derivation for one example, and the subscript is not used.

$$\mathcal{L}_{gen}(\mathcal{D}_{train}) = \sum_{z=1}^{|\mathcal{D}_{train}|} \ln p(\mathbf{v}_z, \mathbf{e}_{yz}) = \sum_{z=1}^{|\mathcal{D}_{train}|} \ln p(\mathbf{v}_z, y_{kz}) \tag{3.29}$$

The probability $p(\mathbf{v}, \mathbf{e}_y)$ is expanded to include the sum over the hidden node vector $\mathbf{h}$.

$$\ln p(\mathbf{v}, \mathbf{e}_y) = \ln \sum_{\mathbf{h}} p(\mathbf{v}, \mathbf{h}, \mathbf{e}_y) \tag{3.30}$$

The Boltzmann distribution is plugged into the equation, with the partition function defined as: $Z = \sum_{\mathbf{v}} \sum_{\mathbf{h}} \sum_{\mathbf{e}_y} \exp\left(-E(\mathbf{v}, \mathbf{h}, \mathbf{e}_y)\right)$. By properties of logarithms, the natural log fraction is split into terms.

$$\ln p(\mathbf{v}, \mathbf{e}_y) = \ln \left[\sum_{\mathbf{h}} e^{-E(\mathbf{v},\mathbf{h},\mathbf{e}_y)} \right] - \ln \left[\sum_{\mathbf{v}'} \sum_{\mathbf{h}'} \sum_{\mathbf{e}'_y} e^{-E(\mathbf{v}',\mathbf{h}',\mathbf{e}'_y)} \right] \tag{3.31}$$

The derivative with respect to the parameter set $\boldsymbol{\theta} \in \{\mathbf{b}, \mathbf{c}, \mathbf{d}, \mathbf{W}, \mathbf{U}\}$ is taken.

$$\begin{aligned} \frac{\partial}{\partial \boldsymbol{\theta}} \ln p(\mathbf{v}, \mathbf{e}_y) &= \left[\frac{\sum_{\mathbf{h}} e^{-E(\mathbf{v},\mathbf{h},\mathbf{e}_y)}}{\sum_{\mathbf{h}'} e^{-E(\mathbf{v},\mathbf{h}',\mathbf{e}_y)}} \times (-1) \frac{\partial E(\mathbf{v},\mathbf{h},\mathbf{e}_y)}{\partial \boldsymbol{\theta}} \right] \\ &- \left[\frac{\sum_{\mathbf{v}''} \sum_{\mathbf{h}''} \sum_{\mathbf{e}''_y} e^{-E(\mathbf{v}'',\mathbf{h}'',\mathbf{e}''_y)}}{\sum_{\mathbf{v}'''} \sum_{\mathbf{h}'''} \sum_{\mathbf{e}'''_y} e^{-E(\mathbf{v}''',\mathbf{h}''',\mathbf{e}'''_y)}} \times (-1) \frac{\partial E(\mathbf{v}'',\mathbf{h}'',\mathbf{e}''_y)}{\partial \boldsymbol{\theta}} \right] \end{aligned} \tag{3.32}$$

Simplifications similar to those in Section 2.4.4 are applied, producing the expected values over the conditional and joint PDF for the two terms, respectively.

$$\begin{aligned} \frac{\partial}{\partial \boldsymbol{\theta}} \ln p(\mathbf{v}, \mathbf{e}_y) &= -\left[\sum_{\mathbf{h}} p(\mathbf{h}|\mathbf{v}, \mathbf{e}_y) \times \frac{\partial E(\mathbf{v},\mathbf{h},\mathbf{e}_y)}{\partial \boldsymbol{\theta}} \right] \\ &+ \left[\sum_{\mathbf{v}'} \sum_{\mathbf{h}'} \sum_{\mathbf{e}'_y} p(\mathbf{v}', \mathbf{h}', \mathbf{e}'_y) \times \frac{\partial E(\mathbf{v}',\mathbf{h}',\mathbf{e}'_y)}{\partial \boldsymbol{\theta}} \right] \end{aligned} \tag{3.33}$$

The first term is the expectation of the energy derivative with respect to the conditional probability $p(\mathbf{h}|\mathbf{v}, \mathbf{e}_y)$, and the second term takes the expectation of the energy derivative with respect to the full joint PDF of the model.

$$\frac{\partial}{\partial \boldsymbol{\theta}} \ln p(\mathbf{v}, \mathbf{e}_y) = -\mathbb{E}_{p(\mathbf{h}|\mathbf{v},\mathbf{e}_y)} \left\{ \frac{\partial E(\mathbf{v}, \mathbf{h}, \mathbf{e}_y)}{\partial \boldsymbol{\theta}} \right\} + \mathbb{E}_{p(\mathbf{v}',\mathbf{h}',\mathbf{e}'_y)} \left\{ \frac{\partial E(\mathbf{v}', \mathbf{h}', \mathbf{e}'_y)}{\partial \boldsymbol{\theta}} \right\} \tag{3.34}$$

Derivatives of the scalar Bernoulli-Bernoulli energy function with respect to $\boldsymbol{\theta}$ are shown. First is the derivative with respect to a visible node bias b_i.

$$\frac{\partial E(\mathbf{v},\mathbf{h},\mathbf{e}_y)}{\partial b_i} = \frac{-\sum_{i=1}^{N} b_i v_i - \sum_{j=1}^{M} c_j h_j - \sum_{i=1}^{N}\sum_{j=1}^{M} v_i W_{ij} h_j - d_k - \sum_{j=1}^{M} h_j U_{jk}}{\partial b_i} \tag{3.35}$$

$$\frac{\partial E(\mathbf{v},\mathbf{h},\mathbf{e}_y)}{\partial b_i} = -v_i \tag{3.36}$$

The derivative with respect to the individual hidden node biases c_j are shown.

$$\frac{\partial E(\mathbf{v},\mathbf{h},\mathbf{e}_y)}{\partial c_j} = \frac{-\sum_{i=1}^{N} b_i v_i - \sum_{j=1}^{M} c_j h_j - \sum_{i=1}^{N}\sum_{j=1}^{M} v_i W_{ij} h_j - d_k - \sum_{j=1}^{M} h_j U_{jk}}{\partial c_j} \tag{3.37}$$

$$\frac{\partial E(\mathbf{v},\mathbf{h},\mathbf{e}_y)}{\partial c_j} = -h_j \tag{3.38}$$

The derivative with respect to the label bias d_k is taken.

$$\frac{\partial E(\mathbf{v},\mathbf{h},\mathbf{e}_y)}{\partial d_k} = \frac{-\sum_{i=1}^{N} b_i v_i - \sum_{j=1}^{M} c_j h_j - \sum_{i=1}^{N}\sum_{j=1}^{M} v_i W_{ij} h_j - d_k - \sum_{j=1}^{M} h_j U_{jk}}{\partial d_k} \tag{3.39}$$

When Class k is chosen, $y_k = 1$:

$$\frac{\partial E(\mathbf{v},\mathbf{h},\mathbf{e}_y)}{\partial d_k} = -1 \tag{3.40}$$

When Class k is not chosen, $y_k = 0$:

$$\frac{\partial E(\mathbf{v},\mathbf{h},\mathbf{e}_y)}{\partial d_k} = 0 \tag{3.41}$$

For W_{ij}, the derivative is as shown.

$$\frac{\partial E(\mathbf{v},\mathbf{h},\mathbf{e}_y)}{\partial W_{ij}} = \frac{-\sum_{i=1}^{N} b_i v_i - \sum_{j=1}^{M} c_j h_j - \sum_{i=1}^{N}\sum_{j=1}^{M} v_i W_{ij} h_j - d_k - \sum_{j=1}^{M} h_j U_{jk}}{\partial W_{ij}} \tag{3.42}$$

$$\frac{\partial E(\mathbf{v},\mathbf{h},\mathbf{e}_y)}{\partial W_{ij}} = -v_i h_j \tag{3.43}$$

The derivative with respect to the weight connecting h_j to y_k, U_{jk}, is taken.

$$\frac{\partial E(\mathbf{v},\mathbf{h},\mathbf{e}_y)}{\partial U_{jk}} = \frac{-\sum_{i=1}^{N} b_i v_i - \sum_{j=1}^{M} c_j h_j - \sum_{i=1}^{N}\sum_{j=1}^{M} v_i W_{ij} h_j - d_k - \sum_{j=1}^{M} h_j U_{jk}}{\partial U_{jk}} \tag{3.44}$$

When Class k is chosen, $y_k = 1$:

$$\frac{\partial E(\mathbf{v},\mathbf{h},\mathbf{e}_y)}{\partial U_{jk}} = -h_j \tag{3.45}$$

When Class k is not chosen, $y_k = 0$:

$$\frac{\partial E(\mathbf{v},\mathbf{h},\mathbf{e}_y)}{\partial U_{jk}} = 0 \tag{3.46}$$

This generative training equation for the classification RBM has similar issues with tractability as with the non-classification RBM. Again, sampling methods such as MCMC and CD_K are employed to estimate the derivatives used in the gradient ascent updates.

3.1.3.2 Discriminative classification RBMs (DRBMs) As in [Larochelle et al., 2012], the joint distribution $\ln p(y_k, \mathbf{v})$ can be split into a sum of terms, $\ln p(y_k|\mathbf{v}) + \ln p(\mathbf{v})$. For discriminative RBM training, only the first term is used. The CD_K approximation is not used for discriminative training since the cost function $\ln p(y_k|\mathbf{v})$ can be calculated directly from the data $\mathbf{v}$. Again, the purpose of the CD_K approximation is to generate unbiased samples from the joint PDF of the complete model.

Again, the z subscript indicates an individual training example in the first equation. The remaining derivations assume one training example, and the subscript z is not used.

$$\mathcal{L}_{dis}(\mathcal{D}_{train}) = \sum_{z=1}^{|\mathcal{D}_{train}|} \ln p(\mathbf{e}_{yz}|\mathbf{v}_z) = \sum_{z=1}^{|\mathcal{D}_{train}|} \ln p(y_{kz}|\mathbf{v}_z) \tag{3.47}$$

Just as with the generative training, for the discriminative training the natural log of the posterior, rewritten in Equation 3.48, is taken to simplify the overall expression.

$$p(y_k|\mathbf{v}) = \frac{\exp(d_k)\prod_{j=1}^{M}\left(1+\exp\left(\sum_{i=1}^{N} W_{ij}v_i + c_j + U_{jk}\right)\right)}{\sum_{l=1}^{C}\exp(d_l)\prod_{j=1}^{M}\left(1+\exp\left(\sum_{i=1}^{N} W_{ij}v_i + c_j + U_{jl}\right)\right)} \tag{3.48}$$

In preparation for taking the partial derivative with respect to the parameters in the set $\boldsymbol{\theta}$, the indices are changed from i to q and from j to r. The variables i and j will have sums in the derivative, and those sums are separate from the specific parameter with respect to which the derivative is taken.

$$\ln[p(y_k|\mathbf{v})] = \ln\left[\frac{\exp(d_k)\prod_{r=1}^{M}\left(1+\exp\left(\sum_{q=1}^{N} W_{qr}v_q + c_r + U_{rk}\right)\right)}{\sum_{l=1}^{C}\exp(d_l)\prod_{r=1}^{M}\left(1+\exp\left(\sum_{q=1}^{N} W_{qr}v_q + c_r + U_{rl}\right)\right)}\right] \tag{3.49}$$

Due to the complexity of this derivation, the remainder of the equations will be separated by solving for the gradient updates of the first term, followed by the second term. The first term will be set equal to A and the second term equals B.

The first term is the numerator of the natural log.

$$A = \ln\exp(d_k)\prod_{r=1}^{M}\left(1+\exp\left(\sum_{q=1}^{N} W_{qr}v_q + c_r + U_{rk}\right)\right) \tag{3.50}$$

The second term is the denominator of the natural log.

$$B = -\ln\sum_{l=1}^{C}\exp(d_l)\prod_{r=1}^{M}\left(1+\exp\left(\sum_{q=1}^{N} W_{qr}v_q + c_r + U_{rl}\right)\right) \tag{3.51}$$

Equation 3.52 defines more notation used for simplification.

$$o(\mathbf{v}) = c_j + U_{jk} + \sum_{i=1}^{N} W_{ij}v_i \tag{3.52}$$

This section details the steps to obtain $\frac{\partial A}{\partial \boldsymbol{\theta}}$.

$$\frac{\partial A}{\partial \boldsymbol{\theta}} = \frac{\partial}{\partial \boldsymbol{\theta}}\ln\exp(d_k)\prod_{r=1}^{M}\left(1+\exp\left(\sum_{q=1}^{N} W_{qr}v_q + c_r + U_{rk}\right)\right) \tag{3.53}$$

Properties of natural logs are used to cancel exponentials and turn products into sums.

$$\frac{\partial A}{\partial \boldsymbol{\theta}} = \frac{\partial}{\partial \boldsymbol{\theta}} \left[d_k + \sum_{r=1}^{M} \ln \left(1 + \exp \left(\sum_{q=1}^{N} W_{qr} v_q + c_r + U_{rk} \right) \right) \right] \tag{3.54}$$

For $\boldsymbol{\theta} = d_k$:

$$\frac{\partial A}{\partial d_k} = 1 + 0 = 1 \tag{3.55}$$

For $\boldsymbol{\theta} = \{W_{ij}, U_{jk}, b_i, c_j\}$:

$$\frac{\partial A}{\partial \boldsymbol{\theta}} = 0 + \sum_{r=1}^{M} \frac{\exp\left(\sum_{q=1}^{N} W_{qr} v_q + c_r + U_{rk}\right)}{1 + \exp\left(\sum_{q=1}^{N} W_{qr} v_q + c_r + U_{rk}\right)} \frac{\partial o(\mathbf{v})}{\partial \boldsymbol{\theta}} \tag{3.56}$$

$$\frac{\partial A}{\partial \boldsymbol{\theta}} = \sum_{r=1}^{M} \sigma \Big(\sum_{q=1}^{N} W_{qr} v_q + c_r + U_{rk} \Big) \frac{\partial o(\mathbf{v})}{\partial \boldsymbol{\theta}} \tag{3.57}$$

This section details the steps to obtain $\frac{\partial B}{\partial \boldsymbol{\theta}}$.

$$\frac{\partial B}{\partial \boldsymbol{\theta}} = \frac{\partial}{\partial \boldsymbol{\theta}} \left[-\ln \sum_{l'=1}^{C} \exp(d_{l'}) \prod_{r=1}^{M} \left(1 + \exp \left(\sum_{q=1}^{N} W_{qr} v_q + c_r + U_{rl'} \right) \right) \right] \tag{3.58}$$

Equations 3.59 to 3.61 give a simplification leading to the final solution. The simplification involves introducing an identity using exponents and natural logs.

$$\begin{gathered} \textstyle\sum_{l'=1}^{C} \exp(d_{l'}) \prod_{r=1}^{M} \left(1 + \exp\left(\sum_{q=1}^{N} W_{qr} v_q + c_r + U_{rl'}\right)\right) = \\ \textstyle\sum_{l'=1}^{C} \exp\left(\ln\left(e^{d_{l'}} \prod_{r=1}^{M} \left(1 + \exp\left(\sum_{q=1}^{N} W_{qr} v_q + c_r + U_{rl'}\right)\right)\right)\right) \end{gathered} \tag{3.59}$$

Properties of natural logs are used to turn the multiplication between $\exp(d'_l)$ and the product over r into a sum.

$$= \sum_{l'=1}^{C} \exp \left(\ln(\exp(d_{l'})) + \ln \left(\prod_{r=1}^{M} \left(1 + \exp \left(\sum_{q=1}^{N} W_{qr} v_q + c_r + U_{rl'} \right) \right) \right) \right) \tag{3.60}$$

Natural log properties are used again to cancel an exponential and also to turn the log of a product into the sum of a log.

$$= \sum_{l'=1}^{C} \exp\left(d_{l'} + \sum_{r=1}^{M} \ln\left(1 + \exp\left(\sum_{q=1}^{N} W_{qr} v_q + c_r + U_{rl'}\right)\right)\right) \tag{3.61}$$

The simplification from Equation 3.61 will be applied to Equation 3.58 to produce Equation 3.62.

$$\frac{\partial B}{\partial \boldsymbol{\theta}} = \frac{\partial}{\partial \boldsymbol{\theta}} \left[- \ln \sum_{l'=1}^{C} \exp\left(d_{l'} + \sum_{r=1}^{M} \ln\left(1 + \exp\left(\sum_{q=1}^{N} W_{qr} v_q + c_r + U_{rl'}\right)\right)\right)\right] \tag{3.62}$$

For $\boldsymbol{\theta} = d_k$:

$$\frac{\partial B}{\partial d_k} = - \frac{\exp\left(d_k + \sum_{r=1}^{M} \ln\left[1 + \exp\left(c_r + U_{rk} + \sum_{q=1}^{N} W_{qr} v_q\right)\right]\right)}{\sum_{l'=1}^{C} \exp\left(d_{l'} + \sum_{r=1}^{M} \ln\left[1 + \exp\left(c_r + U_{rl'} + \sum_{q=1}^{N} W_{qr} v_q\right)\right]\right)} [1 + 0] \tag{3.63}$$

This is simplified to $p(y_k|\mathbf{v})$.

$$\frac{\partial B}{\partial d_k} = -p(y_k|\mathbf{v}) \tag{3.64}$$

For $\boldsymbol{\theta} \neq d_y$:

$$\frac{\partial B}{\partial \boldsymbol{\theta}} = \frac{\partial}{\partial \boldsymbol{\theta}} \left[- \ln \sum_{l'=1}^{C} \exp\left(d_{l'} + \sum_{r=1}^{M} \ln\left(1 + \exp\left(\sum_{q=1}^{N} W_{qr} v_q + c_r + U_{rl'}\right)\right)\right)\right] \tag{3.65}$$

Derivatives are taken with respect to $\boldsymbol{\theta}$.

$$\begin{aligned} \frac{\partial B}{\partial \boldsymbol{\theta}} &= - \left[\frac{\sum_{l=1}^{C} \exp\left(d_l + \sum_{r=1}^{M} \ln\left(1 + \exp\left(\sum_{q=1}^{N} W_{qr} v_q + c_r + U_{rl}\right)\right)\right)}{\sum_{l'=1}^{C} \exp\left(d_{l'} + \sum_{r=1}^{M} \ln\left(1 + \exp\left(\sum_{q=1}^{N} W_{qr} v_q + c_r + U_{rl'}\right)\right)\right)} \right] \\ &\times \left[0 + \sum_{r=1}^{M} \frac{0 + \exp\left(\sum_{q=1}^{N} W_{qr} v_q + c_r + U_{rl}\right)}{1 + \exp\left(\sum_{q=1}^{N} W_{qr} v_q + c_r + U_{rl}\right)} \right] \left[\frac{\partial o(\mathbf{v})}{\partial \boldsymbol{\theta}} \right] \end{aligned} \tag{3.66}$$

The definition of $p(y_k|\mathbf{v})$ is applied.

$$\frac{\partial B}{\partial \boldsymbol{\theta}} = -\left[\sum_{l=1}^{C} p(y_l|\mathbf{v}) \sum_{r=1}^{M} \frac{\exp\left(\sum_{q=1}^{N} W_{qr} v_q + c_r + U_{rl}\right)}{1 + \exp\left(\sum_{q=1}^{N} W_{qr} v_q + c_r + U_{rl}\right)}\right] \left[\frac{\partial o(\mathbf{v})}{\partial \boldsymbol{\theta}}\right] \tag{3.67}$$

The definition of the sigmoid function is used.

$$\frac{\partial B}{\partial \boldsymbol{\theta}} = -\sum_{l=1}^{C} \sum_{r=1}^{M} \sigma\left(\sum_{q=1}^{N} W_{qr} v_q + c_r + U_{rl}\right) p(y_l|\mathbf{v}) \frac{\partial o(\mathbf{v})}{\partial \boldsymbol{\theta}} \tag{3.68}$$

In Equation 3.69, the derivatives from the first term A and the second term B are combined. For $\boldsymbol{\theta} = d_k$ and C is the total number of classes:

$$\frac{\partial \ln[p(y_k|\mathbf{v})]}{\partial \boldsymbol{\theta}} = 1 - p(y_k|\mathbf{v}) \ \forall y \in \{1, \cdots, C\} \tag{3.69}$$

For $\boldsymbol{\theta} = \{W_{ij}, U_{jy}, b_i, c_j\}$:

$$\begin{aligned} \frac{\partial \ln[p(y_k|\mathbf{v})]}{\partial \boldsymbol{\theta}} &= \sum_{r=1}^{M} \sigma\left(\sum_{q=1}^{N} W_{qr} v_q + c_r + U_{ry}\right) \left[\frac{\partial o(\mathbf{v})}{\partial \boldsymbol{\theta}}\right] \\ &- \sum_{l=1}^{C} \sum_{r=1}^{M} \sigma\left(\sum_{q=1}^{N} W_{qr} v_q + c_r + U_{rl}\right) p(l|\mathbf{v}) \left[\frac{\partial o(\mathbf{v})}{\partial \boldsymbol{\theta}}\right] \end{aligned} \tag{3.70}$$

Taking the derivative of $o(\mathbf{v})$ with respect to the parameter set $\boldsymbol{\theta}$.

$$\begin{aligned} & o(\mathbf{v}) = c_j + U_{jk} + \sum_{i=1}^{N} W_{ij} v_i \\ & \frac{\partial o(\mathbf{v})}{\partial W_{ij}} = v_i \\ & \frac{\partial o(\mathbf{v})}{\partial U_{jk}} = 1 \\ & \frac{\partial o(\mathbf{v})}{\partial c_j} = 1 \\ & \frac{\partial o(\mathbf{v})}{\partial b_i} = 0 \\ & \frac{\partial o(\mathbf{v})}{\partial d_k} = 0 \end{aligned} \tag{3.71}$$

3.1.3.3 Hybrid classification RBMs (HRBMs) [Larochelle et al., 2012] states that the partial derivatives, taken with respect to each of the parameters in $\boldsymbol{\theta}$, are combined with a hyperparameter, $\alpha \in [0, 1]$, for hybrid training. The gradients for generative and for discriminative training are computed separately, then combined before performing the weight update $\Delta\boldsymbol{\theta^t}$. The z subscript indicates individual training examples.

$$\mathcal{L}_{hyb}(\mathcal{D}_{train}) = (1 - \alpha)\mathcal{L}_{dis}(\mathcal{D}_{train}) + \alpha\mathcal{L}_{gen}(\mathcal{D}_{train}) \tag{3.72}$$

$$\mathcal{L}_{hyb}(\mathcal{D}_{train}) = (1 - \alpha) \sum_{z=1}^{|\mathcal{D}_{train}|} \ln p(y_{kz}|\mathbf{v}_z) + \alpha \sum_{z=1}^{|\mathcal{D}_{train}|} \ln p(y_{kz}, \mathbf{v}_z) \tag{3.73}$$

4.0 FFN INITIALIZATION METHODS

It can be a daunting task to select parameters to initialize FFNs, like the one shown in Figure 4.1. In this simple network configuration, there are 8 weight and 3 bias parameters that must be determined, creating an 11 dimensional parameter space that is searched during training for the optimal parameters. The optimal parameters produce the smallest network error on the training data and also will allow the network to perform with low error on the unseen test data.

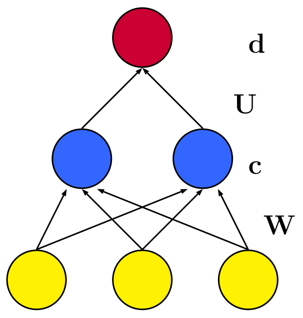


Figure 4.1: Randomly initialized FFN model

This chapter presents different approaches for initializing FFNs. Previous methods are presented in Sections 4.1 and 4.2. Section 4.1 describes techniques for initializing parameter values without using information from the training data. Methods applying information from the training dataset for model initialization are given in Section 4.2. Description of these methods will provide a basis for the development of the new pretraining method, as described in Section 4.3. This new method aims to provide more information from the data by initializing FFNs using cRBMs instead of nRBMs and compensates for converting cRBM models to FFN models.

4.1 METHODS FOR RANDOM PARAMETER INITIALIZATION

The purpose of graphical models such as FFNs and RBMs is to be able to learn information from or about a dataset. Beginning efforts at training FFN models used random values to initialize the weight and bias parameters [Larochelle et al., 2009],[Erhan et al., 2010],[Hinton, 2007]. It has been discovered empirically that the initial FFN parameter values influence the trained model [Larochelle et al., 2009],[Erhan et al., 2010]. This is also mentioned by Sarle [Sarle, 2002]. Since the weights and biases define hyperplanes within the input space which are required to separate the classes, these parameters ideally should have similarly ranging values as the training data [Sarle, 2002],[Erhan et al., 2010]. This not only helps to ensure that the model may learn the correct classifications, but also could reduce the number of training iterations required to move the hyperplanes to locations that enable the model to achieve the desired results [Sarle, 2002].

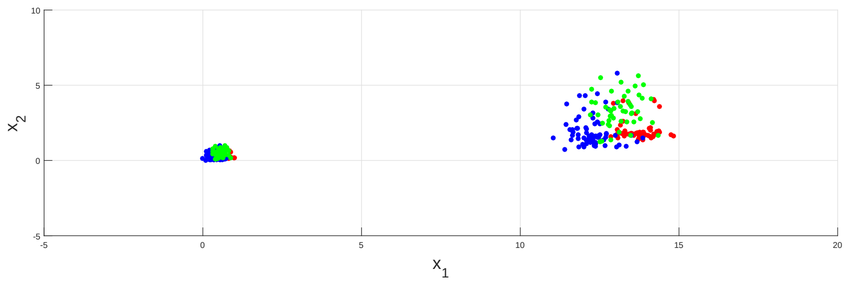


Figure 4.2: Class 1: red, Class 2: blue, Class 3: green

In Figures 4.2 and 4.3, Group 1, the set of points with larger variance, is an example of a dataset with its original values. Group 2, the set of points with smaller variance, is the corresponding normalized data, confined within the range $[0, 1]$ for both x_1 and x_2. In each group, there are three classes, shown using red, blue, and green. Figure 4.2 shows the difference that data normalization makes. Additionally, Figure 4.3 shows that the initial values of three FFN hyperplanes are all located within the range of the Group 2 data, but completely miss the space of the Group 1 data. In this case, normalizing the data allowed the initial hyperplanes and training data to be within a similar data space. Not initializing network parameters in a similar range as the training data may prevent the network from achieving the objective of separating the classes.

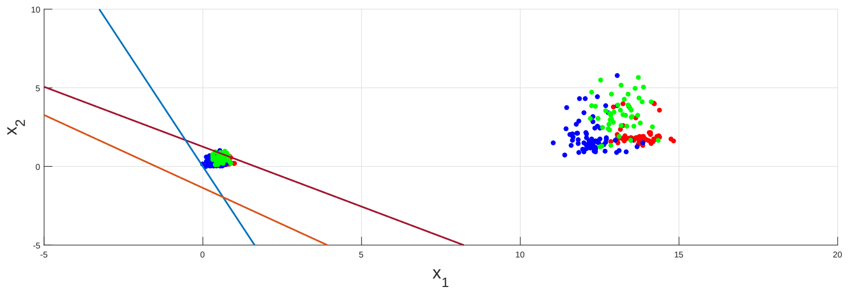


Figure 4.3: Class 1: red, Class 2: blue, Class 3: green

Shifting the data to a selected range and constraining the initial network parameters to a similar range provides another benefit derived from the choice of activation function [Sarle, 2002]. This activation function, $f(\mathbf{x})$, for a generic vector input $\mathbf{x}$ is applied to the linear combination of each layer's inputs and parameters as from the visible nodes to the hidden nodes, $f\left(\sum_{i=1}^{N} W_{ij}v_i + c_j\right)$. Common choices for these activation functions include the sigmoid function and the hyperbolic tangent function as shown in Figure 4.4. The difference between these two activation functions is the limiting value as the argument tends toward negative infinity. The asymptotic value for the sigmoid is 0, and that of the hyperbolic tangent is -1. Both activation functions are differentiable everywhere, which is important for calculating valid derivatives during training, with the largest derivative values provided in the linear region.

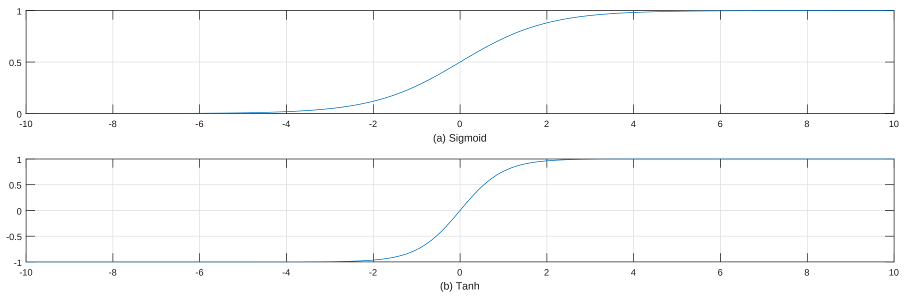


Figure 4.4: Common activation functions

Derivative update values outside of the activation function's linear range are small and provide little changes for consecutive parameter updates. Since smaller changes are achieved per iteration, more iterations may be required until reaching the desired optimal value [Sarle, 2002]. Scaling the input data to fall within the linear region of the activation function avoids operating in the flat saturation regions [Sarle, 2002]. Initializing the parameters in the same region then ensures training begins with the network parameters at a relevant spot [Sarle, 2002]. Even with these methods, the network parameters are still chosen randomly. There is no guarantee that training FFNs beginning from random initialization points will allow the network to perform well on training data or even to generalize to unseen test data.

4.2 METHODS FOR PARAMETER INITIALIZATION USING DATA

An alternate method to initialize a FFN model, known as pretraining, applies parameters resulting from unsupervised learning of a generative model such as an RBM [Hinton, 2007], [Mohamed et al., 2009] [Erhan et al., 2010]. With pretraining, improvements are expected as compared to using random parameter initializations [Sarle, 2002], [Larochelle et al., 2009], [Erhan et al., 2010]. Advantages from initializing FFNs with unsupervised pretraining methods are illustrated in the differences between pretrained FFNs and non-pretrained FFNs in [Erhan et al., 2010]. When training on the InfiniteMNIST dataset in [Erhan et al., 2010], it is noted how the digit features are more well-defined after backpropagation in the FFNs initialized with pretrained parameters than in the FFNs initialized with randomly selected parameters.

Even before FFN backpropagation training, the pretrained FFN hidden layers show digit-shaped features [Erhan et al., 2010]. This observation indicates that pretraining, by choosing an initialization dependent on the training data, helps to set the model along a training path that results in a model which is more tuned to the data domain than a randomly initialized model [Erhan et al., 2010]. In addition, Figures 5 and 6 of [Erhan et al., 2010] illustrate in reduced dimension function space that the functions of pretrained and non-pretrained networks are completely separate from one another, confirming that pretraining helps in determining the model's final parameter set.

Each pretraining approach is characterized by two distinct phases, where a generative model is trained in the first phase, and a discriminative model is trained in the second phase [Larochelle et al., 2012]. The generative model parameters are initialized randomly, and its parameters after training are used to initialize the discriminative model [Larochelle et al., 2012], [Larochelle and Bengio, 2008].

4.2.1 Single layer nRBM pretraining

Figure 4.5 shows an nRBM with only visible and hidden nodes [Hinton, 2007], [Mohamed et al., 2009],[Erhan et al., 2010]. The number of nRBM visible and hidden nodes must match that of the FFN.

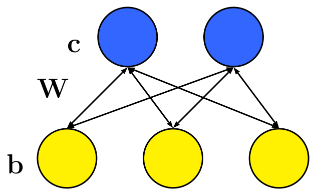


Figure 4.5: Randomly initialized nRBM

Training the nRBM is described in Chapter 2 [Hinton, 2007], [Mohamed et al., 2009]. After pretraining, the nRBM is transformed to look like a FFN, and the pretrained parameters initialize the corresponding FFN parameters using the nRBM forward direction

[Hinton, 2007], [Mohamed et al., 2009]. Comparing Equations 4.1 and 4.2 below will determine how the nRBM parameters initialize the FFN. First, the equation for one FFN hidden layer node is presented.

$$h_j = \sigma\left(c_j + \sum_{i=1}^{N} v_i W_{ij}\right), \ \forall j \in \{1, M\} \tag{4.1}$$

Next, the forward direction equation for one nRBM hidden layer node is provided.

$$h_j = \sigma\left(c_j + \sum_{i=1}^{N} v_i W_{ij}\right), \ \forall j \in \{1, M\} \tag{4.2}$$

The equations are equivalent, indicating a one-to-one correspondence between the nRBM and the FFN **W** matrix and **c** vector. Figure 4.6 illustrates initialization of the FFN with the nRBM. Pretrained parameters are indicated with bold arrows, and thin arrows denote randomly chosen parameters. Since FFNs are unidirectional, the backward direction of the nRBM is not needed, and the **b** vector is discarded [Hinton, 2007],[Mohamed et al., 2009].

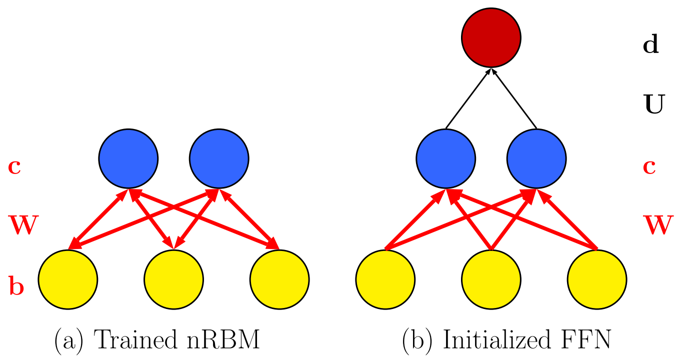


Figure 4.6: Converting nRBMs to FFNs

Figure 4.6 and Equations 4.1 and 4.2 show that the FFN does not have pretrained parameters for the **U** matrix and **d** vector in the classification layer, so these parameters are initialized randomly [Hinton, 2007], [Mohamed et al., 2009], [Erhan et al., 2010]. After initializing the FFN, backpropagation training is performed [Hinton, 2007], [Mohamed et al., 2009]. This pretraining method is known as greedy layerwise pretraining (GLP) [Hinton, 2007], [Erhan et al., 2010].

4.2.2 Multiple layers nRBM pretraining

A method for pre-training deep FFN models is derived from the unsupervised pre-training of a generative model known as a deep belief network (DBN), shown in Figure 4.7 [Hinton, 2007],[Larochelle et al., 2009], [Erhan et al., 2010]. Adding hidden layers to create deep models applies more transformations to the input features. Feature evolution is illustrated in Figure 3 of [Erhan et al., 2010] using the InfiniteMNIST dataset. Features at lower levels are curves and lines, while features at higher levels are numbers, the objects that the model is designed to classify [Erhan et al., 2010]. With speech recognition, the lower level features could represent disjoint sections in the time-frequency region space. Features from higher level hidden nodes may indicate regions corresponding to phonemes, units of sound that are combined to create the sounds of words for a specific language. The number of FFN parameters increases as more layers are added.

4.2.2.1 DBN The generative DBN model has only visible and hidden nodes and is created by stacking nRBMs using GLP, which builds the DBN to its full size [Hinton, 2007], [Mohamed et al., 2009]. The first step in GLP is to train an nRBM with CD_K [Hinton, 2007], [Mohamed et al., 2009]. After a single nRBM is trained, another set of nodes is put on top of the original nRBM model [Hinton, 2007], [Mohamed et al., 2009]. The visible nodes' parameters are frozen, and the new RBM exists between the original hidden nodes and the additional nodes [Hinton, 2007], [Mohamed et al., 2009]. The original hidden nodes become the new nRBM visible nodes, and the additional nodes are the new nRBM hidden nodes [Hinton, 2007], [Mohamed et al., 2009]. By freezing the parameters of the previously trained layers below, only a single nRBM is trained at a time using CD_K. This stacking process is illustrated in Figure 4.7.

In the Figure 4.7, the current visible nodes are yellow and the current hidden nodes are blue. These colored nodes indicate the layers that are being trained for a given step, and the double headed arrow between the layers indicates that the training uses CD_K as with individual nRBMs. The parameters corresponding to the black nodes are not altered during a given training phase [Erhan et al., 2010]. The number of phases comprising pre-training equals the number of layers, illustrating that this method operates on each layer individually [Erhan et al., 2010].

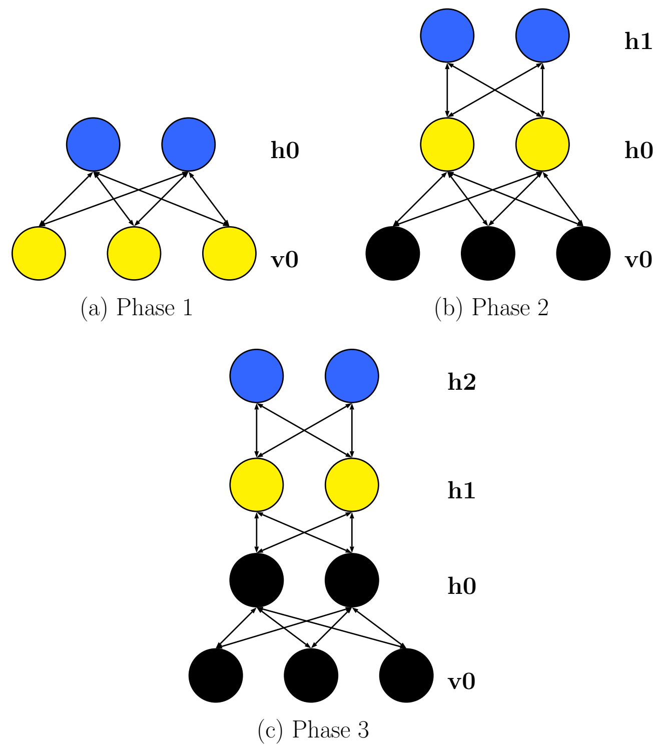


Figure 4.7: Greedy layerwise pre-training: three phases

When stacking nRBMs, the hidden layer probability-like values output from one hidden layer are passed to the next nRBM as the visible layer inputs, instead of using sampled values from the hidden layer [Hinton, 2007], [Larochelle et al., 2009]. This helps to reduce the variability in the training data for the next nRBM layer [Hinton, 2007]. Otherwise, many more binary examples would be required to be added to the training data so that the number of times that each node is active, relative to all possible combinations of within-layer activations of other nodes, corresponds to the probability resulting from the previous layer's output.

After the RBM pre-training is completed for the last layer and the desired number of hidden layers is obtained, the set of DBN parameters is used to initialize a FFN model [Fischer and Igel, 2012], [Erhan et al., 2010]. The generative capabilities of the DBN are ignored by eliminating the biases for the visible nodes [Erhan et al., 2010]. Again, randomly chosen parameter values must be used for the FFN classification layer [Hinton, 2007], [Mohamed et al., 2009]. Once the FFN is initialized, backpropagation training can be performed [Fischer and Igel, 2012], [Erhan et al., 2010].

4.2.2.2 Associative DBN (aDBN) The difference between aDBN and DBN models is in the last three layers, since an aDBN model implements a cRBM in those layers [Hinton, 2007], [Mohamed et al., 2009]. Figure 4.9 illustrates cRBMs, which were explained in Chapter 3. Constructing an aDBN uses GLP as described in Section 4.2.2.1 until the last hidden layer [Hinton, 2007], [Mohamed et al., 2009]. At the last hidden layer, a cRBM model is applied, and this cRBM is trained using CD_K, adapting the calculation of the hidden node values to include inputs from both the $\mathbf{v}$ and $\mathbf{e}_y$ data and label vectors [Hinton, 2007], [Mohamed et al., 2009].

Even though the aDBN has a label layer, the model itself is still generative [Hinton, 2007], [Mohamed et al., 2009]. A hybrid equation similar to that given in Chapter 3, describing generative and discriminative probability functions, is applied [Mohamed et al., 2009]. In one method of fine tuning, gradients of the discriminative term $p(C|\mathbf{v})$ from the hybrid equation $f(\mathbf{v}, \mathbf{e}_y) = \alpha p(C|\mathbf{v}) + p(\mathbf{v}|C)$ are used to train all of the model parameters after GLP [Mohamed et al., 2009]. Figure 4.8 shows an aDBN model, as illustrated in [Hinton, 2007] and [Mohamed et al., 2009]. It is important to note that this is still a generative model, which was used for classification [Hinton, 2007], [Mohamed et al., 2009]. It does not become a true FFN, since as a type of DBN, the joint PDF has been learned instead of the posterior PDF. This issue is addressed in the next section.

4.3 NEW METHODS FOR PARAMETER INITIALIZATION USING DATA

Models introduced in this work take advantage of the label layer included in the cRBMs, using modified cRBM models for FFN initialization. Experiments in this work focused on pretraining FFN models with one hidden layer. Methods to extend this initialization technique beyond one hidden layer were not explored but will be proposed in Chapter 6.

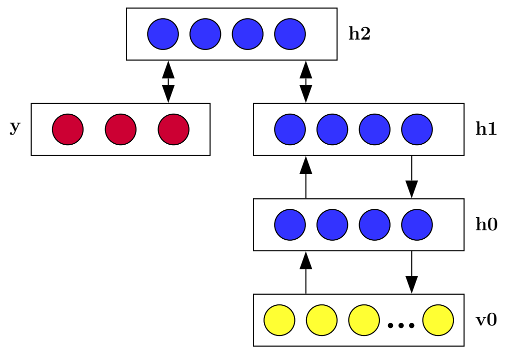


Figure 4.8: DBN with top level associative layer

The last layer of a FFN uses the softmax activation function to perform the actual classification. Pretraining all but the classification parameters provides gains, and pretraining the classification parameters as with an aDBN also improves performance [Hinton, 2007], [Mohamed et al., 2009]. Within the structure of a FFN, it is hoped that initializing all layers with pretrained parameters will improve performance beyond the gains of initializing FFNs with DBN parameters [Hinton, 2007], [Mohamed et al., 2009].

For initializing a FFN with one hidden layer, the parameters of a single cRBM are sufficient. The number of visible, hidden, and label nodes of the cRBM must match the number of corresponding nodes in the FFN model to be initialized. The cRBM models may be trained with any of the generative, discriminative, or hybrid methods described in Chapter 3. After cRBM training completes, the parameters are used to initialize the FFN by unfolding the cRBM. Once the FFN model is initialized, backpropagation training occurs, and the trained FFN model is evaluated.

4.3.1 Initializing FFNs with cRBMs

To visualize the unfolding process, Figure 4.9 shows the randomly initialized cRBM model. Figures 4.10 and 4.11, presented later, show the mapping between the trained cRBM and the initialized FFN. In these figures, similar notation is used as in Chapters 2 and 3.

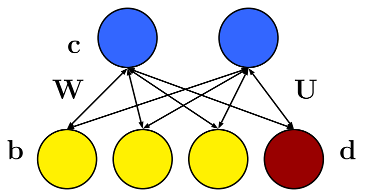


Figure 4.9: Randomly initialized cRBM

The cRBM provides the FFN with pretrained values for the **W** and **U** matrices and the **c** and **d** vectors. Comparison of the hidden node equations between the cRBM model and the FFN model will illustrate why, when, and which parameter alterations should be considered. Again, the equation for one FFN hidden layer node is presented.

$$h_j = \sigma\left(c_j + \sum_{i=1}^{N} v_i W_{ij}\right),\ \forall j \in \{1, M\} \tag{4.3}$$

Next, the equation for one cRBM hidden layer node is given. This equation includes the simplifications of an output layer vector with only one active node y_k and is true for all $k \in \{1, C\}$.

Comparing Equations 4.3 and 4.4 shows that there is no change to the scalar W_{ij} between the pretrained cRBM model and the FFN model. Therefore, the **W** matrix from the pretrained cRBM model can be substituted directly for the FFN **W** matrix. However, Equation 4.4 shows an additional term which comes from the weight matrix **U**, between the hidden nodes and the label nodes.

$$h_j = \sigma\left(c_j + \sum_{i=1}^{N} v_i W_{ij} + U_{kj}\right),\ \forall j \in \{1, M\}, \forall k \in \{1, C\} \tag{4.4}$$

This additional term appears because the label nodes are included in the cRBM models. As shown in Figure 4.9, including these label nodes increases the number of inputs to the hidden layer nodes by the number of classes in the data [Larochelle and Bengio, 2008], [Larochelle et al., 2012]. Because of the assumption that the vector $\mathbf{e}_y$ is zero except at index k for any single hidden node h_j, the influence of the matrix $\mathbf{U}$ is reduced to a scalar U_{kj}. Therefore, U_{kj} can be treated as a bias parameter, as shown in Equation 4.5 by grouping the bias terms.

$$h_j = \sigma\left([c_j + U_{kj}] + \sum_{i=1}^{N} v_i W_{ij}\right), \ \forall j \in \{1, M\}, \forall k \in \{1, C\} \tag{4.5}$$

Training modifies the $\mathbf{U}$ parameters, and these parameters influence the location of the cRBM hidden node hyperplanes, along with the values for the other parameters. When initializing the FFN with pretrained parameters, moving the hyperplanes' locations away from those data clusters found during pretraining may not be beneficial, especially if those clusters relate to the desired data structure as determined by the labels. The hyperplanes would shift during FFN initialization by omitting the effects of these U_{kj} parameters, caused by the loss of the label nodes as hidden layer inputs. To reduce these effects during FFN initialization, the FFN hidden node bias vector $\mathbf{c}$ becomes $\bar{\mathbf{c}}$.

$$\bar{c}_j = c_j + U_{kj} \tag{4.6}$$

Equation 4.6 is only valid during pretraining, when the assumption that $\mathbf{e}_y = y_k$ is true. Since the bias changes are implemented after pretraining, the effect of all of the label vectors $\mathbf{e}_y$ should be taken into account over the entire training set of N_{tr} examples. This information is summarized per class k using the average $\overline{y_k}$ and is multiplied by the corresponding parameters in the $\mathbf{U}$ matrix for each class k. To account for the effect from each class k, the product of $\overline{y}_k$ with the $\mathbf{U}$ parameters is summed over all classes as shown in Equation 4.7. This is also shown in vector form in Equation 4.8.

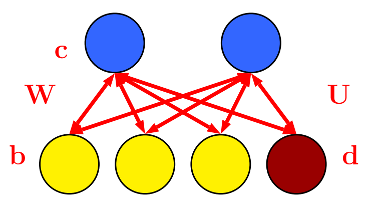
$$\bar{c}_j = c_j + \sum_{k=1}^{C} U_{kj} \bar{y}_k \tag{4.7}$$

$$\bar{\mathbf{c}} = \mathbf{c} + \sum_{k=1}^{C} \mathbf{U}_k \left[\frac{1}{N_{tr}} \sum_{n=1}^{N_{tr}} y_k(n) \right] \tag{4.8}$$

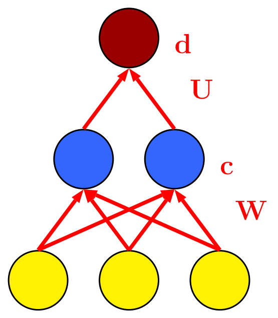
Since the label nodes of the cRBM model only ever receive input from the hidden nodes, there is no need to adjust the **U** or **d** parameters when substituting pretrained cRBM parameters into the FFN models. The equation for the label nodes of the FFN with a softmax final layer is the same as Equation 4.9 for the cRBM.

$$p(y_k|\mathbf{h}) = \frac{\exp\left(d_k + \sum_{j=1}^{M} h_j U_{jk}\right)}{\sum_{l=1}^{C} \exp\left(d_l + \sum_{j=1}^{M} h_j U_{jl}\right)} \tag{4.9}$$

The FFNs initialized without and with bias compensation **c** and $\bar{\mathbf{c}}$ are illustrated below in Figures 4.10 and 4.11. Comparison of the two models before backpropagation training would provide insight on the effect of the hidden layer biases. It is expected that the bias adjustment will provide better results.

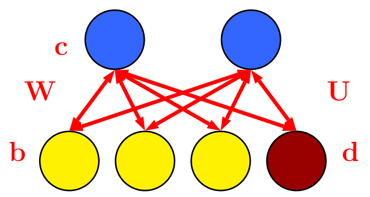


(a) Trained cRBM

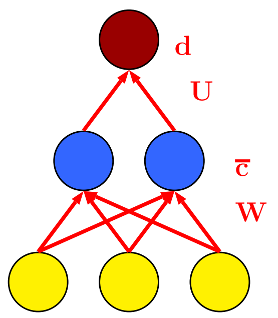


(b) Initialized FFN

Figure 4.10: Converting cRBMs to FFNs: unadjusted hidden biases



(a) Trained cRBM



(b) Initialized FFN

Figure 4.11: Converting cRBMs to FFNs: adjusted hidden biases

5.0 EXPERIMENTS

The goal of these experiments is to evaluate the performance of different pretraining techniques for two-layer (one-hidden-layer) FFN models on a classification task. The sample dataset has 178 examples, and each example is characterized by 13 inputs and belongs to one of three classes. Section 5.1 provides details regarding the training methods and describes the types of models used in the experiments. Section 5.2 presents and discusses the experimental results.

5.1 METHODS AND MODELS

Explanation of the model training methods is provided before introducing the different models evaluated in these experiments as listed in Table 5.3. All FFN models have 13 input nodes, one hidden layer with 10 nodes, and three output nodes. The input values are mapped in the range of $[0, 1]$, and this mapping is required for the RBMs with Bernoulli visible nodes. Since the RBMs are used to initialize FFNs, it is important that the information learned by the pretrained parameters is relevant to the FFNs as well. Therefore, the RBMs and the FFNs must be trained using the same data.

The FFN hidden nodes use a sigmoid activation function, which also matches the $p(H_j = 1|\mathbf{v})$ for the RBMs, and the FFN output layer uses a softmax activation function for classification. The cross-entropy performance function is used also since the model performs classification. All FFN random parameters are chosen from a uniform distribution between $(0, 0.1)$, and so the parameters are within the range of the inputs as described in Chapter 4. After initialization, the FFN models are trained with backpropagation using scaled conjugate gradient descent. [1] Conjugate gradient descent reformulates the error function surface to optimize the direction selection for subsequent training steps and is expected to converge to a solution faster than the steepest gradient descent method [Shewchuk et al., 1994].

The only regularization applied to the FFNs besides pretraining is early stopping, which measures the errors on the validation dataset [Erhan et al., 2010].[2] After the number of validation errors increases for a number of consecutive training iterations, training ends. Early stopping helps the FFN models to generalize well to test data, which is important for the networks that are initialized randomly. To assess fairly the effects of pretraining methods, these techniques should be applied in situations where training already performs well. To maintain consistency in the training methods between the FFN models initialized randomly and those initialized with pretrained parameters, early stopping is used again. Explanation of how the divisions between training, validation, and testing data were chosen are explained in Section 5.2. Seventy percent of the data was chosen for training to give the network a majority of training examples while having enough remaining examples for testing and validation. The FFN initialization parameters are varied while the training dataset is held constant, and each model's metrics are calculated over 100 trials. The randomization seed is reset per parameter configuration to ensure more consistency in the results. Table 5.1 summarizes the FFN training parameters.

All RBM models are Bernoulli-Bernoulli models. Using Bernoulli hidden nodes finds binary-valued clusters in the data, which is important since the RBMs are used to initialize FFNs that perform classification. Bernoulli visible nodes can be applied after the data is scaled to the range of $[0, 1]$. For the cRBM models, the label nodes should be Bernoulli nodes to indicate probability-like values of choosing a particular cluster. Future work could examine the difference between RBMs modeling the scaled input data using Bernoulli visible nodes and RBMs with visible nodes matching the distribution of the unscaled input data.

The nRBM models have 13 visible nodes and 10 hidden nodes, while the cRBM models have 13 visible nodes, 3 label nodes, and 10 hidden nodes. Random parameters again were chosen using the same method as with the FFNs for consistency between the model types.

[1] The code used to create, initialize, and train FFN models is derived from the Matlab Neural Network toolbox, Versions R2015b and R2016a.

[2] Full batch training methods are used, so iterations and epochs are equivalent.

The nRBMs were trained as described in Chapter 2 for either 1000 or 2000 epochs, and the cRBMs were trained as described in Chapter 3 for either 1000 or 2000 epochs [3]. For the cRBM models, a hyperparameter, $\alpha \in [0.0, 1.0]$, controls the proportion of summed generative and discriminative parameter gradients. For $\alpha = 0.0$, complete discriminative training is performed, and for $\alpha = 1.0$, complete generative training is applied. The proportion of generative training increases with α.

Validation data for nRBM models may monitor the models' kurtosis, since nRBMs do not measure classification error. Since use of validation data between the nRBM and cRBM models could not be interpreted in the same way, training with a fixed number of epochs provides more consistency for comparing training methods. No other regularization methods are applied.

The FFN and RBM hyperparameter values were chosen from the defaults specified in the code. These values were maintained since the randomly initialized FFN models provided low error rates on the test data. The number of RBM training epochs was doubled to determine any effect of training time on the performance of purely discriminative versus purely generative models [Lasserre et al., 2006]. As many training hyperparameters as possible were kept consistent between FFN and RBM training. Since the cRBM models are directly compared as classifiers to the FFNs, keeping hyperparameters such as the learning rate and momentum the same emphasizes the differences in training methods inherent to the model types. Also, the nRBM and cRBM training used the same hyperparameters to be able to compare FFN initialization methods more directly. Table 5.2 summarizes the RBM training parameters.

[3]The code used to create, initialize, and train RBM models was derived from that updated by Søren Kaae Sønderby. The code can be found at: `https://github.com/skaae/rbm_toolbox`.

Table 5.1: FFN training parameters

Parameter Name	**Value**
Number of trials	100
Training function	scaled conjugate gradient descent
Early stopping	yes
Number of early stopping validation checks	6
Learning rate	0.01
Momentum	0.9
Batch method	full
Batch size	124
Performance function	cross-entropy
Training ratio	70%
Validation ratio	10%
Testing ratio	20%
L_2 normalization	0
Range of inputs	$[0, 1]$

Each FFN is evaluated both before and after backpropagation training. These phases of training are labeled as the Initialized and Trained stages when referring to the models. When comparing across model types, the same phases will be used. To determine the effect of backpropagation on pretraining, the different phases will be compared on individual model types. [Larochelle and Bengio, 2008] and [Larochelle et al., 2012] report comparable results on classification tasks between cRBM models and traditional FFN models. This is examined in addition to using cRBMs for FFN pretraining. Table 5.3 summarizes the list of models used in the experiments.

Table 5.2: RBM training parameters

Parameter Name	**Value**
Number of trials	100
Training approximation	CD_1
Range of α	$[0, 1]$
Early stopping?	No
Number of training epochs	1000, 2000
Learning rate	0.01
Momentum	0.9
Batch method	full
Batch size	124
Training ratio	70%
Validation ratio	10%
Testing ratio	20%
L_2 normalization	0
Range of inputs	$[0, 1]$

Table 5.3: Experimental models

Model	**Description**
r12N:	All parameters initialized randomly
	All parameters trained with backpropagation
	Baseline model for comparison
p1N_1:	Initialize using $\mathbf{W}$ and $\mathbf{c}$ taken from nRBM
	Initialize using same random $\mathbf{U}$ and $\mathbf{d}$ as r12N
	$\mathbf{W}$ and $\mathbf{c}$ not trained with backpropagation
p1N_2:	Initialize using $\mathbf{W}$ and $\mathbf{c}$ taken from nRBM
	Initialize using same random $\mathbf{U}$ and $\mathbf{d}$ as r12N
	All parameters trained with backpropagation
p12N, $\alpha = 0.0$:	Initialize using $\mathbf{W}$, $\mathbf{c}$, $\mathbf{U}$, $\mathbf{d}$ taken from DRBM
	All parameters trained with backpropagation
p12N, $0.1 \leq \alpha \leq 0.9$:	Initialize using $\mathbf{W}$, $\mathbf{c}$, $\mathbf{U}$, $\mathbf{d}$ taken from HRBM
	All parameters trained with backpropagation
p12N, $\alpha = 1.0$:	Initialize using $\mathbf{W}$, $\mathbf{c}$, $\mathbf{U}$, $\mathbf{d}$ taken from GRBM
	All parameters trained with backpropagation
p12bN, $\alpha = 0.0$:	Initialize using $\mathbf{W}$, $\bar{\mathbf{c}}$, $\mathbf{U}$, $\mathbf{d}$ trained from DRBM
	All parameters trained with backpropagation
p12bN, $0.1 \leq \alpha \leq 0.9$:	Initialize using $\mathbf{W}$, $\bar{\mathbf{c}}$, $\mathbf{U}$, $\mathbf{d}$ trained from HRBM
	All parameters trained with backpropagation
p12bN, $\alpha = 1.0$:	Initialize using $\mathbf{W}$, $\bar{\mathbf{c}}$, $\mathbf{U}$, $\mathbf{d}$ trained from GRBM
	All parameters trained with backpropagation
RBM:	All parameters initialized randomly
	nRBMs and cRBMs trained as in Ch 2 and Ch 3

5.2 RESULTS

In these experiments, the model performance is quantified by the mean and standard deviation of the model's classification error rates along with the number of epochs required for backpropagation training. Each of the average and standard deviation values are calculated over the set of 100 trials for each model, and results are presented this way as $\mu \pm \sigma$.

Section 5.2.1 explains the choice for test and validation data proportions. Tables 5.7 and 5.8 are given in Section 5.2.2. These tables provide the results per model on the Train 20% data subset when RBMs are trained using 1000 and 2000 epochs, respectively. Sections 5.2.3 through 5.2.5 evaluate the effects of the different model initializations. Section 5.2.6 examines the effect of pretraining on the number of required backpropagation training epochs. Section 5.2.7 looks at the effect of pretraining on the training error.

5.2.1 Varying testing and validation data proportions

After choosing seventy percent of the data for training, the proportions of validation and test data were selected. It was observed that varying the size of the test and validation sets causes different results on the baseline FFN and cRBM models as summarized in the next set of tables. Three quantities of test and validation data were used in training models, $\{20\%, 10\%\}$, $\{15\%, 15\%\}$, and $\{10\%, 20\%\}$. Results on the baseline model (r12N) are summarized in Table 5.4.

Table 5.4: Baseline FFN: varying data proportions

Test,Val	**Num train epochs**	**Train % error**	**Val % error**	**Test % error**
0.20, 0.10	28.235 ± 5.618	0.134 ± 0.396	7.009 ± 2.464	0.032 ± 0.297
0.15, 0.15	37.751 ± 8.716	0.020 ± 0.150	0.031 ± 0.309	3.590 ± 0.831
0.10, 0.20	46.114 ± 13.168	0.020 ± 0.150	0.000 ± 0.000	7.333 ± 2.605

Changing the proportions of test and validation data affects the test results when training baseline FFNs, and the percentage of errors on the test dataset increases as the proportion of test data decreases. This is explained by the fact that individual errors on smaller datasets are weighted more heavily.

The upper limit of the average percent error on the training data across the three datasets is less than 0.600%, indicating that training completed successfully on all three data subsets. The next tables show the performance of training cRBM models across the three different data subsets. Table 5.5 provides the information for the cRBMs trained using 1000 epochs, and the results for trained cRBMs using 2000 epochs are in Table 5.6.

Table 5.5: cRBM 1000 Ep: varied proportions

Test, Val	**Train % error**	**Test % error**
0.20, 0.10	1.576 ± 0.734	1.194 ± 1.235
0.15, 0.15	2.523 ± 0.754	6.455 ± 1.691
0.10, 0.20	1.889 ± 0.733	13.702 ± 2.189

Table 5.6: cRBM 2000 Ep: varied proportions

Test, Val	**Train % error**	**Test % error**
0.20, 0.10	0.455 ± 0.326	0.672 ± 1.205
0.15, 0.15	1.274 ± 0.383	6.040 ± 0.871
0.10, 0.20	0.620 ± 0.340	13.040 ± 1.428

In each of the cRBM results tables, the average training percent errors are within one percent of each other across the three dataset divisions. Also, the averages of standard deviations show little variation across the three dataset divisions as on the training data for the baseline FFN models. These results reinforce that training was performed successfully on the three data subsets, and that no subset of training data was more problematic for training. The cRBM test percent errors generally increase as the size of the test dataset decreases.

Using the smallest weight, or percent increment, per error per trial may help to detect trends more easily among the pretrained models. Otherwise, results may seem to have large percent errors that may actually be attributed to one or two misclassifications. Use of a larger dataset would also minimize this issue. Since both the baseline FFN and the cRBMs have the smallest percentage of errors on the test data, and since this dataset has the most examples, (Test Percent = 20%), further analysis comparing the FFN parameter pretraining methods will use this data division.

5.2.2 Pretrained model test results

In Tables 5.7 and 5.8, the p1N_1 and p1N_2 models do not have values in the cRBM column. These models are initialized from nRBMs, and their corresponding nRBMs cannot be evaluated for classification. The r12N and cRBM values are not repeated for the p12N and p12bN subsections.

5.2.3 p1N_1 and p1N_2 models

These p1N_1 and p1N_2 models are initialized using the technique described in Section 4.2.1. The purpose of the p1N_1 model is to determine the effects of training the parameters between the inputs and hidden nodes with one method and the parameters between the hidden nodes and the outputs with a different method. The p1N_2 models provide the closet comparison to multi-hidden-layer FFNs, initialized similarly, from [Hinton, 2007], [Mohamed et al., 2009].

Table 5.7: Test percent error: RBM 1000 epochs

Model	**Baseline (r12N)**	**Initialized**	**Trained**	**cRBM**
$p1N\_1$	0.056 ± 0.391	66.417 ± 8.856	16.722 ± 4.307	-
$p1N\_2$	0.056 ± 0.391	66.417 ± 8.856	0.833 ± 1.451	-
$p12N$ $\alpha = 0.0$	0.056 ± 0.391	10.778 ± 6.988	0.056 ± 0.391	6.306 ± 3.461
$\alpha = 0.1$	0.028 ± 0.278	6.306 ± 4.568	0.000 ± 0.000	2.833 ± 2.433
$\alpha = 0.2$	0.028 ± 0.278	5.472 ± 2.748	0.000 ± 0.000	1.389 ± 1.994
$\alpha = 0.3$	0.028 ± 0.278	6.167 ± 2.452	0.000 ± 0.000	0.500 ± 1.272
$\alpha = 0.4$	0.028 ± 0.278	7.694 ± 3.323	0.000 ± 0.000	0.139 ± 0.609
$\alpha = 0.5$	0.028 ± 0.278	9.611 ± 3.584	0.000 ± 0.000	0.111 ± 0.547
$\alpha = 0.6$	0.028 ± 0.278	11.056 ± 3.231	0.000 ± 0.000	0.028 ± 0.278
$\alpha = 0.7$	0.028 ± 0.278	12.139 ± 2.754	0.000 ± 0.000	0.028 ± 0.278
$\alpha = 0.8$	0.028 ± 0.278	12.694 ± 2.311	0.000 ± 0.000	0.000 ± 0.000
$\alpha = 0.9$	0.028 ± 0.278	13.056 ± 2.032	0.000 ± 0.000	0.167 ± 0.663
$\alpha = 1.0$	0.028 ± 0.278	12.972 ± 2.304	0.000 ± 0.000	1.639 ± 2.055
$p12bN$ $\alpha = 0.0$	-	14.917 ± 6.777	0.000 ± 0.000	-
$\alpha = 0.1$	-	10.833 ± 6.494	0.000 ± 0.000	-
$\alpha = 0.2$	-	7.194 ± 5.871	0.000 ± 0.000	-
$\alpha = 0.3$	-	4.833 ± 4.764	0.000 ± 0.000	-
$\alpha = 0.4$	-	3.667 ± 4.404	0.000 ± 0.000	-
$\alpha = 0.5$	-	3.389 ± 3.500	0.000 ± 0.000	-
$\alpha = 0.6$	-	3.333 ± 3.442	0.000 ± 0.000	-
$\alpha = 0.7$	-	3.611 ± 3.498	0.000 ± 0.000	-
$\alpha = 0.8$	-	3.972 ± 3.624	0.000 ± 0.000	-
$\alpha = 0.9$	-	4.473 ± 3.846	0.000 ± 0.000	-
$\alpha = 1.0$	-	6.250 ± 4.491	0.000 ± 0.000	-

Table 5.8: Test percent error: RBM 2000 epochs

Model	**Baseline (r12N)**	**Initialized**	**Trained**	**cRBM**
$p1N\_1$	0.056 ± 0.391	65.667 ± 13.822	0.917 ± 1.631	-
$p1N\_2$	0.056 ± 0.391	65.667 ± 13.822	0.000 ± 0.000	-
$p12N$ $\alpha = 0.0$	0.111 ± 0.876	10.167 ± 2.097	0.000 ± 0.000	0.417 ± 1.332
$\alpha = 0.1$	0.056 ± 0.391	12.528 ± 2.418	0.000 ± 0.000	0.278 ± 0.838
$\alpha = 0.2$	0.056 ± 0.391	12.889 ± 2.071	0.000 ± 0.000	0.778 ± 1.254
$\alpha = 0.3$	0.056 ± 0.391	12.833 ± 1.965	0.000 ± 0.000	1.056 ± 1.355
$\alpha = 0.4$	0.056 ± 0.391	12.861 ± 1.879	0.000 ± 0.000	0.889 ± 1.302
$\alpha = 0.5$	0.056 ± 0.391	12.972 ± 1.976	0.000 ± 0.000	0.583 ± 1.137
$\alpha = 0.6$	0.056 ± 0.391	13.194 ± 1.989	0.000 ± 0.000	0.361 ± 0.939
$\alpha = 0.7$	0.056 ± 0.391	13.444 ± 1.963	0.000 ± 0.000	0.361 ± 0.939
$\alpha = 0.8$	0.056 ± 0.391	13.667 ± 1.961	0.000 ± 0.000	0.222 ± 0.757
$\alpha = 0.9$	0.056 ± 0.391	13.944 ± 2.051	0.000 ± 0.000	0.222 ± 0.757
$\alpha = 1.0$	0.056 ± 0.391	13.972 ± 2.511	0.000 ± 0.000	2.222 ± 2.649
$p12bN$ $\alpha = 0.0$	-	0.611 ± 1.702	0.000 ± 0.000	-
$\alpha = 0.1$	-	0.444 ± 1.462	0.000 ± 0.000	-
$\alpha = 0.2$	-	1.222 ± 2.101	0.000 ± 0.000	-
$\alpha = 0.3$	-	3.167 ± 3.351	0.000 ± 0.000	-
$\alpha = 0.4$	-	4.861 ± 3.795	0.000 ± 0.000	-
$\alpha = 0.5$	-	5.917 ± 4.116	0.000 ± 0.000	-
$\alpha = 0.6$	-	6.694 ± 4.327	0.000 ± 0.000	-
$\alpha = 0.7$	-	7.083 ± 4.644	0.000 ± 0.000	-
$\alpha = 0.8$	-	7.028 ± 4.745	0.000 ± 0.000	-
$\alpha = 0.9$	-	6.556 ± 4.828	0.000 ± 0.000	-
$\alpha = 1.0$	-	8.611 ± 5.549	0.000 ± 0.000	-

5.2.3.1 Effect of initializing with pretrained W and c parameters Comparing the results for the r12N and p1N_2 Trained models shows that the error is better for the nRBMs trained for 2000 epochs than those trained for 1000 epochs on this dataset. The nRBM trained using 2000 epochs produces a trained p1N_2 model with zero error on this test data, which is less than the r12N baseline. This result shows that pretraining the first layer of weights and hidden layer biases may provide improvements over random FFN initialization. From [Hinton, 2007], Table 1 shows that initializing a multi-layer FFN by GL pretraining of a DBN and applying backpropagation training provides improvement over a multi-layer FFN initialized with randomly chosen parameters. Pretraining all but the classification layer parameters may provide benefits for multi-layer FFNs as well.

5.2.3.2 Effect of testing using randomly chosen classification parameters The results for the Initialized p1N_1 and p1N_2 models initialized with nRBMs trained for either 1000 or 2000 epochs show that the effects of randomly chosen classification layer parameters dominate the classification results. For these models, the error on the test set shows that the model chooses the incorrect answer two out of three times. Since this dataset has three classes, the model with randomly chosen classification parameters selects a class completely randomly. Not surprisingly, pretraining the $\mathbf{W}$ and $\mathbf{c}$ parameters does not compensate for a lack of training for the classification parameters.

5.2.3.3 Effect of applying different training methods per layer Comparing the results of Trained p1N_1 and p1N_2 models initialized with nRBMs trained with 1000 or 2000 epochs illustrates the benefits of backpropagation training. In each case, classification using parameters trained with different methods provides worse results on this test data.

The pretrained parameters are taken from nRBM models, which are trained generatively without any label information. These nRBM models' parameters are able to identify clusters in the data, which have no guaranteed relationship to the dataset classes. Training the p1N_1 models essentially allows the hidden node values produced by the nRBMs' parameters to be used for classification. The ability of the clusters to predict classes may affect this p1N_1 model's performance.

If the clusters provide information to predict the classes well, training the $\mathbf{W}$ and $\mathbf{c}$ parameters using nRBMs could show improvements from these results. In this case, the learned nRBM parameters would provide a nonlinear data transformation, allowing for a more efficient class separation. This could produce similar effects as a support vector machine (SVM) model with kernel functions which first maps inputs into a higher dimensional space before performing linear classification [Theodoridis and Koutroumbas, 2008].

5.2.4 p12N models

The p12N models are initialized as described in Section 4.3.1. The $\mathbf{c}$ parameters are taken directly from the cRBM model without any adjustments. The purpose of these models is to determine the effect of initializing all of the FFN parameters as compared to all but the classification layer parameters.

5.2.4.1 Effect of initializing with pretrained W, c, U, and d parameters

The Trained p12N FFNs generally show improvements as compared to the r12N FFNs on this test data. For only one configuration ($\alpha = 0.0$ in Table 5.7) is the result equivalent to the baseline.

Pretraining all model parameters, even if the cRBM biases are not compensated for the lack of inputs when unfolded, provides improvements over FFN random initializations on this test dataset. Since the Trained p1N_2 model in Table 5.8 achieved the same results as most of the p12N Trained models, it is not required to pretrain the whole model to achieve better than randomly initialized models. For these data, backpropagation is able to compensate for randomly initialized classification parameters.

Regardless, model improvements can still be seen by comparing the Initialized p1N_1 and p1N_2 models with the Initialized p12N models. This shows that pretraining the classification parameters provides model improvements, since performance before backpropagation training is better than random selection. While the Initialized p12N model performance on this test data is not better than the standalone cRBM models, it is hoped that these pretraining methods employing cRBMs can be extended for multi-hidden-layer FFN models. In contrast, the cRBM models are restricted to have one hidden layer. The advantages of adding more hidden layers is initializing deeper FFN models that extract more abstract and meaningful features, useful for classification.

[Mohamed et al., 2009] reports improvements in results when learning the weights between the classes and the last hidden layer. Using the training data for pretraining the classification layer parameters would be expected to provide improvements, especially when considering the influence of these parameters, as indicated in Section 5.2.3.

5.2.4.2 Effect of α On these data, α affects the evaluation of the cRBM models most significantly. However, for cRBMs trained with 1000 epochs, adding generative training reduces the average and standard deviation of the test errors when initializing the p12N models. This is shown since the Trained p12N models, except for $\alpha = 0.0$, consistently produce errors of 0.000 ± 0.000. The Initialized p12N models have larger errors, and the σ values are rather consistent in both Tables 5.7 and 5.8. For both sets of results, the cRBM with $\alpha = 0.8$ produces both the smallest average and standard deviation values. This agrees with [Larochelle and Bengio, 2008], which mentioned obtaining the best results with the hybrid models. For the Table 5.7 results, the purely discriminative cRBM model achieved the largest average and standard deviation values, while the purely generative cRBM model achieved the same for the Table 5.8 results. It is mentioned in [Lasserre et al., 2006] that the asymptotic limit of generative models' error is achieved after fewer training iterations than that of discriminative models. This may help to explain the trend in the cRBM results when comparing Tables 5.7 and 5.8 for each α. For $0.0 \leq \alpha \leq 0.2$, the cRBMs trained for fewer iterations have larger error averages and standard deviations. For $0.3 \leq \alpha \leq 1.0$, the cRBMs trained for fewer iterations have smaller error averages and standard deviations. As α increases, the models become more generative.

5.2.5 p12bN models

The p12bN models are initialized as described in Section 4.3.1. The $\bar{\mathbf{c}}$ parameters are the adjusted $\mathbf{c}$ parameters taken from the cRBM model. The purpose of these models is to determine the effect of adjusting the hidden layer bias value when initializing all of the FFN parameters.

5.2.5.1 Effect of initializing with pretrained W, U, d, and $\bar{\mathbf{c}}$ parameters Using the altered bias values, $\bar{\mathbf{c}}$, for the p12bN models shows the most impact for the Initialized FFN models. Both Tables 5.7 and 5.8 illustrate these results. For the results in Table 5.7 and comparing the Initialized p12N models with the Initialized p12bN models, adjusting the bias parameters begins to show the most improvement starting at $\alpha = 0.4$ when the μ values decrease. While the σ values do not generally decrease, the average range of the percent error decreases for the Initialized p12bN models.

For the results in Table 5.8, comparing the Initialized p12N models with the Initialized p12bN models shows that the adjusted $\bar{\mathbf{c}}$ parameters cause the average range of the percent error to be less than that of the Initialized p12N models until $\alpha = 0.5$. Beyond $\alpha = 0.5$, the average ranges of percent error begin to overlap between the Initialized p12N and Initialized p12bN models. The difference in the range of α for the lowest average percent error values for the p12bN models may also be explained by the connection between the amount of training time and the type of model as explained earlier [Lasserre et al., 2006]. The best α hyerparameter will need to be determined for each model and each dataset.

5.2.6 Effect of pretraining on the number of backpropagation epochs

Another important observation from these results is the number of epochs required to train randomly initialized FFN models versus pretrained FFN models. These results are provided in Tables 5.9 and 5.10.

Table 5.9: Avg train Ep: 1000 Ep cRBMs

Model	**r12N**	**Pretrained**
p1N_1	28.090 ± 5.812	16.050 ± 0.833
p1N_2	28.090 ± 5.812	26.410 ± 6.811
p12N	28.248 ± 5.601	16.691 ± 2.118
p12bN	28.248 ± 5.601	16.986 ± 1.412

Table 5.10: Avg train Ep: 2000 Ep cRBMs

Model	**r12N**	**Pretrained**
p1N_1	27.550 ± 4.236	17.120 ± 1.281
p1N_2	27.550 ± 4.236	18.910 ± 2.104
p12N	28.090 ± 5.312	16.708 ± 1.870
p12bN	28.090 ± 5.312	16.527 ± 1.140

The results in Tables 5.9 and 5.10 illustrate that, generally, fewer backpropagation training epochs are required when starting from a pretrained model than when starting from a randomly initialized model. The only exception is in Table 5.9, for p1N_2. In this case, the slight decrease in the average number of training epochs is offset by the increase in the average standard deviation of training epochs.

Comparing the p1N_1 model with the p12N and p12bN models shows that roughly the same number of training epochs is required. Interestingly, only the randomly initialized classification parameters are trained with the p1N_1 model, as compared to the full sets of, albeit pretrained, model parameters with the p12N and p12bN models. So fewer parameters are trained with the p1N_1 model than with the p12N or p12bN models. The p1N_1 models did not perform as well on the test data as the p12N and p12bN models, which illustrates the significance of pretraining the classification model parameters even though a comparable number of training epochs are required for all types of pretrained FFNs.

5.2.7 Effect of pretraining on the training error

Tables 5.11 and 5.12 show the results of the randomly initialized and pretrained FFNs on the training data. The FFN models in Table 5.11 are initialized with RBMs trained for 1000 epochs, and the FFN models in Table 5.12 are initialized with RBMs trained for 2000 epochs.

Table 5.11: Training percent error: 1000 epochs cRBMs

Pretrained Model	**Baseline (r12N)**	**Trained Phase**
p1N_1	0.210 ± 0.655	7.702 ± 1.509
p1N_2	0.210 ± 0.655	0.024 ± 0.242
p12N	0.128 ± 0.372	0.130 ± 0.265
p12bN	0.128 ± 0.372	0.014 ± 0.060

Table 5.12: Training percent error: 2000 epochs cRBMs

Pretrained Model	**Baseline (r12N)**	**Trained Phase**
p1N_1	0.194 ± 0.539	2.040 ± 0.518
p1N_2	0.194 ± 0.539	0.073 ± 0.430
p12N	0.242 ± 0.657	0.007 ± 0.036
p12bN	0.242 ± 0.657	0.003 ± 0.020

These results show that the standard deviation value usually decreases for pretrained models. The only exception is the entry for p1N_1 in Table 5.11. The p1N_2 model, however, shows a decrease in σ from 0.655 to 0.242. Similarly for the average results in both tables, the increase in μ occurs for the p1N_1 model only. Pretraining decreases μ for p1N_2 in both cases. The p12N models have roughly the same average training error as the randomly initialized FFNs in Table 5.11, but decreased average values in Table 5.12. Therefore, for pretrained models, where all parameters are trained with backpropagation, the μ and σ values generally decrease on this training data as compared to models initialized randomly. The p12bN model greatly reduces the error for both Tables 5.11 and 5.12.

Obtaining improvements on both the training set and the test set indicates increased generalization capabilities, since training set improvements did not come at the cost of test set improvements [Erhan et al., 2010]. In a study of pretraining by [Erhan et al., 2010], it was found that pretraining seems to provide similar benefits to other regularization methods such as L_1 or L_2 regularization and early stopping. For these tests, only early-stopping was applied to the FFN models, and the same early-stopping conditions were applied to the randomly initialized models and to the pretrained models. No regularization was used during any RBM model training. [Erhan et al., 2010] suggests that this pretraining regularization helps to determine a region in the cost function space that provides better generalization capabilities for the trained FFN. This region of parameter space, selected using the training data, provides more appropriate model parameter values than random initialization [Erhan et al., 2010].

Overall, the p12bN model performed the best or tied for the best in calculating the model error on either the training or the test datasets. Comparing both the average of averages and average of standard deviation values between the randomly initialized FFN and the p12bN models shows that the p12bN model overall, requires fewer backpropagation training epochs. Lastly, between the p12N and p12bN models, the p12bN model had a slightly lower average standard deviation of the number of required training epochs.

6.0 CONCLUSIONS

This thesis provided a new approach to initializing the parameters of a discriminative FFN using the trained parameters of a cRBM. The nonlinear optimization problem of finding a minimum value, in the surface of a FFN's complex, high-dimensional error space, is influenced by selecting the initialization location. To obtain optimal performance on unseen data, it is desired to find the most appropriate minimum in the error surface as possible. Therefore, initializing FFNs is a critical step that results in trained networks with different parameters and decision-making abilities.

Trained models, initialized with pretrained parameters, have been shown to improve a FFN's abilities on both the training data and on test datasets. However, previous pretraining methods still relied on random initializations for the last layer of parameters, which are critical for the network's abilities to make decisions. The FFN training methods must compensate for reducing the difference imposed by the randomization. In addition, these methods may not be as effective if there is a mismatch between the clusters found in the data by the generative model and the known classes as defined in the training data. In that case, the information provided by the pretrained parameters may not be as useful for improving the model's overall knowledge of the data space.

This proposed method used a cRBM structure that incorporates the class information and provided initializations for all of the FFN parameters, which was shown to be beneficial, especially in reducing the standard deviation of model error. A unique aspect of this approach was in altering the hidden layer bias parameters to compensate for the differences in cRBM and FFN structure when adapting the cRBM parameters to the FFN. This alteration was been shown to provide meaningful parameters to the network through lower classification test errors when evaluating the network before training, although this improvement depended in part on the number of pretraining epochs used and the proportion of generative and discriminative hybrid training. Regardless, it was illustrated that the hidden layer bias adjustment is capable of providing an initialized model that achieved a lower error range than the corresponding model without the bias adjustment. Models with and without the bias adjustment showed that backpropagation training reduced the standard deviation of network errors from those of the randomly initialized networks. Disadvantages of this proposed pretraining approach, as with many pretraining methods, include the necessity of two training phases. In general, techniques that can reduce the standard deviation of the test error are promising.

Future work should include tests of these techniques using larger datasets. While these tested pretraining methods were shown to provide some error reductions, it was difficult to compare exactly the effect of bias compensation on the trained FFNs initialized with cRBMs since both the p12N and p12bN models achieved zero error after backpropagation. Other studies could include more tests that vary the number of training epochs for the pretraining models to evaluate the effect of this with the range of α [Lasserre et al., 2006].

The abilities of these pretraining techniques to extend to larger models should be evaluated. Suggested extensions of this method to larger models are described in Section 6.1. Comparisons of these models to aDBNs could provide insight into the effect of hybrid training when applied at different points in the overall training procedure. With these methods, hybrid training is used to determine FFN initialization parameters, and with aDBNs, similar hybrid techniques are applied as a fine-tuning technique for the generative models [Mohamed et al., 2009], [Hinton, 2007].

6.1 PROPOSED GENERALIZED MODELS

The pretrained FFN models evaluated in these experiments only had one hidden layer. However, the initialization methods described in Section 4.3 can be extended to models with multiple layers. All pretraining models would have their parameters initialized randomly. Each FFN initialized by a pretrained model would have the same number of hidden layers and the same number of nodes per layer as the pretrained model. The bias vector $\mathbf{b}$ is discarded when initializing FFNs.

The first type of model has the most similarities with the aDBN described in Section 4.2.2.2. GLP would be performed using nRBMs to build the base of the model until pretraining the second-to-last hidden layer. Then, a cRBM would be added on the top of the pretrained model. The cRBM layer could be trained for any value of α, used to weight the contributions of generative and discriminative training effects. Pretraining of the completed model ends upon completion of cRBM training. At this point, two sub-models could be obtained where the last hidden layer bias vector is initialized with $\mathbf{c}$ or with $\bar{\mathbf{c}}$. As in Chapter 4, the adjusted $\bar{\mathbf{c}}$ vector compensates for the loss of label nodes as inputs to the last hidden layer when comparing between the cRBM and FFN. The other parameters $\mathbf{W}$, $\mathbf{U}$, and $\mathbf{d}$ are inserted directly into the FFN. After initialization, the FFN is trained with backpropagation.

The second type of model uses the idea of stacking RBMs as in the GLP. Instead of nRBMs, cRBMs will be applied to all phases of GLP. After a single cRBM is pretrained, the hidden layer biases of the FFN will be initialized either using $\mathbf{c}$ or with $\bar{\mathbf{c}}$. The weight matrix $\mathbf{W}$ from the cRBM is kept, but the weight matrix $\mathbf{U}$ and the bias vector $\mathbf{d}$ are discarded. From the first cRBM, the hidden layer nodes become the next cRBM visible layer nodes, and the process is repeated until the model reaches its second-to-last hidden layer. When the final cRBM is added to the model and pretrained, all of its parameters are kept. Again, the final hidden layer bias parameters are chosen to be either $\mathbf{c}$ or $\bar{\mathbf{c}}$. All parameters from the pretrained model are used to initialize the FFN model, and then backpropagation training is performed.

It should be noted that, for this second type of model extension, the α value chosen for cRBM pretraining could be kept the same while pretraining all layers. This would ensure that all model initialization parameters receive the same amounts of discriminative and generative training during pretraining. However, varying the α value from more generative to more discriminative values as the pretrained model is constructed from the first cRBM to the last cRBM may make a difference. While the model is closer to the raw data input to the first cRBM, a more generative pretraining approach may capture more of the data variability. Stepping the α values to become more discriminative as the number of cRBM layers increases may help to transform the features extracted from the raw data toward those that provide more use in classification. This may further improve the process of determining a relevant location in parameter space for the FFN.

In addition, each cRBM layer of the pretrained model does not have to stay consistent with choosing hidden layer biases $\mathbf{c}$ or $\overline{\mathbf{c}}$. However, as it is believed that the $\overline{\mathbf{c}}$ parameters maintain the benefits of model pretraining during FFN initialization, configurations will be chosen to use only these values. In comparing the two proposed models, it is expected that the second, using cRBMs for each layer, will provide improvements over deeper FFNs initialized with mostly nRBMs and one cRBM for the label layer. Incorporating the label information at each hidden layer may help to maintain the link between the data and labels as more complex features are created with each hidden layer added.

APPENDIX

EQUATION DERIVATIONS

A.1 DERIVATION OF GENERALIZED EXPONENTIAL FORMS

These derivations are taken from [Jebara]. The first proof shows how to derive the general exponential equation when starting from a Bernoulli distribution. The second is the same proof when beginning with a Gaussian distribution.

A.1.1 Bernoulli

The general equation for a Bernoulli PDF follows using a generic variable x_i.

$$p\left(x_i\right) = \alpha_i^{x_i}\left(1-\alpha_i\right)^{1-x_i} \tag{A.1}$$

An identity step is used by taking the exponent of a natural log.

$$p\left(x_i\right) = \exp\left[\ln\left(\alpha_i^{x_i}\left(1-\alpha_i\right)^{1-x_i}\right)\right] \tag{A.2}$$

Log properties regarding raised powers and products are applied.

$$p\left(x_i\right) = \exp\left[x_i \ln\left(\alpha_i\right) + \left(1-x_i\right)\ln\left(1-\alpha_i\right)\right] \tag{A.3}$$

The second log function is distributed among the $(1 - x_i)$ terms.

$$p(x_i) = \exp\left[x_i \ln(\alpha_i) + \ln(1 - \alpha_i) - x_i \ln(1 - \alpha_i)\right] \tag{A.4}$$

Common log terms multiplied by x_i are gathered together.

$$p(x_i) = \exp\left[x_i \ln\left(\frac{\alpha_i}{1 - \alpha_i}\right) + \ln(1 - \alpha_i)\right] \tag{A.5}$$

Set $\boldsymbol{\theta_i} = \theta_i = \ln\left(\frac{\alpha_i}{1-\alpha_i}\right)$, and apply another identity by adding $\ln(1) = 0$.

$$p(x_i) = \exp\left[x_i\theta_i + \ln(1 - \alpha_i) - \ln(1)\right] \tag{A.6}$$

Combine the last two log terms.

$$p(x_i) = \exp\left[x_i\theta_i - \ln\left(\frac{1}{1 - \alpha_i}\right)\right] \tag{A.7}$$

Apply the identity of adding $\alpha_i - \alpha_i = 0$ to the numerator.

$$p(x_i) = \exp\left[x_i\theta_i - \ln\left(\frac{1 - \alpha_i + \alpha_i}{1 - \alpha_i}\right)\right] \tag{A.8}$$

Simplify the fraction in the second log term.

$$p(x_i) = \exp\left[x_i\theta_i - \ln\left(1 + \frac{\alpha_i}{1 - \alpha_i}\right)\right] \tag{A.9}$$

Apply inverse operations of natural log and exponent to the fraction in the second log term.

$$p(x_i) = \exp\left[x_i\theta_i - \ln\left(1 + \exp\left(\ln\left(\frac{\alpha_i}{1 - \alpha_i}\right)\right)\right)\right] \tag{A.10}$$

Substitute θ_i in the log term to obtain the general exponential equation.

$$p(x_i) = \exp\left[x_i\theta_i - \ln(1 + \exp(\theta_i))\right] \tag{A.11}$$

Pull the exponential family parameters from the general exponential family PDF equation.

$$\begin{aligned} & r_i\left(x_i\right) = 1 \\ & f_a\left(x_i\right) = x_i \\ & \boldsymbol{\theta}_i = \ln\left(\tfrac{\alpha_i}{1-\alpha_i}\right) \\ & A_i\left(\boldsymbol{\theta}_i\right) = \ln\left(1 + e^{\boldsymbol{\theta_i}}\right) \end{aligned} \tag{A.12}$$

A.1.2 Gaussian

The general equation for a Gaussian PDF follows using a generic variable x_i.

$$p\left(x_i\right) = \frac{1}{\sqrt{2\pi\sigma_i^2}} \exp\left(-\frac{\left(x_i - \mu_i\right)^2}{2\sigma_i^2}\right) \tag{A.13}$$

Simplify to bring the σ_i parameter to the numerator.

$$p\left(x_i\right) = \frac{\sigma_i^{-1}}{\sqrt{2\pi}} \exp\left(-\frac{\left(x_i - \mu_i\right)^2}{2\sigma_i^2}\right) \tag{A.14}$$

Apply an identity using the exponent and natural log functions, excluding the constant $\frac{1}{\sqrt{2\pi}}$.

$$p\left(x_i\right) = \frac{1}{\sqrt{2\pi}} \exp\left(\ln\left[\sigma_i^{-1} \exp\left(-\frac{\left(x_i - \mu_i\right)^2}{2\sigma_i^2}\right)\right]\right) \tag{A.15}$$

Use log properties with exponents, products, and identities. Expand the squared terms.

$$p\left(x_i\right) = \frac{1}{\sqrt{2\pi}} \exp\left(-\ln\left(\sigma_i\right) - \frac{x_i^2}{2\sigma_i^2} + \frac{x_i\mu_i}{\sigma_i^2} - \frac{\mu_i^2}{2\sigma_i^2}\right) \tag{A.16}$$

Reorganize the terms.

$$p\left(x_i\right) = \frac{1}{\sqrt{2\pi}} \exp\left(\frac{x_i\mu_i}{\sigma_i^2} - \frac{x_i^2}{2\sigma_i^2} - \frac{\mu_i^2}{2\sigma_i^2} - \ln\left(\sigma_i\right)\right) \tag{A.17}$$

Pull out terms corresponding to the positions of known parameters from the general exponential equation.

$$\begin{aligned}
&r_i\left(x_i\right) = \tfrac{1}{\sqrt{2\pi}} \\
&f_a\left(x_i\right) = \left[x_i, x_i^2\right] \\
&\boldsymbol{\theta}_i = \left[\tfrac{\mu_i}{\sigma_i^2}, \tfrac{-1}{2\sigma_i^2}\right] \\
&A_i\left(\boldsymbol{\theta}_i\right) = \tfrac{\mu_i^2}{2\sigma_i^2} + \ln\left(\sigma_i\right)
\end{aligned} \tag{A.18}$$

A.2 GAUSSIAN-BERNOULLI ENERGY EQUATION

In the Gaussian-Bernoulli RBM, each of the visible nodes is represented by a Gaussian PDF, while the hidden nodes apply the Bernoulli PDF. In this case, the visible nodes must represent values outside of the set of $\{0, 1\}$, while the hidden nodes can still perform clustering to binary states $\{0, 1\}$ [Welling et al.].

Chapter 2 of the thesis by [Manoharan, 2015] provides a summary of the evolution of the energy function for Gaussian-Bernoulli RBMs. In this section, the Gaussian-Bernoulli energy equation derivation applies the parameters and sufficient statistics from the Gaussian and Bernoulli distributions to the method used in constructing the overall PDF for RBMs [Welling et al.], [Jebara].

The parameters $\boldsymbol{\theta_i}$ and sufficient statistics $\mathbf{f}\left(v_i\right)$ for the Gaussian distribution are listed.

$$\begin{aligned}
&\boldsymbol{\theta_i} = \left[\tfrac{\mu_i}{\sigma_i^2}, \tfrac{-1}{2\sigma_i^2}\right] \\
&\mathbf{f}\left(v_i\right) = \left[v_i, v_i^2\right]
\end{aligned} \tag{A.19}$$

Below are the parameters $\boldsymbol{\lambda_j}$ and sufficient statistics $\mathbf{g}\left(h_j\right)$ for the Bernoulli distribution.

$$\begin{aligned}
&\boldsymbol{\lambda_j} = \ln\left(\tfrac{\beta_j}{1-\beta_j}\right) = c_j \\
&\mathbf{g}\left(h_j\right) = h_j
\end{aligned} \tag{A.20}$$

In the Bernoulli-Bernoulli RBM, it was shown that the node bias b_i equals $\ln\left(\frac{\alpha_i}{1-\alpha_i}\right)$. This is applied again. The derivation begins with the joint PDF for the visible and hidden nodes for an RBM that uses exponential family distributions.

$$p(\mathbf{v},\mathbf{h}) \propto \exp\left[\sum_{i,a}\theta_{ia}f_{ia}(v_i) + \sum_{j,b}\lambda_{jb}g_{jb}(h_j) + \sum_{i,j,a,b}W_{ia}^{jb}f_a(v_i)\,g_b(h_j)\right] \tag{A.21}$$

The distributions' parameters and sufficient statistics are plugged into this joint PDF equation. It is assumed that the weight matrix $\mathbf{W}$ expands into the vector $\left[\frac{W'_{ij}}{\sigma_i^2}, 0\right]$ where the set of weights, W'_{ij}, are the weights connecting the visible and hidden layers.

$$p(\mathbf{v},\mathbf{h}) \propto \exp\left[\sum_i\left(\frac{\mu_i}{\sigma_i^2}v_i + \frac{-1}{2\sigma_i^2}v_i^2\right) + \sum_j(c_jh_j) + \sum_i\sum_j\left(\frac{W'_{ij}}{\sigma_i^2}v_ih_j\right)\right] \tag{A.22}$$

An identity is performed by adding $\frac{\mu_i^2}{2\sigma_i^2} - \frac{\mu_i^2}{2\sigma_i^2} = 0$. Numerator terms are shifted around also.

$$\begin{aligned}&p(\mathbf{v},\mathbf{h}) \propto \\ &\exp\left[\sum_i\left(\frac{\mu_iv_i}{\sigma_i^2} - \frac{v_i^2}{2\sigma_i^2} + \frac{\mu_i^2}{2\sigma_i^2} - \frac{\mu_i^2}{2\sigma_i^2}\right) + \sum_j(c_jh_j) + \sum_i\sum_j\left(W'_{ij}\frac{v_i}{\sigma_i^2}h_j\right)\right]\end{aligned} \tag{A.23}$$

Completion of the square is performed for the terms that sum over i. The $-\frac{1}{2}$ is pulled from the same terms.

$$p(\mathbf{v},\mathbf{h}) \propto \exp\left[\sum_i\left(\frac{-1}{2}\left(\frac{v_i-\mu_i}{\sigma_i}\right)^2 + \frac{\mu_i^2}{2\sigma_i^2}\right) + \sum_j(c_jh_j) + \sum_i\sum_j\left(W'_{ij}\frac{v_i}{\sigma_i^2}h_j\right)\right] \tag{A.24}$$

The property for the product of exponential terms is applied, and $\exp\left(\sum_i\frac{\mu_i^2}{2\sigma_i^2}\right)$ is separated from the other terms.

$$\begin{aligned}&p(\mathbf{v},\mathbf{h}) \propto \\ &\exp\left(\sum_i\frac{\mu_i^2}{2\sigma_i^2}\right)\exp\left[\sum_i\frac{-1}{2}\left(\frac{v_i-\mu_i}{\sigma_i}\right)^2 + \sum_j(c_jh_j) + \sum_i\sum_j\left(W'_{ij}\frac{v_i}{\sigma_i^2}h_j\right)\right]\end{aligned} \tag{A.25}$$

This separated term is a constant, and can be dropped out from the rest of the equation as long as the proportionality remains in place.

$$p\left(\mathbf{v},\mathbf{h}\right) \propto \exp\left[\sum_i \frac{-1}{2}\left(\frac{v_i-\mu_i}{\sigma_i}\right)^2 + \sum_j \left(c_j h_j\right) + \sum_i \sum_j \left(W'_{ij}\frac{v_i}{\sigma_i^2}h_j\right)\right] \tag{A.26}$$

By definition of the Boltzmann distribution, the energy equation is the negative of the exponential term argument for the joint PDF.

$$E\left(\mathbf{v},\mathbf{h}\right) = \sum_i \frac{1}{2}\left(\frac{v_i-\mu_i}{\sigma_i}\right)^2 - \sum_j \left(c_j h_j\right) - \sum_i \sum_j \left(W'_{ij}\frac{v_i}{\sigma_i^2}h_j\right) \tag{A.27}$$

A.3 DERIVING $P(\mathbf{V}|\mathbf{H})$

The following derivation is the complement for that of Section 2.3.3.1. A Bernoulli-Bernoulli RBM is assumed. The equation for $p(\mathbf{v}|\mathbf{h})$ is derived beginning with the definition of conditional PDFs.

$$p\left(\mathbf{v}|\mathbf{h}\right) = \frac{p\left(\mathbf{v},\mathbf{h}\right)}{p\left(\mathbf{h}\right)} \tag{A.28}$$

Both the joint Boltzmann distribution and the marginal distribution for the hidden nodes are plugged into the right hand side.

$$p\left(\mathbf{v}|\mathbf{h}\right) = \frac{\frac{1}{Z}e^{-E(\mathbf{v},\mathbf{h})}}{\frac{1}{Z}\sum_{\mathbf{v}'} e^{-E(\mathbf{v}',\mathbf{h})}} \tag{A.29}$$

The vector energy equation of a Bernoulli-Bernoulli RBM is applied. As before, the partition function can be canceled.

$$p\left(\mathbf{v}|\mathbf{h}\right) = \frac{e^{-\left(-\mathbf{b}^T\mathbf{v}-\mathbf{c}^T\mathbf{h}-\mathbf{v}^T\mathbf{W}\mathbf{h}\right)}}{\sum_{\mathbf{v}'} e^{-\left(-\mathbf{b}^T\mathbf{v}'-\mathbf{c}^T\mathbf{h}-\mathbf{v}'^T\mathbf{W}\mathbf{h}\right)}} \tag{A.30}$$

Further simplifications result by switching the PDF notation from vector to scalar.

$$p\left(\mathbf{v}|\mathbf{h}\right) = \frac{\exp\left(\sum_{i=1}^{N} v_i b_i + \sum_{j=1}^{M} h_j c_j + \sum_{i=1}^{N} \sum_{j=1}^{j=M} W_{ij} v_i h_j\right)}{\sum_{\mathbf{v}'} \exp\left(\sum_{i=1}^{N} v_i' b_i + \sum_{j=1}^{M} h_j c_j + \sum_{i=1}^{N} \sum_{j=1}^{M} W_{ij} v_i' h_j\right)} \tag{A.31}$$

Common terms that can be canceled from the numerator and denominator are separated. With this derivation, the common terms relate to the hidden layer. Also, v_i is pulled out from the remaining terms.

$$p\left(\mathbf{v}|\mathbf{h}\right) = \frac{\exp\left(\sum_{j=1}^{M} h_j c_j\right) \exp\left(\sum_{i=1}^{N} v_i \left(b_i + \sum_{j=1}^{M} W_{ij} h_j\right)\right)}{\exp\left(\sum_{j=1}^{M} h_j c_j\right) \sum_{\mathbf{v}'} \exp\left(\sum_{i=1}^{N} v_i' \left(b_i + \sum_{j=1}^{M} W_{ij} h_j\right)\right)} \tag{A.32}$$

The properties of products of exponentials are applied.

$$p\left(\mathbf{v}|\mathbf{h}\right) = \frac{\prod_{i=1}^{N} \exp\left(v_i \left(b_i + \sum_{j=1}^{M} W_{ij} h_j\right)\right)}{\sum_{\mathbf{v}'} \prod_{i=1}^{N} \exp\left(v_i' \left(b_i + \sum_{j=1}^{M} W_{ij} h_j\right)\right)} \tag{A.33}$$

The vector sum $\sum_{\mathbf{v}'}$ is expanded over the N nodes to simplify the denominator.

$$p\left(\mathbf{v}|\mathbf{h}\right) = \frac{\prod_{i=1}^{N} \exp\left(v_i \left(b_i + \sum_{j=1}^{M} W_{ij} h_j\right)\right)}{\sum_{v_1'} \cdots \sum_{v_N'} \prod_{i=1}^{N} \exp\left(v_i' \left(b_i + \sum_{j=1}^{M} W_{ij} h_j\right)\right)} \tag{A.34}$$

As with the $p\left(\mathbf{h}|\mathbf{v}\right)$, the product is expanded also.

$$p\left(\mathbf{v}|\mathbf{h}\right) = \frac{\prod_{i=1}^{N} \exp\left(v_i \left(b_i + \sum_{j=1}^{M} W_{ij} h_j\right)\right)}{\sum_{v_1'} \cdots \sum_{v_N'} \left[\exp\left(v_1' \left(b_1 + \sum_{j=1}^{M} W_{1j} h_j\right)\right)\right] \cdots \left[\exp\left(v_N' \left(b_N + \sum_{j=1}^{M} W_{Nj} h_j\right)\right)\right]} \tag{A.35}$$

At this step, the values of the visible nodes can be treated as constants with respect to all but one of the sums. For example, when summing over v_1, this variable is constant with respect to v_2, v_3, and up through v_N. Since the exponential terms have already been separated into separate products, the products can be matched with their respective sum value.

$$p\left(\mathbf{v}|\mathbf{h}\right) = \frac{\prod_{i=1}^{N} \exp\left(v_i \left(b_i + \sum_{j=1}^{M} W_{ij} h_j\right)\right)}{\left[\sum_{v_1'} \exp\left(v_1' \left(b_1 + \sum_{j=1}^{M} W_{1j} h_j\right)\right)\right] \cdots \left[\sum_{v_N'} \exp\left(v_N' \left(b_N + \sum_{j=1}^{M} W_{Nj} h_j\right)\right)\right]} \tag{A.36}$$

The products are consolidated in the denominator.

$$p\left(\mathbf{v}|\mathbf{h}\right) = \frac{\prod_{i=1}^{N} \exp\left(v_i\left(b_i + \sum_{j=1}^{M} W_{ij}h_j\right)\right)}{\prod_{i=1}^{N}\left[\sum_{v_i'} \exp\left(v_i'\left(b_i + \sum_{j=1}^{M} W_{ij}h_j\right)\right)\right]} \tag{A.37}$$

The product over j, common to the numerator and denominator, is pulled from both parts.

$$p\left(\mathbf{v}|\mathbf{h}\right) = \prod_{i=1}^{N} \frac{\exp\left(v_i\left(b_i + \sum_{j=1}^{M} W_{ij}h_j\right)\right)}{\sum_{v_i'} \exp\left(v_i'\left(b_i + \sum_{j=1}^{M} W_{ij}h_j\right)\right)} \tag{A.38}$$

This step sums over the accepted values for v_i which are $\{0, 1\}$.

$$p\left(\mathbf{v}|\mathbf{h}\right) = \prod_{i=1}^{N} \frac{\exp\left(v_i\left(b_i + \sum_{j=1}^{M} W_{ij}h_j\right)\right)}{\exp\left((0)\left(b_i + \sum_{j=1}^{M} W_{ij}h_j\right)\right) + \exp\left((1)\left(b_i + \sum_{j=1}^{M} W_{ij}h_j\right)\right)} \tag{A.39}$$

Set $p\left(v_i\right) = \frac{\exp\left(v_i\left(b_i+\sum_{j=1}^{M} W_{ij}h_j\right)\right)}{1+\exp\left(b_i+\sum_{j=1}^{M} W_{ij}h_j\right)}$. Conditional independence of the visible nodes is achieved.

$$p\left(\mathbf{v}|\mathbf{h}\right) = \prod_{i=1}^{N} p\left(v_i|\mathbf{h}\right) \tag{A.40}$$

Again, the sigmoid function is derived when calculating $p\left(V_i = 1|\mathbf{h}\right)$.

$$p\left(V_i = 1|\mathbf{h}\right) = \frac{\exp\left(b_i + \sum_{j=1}^{M} W_{ij}h_j\right)}{1 + \exp\left(b_i + \sum_{j=1}^{M} W_{ij}h_j\right)} \tag{A.41}$$

Multiply both the numerator and denominator by a common term.

$$p\left(V_i = 1|\mathbf{h}\right) = \frac{\exp\left(b_i + \sum_{j=1}^{M} W_{ij}h_j\right)}{1 + \exp\left(b_i + \sum_{j=1}^{M} W_{ij}h_j\right)}\left[\frac{\exp\left(-\left(b_i + \sum_{j=1}^{M} W_{ij}h_j\right)\right)}{\exp\left(-\left(b_i + \sum_{j=1}^{M} W_{ij}h_j\right)\right)}\right] \tag{A.42}$$

Simplify using the identity property.

$$p\left(V_i = 1|\mathbf{h}\right) = \frac{1}{\exp\left(-\left(b_i + \sum_{j=1}^{M} W_{ij}h_j\right)\right) + 1} \tag{A.43}$$

Apply the definition of the sigmoid function.

$$p\left(V_i = 1|\mathbf{h}\right) = \sigma\left(b_i + \sum_{j=1}^{M} W_{ij} h_j\right) \tag{A.44}$$

Equation A.45 is given for completeness.

$$p\left(V_i = 0|\mathbf{h}\right) = \frac{1}{1 + \exp\left(b_i + \sum_{j=1}^{M} W_{ij} h_j\right)} = 1 - p\left(V_i = 1|\mathbf{h}\right) \tag{A.45}$$

A.4 INTEGRAL SEPARATION OF TERMS

The derivation for the separation of terms in the denominator for Equations A.33 through A.36 and 2.49 through 2.51 can be explained using integrals. This explanation helps to establish similarities of this characteristic of exponential distributions for binary and continuous vector values.

The PDF for a vector of variables, $\mathbf{h}$, is calculated over the integral of a vector of continuous-valued variables $\mathbf{v}$.

$$p\left(\mathbf{h}\right) = \int_{\mathbf{v}} p\left(\mathbf{v}, \mathbf{h}\right) d\mathbf{v} \tag{A.46}$$

The PDF is substituted for $p\left(\mathbf{v}, \mathbf{h}\right)$.

$$p\left(\mathbf{h}\right) = \frac{1}{Z} \int_{\mathbf{v}} e^{-\left(-\mathbf{b}^T\mathbf{v} - \mathbf{c}^T\mathbf{h} - \mathbf{v}^T\mathbf{W}\mathbf{h}\right)} d\mathbf{v} \tag{A.47}$$

The scalar version of the PDF equation is applied. The terms referring to the hidden nodes are separated, and the common factor for the visible nodes is pulled from the remaining terms.

$$p\left(\mathbf{h}\right) = \frac{1}{Z} \int_{\mathbf{v}} \exp\left(\sum_{j=1}^{M} h_j c_j\right) \exp\left(\sum_{i=1}^{N} v_i \left(b_i + \sum_{j=1}^{M} W_{ij} h_j\right)\right) d\mathbf{v} \tag{A.48}$$

Scalar values are pulled from the integral.

$$p(\mathbf{h}) = \frac{1}{Z} \exp\left(\sum_{j=1}^{M} h_j c_j\right) \int_{\mathbf{v}} \exp\left(\sum_{i=1}^{N} v_i \left(b_i + \sum_{j=1}^{M} W_{ij} h_j\right)\right) d\mathbf{v} \tag{A.49}$$

Properties of exponents are applied to turn sums of the argument into products of the terms.

$$p(\mathbf{h}) = \frac{1}{Z} \exp\left(\sum_{j=1}^{M} h_j c_j\right) \int_{\mathbf{v}} \prod_{i=1}^{N} \exp\left(v_i \left(b_i + \sum_{j=1}^{M} W_{ij} h_j\right)\right) d\mathbf{v} \tag{A.50}$$

The integral over the vector $\mathbf{v}$ is separated into individual scalar components.

$$\begin{aligned} &p(\mathbf{h}) = \tfrac{1}{Z} \exp\left(\textstyle\sum_{j=1}^{M} h_j c_j\right) \times \\ &\textstyle\int_{v_1} \cdots \int_{v_N} \prod_{i=1}^{N} \exp\left(v_i \left(b_i + \sum_{j=1}^{M} W_{ij} h_j\right)\right) dv_1 \cdots dv_N \end{aligned} \tag{A.51}$$

The individual product terms are separated.

$$\begin{aligned} &p(\mathbf{h}) = \tfrac{1}{Z} \exp\left(\textstyle\sum_{j=1}^{M} h_j c_j\right) \times \\ &\textstyle\int_{v_1} \cdots \int_{v_N} \left[\exp\left(v_1 \left(b_1 + \sum_{j=1}^{M} W_{1j} h_j\right)\right)\right] \times \cdots \times \\ &\left[\exp\left(v_N \left(b_N + \textstyle\sum_{j=1}^{M} W_{Nj} h_j\right)\right)\right] dv_1 \cdots dv_N \end{aligned} \tag{A.52}$$

Each of the factors that is not evaluated directly over v_i can be treated as a constant in the integral over v_i. This allows the integrals to be factored and separated.

$$\begin{aligned} &p(\mathbf{h}) = \tfrac{1}{Z} \exp\left(\textstyle\sum_{j=1}^{M} h_j c_j\right) \times \\ &\left[\textstyle\int_{v_1} \exp\left(v_1 \left(b_1 + \sum_{j=1}^{M} W_{1j} h_j\right)\right) dv_1\right] \times \cdots \times \\ &\left[\textstyle\int_{v_N} \exp\left(v_N \left(b_N + \sum_{j=1}^{M} W_{Nj} h_j\right)\right) dv_N\right] \end{aligned} \tag{A.53}$$

The product of terms is consolidated over i.

$$p(\mathbf{h}) = \frac{1}{Z} \exp\left(\sum_{j=1}^{M} h_j c_j\right) \prod_{i=1}^{N} \left[\int_{v_i} \exp\left(v_i \left(b_i + \sum_{j=1}^{M} W_{ij} h_j\right)\right) dv_i\right] \tag{A.54}$$

The product term in above equation is the continuous valued version of the denominator in Equations 2.49 through 2.53. Replacing the integrals with sums produces a similar result.

A.5 SIGMOID TO SOFTMAX

This section illustrates the relationship between the sigmoid function, used to classify two classes, and the softmax function, used for three or more classes. The output values are like probabilities, since each sigmoid or softmax output is in the range of $[0, 1]$. This derivation of the softmax function taken from [Hinton, 2010].

A generic variable x takes values $\{0, 1\}$ in a sigmoid function.

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{A.55}$$

Multiplying each term by e^x leads to the form of sigmoid that can be generalized to a softmax. The denominator, summed over $x = 0$ and $x = 1$, can be considered as a normalization term in the binary case.

$$\sigma(x) = \frac{e^x}{1 + e^x} \tag{A.56}$$

The final softmax equation is determined by normalizing over K possible values for x.

$$softmax(x) = \frac{e^x}{\sum_{i=1}^{K} 1 + e^{x_i}} \tag{A.57}$$

A.6 THEORETICAL GENERATIVE RBM DERIVATIONS

These equations are expanded from the derivations provided in [Fischer and Igel, 2012]. The calculations are an exact evaluation of the training equations, instead of substituting for the change in the energy function at the expectation approximation in Section 2.4.4.1.

With respect to the weights W_{ij} [Fischer and Igel, 2012]:

$$\frac{\partial \ln\left[p\left(\mathbf{v}\right)\right]}{\partial W_{ij}} = -\sum_{\mathbf{h}} p\left(\mathbf{h}|\mathbf{v}\right) \times \left[-v_i h_j\right] + \sum_{\mathbf{v}'}\sum_{\mathbf{h}'} p\left(\mathbf{v}', \mathbf{h}'\right) \times \left[-v_i h_j\right] \tag{A.58}$$

$$= \sum_{h_j} p\left(h_j|\mathbf{v}\right) \sum_{\mathbf{h}_{-j}} p\left(\mathbf{h}_{-j}|\mathbf{v}\right) \times \left[v_i h_j\right] - \sum_{\mathbf{v}'} p\left(\mathbf{v}'\right) \sum_{h_j} p\left(h_j|\mathbf{v}'\right) \sum_{\mathbf{h}_{-j}} p\left(\mathbf{h}_{-j}|\mathbf{v}'\right) \times \left[v_i h_j\right] \tag{A.59}$$

$$h_j \in \{0, 1\}$$

$$\textstyle\sum_{h_j} p\left(h_j|\mathbf{v}\right) = p\left(H_j = 0|\mathbf{v}\right) + p\left(H_j = 1|\mathbf{v}\right) = 1 \tag{A.60}$$

$$\textstyle\sum_{\mathbf{h}_{k=1;k\neq j}}^{M} p\left(\mathbf{h}_k|\mathbf{v}\right) = \sum_{h_1} p\left(h_1|\mathbf{v}\right) \cdots \sum_{h_M} p\left(h_M|\mathbf{v}\right) = 1$$

$$\frac{\partial \ln\left[p\left(\mathbf{v}\right)\right]}{\partial W_{ij}} = p\left(H_j = 1|\mathbf{v}\right) v_i - \sum_{\mathbf{v}'} p\left(\mathbf{v}'\right) p\left(H_j = 1|\mathbf{v}'\right) v_i \tag{A.61}$$

$$p\left(H_j = 1|\mathbf{v}\right) = \sigma\left(c_j + \sum_{i=1}^{N} W_{ij} v_i\right) \tag{A.62}$$

$$\frac{\partial \ln\left[p\left(\mathbf{v}\right)\right]}{\partial W_{ij}} = \sigma\left(c_j + \sum_{i=1}^{N} W_{ij} v_i\right) v_i - \sum_{\mathbf{v}'} p\left(\mathbf{v}'\right) \sigma\left(c_j + \sum_{i=1}^{N} W_{ij} v_i'\right) v_i \tag{A.63}$$

The derivations for the biases c_j, b_i use the same simplifications as with the weights W_{ij}. With respect to the hidden layer biases c_j [Fischer and Igel, 2012]:

$$\frac{\partial \ln\left[p\left(\mathbf{v}\right)\right]}{\partial c_j} = -\sum_{\mathbf{h}} p\left(\mathbf{h}|\mathbf{v}\right) \times \left[-h_j\right] + \sum_{\mathbf{v}'}\sum_{\mathbf{h}'} p\left(\mathbf{v}', \mathbf{h}'\right) \times \left[-h_j\right] \tag{A.64}$$

$$\frac{\partial \ln\left[p\left(\mathbf{v}\right)\right]}{\partial c_j} = \sum_{h_j} p\left(h_j|\mathbf{v}\right) h_j - \sum_{\mathbf{v}'} p\left(\mathbf{v}'\right) \sum_{h_j} p\left(h_j|\mathbf{v}\right) h_j \tag{A.65}$$

$$\frac{\partial \ln\left[p\left(\mathbf{v}\right)\right]}{\partial c_j} = \sigma\left(c_j + \sum_{i=1}^{N} W_{ij} v_i\right) - \sum_{\mathbf{v}'} p\left(\mathbf{v}'\right) \sigma\left(c_j + \sum_{i=1}^{N} W_{ij} v'_i\right) \tag{A.66}$$

With respect to the visible layer biases b_i [Fischer and Igel, 2012]:

$$\frac{\partial \ln\left[p\left(\mathbf{v}\right)\right]}{\partial b_i} = -\sum_{\mathbf{h}} p\left(\mathbf{h}|\mathbf{v}\right) \times \left[-v_i\right] + \sum_{\mathbf{v}'} \sum_{\mathbf{h}'} p\left(\mathbf{v}', \mathbf{h}'\right) \times \left[-v_i\right] \tag{A.67}$$

$$\frac{\partial \ln\left[p\left(\mathbf{v}\right)\right]}{\partial b_i} = \sum_{h_j} p\left(h_j|\mathbf{v}\right) v_i - \sum_{\mathbf{v}'} p\left(\mathbf{v}'\right) \sum_{h_j} p\left(h_j|\mathbf{v}'\right) v_i \tag{A.68}$$

$$\frac{\partial \ln\left[p\left(\mathbf{v}\right)\right]}{\partial b_i} = v_i - \sum_{\mathbf{v}'} p\left(\mathbf{v}'\right) v_i \tag{A.69}$$

BIBLIOGRAPHY

C. Bishop. Pattern recognition and machine learning, 2006.

D. M. Blei. Exponential families. 2011. URL `https://www.cs.princeton.edu/courses/archive/fall11/cos597C/lectures/exponential-families.pdf`.

E. Chen. Introduction to restricted boltzmann machines, July 18, 2011 2011. URL `http://blog.echen.me/2011/07/18/introduction-to-restricted-boltzmann-machines/`.

M. K. Chung. Introduction to random fields, July 8, 2016 2007. URL `http://www.stat.wisc.edu/~mchung/teaching/MIA/theories/randomfield.feb.02.2007.pdf`.

A. El-Jaroudi and J. Makhoul. A new error criterion for posterior probability estimation with neural nets. In *Neural Networks, 1990., 1990 IJCNN International Joint Conference on*, pages 185–192. IEEE, 1990.

E. Emberly. Topic 3: Probability theory and boltzmann distribution. URL `http://www.sfu.ca/~eemberly/phys347/lectures/3c_boltzmann_distn.pdf`.

D. Erhan, Y. Bengio, A. Courville, P.-A. Manzagol, P. Vincent, and S. Bengio. Why does unsupervised pre-training help deep learning? *Journal of Machine Learning Research*, 11 (Feb):625–660, 2010.

A. Fischer and C. Igel. *An introduction to restricted Boltzmann machines*, pages 14–36. Springer, 2012. ISBN 3642332749.

K. S. T. T. A.-r. M. Geoffrey Hinton, Nitish Srivastava. Lecture 12c: Restricted boltzmann machines, November 8, 2013 2013. URL `https://www.youtube.com/watch?v=tt-PQNstYp4`.

C. Geyer. Introduction to markov chain monte carlo. *Handbook of Markov Chain Monte Carlo*, pages 3–48, 2011.

G. Hinton. A practical guide to training restricted boltzmann machines. *Momentum*, 9(1): 926, 2010.

G. E. Hinton. To recognize shapes, first learn to generate images. *Progress in brain research*, 165:535–547, 2007.

T. Jebara. Lecture 12: The exponential family of distributions. URL http://www.cs.columbia.edu/~jebara/4771/tutorials/lecture12.pdf.

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

H. Larochelle, Y. Bengio, J. Louradour, and P. Lamblin. Exploring strategies for training deep neural networks. *The Journal of Machine Learning Research*, 10:1–40, 2009. ISSN 1532-4435.

H. Larochelle, M. Mandel, R. Pascanu, and Y. Bengio. Learning algorithms for the classification restricted boltzmann machine. *The Journal of Machine Learning Research*, 13(1): 643–669, 2012. ISSN 1532-4435.

J. A. Lasserre, C. M. Bishop, and T. P. Minka. Principled hybrids of generative and discriminative models. In *Computer Vision and Pattern Recognition, 2006 IEEE Computer Society Conference on*, volume 1, pages 87–94. IEEE, 2006.

S. Manoharan. *Gaussian discrete restricted Boltzmann machine: theory and its applications: a thesis presented in partial fulfilment of the requirements for the degree of Master of Engineering in Electronics and Computer Engineering at Massey University, Albany, New Zealand*. Thesis, 2015.

A.-r. Mohamed, G. Dahl, and G. Hinton. Deep belief networks for phone recognition. In *Nips workshop on deep learning for speech recognition and related applications*, volume 1, page 39, 2009.

J. Niemi. Gibbs sampling, March 3, 2013 2013. URL https://www.youtube.com/watch?v=a_08GKWHFWo.

W. S. Sarle. Subject: Should i normalize/standardize/rescale the data?, August 26, 2016 2002. URL http://www.faqs.org/faqs/ai-faq/neural-nets/part2/.

C. Shalizi. *Logistic Regression*, book section 12. 2012.

J. R. Shewchuk et al. An introduction to the conjugate gradient method without the agonizing pain, 1994.

S. Theodoridis and K. Koutroumbas. Pattern recognition. 2008.

M. Welling, M. Rosen-Zvi, and G. E. Hinton. Exponential family harmoniums with an application to information retrieval. In *Advances in neural information processing systems*, pages 1481–1488.