Analyzing Unsupervised Representation Learning Models Under the View of Dynamical Systems

by

Daniel Jiwoong Im

A thesis submitted in conformity with the requirements
for the degree of Master of Applied Science
School of Engineering
University of Guelph

Abstract

Analyzing Unsupervised Representation Learning Models Under the View of Dynamical Systems

Daniel Jiwoong Im
Master of Applied Science
School of Engineering
University of Guelph, 2015

The focus of this thesis is on unsupervised learning algorithms, particularly from the perspective of representation learning. The two main unsupervised learning methods that we work with are restricted Boltzmann machines and auto-encoders. Both algorithms are acknowledged to capture meaningful representations of the data, in which they learn to represent the higher-level abstractions of the data. In this thesis, the background on unsupervised learning and the two algorithms are thoroughly discussed using details of mathematical formulations.

The objective of this thesis is to take the dynamical systems approach to understand and develop the unsupervised learning models and learning algorithms through the lens of dynamical systems.

Auto-encoders are perhaps the best-known non-probabilistic methods for representation learning. They are conceptually simple and easy to train. Recent theoretical work has shed light on their ability to capture manifold structure, and drawn connections to density modeling. This has motivated researchers to seek ways of auto-encoder scoring, which has furthered their use in classification. Gated auto-encoders (GAEs) are an interesting and flexible extension of auto-encoders which can learn transformations among different images or pixel covariances within images. However, they have been much less studied, theoretically or empirically. For the first part of our study, we examine the GAEs's ability to represent different functions or data distributions. We apply a dynamical systems view to GAEs, deriving a scoring function, and drawing connections to RBMs. On a set of deep learning benchmarks, we also demonstrate their effectiveness for single and multi-label classification.

Energy-based models are popular in machine learning due to the elegance of their formulation and their relationship to statistical physics. Among these, the restricted Boltzmann machine (RBM), and its staple training algorithm contrastive divergence (CD), have been the prototype for some recent advancements in the unsupervised training of deep neural networks. However, CD has limited theoretical motivation, and can, in some cases, produce undesirable behavior. In the second part of our study, we investigate the performance of Minimum Probability Flow (MPF) learning for training RBMs. Unlike CD, with its focus on approximating an intractable partition function via Gibbs sampling, MPF proposes a tractable, consistent, objective function defined in terms of a Taylor expansion of the KL divergence with respect to sampling dynamics. We propose a more general form for the sampling dynamics in MPF, and explore the consequences of different choices for these dynamics for training RBMs. Experimental results show MPF outperforming CD for various RBM configurations.

Contents

Acknowledgments

I would like to thank my supervisor Dr. Graham Taylor. Graham is an excellent supervisor. Comparing myself in September 2013, which was when I first started the Master's program at University of Guelph, to now, I gained tremenduous knowledge in machine learning and wisdom as a researcher. This is all thanks to Graham.

I also would like to thank my thesis committee member, Dr. Medhat Moussa. Professor Moussa and I had numerous meetings since I first joined University of Guelph, and he always gave me fruitful advices.

I wish to thank all my colleagues and friends in the Machine learning lab at University of Guelph. Jan Rudy, Ammar Abuleil, Ethan Buchman and I started our Master's program at the same time and they gave me great support. I also want to thank Gavin Ding, Tomas Sixta, and Carolyn Augusta for many careful proofreads over my papers and provided valuable suggestions. I would like to acknowledge Dinesh Ramdhayan, who is the leader of University of Toronto Go Club. He proofread chapter 2 of this thesis.

Lastly, I would like to thank my parents, my grandmother and sister for all their support and love.

Publications

Publications that are related to this thesis

Daniel Jiwoong Im, and Graham W. Taylor, Scoring and Classifying with Gated Auto-encoders, *http://arxiv.org/pdf/1412.6610v3.pdf*.
Under review at European Conference of Machine Learning (ECML)

Daniel Jiwoong Im, Ethan Buchman, and Graham W. Taylor, An Empirical Investigation of Minimum Probability Flow Learning Under Different Connectivity Patterns.
Under review at European Conference of Machine Learning (ECML)

Daniel Jiwoong Im, Ethan Buchman, and Graham W. Taylor, Understanding Minimum Probability Flow for RBMs Under Various Kinds of Dynamics, *http://arxiv.org/pdf/1412.6617v4.pdf*.
International Conference of Learning Representations (ICLR) workshop

Daniel Jiwoong Im, and Graham W. Taylor, Analyzing the Dynamics of Gated Auto-encoders, In Neural Information Processing Systems Deep Learning Workshop. 2014

Publications that are not related to this thesis

Daniel Jiwoong Im, and Graham W. Taylor, Learning a metric for class-conditional KNN
Under review at International Joint Conference on Artifical Intelligence (IJCAI)

Daniel Jiwoong Im, and Graham W. Taylor, Semi-supervised Hyperspectral Image Classification via Neighborhood Graph Learning
IEEE Geoscience and Remote Sensing Letters (GSL)

Daniel Jiwoong Im, and Graham W. Taylor Improving semi-supervised neural networks for scene understanding by learning the neighborhood graph In Computer Vision and Pattern Recognition Scene Understanding Workshop. 2014

Notation

Scalar quantities are denoted by lower-case letters, such as x. Vector quantities are denoted by bold lower-case letters, such as $\boldsymbol{x}$. Matrices are denoted by upper-case letters, such as W.

Individual elements of vectors and matrices use subscripts, such as x_i and W_{ij}.

Occasionally we use the same letter to denote different, but related, vectors or matrices. We use superscripts to distinguish them, as in $W^{\boldsymbol{v}}$ and $W^{\boldsymbol{h}}$.

$\mathcal{N}(\boldsymbol{\mu}, \Sigma)$ is a Gaussian distribution with mean $\boldsymbol{\mu}$ and covariance Σ and $\mathbb{E}_{p^*}$ is an expectation with respect to distribution p^*. We also use the shorthand notation $\langle \cdot \rangle_{p^*}$ to denote an expectation.

1 Introduction

Computers and humans have very opposite strengths. Computers are precise, accurate, and fast at computing arithmatic and logic, whereas humans can understand and perceive things much better than computers can. For example, a computer can approximate π upto 12.1 trillion digits. On the other hand, humans naturally recognize the perceivable objects and generalize newly seen objects. The goal of machine learning is to solve problems by recognizing patterns and extracting meaningful features from the data based on studying learning mechanisms as humans naturally learn to perceive and recognize things.

After the 2012 breakthrough of deep convolutional neural networks on the Imagenet competition (Krizhevsky et al., 2012), the popularity of deep neural networks has suddenly given rise to many other successes. For example, winning the competitions on predicting the activity of potential drugs or predicting job salaries from job advertisements demonstrated the success of neural networks.

Artificial neural network (ANN) is one of the branches in machine learning that is inspired from the architecture of the brain. ANNs are composed of interconnected artificial neurons called "units", which serve as model neurons. The function of the synapse in the brain is modeled by the modifiable weights, which are associated with each connection (Rumelhart et al., 1986; Hinton, 1992). Typically, machine learning researchers abuse the name and call it simply "the neural network", and we will follow the same convention throughout the thesis.

The neural network itself has been proven to be "a universal approximator"

(Csáji, 2001). In other words, with enough data and the artificial neurons, it can model any kind of continuous function. In fact, we can add more units by adding more layers of artificial neurons to a neural network. This has been empirically shown to be effective and powerful (Bengio et al., 2013a). Nevertheless, the crucial component of the algorithms is in the process of learning. The learning processes are usually viewed as a 'black art' to non-scientists. However, the black art is basically the organization of several layers of artificial neurons in order for such networks to train from the data by being able to recognize patterns, features, and coherence in the data. Machine learning scientists dubbed the name "deep learning" to the idea described above, but more specifically, to the idea of modelling high-level abstractions of data using multiple layers of artificial neurons.

Humans constantly have to process enormous inputs that are obtained through our sensory receptors, and one of the processes is learning (Spitzer, 2006). Humans can either learn from someone's guidance or by themselves. However, human brains spend most of the time learning on our own without the awareness of a learning process. Akin to human learning, machines can be taught supervisedly and unsupervisedly. In fact, many machine learning researchers, including myself, believe that machines can learn much more efficiently through unsupervised learning if we manage to find the proper unsupervised learning mechanisms. This is because humans obtain a lot more unlabeled data than labeled data[†] and derive many insights and structures from those data. As an example, Google brain has experimented with a massive amount of Youtube data with huge processing power, and shown that the network learned to perceive what a lot of Google users perceived (Le et al., 2011).

Two current well-known unsupervised learning algorithms are restricted Boltzmann machines and auto-encoders. Both algorithms are acknowledged to capture meaningful representations of the data, in which they learn the higher-level abstractions of the data that are more meaningful than the original data. For example, representing the image by abstract concepts, such as sets of edges, texture, and shapes, is much more meaningful than pixel intensities.

[†]Unlabeled data refers to data that are obtained with no guidance and labeled data refers to the data that are guided. For example, the photo of an apple with the label "apple" is labeled data and without label "apple" would be unlabeled data

Also, having a good representation makes the given task easier to accomplish. This is first demonstrated by Hinton et al. (2006) through a technique called "unsupervised pre-training". The philosophy behind "unsupervised pre-training" is that the network learns to figure out the structures and patterns from the data itself and the human only gets involved in the latter stage to label the outputs.

The focus of this thesis is on unsupervised learning algorithms, particularly within the subject of representation learning. In this thesis, the background on unsupervised learning and the two algorithms are thoroughly discussed using details of mathematical formulations. The background includes works related to my research and are rudimentary to understanding the work we present. These are presented in chapter 2.

The objective of this thesis is to take the dynamical systems approach to understand and develop the unsupervised learning models and learning algorithms through the lens of dynamical systems.

In chapter 3, we analyze the dynamics of various types of auto-encoders, such as gated auto-encoders and mean-covariance auto-encoders. We demonstrate that these auto-encoders can be scored similarly to classical auto-encoders by computing the potential energy required for running their trajectories. We also reveal the relationnship between auto-encoders and restricted Boltzmann machines through the scope of its potential energy function. Finally, we present an elegant technique to leverage the scoring of families of auto-encoders for multi-labeled classification.

In chapter 4, we expound on minimum probability flow learning under various dynamical systems and leverage it to train restricted Boltzmann machines. The beauty of this algorithm is that it does not require computing or approximating the partition function on any energy-based model. The minimum probability flow learning algorithm requires a dynamical system, which guarantees that the model will converge to its stationary states after evolving over some amount of time. We derive a more general form of the transition matrix that satisfies detailed balance. We also experiment with various dynamics such as single bit flip, factored MPF, and persistent MPF on restricted Boltzmann machines.

More concisely, chapter 3 construe the models' abilities to represent different functions or data distributions; while, chapter 3 explores the learning algorithm which allows the model to approximate or capture the data distributions. Both aspects are very important since we need to understand what the model is capable of modelling, and we need to make sure we have the ability to make full use of models' capabilities by exploring various learning strategies. Additionally, despite the fact that we took a dynamical systems approach to chapter 3 and chapter 4, the dynamical systems in two chapters are quite different from one and another. In chapter 3, we consider the dynamics induced by repeatedly applying the reconstruction function of an autoencoder. The dynamics are characterized by a vector field which represents the linear transformation that occurs when the reconstruction function is applied to data. On the other hand, in chapter 3, we consider the dynamics induced by a Markov Chain which transitions among the states of a generative model. The dynamics are characterized by probability flow in and out of these states. Therefore, the main contribution of this thesis are understanding higher-order auto-encoders in terms of their potential energy functions, and exploring the alternative learning method for RBMs as opposed to the contrastive divergence learning procedure.

2 Background & Related work

Unsupervised learning is a type of machine learning approach that tries to reveal the hidden structure of data sets. Unsupervised learning only uses unlabeled data points to learn a function $f : \mathcal{X} \rightarrow \mathcal{H}$ that maps $\mathbf{x}$ to a new representation $\mathbf{h}$. This contrasts with supervised learning, which takes labeled data points $(\mathbf{x}, \mathbf{y})$ to model a function $f(\mathbf{x}) = \mathbf{y}$ or a probability distribution $p(\mathbf{x}, \mathbf{y})$ where $\mathbf{x}$ is the data and $\mathbf{y}$ is the label. There are several aims of unsupervised learning as the term "hidden structure" can cover a number of broad aspects. For example, one of the goals of unsupervised learning is to find sensible clusters of the data points. Another goal could be to provide a compact, low-dimensional representation of the input, or to find an economical high-dimensional representation such as binary features. As this thesis focuses on deep learning algorithms, the aim of unsupervised learning from a deep learning perspective is to build an internal representation of the input that is useful for subsequent supervised or reinforcement learning approaches. This is often called *unsupervised feature learning* (Bengio et al., 2013a).

Researchers have advocated undsupervised learning for another learning system. Most famously, Hinton et al. (2006) introduced a greedy layerwise unsupervised learning algorithm. The idea behind this greedy layer-by-layer training procedure is to learn hierarchical features one level at a time using an unsupervised learning model called a Restricted Boltzmann Machine (RBM). The training procedure attempts to learn new features[†] using RBMs based on

[†]We use the term features and representations interchangeably.

the features learned by an RBM from the previous layer. The greedy layer-by-layer training procedure is shown in Figure 2.1. This resulting architecture is called Deep Belief Network (DBN).

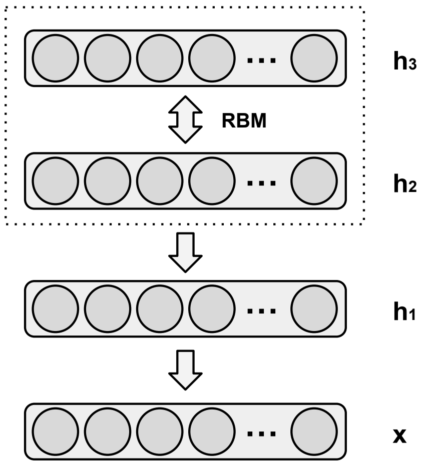

Figure 2.1. Greedy Layer-by-layer training using the restricted Boltzmann machine is shown. This is also known as 3-layer Deep Belief Network

This idea of employing a unsupervised learning algorithm to pre-train features before applying supervised training has been championed by a large number of published works (Hinton et al., 2006; Bengio et al., 2007; Hinton, 2007; Vincent et al., 2008; Dahl et al., 2010; Mohamed and Hinton, 2010). Since then, many of the new unsupervised learning algorithms have been developed by deep learning researchers such as Denoising Auto-encoders, Constractive Auto-encoders, and deep Boltzmann machines (Vincent et al., 2008; Rifai et al., 2011; Salakhutdinov and Hinton, 2009; Bengio et al., 2014).

In general, unsupervised learning algorithms from a deep learning perspective can be segregated into deterministic and probabilistic models. Deterministic models have outcomes that are precisely determined by the data. Thus, the function is many-to-one. On the other hand, probabilistic models have distributions over the the possible outcomes, so they can represent uncertainty. In this chapter, we will go over the models that are good examples of deterministic and probabilistic models.

2.1 Auto-encoders

The most successful and well-known example of a deterministic unsupervised learning method is an auto-encoder. They are conceptually simple and easy to train through the back-propagation algorithm. The encoder of the auto-encoder learns an efficient representations of the data and the decoder learns to reconstruct the original data. Typically, the encoding function is a non-linear function such that

$$\mathbf{h} = f(\mathbf{x}) = f(W\mathbf{x} + \mathbf{b}) \tag{2.1}$$

and the decoding function can be also non-linear but we will focus linear decoding function in this thesis unless the data is binary,

$$\tilde{\mathbf{x}} = g(\mathbf{h}) = R\mathbf{h} + \mathbf{c} \tag{2.2}$$

where the function f is a non-linear function such as a sigmoid function or rectified linear function, and $\theta = \{W, R, \mathbf{b}, \mathbf{c}\}$ are the parameters of the model. In the case of binary data, the decoding function uses a sigmoid function to limit/bound the output to 0 and 1. Additionally, the weights of the encoder and decoder are often tied such that $R = W^T$. For convenience, we will present the operation of the auto-encoder as a single reconstruction function,

$$r(\mathbf{x}) = g(f(\mathbf{x})) = \tilde{\mathbf{x}}.$$

The architecture of the auto-encoder is depicted in Figure 2.2

We can easily extend the auto-encoder to multiple layers by stacking the hidden layers. This is known as a deep auto-encoder. The encoder and decoder are then expressed as

$$h^{(l)} = f(\mathbf{h}^{(l-1)}) = f(W^{(l)}\mathbf{h}^{(l-1)} + \mathbf{b}^{(l)}) \tag{2.3}$$

$$g^{(k)} = f(\mathbf{h}^{(k-1)}) = f(R^{(k)}\mathbf{h}^{(k-1)} + \mathbf{c}^{(k)}) \tag{2.4}$$

where l and k are the indices for layer l and layer k[†], and the first layer refers to the data such that $\mathbf{h}^{(0)} = \mathbf{x}$. Throughout the paper, we will describe the 1-layer version of the auto-encoder, but they can be generalized to deeper layers of an auto-encoder.

[†]Often in practice, the decoder is constructed so that it mirrors the encoder.

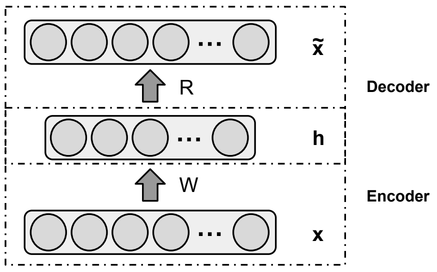

Figure 2.2. The auto-encoder is presented as a feedforward network graph. The encoder and decoder mirror each other with respect to the hidden layer h in the network.

Depending on the type of the data, whether it is binary or real values, the error function becomes either the mean squared or cross entropy error function,

$$L(\mathbf{x}, \tilde{\mathbf{x}}) = \sum_{j=0}^{M} (x_j - \tilde{x}_j)^2 \tag{2.5}$$

$$L(\mathbf{x}, \tilde{\mathbf{x}}) = \sum_{j=0}^{M} x_j \log \tilde{x}_j + (1 - x_j) \log \tilde{x}_j \tag{2.6}$$

where M is the dimensionality of the data. In practice, we use an expected squared error function

$$\mathbb{E}\left[L(x, \tilde{x})\right] = \frac{1}{N} \sum_{i}^{N} L(\mathbf{x}_i, \tilde{\mathbf{x}}_\mathbf{i}) \tag{2.7}$$

where N is the number of the data points. This is more convenient to compare the error values between different models.

Various regularized variants of auto-encoders have been proposed (Vincent et al., 2008; Rifai et al., 2011) as well as theoretical insights into their operations (Swersky et al., 2011; Vincent, 2010; Guillaume and Bengio, 2013). Regularized auto-encoders play a prominent role in generalization for many different reasons. One of the essential reasons is that regularization helps to capture the manifold structure of data and models the data density distribution. Another reason is to achieve an *over-complete* representation so that the representation is more robust against the presence of noise in the data.

There are a number of proposals for regularized auto-encoders. Two well-known examples are the denoising and contractive auto-encoders. The denoising auto-encoder corrupts the input by adding noise, for example, from a Bernoulli or Gaussian distribution, and attempts to reconstruct the original data. The objective function of a denoising auto-encoder is expressed as

$$L_{DAE} = \mathbb{E}[\|r(\mathbf{z}) - \mathbf{x}\|^2] \tag{2.8}$$

where $\mathbf{z} = \mathbf{x} + \epsilon$ is the corrupted input and ϵ is the noise.

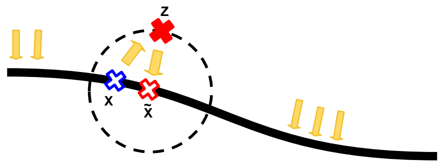

Figure 2.3. This figure demonstrates the idea behind the denoising criterion. $\mathbf{x}$ is the original data that lies on a manifold. $\mathbf{z}$ is the corrupted input by adding Gaussian noise and $\tilde{\mathbf{x}}$ is projected back to the data manifold by trying to reconstruct the original data $\mathbf{x}$. The yellow arrows indicate the direction towards which the auto-encoder tries to reconstruct. This figure is best viewed in colour.

The intuition of the denoising criterion is to be robust near the data distribution by learning to compensate for corruption in the data. Figure 2.3 delineates the procedure and the fundamental idea of the denoising criterion. We assume that the data lies in a low-dimensional non-linear manifold that is embedded in a higher-dimensional space. In Figure 2.3, $\mathbf{x}$ is the original data that lies a on data manifold. $\mathbf{z}$ is the input corrupted by adding a Gaussian noise and $\tilde{\mathbf{x}}$ is projected back to the data manifold by trying to reconstruct the original data $\mathbf{x}$.

Contractive auto-encoders were introduced after denoising auto-encoder. The objective function of a contractive auto-encoder is explicitly defined as

$$L_{CAE} = \mathbb{E}[\|r(\mathbf{x}) - \mathbf{x}\|]^2] + \lambda\mathbb{E}[\|J_f\mathbf{x}\|^2] \tag{2.9}$$

where $\|J_f(\mathbf{x})\|^2$ is a Jacobian matrix with the squared Frobenium norm (Vincent et al., 2008) and λ is the postive hyperparameter that controls the strength of the contractive penalty term. Instead of corrupting the input to contract the surface of the data manifold, the contractive auto-encoder penalizes the tangent of the manifold to be flat at the data points. The Jacobian of the encoding function f in the contractive term is expressed as $\frac{\partial f(\mathbf{x})}{\partial \mathbf{x}}$. Then, the contractive term can be rewritten as

$$L_{CAE} = \mathbb{E}[\|r(\mathbf{x}) - \mathbf{x}\|]^2] + \lambda \| \frac{\partial f(\mathbf{x})}{\partial \mathbf{x}} \|. \tag{2.10}$$

The cost function of denoising auto-encoders has a very similar form compared to that of the contractive auto-encoders. The objective function of a denoising auto-encoder in Equation 2.8 can be re-expressed as

$$L_{DAE} = \mathbb{E}[\|r(\mathbf{x}) - \mathbf{x}\|^2] + \sigma^2 \mathbb{E}[\| \frac{\partial r(\mathbf{x})}{\partial \mathbf{x}} \|^2] + o(\sigma^2) \tag{2.11}$$

as $\sigma \to 0$ where $\mathbf{z} = \mathbf{x} + \boldsymbol{\epsilon}$ is the corrupted input and $\boldsymbol{\epsilon} \sim \mathcal{N}(0, \sigma^2)$ is noise that is independently drawn from the Gaussian distribution with variance of σ^2 (Guillaume and Bengio, 2013). The derivation is presented in Appendix A.1. This provides us with the ability to re-interpret the denoising criterion in terms of a contracting penality term. Notice that the difference between the regularization term of the denoising and contractive auto-encoder is that one regularizes the derivative of the reconstruction function $r(\mathbf{x})$ with the variance σ^2, while the other one regularizes the derivative of the encoding funtion $f(\mathbf{x})$ with the penalty cost as the hyper-parameter λ. Also, the denoising auto-encoder is implicitly contracting by learning to be robust from the corrupted data, while the contractive auto-encoder is explicitly contracting by penalizing the derivative of the encoding function. However, both regularizers train the models to be sensitive to the direction where the density of the data is more concentrated and less sensitive to lower density region in the manifold.

Interestingly, the regularized auto-encoders have a special relationship to energy-based models. For example, the denoising auto-encoder training criterion can be seen as an approximation of the regularized score matching algorithm (Vincent, 2010). Furthermore, the estimate of parameters of an energy-based model under the score matching criterion is a particular form of regularized auto-encoders (Swersky et al., 2011).

2.2 Restricted Boltzmann Machines

Often in machine learning, we start with the assumption that there exists a distribution that explains the observed variables. In statistical machine learning settings, our goal is to model the true distribution of the data samples we observe. Hence, we say that "what our model believes in ought to capture the data samples that came from the data distribution". Energy-based models such as the Markov random fields and RBMs are some of the most popular machine learning tools due to the expressiveness in their structure and formulation, and their relationship to statistical physics.

While we discuss unsupervised probabilistic graphical models, we will use the restricted Boltzmann machine (RBM) as a canonical example. An RBM is an undirected bipartite graph[†] with visible (observed) variables $\mathbf{v} \in \{0,1\}^D$ and hidden (latent) variables $\mathbf{h} \in \{0,1\}^H$ (Smolensky, 1986). The RBM is an energy-based model where the energy of state $\mathbf{v}, \mathbf{h}$ is given by

$$E(\mathbf{v}, \mathbf{h}; \theta) = -\sum_i \sum_j W_{ij} v_i h_j - \sum_i b_i v_i - \sum_j c_j h_j \tag{2.12}$$

where $\theta = \{W, \mathbf{b}, \mathbf{c}\}$ are the parameters of the model. The marginalized probability over visible variables is formulated from the Boltzmann distribution,

$$p(\mathbf{v}; \theta) = \frac{p^*(\mathbf{v}; \theta)}{Z(\theta)} = \frac{1}{Z(\theta)} \sum_{\mathbf{h}} \exp\left(\frac{-1}{\tau} E(\mathbf{v}, \mathbf{h}; \theta)\right) \tag{2.13}$$

such that $Z(\theta) = \sum_{\mathbf{v},\mathbf{h}} \exp\left(\frac{-1}{\tau} E(\mathbf{v}, \mathbf{h}; \theta)\right)$ is a normalizing constant and τ is the thermodynamic temperature. We can marginalize over the binary hidden states in Equation 2.12 and re-express in terms of a new energy $F(\mathbf{v})$,

$$F(\mathbf{v}; \theta) = -\log \sum_{\mathbf{h}} \exp\left(\frac{-1}{\tau} E(\mathbf{v}, \mathbf{h})\right) \tag{2.14}$$

$$= \frac{1}{\tau} \sum_i^D v_i b_i - \frac{1}{\tau} \sum_{j=1}^H \log\left(1 + \exp\left(c_j + \sum_i^D v_i W_{ij}\right)\right) \tag{2.15}$$

$$p(\mathbf{v}; \theta) = \frac{\exp\big(-F(\mathbf{v}; \theta)\big)}{Z(\theta)} \tag{2.16}$$

Following physics, this form of the energy is better known as a free energy, as it expresses the difference between the average energy and the entropy of a distribution; in this case, that of $p(\mathbf{h}|\mathbf{v})$. The derivation is presented in Appendix A.2. Defining the distribution $p(\mathbf{v}; \theta)$ in terms of free energy is convenient since it naturally copes with the presence of latent variables.

[†]Energy-based models have an associated network graph that delineates the configuration of the states. That is, the relationship of each variable to other variables is represented through weights on edges of the network graph.

The key characteristic of an RBM is the simplicity of inference due to conditional independence between visible and hidden states:

$$p(\mathbf{h}|\mathbf{v}) = \prod_j p(h_j|\mathbf{v}), \qquad p(h_j = 1|\mathbf{v}) = \sigma(\sum_i W_{ij}v_i + c_j)$$
$$p(\mathbf{v}|\mathbf{h}) = \prod_i p(v_i|\mathbf{h}), \qquad p(v_i = 1|\mathbf{h}) = \sigma(\sum_j W_{ij}h_j + b_i)$$

where $\sigma(\cdot) = 1/(1 + \exp(-\cdot))$.

2.2.1 Gaussian-Bernoulli Restricted Boltzmann Machines

Previously, we defined restricted Boltzmann machines with the Bernoulli distribution over visible and hidden states, and they are meant model binary data. Here, we describe Gaussian-Bernoulli restricted Boltzmann machines, which has the Gaussian distribution over the visible states and the Bernoulli distribution over the hidden states and they can model real value data such as the pixel intensities of images.

Suppose that we have the similar settings as in previous section, where hidden (latent) variables are binary variables, $\mathbf{h} \in \{0,1\}^H$ but visible variables are real values, $\mathbf{v} \in \mathbb{R}^D$. We can define an energy function as

$$E(\mathbf{v},\mathbf{h};\theta) = -\sum_i \sum_j \frac{W_{ij}v_i h_j}{\sigma_i^2} - \sum_i \frac{(v_i - b_i)^2}{\sigma_i^2} - \sum_j c_j h_j \tag{2.17}$$

where $\theta = \{W, b_i, c_j, \sigma_i\}$ are the paramaters of the model. Then the marginalized probability over visible variables remains the same as Equation 2.13 with the energy function Equation 2.17. Moreover, the general form of the free energy of Gaussian-Bernoulli RBM remains the same as Equation 2.14, but expanding the general form becomes

$$F(\mathbf{v};\theta) = -\log \sum_{\mathbf{h}} \exp\left(\frac{-1}{\tau} E(\mathbf{v},\mathbf{h})\right) \tag{2.18}$$
$$= -\frac{1}{\tau}\sum_i^D \frac{(v_i - b_i)^2}{\sigma^2} - \frac{1}{\tau}\sum_{j=1}^H \log\left(1 + \exp\left(c_j + \sum_i^D \frac{v_i W_{ij}}{\sigma^2}\right)\right) \tag{2.19}$$

One of the reason why the energy function, Equation 2.17, is defined the way it is with addition of σ is that the conditional probability over visible states given hidden states becomes Gaussian distribution,

$$p(\mathbf{v}|\mathbf{h}) = \prod_i p(v_i|\mathbf{h}) = \mathcal{N}(\mathbf{v}; \mathbf{b} + W\mathbf{h}, \boldsymbol{\sigma}^2) \tag{2.20}$$

$$p(v_i|\mathbf{h}) = \mathcal{N}(v_i; b_i + W_{ij}h_j, \sigma^2) \tag{2.21}$$

where $\mathcal{N}(\cdot)$ is the Gaussian distribution with the standard deviation of $\boldsymbol{\sigma}^2$ for visible states. The conditional probability over hidden states given visible states remain same as sigmoid with the denominator of σ,

$$p(\mathbf{h}|\mathbf{v}) = \prod_j p(h_j|\mathbf{v}), \qquad p(h_j = 1|\mathbf{v}) = \sigma(\sum_i \frac{W_{ij}v_i}{\sigma^2} + c_j) \tag{2.22}$$

where $\sigma(\cdot) = 1/(1 + \exp(-\cdot))$.

2.2.2 Training Restricted Boltzmann Machines

As we defined the energy function in Equation 2.12, RBMs are energy-based models and the probability of each configuration is inversely proportional to the (scalar) energy. Therefore, the lower the energy for a given state configuration, the higher the probability that network will be found in that state. As a result of its elegancy, energy-based models provide a unified and definite framework for machine learning practitioners to work with.

One popular way to learn an appropriate energy function is to maximize the likelihood of the data samples with respect to the parameters. Generally, we posit that making the model agree with the data samples will give a good estimate of the true data distribution given large amount of data. The maximum likelihood technique is a classical method of parameter estimation, with many convenient and useful asymptotic guarantees. For energy-based models, we typically minimize the negative log-likihood with repsect to the parameters. For exponential models such as RBMs where $p(x)$ is proportional to the exponential of a negative potential function $F(x)$, the gradient of the data negative log-likelihood takes the form

$$\nabla_\theta = \sum_{x \in \mathcal{D}} \frac{\partial F(x)}{\partial \theta} - \sum_x p(x) \frac{\partial F(x)}{\partial \theta} \tag{2.23}$$

where the sum in the first term is over the dataset, $\mathcal{D}$, and the sum in the second term is over the entire domain of x. The first term has the effect of pushing the parameters in a direction that decreases the energy surface of the model at the training data points, while the second term increases the energy of all possible states.

There is one nasty problem that occurs when performing maximum likelihood learning with the energy-based model. There exists a normalizing constant term, also known as the partition function, which sums over all possible configuration of the states. As we can observe the second term in Equation 2.23, computing the partition function is intractable unless the number of all possible states is small enough, hence we try to approximate the partition function using Markov chain Monte Carlo (MCMC)[†].

The name "restricted Boltzmann machine" comes from the bipartite graph that makes visible states conditionally independent from the hidden states. The conditionally independent distributions $p(\mathbf{v}|\mathbf{h})$ and $p(\mathbf{h}|\mathbf{v})$ naturally lead to a block Gibbs sampling procedure, which is a special case of a MCMC algorithm. Block Gibbs sampling works by groups of two or more variables getting sampled and conditioned on all other group of variables. Thus, we sample $\mathbf{v}|\mathbf{h}$ and $\mathbf{h}|\mathbf{v}$. After running the block Gibbs Chain for n times, we will get $\mathbf{h}_0|\mathbf{v}_0 \rightarrow \mathbf{v}_1|\mathbf{h}_0 \rightarrow \cdots \rightarrow \mathbf{h}_{n-1}|\mathbf{v}_{n-1} \rightarrow \mathbf{v}_n|\mathbf{h}_{n-1}$.

In order to compute a good estimate of the partition function, we need to run the block Gibbs chain for a very long time. Of course, this is cumbersome in practice. Hinton (2002) introduced *the contrastive divergence* learning algorithm, where we only run the Gibbs chain for one time step such that $\mathbf{h}_0|\mathbf{v}_0 \rightarrow \mathbf{v}_1|\mathbf{h}_0 \rightarrow \mathbf{h}_1|\mathbf{v}_1$. This is a heuristic algorithm that approximates the infinite Gibbs chain since we are not using the statistics at the equilibrium states. In the beginning of the learning, the initial weights are already the bad parameters, hence, running the Gibbs chain for very long time is a waste of computation. The reason why *the constrastive divergence* learning algorithm works is that even after only a few steps of the chain, we get the sense of which direction the model is wandering around; thus lowering the probability of the regions that the model wanders away is a good enough heuristic.

We can formalize this by writing the learning updates for CD-k as follows

$$\Delta\theta_{CD-k} \propto -\sum_{j\in\mathcal{D}}\sum_{i\notin\mathcal{D}}\Big(\frac{\partial F_j(\theta)}{\partial\theta} - \frac{\partial F_i(\theta)}{\partial\theta}\Big)T_{ij} \tag{2.24}$$

[†]MCMC approaches to true samples based on the Ergodic Theorem for Markov Chain. The Ergodic Theorem states that if the Markov Chain is an irreducible (time-homogeneous) discrete MC with stationary distribution π, then $\frac{1}{n}\sum^n f(x_i) \rightarrow E[f(x)]$ as $n \rightarrow \infty$.

where T_{ij} is the probability of transitioning from state j to state i in k steps of block Gibbs sampling. We can in principle replace T_{ij} by any other transition operator, so long as it preserves the equilibrium distribution. Indeed, this is what alternative methods, like Persistent CD (Tieleman and Hinton, 2009), achieve.

2.3 Higher-order Generative Models

Learning to merely model data using deterministic or probabilistic models is not good enough for modeling the real world. This is because the world is made up of many entities and the relationships between them. We can observe the interactions of the entities everywhere. In order to model several entities and their relations, we can assign multiple variables to each entity and have them interact with each other to model their relations. We refer to models with three or more sets of interacting variables as higher-order models.

One natural way of formulating higher-order models is to have multiplicative interactions between the variables. Multiplicative interactions are useful for higher-order geneartive models because we can multiply two or more variables and through the parameters modulate the strength of the interactions. Time series data, such as various styles of human motion, are good examples for the use of multiplicative interaction models (Taylor and Hinton, 2009). As well as, learning the depth from pair images (Konda and Memisevic, 2013) and modeling various transformations of two images (Susskind et al., 2011) demonstrate the good use of multiplicative interactions. In this section, we introduce third-order interaction models that originated from auto-encoders and restricted Boltzmann machines.

2.3.1 Gated Auto-encoders

Similar to the classical auto-encoder, the gated auto-encoder (GAE) consists of an encoder $h(\cdot)$ and decoder $r(\cdot)$. While the standard auto-encoder processes a datapoint $\mathbf{x}$, the GAE processes input-output pairs $(\mathbf{x}, \mathbf{y})$. The GAE is usually trained to reconstruct $\mathbf{y}$ given $\mathbf{x}$, though it can also be trained symmetrically, that is, to reconstruct both $\mathbf{y}$ from $\mathbf{x}$ and $\mathbf{x}$ from $\mathbf{y}$. Intuitively, the GAE learns *relations* between the inputs rather than representations of the inputs themselves[†]. If $\mathbf{x} \neq \mathbf{y}$ (e.g. sequential frames of a video), then the mapping units $\mathbf{h}$ learn *transformations*. In the case that $\mathbf{x} = \mathbf{y}$ (i.e. the input is copied), then the mapping units learn pixel covariances. These models are also known as relational auto-encoders.

In the simplest form of the GAE, the M hidden units are given by a basis expansion of $\mathbf{x}$ and $\mathbf{y}$. The hidden unit is then defined as

$$h_k(\mathbf{x}, \mathbf{y}) = \sigma(\sum_i \sum_j W_{ijk} x_i y_j) \tag{2.25}$$

and the output unit is defined as

$$r(y_j|\mathbf{x}, \mathbf{h}) = \sum_i \sum_k W_{ijk} x_i h_k \tag{2.26}$$

where W is the weight parameter of the gated auto-encoder. The weight matrix is a $M \times D \times D$ tensor matrix, and the pair of variables, $\mathbf{x}$ and $\mathbf{y}$, interacts with the weight matrix to tune the third variable $\mathbf{h}$. However, this leads to a parameterization that it is at least quadratic in the number of inputs[‡] and thus, prohibitively large. Therefore, in practice, $\mathbf{x}$, $\mathbf{y}$, and $\mathbf{h}$ are projected onto matrices or ("latent factors"), W^X, W^Y, and W^H, respectively. Figure 2.4 exhibits the architecture of non-factored and factored GAEs.

The number of factors, F, must be the same for X, Y, and H. Thus, the model is completely parameterized by $\theta = \{W^X, W^Y, W^H\}$ such that W^X and W^Y are $F \times D$ matrices (assuming both $\mathbf{x}$ and $\mathbf{y}$ are D-dimensional) and W^H is an $M \times F$ matrix. The encoder function is defined by

$$h(\mathbf{x}, \mathbf{y}) = \sigma(W^H((W^X\mathbf{x}) \odot (W^Y\mathbf{y}))) \tag{2.27}$$

where $\odot$ is element-wise multiplication and $\sigma(\cdot)$ is an activation function. The decoder function is defined by

$$r(\mathbf{x}|\mathbf{y}, \mathbf{h}) = (W^X)^T((W^Y\mathbf{y}) \odot (W^H)^T h(\mathbf{x}, \mathbf{y})). \tag{2.28}$$

Note that the parameters are usually shared between the encoder and decoder. The choice of whether to apply a non-linearity to the output and the

[†]Relational features can be mixed with standard features by simply adding connections that are not gated.

[‡]Equation 2.25 and Equation 2.26 have double summations that make the parameterization grow in quadratic with repsect to the number of inputs.

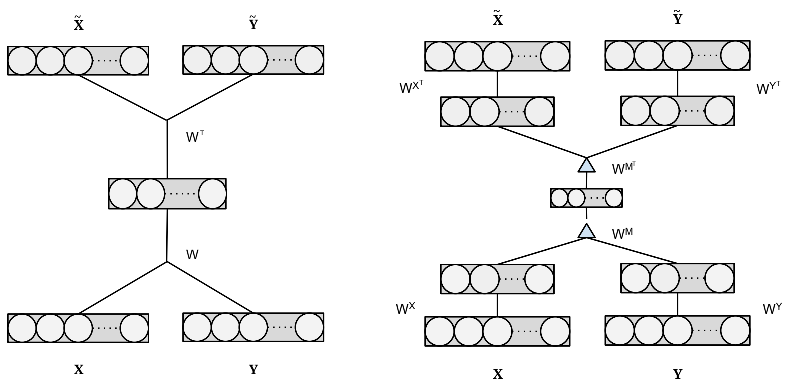

Figure 2.4. The architecture of the non-factored GAE (on the left) and factored GAE (on the right) with tied weights between the encoder and decoder.

specific form of objective function will depend on the nature of the inputs, for example, binary, categorical, or real-valued. Here, we have assumed real-valued inputs for simplicity of presentation. Therefore, Equation 2.28 is a bi-linear function of **h** and we use a squared-error objective:

$$L_{GAE} = \frac{1}{2}\|r(\mathbf{y}|\mathbf{x}) - \mathbf{y}\|^2. \qquad (2.29)$$

We can also constrain the GAE to be a symmetric model by training it to reconstruct both **x** given **y** and **y** given **x** (Memisevic, 2011):

$$L_{GAE} = \frac{1}{2}\|r(\mathbf{y}|\mathbf{x}) - \mathbf{y}\|^2 + \frac{1}{2}\|r(\mathbf{x}|\mathbf{y}) - \mathbf{x}\|^2. \qquad (2.30)$$

The symmetric objective can be thought of as the non-probabilistic analogue of modelling a *joint* distribution over **x** and **y** as opposed to a conditional distribution (Memisevic, 2011).

2.3.2 Mean Covariance Auto-encoders

In section 2.3.1, we saw that the auto-encoder learns to model the pixel intensities. As well, we can view the denoising auto-encoder as learning to model the mean of the pixel intensities based on the corrupted distribution $q(\mathbf{z}|\mathbf{x})$,

$$\mathbb{E}_{q(\mathbf{z}|\mathbf{x})}[\|r(\mathbf{z}) - \mathbf{x}\|^2] = \int \|r(\mathbf{z}) - \mathbf{x}\|^2 q(\mathbf{z}|\mathbf{x}) d\mathbf{z}.$$

This allows us to call the denoising auto-encoder as the *mean* auto-encoder.

Additionally, the GAE learns to model the *relations* in the pixel intensities given the two inputs, $\mathbf{x}$ and $\mathbf{y}$. Now, we shall consider the case when $\mathbf{x}$ and $\mathbf{y}$ are the same input, i.e. $\mathbf{y} = \mathbf{x}$. Based on the definition of GAE, they must model the *relations* between the pixels in $\mathbf{x}$ and $\mathbf{y}$, where $\mathbf{y}$ is $\mathbf{x}$, then this leads to modeling the correlations within the pixels of $\mathbf{x}$. Indeed, in order to fully learn the covariance pattern, the GAEs need to learn from the same image but with the noise in the data[†]. Hence, the GAEs with the setting where $\mathbf{x} = \mathbf{y}$ are called the *covariance* auto-encoder.

This naturally leads to combining the *mean* and *covariance* auto-encoders, which lead to the *mean-covariance* auto-encoders (mcAE) (Memisevic, 2011). The objective function of the *mean-covariance* auto-encoder is expressed as

$$\begin{aligned} L_{MCAE} &= L_{DAE} + L_{DGAE} \\ &= \|r(\mathbf{z}) - \mathbf{x}\|^2 + \|r(\mathbf{z}|\mathbf{x}) - \mathbf{x}\|^2 \end{aligned}$$

where L_{DGAE} is the mean squared error function of the denoised gated auto-encoders with setting of $\mathbf{x} = \mathbf{y}$ and the tied weights between the encoder and decoder, and also the tied weights between the factored layers. When working with the mcAE in practice, the data are often centered at zero and whitened as well. This gets rid of the first and second-order statistics of the data.

2.3.3 Factored Gated Restricted Boltzmann Machines

RBMs are generative models that are precisely trained to model the data distribution $p(\mathbf{x})$. We shall now explain higher-order generative models that have multiplicative interactions. More specifically, we will expound probabilistic energy-based models that have three-way interactions.

We can define the gated RBM as

$$E(\mathbf{v}^x, \mathbf{v}^y, \mathbf{h}) = -\sum_i v_i^x c_i - \sum_j v_j^y a_j - \sum_k h_k b_k - \sum_{i<j<k} v_i^x v_j^y h_k W_{ijk} \tag{2.31}$$

where $\theta = \{W_{ijk}, c_i, a_j, b_k\}$ are the parameters of the model, $\mathbf{v}_x = \{v_1^x \cdots v_D^x\}, \mathbf{v}_y = \{v_1^x \cdots v_D^y\}$ are the visible states and h_k are the hidden states. We can see that the gated RBM has a cubic number of parameters as the weight matrix W_{ijk} has three associated indices. In order to reduce the number of parameters, we

[†]It is not possible to learn the covariance without injecting noise, since there needs to be uncertainty associated with the data.

can factorize the weight matrix W_{ijk} such that $W_{ijk} = W_{if}W_{jf}W_{hf}$. Then, the energy function becomes

$$E(v_i^x, v_j^y, h_k) = -\sum_i v_i^x c_i - \sum_j v_j^y a_j - \sum_k h_k b_k - \sum_i \sum_j \sum_k v_i^x v_j^y h_k W_{if} W_{jf} W_{hf}. \tag{2.32}$$

This allows us to have linearly many parameters per factors. Of course, Equation 2.32 will be less expressive compared to the energy term in Equation 2.31, but the energy with factored weights should still have enough capacity to capture patterns and regularities. We can re-express the energy function in Equation 2.32 as

$$E(v_i^x, v_j^y, h_k) = -\sum_i v_i^x c_i - \sum_j v_j^y a_j - \sum_k h_k b_k - \big(\sum_i v_i^x W_{if}\big)\big(\sum_j v_j^y W_{jf}\big)\big(\sum_k h_k W_{kf}\big). \tag{2.33}$$

The visible variables $\mathbf{v}_x$ and $\mathbf{v}_y$ are the observables, and the hidden unit h_k will be turned on if v_i^x and v_j^y have a particular pattern k. The change in the energy with respect to one of the three variables formulates to

$$\Big[E_f(h_k = 0) - E_f(h_k = 1)\Big] = W_{kf}\big(\sum_i v_i^x W_{if}\big)\big(\sum_j v_j^y W_{jf}\big), \tag{2.34}$$

assuming that the states are binary vectors.

The marginalized probability over the visible states is defined as

$$p(\mathbf{v}_x, \mathbf{v}_y; \theta) = \frac{p^*(\mathbf{v}_x, \mathbf{v}_y; \theta)}{Z(\theta)} = \frac{1}{Z(\theta)} \sum_{\mathbf{h}} \exp\left(\frac{-1}{\tau} E(\mathbf{v}_x, \mathbf{v}_y, \mathbf{h}; \theta)\right) \tag{2.35}$$

and we can re-express in terms of the free energy as

$$\begin{aligned} F(\mathbf{v}_x, \mathbf{v}_y; \theta) &= -\log \sum_{\mathbf{h}} \exp\left(\frac{-1}{\tau} E(\mathbf{v}_x, \mathbf{v}_y, \mathbf{h})\right) \\ &= -\frac{1}{\tau}\mathbf{v}_x^T \mathbf{a} - \frac{1}{\tau}\mathbf{v}_y^T \mathbf{c} - \frac{1}{\tau}\sum_{k=1} \log\left(1 + \exp\left(W^H \big(W^X \mathbf{v}_x\big)\big(W^Y \mathbf{v}_y\big) + \mathbf{b}\right)\right) \\ p(\mathbf{v}_x, \mathbf{v}_y; \theta) &= \frac{\exp\big(-F(\mathbf{v}_x, \mathbf{v}_y; \theta)\big)}{Z(\theta)}. \end{aligned}$$

This particular three-way energy-based model is called a factored gated restricted Boltzmann machine (FGRBM).

The learning rule for the FGRBM is

$$\begin{aligned}\triangle W_{kf} \propto & \langle -\frac{\partial E_f}{\partial W_{kf}} \rangle_{\text{data}} - \langle -\frac{\partial E_f}{\partial W_{kf}} \rangle_{\text{model}} \\ = & \langle h_k \big(\sum_i v_i^x W_{if} \big) \big(\sum_j v_j^y W_{jf} \big) \rangle_{\text{data}} - \langle h_k \big(\sum_i v_i^x W_{if} \big) \big(\sum_j v_j^y W_{jf} \big) \rangle_{\text{model}}\end{aligned} \tag{2.36}$$

where $\langle \cdot \rangle_{\text{data}}$ is the expectation with respect to the conditional distribution $p(\mathbf{h}|\mathbf{v}_x, \mathbf{v}_y)$ and $\langle \cdot \rangle_{\text{model}}$ is the expectation with respect to the joint distribution $p(\mathbf{v}_x, \mathbf{v}_y, \mathbf{h})$. We can apply *the contrastive divergence* algorithm to approximate the second term in Equation 2.36 such that we sample $\mathbf{h}|\mathbf{v}_x, \mathbf{v}_y$, $\mathbf{v}_x|\mathbf{v}_y, \mathbf{h}$, and $\mathbf{v}_y|\mathbf{v}_x, \mathbf{h}$. In general, this is no different than training the RBM with *the constrastive divergence* learning algorithm except that we extend to multiple variables and factorized weights.

In the literature, the three-way interaction RBMs are used to learn image transformations over time (Memisevic and Hinton, 2010) and learn time series data such as modeling human motions (Taylor and Hinton, 2009). For the case of modeling time series, we condition on visible state $\mathbf{v}_{<t}$ to predict the visible state $\mathbf{v}_t$ where $\mathbf{v}_{<t}$ is the previous time step of $\mathbf{v}_t$. The free energy can be formulated as

$$\begin{aligned}F(\mathbf{v}_t|\mathbf{v}_{<t};\theta) &= -\log \sum_{\mathbf{h}} \exp\left(\frac{-1}{\tau} E(\mathbf{v}_t, \mathbf{h}|\mathbf{v}_{<t}) \right) \\ &= \frac{1}{\tau}\mathbf{v}_{<t}^T\mathbf{a} + \frac{1}{\tau}\mathbf{v}_t^T\mathbf{c} - \frac{1}{\tau}\sum_{k=1} \log\left(1 + \exp\left(W^H (W^X \mathbf{v}_{<t})(W^Y \mathbf{v}_t) + \mathbf{b}\right)\right).\end{aligned}$$

During the training, $\mathbf{v}_{<t}$ is clamped. Hence, we do not need to sample the $\mathbf{v}_{<t}$ but only $\mathbf{v}_t$.

2.3.4 Mean-Covariance Restricted Boltzmann Machines

In the previous section, we mentioned that the world is made up of entities and the relations between them and described that the multiplicative interaction models such as the FGRBMs can model relations. Here, consider a model that can model both the entities and the correlations within the entities called *the mean-covariance restricted Boltzmann machine* (mcRBM) (Ranzato and Hinton, 2010).

mcRBM are essentially the heterogeneous compositions of RBM and FGRBMs. The energy function of mcRBM is the combination of the energy function from RBM and FGRBM such that

$$\begin{aligned}E(\mathbf{v},\mathbf{h}^m,\mathbf{h}^c) &= E_{RBM}(\mathbf{v},\mathbf{h}^m) + E_{FGRBM}(\mathbf{v},\mathbf{v},\mathbf{h}^c)\\ &= -\sum_i v_i a_i - \sum_j h_j^m b_j^m - \sum_k h_k^c b_k^c - \sum_{ij} W_{ij} v_i h_j^m - \big(\sum_k h_k^c P_{kf}\big)\big(\sum_i v_i C_{if}\big)^2\end{aligned}$$

where $\theta = \{W, P, C, \mathbf{a}, \mathbf{b}^m, \mathbf{b}^c\}$ are the parameters of the model and $\mathbf{v}$ is the visible state, and $\mathbf{h}^m$ and $\mathbf{h}^c$ are the two sets of hidden states for RBM and FGRBM. The hidden states $\mathbf{h}^m$ represent the intensities of the data and the other hidden states $\mathbf{h}^c$ represent pair-wise dependencies between the intensities of the data. The biases $\mathbf{a}$ are shared by RBM and FGRBM, the parameter $\theta^m = \{W, \mathbf{h}^m\}$ are the weight connections from RBM, the weights $\theta^c = \{P, C, \mathbf{h}^c\}$ are the factored weight connections from FGRBM. The weight P is a non-positive matrix. The inputs of the FGRBM must be tied in order to model the relations within the data.

As has been shown, there are heavy constraints that are added to the energy function. This was required to produce the Gaussian distribution with an inverse covariance matrix with dependency on $\mathbf{h}^c$ (Ranzato and Hinton, 2010) such that

$$\Sigma^{-1} = C\text{diag}(P\mathbf{h}^c)C^T. \tag{2.37}$$

The inverse covariance matrix in Equation 2.37 explains the reason why P has non-positive entries and the inputs are tied. FGRBMs with tied inputs and non-positive P are known as *covariance restricted Boltzmann machines* since they model the correlations within the data.

The marginalized probability over visible variables is formulated from the Boltzmann distribution,

$$p(\mathbf{v};\theta) = \frac{1}{Z(\theta)}\sum_{\mathbf{h}} \exp\left(\frac{-1}{\tau}E(\mathbf{v},\mathbf{h}^m,\mathbf{h}^c;\theta)\right) = \frac{1}{Z(\theta)}\sum_{\mathbf{h}} \exp\left(-F(\mathbf{v},\mathbf{h}^m,\mathbf{h}^c;\theta)\right)$$

where the free energy of mcRBM is expressed as

$$\begin{aligned}F(\mathbf{v}) = -\sum_i v_i a_i - \sum_k \log\left(1 + \exp\left(\big(\sum_f P_{jf}\big)\big(\sum_i v_i C_{if}\big)^2 + b_j\right)\right)\\ - \sum_j \log\left(1 + \exp\left(\sum_i v_i W_{ij} + b_j\right)\right).\end{aligned}$$

The conditional distribution over the visible states becomes

$$p(\mathbf{v}|\mathbf{h}^m, \mathbf{c}) = \mathcal{N}(\Sigma \left(\sum W_{ij} h_j^m \right), \Sigma). \tag{2.38}$$

The conditional distribution over the visible states show that the two sets of hidden states $\mathbf{h}^m$ and $\mathbf{h}^c$ models the Gaussian distribution with the mean and the covariance.

This particular model is applied for modeling the natural images and accoustics of speech (Ranzato and Hinton, 2010; Dahl et al., 2010). In natural images, pixels are very much correlated with very small noise on individual pixels; hence, the mcRBMs have the innate ability to model and produce the samples of natural images. Similarly, speech data has rich correlation structures. Dahl et al. (2010) employed the mcRBM to model the accoustics and then stacked RBMs on top of $\mathbf{h}^m$ and $\mathbf{h}^c$ to further extract hidden features.

3 Analyzing Dynamics on Families of Auto-encoders

The most successful and well-known example of non-probabilistic unsupervised learning is the auto-encoder. Conceptually simple and easy to train via backpropagation, various regularized variants of the model have been proposed (Rifai et al., 2011; Vincent et al., 2008; Swersky et al., 2011) as well as theoretical insights into their operation (Vincent, 2010; Droniou and Sigaud, 2013).

In practice, the latent representation learned by auto-encoders has typically been used to solve a secondary problem, often classification. The most common setup is to train a single auto-encoder on data from all classes and then a classifier is tasked to discriminate among classes. However, this contrasts with the way probabilistic models have typically been used in the past: in that literature, it is more common to train one model per class and use Bayes' rule for classification. There are two challenges to classifying using per-class auto-encoders. First, up until very recently, it was not known how to obtain the score of data under an auto-encoder, meaning how much the model "likes" an input. Second, auto-encoders are non-probabilistic, so even if they can be scored, the scores do not integrate to 1 and therefore the per-class models need to be calibrated.

Kamyshanska and Memisevic (2013) have recently shown how scores can be computed from an auto-encoder by interpreting it as a dynamical system. Although the scores do not integrate to 1, they show how one can combine the unnormalized scores into a generative classifier by learning class-specific normalizing constants from labeled data.

In this chapter, we turn our interest towards a variant of auto-encoders which are capable of learning higher-order features from data (Memisevic, 2011). The main idea is to learn relations between pixel intensities rather than the pixel intensities themselves by structuring the model as a tri-partite graph which connects hidden units to pairs of images. If the images are different, the hidden units learn how the images transform. If the images are the same, the hidden units encode within-image pixel covariances. Learning such higher-order features can yield improved results on recognition and generative tasks.

We adopt a dynamical systems view of gated auto-encoders, demonstrating that they can be scored similarly to the classical auto-encoder. We develop a theory which yields insights into the operation of gated auto-encoders. In addition to the theory, we show in our experiments that a classification model based on gated auto-encoder scoring can outperform a number of other representation learning architectures, including classical auto-encoder scoring. We also demonstrate that scoring can be useful for the structured output task of multi-label classification.

3.1 Gated Auto-Encoder Scoring

In (Kamyshanska and Memisevic, 2013), the authors showed that data could be scored under an auto-encoder by interpreting the model as a *dynamical system*. In contrast to the probabilistic views based on score matching (Swersky et al., 2011; Vincent, 2010; Droniou and Sigaud, 2013) and regularization, the dynamical systems approach permits scoring under models with either linear (real-valued data) or sigmoid (binary data) outputs, as well as arbitrary hidden unit activation functions. The method is also agnostic to the learning procedure used to train the model, meaning that it is suitable for the various types of regularized auto-encoders which have been proposed recently. In this section, we demonstrate how the dynamical systems view can be extended to the GAE.

3.1.1 Vector field representation

Similar to (Kamyshanska and Memisevic, 2013), we will view the GAE as a dynamical system with the vector field defined by

$$F(\mathbf{y}|\mathbf{x}) = r(\mathbf{y}|\mathbf{x}) - \mathbf{y}.$$

The vector field represents the local transformation that $\mathbf{y}|\mathbf{x}$ undergoes as a result of applying the reconstruction function $r(\mathbf{y}|\mathbf{x})$. Repeatedly applying the reconstruction function to an input $\mathbf{y}|\mathbf{x} \rightarrow r(\mathbf{y}|\mathbf{x}) \rightarrow r(r(\mathbf{y}|\mathbf{x})|\mathbf{x}) \rightarrow \cdots r(r \cdots r(\mathbf{y}|\mathbf{x})|\mathbf{x})$ yields a trajectory whose dynamics, from a physics perspective, can be viewed as a force field. At any point, the potential force acting on a point is the gradient of some potential energy (negative goodness) at that point. In this light, the GAE reconstruction may be viewed as pushing pairs of inputs $\mathbf{x}, \mathbf{y}$ in the direction of lower energy.

Our goal is to derive the energy function, which we call a scoring function, and which measures how much a GAE "likes" a given pair of inputs $(\mathbf{x}, \mathbf{y})$ up to normalizing constant. In order to find an expression for the potential energy, the vector field must be able to written as the derivative of a scalar field (Kamyshanska and Memisevic, 2013). To check this, we can submit to Poincaré's integrability criterion: For some open, simple connected set $\mathcal{U}$, a continuously differentiable function $F : \mathcal{U} \rightarrow \Re^m$ defines a gradient field if and only if

$$\frac{\partial F_i(\mathbf{y})}{\partial y_j} = \frac{\partial F_j(\mathbf{y})}{\partial y_i}, \ \forall i, j = 1 \cdots n.$$

The vector field defined by the GAE indeed satisfies Poincaré's integrability criterion; therefore it can be written as the derivative of a scalar field. A derivation is given in Appendix B.1.1. This also applies to the GAE with a symmetric objective function (Equation 2.30) by setting the input as $\boldsymbol{\xi}|\boldsymbol{\gamma}$ such that $\boldsymbol{\xi} = [\mathbf{y}; \mathbf{x}]$ and $\boldsymbol{\gamma} = [\mathbf{x}; \mathbf{y}]$ and following the exact same procedure.

3.1.2 Scoring the GAE

As mentioned in Section 3.1.1, our goal is to find an energy surface, so that we can express the energy for a specific pair $(\mathbf{x}, \mathbf{y})$. From the previous section, we showed that Poincaré's criterion is satisfied and this implies that we can write the vector field as the derivative of a scalar field. Moreover, it illustrates that this vector field is a conservative field and this means that the vector field is a gradient of some scalar function, which in this case is the energy function of a GAE:

$$r(\mathbf{y}|\mathbf{x}) - \mathbf{y} = \nabla E.$$

Hence, by integrating out the trajectory of the GAE $(\mathbf{x}, \mathbf{y})$, we can measure the its energy along a path. Moreover, the line integral of a conservative vector field is path independent, which allows us to take the anti-derivative of the scalar function:

$$
\begin{aligned}
E(\mathbf{y}|\mathbf{x}) = \int (r(\mathbf{y}|\mathbf{x}) - \mathbf{y}) d\mathbf{y} = \int W^Y \left(\left(W^X \mathbf{x} \right) \odot W^H h(\mathbf{u}) \right) d\mathbf{y} - \int \mathbf{y} d\mathbf{y} \\
= W^Y \left(\left(W^X \mathbf{x} \right) \odot W^H \int h\left(\mathbf{u}\right) d\mathbf{y} \right) - \int \mathbf{y} d\mathbf{y}, \qquad (3.1)
\end{aligned}
$$

where $\mathbf{u}$ is an auxiliary variable such that $\mathbf{u} = W^H((W^Y\mathbf{y}) \odot (W^X\mathbf{x}))$ and $\frac{d\mathbf{u}}{d\mathbf{y}} = W^H(W^Y \odot (W^X\mathbf{x} \otimes \mathbf{1}_D))$, and $\otimes$ is the Kronecker product. Moreover, the decoder can be re-formulated as

$$
\begin{aligned}
r(\boldsymbol{y}|\boldsymbol{x}) &= (W^Y)^T (W^X \boldsymbol{x} \odot (W^H)^T h(\boldsymbol{y}, \boldsymbol{x})) \\
&= \left((W^Y)^T \odot (W^X \boldsymbol{x} \otimes \mathbf{1}_D) \right) (W^H)^T h(\boldsymbol{y}, \boldsymbol{x}).
\end{aligned}
$$

Re-writing Equation 3.1 in terms of the auxiliary variable $\mathbf{u}$, we get

$$
\begin{aligned}
E(\mathbf{y}|\mathbf{x}) = &\left((W^Y)^T \odot (W^Y \mathbf{x} \otimes \mathbf{1}_D) \right) (W^H)^T \qquad (3.2) \\
&\int h(\mathbf{u}) \left(W^H \left(W^Y \odot (W^X \mathbf{x} \otimes \mathbf{1}_D) \right) \right)^{-1} d\mathbf{u} - \int \mathbf{y} d\mathbf{y} \\
= &\int h(\mathbf{u}) d\mathbf{u} - \frac{1}{2}\mathbf{y}^2 + \text{const.} \qquad (3.3)
\end{aligned}
$$

A more detailed derivation from Equqation 3.1 to Equation 3.3 is provided in Appendix B.1.2.

Identical to (Kamyshanska and Memisevic, 2013), if $h(\mathbf{u})$ is an element-wise activation function and we know its anti-derivative, then it is very simple to compute $E(\mathbf{x}, \mathbf{y})$.

3.2 Relationship to Restricted Boltzmann Machines

In this section, we relate GAEs through the scoring function to other types of Restricted Boltzmann Machines, such as the Factored Gated Conditional RBM (Taylor and Hinton, 2009) and the Mean-covariance RBM (Ranzato and Hinton, 2010).

3.2.1 Gated Auto-encoder and Factored Gated Conditional Restricted Boltzmann Machines

Kamyshanska and Memisevic (2013) showed that several hidden activation functions defined gradient fields, including sigmoid, softmax, tanh, linear, rectified linear function (ReLU), modulus, and squaring. These activation functions are applicable to GAEs as well.

In the case of the sigmoid activation function, $\sigma = h(\mathbf{u}) = \frac{1}{1+\exp(-\mathbf{u})}$, our energy function becomes

$$\begin{aligned} E_\sigma =& 2\int (1+\exp-(\mathbf{u}))^{-1} d\mathbf{u} - \frac{1}{2}(\mathbf{x}^2+\mathbf{y}^2) + \text{const}, \\ =& 2\sum_k \log(1+\exp(W^H_{k\cdot}(W^X\mathbf{x}\odot W^X\mathbf{y}))) - \frac{1}{2}(\mathbf{x}^2+\mathbf{y}^2) + \text{const}. \end{aligned}$$

Note that if we consider the conditional GAE we reconstruct $\mathbf{x}$ given $\mathbf{y}$ only, this yields

$$E_\sigma(\mathbf{y}|\mathbf{x}) = \sum_k \log(1+\exp(W^H(W^Y_{k\cdot}\mathbf{y}\odot W^X_{k\cdot}\mathbf{x}))) - \frac{\mathbf{y}^2}{2} + \text{const}. \tag{3.4}$$

This expression is identical, up to a constant, to the free energy in a Factored Gated Conditional Restricted Boltzmann Machine (FCRBM) with Gaussian visible units and Bernoulli hidden units. We have ignored biases for simplicity. A derivation including biases is shown in Appendix B.2.1.

3.2.2 Mean-Covariance Auto-encoder and Mean-covariance Restricted Boltzmann Machines

The Covariance auto-encoder (cAE) was introduced in (Memisevic, 2011). It is a specific form of symmetrically trained auto-encoder with identical inputs: $\mathbf{x} = \mathbf{y}$, and tied input weights: $W^X = W^Y$. It maintains a set of relational mapping units to model covariance between pixels. One can introduce a separate set of mapping units connected pairwise to only one of the inputs which model the mean intensity. In this case, the model becomes a Mean-covariance auto-encoder (mcAE).

Theorem 1. *Consider a cAE with encoder and decoder:*

$$\begin{aligned} h(\mathbf{x}) &= h(W^H((W^X\mathbf{x})^2) + \mathbf{b}) \\ r(\mathbf{x}|h) &= (W^X)^T(W^X\mathbf{x}\odot(W^H)^T h(\mathbf{x})) + \mathbf{a}, \end{aligned}$$

where $\theta = \{W^X, W^H, \mathbf{a}, \mathbf{b}\}$ are the parameters of the model, and $h(\mathbf{z}) = \frac{1}{1+\exp(-\mathbf{z})}$ is a sigmoid. Moreover, consider a Covariance RBM (Ranzato and Hinton, 2010) with Gaussian-distributed visibles and Bernoulli-distributed hiddens, with an energy function defined by

$$E^c(\mathbf{x}, \mathbf{h}) = \frac{(\mathbf{a} - \mathbf{x})^2}{\sigma^2} - \sum_f P\mathbf{h}(C\mathbf{x})^2 - \mathbf{b}\mathbf{h}.$$

Then the energy function of the cAE with dynamics $r(\mathbf{x}|\mathbf{y}) - \mathbf{x}$ is equivalent to the free energy of Covariance RBM up to a constant:

$$E(\mathbf{x}, \mathbf{x}) = \sum_k \log\left(1 + \exp\left(W^H(W^X\mathbf{x})^2 + \mathbf{b}\right)\right) - \frac{\mathbf{x}^2}{2} + \text{const.} \tag{3.5}$$

The proof is given in Appendix B.2.2. We can extend this analysis to the mcAE by using the above theorem and the results from (Kamyshanska and Memisevic, 2013).

Corollary 1.1. *The energy function of a mcAE and the free energy of a Mean-covariance RBM (mcRBM) with Gaussian-distributed visibles and Bernoulli-distributed hiddens are equivalent up to a constant. The energy of the mcAE is:*

$$E = \sum_k \log\left(1 + \exp\left(-W^H(W^X\mathbf{x})^2 - \mathbf{b}\right)\right) + \sum_k \log\left(1 + \exp(W\mathbf{x} + \mathbf{c})\right) - \mathbf{x}^2 + \text{const} \tag{3.6}$$

where $\theta^m = \{W, \mathbf{c}\}$ parameterizes the mean mapping units and $\theta^c = \{W^X, W^H, \mathbf{a}, \mathbf{b}\}$ parameterizes the covariance mapping units.

Proof. The proof is very simple. Let $E_{mc} = E_m + E_c$, where E_m is the energy of the mean auto-encoder, E_c is the energy of the covariance auto-encoder, and E_{mc} is the energy of the mcAE. We know from Theorem 1 that E_c is equivalent to the free energy of a covariance RBM, and the results from (Kamyshanska and Memisevic, 2013) show that that E_m is equivalent to the free energy of mean (classical) RBM. As shown in (Ranzato and Hinton, 2010), the free energy of a mcRBM is equal to summing the free energies of a mean RBM and a covariance RBM. □

3.3 Classification with Gated Auto-encoders

Kamyshanska and Memisevic (2013) demonstrated that one application of the ability to assign energy or scores to auto-encoders was in constructing a classifier from class-specific auto-encoders. In this section, we explore two different paradigms for classification. Similar to that work, we consider the usual multi-class problem by first training class-specific auto-encoders, and using their energy functions as confidence scores. We also consider the more challenging structured output problem, specifically, the case of multi-label prediction where a data point may have more than one associated label, and there may be correlations among the labels.

3.3.1 Classification using class-specific gated auto-encoders

One approach to classification is to take several class-specific models and assemble them into a classifier. The best-known example of this approach to fit several directed graphical models and use Bayes' rule to combine them. The process is simple because the models are normalized, or calibrated. While it is possible to apply a similar technique to undirected or non-normalized models such as auto-encoders, one must take care to calibrate them.

The approach proposed in (Kamyshanska and Memisevic, 2013) is to train K class-specific auto-encoders, each of which assigns a non-normalized energy to the data $E_i(\mathbf{x}), i = 1\ldots, K$, and then define the conditional distribution over classes z_i as

$$P(z_i|\mathbf{x}) = \frac{\exp\left(E_i(\mathbf{x}) + B_i\right)}{\sum_j \exp\left(E_j(\mathbf{x}) + B_j\right)}, \tag{3.7}$$

where B_i is a learned bias for class i. The bias terms take the role of calibrating the unnormalized energies. Note that we can similarly combine the energies from a symmetric gated auto-encoder where $\mathbf{x} = \mathbf{y}$ (i.e. a covariance auto-encoder) and apply Equation 3.7. If, for each class, we train both a covariance auto-encoder and a classical auto-encoder (i.e. a "mean" auto-encoder) then we can combine both sets of unnormalized energies as follows

$$P_{mcAE}(z_i|\mathbf{x}) = \frac{\exp(E_i^M(\mathbf{x}) + E_i^C(\mathbf{x}) + B_i)}{\sum_j \exp(E_j^M(\mathbf{x}) + E_j^C(\mathbf{x}) + B_j)}, \tag{3.8}$$

where $E_i^M(\mathbf{x})$ is the energy which comes from the "mean" (standard) auto-encoder trained on class i and $E_i^C(\mathbf{x})$ the energy which comes from the "covariance" (gated) auto-encoder trained on class i. We call the classifiers in Equation 3.7 and Equation 3.8 "Covariance Auto-encoder Scoring" (cAES) and "Mean-Covariance Auto-encoder Scoring" (mcAES), respectively.

The training procedure is summarized as follows:

1. Train a (mean)-covariance auto-encoder individually for each class. Both the mean and covariance auto-encoder have tied weights in the encoder and decoder. The covariance auto-encoder is a gated auto-encoder with tied inputs.

2. Learn the B_i calibration terms using maximum likelihood, and back-propagate to the GAE parameters.

Experimental results

We followed the same experimental setup as (Memisevic and Hinton, 2010) where we used a standard set of "Deep Learning Benchmarks" (Larochelle et al., 2007). We used mini-batch stochastic gradient descent to optimize parameters during training. The hyper-parameters: number of hiddens, number of factors, corruption level, learning rate, weight-decay, momentum rate, and batch sizes were chosen based on a held-out validation set. Corruption levels and weight-decay were selected from $\{0, 0.1, 0.2, 0.3, 0.4, 0.5\}$, and number of hidden and factors were selected from $\{100, 300, 500\}$. We selected the learning rate and weight-decay from the range $(0.001, 0.0001)$.

Classification error results are shown in Table 3.1. First, the error rates of auto-encoder scoring variant methods illustrate that across all datasets AES outperforms cAES and mcAES outperforms both AES and cAES. AE models pixel means and cAE models pixel covariance, while mcAE models both mean and covariance, making it naturally more expressive. We observe that cAES and mcAES achieve lower error rates by a large margin on rotated MNIST with backgrounds (final row). On the other hand, both cAES and mcAES perform poorly on MNIST with random white noise background (second row from bottom). We believe this phenomenon is due to the inability to model covariance in this dataset. In MNIST with random white noise the pixels are typically uncorrelated, where in rotated MNIST with backgrounds the correlations are present and consistent.

3.3.2 Multi-label classification via optimization in label space

The dominant application of deep learning approaches to vision has been the assignment of images to discrete classes (e.g. object recognition). Many

DATA	SVM RBF	RBM	DEEP SAA$_3$	GSM	AES	cAES	mcAES
RECT	2.15	4.71	2.14	0.56	0.84	0.61	**0.54**
RECT$_{\text{IMG}}$	24.04	23.69	24.05	22.51	21.45	22.85	**21.41**
CONVEX	19.13	19.92	18.41	**17.08**	21.52	21.6	20.63
MNIST$_{\text{SMALL}}$	3.03	3.94	3.46	3.70	**2.61**	3.65	3.65
MNIST$_{\text{ROT}}$	11.11	14.69	**10.30**	11.75	11.25	16.5	13.42
MNIST$_{\text{RAND}}$	14.58	9.80	11.28	10.48	**9.70**	18.65	16.73
MNIST$_{\text{ROTIM}}$	55.18	52.21	51.93	55.16	47.14	39.98	**35.52**

Table 3.1. Classification error rates on the Deep Learning Benchmark dataset. SAA$_3$ stands for three-layer Stacked Auto-encoder. SVM and RBM results are from (Vincent, 2010), DEEP and GSM are results from (Memisevic, 2011), and AES is from (Kamyshanska and Memisevic, 2013).

applications, however, involve "structured outputs" where the output variable is high-dimensional and has a complex, multi-modal joint distribution. Structured output prediction may include tasks such as multi-label classification where there are regularities to be learned in the output, and segmentation, where the output is as high-dimensional as the input. A key challenge to such approaches lies in developing models that are able to capture complex, high level structure like shape, while still remaining tractable.

Though our proposed work is based on a deterministic model, we have shown that the energy, or scoring function of the GAE is equivalent, up to a constant, to that of a conditional RBM, a model that has already seen some use in structured prediction problems (Mnih et al., 2011; Li et al., 2013).

GAE scoring can be applied to structured output problems as a type of "post-classification" (Mnih and Hinton, 2010). The idea is to let a naiv̈e, non-structured classifier make an initial prediction of the outputs in a fast, feed-forward manner, and then allow a second model (in our case, a GAE) clean up the outputs of the first model. Since GAEs can model the relationship between input **x** and structured output **y**, we can initialize the output with the output of the naiv̈e model, and then optimize its energy function with respect to the outputs. Input **x** is held constant throughout the optimization.

Li et al. (2013) recently proposed Compositional High Order Pattern Potentials, a hybrid of Conditional Random Fields (CRF) and Restricted Boltzmann

Machines. The RBM provides a global shape information prior to the locally-connected CRF. Adopting the idea of *learning* structured relationships between outputs, we propose an alternate approach which the inputs of the GAE are not $(\mathbf{x}, \mathbf{y})$ but $(\mathbf{y}, \mathbf{y})$. In other words, the post-classification model is a covariance auto-encoder. The intuition behind the first approach is to use a GAE to learn the relationship between the input $\mathbf{x}$ and the output $\mathbf{y}$, whereas the second method aims to learn the correlations between the outputs $\mathbf{y}$.

We denote our two proposed methods GAE_{XY} and GAE_{Y^2}. GAE_{XY} corresponds to a GAE, trained conditionally, whose mapping units directly model the relationship between input and output and GAE_{Y^2} corresponds to a GAE which models correlations between output dimensions. GAE_{XY} defines $E(\mathbf{y}|\mathbf{x})$, while GAE_{Y^2} defines $E(\mathbf{y}|\mathbf{y}) = E(\mathbf{y})$. They differ only in terms of the data vectors that they consume. The training and test procedures are detailed in Algorithm 1[†].

Experimental results

We consider multi-label classification, where the problem is to classify instances which can take on more than one label at a time. We followed the same experimental set up as (Mnih et al., 2011). Four multi-labeled datasets were considered: Yeast (Elisseeff and Weston, 2002) consists of biological attributes, Scene (Boutell et al., 2004) is image-based, and MTurk (Mandel et al., 2010) and MajMin (Mandel and Ellis, 2008) are targeted towards tagging music. Yeast consists of 103 biological attributes and has 14 possible labels, Scene consists of 294 image pixels with 6 possible labels, and MTurk and MajMin each consist of 389 audio features extracted from music and have 92 and 96 possible tags, respectively. Figure 3.1 visualizes the covariance matrix for the label dimensions in each dataset. We can see from this that there are correlations present in the labels which suggests that a structured approach may improve on a non-structured predictor.

We compared our proposed approaches to logistic regression, a standard MLP, and the two structured CRBM training algorithms presented in (Mnih et al., 2011). To permit a fair comparison, we followed the same procedure for training and reporting errors as in that paper, where we cross validated over 10 folds and training, validation, test examples are randomly separated into

[†] In our experiments, we used the cross-entropy loss function for loss_1 and loss_2.

Algorithm 1 Structured Output Prediction with GAE scoring

```
1: procedure MULTI-LABEL CLASSIFICATION($\mathcal{D} = \{(\mathbf{x}_i, \mathbf{y}_i) \in \mathcal{X}_{train} \times \mathcal{Y}_{train}\}$ )
2:     Train a Multi-layer Perceptron (MLP) to learn an input-output mapping $f(\cdot)$:
           $$\underset{\theta_1}{\operatorname{argmin}}\, l(\mathbf{x}, \mathbf{y}; \theta_1) = \sum_i \mathrm{loss}_1\left((f(\mathbf{x}_i; \theta_1) - \mathbf{y}_i\right) \quad (3.9)$$
       where $\mathrm{loss}_1$ is an appropriate loss function for the MLP.
3:     Train a Gated Auto-encoder with inputs $(\mathbf{x}_i, \mathbf{y}_i)$; For the case of $\mathrm{GAE}_{Y^2}$, set $\mathbf{x}_i = \mathbf{y}_i$.
           $$\underset{\theta_2}{\operatorname{argmin}}\, l(\mathbf{x}, \mathbf{y}; \theta_2) = \sum_i \mathrm{loss}_2\left(r(\mathbf{y}_i|\mathbf{x}_i, \theta_2) - \mathbf{y}_i\right) \quad (3.10)$$
       where $\mathrm{loss}_2$ is an appropriate reconstructive loss for the auto-encoder.
4:     for each test data point $\mathbf{x_i} \in \mathcal{X}_{test}$ do
5:         Initialize the output using the MLP.
               $$\mathbf{y}_0 = f(\mathbf{x}_{test}) \quad (3.11)$$
6:         while $\|E(\mathbf{y}_{t+1}|\mathbf{x}) - E(\mathbf{y}_t|\mathbf{x})\| > \epsilon$ or $\leq$ max. iter. do
7:             Compute $\nabla_{\mathbf{y}_t} E$
8:             Update $\mathbf{y}_{t+1} = \mathbf{y}_t - \lambda \nabla_{\mathbf{y}_t} E$
9:             where $\epsilon$ is the tolerance rate with respect to the convergence of the optimization.
```

80%, 10%, and 10% in each fold. The error rate was measured by averaging the errors on each label dimension.

The performance on four multi-label datasets is shown in Table 3.2. We observed that adding a small amount of Gaussian noise to the input $\mathbf{y}$ improved the performance for GAE_{XY}. However, adding noise to the input $\mathbf{x}$ did not have as much of an effect. We suspect that adding noise makes the GAE more robust to the input provided by the MLP. Interestingly, we found that the performance of GAE_{Y2} was negatively affected by adding noise. Both of our proposed methods, GAES_{XY} and GAES_{Y2} generally outperformed the other methods except for GAES_{Y2} on the MajMin dataset. At least for these datasets, there is no clear winner between the two. GAES_{XY} achieved lower error than GAES_{Y2} for Yeast and MajMin, and the same error rate on the MTurk dataset. However, GAES_{Y2} outperforms GAES_{XY} on the Scene dataset. Overall, the results show that GAE scoring may be a promising means of

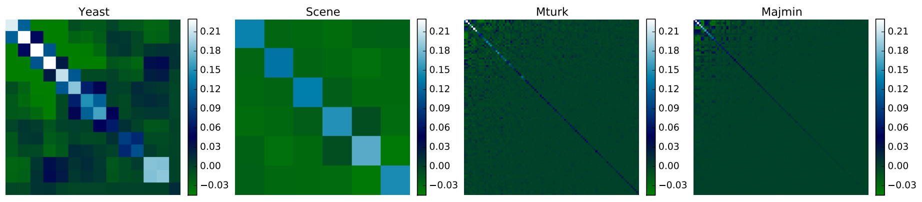

Figure 3.1. Covariance matrices for the multi-label datasets: Yeast, Scene, MTurk, and MajMin.

Method	Yeast	Scene	MTurk	MajMin
LogReg	20.16	10.11	8.10	4.34
HashCRBM*	20.02	8.80	7.24	4.24
MLP	19.79	8.99	7.13	4.23
GAES$_{XY}$	**19.27**	6.83	**6.59**	**3.96**
GAES$_{Y^2}$	19.58	**6.81**	**6.59**	4.29

Figure 3.2. Error rate on multi-label datasets. We report standard errors across 10 repeated runs with different random weight initializations. Note that LogReg and HashCRBM are taken from (Mnih et al., 2011) and we therefore do not report standard errors.

post-classification in structured output prediction.

3.4 Discussion

There have been many theoretical and empirical studies on auto-encoders (Vincent et al., 2008; Rifai et al., 2011; Swersky et al., 2011; Vincent, 2010; Guillaume and Bengio, 2013; Kamyshanska and Memisevic, 2013), however, the theoretical study of gated auto-encoders is limited apart from (Memisevic, 2011; Droniou and Sigaud, 2013). The GAE has several intriguing properties that a classical auto-encoder does not, based on its ability to model relations among pixel intensities rather than just the intensities themselves. This opens up a broader set of applications. In this paper, we derive some theoretical results for the GAE that enable us to gain more insight and understanding of its operation.

We cast the GAE as a dynamical system driven by a vector field in order to analyze the model. In the first part of the paper, by following the same procedure as (Kamyshanska and Memisevic, 2013), we showed that the GAE could be scored according to an energy function. From this perspective, we demonstrated the equivalency of the GAE energy to the free energy of a FCRBM with Gaussian visible units, Bernoulli hidden units, and sigmoid hidden activations. In the same manner, we also showed that the covariance auto-encoder can be formulated in a way such that its energy function is the same as the free energy of a covariance RBM, and this naturally led to a connection between the mean-covariance auto-encoder and mean-covariance RBM. One interesting observation is that Gaussian-Bernoulli RBMs have been reported to be difficult to train (Krizhevsky, 2009; Cho et al., 2011), and the success of training RBMs is highly dependent on the training setup (Wang et al., 2014). Auto-encoders are an attractive alternative, even when an energy function is required.

Structured output prediction is a natural next step for representation learning. The main advantage of our approach compared to other popular approaches such as Markov Random Fields, is that inference over is extremely fast, using a gradient-based optimization of the auto-encoder scoring function. In the future, we plan on tackling more challenging structured output prediction problems.

4 Training Energy-based Models Under Various Kinds of Dynamics

Section 2.2.2 describes the maximum likelihood learning approach to learn an appropriate probabilistic energy function with respect to the parameters. A common problem in maximum likelihood learning in energy-based models is to estimate the parameters of a high-dimensional probabilistic model using gradient descent on the model's negative log likelihood. For exponential family models where $p(x)$ is proportional to the exponential of a negative potential function $F(x)$, the gradient of the data negative log-likelihood takes the form in Equation 2.23. Recall that the first term has the effect of pushing the parameters in a direction that decreases the energy surface of the model at the training data points, while the second term increases the energy of all possible states. Since the second term is intractable for all but trivial models, we cannot, in practice, accommodate for every state of x, but rather resort to sampling. We call states in the sum in the first term *positive particles* and those in the second term *negative particles*, in accordance with their effect on the likelihood (opposite their effect on the energy). Thus, the intractability of the second term becomes a problem of *negative particle selection* (NPS).

The most famous approach to NPS is Contrastive Divergence (CD) (Hinton, 2002), which is the centre-piece of unsupervised neural network learning in energy-based models. CD proposes to sample the negative particles by employing a Markov chain Monte Carlo (MCMC) transition operator a small number of times to each data state. This is in contrast to taking an unbiased sample from the distribution, by applying the MCMC operator a large number of times, until the distribution reaches equilibrium, which is often prohibitive for practical applications. Much research has attempted to better understand this approach and the reasoning behind its success or failure (Sutskever and Tieleman, 2009; MacKay, 2001), leading to many variations being proposed from the perspective of improving the MCMC chain.

Alternatively, we can try to understand or look closely at the model's dynamical system. Typically, there are two stage process in the maximum likelihood learning where one is inference and another one is learning, and we usually have the separate treatments for the two. By taking the dynamical systems approach, we do not have to separate them into the two separate stage procedure. What we mean by "dynamical system approach?" We usually start from some distribution and it evolves to another distribution, more precisely to a stationary distribution. One can think of it as the states evolving towards more probable states. This particular idea is often used in physics and chemistry related fields, where they describe the time-evolution of a system that can be modeled with the countable number of states at any time and probabilistically switches between states. In this chapter, we take a more general approach to the problem of NPS, in particular, through the lens of the Minimum Probability Flow (MPF) algorithm (Sohl-Dickstein et al., 2011). The aim of this chapter is to contrast variations on the dynamics, and experimentally demonstrate that MPF training outperforms CD on Restricted Boltzmann Machines.

4.1 Minimum Probabilty Flow

The key intuition behind MPF is that NPS can be reformulated in a firm theoretical context by treating the model distribution as the end point of some explicit continuous dynamics, and seeking to minimize the flow of probability away from the data under those dynamics. In this context then, NPS is no longer a sampling procedure employed to approximate an intractable function, but arises naturally out of the probability flow from data states to non-data states. That is, MPF provides a theoretical environment for the formal treatment of T_{ij} that offers a much more general perspective of that operator than CD-k can. In the same vein, it better formalizes the notion of minimizing divergence between positive and negative particles.

4.1.1 Dynamics of the Model

The primary mathematical apparatus for MPF is a continuous time Markov chain known as the *master equation*,

$$\dot{p}_i = \sum_{j \neq i} [\Gamma_{ij} p_j^{(t)} - \Gamma_{ji} p_i^{(t)}] \tag{4.1}$$

where j are the data states and i are the non-data states and Γ_{ij} is the probability flow rate from state j to state i. Note that each state is a full vector of variables, and we are theoretically enumerating all states. $\dot{p}_i$ is the rate of change of the probability of state i, that is, the difference between the probability flowing out of any state j into state i and the probability flowing out of state i to any other state j at time t. We can re-express $\dot{p}_i$ in a simple matrix form as

$$\dot{\mathbf{p}} = \mathbf{\Gamma} \mathbf{p} \tag{4.2}$$

by setting $\Gamma_{ii} = -\sum_{i \neq j} \Gamma_{ji} p_i^{(t)}$. We note that if the transition matrix Γ is ergodic, then the model has a unique stationary distribution.

This is a common model for exploring statistical mechanical systems, but it is unwieldly in practice for two reasons, namely, the continuous time dynamics, and exponential size of the state space. For our purposes, we will actually find the former an advantage, and the latter irrelevant.

The objective of MPF is to minimize the KL divergence between the data distribution and the distribution after evolving an infinitesimal amount of time under the dynamics:

$$\theta_{MPF} = \text{argmin}_\theta J(\theta), \; J(\theta) = D_{KL}(p^{(0)} || p^{(\epsilon)}(\theta))$$

Approximating $J(\theta)$ up to a first order Taylor expansion with respect to time t, our objective function reduces to

$$J(\theta) = \frac{\epsilon}{|\mathcal{D}|} \sum_{j \in \mathcal{D}} \sum_{i \notin \mathcal{D}} \Gamma_{ij} \tag{4.3}$$

and θ can be optimized by gradient descent on $J(\theta)$. Since Γ_{ij} captures probability flow from state j to state i, this objective function has the quite elegant interpretation of minimizing the probability flow from data states to non-data states (Sohl-Dickstein et al., 2011).

4.1.2 Form of the Transition Matrix

MPF does not propose to *actually* simulate these dynamics. There is, in fact, no need to, as the problem formulation reduces to a rather simple optimization problem with no intractable component. However, we must provide a means for computing the matrix coefficients Γ_{ij}. Since our target distribution is the distribution defined by the RBM, we require Γ to be a function of the energy, or more particularly, the parameters of the energy function.

A sufficient (but not necessary) means to guarantee that the distribution $\mathbf{p}^{\infty}(\theta)$ is a fixed point of the dynamics is to choose Γ to satisfy detailed balance, that is

$$\Gamma_{ji} p_i^{(\infty)}(\theta) = \Gamma_{ij} p_j^{(\infty)}(\theta). \tag{4.4}$$

The following theorem provides a general form for the transition matrix such that the equilibrium distribution is that of the RBM:

Theorem 1. *Suppose $p_j^{(\infty)}$ is the probability of state j and $p_i^{(\infty)}$ is the probability of state i. Let the transition matrix be*

$$\Gamma_{ij} = g_{ij} \exp\left(\frac{o(F_i - F_j) + 1}{2} (F_j - F_i) \right) \tag{4.5}$$

such that $o(\cdot)$ is any odd function, where g_{ij} is the symmetric connectivity between the states i and j. Then this transition matrix satisfies detailed balance in Equation 4.4.

The proof is provided in Appendix C.1.1. The transition matrix proposed by (Sohl-Dickstein et al., 2011) is thus the simplest case of Theorem 1, found by setting $o(\cdot) = 0$ and $g_{ij} = g_{ji}$:

$$\Gamma_{ij} = g_{ij} \exp\left(\frac{1}{2} (F_j(\theta) - F_i(\theta) \right). \tag{4.6}$$

Given a form for the transition matrix, we can now evaluate the gradient of $J(\theta)$

$$\frac{\partial J(\theta)}{\partial \theta} = \frac{\epsilon}{|\mathcal{D}|} \sum_{j \in \mathcal{D}} \sum_{i \notin \mathcal{D}} \left(\frac{\partial F_j(\theta)}{\partial \theta} - \frac{\partial F_i(\theta)}{\partial \theta} \right) T_{ij}$$
$$T_{ij} = g_{ij} \exp\left(\frac{1}{2} (F_j(\theta) - F_i(\theta)) \right)$$

and observe the similarity to the formulation given for the RBM trained by CD-k (Equation 2.24). Unlike with CD-k, however, this expression was derived through an explicit dynamics and well-formalized minimization objective.

4.2 Probability Flow Rates Γ

At first glance, MPF might appear doomed, due to the size of Γ, namely $2^D \times 2^D$, and the problem of enumerating all of the states. However, the objective function in Equation 4.3 summing over the Γ_{ij}'s only considers transitions between data states j (limited in size by our data set) and non-data states i (limited by the sparseness of our design). By specifying Γ to be sparse, the intractability disappears, and complexity is dominated by the size of the dataset.

Using traditional methods, an RBM can be trained in two ways, either with sampled negative particles, like in CD-k or PCD (also known as stochastic maximum likelihood) (Hinton, 2002; Tieleman and Hinton, 2009), or via an inductive principle, with fixed sets of "fantasy cases", like in general score matching, ratio matching, or pseudolikelihood (Hyvärinen, 2005; Marlin and Freitas, 2011; Besag, 1975). In a similar manner, we can define Γ by specifying the connectivity function g_{ij} either as a distribution from which to sample or as fixed and deterministic.

In this section, we examine various kinds of connectivity functions and their consequences on the probability flow dynamics.

4.2.1 1-bit flip connections

It can be shown that score matching is a special case of MPF in continuous state spaces, where the connectivity function is set to connect all states within a small Euclidean distance r in the limit of $r \to 0$ (Sohl-Dickstein et al., 2011). For simplicity, in the case of a discrete state space (Bernoulli RBM), we can fix the Hamming distance to one instead, and consider that data states are connected to all other states 1-bit flip away:

$$g_{ij} = \begin{cases} 1, & \text{if state } i, j \text{ differs by single bit flip} \\ 0, & \text{otherwise} \end{cases} \tag{4.7}$$

1-bit flip connectivity gives us a sparse Γ with $2^D D$ non-zero terms (rather than a full 2^{2D}), and may be seen as NPS where the only negative particles are those which are 1-bit flip away from data states. Therefore, we only ever evaluate $|\mathcal{D}|D$ terms from this matrix, making the formulation tractable. This was the only connectivity function pursued in (Sohl-Dickstein et al., 2011) and is a natural starting point for the approach.

Algorithm 2 Minimum probability flow learning with single bit-flip connectivity. Note we leave out all g_{ij} since here we are explicit about only connecting states of Hamming distance 1.

```
• Initialize the parameters $\theta$
• for each training example $d \in \mathcal{D}$ do
    1. Compute the list of states, $L$, with Hamming distance 1 from $d$
    2. Compute the probability flow $\Gamma_{id} = \exp\left(\frac{1}{2}(F_d(\theta) - F_i(\theta))\right)$ for each $i \in L$
    3. The cost function for $d$ is $\sum_{i \in L} \Gamma_{id}$
    4. Compute the gradient of the cost function, $\frac{\partial J(\theta)}{\partial \theta} = \sum_{i \in L} \left( \frac{\partial F_d(\theta)}{\partial \theta} - \frac{\partial F_i(\theta)}{\partial \theta} \right) \Gamma_{id}$
    5. Update parameters via gradient descent with $\theta \leftarrow \theta - \lambda \nabla J(\theta)$
  end for
```

4.2.2 Factorized Minimum Probability Flow

Previously, we considered connectivity g_{ij} as a binary indicator function of both states i and j. Instead, we may wish to use a probability distribution, such that g_{ij} is the probability that state j is connected to state i (i.e. $\sum_i g_{ij} = 1$). Following (Sohl-Dickstein, 2011), we simplify this approach by letting $g_{ij} = g_i$, yielding an *independence chain* (Tierney, 1994). This means the probability of being connected to state i is independent of j, giving us an alternative way of constructing a transition matrix such that the objective function can be factorized:

$$J(\theta) = \frac{1}{|\mathcal{D}|} \sum_{j \in \mathcal{D}} \sum_{i \notin \mathcal{D}} g_i \left(\frac{g_j}{g_i} \right)^{\frac{1}{2}} \exp\left(\frac{1}{2} \big(F_j(\mathbf{x}; \theta) - F_i(\mathbf{x}; \theta) \big) \right) \tag{4.8}$$

$$= \left(\frac{1}{|\mathcal{D}|} \sum_{j \in \mathcal{D}} \exp\left(\frac{1}{2} \big(F_j(\mathbf{x}; \theta) + \log g_j \big) \right) \right) \left(\sum_{i \notin \mathcal{D}} g_i \exp\left(\frac{1}{2} \big(-F_i(\mathbf{x}; \theta) + \log g_i \big) \right) \right) \tag{4.9}$$

where $\left(\frac{g_j}{g_i}\right)^{\frac{1}{2}}$ is a scaling term required to counterbalance the difference between g_i and g_j. The independence in the connectivity function allows us to factor all the j terms in 4.8 out of the inner sum, leaving us with a product of sums, something we could not achieve with 1-bit flip connectivity since the connection to state i depends on it being a neighbor of state j. Note that, intuitively, learning is facilitated by connecting data states to states that are probable under the model (i.e. to contrast the divergence). Therefore, we can use $p(v;\theta)$ to approximate g_i. In practice, for each iteration n of learning, we need the g_i and g_j terms to act as constants with respect to updating θ, and thus we sample them from $p(v;\theta^{n-1})$. We can then rewrite the objective function as $J(\theta) = J_{\mathcal{D}}(\theta)J_{\mathcal{S}}(\theta)$

$$\begin{aligned} J_{\mathcal{D}}(\theta) &= \left(\frac{1}{|\mathcal{D}|}\sum_{\mathbf{x}\in\mathcal{D}} \exp\left[\frac{1}{2}\big(F(\mathbf{x};\theta) - F(\mathbf{x};\theta^{n-1})\big)\right]\right) \\ J_{\mathcal{S}}(\theta) &= \left(\frac{1}{|\mathcal{S}|}\sum_{\mathbf{x}'\in\mathcal{S}} \exp\left[\frac{1}{2}\big(-F(\mathbf{x}';\theta) + F(\mathbf{x}';\theta^{n-1})\big)\right]\right) \end{aligned} \tag{4.10}$$

where $\mathcal{S}$ is the sampled set from $p(\mathbf{v};\theta^{n-1})$, and the normalization terms in $\log g_j$ and $\log g_i$ cancel out. Note we use the θ^{n-1} notation to refer to the parameters at the previous iteration, and simply θ for the current iteration.

4.2.3 Persistent Minimum Probability Flow

There are several ways of sampling “fantasy particles” from $p(\mathbf{v};\theta^{n-1})$. Notice that taking the data distribution with respect to θ^{n-1} is necessary for stable learning.

Previously, persistent contrastive divergence (PCD) was developed to improve CD-k learning (Tieleman and Hinton, 2009). Similarly, persistence can be applied to sampling in MPF connectivity functions. For each update, we pick a new sample based on a MCMC sampler which starts from previous samples. Then we update θ^n, which satsifies $J(\theta^n) \leq J(\theta^{n-1})$ (Sohl-Dickstein, 2011). The pseudo-code for persistent MPF is the same as Factored MPF except for drawing new samples, which is indicated by square brackets in Algorithm 3.

As we will show, using persistence in MPF is important for achieving faster convergence in learning. While the theoretical formulation of MPF guarantees eventual convergence, the focus on minimizing the initial probability flow will have little effect if the sampler mixes too slowly. In practice, combining the persistent samples and non-persistent samples gave better performance.

Algorithm 3 Factored [Persistent] MPF learning with probabilistic connectivity.

```
• for each epoch $n$ do
    1. Draw a new sample $S^n$ based on $S^0$ $\left[S^{n-1}\right]$ using an MCMC sampler.
    2. Compute $J_{\mathcal{S}}(\theta)$
    3. for each training example $d \in \mathcal{D}$ do
        (a) Compute $J_d(\theta)$. The cost function for $d$ is $J(\theta) = J_d(\theta) J_{\mathcal{S}}(\theta)$
        (b) Compute the gradient of the cost function,
            $\frac{\partial J(\theta)}{\partial \theta} = J_{\mathcal{S}}(\theta) J_d(\theta) \frac{\partial F_d(\theta)}{\partial \theta} + \frac{1}{|\mathcal{S}|} J_d \sum_{x' \in \mathcal{S}} \left( \frac{\partial F(x')}{\partial \theta} \exp\left[ \frac{1}{2}\left( F(\mathbf{x}';\theta) - F(\mathbf{x}';\theta^{n-1}) \right) \right] \right)$
        (c) Update parameters via gradient descent with $\theta \leftarrow \theta - \lambda \nabla J(\theta)$
  end for
```

4.3 Experiments

We conducted the first empirical study of MPF under different types of connectivity as discussed in Section 4.2. We compared our results to CD-k with varying values for K. We analyzed the MPF variants based on training RBMs and assessed them quantitatively and qualitatively by comparing the log-liklihoods of the test data and samples generated from model. For the experiments, we denote the 1-bit flip, factorized, and persistent methods as MPF-1flip, FMPF, and PMPF, respectively.

The goals of these experiments are to

1. Compare the performance between MPF algorithms under different connectivities; and
2. Compare the performance between MPF and CD-k.

In our experiments, we considered the MNIST and CalTech Silhouette datasets. MNIST consists of 60,000 training and 10,000 test images of size 28 $\times$ 28 pixels containing handwritten digits from the classes 0 to 9. The pixels in MNIST are binarized based on thresholding. From the 60,000 training examples, we set aside 10,000 as validation examples to tune the hyperparameters in our models. The CalTech Silhouette dataset contains the outlines of objects from the CalTech101 dataset, which are centered and scaled on a 28 $\times$ 28 image plane and rendered as filled black regions on a white background creating a silhouette of each object. The training set consists of 4,100 examples, with at least 20 and at most 100 examples in each category. The remaining instances were split evenly between validation and testing[†]. Hyperparameters such as learning rate, number of epochs, and batch size were selected from discrete ranges and chosen based on a held-out validation set. The learning rate for FMPF and PMPF were chosen from the range [0.001, 0.00001] and the learning rate for 1-bit flip was chosen from the range [0.2, 0.001].

4.3.1 MNIST - exact log likelihood

In our first experiment, we trained eleven RBMs on the MNIST digits. All RBMs consisted of 20 hidden units and 784 (28$\times$28) visible units. Due to the small number of hidden variables, we calculated the exact value of the partition function by explicitly summing over all visible configurations. Five RBMs were learned by PCD1, CD1, CD10, CD15, and CD25. Seven RBMs were learned by 1 bit flip, FMPF, and FPMPF[‡]. Block Gibbs sampling is required for FMPF-k and FPMPF-k similar to CD-k training, where the number of steps is given by k.

The average log test likelihood values of RBMs with 20 hidden units are presented in Table 4.1. This table gives a sense of the performance under different types of MPF dynamics when the partition function can be calculated exactly. We observed that PMPF consistently achieved a higher log-likelihood than FMPF. MPF with 1 bit flip was very fast but gave poor performance compared to FMPF and PMPF. We also observed that MPF-1flip outperformed CD1. FMPF always performed slightly worse than CD-k training with the same number of Gibbs steps. However, PMPF always outperformed CD-k.

One advantage of FMPF is that it converges much quicker than CD-k or PMPF. This is because we used twice many samples as PMPF as mentioned in Section 4.2.3. Figure 4.1 shows initial data and the generated samples after

[†]More details on pre-processing the CalTech Silhouettes can be found in http://people.cs.umass.edu/ marlin/data.shtml

[‡]FPMPF is the composition of the FMPF and PMPF connectivities.

Table 4.1. Experimental results on MNIST using 11 RBMs with 20 hidden units each. The average training and test log-probabilities over 10 repeated runs with random parameter initializations are reported.

Method	Average log Test	Average log Train	Time (sec)	Batchsize
CD1	-145.63 $\pm$ 1.30	-146.62 $\pm$ 1.72	831	100
PCD	**-136.10 $\pm$ 1.21**	**-137.13 $\pm$ 1.21**	2620	300
MPF-1flip	-142.13 $\pm$ 2.01	-143.02 $\pm$ 3.96	2931	75
CD10	-135.40 $\pm$ 1.21	-136.46 $\pm$ 1.18	17329	100
FMPF10	-136.37 $\pm$ 0.17	-137.35 $\pm$ 0.19	12533	60
PMPF10	-141.36 $\pm$ 0.35	-142.73 $\pm$ 0.35	11445	25
FPMPF10	**-134.04 $\pm$ 0.12**	**-135.25 $\pm$ 0.11**	22201	25
CD15	-134.13 $\pm$ 0.82	-135.20 $\pm$ 0.84	26723	100
FMPF15	-135.89 $\pm$ 0.19	-136.93 $\pm$ 0.18	18951	60
PMPF15	-138.53 $\pm$ 0.23	-139.71 $\pm$ 0.23	13441	25
FPMPF15	**-133.90 $\pm$ 0.14**	**-135.13 $\pm$ 0.14**	27302	25
CD25	-133.02 $\pm$ 0.08	-134.15 $\pm$ 0.08	46711	100
FMPF25	-134.50 $\pm$ 0.08	-135.63 $\pm$ 0.07	25588	60
PMPF25	-135.95 $\pm$ 0.13	-137.29 $\pm$ 0.13	23115	25
FPMPF25	**-132.74 $\pm$ 0.13**	**-133.50 $\pm$ 0.11**	50117	25

running 100 Gibbs steps from each RBM. PMPF produces samples that are visually more appealing than the other methods.

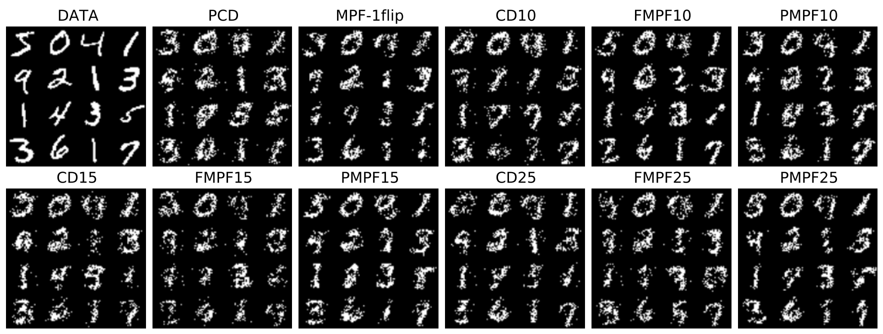

Figure 4.1. Samples generated from the training set. Samples in each panel are generated by RBMs trained under different paradigms as noted above each image.

4.3.2 MNIST - estimating log likelihood

In our second set of experiments, we trained RBMs with 200 hidden units. We trained them exactly as described in Section 4.3.1. These RBMs are able to generate much higher-quality samples from the data distribution, however, the partition function can no longer be computed exactly.

In order to evaluate the model quantitatively, we estimated the test log-likelihood using the Conservative Sampling-based Likelihood estimator (CSL) (Bengio et al., 2013b) and annealed importance sampling (AIS) (Salakhutdinov and Murray, 2008). Given well-defined conditional probabilities $P(\mathbf{v}|\mathbf{h})$ of a model and a set of latent variable samples S collected from a Markov chain, CSL computes

$$\log \hat{f}(\mathbf{v}) = \log \text{mean}_{h \in S} P(\mathbf{v}|\mathbf{h}). \tag{4.11}$$

The advantage of CSL is that sampling latent variables $\mathbf{h}$ instead of $\mathbf{v}$ has the effect of reducing the variance of the estimator. Also, in contrast to annealed importance sampling (AIS) (Salakhutdinov and Murray, 2008), which tends to overestimate, CSL is much more conservative in its estimates. However, most of the time, CSL is far off from the true estimator, so we bound our negative log-likelihood estimate from above and below using both AIS and CSL.

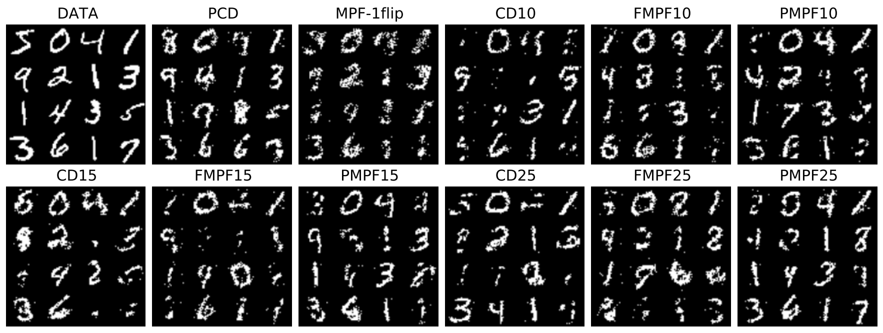

Figure 4.2. Samples generated from the training set. Samples in each panel are generated by RBMs trained under different paradigms as noted above each image.

Table 4.2 demonstrates the test log-likelihood of various RBMs with 200 hidden units. The ranking of the different training paradigms with respect to performance was similar to what we observed in Section 4.3.1 with PMPF emerging as the winner. However, contrary to the first experiment, we observed that MPF with 1 bit flip did not perform well. Moreover, FMPF and PMPF both tended to give higher test log-likelihoods than CD-k training. Smaller batch sizes worked better with MPF when the number of hiddens was increased. Once again, we observed smaller variances compared to CD-k with

Table 4.2. Experimental results on MNIST using 11 RBMs with 200 hidden units each. The average estimated training and test log-probabilities over 10 repeated runs with random parameter initializations are reported. Likelihood estimates are made with CSL (Bengio et al., 2013b) and AIS (Salakhutdinov and Murray, 2008)

	CSL		AIS			
Method	Avg. log Test	Avg. log Train	Avg. log Test	Avg. log Train	Time (sec)	Batchsize
CD1	-138.63 $\pm$ 0.48	-138.70 $\pm$ 0.45	-98.75 $\pm$ 0.66	-98.61 $\pm$ 0.66	1258	100
PCD1	**-114.14 $\pm$ 0.26**	**-114.13 $\pm$ 0.28**	**-88.82 $\pm$ 0.53**	**-89.92 $\pm$ 0.54**	2614	100
MPF-1flip	-179.73 $\pm$ 0.085	-179.60 $\pm$ 0.07	-141.95 $\pm$ 0.23	-142.38 $\pm$ 0.74	4575	75
CD10	-117.74 $\pm$ 0.14	-117.76 $\pm$ 0.13	-91.94 $\pm$ 0.42	-92.46 $\pm$ 0.38	24948	100
FMPF10	-115.11 $\pm$ 0.09	-115.10 $\pm$ 0.07	-91.21 $\pm$ 0.17	-91.39 $\pm$ 0.16	24849	25
PMPF10	-114.00 $\pm$ 0.08	-113.98 $\pm$ 0.09	-89.26 $\pm$ 0.13	-89.37 $\pm$ 0.13	24179	25
FPMPF10	**-112.45 $\pm$ 0.03**	**-112.45 $\pm$ 0.03**	**-83.83 $\pm$ 0.23**	**-83.26 $\pm$ 0.23**	24354	25
CD15	-115.96 $\pm$ 0.12	-115.21 $\pm$ 0.12	-91.32 $\pm$ 0.24	-91.87 $\pm$ 0.21	39003	100
FMPF15	-114.05 $\pm$ 0.05	-114.06 $\pm$ 0.05	-90.72 $\pm$ 0.18	-90.93 $\pm$ 0.20	26059	25
PMPF15	-114.02 $\pm$ 0.11	-114.03 $\pm$ 0.09	-89.25 $\pm$ 0.17	-89.85 $\pm$ 0.19	26272	25
FPMPF15	**-112.58 $\pm$ 0.03**	**-112.60 $\pm$ 0.02**	**-83.27 $\pm$ 0.15**	**-83.84 $\pm$ 0.13**	26900	25
CD25	-114.50 $\pm$ 0.10	-114.51 $\pm$ 0.10	-91.36 $\pm$ 0.26	-91.04 $\pm$ 0.25	55688	100
FMPF25	-113.07 $\pm$ 0.06	-113.07 $\pm$ 0.07	-90.43 $\pm$ 0.28	-90.63 $\pm$ 0.27	40047	25
PMPF25	-113.70 $\pm$ 0.04	-113.69 $\pm$ 0.04	-89.21 $\pm$ 0.14	-89.79 $\pm$ 0.13	52638	25
FPMPF25	**-112.38 $\pm$ 0.02**	**-112.42 $\pm$ 0.02**	**-83.25 $\pm$ 0.27**	**-83.81 $\pm$ 0.28**	53379	25

both forms of MPF, especially with FMPF. We noted that FMPF and PMPF always have smaller variance compared to CD-k. This implies that FMPF and PMPF are less sensitive to random weight initialization. Figure 4.2 shows initial data and generated samples after running 100 Gibbs steps for each RBM. PMPF clearly produces samples that look more like digits.

4.3.3 Caltech 101 Silhouettes - estimating log likelihood

Finally, we evaluated the same set of RBMs on the Caltech-101 Silhouettes dataset. Compared to MNIST, this dataset contains much more diverse structures with richer correlation among the pixels. It has 10 times more categories, contains less training data per category, and each object covers more of the image. For these reasons, we use 500 hidden units per RBM. The estimated average log-likelihood of train and test data is presented in Table 4.3.

The results for Caltech 101 Silhouettes are consistent with MNIST. In every case, we observed a larger margin between PMPF and CD-k when the number of sampling steps was smaller. In addition, the single bit flip technique was not particularly successful, especially as the number of latent

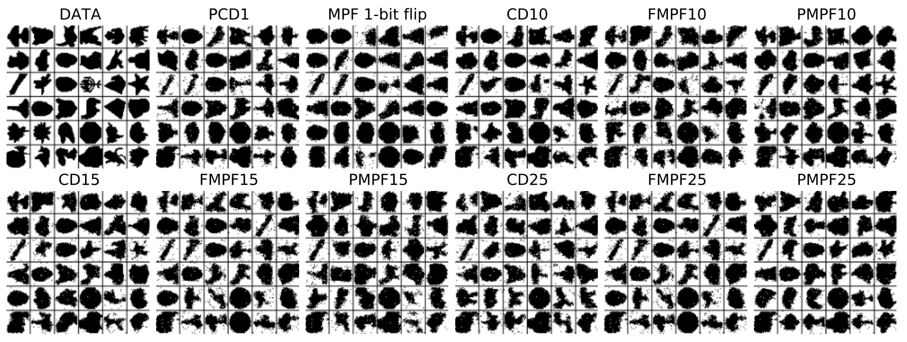

Figure 4.3. Random samples generated by RBMs with different training procedures.

variables grew. We speculate that the reason for this might have to do with the slow rate of convergence for the dynamic system. Moreover, PMPF works better than FMPF for similar reasons. By having persistent samples as the learning progresses, the dynamics always begin closer to equilibrium, and hence converge more quickly. Figure 4.3 shows initial data and generated samples after running 100 Gibbs steps for each RBM on Caltech28 dataset.

4.4 Discussion

MPF is an unsupervised learning algorithm that can be employed off-the-shelf to any energy-based model. It has a number of favorable properties but has not seen application proportional to its potential. In this chapter, we first expounded on MPF and its connections to CD-k training, which allowed us to gain a better understanding and perspective to CD-k. We proved a general form for the transition matrix such that the equilibrium distribution converges to that of an RBM. This may lead to future extensions of MPF based on the choice of $o(\cdot)$ in Equation 4.5.

One of the merits of MPF is that the choice of designing a dynamic system by defining a connectivity function is left open as long as it satisfies the fixed point equation. We thoroughly explored three different connectivity structures, noting that connectivity can be designed inductively or by sampling. Finally, we showed empirically that MPF, and in particular, PMPF, outperforms CD-k

Table 4.3. Experimental results on Caltech-101 Silhouettes using 11 RBMs with 500 hidden units each. The average estimated training and test log-probabilities over 10 repeated runs with random parameter initializations are reported. Likelihood estimates are made with CSL (Bengio et al., 2013b) and AIS (Salakhutdinov and Murray, 2008).

	CSL		AIS			
Method	Avg. log Test	Avg. log Train	Avg. log Test	Avg. log Train	Time (sec)	Batchsize
CD1	-251.30 ± 1.80	-252.04 ± 1.56	-141.87 ± 8.80	-142.88 ± 8.85	300	100
PCD1	**-199.89 ± 1.53**	**-199.95 ± 1.31**	**-124.56 ± 0.24**	**-116.56 ± 2.40**	784	100
MPF-1flip	-281.55 ± 1.68	-283.03 ± 0.60	-164.96 ± 0.23	-170.92 ± 0.20	505	100
CD10	-207.77 ± 0.92	-207.16 ± 1.18	-128.17 ± 0.20	-120.65 ± 0.19	4223	100
FMPF10	-211.30 ± 0.84	-211.39 ± 0.90	-135.59 ± 0.16	-135.57 ± 0.18	2698	20
PMPF10	-203.13 ± 0.12	-203.14 ± 0.10	-128.85 ± 0.15	-123.06 ± 0.15	11973	20
FPMPF10	**-204.15 ± 0.37**	**-203.92 ± 0.31**	**-123.35 ± 0.16**	**-108.81 ± 0.15**	7610	20
CD15	-205.12 ± 0.87	-204.87 ± 1.13	-125.08 ± 0.24	-117.09 ± 0.21	6611	100
FMPF15	-210.66 ± 0.24	-210.19 ± 0.30	-130.28 ± 0.14	-128.57 ± 0.15	3297	20
PMPF15	-201.47 ± 0.13	-201.67 ± 0.10	-127.09 ± 0.10	-121 ± 0.12	18170	20
FPMPF15	**-201.50 ± 0.13**	**-201.70 ± 0.10**	**-122.33 ± 0.13**	**-107.88 ± 0.14**	9603	20
CD25	-201.56 ± 0.11	-201.50 ± 0.13	-124.80 ± 0.20	-117.51 ± 0.23	13745	100
FMPF25	-206.93 ± 0.13	-206.86 ± 0.11	-129.96 ± 0.07	-127.15 ± 0.07	10542	10
FPMPF25	**-199.69 ± 0.11**	**-199.53 ± 0.14**	**-122.75 ± 0.13**	**-108.32 ± 0.12**	18550	10
PMPF25	-199.53 ± 0.11	-199.51 ± 0.12	-127.81 ± 020	-122.23 ± 0.17	23998	10

for training generative models. Until now, RBM training was dominated by methods based on CD-k; however, our results indicate that MPF is a practical and effective alternative.

5 Summary

Throughout the thesis, we reviewed some fundamental machine learning tools such as the families of RBMs and auto-encoders, and used them as the basis to build better insights for unsupervised learning mechanisms.

The theme of chapter 3 project was to advance the theory of gated auto-encoders by revealing the model's capability to model the data distributions. This was done by scoring GAEs according to an energy function. From this perspective, we demonstrated the equivalency of the GAE energy to the free energy of a FCRBM with Gaussian visible units, Bernoulli hidden units, and sigmoid hidden activations. In the same manner, we also showed that the covariance auto-encoder can be formulated in a way such that its energy function is the same as the free energy of a covariance RBM, and this naturally led to a connection between the mean-covariance auto-encoder and mean-covariance RBM. For practical purposes, we also demonstrated an example of how this score can be used to solve multi-label classification.

In chapter 4, we explored the learning algorithm which allows us to approximate or capture the data distributions. More precisely, we investigated a specific learning algorithm for energy-based models, namely the minimum probability flow learning. We also derived a general form for the transition matrix such that the equilibrium distribution converges to that of an RBM, and we explored different types of dynamics for the MPF learning algorithm. One of the merits of the MPF is that the choice of designing a dynamical system by defining a connectivity function is left open as long as it satisfies the fixed point equation.

While conducting these two research projects, the common approach we took was to analyze the models and learning algorithms through the lens of dynamical systems. For example, the potential energy of gated auto-encoders were discovered by defining the vector field in the input space and integrating over the trajectory that is created by the gated auto-encoders. Moreover, contrastive divergence learning algorithm, which used to be seen as a heuristic function, was re-interpreted based on the gradient of minimum probability flow learning expression. This thesis emphasized the dynamical systems approach to understand and develop machine learning algorithms. For future contribution, we can try to analyze the learning dynamics of different models. For example, families of restricted Boltzmann machines and auto-encoders share the same energy function; though, their learning procedure is very different from one and the other. Hence, throughly analzying the learning dynamics of two models can be extended from this thesis.

References

Y. Bengio, P. Lamblin, D. Popovici, and H. Larochelle. Greedy layer-wise training of deep networks. In *Advances in Neural Information Processing Systems*, 2007.

Y. Bengio, A. Courville, and P. Vicent. Representation learning: A review and new perspectives. *IEEE Transactions on Pattern Analysis & Machine Intelligence*, 35(8):1798–1828, 2013a.

Y. Bengio, L. Yao, and K. Cho. Bounding the test log-likelihood of generative models. In *Proceedings of the International Conference of Learning Representations (ICLR)*, 2013b.

Y. Bengio, E. Laufer, A. Guillance, and J. Yosinski. Deep generative stochastic networks trainable by backprop. In *Proceedings of the International Conference of Machine Learning (ICML)*, 2014.

J. Besag. Statistical analysis of non-lattice data. *The Statistician*, 24:179–195, 1975.

M. R. Boutell, J. Luob, X. Shen, and C. M. Brown. Learning multi-label scene classification. *Pattern Recognition*, 37:1757–1771, 2004.

K. Cho, A. Ilin, and T. Raiko. Improved learning of gaussian-bernoulli restricted boltzmann machines. In *ICANN*, pages 10–17, 2011.

B. C. Csáji. Approximation with artificial neural networks. Technical report, Faculty of Sciences; EÃűtvÃűs LorÃąnd University, Hungary, 2001.

G. Dahl, M. Ranzato, A.-r. Mohamed, and G. E. Hinton. Phone recognition with the mean-covariance restricted boltzmann machine. In *Nueral Information Processing Systsems (NIPS)*, 2010.

A. Droniou and O. Sigaud. Gated autoencoders with tied input weights. In *ICML*, 2013.

A. Elisseeff and J. Weston. A kernel method for multi-labelled classification. In *NIPS*, 2002.

A. Guillaume and Y. Bengio. What regularized auto-encoders learn from the data generating distribution. In *International Conference on Learning Representations*, 2013.

G. Hinton. To recognize shapes, first learn to generate images. *Computational Neuroscience: Theoretical Insights into Brain Function. Elsevie*, pages 535–547, 2007.

G. Hinton, S. Osindero, and Y.-W. Teh. A fast learning algorithm for deep belief nets. neural computation. *Neural Computation*, 18:1527–1554, 2006.

G. E. Hinton. How neural networks learn from experience. In *Scientific American*, 1992.

G. E. Hinton. Training products of experts by minimizing contrastive divergence. *Neural Computation*, 14:1771–1880, 2002.

A. Hyvärinen. Estimation of non-normalized statistical models by score matching. *Journal of Machine Learning Research*, 6:695–709, 2005.

H. Kamyshanska and R. Memisevic. On autoencoder scoring. In *ICML*, pages 720–728, 2013.

K. Konda and R. Memisevic. Unsupervised learning of depth and motion. In *arXiv:1312.3429v2*, 2013.

A. Krizhevsky. Learning multiple layers of features from tiny images. Technical report, Department of Computer Science, University of Toronto, 2009.

A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *NIPS*, 2012.

H. Larochelle, D. Erhan, A. Courville, J. Bergstra, and Y. Bengio. An empirical evaluation of deep architectures on problems with many factors of variation. In *ICML*, 2007.

Q. Le, M. Ranzato, R. Monga, M. Devin, K. Chen, G. S. Corrado, J. Dean, and A. Y. Ng. Building high-level features using large scale unsupervised learning. *http://arxiv.org/pdf/1112.6209v5.pdf*, 2011.

Y. Li, D. Tarlow, and R. Zemel. Exploring compositional high order pattern potentials for structured output learning. In *CVPR*, 2013.

D. J. C. MacKay. Failures of the one-step learning algorithm, 2001. URL `http://www.inference.phy.cam.ac.uk/mackay/abstracts/gbm.html`. Unpublished Technical Report.

M. I. Mandel and D. P. W. Ellis. A web-based game for collecting music metadata. *Journal New of Music Research*, 37:151–165, 2008.

M. I. Mandel, D. Eck, and Y. Bengio. Learning tags that vary within a song. In *ISMIR*, 2010.

B. M. Marlin and N. d. Freitas. Asymptotic efficiency of deterministic estimators for discrete energy-based models: Ratio matching and pseudolikelihood. In *Proceedings of the Uncertainty in Artificial Intelligence (UAI)*, 2011.

R. Memisevic. Gradient-based learning of higher-order image features. In *International Conference on Computer Vision*, 2011.

R. Memisevic and G. E. Hinton. Learning to represent spatial transformations with factored higher-order boltzmann machines. *Neural Computation*, 22(6): 1473–1492, 2010.

V. Mnih and G. Hinton. Learning to detect roads in high-resolution aerial images. In *Proceedings of the 11th European Conference on Computer Vision (ECCV)*, September 2010.

V. Mnih, H. Larochelle, and G. E. Hinton. Conditional restricted boltzmann machines for structured output prediction. In *UAI*, 2011.

A.-r. Mohamed and G. Hinton. Phone recognition using restricted boltzmann machines. In *International Conference on Acoustics, Speech and Signal Processing*, 2010.

M. Ranzato and G. E. Hinton. Modeling pixel means and covariances using factorized third-order boltzmann machines. In *Computer Vision and Pattern Recognition Conference (CVPR)*, 2010.

S. Rifai, P. Vincent, X. Muller, X. Glorot, and Y. Bengio. Contractive auto-encoders:explicit invariance during feature extraction. In *International Conference on Machine Learning*, 2011.

D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning internal representations by error propagation. *In Rumelhart, D. E. and McClelland, J. L., editors, Parallel Distributed Processing: Explorations in the Microstructure of Cognition. Volume 1: Foundations, MIT Press*, 1:318–362, 1986.

R. Salakhutdinov and G. Hinton. Deep boltzmann machines. In *International Conference on Artificial Intelligence and Statistics*, 2009.

R. Salakhutdinov and I. Murray. On the quantitative analysis of deep belief networks. In *Proceedings of the International Conference of Machine Learning (ICML)*, 2008.

P. Smolensky. Information processing in dynamical systems: Foundations of harmony theory. In *Parallel Distributed Processing: Volume 1: Foundations*, pages 194–281. MIT Press, 1986.

J. Sohl-Dickstein. Persistent minimum probability flow. Technical report, Redwood Centre for Theoretical Neuroscience, 2011.

J. Sohl-Dickstein, P. Battaglino, and M. R. DeWeese. Minimum probability flow learning. In *Proceedings of the International Conference of Machine Learning (ICML)*, 2011.

M. Spitzer. Brain research and learning over the life cycle. In *In Schooling for tomorrow:Personalising education*, pages 47–62. 2006.

J. Susskind, R. Memisevic, G. Hinton, and M. Pollefeys. Modeling the joint density of two images under a variety of transformations. In *Computer Vision and Pattern Recognition*, 2011.

I. Sutskever and T. Tieleman. On the convergence properties of contrastive divergence. In *Proceedings of the AI & Statistics (AI STAT)*, 2009.

K. Swersky, M. Ranzato, D. Buchman, N. D. Freitas, and B. M. Marlin. On autoencoders and score matching for energy based models. In *International Conference on Machine Learning*, pages 1201–1208, 2011.

G. W. Taylor and G. Hinton. Factored conditional restricted boltzmann machines for modeling motion style. In *International Conference on Machine Learning*, 2009.

T. Tieleman and G. E. Hinton. Using fast weights to improve persistent contrastive divergence. In *Proceedings of the International Conference of Machine Learning (ICML)*, 2009.

L. Tierney. Markov chains for exploring posterior distributions. *Annals of Statistics*, 22:1701–1762, 1994.

P. Vincent. A connection between score matching and denoising auto-encoders. *Neural Computation*, 23(7):1661–1674, 2010.

P. Vincent, H. Larochelle, Y. Bengio, and P.-A. Manzagol. Extracting and composing robust features with denoising autoencoders. In *International Conference on Machine Learning*, 2008.

N. Wang, J. Melchior, and L. Wiskott. Gaussian-binary restricted boltzmann machines on modeling natural image statistics. Technical report, Institut fur Neuroinformatik Ruhr-Universitat Bochum, Bochum, 44780, Germany, 2014.

A Background & Related Work

A.1 Auto-encoders

The objective function of denoising auto-encoder is expressed as

$$L_{DAE} = \mathbb{E}[\|r(\mathbf{x}) - \mathbf{x}\|^2] + \sigma^2 \mathbb{E}[\|\frac{\partial r(\mathbf{z})}{\partial \mathbf{x}}\|^2] + o(\sigma^2) \tag{A.1}$$

as $\sigma \to 0$ where $\mathbf{z} = \mathbf{x} + \boldsymbol{\epsilon}$ ais the corrupted input and $\boldsymbol{\epsilon} \sim \mathcal{N}(0, \sigma^2)$ is the noise that is independently drawn form the Gaussian distribution with variance of σ^2.

Minimizing the mean square error of a denoising auto-encoder is

$$\begin{aligned} L_{DAE} &= \mathbb{E}[\|r(\mathbf{z}) - \mathbf{x}\|^2] \\ &= \mathbb{E}[\|r(\mathbf{x} + \boldsymbol{\epsilon}) - \mathbf{x}\|^2]. \end{aligned}$$

We can Taylor expend the funtion $r(\mathbf{z})$ around $\mathbf{x}$, then we can express the objective function as

$$\begin{aligned} L_{DAE} &= \mathbb{E}[\|r(\mathbf{x}) + \boldsymbol{\epsilon}\frac{\partial r(\mathbf{x})}{\partial \mathbf{x}} + o(\sigma^2) - \mathbf{x}\|^2] \\ &= \mathbb{E}[\|r(\mathbf{x}) - \mathbf{x} + \boldsymbol{\epsilon}\frac{\partial r(\mathbf{x})}{\partial \mathbf{x}} + o(\sigma^2)\|^2] \\ &= \mathbb{E}[\|r(\mathbf{x}) - \mathbf{x}\|^2 + 2(r(\mathbf{x}) - \mathbf{x})\boldsymbol{\epsilon}\frac{\partial r(\mathbf{x})}{\partial \mathbf{x}} + \|\boldsymbol{\epsilon}\frac{\partial r(\mathbf{x})}{\partial \mathbf{x}}\|^2] + o(\sigma^2) \\ &= \mathbb{E}[\|r(\mathbf{x}) - \mathbf{x}\|^2] + 2\mathbb{E}[(r(\mathbf{x}) - \mathbf{x})\boldsymbol{\epsilon}\frac{\partial r(\mathbf{x})}{\partial \mathbf{x}}] + \mathbb{E}[\boldsymbol{\epsilon}^T\frac{\partial r(\mathbf{x})}{\partial \mathbf{x}}^T\frac{\partial r(\mathbf{x})}{\partial \mathbf{x}}\boldsymbol{\epsilon}] + o(\sigma^2) \\ &= \mathbb{E}[\|r(\mathbf{x}) - \mathbf{x}\|^2] + 2\mathbb{E}[\boldsymbol{\epsilon}]\mathbb{E}[(r(\mathbf{x}) - \mathbf{x})\frac{\partial r(\mathbf{x})}{\partial \mathbf{x}}] + Tr\mathbb{E}[\boldsymbol{\epsilon}^T\boldsymbol{\epsilon}]\mathbb{E}[\|\frac{\partial r(\mathbf{x})}{\partial \mathbf{x}}\|^2] + o(\sigma^2) \end{aligned}$$

Since $\mathbb{E}[\boldsymbol{\epsilon}] = 0$ and $\mathbf{E}[\boldsymbol{\epsilon}^t \boldsymbol{\epsilon}] = \sigma^2$, we dervie

$$L_{DAE} = \mathbb{E}[\|r(\mathbf{x}) - \mathbf{x}\|^2] + \sigma^2 \mathbb{E}[\|\frac{\partial r(\mathbf{x})}{\partial \mathbf{x}}\|^2] + o(\sigma^2).$$

A.2 Restricted Boltzmann Machines

$$F(\mathbf{v};\theta) = -\log \sum_{\mathbf{h}} \exp\left(\frac{-1}{\tau} E(\mathbf{v},\mathbf{h})\right) = \frac{1}{\tau}\mathbf{v}^T\mathbf{b} - \frac{1}{\tau}\sum_{j=1}^{H} \log\left(1 + \exp\left(\mathbf{v}^T W + \mathbf{b}\right)\right)$$

This form of the energy is better known as a free energy, as it expresses the difference between the average energy and the entropy of a distribution, in this case, that of $p(\mathbf{h}|\mathbf{v})$.

$$\begin{aligned}
F(\mathbf{v};\theta) &= -\log \sum_{\mathbf{h}} \exp\left(\frac{-1}{\tau} E(\mathbf{v},\mathbf{h})\right) \\
&= -\log\Big(p(\mathbf{v})Z\Big) \\
&= -\log p(\mathbf{v}) - \log Z \\
&= -\Big(\log p(\mathbf{v}) - \log Z\Big) \sum_{\mathbf{h}} p(\mathbf{h}|\mathbf{v}) \\
&= -\sum_{\mathbf{h}} \Big(p(\mathbf{h}|\mathbf{v})\Big(\log p(\mathbf{v}) - \log Z\Big)\Big) \\
&= -\sum_{\mathbf{h}} \Big(p(\mathbf{h}|\mathbf{v})\Big(\log p(\mathbf{v},\mathbf{h}) - \log p(\mathbf{h}|\mathbf{v}) - \log Z\Big)\Big) \\
&= -\sum_{\mathbf{h}} \Big(p(\mathbf{h}|\mathbf{v})\Big(\log\Big(p(\mathbf{v},\mathbf{h})Z\Big) - \log p(\mathbf{h}|\mathbf{v})\Big)\Big) \\
&= -\sum_{\mathbf{h}} \Big(p(\mathbf{h}|\mathbf{v})\Big(E(\mathbf{v},\mathbf{h}) - \log p(\mathbf{h}|\mathbf{v})\Big)\Big) \\
&= -\sum_{\mathbf{h}} p(\mathbf{h}|\mathbf{v}) E(\mathbf{v},\mathbf{h}) - \mathbb{H}\big(p(\mathbf{h}|\mathbf{v})\big)
\end{aligned}$$

where $\mathbb{H}\big(p(\mathbf{h}|\mathbf{v})\big)$ is the entropy of $p(\mathbf{h}|\mathbf{v})$.

B Analyzing Dynamics on Families of Auto-encoders

B.1 Gated Auto-Encoder Scoring

B.1.1 Vector field representation

.

To check that the vector field can be written as the derivative of a scalar field, we can submit to Poincaré's integrability criterion: For some open, simple connected set $\mathcal{U}$, a continuously differentiable function $F : \mathcal{U} \rightarrow \Re^m$ defines a gradient field if and only if

$$\frac{\partial F_i(\mathbf{y})}{\partial y_j} = \frac{\partial F_j(\mathbf{y})}{\partial y_i}, \ \forall i, j = 1 \cdots n.$$

Considering the GAE, note that i^{th} component of the decoder $r_i(\mathbf{y}|\mathbf{x})$ can be rewritten as

$$r_i(\mathbf{y}|\mathbf{x}) = (W^Y_{.i})^T (W^X \mathbf{x} \odot (W^H)^T h(\mathbf{y}, \mathbf{x})) = (W^Y_{.i} \odot W^X \mathbf{x})^T (W^H)^T h(\mathbf{y}, \mathbf{x}).$$

The derivatives of $r_i(\mathbf{y}|\mathbf{x}) - y_i$ with respect to y_j are

$$\begin{aligned}
\frac{\partial r_i(\mathbf{y}|\mathbf{x})}{\partial y_j} &= (W^Y_{.i} \odot W^X \mathbf{x})^T (W^H)^T \frac{\partial h(\mathbf{x}, \mathbf{y})}{\partial y_j} = \frac{\partial r_j(\mathbf{y}|\mathbf{x})}{\partial y_i} \\
\frac{\partial h(\mathbf{y}, \mathbf{x})}{\partial y_j} &= \frac{\partial h(\mathbf{u})}{\partial \mathbf{u}} W^H (W^Y_{.j} \odot W^X \mathbf{x})
\end{aligned} \tag{B.1}$$

where $\mathbf{u} = W^H((W^Y \mathbf{y}) \odot (W^X \mathbf{x}))$. By substituting Equation B.1 into $\frac{\partial F_i}{\partial y_j}, \frac{\partial F_j}{\partial y_i}$, we have

$$\frac{\partial F_i}{\partial y_j} = \frac{\partial r_i(\mathbf{y}|\mathbf{x})}{\partial y_j} - \delta_{ij} = \frac{\partial r_j(\mathbf{y}|\mathbf{x})}{\partial y_i} - \delta_{ij} = \frac{\partial F_j}{\partial y_i}$$

where $\delta_{ij} = 1$ for $i = j$ and 0 for $i \neq j$. Similarly, the derivatives of $r_i(\mathbf{y}|\mathbf{x}) - y_i$ with respect to x_j are

$$\begin{aligned}\frac{\partial r_i(\mathbf{y}|\mathbf{x})}{\partial x_j} =& (W^Y_{\cdot i} \odot W^X_{\cdot j})^T (W^H)^T h(\mathbf{x}, \mathbf{y}) + (W^Y_{\cdot i} \odot W^X \mathbf{x})(W^H)^T \frac{\partial h}{\partial x_j} = \frac{\partial r_j(\mathbf{y}|\mathbf{x})}{\partial x_i},\\ \frac{\partial h(\mathbf{y}, \mathbf{x})}{\partial x_j} =& \frac{\partial h(\mathbf{u})}{\partial \mathbf{u}} W^H (W^Y_{\cdot j} \odot W^X \mathbf{x}).\end{aligned} \tag{B.2}$$

By substituting Equation B.2 into $\frac{\partial F_i}{\partial x_j}, \frac{\partial F_j}{\partial x_i}$, this yields

$$\frac{\partial F_i}{\partial x_j} = \frac{\partial r_i(\mathbf{x}|\mathbf{y})}{\partial x_j} = \frac{\partial r_j(\mathbf{x}|\mathbf{y})}{\partial x_i} = \frac{\partial F_j}{\partial x_i}.$$

B.1.2 Deriving an Energy Function

Integrating out the GAE's trajectory, we have

$$\begin{aligned}E(\mathbf{y}|\mathbf{x}) =& \int_C (r(\mathbf{y}|\mathbf{x}) - \mathbf{y}) d\mathbf{y}\\ =& \int W^Y \left(\left(W^X \mathbf{x} \right) \odot W^H h(\mathbf{u}) \right) d\mathbf{y} - \int \mathbf{y} d\mathbf{y}\\ =& W^Y \left(\left(W^X \mathbf{x} \right) \odot W^H \int h\left(\mathbf{u}\right) d\mathbf{u} \right) - \int \mathbf{y} d\mathbf{y},\end{aligned} \tag{B.3}$$

where $\mathbf{u}$ is an auxiliary variable such that $\mathbf{u} = W^H((W^Y \mathbf{y}) \odot (W^X \mathbf{x}))$ and $\frac{d\mathbf{u}}{d\mathbf{y}} = W^H(W^Y \odot (W^X \mathbf{x} \otimes \mathbf{1}_D))$, where $\otimes$ is the Kronecker product. Consider the symmetric objective function, which is defined in Equation 2.30. Then we have to also consider the vector field system where both symmetric cases $\mathbf{x}|\mathbf{y}$ and $\mathbf{y}|\mathbf{x}$ are valid. As mentioned in Section 3.1.1, let $\xi = [\mathbf{x}; \mathbf{y}]$ and $\gamma = [\mathbf{y}; \mathbf{x}]$. As well, let $W^\xi = \text{diag}(W^X, W^Y)$ and $W^\gamma = \text{diag}(W^Y, W^X)$ where they are block diagonal matrices. Consequently, the vector field becomes

$$F(\boldsymbol{\xi}|\gamma) = r(\boldsymbol{\xi}|\gamma) - \boldsymbol{\xi}, \tag{B.4}$$

and the energy function becomes

$$\begin{aligned}E(\boldsymbol{\xi}|\gamma) =& \int (r(\boldsymbol{\xi}|\gamma) - \boldsymbol{\xi}) d\boldsymbol{\xi}\\ =& \int (W^\xi)^T ((W^\gamma \boldsymbol{\gamma}) \odot (W^H)^T h(\boldsymbol{u})) d\boldsymbol{\xi} - \int \boldsymbol{\xi} d\boldsymbol{\xi}\\ =& (W^\xi)^T ((W^\gamma \boldsymbol{\gamma}) \odot (W^H)^T \int h(\boldsymbol{u}) d\boldsymbol{u}) - \int \boldsymbol{\xi} d\boldsymbol{\xi}\end{aligned}$$

where $\boldsymbol{u}$ is an auxiliary variable such that $\boldsymbol{u} = W^H\left((W^{\xi}\boldsymbol{\xi}) \odot (W^{\gamma}\boldsymbol{\gamma})\right)$. Then

$$\frac{d\boldsymbol{u}}{d\boldsymbol{\xi}} = W^H\left(W^{\xi} \odot (W^{\gamma}\boldsymbol{\gamma} \otimes \mathbf{1}_D)\right).$$

Moreover, note that the decoder can be re-formulated as

$$\begin{aligned} r(\boldsymbol{\xi}|\gamma) &= (W^{\xi})^T(W^{\gamma}\boldsymbol{\gamma} \odot (W^H)^T h(\boldsymbol{\xi},\gamma)) \\ &= \left((W^{\xi})^T \odot (W^{\gamma}\boldsymbol{\gamma} \otimes \mathbf{1}_D)\right)(W^H)^T h(\boldsymbol{\xi},\gamma). \end{aligned}$$

Re-writing the first term of Equation B.3 in terms of the auxiliary variable $\boldsymbol{u}$, the energy reduces to

$$\begin{aligned} E(\boldsymbol{\xi}|\gamma) &= \left((W^{\xi})^T \odot (W^{\gamma}\boldsymbol{\gamma} \otimes \mathbf{1}_D)\right)(W^H)^T \int h(\boldsymbol{u})\left(W^H(W^{\xi} \odot (W^{\gamma}\boldsymbol{\gamma} \otimes \mathbf{1}_D))\right)^{-1} d\boldsymbol{u} - \int \boldsymbol{\xi} d\boldsymbol{\xi} \\ &= \left((W^{\xi})^T \odot (W^{\gamma}\boldsymbol{\gamma} \otimes \mathbf{1}_D)\right)(W^H)^T \left((W^{\xi} \odot (W^{\gamma}\boldsymbol{\gamma} \otimes \mathbf{1}_D))W^H\right)^{-T} \int h(\boldsymbol{u}) d\boldsymbol{u} - \int \boldsymbol{\xi} d\boldsymbol{\xi} \\ &= \int h(\boldsymbol{u}) d\boldsymbol{u} - \int \boldsymbol{\xi} d\boldsymbol{\xi} \\ &= \int h(\boldsymbol{u}) d\boldsymbol{u} - \frac{1}{2}\boldsymbol{\xi}^2 + \text{const.} \end{aligned}$$

B.2 Relation to other types of Restricted Boltzmann Machines

B.2.1 Gated Auto-encoder and Factored Gated Conditional Restricted Boltzmann Machines

Suppose that the hidden activation function is a sigmoid. Moreover, we define our Gated Auto-encoder to consist of an encoder $h(\cdot)$ and decoder $r(\cdot)$ such that

$$\begin{aligned} h(\mathbf{x},\mathbf{y}) &= h(W^H((W^X\mathbf{x}) \odot (W^Y\mathbf{y})) + \mathbf{b}) \\ r(\mathbf{x}|\mathbf{y},h) &= (W^X)^T((W^Y\mathbf{y}) \odot (W^H)^T h(\mathbf{x},\mathbf{y})) + \mathbf{a}, \end{aligned}$$

where $\theta = \{W^H, W^X, W^Y, \mathbf{b}\}$ is the parameters of the model. Note that the weights are not tied in this case. The energy function for the Gated Auto-encoder will be:

$$\begin{aligned} E_{\sigma}(\mathbf{x}|\mathbf{y}) &= \int (1 + \exp(-W^H(W^X\mathbf{x}) \odot (W^Y\mathbf{y}) - \mathbf{b}))^{-1} d\mathbf{u} - \frac{\mathbf{x}^2}{2} + \mathbf{a}\mathbf{x} + \text{const} \\ &= \sum_k \log(1 + \exp(-W^H_{k\cdot}(W^X\mathbf{x}) \odot (W^Y\mathbf{y}) - b_k)) - \frac{\mathbf{x}^2}{2} + \mathbf{a}\mathbf{x} + \text{const.} \end{aligned}$$

Now consider the free energy of a Factored Gated Conditional Restricted Boltzmann Machine (FCRBM).

The energy function of a FCRBM with Gaussian visible units and Bernoulli hidden units is defined by

$$E(\mathbf{x},\mathbf{h}|\mathbf{y}) = \frac{(\mathbf{a}-\mathbf{x})^2}{2\sigma^2} - \mathbf{bh} - \sum_f W^X_{f\cdot}\mathbf{x} \odot W^Y_{f\cdot}\mathbf{y} \odot W^H_{f\cdot}\mathbf{h}.$$

Given $\mathbf{y}$, the conditional probability density assigned by the FCRBM to data point $\mathbf{x}$ is

$$p(\mathbf{x}|\mathbf{y}) = \frac{\sum_{\mathbf{h}} \exp -(E(\mathbf{x},\mathbf{h}|\mathbf{y}))}{Z(\mathbf{y})} = \frac{\exp(-F(\mathbf{x}|\mathbf{y}))}{Z(\mathbf{y})}$$

$$-F(\mathbf{x}|\mathbf{y}) = \log\left(\sum_{\mathbf{h}} \exp(-E(\mathbf{x},\mathbf{h}|\mathbf{y}))\right)$$

where $Z(\mathbf{y}) = \sum_{\mathbf{x},\mathbf{h}} \exp(E(\mathbf{x},\mathbf{h}|\mathbf{y}))$ is the partition function and $F(\mathbf{x}|\mathbf{y})$ is the free energy function. Expanding the negative of the free energy function, which is the potential function, the equation becomes

$$\begin{aligned}
-F(\mathbf{x}|\mathbf{y}) &= \log\sum_{\mathbf{h}} \exp(-E(\mathbf{x},\mathbf{h}|\mathbf{y})) \\
&= \log\sum_{\mathbf{h}} \exp\left(\frac{-(\mathbf{a}-\mathbf{x})^2}{2\sigma^2} + \mathbf{bh} + \sum_f W^X_{f\cdot}\mathbf{x} \odot W^Y_{f\cdot}\mathbf{y} \odot W^H_{f\cdot}\mathbf{h}\right) \\
&= -\frac{(\mathbf{a}-\mathbf{x})^2}{2\sigma^2} + \log\left(\sum_{\mathbf{h}} \exp\left(\mathbf{bh} + \sum_f W^X_{f\cdot}\mathbf{x} \odot W^Y_{f\cdot}\mathbf{y} \odot W^H_{f\cdot}\mathbf{h}\right)\right) \\
&= -\frac{(\mathbf{a}-\mathbf{x})^2}{2\sigma^2} + \log\left(\sum_{\mathbf{h}}\prod_k \exp\left(b_k h_k + \sum_f (W^X_{f\cdot}\mathbf{x} \odot W^Y_{f\cdot}\mathbf{y}) \odot W^H_{fk} h_k\right)\right) \\
&= -\frac{(\mathbf{a}-\mathbf{x})^2}{2\sigma^2} + \sum_k \log\left(1 + \exp\left(b_k + \sum_f \left((W^H_{fk})^T (W^X\mathbf{x} \odot W^Y\mathbf{y})\right)\right)\right).
\end{aligned}$$

Note that we can center the data by subtracting mean of $\mathbf{x}$ and dividing by its standard deviation, and therefore assume that $\sigma^2 = 1$. Substituting, we have

$$\begin{aligned}
-F(\mathbf{x}|\mathbf{y}) &= -\frac{(\mathbf{a}-\mathbf{x})^2}{2} + \sum_k \log\left(1 + \exp\left(-b_k - \sum_f (W^H_{fk})^T (W^X\mathbf{x} \odot W^Y\mathbf{y})\right)\right) \\
&= \sum_k \log\left(1 + \exp\left(b_k + \sum_f (W^H_{fk})^T (W^X\mathbf{x} \odot W^Y\mathbf{y})\right)\right) - \mathbf{a}^2 + \mathbf{ax} - \frac{\mathbf{x}^2}{2} \\
&= \sum_k \log\left(1 + \exp\left(b_k + \sum_f (W^H_{fk})^T (W^X\mathbf{x} \odot W^Y\mathbf{y})\right)\right) + \mathbf{ax} - \frac{\mathbf{x}^2}{2} + \text{const}
\end{aligned}$$

Letting $W^H = (W^H)^T$, formulation becomes

$$= \sum_k \log\left(1 + \exp\left(b_k + \sum_f W^H_{kf}(W^X\mathbf{x} \odot W^Y\mathbf{y})\right)\right) + \mathbf{ax} - \frac{\mathbf{x}^2}{2} + \text{const}$$

Hence, the Conditional Gated auto-encoder and the FCRBM are equal up to a constant.

B.2.2 Mean-covariance Auto-encoder and mean-covariance Restricted Boltzmann Machines

Theorem 2. *Consider a covariance auto-encoder with an encoder and decoder,*

$$\begin{aligned} h(\mathbf{x},\mathbf{x}) &= h(W^H((W^F\mathbf{x})^2) + \mathbf{b}) \\ r(\mathbf{x}|\mathbf{y}=\mathbf{x},h) &= (W^F)^T(W^F\mathbf{y} \odot (W^H)^T h(\mathbf{x},\mathbf{y})) + \mathbf{a}, \end{aligned}$$

where $\theta = \{W^F, W^H, \mathbf{a}, \mathbf{b}\}$ are the parameters of the model. Moreover, consider a covariance Restricted Boltzmann Machine with Gaussian distribution over the visibles and Bernoulli distribution over the hiddens, such that its energy function is defined by

$$E^c(\mathbf{x},\mathbf{h}) = \frac{(\mathbf{a}-\mathbf{x})^2}{\sigma^2} - \sum_f P\mathbf{h}(C\mathbf{x})^2 - \mathbf{bh},$$

where $\theta = \{P, C, \mathbf{a}, \mathbf{b}\}$ are its parameters. Then the energy function for a covariance Auto-encoder with dynamics $r(\mathbf{x}|\mathbf{y}) - \mathbf{x}$ is equivalent to the free energy of a covariance Restricted Boltzmann Machine. The energy function of the covariance Auto-encoder is

$$E(\mathbf{x},\mathbf{x}) = \sum_k \log(1 + \exp(W^H(W^F\mathbf{x})^2 + \mathbf{b})) - \mathbf{x}^2 + \mathit{const} \tag{B.5}$$

Proof. Note that the covariance auto-encoder is the same as a regular Gated Auto-encoder, but setting $\mathbf{y} = \mathbf{x}$ and making the factor loading matrices the same, i.e. $W^F = W^Y = W^X$. Then applying the general energy equation for GAE, Equation 3.3, to the covariance auto-encoder, we get

$$\begin{aligned} E(\mathbf{x},\mathbf{x}) &= \int h(\mathbf{u})d\mathbf{u} - \frac{1}{2}\mathbf{x}^2 + \text{const} \\ &= \sum_k \log(1 + \exp(W^H(W^F\mathbf{x})^2 + \mathbf{b})) - \mathbf{x}^2 + \mathbf{ax} + \text{const}, \end{aligned} \tag{B.6}$$

where $\mathbf{u} = W^H(W^F\mathbf{x})^2 + \mathbf{b}$.

Now consider the free energy of the mean-covariance Restricted Boltzmann Machine (mcRBM) with Gaussian distribution over the visible units and Bernoulli distribution over the hidden units:

$$
\begin{aligned}
-F(\mathbf{x}|\mathbf{y}) &= \log \sum_{\mathbf{h}} \exp\left(-E(\mathbf{x}, \mathbf{h}|\mathbf{y})\right) \\
&= \log \sum_{h} \exp\left(-\frac{(\mathbf{a}-\mathbf{x})^2}{\sigma^2} + (P\mathbf{h})(C\mathbf{x})^2 + \mathbf{bh}\right) \\
&= \log \sum_{h} \prod_{k} \exp\left(-\frac{(\mathbf{a}-\mathbf{x})^2}{\sigma^2} + \sum_{f} (P_{fk}h_k)(C\mathbf{x})^2 + b_k h_k\right) \\
&= \sum_{k} \log\left(1 + \exp\left(\sum_{f} (P_{fk}h_k)(C\mathbf{x})^2\right)\right) - \frac{(\mathbf{a}-\mathbf{x})^2}{\sigma^2}.
\end{aligned}
$$

As before, we can center the data by subtracting mean of $\mathbf{x}$ and dividing by its standard deviation, and therefore assume that $\sigma^2 = 1$. Substituting, we have

$$
= \sum_{k} \log\left(1 + \exp\left(\sum_{f} (P_{fk}h_k)(C\mathbf{x})^2\right)\right) - (\mathbf{a}-\mathbf{x})^2. \tag{B.7}
$$

Letting $W^H = P^T$ and $W^F = C$, we get

$$
= \sum_{k} \log\left(1 + \exp\left(\sum_{f} (P_{fk}h_k)(C\mathbf{x})^2\right)\right) - \mathbf{x}^2 + \mathbf{ax} + \text{const.} \tag{B.8}
$$

Therefore, the two equations are equivalent. $\square$

C Training Energy-based Models Under the Various Kinds of Dynamics

C.1 Minimum Probability Flow

C.1.1 Dynamics of The Model

Theorem 1. *Suppose $p_j^{(\infty)}$ is the probability of state j and $p_i^{(\infty)}$ is the probability of state i. Let the transition matrix be*

$$\Gamma_{ij} = g_{ij} \exp\left(\frac{o(F_i - F_j) + 1}{2}(F_j - F_i)\right) \tag{C.1}$$

such that $o(\cdot)$ is any odd function, where g_{ij} is the symmetric connectivity between the states i and j. Then this transition matrix satisfies detailed balance in Equation C.2.

Proof. By cancalling out the partition function, the detailed balance Equation C.2 can be formulated to be

$$\Gamma_{ji} \exp(-F_i) = \Gamma_{ij} \exp(-F_j) \tag{C.2}$$

where $F_i = F(\mathbf{v} = i; \theta)$ We substitute transition matrix defined in Equation 4.5, then we get the following after straight forward formula manipulation.

$$\Gamma_{ji} \exp\left(-F_i\right)/\Gamma_{ij} \exp\left(-F_j\right)) = 1$$
$$\exp\left(\frac{o(F_i - F_j) + 1}{2}(F_j - F_i) - F_i\right) / \exp\left(\frac{o(F_j - F_i) + 1}{2}(F_i - F_j) - F_j\right) = 1$$
$$\exp\left(\frac{o(F_i - F_j) + 1}{2}(F_j - F_i) - F_i - \frac{o(F_j - F_i) + 1}{2}(F_i - F_j) + F_j\right) = 1$$
$$\frac{o(F_i - F_j) + 1}{2}(F_j - F_i) - F_i - \frac{o(F_j - F_i) + 1}{2}(F_i - F_j) + F_j = 0$$
$$(F_i - F_j)\left(\frac{o(F_i - F_j) + 1}{2} + \frac{o(F_j - F_i) + 1}{2} - 1\right) = 0$$
$$(F_i - F_j)\left(\frac{o(F_i - F_j)}{2} + \frac{o(F_j - F_i)}{2}\right) = 0$$

Notice that since $o(\cdot)$ is an odd function, this makes the term $\left(\frac{o(F_i - F_j)}{2} + \frac{o(F_j - F_i)}{2}\right) = 0$. Therefore, the detailed balance criterion is satisfied. $\square$